Fast Models

Version 8.1

Reference Manual

Fast Models
Reference Manual

Release Information

Change History

Date	Issue	Confidentiality	Change
February 2008	A	Confidential	New document, based on previous documentation.
June 2008	B	Confidential	Update for System Generator v4.0.
August 2008	C	Confidential	Update for System Generator v4.0 SP1.
December 2008	D	Confidential	Update for Fast Models v4.1.
March 2009	E	Confidential	Update for Fast Models v4.2.
May 2009	F	Confidential	Update for Fast Models v5.0.
September 2009	G	Non Confidential	Update for Fast Models v5.1
February 2010	H	Non Confidential	Update for Fast Models v5.2
October 2010	I	Non-Confidential	Update for Fast Models v6.0.
May 2011	J	Non-Confidential	Update for Fast Models v6.1.
July 2011	K	Confidential	Update for Fast Models v6.2.
November 2011	L	Non-Confidential	Update for Fast Models v7.0.
May 2012	M	Non-Confidential	Update for Fast Models v7.1.
December 2012	N	Non-Confidential	Update for Fast Models v8.0.
May 2013	O	Non-Confidential	Update for Fast Models v8.1.

Product Status

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

Web Address

`http://www.arm.com`

Contents
Fast Models Reference Manual

Preface

This preface introduces the *Fast Models Reference Manual*. It contains the following sections:

- *About this book* on page vii
- *Feedback* on page xi.

About this book

This book provides a reference for signaling, clock, bus, generic peripheral, and processor components included in Fast Models. These provide a *Programmer's View* (PV) of the processor and peripheral components.

Intended audience

This book has been written for experienced hardware and software developers to aid the development of ARM architecture-based products using the components as part of a development environment.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an overview of the Fast Models bus and peripheral.

Chapter 2 *Accuracy and Functionality*

Read this chapter for an insight into the accuracy and functionality of virtual platforms produced with the Fast Models Portfolio.

Chapter 3 *Framework Protocols*

Read this chapter for a description of the Fast Models signaling and associated protocols.

Chapter 4 *Processor Components*

Read this chapter for a description of the Fast Models processor.

Chapter 5 *Peripheral and Interface Components*

Read this chapter for a description of the Fast Models peripherals and interfaces.

Chapter 6 *Versatile Express Model: Platform and Components*

Read this chapter for details on the components that are specific to the *Versatile*™ *Express* (VE) Baseboard FVP.

Chapter 7 *Emulation Baseboard Model: Platform and Components*

Read this chapter for details on the components that are specific to the *Emulation Baseboard* (EB) FVP.

Chapter 8 *Microcontroller Prototyping System: Platform and Components*

Read this chapter for details on the components that are specific to the *Microcontroller Prototyping System*® (MPS) FVP.

Appendix A *AEM ARMv7-A specifics*

Read this appendix for details on the characteristics and capabilities that are specific to the *Architecture Envelope Model* (AEM) ARMv7-A.

Glossary

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

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

Conventions

Conventions that this book can use are described in:

- *Typographical conventions*
- *Signals*.

Typographical conventions

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal 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` (with "mono" underlined)	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Signals

The signal conventions are:

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

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

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

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

This book contains information that is specific to this product. The following publications provide reference information about the ARM architecture:

- *AMBA® Specification* (ARM IHI 0011)

- *ARM Architecture Reference Manuals*, http://infocenter.arm.com/help/topic/com.arm.doc.set.architecture/index.html
- *ARM Generic Interrupt Controller Architecture Specification* (ARM IHI 0048)
- *Cortex-R5 Technical Reference Manual*, http://infocenter.arm.com/help/topic/com.arm.doc.subset.cortexr.r5/index.html.

The following publications provide information about related ARM products and toolkits:

- *RealView® Development Suite Real-Time System Models User Guide* (ARM DUI 0424)
- *RealView Emulation Baseboard User Guide (Lead Free)* (ARM DUI 0411)
- *Fast Models User Guide* (ARM DUI 0370)
- *LISA+ Language for Fast Models Reference Manual* (ARM DUI 0372)
- *Component Architecture Debug Interface v2.0 Developer Guide* (ARM DUI 0444)
- *AMBA-PV Extensions to OSCI TLM 2.0 Developers Guide* (ARM DUI 0455)
- *Real-Time System Model VE Cortex-A15 Cortex-A7 CCI-400 User Guide* (ARM DUI 0585).

The following publications provide information about ARM PrimeCell® and other peripheral or controller devices:

- *PrimeCell UART (PL011) Technical Reference Manual* (ARM DDI 0183)
- *ARM PrimeCell Synchronous Serial Port (PL022) Technical Reference Manual* (ARM DDI 0194)
- *ARM PrimeCell Real Time Clock (PL030) Technical Reference Manual* (ARM DDI 0140)
- *ARM PrimeCell Real Time Clock (PL031) Technical Reference Manual* (ARM DDI 0224)
- *ARM PrimeCell Advanced Audio CODEC Interface (PL041) Technical Reference Manual* (ARM DDI 0173)
- *ARM PrimeCell PS2 Keyboard/Mouse Interface (PL050) Technical Reference Manual* (ARM DDI 0143)
- *ARM PrimeCell General Purpose Input/Output (PL060) Technical Reference Manual* (ARM DDI 0142)
- *PrimeCell DMA Controller (PL080) Technical Reference Manual* (ARM DDI 0196)
- *PrimeCell Color LCD Controller (PL110) Technical Reference Manual* (ARM DDI 0161)
- *PrimeCell Color LCD Controller (PL111) Technical Reference Manual* (ARM DDI 0293)
- *ARM PrimeCell Multimedia Card Interface (PL180) Technical Reference Manual* (ARM DDI 0172)
- *ARM PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual* (ARM DDI 0273)
- *PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual* (ARM DDI 0246)
- *PrimeCell Dynamic Memory Controller (PL340) Technical Reference Manual* (ARM DDI 0331)
- *PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual* (ARM DDI 0380)
- *PrimeCell TrustZone Address Space Controller (PL380) Technical Reference Manual* (ARM DDI 0431)

- *PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual* (ARM DDI 0416A)
- *ARM Dual Timer Module (SP804) Technical Reference Manual* (ARM DDI 0271)
- *ARM Watchdog Module (SP805) Technical Reference Manual* (ARM DDI 0270)
- *PrimeXsys® System Controller (SP810) Technical Reference Manual* (ARM DDI 0254)
- *PrimeCell Infrastructure AMBA 3 AXI TrustZone® Memory Adapter (BP141) Technical Overview* (ARM DTO 0017)
- *AMBA 3 TrustZone Interrupt Controller (SP890) Revision: r0p0 Technical Overview* (ARM DTO 0013)
- *PrimeCell Infrastructure AMBA 3 TrustZone Protection Controller (BP147) Technical Overview* (ARM DTO 0015)
- *AMBA AXI and ACE Protocol Specification* (ARM IHI 0022)
- *Motherboard Express µATX V2M-P1 Technical Reference Manual* (ARM DUI 0447)
- *Cortex-R7 MPCore Technical Reference Manual* (ARM DUI 0458).

Other publications

This section lists relevant documents published by third parties:

- For more information on the Open SystemC Initiative, see OSCI, `http://www.systemc.org`.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

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

- the title
- the number, ARM DUI 0423O
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces Fast Models. It contains the following section:

- *About the components* on page 1-2.

1.1 About the components

The *Programmer's View* (PV) models of processors and devices work at a level where functional behavior is equivalent to what a programmer would see using the respective hardware. Timing accuracy is sacrificed to achieve fast simulation execution speeds. This means that you can use the PV models for confirming software functionality, but you must not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior.

Chapter 2
Accuracy and Functionality

This chapter describes the accuracy and functional scope of virtual platforms using models from Fast Models. It contains the following sections:

2.1 Model Capabilities Overview

Fast Models attempt to accurately model the hardware, but compromises exist between speed of execution, accuracy and other criteria. This means that a processor model might not be expected to match the hardware under certain conditions.

For a detailed description of timing, bus and cache modeling limitations see *Functional caches in Fast Models* on page 2-3 and *How Accurate are Fast Models?* on page 2-5.

2.1.1 What Fast Models can do

Fast Models do:

- Instruction accurate modeling
- Correct execution of architecturally-correct code.

2.1.2 What Fast Models cannot do

Fast Models cannot be used to:

- Validate the hardware
- Model unpredictable behavior
- Model cycle counting
- Model timing sensitive behavior
- Model SMP instruction scheduling
- Model software performance
- Model bus traffic.

2.2 Functional caches in Fast Models

Fast Models implement two different types of cache model:

- Register model.
- Functional model.

A register model provides all the cache control registers so that cache operations succeed, but does not model the state of the cache. All accesses go to memory.

A functional model tracks cache state and its contents at each level of the memory hierarchy. If cache management is not done correctly, incorrect data might be returned, as it would on real hardware.

Fast Models provide:

- register models of caches on all processors that support caches and also the PL310 cache controller
- functional models of L1 caches on Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A15, and Cortex-R4 processors, and the L2 cache on the Cortex-A7, Cortex-A8, and Cortex-A15 processors, and the PL310 cache controller.

2.2.1 Configuring functional caches

Functional caches are, by default, disabled. Configuration parameters are described in Table 2-1.

Table 2-1 Parameters to control functional cache behavior

Component	Parameter	Description	Type	Allowed value	Default value
Cortex-A15[a]	`l1_dcache-state_modelled`	Set whether L1 D-cache has stateful implementation.	Boolean	true/false	false
Cortex-A15[a]	`l1_icache-state_modelled`	Set whether L1 I-cache has stateful implementation.	Boolean	true/false	false
Cortex-A15[a]	`l2_cache-state_modelled`	Enable unified Level 2 cache state model.	Boolean	true/false	false
Cortex-A9	`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	true/false	false
Cortex-A9	`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	true/false	false
Cortex-A8[a]	`l1_dcache-state_modelled`	Set whether L1 D-cache has stateful implementation.	Boolean	true/false	false
Cortex-A8[a]	`l1_icache-state_modelled`	Set whether L1 I-cache has stateful implementation.	Boolean	true/false	false
Cortex-A8[a]	`l2_cache-state_modelled`	Enable unified Level 2 cache state model.	Boolean	true/false	false
Cortex-A7	`l1_dcache-state_modelled`	Set whether L1 D-cache has stateful implementation.	Boolean	true/false	false
Cortex-A7	`l1_icache-state_modelled`	Set whether L1 I-cache has stateful implementation.	Boolean	true/false	false

Table 2-1 Parameters to control functional cache behavior (continued)

Component	Parameter	Description	Type	Allowed value	Default value
Cortex-A7	`l2_cache-state_modelled`	Enable unified Level 2 cache state model.	Boolean	true/false	false
Cortex-A5	`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	true/false	false
Cortex-A5	`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	true/false	false
Cortex-R7	`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	true/false	false
Cortex-R7	`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	true/false	false
Cortex-R4	`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	true/false	false
Cortex-R4	`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	true/false	false
PL310 L2CC	`cache-state_modelled`	Enable unified Level 2 cache state model.	Boolean	true/false	false

a. Correct modeling of hardware-maintained coherency requires that `l2_cache-state_modelled` is enabled if any of the per processor `l1_dcache-state_modelled` parameters are enabled.

2.2.2 Performance effects of enabling functional caches

In general, enabling functional cache modeling is likely to reduce performance.

The L1 and L2 functional caches can be enabled in any combination, but ARM recommends that functional behavior is either disabled completely or enabled for both I and D L1 caches and the L2 cache to give consistent system operation.

Enabling functional caches in PL310 L2CC and not enabling functional L1 caches causes model performance to be significantly reduced.

If platform memory is being modeled outside of the Fast Models environment, for example in a SystemC environment, use of functional cache modeling might significantly improve performance if there is no alternate fast route to memory.

2.3 How Accurate are Fast Models?

Fast Models aim to be accurate to the view of the system programmer. Architecturally-correct software is not able to tell the difference between running on hardware and running on the model.

Software is able to detect differences between hardware and the model, but these differences generally depend on behavior that is not precisely specified. For example, it is possible to detect differences in the exact timings of instructions and bus-transactions, effects of speculative prefetch and cache victim selection. Certain classes of behavior are specified as unpredictable and these cases are detectable by software. A program that relies on such behavior, even unintentionally, is not guaranteed to work reliably on any device, or on a Fast Model. Programs that exploit this behavior might execute differently between the hardware and the model.

Fast Models are only concerned with accuracy from the point of view of the program running on the processors. They do not attempt to accurately model bus transactions. PVBus provides the infrastructure to ensure that the program gets the correct results.

The rest of this section looks at specific areas where the model might differ from hardware.

2.3.1 Timing

The most important thing that Fast Models do not model is accurate instruction timing. The simulation as a whole has a very accurate concept of timing, but the *Code Translation* (CT) processors do not dispatch instructions with any claim to simulating device-like timing. In general, a processor issues a set of instructions (a “quantum”) at the same point in simulation time, and then waits for some amount of time before executing the next quantum. The timing is arranged so that the processor averages one instruction per clock tick.

The consequences of this are:

- The perceived performance of software running on the model differs from real-world software. (In particular, memory accesses and arithmetic operations all take the same amount of time.)
- A program might be able to detect the quantized execution behavior of a processor, for example by polling a high-resolution timer.
- All instructions in a quantum are effectively atomic.

  Note

  This might mask some race-condition bugs in software.

2.3.2 Bus traffic

PVBus can simulate the behavior of individual bus transactions passing through a hierarchy of bus fabric, but it employs several techniques to optimize this process:

1. PVBus generally decodes the path between a bus master and the bus slave the first time a transaction is issued. All subsequent transactions to the same address are automatically sent to the same slave, without passing through the intervening fabric.
2. For accesses to normal memory, the master can cache a pointer to the (host) storage that holds the data contents of the memory. The master can read and write directly to this memory without generating bus-transactions.

3. For instruction-fetch, and for operations such as repeated DMA from framebuffer memory, PVBus provides an optimization called “snooping”, that informs the master if anyone else could have modified the contents of memory. If no changes have occurred the master can avoid the need to re-read memory contents.

If a piece of bus fabric wants to intercept and log all bus transactions, it can defeat these optimizations by claiming to be a slave device. It can then log all transactions and can reissue identical transactions on its own master port. However, doing this slows all bus transactions and significantly impacts simulation performance.

Note

If direct accesses to memory by the CT engine are intercepted by the fabric, the processor is forced to single step. Execution is much slower than normal operation with translated code.

The bus traffic generated by a processor is not representative of real traffic for the following reasons:

Timing Differences

These can be caused by re-ordering and buffering of memory accesses, out-of-order execution, speculative prefetch and drain-buffers. They are not modeled, since they are not visible to the programmer except in situations where a multiprocessor program contains race-conditions that violate serial-consistency expectations.

Bus Contention

Fast Models do not model the time taken for a bus transaction, so they cannot model the effects of multiple transactions contending for bus availability.

Size of Access

Fast Models do not attempt to generate the same types of burst transaction from the processor for accesses to multiple consecutive locations.

Instruction Fetch

The behavior of the instruction prefetch unit of a processor is not modeled to match the hardware implementation. See *Instruction prefetch*.

Behavioral Differences

In some software, the trace of instruction execution is dependent on timing effects. For example, if a loop polls a device waiting for a 10ms time-out, the number of iterations of the polling loop depends on the rate of instruction execution.

2.3.3 Instruction prefetch

The CT engine in the processor models relies on the PVBus optimizations described in *Bus traffic* on page 2-5. It only performs code-translation if it has been able to prefetch and snoop the underlying memory. After this has been done, it has no further need to issue bus transactions until the snoop handling detects an external modification to memory.

If the CT engine cannot get prefetch access to memory, it drops to single-stepping. This single-stepping is very slow (~1000x slower than translated code execution). It also generates spurious bus-transactions for each step.

Real processors attempt to prefetch instructions ahead of execution and predict branch destinations to keep the prefetch queue full. The instruction prefetch behavior of a processor can be observed by a program that writes into its own prefetch queue (without using explicit barriers). The architecture does not define the results.

The CT engine processes code in blocks. The effect is as if the processor filled its prefetch queue with a block of instructions, then executed the block to completion. As a result, this virtual prefetch queue is sometimes larger and sometimes smaller than the corresponding hardware. In the current implementation, the virtual prefetch queue can follow small forward branches.

With an L1 instruction cache turned on, the instruction block size is limited to a single cache-line. The processor ensures that a line is present in the cache at the point where it starts executing instructions from that line.

In real hardware, the effects of the instruction prefetch queue are to cause additional fetch transactions. Some of these are redundant because of incorrect branch prediction. This causes extra cache and bus pressure.

2.3.4 Out-of-order execution and write-buffers

The current CT implementation always executes instructions sequentially in program order. One instruction is completely retired before the next starts to execute. In a real processor, multiple memory accesses can be outstanding at once, and can complete in a different order from their program order. Writes can also be delayed in a write-buffer.

The programmer visible effects of these behaviors is defined in the architecture as the Weakly Ordered memory model, which the programmer must be aware of when writing lock-free multiprocessor code.

Within Fast Models, all memory accesses can be observed to happen in program order, effectively as if all memory is Strongly Ordered.

2.3.5 Caches

The effects of caches are programmer visible because they can cause a single memory location to exist as multiple inconsistent copies. If caches are not correctly maintained, reads can observe stale copies of locations, and flushes/cleans can cause writes to be lost.

There are three ways in which incorrect cache maintenance can be programmer visible:

From the D-side interface of a single processor

The only way of detecting the presence of caches is to create aliases in the memory map, so that the same range of physical addresses can be observed as both cached and non-cached memory.

From the D-side of a single processor to its I-side

Stale instruction data can be fetched when new instructions have been written by the D-side. This can either be because of deliberate self-modifying code, or as a consequence of incorrect OS demand paging.

Between one processor and another device

For example, another processor in a non-coherent MP system, or an external DMA device.

Fast Models with cache-state modeling enabled can replicate all of these failure-cases. However, they do not attempt to reproduce the following effects of caches:

- Changes to timing behavior of a program because of cache hits/misses (because the timing of memory accesses is not modeled).
- Ordering of line-fill and eviction operations.
- Cache usage statistics (because the models do not generate accurate bus traffic).
- Device-accurate allocation of cache victim lines (which is not possible without accurate bus traffic modeling).
- Write-streaming behavior where a cache spots patterns of writes to consecutive addresses and automatically turns off the write-allocation policy.

The Cortex-A9 and Cortex-A5 models do not model device-accurate MESI behavior. The Cortex-A15 and Cortex-A7 models do simulate hardware MOESI state handling, and can handle cache-to-cache snoops. In addition, they model the AMBA-4 ACE cache-coherency protocols over their PVBus ports, so can be connected to a model of an ACE Coherent Interconnect (such as the CCI-400 model) to simulate coherent sharing of cache contents between processors.

It is not currently possible to insert any devices between the processor and its L1 caches. In particular, you can not model L1 traffic, although you can tell the model not to model the state of L1 caches.

Chapter 3
Framework Protocols

This chapter describes the common framework protocols with Fast Models. It contains the following sections:

3.1 About the framework and protocols

This section describes the common protocols and components provided by the Fast Models framework. These are:

- *Signaling protocols*
- *Clocking components and protocols*
- *Debug interface protocols*.

Programmer's View (PV) models of processors and devices work at a level where functional behavior matches what can be observed from software. Accuracy in timing is sacrificed to achieve fast simulation speeds.

3.1.1 Signaling protocols

The signaling protocols are very simple protocols used by many components to indicate changes in state for signals such as interrupt and reset. There are two variants of each signaling protocol:

- one that permits components to signal a state change to other components
- one that permits the other components to passively query the current state of the signal.

3.1.2 Clocking components and protocols

The clocking components and protocols provide a mechanism for systems to regulate the execution rate of components within a system. Clocking includes the concept of clock rates, dividers to change clock rates, and timers to generate callbacks based on those clock rates.

If the MasterClock component is instantiated in a system, it provides a consistent master clock rate. Although this rate is not defined, you can consider this to be 1Hz. ClockDivider components are able to convert this clock rate into a new rate using a multiplier and divider. You can cascade ClockDivider components to produce many different clock rates within a system. The maximum ratio of any two clocks in the system must be less than 2^{32}.

ClockTimer components can be instantiated by a component and connected to any MasterClock or ClockDivider output. The ClockTimer can then be used to generate callbacks after a given number of ticks of that clock. The ClockTimer can invoke a behavior on the component to permit the component to perform work. The component can then request the ClockTimer to repeat its count.

All clocking components that communicate use an opaque ClockSignal protocol. ClockSignal protocols have no behaviors that can be invoked by the user.

3.1.3 Debug interface protocols

LISA components can expose aspects of their internal state to a debugger so that they become visible and usable in the debugger. Some aspects are supported by the native LISA language, and some are supported by debug interface protocols. For more information, see *Debug interface protocols* on page 3-10.

3.2 Signaling Protocols

This section describes the common signaling protocols within the Fast Models Portfolio. It contains the following sections:

- *Signal protocol*
- *StateSignal protocol*
- *Value protocol*
- *ValueState protocol*.

3.2.1 Signal protocol

The Signal protocol has the following behavior:

`setValue(enum sg::Signal::State) : void`

indicates a state change.

Legal values for `sg::Signal::State` are:

- `sg::Signal::Set`
- `sg::Signal::Clear`

3.2.2 StateSignal protocol

The StateSignal protocol has two behaviors:

`setValue(enum sg::Signal::State) : void`

indicates a value change

`getValue() : sg::Signal::State`

reads the current value.

3.2.3 Value protocol

The Value protocol has the following behavior:

`setValue(uint32_t value)`

indicates a state change.

3.2.4 ValueState protocol

The ValueState protocol has the following behaviors:

`setValue(uint32_t value) : void`

indicates a value change

`getValue() : uint32_t`

read the current value state.

3.3 Clocking components and protocols

This section describes the common clocking components and protocols within Fast Models. It contains the following sections:

- *MasterClock component*
- *ClockDivider component* on page 3-5
- *ClockTimer component* on page 3-6
- *ClockTimer64 component* on page 3-8.

3.3.1 MasterClock component

The MasterClock component provides a single ClockSignal output that can be used to drive the ClockSignal input of ClockDividers, ClockTimers and other clocking components. The rate of the MasterClock is not defined because all clocking is relative, but can be considered to be 1Hz.

A system might contain more than one MasterClock, all of which generate the same ClockSignal rate.

Figure 3-1 shows a view of the component in System Canvas.

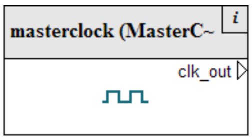


Figure 3-1 MasterClock in System Canvas

This component is written in C++.

Ports

Table 3-1 provides a brief description of the ports. For more information, see the component documentation.

Table 3-1 MasterClock ports

Name	Port protocol	Type	Description
clk_out	ClockSignal	master	master clock rate

Additional protocols

The MasterClock component has no additional protocols.

Parameters

The MasterClock component has no parameters.

Registers

The MasterClock component has no registers.

Debug features

The MasterClock component has no debug features.

Verification and testing

The MasterClock component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The MasterClock component has no effect on the performance of a PV system.

Library dependencies

The MasterClock component has no dependencies on external libraries.

3.3.2 ClockDivider component

The ClockDivider component uses a configurable ratio to convert the ClockSignal rate at its input to a new ClockSignal rate at its output. Changes to the input rate or ratio take effect immediately and clocking components dependent on the output rate continue counting at the new rate.

For examples of the use of ClockDividers, see the `VEMotherBoard.lisa` component in the `%PVLIB_HOME%\examples\FVP_VE\LISA` directory of your Fast Models distribution. On Linux, use the `$PVLIB_HOME/examples/FVP_VE/LISA` directory instead.

Figure 3-2 shows a view of the component in System Canvas.

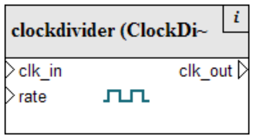


Figure 3-2 ClockDivider in System Canvas

This component is written in C++.

Ports

Table 3-2 provides a brief description of the ClockDivider component ports.

Table 3-2 ClockDivider ports

Name	Port protocol	Type	Description
clk_in	ClockSignal	slave	source clock rate
clk_out	ClockSignal	master	output clock rate
rate	ClockRateControl	slave	permits the clock divider ratio to be changed dynamically

ClockRateControl protocol

The ClockRateControl protocol can be used to set the ratio of a ClockDivider component.

`set(uint32_t mul, uint32_t div) : void`

sets the multiplier/divider that determines the clock divider ratio:

$\texttt{clk\_out}_{freq} = \texttt{clk\_in}_{freq} * \texttt{mul} / \texttt{div}$

Parameters

Table 3-3 provides a description of the configuration parameters for the ClockDivider component.

Table 3-3 ClockDivider configuration parameters

Parameter name	Description	Type	Allowed value	Default value
mul	clock rate multiplier	Integer	uint_32	1
div	clock rate divider	Integer	uint_32	1

Registers

The ClockDivider component has no registers.

Debug features

The ClockDivider component has no debug features.

Verification and testing

The ClockDivider component was tested as part of the VE example system by running VE test suites and by booting operating systems.

Performance

The ClockDivider component does not normally incur a runtime performance cost. However, reprogramming the clock rate causes all related clocks and timers to be recalculated.

Library dependencies

The ClockDivider component has no dependencies on external libraries.

3.3.3 ClockTimer component

The ClockTimer component provides a mechanism for other components to schedule a callback after a number of ticks at a given ClockSignal rate.

Figure 3-3 shows a view of the component in System Canvas.

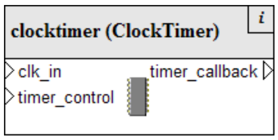


Figure 3-3 ClockTimer in System Canvas

This component is written in C++.

Ports

Table 3-4 provides a brief description of the ClockTimer component ports.

Table 3-4 ClockTimer ports

Name	Port protocol	Type	Description
timer_callback	TimerCallback	slave	port on which a signal is sent after the number of scheduled ticks has elapsed
timer_control	TimerControl	slave	permits the timer to be set, canceled and queried
clk_in	ClockSignal	slave	determines the tick rate of the timer

TimerControl protocol

The TimerControl protocol is used to control the actions of a ClockTimer component. It permits a timer to be set to schedule a callback after a given number of ticks at the rate of the clock input.

If a timer is set while it is counting, it starts counting the new number of ticks without sending the original callback. Cancelling a timer when it is not active has no effect.

The TimerControl protocol has the following behaviors:

`set(uint32_t ticks) : void`

sets the timer to countdown the given number of ticks

`cancel()` cancels an active timer, preventing the callback being invoked

`isSet() : bool`

checks whether a timer is set to generate a callback

`remaining() : uint32_t`

returns how many ticks remain before the callback is invoked.

TimerCallback protocol

The TimerCallback protocol is used to invoke a behavior on the component which set the timer. The timer_control port of the timer must not be used during the invoked behavior.

`signal() : uint32_t`

Invoked by the timer when the timer countdown expires. The invoked behavior returns the number of ticks after which it is called again or 0 to make the timer one-shot.

Parameters

The ClockTimer component has no parameters.

Registers

The ClockTimer component has no registers.

Debug features

The ClockTimer component has no debug features.

Verification and testing

Validation of the ClockTimer component consisted of booting operating systems and VE test suites on a VE model that contained a ClockTimer component.

Performance

An active ClockTimer component incurs no simulation overhead. For best performance, avoid having your performance-critical code frequently cancel timers or query the number of remaining ticks.

Library dependencies

The ClockTimer component has no dependencies on external libraries.

3.3.4 ClockTimer64 component

The ClockTimer64 component provides a mechanism for other components to schedule a callback after a number of ticks at a given ClockSignal rate.

Figure 3-3 on page 3-6 shows a view of the component in System Canvas.

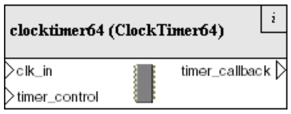


Figure 3-4 ClockTimer64 in System Canvas

This component is written in C++.

Ports

Table 3-4 on page 3-7 provides a brief description of the ClockTimer64 component ports.

Table 3-5 ClockTimer ports

Name	Port protocol	Type	Description
timer_callback	TimerCallback64	master	port on which a signal is sent after the number of scheduled ticks has elapsed
timer_control	TimerControl64	slave	permits the timer to be set, canceled and queried
clk_in	ClockSignal	slave	determines the tick rate of the timer

TimerControl64 protocol

The TimerControl64 protocol is used to control the actions of a ClockTimer64 component. It permits a timer to be set to schedule a callback after a given number of ticks at the rate of the clock input.

If a timer is set while it is counting, it starts counting the new number of ticks without sending the original callback. Cancelling a timer when it is not active has no effect.

The TimerControl64 protocol has the following behaviors:

`set(uint64_t ticks) : void`
: sets the timer to countdown the given number of ticks

`cancel()`
: cancels an active timer, preventing the callback being invoked

`isSet() : bool`
: checks whether a timer is set to generate a callback

`remaining() : uint64_t`
: returns how many ticks remain before the callback is invoked.

TimerCallback64 protocol

The TimerCallback64 protocol is used to invoke a behavior on the component which set the timer. The timer_control port of the timer must not be used during the invoked behavior.

`signal() : uint64_t`
: Invoked by the timer when the timer countdown expires. The invoked behavior returns the number of ticks after which it is called again or 0 to make the timer one-shot.

Parameters

The ClockTimer64 component has no parameters.

Registers

The ClockTimer64 component has no registers.

Debug features

The ClockTimer64 component has no debug features.

Verification and testing

This component has been tested using internal unit tests.

Performance

An active ClockTimer64 component incurs no simulation overhead. For best performance, avoid having your performance-critical code frequently cancel timers or query the number of remaining ticks.

Library dependencies

The ClockTimer64 component has no dependencies on external libraries.

3.4 Debug interface protocols

LISA components can expose aspects of their internal state to a debugger so that they become visible and usable in the debugger.

Some aspects are supported by the native LISA language, and some are supported by debug interface protocols:

- Registers and other state variables. See the REGISTER declaration in the section relating to resources in the *LISA+ Language for Fast Models Reference Manual*, and pay special attention to the `read_function` and `write_function` attributes.
- Memories and other memory-like objects. See the MEMORY declaration in the section relating to resources in the *LISA+ Language for Fast Models Reference Manual*, and pay special attention to the `read_function` and `write_function` attributes.
- Disassembly. See *CADIProtocol* and *CADIDisassemblerProtocol* on page 3-12.
- Instruction count. See *CADIProtocol*.
- Setting, configuring and clearing of breakpoints of any kind, for example, code breakpoints, register breakpoints, watchpoints. See *CADIProtocol*.
- Single stepping and instruction stepping support. See *CADIProtocol*.

If displaying and editing memory and registers is sufficient, it is not necessary to implement the CADIProtocol and CADIDisassemblerProtocol. For the other debug features listed, the CADIProtocol and CADIDisassemblerProtocol must be implemented by the component to enable them.

The CADIProtocol and CADIDisassemblerProtocol permit you to implement the features on a *Component Architecture Debug Interface* (CADI) interface level. This means that the component code takes full responsibility for the implementation of these interfaces. For more information about individual function calls, see the *Component Architecture Debug Interface v2.0 Developer Guide*,
`http://infocenter.arm.com/help/topic/com.arm.doc.dui0444-/index.html`.

Note

All protocol behaviors, that is, all sets of functionalities, that are listed in *CADIProtocol* are optional. A component only has to implement the set of functions for the functionality that it intends to support.

3.4.1 CADIProtocol

You must define an internal slave port of this type, and the name of this port must always be `cadi_port`. In this port, the following functionality can be implemented:

```
    optional slave behavior CADIBptGetList(uint32_t, uint32_t, uint32_t *,
eslapi::CADIBptDescription_t *):eslapi::CADIReturn_t;
    optional slave behavior CADIBptRead(eslapi::CADIBptNumber_t,
eslapi::CADIBptRequest_t *):eslapi::CADIReturn_t;
    optional slave behavior CADIBptSet(eslapi::CADIBptRequest_t *,
eslapi::CADIBptNumber_t *):eslapi::CADIReturn_t;
    optional slave behavior CADIBptClear(eslapi::CADIBptNumber_t):eslapi::CADIReturn_t;
    optional slave behavior CADIBptConfigure(eslapi::CADIBptNumber_t,
eslapi::CADIBptConfigure_t):eslapi::CADIReturn_t;
    optional slave behavior CADIModifyTargetFeatures(eslapi::CADITargetFeatures_t
*):eslapi::CADIReturn_t;
```

All these functions need to be implemented to enable any kind of breakpoint for the component. The component needs to maintain and keep track of all existing breakpoints. For example:

- `CADIBptSet()` and `CADIBptClear()` are used to add and remove breakpoints
- `CADIBptConfigure()` is used to enable and disable breakpoints
- `CADIBptGetList()` and `CADIBptRead()` return the current state of the breakpoint list.

The component must also implement `CADIModifyTargetFeatures`. This function permits you to override the automatic default `CADITargetFeatures_t` that is provided for this component by System Generator just before it is returned to the debugger. Specifically, a component that wants to support any kind of breakpoint must override the `handledBreakpoints` and `nrBreakpointsAvailable` fields of `CADITargetFeatures_t`. For example:

```
targetFeatures->handledBreakpoints = CADI_TARGET_FEATURE_BPT_PROGRAM |
CADI_TARGET_FEATURE_BPT_REGISTER;
// code and register breakpoints supported
targetFeatures->nrBreakpointsAvailable = 0x7fffffff;
// virtually infinite number of breakpoints supported.
```

By default, LISA components do not support any type of breakpoints. By implementing `CADIBpt...()` functions, breakpoint support of any kind can be added. In addition to implementing the stated functions, the component must call `simBreakpointHit(bptNumber)` and then `simHalt()` when an enabled breakpoint is hit. On a breakpoint hit the component must first call `simBreakpointHit()` for each breakpoint that was hit (one or more, usually just one) and then call `simHalt()` once after all `simBreakpointHit()` calls. The `simHalt()` call must always be the last call in the sequence.

A component that wants to provide disassembly must implement the following `CADIGetDisassembler()` behavior and return a CADIDisassembler interface implementation. This is automatically placed behind the `CADI::CADIGetDisassembler()` and the `CADI::ObtainInterface(“eslapi.CADIDisassembler2”)` functions.

```
    optional slave behavior CADIGetDisassembler():eslapi::CADIDisassembler*;
```

To do this, instantiate a `CADIDisassemblerAdapter` object in behavior `init` and return its address in this `CADIGetDisassembler()` function. This object must point to an internal slave port that implements the CADIDisassemblerProtocol protocol. See Example 3-1:

Example 3-1 Skeleton code for implementing disassembly

```
component FOO
{
    behavior init()
    {
        disassemblerAdapter = new
CADIDisassemblerAdapter(disassPort.getAbstractInterface());
        // ...
    }
    internal slave port <CADIProtocol> cadi_port
    {
        slave behavior CADIGetDisassembler():eslapi::CADIDisassembler*
        {
            return disassemblerAdapter;
        }
        // ...
    }
    internal slave port<CADIDisassemblerProtocol> disassPort
    {
        // ...
    }
}
```

The following function implements the instruction *stepping a component*. It must set up an internal state that stops the simulation when the requested number of instructions is executed completely (exactly like a breakpoint). It must call `simRun()` from within `CADIExecSingleStep()` after setting up this stepping state, and later it must call `simHalt()` when the execution of the required number of instructions is finished.

```
    optional slave behavior CADIExecSingleStep(uint32_t instructionCount, int8_t
stepCycle, int8_t stepOver):eslapi::CADIReturn_t;
```

The following function is used for debugging purposes only, and should not be implemented. The function must not alter the state of any component in any way.

```
    optional slave behavior callbackModeChange(uint32_t newMode,
eslapi::CADIBptNumber_t bptNumber);
```

By implementing any of the following functions, the component can enable the instruction and cycle count display:

```
    optional slave behavior CADIGetInstructionCount(uint64_t
&instructionCount):eslapi::CADIReturn_t;
    optional slave behavior CADIGetCycleCount(uint64_t &instructionCount, bool
systemCycles):eslapi::CADIReturn_t;
```

——— **Note** ———

Fast Models systems are not cycle accurate, so you usually only implement an instruction counter, if at all.

3.4.2 CADIDisassemblerProtocol

The following functions are in a different port, of type CADIDisassemblerProtocol, that can have any name and only need to be implemented when disassembly must be exposed in the debugger. All these functions must be implemented if disassembly is supported. None of them is optional. The functionality of this port is then exposed by `CADIProtocol::CADIGetDisassembler()`. See *CADIProtocol* on page 3-10 for information on how to use this port and `CADIDisassemblerAdapter`.

In the function `slave behavior GetType(): eslapi::CADIDisassemblerType;` components must always return `eslapi::CADI_DISASSEMBLER_TYPE_STANDARD`.

The function `slave behavior GetModeCount(): uint32_t;` returns the number of supported disassembler modes, and at least 1 mode must be returned.

The function `slave behavior GetModeNames(eslapi::CADIDisassemblerCB *callback_);` returns information about all supported modes. A component that only supports one mode calls, for example, `callback_->ReceiveModeName(0, "Normal");` only once. This is similar for multiple modes with different names and Ids.

The function `slave behavior GetCurrentMode(): uint32_t;` returns the most suitable mode of disassembly at the time, based on the current state variables of the component:

For the function:

```
slave behavior GetSourceReferenceForAddress(eslapi::CADIDisassemblerCB *callback_, const
eslapi::CADIAddr_t &address): eslapi::CADIDisassemblerStatus;
```

components must return `eslapi::CADI_DISASSEMBLER_STATUS_ERROR;`

For the function:

```
slave behavior GetAddressForSourceReference(const char *sourceFile, uint32_t sourceLine,
eslapi::CADIAddr_t &address): eslapi::CADIDisassemblerStatus;
```

components must return `eslapi::CADI_DISASSEMBLER_STATUS_ERROR;`

The following function is the main disassembler function:

```
slave behavior GetDisassembly(eslapi::CADIDisassemblerCB *callback_,
                              const eslapi::CADIAddr_t &address,
                              eslapi::CADIAddr_t &nextAddr,
                              const uint32_t mode,
                              uint32_t desiredCount):
                              eslapi::CADIDisassemblerStatus;
```

The component must call `callback_` for all disassembler lines for the specified address and `desiredCount`, and it must finally set `nextAddr` to the next disassembled address at that point after the requested block.

```
// Query if an instruction is a call instruction
    slave behavior GetInstructionType(const eslapi::CADIAddr_t &address,
eslapi::CADIDisassemblerInstructionType &insn_type): eslapi::CADIDisassemblerStatus;
```

Components should return `insn_type = eslapi::CADI_DISASSEMBLER_INSTRUCTION_TYPE_NOCALL;` and return `eslapi::CADI_DISASSEMBLER_STATUS_OK;`

See the *Component Architecture Debug Interface v2.0 Developer Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0444-/index.html` for the exact semantics of all the functions and parameters. The information in this section is intended only to provide additional information for the LISA component use case.

Chapter 4
Processor Components

This chapter introduces the *Code Translation* (CT) processor components. It contains the following sections:

- *Additional protocols* on page 4-109.

4.1 About the Code Translation processor components

This chapter provides an overview of the *Programmer's View* (PV) processor models of the following ARM processors in Fast Models:

- Cortex™-A15
- Cortex-A9
- Cortex-A8
- Cortex-A7
- Cortex-A5
- Cortex-R7
- Cortex-R5
- Cortex-R4
- Cortex-M4
- Cortex-M3
- ARMv7-A AEM
- ARM1176JZF-S™
- ARM1136JF-S™
- ARM968E-S™
- ARM926EJ-S™.

For more information about the functionality of the hardware that the models simulate, see the relevant processor technical reference manual.

PV models of processors and devices match functional behavior with what can be observed from software. Accuracy in timing is sacrificed to achieve fast simulation speeds.

PV models translate A32 instructions on the fly and cache the translation to enable fast execution of A32 code. They also use efficient PV bus models to enable fast access to memory and devices.

The PV models implement most of the processor features but differ in certain key ways to permit the models to run faster:

- timing is approximate
- caches, including smartcache, are implemented in selected processor models, although cache control registers are implemented in all processor models
- write buffers are not implemented
- micro-architectural features, such as MicroTLB or branch cache, are not implemented
- by default, device-accurate modeling of multiple TLBs is turned off to improve simulation performance
- the model uses a simplified view of the external buses
- except for the Cortex-A9 and Cortex-A5 processors, there is a single memory access port combining instruction, data, DMA and peripheral access
- the Cortex-A9 model has two memory access ports, but only one is used
- for processors that support Jazelle, only trivial implementations are implemented
- the Cortex-A15 processor, Cortex-A7 processor and the ARMv7-A - *Architecture Envelope Model* (AEM) have a full CP14 implementation.

Performance counters are only partially implemented and only on certain processors.

4.2 ARMCortexA15x*n*CT

Figure 4-1 shows a view of the ARMCortexA15x*n*CT component in the Fast Models Portfolio, with all vectored ports collapsed. This is a single component that represents a Cortex-A15 multiprocessor containing from one to four processors. The x*n* in the component name indicates how many processors are included in the component, so choose the one with the required number of processors for your system. The component is based on r2p0 of the Cortex-A15 processor. All variants of the ARMCortexA15x*n*CT component are described in this section, the only difference being the number of processors.

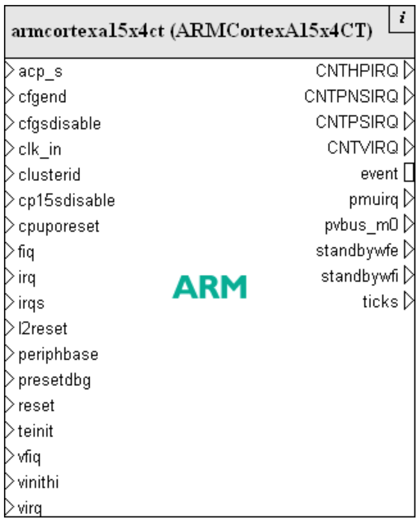


Figure 4-1 ARMCortexA15CT in System Canvas

This component is written in C++.

4.2.1 Ports

Table 4-1 provides a brief description of the ports in the ARMCortexA15x*n*CT component. For more information, see the processor technical reference manual.

Table 4-1 ARMCortexA15x*n*CT ports

Name	Port Protocol	Type	Description
`CNTHPIRQ[0-3]`	Signal	master	Hypervisor physical timer event
`CNTPNSIRQ[0-3]`	Signal	master	Non-secure physical timer event
`CNTPSIRQ[0-3]`	Signal	master	Secure physical timer event
`CNTVIRQ[0-3]`	Signal	master	Virtual timer event
`acp_s`	PVBus	slave	*Advanced eXtensible Interface* (AXI™) ACP slave port
`cfgend[0-3]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgsdisable`	Signal	slave	Disable write access to some *Generic Interrupt Controller* (GIC) registers.
`clk_in`	ClockSignal	slave	Main processor clock input.

Table 4-1 ARMCortexA15x*n*CT ports (continued)

Name	Port Protocol	Type	Description
`clusterid`	Value	slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
`cntvalueb`	Private	slave	Synchronous counter value. This must be connected to the MemoryMappedCounterModule component.
`cpuporeset[0-3]`	Signal	slave	Power on reset. Initializes all the processor logic, including the NEON and VFP logic, Debug, PTM, breakpoint and watchpoint logic in the processor CLK domain.
`cp15sdisable[0-3]`	Signal	slave	Disable write access to some secure cp15 registers.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`fiq[0-3]`	Signal	slave	Processor FIQ signal input.
`irq[0-3]`	Signal	slave	Processor IRQ signal input.
`irqs[0-223]`	Signal	slave	Shared peripheral interrupts.
`l2reset`	Signal	slave	Reset shared L2 memory system, interrupt controller and timer logic.
`periphbase`	Value	slave	Base of private peripheral region.
`pmuirq[0-3]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`presetdbg`	Signal	slave	Initializes the shared debug APB, Cross Trigger Interface (CTI), and Cross Trigger Matrix (CTM) logic.
`pvbus_m0`	PVBus	master	AXI bus master channel.
`reset[0-3]`	Signal	slave	Individual processor reset signal.
`standbywfe[0-3]`	Signal	master	Indicates if a processor is in *Wait For Event* (WFE) state.
`standbywfi[0-3]`	Signal	master	Indicates if a processor is in *Wait For Interrupt* (WFI) state.
`teinit[0-3]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0-3]`	InstructionCount	master	Processor instruction count for visualization.

Table 4-1 ARMCortexA15x*n*CT ports (continued)

Name	Port Protocol	Type	Description
`vfiq[0-3]`	Signal	slave	Processor virtual FIQ signal input.
`vinithi[0-3]`	Signal	slave	Initialize with high vectors enabled after a reset.
`virq[0-3]`	Signal	slave	Processor virtual IRQ signal input.

4.2.2 Additional protocols

The ARMCortexA15x*n*CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.2.3 Parameters

Table 4-2 provides a description of the configuration parameters for the ARMCortexA15x*n*CT component. These parameters are set once, irrespective of the number of Cortex-A15 processors in your system. If you have multiple Cortex-A15 processors, then each processor has its own configuration parameters, as shown in Table 4-3 on page 4-7.

Table 4-2 ARMCortexA15x*n*CT parameters

Parameter	Description	Type	Allowed Value	Default Value
`CFGSDISABLE`	Disable some accesses to GIC registers.	Boolean	`true` or `false`	`false`
`CLUSTER_ID`	Processor cluster ID value.	Integer	0-15	0
`IMINLN`	Instruction cache minimum line size: false=32 bytes, true=64 bytes.	Boolean	`true` or `false`	`true`
`PERIPHBASE`	Base address of peripheral memory space.	Integer	-	`0x13080000`[a]
`dic-spi_count`	Number of shared peripheral interrupts implemented.	Integer	0-224, in increments of 32	64
`internal_vgic`	Configures whether the model of the processor contains a *Virtualized Generic Interrupt Controller* (VGIC).	Boolean	`true` or `false`	`true`
`l1_dcache-state_modelled`	Set whether L1 D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`l1_icache-state_modelled`	Set whether L1 I-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`l2_cache-size`	Set L2 cache size in bytes.	Integer	512KB, 1MB, 2MB, 4MB	`0x400000`
`l2_cache-state_modelled`	Set whether L2 cache has stateful implementation.	Boolean	`true` or `false`	`false`
`l2-data-slice`	L2 data RAM slice.	Integer	0, 1 or 2	0
`l2-tag-slice`	L2 tag RAM slice.	Integer	0 or 1	0

a. If you are using the ARMCortexA15x*n*CT component on a VE model platform, this parameter is set automatically to `0x1F000000` and is not visible in the parameter list.

Table 4-3 provides a description of the configuration parameters for each ARMCortexA15x*n*CT component processor. These parameters are set individually for each processor you have in your system.

Table 4-3 ARMCortexA15x*n*CT individual processor parameters

Parameter	Description	Type	Allowed Value	Default Value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`DBGROMADDR`	This value is used to initialize the CP15 **DBGDRAR** register. Bits[39:12] of this register specify the ROM table physical address.	Integer	`0x00000000-0xFFFFFFFF`	`0x12000003`
`DBGROMADDRV`	If `true`, this sets bits[1:0] of the CP15 **DBGDRAR** to indicate that the address is valid.	Boolean	`true` or `false`	`true`
`DBGSELFADDR`	This value is used to initialize the CP15 **DBGDSAR** register. Bits[39:17] of this register specify the ROM table physical address.	Integer	`0x00000000-0xFFFFFFFF`	`0x00010003`
`DBGSELFADDRV`	If `true`, this sets bits[1:0] of the CP15 **DBGDSAR** to indicate that the address is valid.	Boolean	`true` or `false`	`true`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`ase-present`[a]	Set whether CT model has been built with NEON™ support.	Boolean	`true` or `false`	`true`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	No limit except memory	[Empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000-0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00-0xFF`	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000-0xFFFFFFFF`	`0x0`

Table 4-3 ARMCortexA15x*n*CT individual processor parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000-0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000-0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000-0xFFFFFFFF`	`0x0F000000`
`vfp-enable_at_reset`[b]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[a]	Set whether CT model has been built with VFP support.	Boolean	`true` or `false`	`true`

a. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A15 model. The options are:

vfp present and ase present

NEON and VFPv4-D32 supported.

vfp present and ase not present

VFPv4-D16 supported.

vfp not present and ase present

Illegal. Forces vfp-present to true so model has NEON and VFPv4-D32 support.

vfp not present and ase not present

Model has neither NEON nor VFPv4-D32 support.

b. This is a model-specific behavior with no hardware equivalent.

4.2.4 Registers

The ARMCortexA15x*n*CT component provides the registers specified by the technical reference manual for the Cortex-A15 processor with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.2.5 Caches

The ARMCortexA15x*n*CT component implements L1 and L2 cache as architecturally defined.

4.2.6 Debug Features

The ARMCortexA15x*n*CT component exports a CADI debug interface.

Registers

All processor, VFP, CP14 and CP15 registers, apart from performance counter registers, are visible in the debugger. All CP14 debug registers are implemented. See the processor technical reference manual for a detailed description of available registers.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value cause execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA15x*n*CT component presents three 4GB views of virtual address space, that is, one in hypervisor, one in secure mode and one in non-secure mode.

4.2.7 Verification and Testing

The ARMCortexA15x*n*CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-A15 processor
- booting of Linux on an example system containing an ARMCortexA15x*n*CT component.

4.2.8 Performance

The ARMCortexA15x*n*CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.2.9 Library dependencies

The ARMCortexA15x*n*CT component has no dependencies on external libraries.

4.2.10 Differences between the CT model and RTL Implementations

The ARMCortexA15x*n*CT component differs from the corresponding revision of the ARM Cortex-A15 RTL implementation in the following ways:

- The GIC does not respect the CFGSDISABLE signal. This leads to some registers being accessible when they should not be.
- The Broadcast *Translation Lookaside Buffer* (TLB) or cache operations in the ARMCortexA15x*n*CT model do not cause other processors in the cluster that are asleep because of *Wait For Interrupt* (WFI) to wake up.
- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- When modeling the SCU, coherency operations are represented by a memory write followed by a read to refill from memory, rather than using cache-to-cache transfers.
- ETM registers are not implemented.
- TLB bitmap registers are implemented as RAZ/WI.
- The Cortex-A15 mechanism to read the internal memory used by the Cache and TLB structures through the implementation defined region of the system coprocessor interface is not supported. This includes the RAM Index Register, IL1DATA Registers, DL1DATA Registers, and associated functionality.

4.3 ARMCortexA9MPxnCT

Figure 4-2 shows a view of the ARMCortexA9MPx*n*CT component in Fast Models, with all vectored ports collapsed. This is a single component that represents a Cortex-A9 multiprocessor containing from one to four processors. The x*n* in the component name indicates how many processors are included in the component, so choose the one with the required number of processors for your system. The component is based on r3p0 of the Cortex-A9 processor. All variants of the ARMCortexA9MPx*n*CT component are described in this section, the only difference being the number of processors.

The ARMCortexA9MPx*n*CT component implements the following additional peripherals that are not present in the basic Cortex-A9 processor:

- *Snoop Control Unit* (SCU)
- *Generic Interrupt Controller* (GIC)
- *private timer and watchdog for each processor*
- *global timer*
- *Advanced Coherency Port* (ACP).

These embedded peripherals are more fully documented elsewhere. See the processor technical reference manual.

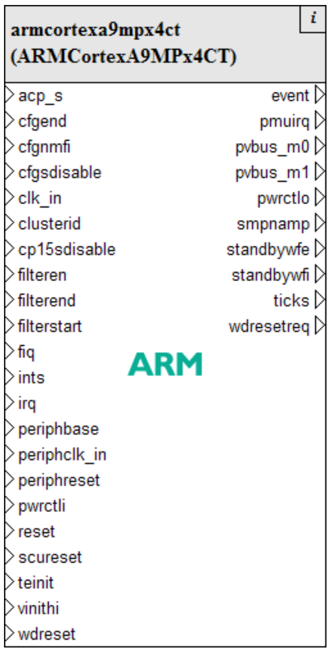


Figure 4-2 ARMCortexA9MPCT in System Canvas[1]

This component is written in C++.

1. Some of the ports are hidden.

4.3.1 Ports

Table 4-4 provides a brief description of the ports in the ARMCortexA9MPx*n*CT component. For more information, see the processor technical reference manual.

Table 4-4 ARMCortexA9MPx*n*CT ports

Name	Port protocol	Type	Description
`acp_s`	PVBus	slave	Slave channel
`cfgend[0-3]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgnmfi[0-3]`	Signal	slave	Enable nonmaskable FIQ interrupts after a reset.
`cfgsdisable`	Signal	slave	Disable write access to some GIC registers.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Value read in MPIDR register.
`cp15sdisable[0-3]`	Signal	slave	Disable write access to some cp15 registers.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`filteren`	Signal	slave	Enable filtering of address ranges between master bus ports.
`filterend`	Value	slave	End of region mapped to pvbus_m1.
`filterstart`	Value	slave	Start of region mapped to pvbus_m1.
`fiq[0-3]`	Signal	slave	Processor FIQ signal input.
`irq[0-3]`	Signal	slave	Processor IRQ signal input.
`ints[0-223]`	Signal	slave	Shared peripheral interrupts.
`periphbase`	Value	slave	Base of private peripheral region.
`periphclk_in`	ClockSignal	slave	Timer/watchdog clock rate.
`periphreset`	Signal	slave	Timer and GIC reset signal.
`pmuirq[0-3]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`pvbus_m0`	PVBus	master	AXI master 0 bus master channel.
`pvbus_m1`	PVBus	master	AXI master 1 bus master channel.
`pwrctli[0-3]`	Value	slave	Reset value for SCU processor status register.
`pwrctlo[0-3]`	Value	master	SCU processor status register bits.
`reset[0-3]`	Signal	slave	Individual processor reset signal.
`scureset`	Signal	slave	SCU reset signal.

Table 4-4 ARMCortexA9MPx*n*CT ports (continued)

Name	Port protocol	Type	Description
`smpnamp[0-3]`	Signal	master	Indicates which processors are in SMP mode.
`standbywfe[0-3]`	Signal	master	Indicates if a processor is in WFE state.
`standbywfi[0-3]`	Signal	master	Indicates if a processor is in WFI state.
`teinit[0-3]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0-3]`	InstructionCount	master	Processor instruction count for visualization.
`vinithi[0-3]`	Signal	slave	Initialize with high vectors enabled after a reset.
`wdreset[0-3]`	Signal	slave	Watchdog timer reset signal.
`wdresetreq[0-3]`	Signal	master	Watchdog timer IRQ outputs.

4.3.2 Additional protocols

The ARMCortexA9MPx*n*CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.3.3 Parameters

Table 4-5 provides a description of the configuration parameters for the ARMCortexA9MPx*n*CT component. These parameters are set once, irrespective of the number of Cortex-A9 processors in your system. If you have multiple Cortex-A9 processors, then each processor has its own parameters.

Table 4-5 ARMCortexA9MPx*n*CT parameters

Parameter	Description	Type	Allowed value	Default value
`CLUSTER_ID`	Processor cluster ID value.	Integer	0-15	0
`CFGSDISABLE`	Disable some accesses to GIC registers.	Boolean	`true` or `false`	`false`
`FILTEREN`	Enable filtering of accesses through pvbus_m0.	Boolean	`true` or `false`	`false`
`FILTERSTART`	Base of region filtered to pvbus_m0.	Integer	must be aligned on 1MB boundary	`0x0`
`FILTEREND`	End of region filtered to pvbus_m0.	Integer	must be aligned on 1MB boundary	`0x0`
`PERIPHBASE`	Base address of peripheral memory space.	Integer	-	`0x13080000`[a]
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`

Table 4-5 ARMCortexA9MPx*n*CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	`true` or `false`	`false`[b]
`dic-spi_count`	Number of shared peripheral interrupts implemented.	Integer	0-223, in increments of 32	64
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`

a. If you are using the ARMCortexA9MPx*n*CT component on a VE model platform, this parameter is set automatically to `0x1F000000` and is not visible in the parameter list.

b. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

Table 4-6 provides a description of the configuration parameters for each ARMCortexA9MPx*n*CT component processor. These parameters are set individually for each processor you have in your system.

Table 4-6 ARMCortexA9MPx*n*CT individual processor parameters[a]

Parameter	Description	Type	Allowed value	Default value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`SMPnAMP`	Set whether the processor is part of a coherent domain.	Boolean	`true` or `false`	`false`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`POWERCTLI`	Default power control state for processor.	Integer	0-3	0
`ase-present`[b]	Set whether the model has NEON support.	Boolean	`true` or `false`	`true`
`dcache-size`	Set D-cache size in bytes.	Integer	16KB, 32KB, or 64KB	`0x8000`
`icache-size`	Set I-cache size in bytes.	Integer	16KB, 32KB, or 64KB	`0x8000`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-cmd_line`[c]	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]

Table 4-6 ARMCortexA9MPx*n*CT individual processor parameters[a] (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`vfp-enable_at_reset`[d]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[b]	Set whether model has VFP support.	Boolean	`true` or `false`	`true`

a. For the ARMCortexA9MPxnCT processors, the instance name for each processor consists of the normal instance name (in the provided examples, coretile.core) with a per processor suffix. For example the first processor in the example Cortex-A9MP platform has the instance name coretile.core.cpu0.

b. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A9 model. The options are:

vfp present and ase present
NEON and VFPv3-D32 supported.

vfp present and ase not present
VFPv3-D16 supported.

vfp not present and ase present
Illegal. Forces vfp-present to true so model has NEON and VFPv3-D32 support.

vfp not present and ase not present
Model has neither NEON nor VFPv3-D32 support.

c. The value of argv[0] points to the first command line argument, not to the name of an image.

d. This is a model specific behavior with no hardware equivalent.

4.3.4 Registers

The ARMCortexA9MPx*n*CT component provides the registers specified by the technical reference manual for the Cortex-A9 processor with the following exceptions:

- integration and test registers are not implemented.

4.3.5 Caches

The ARMCortexA9MPx*n*CT component implements L1 cache as architecturally defined, but does not implement L2 cache. If you require a L2 cache you can add a PL310 Level 2 Cache Controller component.

4.3.6 Debug features

The ARMCortexA9MPx*n*CT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

The ARMCortexA9MPx*n*CT component also exports the SCU, Watchdog/Timer and GIC registers.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA9MPx*n*CT component presents two 4GB views of virtual memory, one as seen from secure mode and one as seen from normal mode.

4.3.7 Verification and testing

The ARMCortexA9MPx*n*CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-A9 processor
- booting of Linux on an example system containing an ARMCortexA9MPx*n*CT component.

4.3.8 Performance

The ARMCortexA9MPx*n*CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.3.9 Library dependencies

The ARMCortexA9MPx*n*CT component has no dependencies on external libraries.

4.3.10 Differences between the CT model and RTL implementations

The ARMCortexA9MPx*n*CT component differs from the corresponding revision of the ARM Cortex-A9 RTL implementation in the following ways:

- The ARMCortexA9MPx*n*CT does not implement address filtering within the SCU. The enable bit for this feature is ignored.
- The GIC does not respect the CFGSDISABLE signal. This leads to some registers being accessible when they must not be.
- The SCU enable bit is ignored. The SCU is always enabled.
- The SCU ignores the invalidate all register.
- The Broadcast TLB or cache operations in the ARMCortexA9MPx*n*CT model do not cause other processors in the cluster that are asleep because of WFI to wake up.
- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- The model cannot be configured with a 128-entry TLB.
- When modeling the SCU, coherency operations are represented by a memory write followed by a read to refill from memory, rather than using cache-to-cache transfers.

4.4 ARMCortexA9UPCT

Figure 4-3 shows a view of the ARMCortexA9UPCT component in System Canvas. This is a model of the Cortex-A9 uniprocessor, which is a single processor. The component is based on r3p0 of the Cortex-A9 processor.

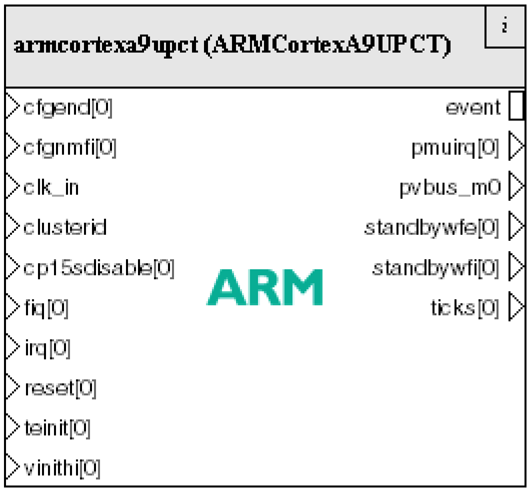


Figure 4-3 ARMCortexA9UPCT in System Canvas

This component is written in C++.

4.4.1 Ports

Table 4-7 provides a brief description of the ports in the ARMCortexA9UPCT component. For more information, see the processor technical reference manual.

Table 4-7 ARMCortexA9UPCT ports

Name	Port protocol	Type	Description
`cfgend[0]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgnmfi[0]`	Signal	slave	Enable nonmaskable FIQ interrupts after a reset.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Value read in MPIDR register.
`cp15sdisable[0]`	Signal	slave	Disable write access to some cp15 registers.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`fiq[0]`	Signal	slave	Processor FIQ signal input.
`irq[0]`	Signal	slave	Processor IRQ signal input.
`pmuirq[0]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`pvbus_m0`	PVBus	master	AXI master 0 bus master channel.
`reset[0]`	Signal	slave	Processor reset signal.
`standbywfe[0]`	Signal	master	Indicates if a processor is in WFE state.

Table 4-7 ARMCortexA9UPCT ports (continued)

Name	Port protocol	Type	Description
`standbywfi[0]`	Signal	master	Indicates if a processor is in WFI state.
`teinit[0]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0]`	InstructionCount	master	Processor instruction count for visualization.
`vinithi[0]`	Signal	slave	Initialize with high vectors enabled after a reset.

4.4.2 Additional protocols

The ARMCortexA9UPCT component has two additional protocols. See *Additional protocols* on page 4-109.

4.4.3 Parameters

Table 4-8 lists the parameters set at the processor level for the ARMCortexA9UPCT component.

Table 4-8 ARMCortexA9UPCT parameters[a]

Parameter	Description	Type	Allowed value	Default value
`CLUSTER_ID`	Processor cluster ID value.	Integer	0-15	0
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	`true` or `false`	`false`[b]
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`

a. For the ARMCortexA9UP processor, the instance name for the processor consists of the normal instance name (in the provided examples, coretile.core) with a suffix of cpu0. In the example Cortex-A9 platform the instance name is coretile.core.cpu0.

b. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

Table 4-9 provides a description of the processor configuration parameters for the ARMCortexA9UPCT component.

Table 4-9 ARMCortexA9UPCT individual processor parameters

Parameter	Description	Type	Allowed value	Default value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`

Table 4-9 ARMCortexA9UPCT individual processor parameters (continued)

Parameter	Description	Type	Allowed value	Default value
TEINIT	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	true or false	false
VINITHI	Initialize with high vectors enabled.	Boolean	true or false	false
POWERCTLI	Default power control state for processor.	Integer	0-3	0
ase-present[a]	Set whether model has NEON support.	Boolean	true or false	true
min_sync_level	Controls the minimum syncLevel.	Integer	0-3	0
semihosting-cmd_line[b]	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
semihosting-enable	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to false.	Boolean	true or false	true
semihosting-ARM_SVC	A32 SVC number for semihosting.	Integer	0x000000 - 0xFFFFFF	0x123456
semihosting-Thumb_SVC	T32 SVC number for semihosting.	Integer	0x00 - 0xFF	0xAB
semihosting-heap_base	Virtual address of heap base.	Integer	0x00000000 - 0xFFFFFFFF	0x0
semihosting-heap_limit	Virtual address of top of heap.	Integer	0x00000000 - 0xFFFFFFFF	0x0F000000
semihosting-stack_base	Virtual address of base of descending stack.	Integer	0x00000000 - 0xFFFFFFFF	0x10000000
semihosting-stack_limit	Virtual address of stack limit.	Integer	0x00000000 - 0xFFFFFFFF	0x0F000000
vfp-enable_at_reset[c]	Enable coprocessor access and VFP at reset.	Boolean	true or false	false
vfp-present[b]	Set whether the model has VFP support.	Boolean	true or false	true
dcache-size	Set D-cache size in bytes.	Integer	16KB, 32KB, or 64KB	0x8000
icache-size	Set I-cache size in bytes.	Integer	16KB, 32KB, or 64KB	0x8000

a. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A9 model. The options are:

vfp present and ase present
NEON and VFPv3-D32 supported.

vfp present and ase not present
VFPv3-D16 supported.

vfp not present and ase present
Illegal. Forces vfp-present to true so model has NEON and VFPv3-D32 support.

vfp not present and ase not present
Model has neither NEON nor VFPv3-D32 support.

b. The value of argv[0] points to the first command line argument, not to the name of an image.

c. This is a model specific behavior with no hardware equivalent.

Note

For the ARMCortexA9UP processor, the instance name for the processor consists of the normal instance name (in the provided examples, coretile.core) with a suffix of cpu0. In the example Cortex-A9 platform the instance name is coretile.core.cpu0.

4.4.4 Registers

The ARMCortexA9UPCT component provides the registers specified by the technical reference manual for the Cortex-A9 processor with the following exceptions:

- integration and test registers are not implemented.

4.4.5 Caches

The ARMCortexA9UPCT component implements L1 cache as architecturally defined, but does not implement L2 cache. If you require a L2 cache you can add a PL310 Level 2 Cache Controller component.

4.4.6 Debug features

The ARMCortexA9UPCT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA9UPCT component presents two 4GB views of virtual memory, one as seen from secure mode and one as seen from normal mode.

4.4.7 Verification and testing

The ARMCortexA9UPCT component has been tested using:

- the architecture validation suite tests for the Cortex-A9 processor
- booting of Linux on an example system containing an ARMCortexA9UPCT component.

4.4.8 Performance

The ARMCortexA9UPCT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.4.9 Library dependencies

The ARMCortexA9UPCT component has no dependencies on external libraries.

4.4.10 Differences between the CT model and RTL implementations

The ARMCortexA9UPCT component differs from the corresponding revision of the ARM Cortex-A9 RTL implementation in the following ways:

- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- The model cannot be configured with a 128-entry TLB.

4.5 ARMCortexA8CT

Figure 4-4 shows a view of the ARMCortexA8CT component in System Canvas. This component is based on r2p1 of the Cortex-A8 processor.

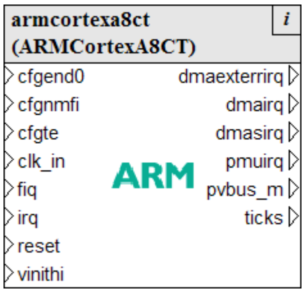


Figure 4-4 ARMCortexA8CT in System Canvas

This component is written in C++.

4.5.1 Ports

Table 4-10 provides a brief description of the ports in the ARMCortexA8CT component. For more information, see the processor technical reference manual.

Table 4-10 ARMCortexA8CT ports

Name	Port protocol	Type	Description
`clk_in`	ClockSignal	slave	clock input
`pvbus_m`	PVBus	master	master port for all memory accesses
`reset`	Signal	slave	asynchronous reset signal input
`irq`	Signal	slave	asynchronous IRQ signal input
`fiq`	Signal	slave	asynchronous FIQ signal input
`pmuirq`	Signal	master	performance monitoring unit IRQ output
`dmairq`	Signal	master	normal *PreLoad Engine* (PLE) interrupt output
`dmasirq`	Signal	master	secure PLE interrupt output
`dmaexterrirq`	Signal	master	PLE error interrupt output
`ticks`	InstructionCount	master	output that can be connected to a visualization component
`cfgend0`	Signal	slave	initialize to BE8 endianness after a reset
`cfgnmfi`	Signal	slave	enable nonmaskable FIQ interrupts after a reset
`cfgte`	Signal	slave	initialize to take exceptions in T32 state after a reset
`vinithi`	Signal	slave	initialize with high vectors enabled after a reset

4.5.2 Additional protocols

The ARMCortexA8CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.5.3 Parameters

Table 4-11 provides a description of the configuration parameters for the ARMCortexA8CT component. For more information, see the processor technical reference manual.

Table 4-11 ARMCortexA8CT parameters

Parameter	Description	Type	Allowed value	Default value
`CFGEND0`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`CFGTE`	Initialize to take exceptions in T32 state. Model starts in T32 state.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`l1_dcache-state_modelled`[a]	Include Level 1 data cache state model.	Boolean	`true` or `false`	`false`
`l1_icache-state_modelled`[a]	Include Level 1 instruction cache state model.	Boolean	`true` or `false`	`false`
`l2_cache-state_modelled`[a]	Include unified Level 2 cache state model.	Boolean	`true` or `false`	`false`
`l1_dcache-size`	Set L1 D-cache size in bytes.	Integer	16KB or32KB	`0x8000`
`l1_icache-size`	Set L1 I-cache size in bytes.	Integer	16KB or32KB	`0x8000`
`l2_cache-size`	Set L2 cache size in bytes.	Integer	128KB - 1024KB	`0x40000`
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	`true` or `false`	`false`[b]
`implements_vfp`	Set whether the model has been built with VFP and NEON support.	Boolean	`true` or `false`	`true`
`master_id`	master ID presented in bus transactions	Integer	`0x0000 - 0xFFFF`	`0x0`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	uint24_t	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	uint8_t	`0xAB`
`semihosting-cmd_line`[c]	Command line available to semihosting SVC calls.	String	No limit except memory	[Empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`

Table 4-11 ARMCortexA8CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`siliconID`	Value as read by the system coprocessor siliconID register.	Integer	uint32_t	`0x41000000`
`vfp-enable_at_reset`[d]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`

a. These three parameters permit you to define the cache state in your model. The default setting is for no caches. Any combination of true/false settings for the cache state parameters is valid. For example, if all three parameters are set to true, your model has L1 and L2 caches.

b. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

c. The value of argv[0] points to the first command line argument, not to the name of an image.

d. This is a model specific behavior with no hardware equivalent.

4.5.4 Registers

The ARMCortexA8CT component provides the registers specified by the technical reference manual for the ARM Cortex-A8 with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented
- the PLE model is purely register based and has no implemented behavior.

4.5.5 Caches

The ARMCortexA8CT component implements a PV-accurate view of the L1 and L2 caches.

4.5.6 Debug features

The ARMCortexA8CT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA8CT component presents two 4GB views of virtual memory, one as seen from secure mode and one as seen from normal mode.

4.5.7 Verification and testing

The ARMCortexA8CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-A8 processor
- booting of Linux on an example system containing an ARMCortexA8CT component.

4.5.8 Performance

The ARMCortexA8CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.5.9 Library dependencies

The ARMCortexA8CT component has no dependencies on external libraries.

4.5.10 Differences between the CT model and RTL implementations

The ARMCortexA8CT component differs from the corresponding revision of the ARM Cortex-A8 RTL implementation in the following ways:

- There is a single memory port combining instruction, data, DMA and peripheral access.
- ARMCortexA8CT L2 cache write allocate policy is not configurable. It defaults to write-allocate. Writes to the configuration register succeed but are ignored, meaning that data can be unexpectedly stored in the L2 cache.
- Unaligned accesses with the MMU disabled on the ARMCortexA8CT do not cause data aborts.

4.6 ARMCortexA7x*n*CT

Figure 4-5 shows a view of the ARMCortexA7x*n*CT component in the Fast Models Portfolio, with all vectored ports collapsed. This is a single component that represents a Cortex-A7 multiprocessor containing from one to four processors. The x*n* in the component name indicates how many processors are included in the component, so choose the one with the required number of processors for your system. All variants of the ARMCortexA7x*n*CT component are described in this section, the only difference being the number of processors.

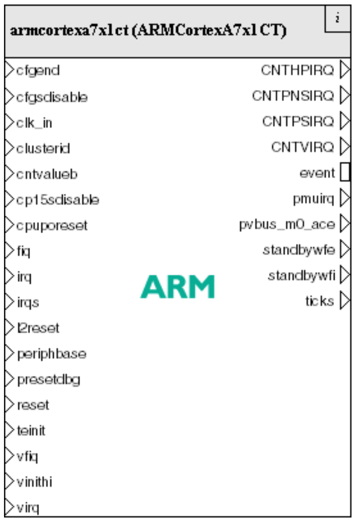


Figure 4-5 ARMCortexA7CT in System Canvas

This component is written in C++.

4.6.1 Ports

Table 4-12 provides a brief description of the ports in the ARMCortexA7x*n*CT component.

Table 4-12 ARMCortexA7x*n*CT ports

Name	Port Protocol	Type	Description
`CNTHPIRQ[0-3]`	Signal	master	Hypervisor physical timer event
`CNTPNSIRQ[0-3]`	Signal	master	Non-secure physical timer event
`CNTPSIRQ[0-3]`	Signal	master	Secure physical timer event
`CNTVIRQ[0-3]`	Signal	master	Virtual timer event
`cfgend[0-3]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgsdisable`	Signal	slave	Disable write access to some GIC registers.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
`cntvalueb`	Private	slave	Synchronous counter value. This must be connected to the MemoryMappedCounterModule component.

Table 4-12 ARMCortexA7x*n*CT ports (continued)

Name	Port Protocol	Type	Description
`cp15sdisable[0-3]`	Signal	slave	Disable write access to some secure cp15 registers.
`cpuporeset[0-3]`	Signal	slave	Power on reset. Initializes all the processor logic, including the NEON and VFP logic, Debug, PTM, breakpoint and watchpoint logic in the processor CLK domain.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENTO signals that are present on hardware.
`fiq[0-3]`	Signal	slave	Processor FIQ signal input.
`irq[0-3]`	Signal	slave	Processor IRQ signal input.
`irqs[0-223]`	Signal	slave	Shared peripheral interrupts.
`l2reset`	Signal	slave	Reset shared L2 memory system, interrupt controller and timer logic.
`periphbase`	Value	slave	Base of private peripheral region.
`pmuirq[0-3]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`pvbus_m0`	PVBus	master	AXI bus master channel.
`presetdbg`	Signal	slave	Initializes the shared debug APB, *Cross Trigger Interface* (CTI), and *Cross Trigger Matrix* (CTM) logic.
`reset[0-3]`	Signal	slave	Individual processor reset signal.
`standbywfe[0-3]`	Signal	master	Indicates if a processor is in *Wait For Event* (WFE) state.
`standbywfi[0-3]`	Signal	master	Indicates if a processor is in *Wait For Interrupt* (WFI) state.
`teinit[0-3]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0-3]`	InstructionCount	master	Processor instruction count for visualization.
`vfiq[0-3]`	Signal	slave	Processor virtual FIQ signal input.
`vinithi[0-3]`	Signal	slave	Initialize with high vectors enabled after a reset.
`virq[0-3]`	Signal	slave	Processor virtual IRQ signal input.

4.6.2 Additional protocols

The ARMCortexA7x*n*CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.6.3 Parameters

Table 4-13 provides a description of the configuration parameters for the ARMCortexA7x*n*CT component. These parameters are set once, irrespective of the number of Cortex-A7 processors in your system. If you have multiple Cortex-A7 processors, then each processor has its own configuration parameters, as shown in Table 4-14.

Table 4-13 ARMCortexA7x*n*CT parameters

Parameter	Description	Type	Allowed Value	Default Value
`CFGSDISABLE`	Disable some accesses to GIC registers.	Boolean	`true` or `false`	`false`
`CLUSTER_ID`	Processor cluster ID value.	Integer	0-15	0
`PERIPHBASE`	Base address of peripheral memory space.	Integer	`0x0000000000` - `0xFFFFFFFFFF`	`0x13080000`[a]
`internal_vgic`	Configures whether the model of the processor contains a VGIC.	Boolean	`true` or `false`	`true`
`dic-spi_count`	Number of shared peripheral interrupts implemented.	Integer	0-480, in increments of 32	64
`l1_dcache-state_modelled`	Set whether L1 D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`l1_icache-state_modelled`	Set whether L1 I-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`l2_cache-size`	Set L2 cache size in bytes.	Integer	0 (no L2 cache), 128KB, 256KB, 512KB, 1024KB	`0x800000`
`l2_cache-state_modelled`	Set whether L2 cache has stateful implementation.	Boolean	`true` or `false`	`false`

a. If you are using the ARMCortexA7x*n*CT component on a VE model platform, this parameter is set automatically to `0x2C000000` and is not visible in the parameter list.

Table 4-14 provides a description of the configuration parameters for each ARMCortexA7x*n*CT component processor. These parameters are set individually for each processor you have in your system.

Table 4-14 ARMCortexA7x*n*CT individual processor parameters

Parameter	Description	Type	Allowed Value	Default Value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`DBGROMADDR`	This value is used to initialize the CP15 **DBGDRAR** register. Bits[39:12] of this register specify the ROM table physical address.	Integer	`0x00000000` - `0xFFFFFFFF`	`0x12000003`
`DBGROMADDRV`	If `true`, this sets bits[1:0] of the CP15 **DBGDRAR** to indicate that the address is valid.	Boolean	`true` or `false`	`true`

Table 4-14 ARMCortexA7x*n*CT individual processor parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
`DBGSELFADDR`	This value is used to initialize the CP15 **DBGDSAR** register. Bits[39:17] of this register specify the ROM table physical address.	Integer	`0x00010003`	`0x00010003`
`DBGSELFADDRV`	If `true`, this sets bits[1:0] of the CP15 **DBGDSAR** to indicate that the address is valid.	Boolean	`true` or `false`	`true`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`ase-present`[a]	Set whether CT model has been built with NEON™ support.	Boolean	`true` or `false`	`true`
`l1_dcache-size`	Size of L1 D-cache.	Integer	`0x2000 - 0x10000`	`0x8000`
`l1_icache-size`	Size of L1 I-cache.	Integer	`0x2000 - 0x10000`	`0x8000`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`vfp-enable_at_reset`[b]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[a]	Set whether CT model has been built with VFP support.	Boolean	`true` or `false`	`true`

a. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A7 model. The options are:

vfp present and ase present
NEON and VFPv4-D32 supported.

vfp present and ase not present
VFPv4-D16 supported.

vfp not present and ase present
Illegal. Forces vfp-present to true so model has NEON and VFPv4-D32 support.

vfp not present and ase not present
Model has neither NEON nor VFPv4-D32 support.

b. This is a model-specific behavior with no hardware equivalent.

4.6.4 Registers

The ARMCortexA7x*n*CT component provides the registers specified by the technical reference manual for the Cortex-A7 processor with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.6.5 Caches

The ARMCortexA7x*n*CT component implements L1 and L2 cache as architecturally defined.

4.6.6 Debug Features

The ARMCortexA7x*n*CT component exports a CADI debug interface.

Registers

All processor, VFP, CP14 and CP15 registers, apart from performance counter registers, are visible in the debugger. All CP14 debug registers are implemented.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value cause execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA7x*n*CT component presents three 4GB views of virtual address space, that is, one in hypervisor, one in secure mode and one in non-secure mode.

4.6.7 Verification and Testing

The ARMCortexA7x*n*CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-A7 processor
- booting of Linux on an example system containing an ARMCortexA7x*n*CT component
- booting Linux on an example system containing an ARMCortexA7x*n*CT component and an ARMCortexA15x*n*CT *Cache Coherent Interconnect* (CCI400) model.

4.6.8 Performance

The ARMCortexA7x*n*CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.6.9 Library dependencies

The ARMCortexA7x*n*CT component has no dependencies on external libraries.

4.6.10 Differences between the CT model and RTL Implementations

The ARMCortexA7x*n*CT component differs from the corresponding revision of the ARM Cortex-A7 RTL implementation in the following ways:

- The ARMCortexA7x*n*CT component does not implement address filtering within the SCU. The enable bit for this feature is ignored.
- The GIC does not respect the CFGSDISABLE signal. This leads to some registers being accessible when they should not be.
- The Broadcast *Translation Lookaside Buffer* (TLB) or cache operations in the ARMCortexA7x*n*CT model do not cause other processors in the cluster that are asleep because of Wait For Interrupt (WFI) to wake up.
- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- ETM registers are not implemented.
- The Cortex-A7 mechanism to read the internal memory used by the Cache and TLB structures through the implementation defined region of the system coprocessor interface is not supported.

4.7 ARMCortexA5MPxnCT

Figure 4-6 shows a view of the ARMCortexA5MPx*n*CT component in Fast Models, with all vectored ports collapsed. This is a single component that represents a Cortex-A5 multiprocessor containing from one to four processors. The x*n* in the component name indicates how many processors are included in the component, so choose the one with the required number of processors for your system. The component is based on r0p0 of the Cortex-A5 processor. All variants of the ARMCortexA5MPx*n*CT component are described in this section, the only difference being the number of processors.

The ARMCortexA5MPx*n*CT component implements the following additional peripherals, that are not present in the basic Cortex-A5 processor:

- *Snoop Control Unit* (SCU)
- *Generic Interrupt Controller* (GIC)
- *private timer and watchdog for each processor*
- *global timer*
- *Advanced Coherency Port* (ACP).

These embedded peripherals are more fully documented elsewhere. See the processor technical reference manual.

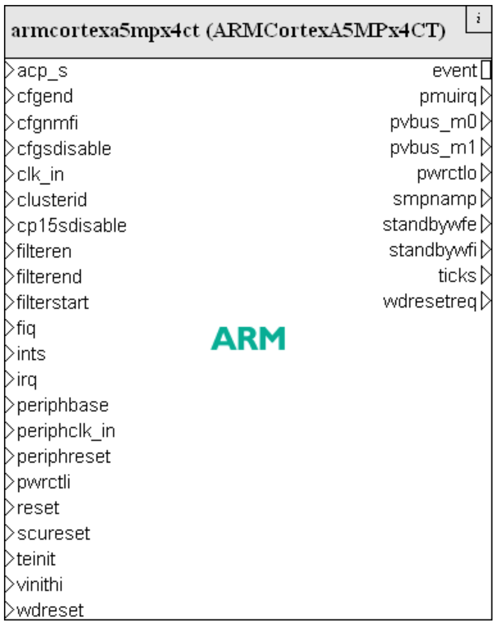


Figure 4-6 ARMCortexA5MPCT in System Canvas[1]

This component is written in C++.

1. Array ports have been collapsed in this diagram.

4.7.1 Ports

Table 4-15 provides a brief description of the ports in the ARMCortexA5MPx*n*CT component. For more information, see the processor technical reference manual.

Table 4-15 ARMCortexA5MPx*n*CT ports

Name	Port protocol	Type	Description
`acp_s`	PVBus	slave	Slave channel
`cfgend[0-3]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgnmfi[0-3]`	Signal	slave	Enable nonmaskable FIQ interrupts after a reset.
`cfgsdisable`	Signal	slave	Disable write access to some GIC registers.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Value read in MPIDR register.
`cp15sdisable[0-3]`	Signal	slave	Disable write access to some cp15 registers.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`filteren`	Signal	slave	Enable filtering of address ranges between master bus ports.
`filterend`	Value	slave	End of region mapped to pvbus_m1.
`filterstart`	Value	slave	Start of region mapped to pvbus_m1.
`fiq[0-3]`	Signal	slave	Processor FIQ signal input.
`irq[0-3]`	Signal	slave	Processor IRQ signal input.
`ints[0-223]`	Signal	slave	Shared peripheral interrupts.
`periphbase`	Value	slave	Base of private peripheral region.
`periphclk_in`	ClockSignal	slave	Timer/watchdog clock rate.
`periphreset`	Signal	slave	Timer and GIC reset signal.
`pmuirq[0-3]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`pvbus_m0`	PVBus	master	AXI master 0 bus master channel.
`pvbus_m1`	PVBus	master	AXI master 1 bus master channel.
`pwrctli[0-3]`	Value	slave	Reset value for SCU processor status register.
`pwrctlo[0-3]`	Value	master	SCU processor status register bits.
`reset[0-3]`	Signal	slave	Individual processor reset signal.
`scureset`	Signal	slave	SCU reset signal.

Table 4-15 ARMCortexA5MPx*n*CT ports (continued)

Name	Port protocol	Type	Description
`smpnamp[0-3]`	Signal	master	Indicates which processors are in SMP mode.
`standbywfe[0-3]`	Signal	master	Indicates if a processor is in WFE state.
`standbywfi[0-3]`	Signal	master	Indicates if a processor is in WFI state.
`teinit[0-3]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0-3]`	InstructionCount	master	Processor instruction count for visualization.
`vinithi[0-3]`	Signal	slave	Initialize with high vectors enabled after a reset.
`wdreset[0-3]`	Signal	slave	Watchdog timer reset signal.
`wdresetreq[0-3]`	Signal	master	Watchdog timer IRQ outputs.

4.7.2 Additional protocols

The ARMCortexA5MPx*n*CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.7.3 Parameters

Table 4-16 provides a description of the configuration parameters for the ARMCortexA5MPx*n*CT component. These parameters are set once, irrespective of the number of Cortex-A5 processors in your system. If you have multiple Cortex-A5 processors, then each processor has its own parameters.

Table 4-16 ARMCortexA5MPx*n*CT parameters

Parameter	Description	Type	Allowed value	Default value
`CLUSTER_ID`	Processor cluster ID value.	Integer	0-15	0
`CFGSDISABLE`	Disable some accesses to GIC registers.	Boolean	`true` or `false`	`false`
`FILTEREN`	Enable filtering of accesses through pvbus_m0.	Boolean	`true` or `false`	`false`
`FILTERSTART`	Base of region filtered to pvbus_m0.	Integer	must be aligned on 1MB boundary	`0x0`
`FILTEREND`	End of region filtered to pvbus_m0.	Integer	must be aligned on 1MB boundary	`0x0`
`PERIPHBASE`	Base address of peripheral memory space.	Integer	-	`0x13080000`[a]
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`

Table 4-16 ARMCortexA5MPx*n*CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	`true` or `false`	`false`[b]
`dic-spi_count`	Number of shared peripheral interrupts implemented.	Integer	0-223, in increments of 32	64
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`

a. If you are using the ARMCortexA5MPx*n*CT component on a VE model platform, this parameter is set automatically to `0x1F000000` and is not visible in the parameter list.

b. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

Table 4-17 provides a description of the configuration parameters for each ARMCortexA5MPx*n*CT component processor. These parameters are set individually for each processor you have in your system.

Table 4-17 ARMCortexA5MPx*n*CT individual processor parameters

Parameter	Description	Type	Allowed value	Default value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`SMPnAMP`	Set whether the processor is part of a coherent domain.	Boolean	`true` or `false`	`false`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`POWERCTLI`	Default power control state for processor.	Integer	0-3	0
`ase-present`[a]	Set whether the model has NEON support.	Boolean	`true` or `false`	`true`
`dcache-size`	Set D-cache size in bytes.	Integer	4KB, 8KB, 16KB, 32KB, or 64KB	`0x8000`
`icache-size`	Set I-cache size in bytes.	Integer	4KB, 8KB, 16KB, 32KB, or 64KB	`0x8000`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0

Table 4-17 ARMCortexA5MPx*n*CT individual processor parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`vfp-enable_at_reset`[b]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[b]	Set whether model has VFP support.	Boolean	`true` or `false`	`true`

a. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A5 model. The options are:

vfp present and ase present
NEON and VFPv3-D32 supported.

vfp present and ase not present
VFPv3-D16 supported.

vfp not present and ase present
Illegal. Forces vfp-present to true so model has NEON and VFPv3-D32 support.

vfp not present and ase not present
Model has neither NEON nor VFPv3-D32 support.

b. This is a model specific behavior with no hardware equivalent.

4.7.4 Registers

The ARMCortexA5MPx*n*CT component provides the registers specified by the technical reference manual for the Cortex-A5 processor with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.7.5 Caches

The ARMCortexA5MPx*n*CT component implements L1 cache as architecturally defined, but does not implement L2 cache. If you require a L2 cache you can add a PL310 Level 2 Cache Controller component.

4.7.6 Debug features

The ARMCortexA5MPx*n*CT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

The ARMCortexA5MPx*n*CT component also exports the SCU, Watchdog/Timer and GIC registers.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA5MPx*n*CT component presents two 4GB views of virtual memory, one as seen from secure mode and one as seen from normal mode.

4.7.7 Verification and testing

The ARMCortexA5MPx*n*CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-A5 processor
- booting of Linux on an example system containing an ARMCortexA5MPx*n*CT component.

4.7.8 Performance

The ARMCortexA5MPx*n*CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.7.9 Library dependencies

The ARMCortexA5MPx*n*CT component has no dependencies on external libraries.

4.7.10 Differences between the CT model and RTL implementations

The ARMCortexA5MPx*n*CT component differs from the corresponding revision of the ARM Cortex-A5 RTL implementation in the following ways:

- There is a single memory port combining instruction, data, DMA and peripheral access.
- The ARMCortexA5MPx*n*CT does not implement address filtering within the SCU. The enable bit for this feature is ignored.
- The GIC does not respect the CFGSDISABLE signal. This leads to some registers being accessible when they must not be.
- The SCU enable bit is ignored. The SCU is always enabled.
- The SCU ignores the invalidate all register.
- The Broadcast TLB or cache operations in the ARMCortexA5MPx*n*CT model do not cause other processors in the cluster that are asleep because of Wait For Interrupt(WFI) to wake up.
- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- When modeling the SCU, coherency operations are represented by a memory write followed by a read to refill from memory, rather than using cache-to-cache transfers.

4.8 ARMCortexA5CT

Figure 4-7 shows a view of the ARMCortexA5CT component in System Canvas. This is a model of the Cortex-A5 uniprocessor, which is a single processor. The component is based on r0p0 of the Cortex-A5 processor.

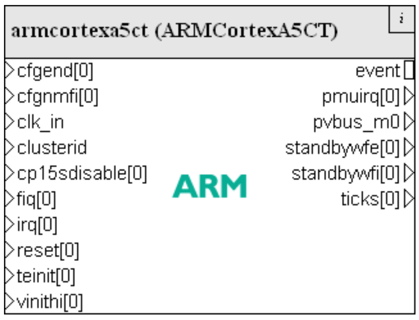


Figure 4-7 ARMCortexA5CT in System Canvas

This component is written in C++.

4.8.1 Ports

Table 4-18 provides a brief description of the ports in the ARMCortexA5CT component. For more information, see the processor technical reference manual.

Table 4-18 ARMCortexA5CT ports

Name	Port protocol	Type	Description
`cfgend[0]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgnmfi[0]`	Signal	slave	Enable nonmaskable FIQ interrupts after a reset.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Value read in MPIDR register.
`cp15sdisable[0]`	Signal	slave	Disable write access to some cp15 registers.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`fiq[0]`	Signal	slave	Processor FIQ signal input.
`irq[0]`	Signal	slave	Processor IRQ signal input.
`pmuirq[0]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`pvbus_m0`	PVBus	master	AXI master 0 bus master channel.
`reset[0]`	Signal	slave	Processor reset signal.
`standbywfe[0]`	Signal	master	Indicates if a processor is in WFE state.
`standbywfi[0]`	Signal	master	Indicates if a processor is in WFI state.
`teinit[0]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0]`	InstructionCount	master	Processor instruction count for visualization.
`vinithi[0]`	Signal	slave	Initialize with high vectors enabled after a reset.

4.8.2 Additional protocols

The ARMCortexA5CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.8.3 Parameters

Table 4-19 lists the parameters set at the processor level for the ARMCortexA5CT component.

Table 4-19 ARMCortexA5CT parameters

Parameter	Description	Type	Allowed value	Default value
`CLUSTER_ID`	Processor cluster ID value.	Integer	0-15	0
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	`true` or `false`	`false`[a]
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`

a. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

Table 4-20 provides a description of the processor configuration parameters for the ARMCortexA5CT component.

Table 4-20 ARMCortexA5CT individual processor parameters

Parameter	Description	Type	Allowed value	Default value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`POWERCTLI`	Default power control state for processor.	Integer	0-3	0
`ase-present`[a]	Set whether model has NEON support.	Boolean	`true` or `false`	`true`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]

Table 4-20 ARMCortexA5CT individual processor parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`vfp-enable_at_reset`[b]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[b]	Set whether the model has VFP support.	Boolean	`true` or `false`	`true`
`dcache-size`	Set D-cache size in bytes.	Integer	4KB, 8KB, 16KB, 32KB, or 64KB	`0x8000`
`icache-size`	Set I-cache size in bytes.	Integer	4KB, 8KB, 16KB, 32KB, or 64KB	`0x8000`

a. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A5 model. The options are:

vfp present and ase present
NEON and VFPv3-D32 supported.

vfp present and ase not present
VFPv3-D16 supported.

vfp not present and ase present
Illegal. Forces vfp-present to true so model has NEON and VFPv3-D32 support.

vfp not present and ase not present
Model has neither NEON nor VFPv3-D32 support.

b. This is a model specific behavior with no hardware equivalent.

4.8.4 Registers

The ARMCortexA5CT component provides the registers specified by the technical reference manual for the Cortex-A5 with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.8.5 Caches

The ARMCortexA5CT component implements L1 cache as architecturally defined, but does not implement L2 cache. If you require a L2 cache you can add a PL310 Level 2 Cache Controller component.

4.8.6 Debug features

The ARMCortexA5CT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexA5CT component presents two 4GB views of virtual memory, one as seen from secure mode and one as seen from normal mode.

4.8.7 Verification and testing

The ARMCortexA5CT component has been tested using:

- the architecture validation suite tests for the Cortex-A5
- booting of Linux on an example system containing an ARMCortexA5CT component.

4.8.8 Performance

The ARMCortexA5CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.8.9 Library dependencies

The ARMCortexA5CT component has no dependencies on external libraries.

4.8.10 Differences between the CT model and RTL implementations

The ARMCortexA5CT component differs from the corresponding revision of the ARM Cortex-A5 RTL implementation in the following ways:

- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.

4.9 ARMCortexR7MPxnCT

Figure 4-8 shows a view of the ARMCortexR7MPx*n*CT component in Fast Models, with all vectored ports collapsed. This is a single component that represents a Cortex-R7 multiprocessor containing from one to two processors. The x*n* in the component name indicates how many processors are included in the component, so choose the one with the required number of processors for your system. The component is based on r0p0 of the Cortex-R7 processor. All variants of the ARMCortexR7MPx*n*CT component are described in this section, the only difference being the number of processors.

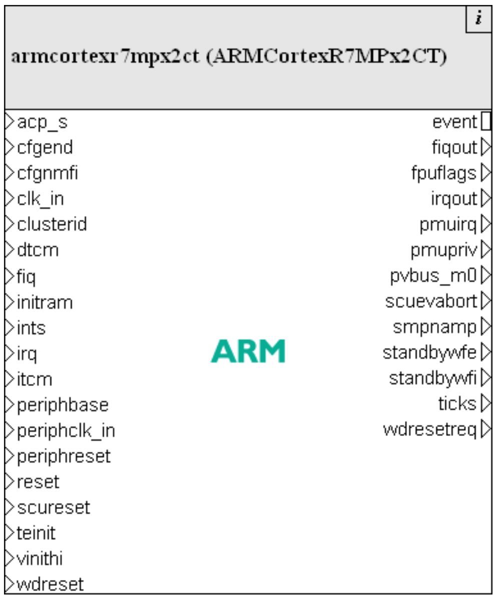


Figure 4-8 ARMCortexR7MPCT in System Canvas

This component is written in C++.

4.9.1 Ports

Table 4-21 provides a brief description of the ports in the ARMCortexR7MPx*n*CT component. For more information, see the processor technical reference manual.

Table 4-21 ARMCortexR7MPx*n*CT ports

Name	Port protocol	Type	Description
`acp_s`	PVBus	slave	Slave channel
`cfgend`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgnmfi`	Signal	slave	Enable nonmaskable FIQ interrupts after a reset.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Value read in MPIDR register.
`dtcm[0-1]`	PVBus	slave	Port to allow external components to write to data TCM.

Table 4-21 ARMCortexR7MPx*n*CT ports (continued)

Name	Port protocol	Type	Description
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`fiq[0-1]`	Signal	slave	Processor FIQ signal input.
`fiqout[0-1]`	Signal	master	Output of individual processor nFIQ from the interrupt controller.
`fpuflags[0-1]`	ValueState	master	Floating-point unit output flags.
`initram[0-1]`	Signal	slave	Initialize with ITCM enabled after reset.
`ints[0-479]`	Signal	slave	Shared peripheral interrupts.
`irq[0-1]`	Signal	slave	Processor IRQ signal input.
`irqout[0-1]`	Signal	master	Output of individual processor nIRQ from the interrupt controller.
`itcm[0-1]`	PVBus	slave	Port to allow external components to write to instruction TCM.
`periphbase`	Value	slave	Base of private peripheral region.
`periphclk_in`	ClockSignal	slave	Timer/watchdog clock rate.
`periphreset`	Signal	slave	Timer and GIC reset signal.
`pmuirq[0-1]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`pmupriv[0-1]`	StateSignal	master	This signal gives the status of the Cortex-R7 processor.
`pvbus_m0`	PVBus	master	AXI master 0 bus master channel.
`reset[0-1]`	Signal	slave	Individual processor reset signal.
`scuevabort`	Signal	master	Indicates that an external abort has occurred during a coherency eviction.
`scureset`	Signal	slave	SCU reset signal.
`smpnamp[0-1]`	Signal	master	Indicates which processors are in SMP mode.
`standbywfe[0-1]`	Signal	master	Indicates if a processor is in WFE state.
`standbywfi[0-1]`	Signal	master	Indicates if a processor is in WFI state.
`teinit`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0-1]`	InstructionCount	master	Processor instruction count for visualization.

Table 4-21 ARMCortexR7MPx*n*CT ports (continued)

Name	Port protocol	Type	Description
`vinithi[0-1]`	Signal	slave	Initialize with high vectors enabled after a reset.
`wdreset[0-1]`	Signal	slave	Watchdog timer reset signal.
`wdresetreq[0-1]`	Signal	master	Watchdog timer IRQ outputs.

4.9.2 Additional protocols

The ARMCortexR7MPx*n*CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.9.3 Parameters

Table 4-22 provides a description of the configuration parameters for the ARMCortexR7MPx*n*CT component. These parameters are set once, irrespective of the number of Cortex-R7 processors in your system. If you have multiple Cortex-R7 processors, then each processor has its own parameters.

Table 4-22 ARMCortexR7MPx*n*CT parameters

Parameter	Description	Type	Allowed value	Default value
`CLUSTER_ID`	Processor cluster ID value.	Integer	`0x0 - 0xF`	`0x0`
`LOCK_STEP`	Affects dual-processor configurations only, and ignored by single-processor configurations.	Integer	0 - Disable. Set for two independent processors. 1 - Lock Step. Appears to the system as two processors but is internally modeled as a single processor. 3 - Split Lock. Appears to the system as two processors but can be statically configured from reset either as two independent processors or two locked processors. For the model, these are equivalent to Disable and Lock Step, respectively, except for the value of build options registers. The model does not support dynamically splitting and locking the processor.	0
`MFILTEREN`	Enables filtering of address ranges.	Boolean	`true` or `false`	`false`
`MFILTEREND`	Specifies the end address for address filtering.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`MFILTERSTART`	Specifies the start address for address filtering.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`PERIPHBASE`	Base address of peripheral memory space.	Integer	`0x00000000 - 0xFFFFFFFF`	`0xAE100000`[a]

Table 4-22 ARMCortexR7MPx*n*CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`ecc_on`	Enable Error Correcting Code.	Boolean	`true` or `false`	`false`
`dic-spi_count`	Number of shared peripheral interrupts implemented.	Integer	`0x0 - 0xE`	`0x40`
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`

a. If you are using the ARMCortexR7MPx*n*CT component on a VE model platform, this parameter is set automatically to `0x1F000000` and is not visible in the parameter list.

Table 4-23 provides a description of the configuration parameters for each ARMCortexR7MPx*n*CT component processor. These parameters are set individually for each processor you have in your system.

Table 4-23 ARMCortexR7MPx*n*CT individual processor parameters[a]

Parameter	Description	Type	Allowed value	Default value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`DP_FLOAT`	Sets whether double-precision instructions are available.	Boolean	`true` or `false`	`true`
`NUM_MPU_REGION`	Sets the number of MPU regions.	Integer	12-16	16
`POWERCTLI`	Default power control state for processor.	Integer	0-3	0
`SMPnAMP`	Set whether the processor is part of a coherent domain.	Boolean	`true` or `false`	`false`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`dcache-size`	Set D-cache size in bytes.	Integer	`0x00000000 - 0x10000000`	`0x8000`
`icache-size`	Set I-cache size in bytes.	Integer	`0x00000000 - 0x10000000`	`0x8000`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`

Table 4-23 ARMCortexR7MPx*n*CT individual processor parameters[a] (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-cmd_line`[b]	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`tcm-present`	Disables the DTCM and ITCM.	Boolean	`true` or `false`	`true`
`vfp-enable_at_reset`[c]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[b]	Set whether model has VFP support.	Boolean	`true` or `false`	`true`

a. For the ARMCortexR7MPx*n*CT processors, the instance name for each processor consists of the normal instance name (in the provided examples, coretile.core) with a per processor suffix. For example the first processor in the example Cortex-R7MP platform has the instance name coretile.core.cpu0.

b. The value of argv[0] points to the first command line argument, not to the name of an image.

c. This is a model specific behavior with no hardware equivalent.

4.9.4 Registers

The ARMCortexR7MPx*n*CT component provides the registers specified by the technical reference manual for the Cortex-R7 processor with the following exceptions:

- integration and test registers are not implemented.

4.9.5 Caches

The ARMCortexR7MPx*n*CT component implements L1 cache as architecturally defined, but does not implement L2 cache. If you require a L2 cache you can add a PL310 Level 2 Cache Controller component.

4.9.6 Debug features

The ARMCortexR7MPx*n*CT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

The ARMCortexR7MPx*n*CT component also exports the SCU, Watchdog/Timer and GIC registers.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexR7MPx*n*CT component presents two 4GB views of physical memory, one as seen from secure mode and one as seen from normal mode.

4.9.7 Verification and testing

The ARMCortexR7MPx*n*CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-R7 processor
- booting of Linux on an example system containing an ARMCortexR7MPx*n*CT component.

4.9.8 Performance

The ARMCortexR7MPx*n*CT component provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.9.9 Library dependencies

The ARMCortexR7MPx*n*CT component has no dependencies on external libraries.

4.9.10 Differences between the CT model and RTL implementations

The ARMCortexR7MPx*n*CT component differs from the corresponding revision of the ARM Cortex-R7 RTL implementation in the following ways:

- The ARMCortexR7MPx*n*CT does not implement address filtering within the SCU. The enable bit for this feature is ignored.

- The GIC does not respect the CFGSDISABLE signal. This leads to some registers being accessible when they must not be.
- The SCU enable bit is ignored. The SCU is always enabled.
- The SCU ignores the invalidate all register.
- The Broadcast TLB or cache operations in the ARMCortexR7MPx*n*CT model do not cause other processors in the cluster that are asleep because of WFI to wake up.
- The RR bit in the SCTLR is ignored.
- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- The model cannot be configured with a 128-entry TLB.
- When modeling the SCU, coherency operations are represented by a memory write followed by a read to refill from memory, rather than using cache-to-cache transfers.

4.10 ARMCortexR5CT

Figure 4-9 shows a view of the ARMCortexR5CT component in the Fast Models Portfolio, with all vectored ports collapsed. This is a single component that represents a Cortex-R5 multiprocessor containing from one to two processors. The x*n* in the component name indicates how many processors are included in the component, so select the one with the required number of processors for your system. The component is based on r1p2 of the Cortex-R5 processor. All variants of the ARMCortexR5CT component are described in this section, the only difference being the number of processors.

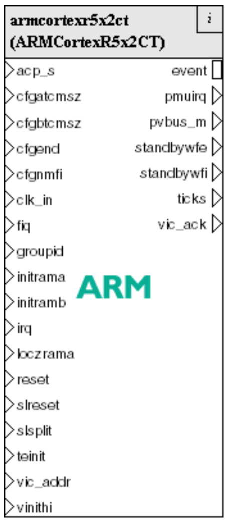


Figure 4-9 ARMCortexR5CT in System Canvas

This component is written in C++.

4.10.1 Ports

Table 4-24 provides a brief description of the ports in the ARMCortexR5CT component. For more information, see the processor technical reference manual.

Table 4-24 ARMCortexR5CT ports

Name	Port Protocol	Type	Description
`acp_s`	PVBus	Slave	ACP slave port.
`cfgatcmsz[2]`	Value	Slave	ATCM size.
`cfgbtcmsz[2]`	Value	Slave	BTCM size.
`cfgend[2]`	Signal	Slave	This signal is for EE bit initialization.
`cfgnmfi[2]`	Signal	Slave	Controls nonmaskable FIQ interrupts.
`clk_in`	ClockSignal	Slave	The clock signal connected to the clk_in port is used to determine the rate at which the processor executes instructions.
`event[2]`	Signal	Peer	This peer port of event input (and output) is for wakeup from WFE.
`fiq[2]`	Signal	Slave	This signal drives the FIQ interrupt handling of the processor.

Table 4-24 ARMCortexR5CT ports (continued)

Name	Port Protocol	Type	Description
`groupid`	Value	Slave	Group ID used for MPIDR.
`initrama[2]`	Signal	Slave	If ATCM is enabled at reset.
`initramb[2]`	Signal	Slave	If BTCM is enabled at reset.
`irq[2]`	Signal	Slave	This signal drives the interrupt handling of the processor.
`loczrama[2]`	Signal	Slave	Location of ATCM at reset.
`pmuirq[2]`	Signal	Master	Interrupt signal from performance monitoring unit.
`pvbus_m`	PVBus	Master	The processor generates bus requests on this port.
`reset[2]`	Signal	Slave	Raising this signal puts the processor into reset mode.
`slreset`	Signal	Slave	Split lock signal. Contact ARM for details.
`slsplit`	Signal	Slave	Split lock signal. Contact ARM for details.
`standbywfe[2]`	Signal	Master	This signal indicates if a processor is in wfe state.
`standbywfi[2]`	Signal	Master	This signal indicates if a processor is in WFI state.
`teinit[2]`	Signal	Slave	Default exception handling state.
`ticks[2]`	InstructionCount	Master	Connect this port to one of the two ticks ports on a visualization component to display a running instruction count.
`vic_ack[2]`	Signal	Master	Vic acknowledge port to primary VIC.
`vic_addr[2]`	ValueState	Slave	Vic address port from primary VIC.
`vinithi`	Signal	Slave	This signal controls the location of the exception vectors at reset.

4.10.2 Additional protocols

The ARMCortexR5CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.10.3 Parameters

Table 4-25 provides a description of the configuration parameters for the ARMCortexR5CT component. These parameters are set once, irrespective of the number of Cortex-R5 processors in your system. If you have multiple Cortex-R5 processors, then each processor has its own parameters.

Table 4-25 ARMCortexR5CT parameters

Parameter	Description	Type	Allowed Value	Default Value
`GROUP_ID`	Value read in GROUP ID register field, bits[15:8] of the MPIDR.	Integer	0-15	0
`INST_ENDIAN`	Controls whether the model supports the instruction endianness bit. For more information, see the *ARM Architecture Reference Manuals*.	Boolean	`true` or `false`	`true`
`LOCK_STEP`	Affects dual-processor configurations only, and ignored by single-processor configurations.	Integer	0 - Disable. Set for two independent processors. 1 - Lock Step. Appears to the system as two processors but is internally modeled as a single processor. 3 - Split Lock. Appears to the system as two processors but can be statically configured from reset either as two independent processors or two locked processors. For the model, these are equivalent to Disable and Lock Step, respectively, except for the value of build options registers. The model does not support dynamically splitting and locking the processor.	0
`MICRO_SCU`	Controls whether the effects of the MicroSCU are modeled. For more information, see the *Cortex-R5 and Cortex-R5F Technical Reference Manual*.	Boolean	`true` or `false`	`true`
`NUM_BREAKPOINTS`	Controls with how many breakpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.	Integer	2-8	3
`NUM_WATCHPOINTS`	Controls with how many watchpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.	Integer	1-8	2

Table 4-25 ARMCortexR5CT parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
`SLSPLIT`	Sets whether the model starts in split mode or locked mode. Contact ARM for details.	Boolean	`true` or `false`. If `True`, model starts up in split mode. If `False`, model starts up in locked mode. This only has an effect if the `LOCK_STEP` parameter is set to 3.	`false`
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`

Table 4-26 provides a description of the configuration parameters for each ARMCortexR5MPx*n*CT component processor. These parameters are set individually for each processor you have in your system.

Table 4-26 ARMCortexR5CT individual processor parameters

Parameter	Description	Type	Allowed Value	Default Value
`CFGATCMSZ`	Sets the size of the ATCM.	Integer	`0x00000000 - 0xE`	`0xE`
`CFGBTCMSZ`	Sets the size of the BTCM.	Integer	`0x00000000 - 0xE`	`0xE`
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGIE`	Set the reset value of the instruction endian bit.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`DP_FLOAT`	Sets whether double-precision instructions are available. For more information, see the *ARM Architecture Reference Manuals*.	Boolean	`true` or `false`. If `True`, then double precision VFP is supported. If `False`, then the VFP is single precision only.	`true`
`NUM_MPU_REGION`	Sets the number of MPU regions.	Integer	`0x00`, `0xC`, `0x10`. `0` = no MPU.	`0xC`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`

Table 4-26 ARMCortexR5CT individual processor parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
`atcm_base`[a]	Model-specific. Sets the base address of the ATCM. For more information about the processor configuration signals, see the *Cortex-R5 and Cortex-R5F Technical Reference Manual*.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x40000000`
`btcm_base`[a]	Model-specific. Sets the base address of the BTCM. For more information about the processor configuration signals, see the *Cortex-R5 and Cortex-R5F Technical Reference Manual*.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x00000000`
`dcache-size`	Set D-cache size in bytes.	Integer	`0x1000 - 0x10000`	`0x10000`
`icache-size`	Set I-cache size in bytes.	Integer	`0x1000 - 0x10000`	`0x10000`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	No limit except memory	[Empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `False`.	Boolean	`true` or `false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`

Table 4-26 ARMCortexR5CT individual processor parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`vfp-enable_at_reset`[a]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`	Set whether model has VFP support.	Boolean	`true` or `false`	`true`

a. This is a model-specific behavior with no hardware equivalent.

4.10.4 Registers

The ARMCortexR5CT component provides the registers specified by the technical reference manual for the Cortex-R5 with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.10.5 Caches

The ARMCortexR5CT component implements L1 cache as architecturally defined, but does not implement L2 cache.

4.10.6 Debug Features

The ARMCortexR5CT component exports a CADI debug interface.

Registers

All Core, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value cause execution to stop on entry to the associated exception vector.

Memory

The ARMCortexR5CT component presents a single 4GB view of memory.

4.10.7 Verification and Testing

The ARMCortexR5CT component has been tested using the architecture validation suite tests for the ARM Cortex-R5.

4.10.8 Performance

The ARMCortexR5CT component provides high performance in all areas except VFP and instruction set execution which currently does not use code translation technology.

4.10.9 Library dependencies

The ARMCortexR5CT component has no dependencies on external libraries.

4.10.10 Differences between the CT model and RTL Implementations

The ARMCortexR5CT component differs from the corresponding revision of the ARM Cortex-R5 RTL implementation in the following ways:

- The RR bit in the SCTLR is ignored.
- The Low Latency Peripheral Port is not modeled.
- The model only has a single bus master port combining instruction, data, DMA and peripheral accesses. The CP15 control registers associated with Peripheral buses preserve values but do not have any other effect.
- The model only supports static split lock and not dynamic split lock. Contact ARM for details.
- TCMs are modeled internally and the model does not support external TCMs or the ports associated with them.
- The model cannot experience an ECC error and does not support fault injection into the system, so we do not provide the ability to set error schemes for the caches or TCMs. If you require a particular value in the Build Options registers despite this limitation, please contact ARM.

4.11 ARMCortexR4CT

Figure 4-10 shows a view of the ARMCortexR4CT component in System Canvas. The component is based on r1p2 of the Cortex-R4 processor.

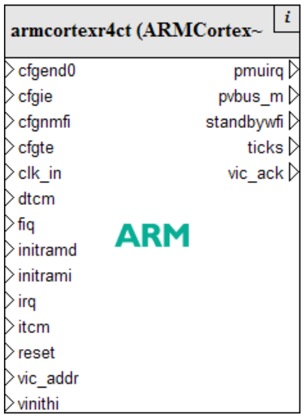


Figure 4-10 ARMCortexR4CT in System Canvas

This component is written in C++.

4.11.1 Ports

Table 4-27 provides a brief description of the ports in the ARMCortexR4CT component. For more information, see the processor technical reference manual.

Table 4-27 ARMCortexR4CT ports

Name	Port protocol	Type	Description
`clk_in`	ClockSignal	slave	clock input
`pvbus_m`	PVBus	master	master port for all memory accesses
`reset`	Signal	slave	asynchronous reset signal input
`irq`	Signal	slave	asynchronous IRQ signal input
`fiq`	Signal	slave	asynchronous FIQ signal input
`pmuirq`	Signal	master	performance monitoring unit IRQ output
`vic_addr`	ValueState	slave	address input for connection to PL192 VIC
`vic_ack`	Signal	master	acknowledge signal output for PL192 VIC
`cfgie`[a]	Signal	slave	configure instruction endianness after a reset
`ticks`	InstructionCount	master	output that can be connected to a visualization component
`cfgend0`	Signal	slave	initialize to BE8 endianness after a reset
`cfgnmfi`	Signal	slave	enable nonmaskable FIQ interrupts after a reset
`cfgte`	Signal	slave	initialize to take exceptions in T32 state after a reset
`vinithi`	Signal	slave	initialize with high vectors enabled after a reset

Table 4-27 ARMCortexR4CT ports (continued)

Name	Port protocol	Type	Description
`standbywfi`	Signal	master	signal that the processor is in standby waiting for interrupts
`initrami`	Signal	slave	initialize with ITCM enabled after reset
`initramd`	Signal	slave	initialize with DTCM enabled after reset
`itcm`	PVBus	slave	slave access to ITCM
`dtcm`	PVBus	slave	slave access to DTCM

a. This is implemented in the model, although it is optional in hardware.

4.11.2 Additional protocols

The ARMCortexR4CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.11.3 Parameters

Table 4-28 provides a description of the configuration parameters for the ARMCortexR4CT component.

Table 4-28 ARMCortexR4CT parameters

Parameter	Description	Type	Allowed value	Default value
`CFGEND0`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CFGIE`	Configure instructions as big endian.	Boolean	`true` or `false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true` or `false`	`false`
`CFGTE`	Initialize to take exceptions in T32 state. Model starts in T32 state.	Boolean	`true` or `false`	`false`
`INITRAMD`	Set or reset the INITRAMD signal.	Boolean	`true` or `false`	`false`
`INITRAMI`	Set or reset the INITRAMI signal.	Boolean	`true` or `false`	`false`
`LOCZRAMI`	Set or reset the LOCZRAMI signal.	Boolean	`true` or `false`	`false`
`NUM_MPU_REGION`	Number of MPU regions.	Integer	0, 8, 12	8
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`dcache-size`	Set D-cache size in bytes.	Integer	4KB, 8KB, 16KB, 32KB, or 64KB	`0x10000`

Table 4-28 ARMCortexR4CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`dtcm0_base`	Base address of DTCM at startup.	Integer	uint32_t	`0x00800000`
`dtcm0_size`	Size of DTCM in KB.	Integer	`0x0000 - 0x2000`	`0x8`
`icache-size`	Set I-cache size in bytes.	Integer	4KB, 8KB, 16KB, 32KB, or 64KB	`0x10000`
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true` or `false`	`false`
`implements_vfp`	Set whether the model has been built with VFP support.	Boolean	`true` or `false`	`true`
`itcm0_base`	Base address of ITCM at startup.	Integer	uint32_t	`0x00000000`
`itcm0_size`	Size of ITCM in KB.	Integer	`0x0000 - 0x2000`	`0x8`
`master_id`	master ID presented in bus transactions	Integer	`0x0000 - 0xFFFF`	`0x0`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	24-bit integer	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	8-bit integer	`0xAB`
`semihosting-cmd_line`[a]	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F0000000`
`vfp-enable_at_reset`[b]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`

a. The value of argv[0] points to the first command line argument, not to the name of an image.

b. This is a model specific behavior with no hardware equivalent.

4.11.4 Registers

The ARMCortexR4CT component provides the registers specified by the technical reference manual for the Cortex-R4 with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.11.5 Caches

The ARMCortexR4CT component implements a PV-accurate view of cache.

4.11.6 Debug features

The ARMCortexR4CT component exports a CADI debug interface.

Registers

All processor and CP15 registers are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by pseudoregisters that are available in the debugger register window. When debugger support is added to directly support processor exceptions, the pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARMCortexR4CT component presents one 4GB view of virtual memory.

4.11.7 Verification and testing

The ARMCortexR4CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-A4
- booting of uClinux on an example system containing an ARMCortexR4CT component.

4.11.8 Performance

The ARMCortexR4CT component provides high performance in all areas except with instructions in protection regions smaller than 1KB, and VFP instruction set execution which currently does not use code translation technology.

4.11.9 Library dependencies

The ARMCortexR4CT component has no dependencies on external libraries.

4.11.10 Differences between the CT model and RTL implementations

The ARMCortexR4CT component differs from the corresponding revision of the ARM Cortex-R4 RTL implementation in the following ways:

- There is a single memory port combining instruction, data, DMA and peripheral access.
- The combined AXI slave port is not supported.
- ECC and parity schemes are not supported (although the registers might be present)
- The dual processor redundancy configuration is not supported.
- The hardware refers to the TCMs as A and B. The model refers to these as ‘i’ and ‘d’.
- The RTL permits two data TCMs, B0 and B1, to be configured for extra bandwidth. These are not modeled.

4.12 ARMCortexM4CT

Figure 4-11 shows a view of the ARMCortexM4CT component in System Canvas. The component is based on r0p0 of the Cortex-M4 processor.

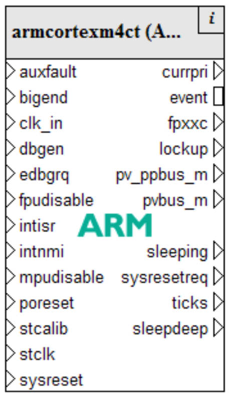


Figure 4-11 ARMCortexM4CT in System Canvas

This component is written in C++.

4.12.1 Ports

Table 4-29 provides a brief description of the ports in the ARMCortexM4CT component. For more information, see the processor technical reference manual.

Table 4-29 ARMCortexM4CT ports

Name	Port protocol	Type	Description
`auxfault`	Value	slave	auxiliary fault status information
`bigend`	Signal	slave	configure endianness after a reset
`clk_in`	ClockSignal	slave	clock input
`currpri`	Value	master	indicates the current execution priority of the processor
`edbgrq`	Signal	master	external debug request
`event`	Signal	peer	event input and output for wakeup from WFE. This port combines the TXEV and RXEVsignals
`intisr[0-239]`	Signal	slave	external interrupt signals
`intnmi`	Signal	slave	nonmaskable interrupt
`lockup`	Signal	master	asserted when processor is in lockup state
`poreset`	Signal	slave	asynchronous power-on reset signal input
`pvbus_m`	PVBus	master	master port for all memory accesses except those on the External Private Peripheral Bus
`pv_ppbus_m`	PVBus	master	master port for memory accesses on the External Private Peripheral Bus

Table 4-29 ARMCortexM4CT ports (continued)

Name	Port protocol	Type	Description
`reset`	Signal	slave	asynchronous reset signal input (not debug components)
`sleepdeep`	Signal	master	indicates that the processor is in deep sleep
`sleeping`	Signal	master	indicates that the processor is in sleep
`stcalib`	Value	slave	SysTick calibration value
`stclk`	ClockSignal	slave	reference clock input for SysTick
`sysreset`	Signal	slave	asynchronous reset signal input
`sysresetreq`	Signal	master	system reset request
`ticks`	InstructionCount	master	output that can be connected to a visualization component
`dbgen`[a]	Signal	slave	enable hardware debugger access
`fpudisable`	Signal	slave	disable FPU on next reset
`mpudisable`	Signal	slave	disable MPU on next reset
`fpxxc`	Value	master	cumulative exception flags from the *Floating Point Status and Control Register* (FPSCR). This value port combines the five RTL signals FPIXC, FPIDC, FPOFC, FPUFC, FPDZC and FPIOC.

a. Since the CT model does not provide a DAP port or halting debug capability, this signal is ignored.

4.12.2 Additional protocols

The ARMCortexM4CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.12.3 Parameters

Table 4-30 provides a description of the configuration parameters for the ARMCortexM4CT component.

Table 4-30 ARMCortexM4CT parameters

Parameter	Description	Type	Allowed value	Default value
`BB_PRESENT`	Enable bitbanding	Boolean	`true` or `false`	`true`
`BIGENDINIT`	Initialize processor to big endian mode	Boolean	`true` or `false`	`false`
`LVL_WIDTH`	Number of bits of interrupt priority.	Integer	3-8	3
`NUM_IRQ`	Number of user interrupts.	Integer	1-240	16
`NUM_MPU_REGION`	Number of MPU regions.	Integer	0,8	8

Table 4-30 ARMCortexM4CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`master_id`	master ID presented in bus transactions	Integer	`0x0000 - 0xFFFF`	`0x0`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	8-bit integer	`0xAB`
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10700000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10700000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10800000`
`vfp-present`	Set whether the model has VFP support	Boolean	`true` or `false`	`true`

4.12.4 Registers

The ARMCortexM4CT component provides the registers specified by the technical reference manual for the Cortex-M4. Exceptions are listed in the section *Differences between the CT model and RTL implementations* on page 4-67.

4.12.5 Caches

The ARMCortexM4CT component does not implement any caches.

4.12.6 Debug features

The ARMCortexM4CT component exports a CADI debug interface.

Registers

All processor and implemented registers are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

Memory

The ARMCortexM4CT component presents one 4GB view of virtual memory.

4.12.7 Verification and testing

The ARMCortexM4CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-M4.
- booting of uClinux on an example system containing an ARMCortexM4CT component.

4.12.8 Performance

The ARMCortexM4CT component provides high performance in all areas except with instructions in protection regions smaller than 1KB and FP instruction set execution which does not use code translation technology.

4.12.9 Library dependencies

The ARMCortexM4CT component has no dependencies on external libraries.

4.12.10 Differences between the CT model and RTL implementations

The ARMCortexM4CT component differs from the corresponding revision of the ARM Cortex-M4 RTL implementation in the following ways:

- The *Wakeup Interrupt Controller* (WIC) is not currently implemented.
- Power control is not implemented. Powering down of the processor is not supported. The processor must still be clocked even if it has asserted the sleeping or sleepdeep signals.
- Only the minimal level of debug support is provided (no DAP, FPB, DWT or halting debug capability).
- Debug-related components are not implemented. Processor debug registers and system debug registers are not implemented.
- Debug interface port registers are not implemented.
- TPIU registers are not implemented.
- ETM registers are not implemented.
- No trace support (no ETM, ITM, TPUI or HTM).
- There is no supported equivalent of the RESET_ALL_REGS configuration setting in RTL (that forces all registers to have a well defined value on reset).
- Disabling processor features using the Auxiliary Control Register is not supported.

- Only a single `pvbus_m` master port is provided. This combines the ICode, DCode and System bus interfaces of the RTL. The external PPB bus is provided by the `pv_ppbus_m` master port.
- In privileged mode, STRT and LDRT to the PPB region are not forbidden access.
- The RTL implements the ROM table as an external component on the External Private Peripheral Bus. In the CT model the ROM table is implemented internally as a fallback if an external PPB access in the ROM table address region aborts. This permits the default ROM table to be overridden (by implementing an external component connected to the external PPB to handle accesses to these addresses) without requiring every user of the processor to implement and connect a ROM table component.

4.13 ARMCortexM3CT

Figure 4-12 shows a view of the ARMCortexM3CT component in System Canvas. The component is based on r2p0 of the Cortex-M3 processor.

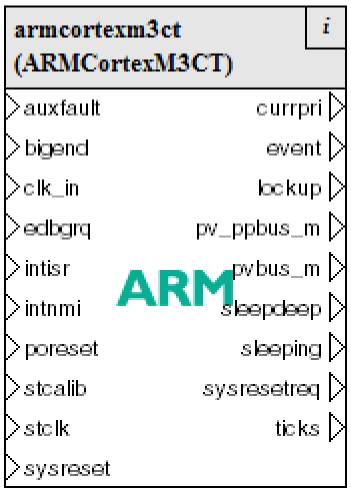


Figure 4-12 ARMCortexM3CT in System Canvas

This component is written in C++.

4.13.1 Ports

Table 4-31 provides a brief description of the ports in the ARMCortexM3CT component. For more information, see the processor technical reference manual.

Table 4-31 ARMCortexM3CT ports

Name	Port protocol	Type	Description
`auxfault`	Value	slave	auxiliary fault status information
`bigend`	Signal	slave	configure data endianness after a reset
`clk_in`	ClockSignal	slave	clock input
`currpri`	Value	master	indicates the current execution priority of the processor
`edbgrq`	Signal	slave	external debug request
`event`	Signal	peer	event input and output for wakeup from WFE. This port combines the TXEV and RXEVsignals
`intisr[0-239]`	Signal	slave	external interrupt signals
`intnmi`	Signal	slave	nonmaskable interrupt
`lockup`	Signal	master	asserted when processor is in lockup state
`poreset`	Signal	slave	asynchronous power-on reset signal input
`pvbus_m`	PVBus	master	master port for all memory accesses except those on the on the External Private Peripheral Bus

Table 4-31 ARMCortexM3CT ports (continued)

Name	Port protocol	Type	Description
`pv_ppbus_m`	PVBus	master	master port for memory accesses on the External Private Peripheral Bus
`sleepdeep`	Signal	master	indicates that the processor is in deep sleep
`sleeping`	Signal	master	indicates that the processor is in sleep
`stcalib`	Value	slave	SysTick calibration value
`stclk`	ClockSignal	slave	reference clock input for SysTick
`sysreset`	Signal	slave	asynchronous reset signal input
`sysresetreq`	Signal	master	system reset request
`ticks`	InstructionCount	master	output that can be connected to a visualization component

4.13.2 Additional protocols

The ARMCortexM3CT component has has two additional protocols. See *Additional protocols* on page 4-109.

4.13.3 Parameters

Table 4-32 provides a description of the configuration parameters for the ARMCortexM3CT component.

Table 4-32 ARMCortexM3CT parameters

Parameter	Description	Type	Allowed value	Default value
`BIGENDINIT`	Initialize processor to big endian mode	Boolean	true/false	false
`LVL_WIDTH`	Number of bits of interrupt priority.	Integer	3-8	3
`NUM_IRQ`	Number of user interrupts.	Integer	1-240	16
`NUM_MPU_REGION`	Number of MPU regions.	Integer	0, 8	8
`master_id`	master ID presented in bus transactions	Integer	`0x0000 - 0xFFFF`	`0x0`
`min_sync_level`	Controls the minimum syncLevel.	Integer	0-3	0
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	8-bit integer	`0xAB`
`semihosting-cmd_line`[a]	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]

Table 4-32 ARMCortexM3CT parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true` or `false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10700000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10700000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10800000`

a. The value of argv[0] points to the first command line argument, not to the name of an image.

4.13.4 Registers

The ARMCortexM3CT component provides the registers specified by the technical reference manual for the Cortex-M3. Exceptions are listed in the section *Differences between the CT model and RTL implementations* on page 4-72.

4.13.5 Caches

The ARMCortexM3CT component does not implement any caches.

4.13.6 Debug features

The ARMCortexM3CT component exports a CADI debug interface.

Registers

All processor and implemented registers are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

Memory

The ARMCortexM3CT component presents one 4GB view of virtual memory.

4.13.7 Verification and testing

The ARMCortexM3CT component has been tested using:

- the architecture validation suite tests for the ARM Cortex-M3
- booting of uClinux and RTX on an example system containing an ARMCortexM3CT component.

4.13.8 Performance

The ARMCortexM3CT component provides high performance in all areas except with instructions in protection regions smaller than 1KB.

4.13.9 Library dependencies

The ARMCortexM3CT component has no dependencies on external libraries.

4.13.10 Differences between the CT model and RTL implementations

The ARMCortexM3CT component differs from the corresponding revision of the ARM Cortex-M3 RTL implementation in the following ways:

- The WIC is not currently implemented.
- Power control is not implemented, so the processor does not set the SLEEPING or SLEEPDEEP signals. It does not support powering down of the processor.
- Only the minimal level of debug support is provided (no DAP, FPB, DWT or halting debug capability).
- Debug-related components are not implemented. Processor debug registers and system debug registers are not implemented.
- Debug interface port registers are not implemented.
- TPIU registers are not implemented.
- ETM registers are not implemented.
- The processor must still be clocked even if it has asserted the sleeping or sleepdeep signals.
- Disabling processor features using the Auxiliary Control Register is not supported.
- Only a single `pvbus_m` master port is provided. This combines the ICode, DCode and System bus interfaces of the RTL. The external PPB bus is provided by the `pv_ppbus_m` master port.
- In privileged mode, STRT and LDRT to the PPB region are not forbidden access.
- No trace support (no ETM, ITM, TPUI or HTM).
- There is no supported equivalent of the RESET_ALL_REGS configuration setting in RTL (that forces all registers to have a well defined value on reset).
- The RTL implements the ROM table as an external component on the External Private Peripheral Bus. In the CT model the ROM table is implemented internally as a fallback if an external PPB access in the ROM table address region aborts. This permits the default

ROM table to be overridden (by implementing an external component connected to the external PPB to handle accesses to these addresses) without requiring every user of the processor to implement and connect a ROM table component.

4.14 ARMv7-A AEM

The ARM *Architecture Envelope Model* (AEM) is a highly-configurable simulation model of a processor compliant to the ARMv7-A architecture supporting multiprocessor, large physical address and virtualization extensions. For information on characteristics and capabilities specific to the AEM, see Appendix A *AEM ARMv7-A specifics*.

Figure 4-13 shows a view of the ARMAEMv7AMPCT model in the Fast Models Portfolio, with all vectored ports collapsed.

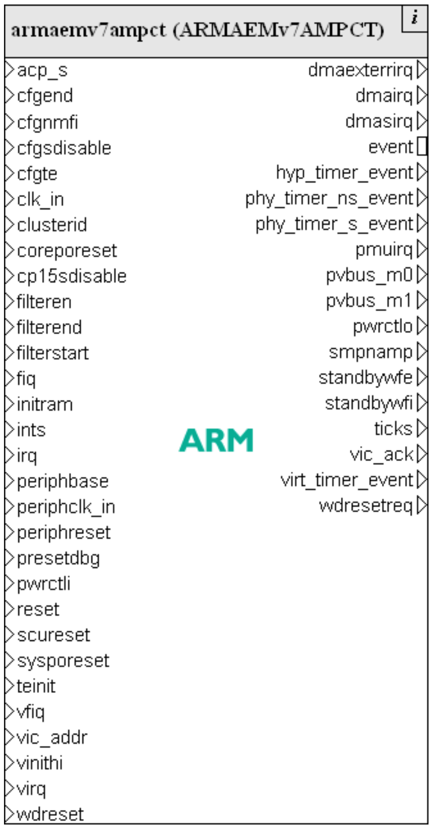


Figure 4-13 ARMAEMv7AMPCT in System Canvas

This model is written in C++.

4.14.1 Ports

Table 4-33 provides a brief description of the ports in the ARMAEMv7AMPCT model.

Table 4-33 ARMAEMv7AMPCT ports

Name	Port Protocol	Type	Description
`acp_s`	PVBus	slave	AXI ACP slave port
`cfgend[0-3]`	Signal	slave	Initialize to BE8 endianness after a reset.
`cfgnmfi[0-3]`	Signal	slave	Disables FIQ mask in *Current Program Status Register* (CPSR).

Table 4-33 ARMAEMv7AMPCT ports (continued)

Name	Port Protocol	Type	Description
`cfgsdisable`	Signal	slave	Disable write access to some GIC registers.
`cfgte`	Signal	slave	Initializes to take exceptions in T32 state after a reset.
`clk_in`	ClockSignal	slave	Main processor clock input.
`clusterid`	Value	slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
`coreporeset[0-3]`	Signal	slave	Individual processor reset signal.
`cp15sdisable[0-3]`	Signal	slave	Disable write access to some secure cp15 registers.
`dmairq`	Signal	master	Interrupt signal from L1 DMA.
`dmasirq`	Signal	master	Secure interrupt signal from L1 DMA.
`event`	Signal	peer	Event input and output for wakeup from WFE. This port amalgamates the EVENTI and EVENT0 signals that are present on hardware.
`filteren`	Signal	slave	Enables filtering of address ranges between master bus ports.
`filterend`	Value	slave	Sets end of region mapped to pvbus_m1.
`filterstart`	Value	slave	Sets start of region mapped to pvbus_m1
`fiq[0-3]`	Signal	slave	Processor FIQ signal input.
`hyp_timer_event[0-3]`	Signal	master	Interface to SoC level counter module.
`initram`	Signal	slave	Initializes with ITCM enabled after reset.
`ints[0-223]`	Signal	slave	Drives the interrupt port of the GIC.
`irq[0-3]`	Signal	slave	Processor IRQ signal input.
`periphbase`	Value	slave	Base of private peripheral region.
`periphclk_in`	ClockSignal	slave	Timer/watchdog clock rate.
`periphreset`	Signal	slave	Resets the timer and interrupt controller.
`phy_timer_ns_event[0-3]`	Signal	master	Interface to SoC level counter module.
`phy_timer_s_event[0-3]`	Signal	master	Interface to SoC level counter module.
`pmuirq[0-3]`	Signal	master	*Performance Monitoring Unit* (PMU) interrupt signal.
`presetdbg[0-3]`	Signal	slave	Individual processor reset signal.
`pvbus_m0`	PVBus	master	AXI master 0 bus master channel.
`pvbus_m1`	PVBus	master	AXI master 1 bus master channel.

Table 4-33 ARMAEMv7AMPCT ports (continued)

Name	Port Protocol	Type	Description
`pwrctli[0-3]`	Value	slave	Resets the reset value for SCU processor status register.
`pwrctlo[0-3]`	Value	master	Sends SCU processor status register bits.
`reset[0-3]`	Signal	slave	Individual processor reset signal.
`scureset`	Signal	slave	Resets SCU.
`smpnamp[0-3]`	Signal	master	Indicates AMP or SMP mode for each processor.
`standbywfe[0-3]`	Signal	master	Indicates if a processor is in WFE state.
`standbywfi[0-3]`	Signal	master	Indicates if a processor is in WFI state.
`sysporeset`	Signal	slave	Individual processor reset signal.
`teinit[0-3]`	Signal	slave	Initialize to take exceptions in T32 state after a reset.
`ticks[0-3]`	InstructionCount	master	Processor instruction count for visualization.
`vfiq[0-3]`	Signal	slave	Processor virtual FIQ signal input.
`vic_ack`	Signal	master	Acknowledge port to primary VIC.
`vic_addr`	ValueState	slave	Address port from primary VIC.
`vinithi[0-3]`	Signal	slave	Initialize with high vectors enabled after a reset.
`virq[0-3]`	Signal	slave	Processor virtual IRQ signal input.
`virt_timer_event[0-3]`	Signal	master	Interface to SoC level counter module.
`wdreset[0-3]`	Signal	slave	Resets individual watchdog.
`wdresetreq[0-3]`	Signal	master	Resets rest of the Cortex-A9 MP system.

4.14.2 Additional protocols

The ARMAEMv7AMPCT model has two additional protocols. See *Additional protocols* on page 4-109.

4.14.3 Parameters

In the platform models, configuration parameters are located in the parameter hierarchy at `cluster.parameter_name`. There are a number of other parameters that adjust the behavior of external platform components on the *Versatile*™ *Express* (VE) system board. These are:

- *Multiprocessor configuration* on page 4-77
- *General processor configuration* on page 4-77
- *Memory configuration* on page 4-78
- *Cache geometry configuration* on page 4-80
- *Debug architecture configuration* on page 4-82
- *Processor configuration* on page 4-82

- *Semihosting configuration* on page 4-83
- *Message configuration* on page 4-84.

Multiprocessor configuration

You can configure this model as a multiprocessor, so there are separate groups of configuration parameters for each processor in the system. In cases where fewer processors than the maximum number possible are instantiated, the parameters from `cpu0` are always used first. See Table 4-34.

Table 4-34 Multiprocessing parameters

Parameter	Description	Default
`cluster_id`	Value for Cluster ID that is available to target programs in MPIDR.	0
`multiprocessor_extensions`	Enable the instruction set changes introduced with the ARMv7 Multiprocessor Extensions.	`true`
`num_cores`	Number of processors implemented. To instantiate more than one processor, set parameter `multiprocessor_extensions`.	1
`vmsa.cachetlb_broadcast`	Enable broadcasting of cache and TLB maintenance operations that apply to the inner shared domain.	`true`
`cpu[n].SMPnAMP`	Place this processor inside the inner shared domain, and participate in the coherency protocol that arranges inner cache coherency among other processors in the domain.	`false`

General processor configuration

This section describes processor configuration parameters. See Table 4-35.

Table 4-35 Processor configuration parameters

Parameter	Description	Default
`auxilliary_feature_register0`	Value for AFR0 ID register	0
`cpuID`	Value for main processor ID register	`0x411fc081`
`dic-spi_count`	Number of shared peripheral interrupts implemented.	64
`dtcm0_base`	DTCM base address at reset	0
`dtcm0_enable`	Enable DTCM at reset	`false`
`dtcm0_size`	DTCM size in KB	32
`FILTEREN`	Enable filtering of accesses between master bus ports. This is usually not used inside a VE system and should be left `false`.	`false`
`FILTEREND`	End of region filtered to `pvbus_m1`. Values must be aligned to a 1MB boundary.	0
`FILTERSTART`	Start of region filtered to `pvbus_m1`. Values must be aligned to a 1MB boundary.	0
`implements_ple_like_a8`	Add support for the PLE from Cortex-A8	`false`
`IS_VALIDATION`[a]	Reserved. Enables A9-validation-like trickbox-coprocessor, which is only usable in validation platform model.	`false`

Table 4-35 Processor configuration parameters (continued)

Parameter	Description	Default
`itcm0_base`	ITCM base address at reset	`0x40000000`
`itcm0_enable`	Enable ITCM at reset	`false`
`itcm0_size`	ITCM size in KB	32
`PERIPHBASE`[b]	Base address of MP "private" peripherals (WatchdogTimers, GIC) (bits 31:13 used).	`0x13080000`
`siliconID`	Value for Auxilliary ID register	`0x41000000`
`CFGSDISABLE`	Disables access to some registers in the internal interrupt controller peripheral.	`false`
`implements_virtualization`	Implement the Virtualization extension in this processor. When set, this also enables LPAE.	`false`
`implements_lpae`	Implement the Large Physical Address extension in this processor.	`false`
`use_Cortex-A15_peripherals`	Changes the layout of the internal peripheral memory map to mimic that of the Cortex-A15. See *Internal peripherals* on page A-10.	`false`
`delayed_CP15_operations`	Delay the functional effect of CP15 operations. See *Delayed operation of CP15 instructions* on page A-3.	`false`
`take_ccfail_undef`	Take undefined exceptions even if the instruction failed its condition codes check. See *An undefined instruction failed its condition code check* on page A-4.	`false`
`low_latency_mode`	Run only a single instruction between checks for IRQ and other events. This ensures that when the platform raises an interrupt, the exception vector is taken immediately, but it involves a considerable penalty in performance.	`false`

a. IS_VALIDATION is not exposed in the VE platform model, and fixed as `false`.

b. PERIPHBASE is not exposed in the VEplatform model, and fixed as `0x2C000000`.

Memory configuration

This section describes memory configuration parameters. See Table 4-36.

Table 4-36 Memory configuration parameters

Parameter	Description	Default
`vmsa.implements_fcse`	Support `fcse` in this processor	`false`
`vmsa.infinite_write_buffer`	Enable infinite write-buffer. See *Infinite write buffer* on page A-2.	`false`
`vmsa.write_buffer_delay`	Elapsed time between natural buffer drains. See *Infinite write buffer* on page A-2.	1000
`vmsa.delayed_read_buffer`	Enable deferred read values in conjunction with `use_IR`. See *Memory operation reordering* on page A-5.	`false`
`vmsa.cache_incoherence_check`	Enable the check for cache incoherence. See *Cache incoherence check* on page A-3.	`false`

Table 4-36 Memory configuration parameters (continued)

Parameter	Description	Default
`vmsa.memory_marking_check`	Enable the check for inconsistent memory marking in the TLB. See *Memory marking check* on page A-5.	`false`
`vmsa.instruction_tlb_lockable_entries`	Number of lockable entries in instruction TLB	32
`vmsa.instruction_tlb_size`	Total number of entries in instruction TLB	32
`vmsa.main_tlb_lockable_entries`	Number of lockable entries in data or unified TLB	32
`vmsa.main_tlb_size`	Total number of entries in data or unified TLB	32
`vmsa.separate_tlbs`	Separate ITLB and DTLB. If the TLB is unified, its size is defined by parameter `vmsa.main_tlb_size`.	`true`
`vmsa.tlb_prefetch`	Enables aggressive pre-fetching into the TLB. See *Aggressively pre-fetching TLB* on page A-2.	`false`
`vmsa.implements_outer_shareable`	Distinguish between inner shareable and outer shareable memory access types. Outer shareable is implemented as Non Cacheable.	`true`
`vmsa.access_flags_hardware_management`	Enable support for the hardware management of the Access Flag in the pagetables.	`true`
`dcache-state_modelled`	Allow line allocation in D-side caches at all levels	`true`
`icache-state_modelled`	Allow line allocation in I-side caches at all levels. Unified caches allocate lines only if these parameters are enabled at both I-side and D-side.	`true`

The `[d|i]cache-state_modelled` parameters control the way that caches are simulated. When switched on, the default mode, all cache behaviors and maintenance operations are modeled fully.

If `false`, the cache is still present in the programmer's view of the processor but in the simulated implementation there are no memory lines associated with the cache at this level. The programmer-view effect of this is as though the cache cleans and invalidates any line as soon as it is loaded, and can never become incoherent with its backing memory. Although this is an architecturally legal behavior, it is not realistic to any current hardware and is less likely to expose problems in target software. It can, however, be useful when debugging problems that are suspected to be related to cache maintenance, and also has the side effect of permitting the model to run faster.

Compare this to the effect of setting `cpu[n].l2dcache-size_bytes = 0`, which is to simulate a processor that contains only Level 1 caches. In this case, the ID code registers do not describe a Level 2 cache. Level 2 is entirely absent from the processor.

Cache geometry configuration

You can configure the multiprocessor with up to four levels of cache. The cache layout is not required to be symmetrical for each processor in the multiprocessor, so the parameters listed in Table 4-37 are repeated in groups `cpu0-cpu3` corresponding to the view for each processor of the memory hierarchy.

Table 4-37 General cache configuration parameters

Parameter	Description	Default
`cpu[n].cache-coherency_level`	1-based-Level of cache coherency. A value of 2 means that the L2 caches, and all subsequent levels, are coherent.	2
`cpu[n].cache-unification_level`	1-based-Level of cache unification. A value of 2 means that the L2 caches, and all subsequent levels, are unified.	2
`cpu[n].cache-outer_level`	Level at which outer cache attributes start to be used. L1 caches always uses inner attributes. A value of 2 means that the L2 caches, and all subsequent levels, use outer attributes.	2

Each cache block in the system is configured using the parameters listed in Table 4-38, which are repeated for groups `cpu0-cpu3`, and within each group in caches `l1icache`, `l1dcache-l4icache`, `l4dcache`.

The number and type of cache blocks are active depending on the unification level of each processor. Before the unification level, caches are separate on the instruction and data sides, and both sets of parameters are used. After the unification level, the data and instruction sides are unified, and the single cache block is described using the data side parameters only.

Table 4-38 Cache block configuration parameters

Parameter	Description	Default
`cpu[n].[cache]-size_bytes`	Zero if the cache is not present, otherwise the total size in bytes of the cache. Must be divisible by the line length and associativity, and represent a number of cache sets not greater than 32768.	32768
`cpu[n].[cache]-linelength_bytes`	Length of each cache line. Must be 32 or 64.	32
`cpu[n].[cache]-associativity`	Associativity of this cache. Must be between 1 and 1024.	4
`cpu[n].[cache]-read_allocate`	Support allocate-on-read in this cache	`true`
`cpu[n].[cache]-write_allocate`[a]	Support allocate-on-write in this cache	`true`
`cpu[n].[cache]-write_back`[a]	Support write-back in this cache	`true`
`cpu[n].[cache]-write_through`[a]	Support write-through in this cache	`true`
`cpu[n].[cache]-treat_invalidate_as_clean`[a]	Always clean dirty cache lines before invalidating them. See *Other checks* on page A-6.	`false`
`cpu[n].[cache]-shared_key`	If nonzero, mark this cache as being shared with other processors.	0

a. This parameter is not applicable to instruction-side caches.

The parameters for each processor describe the view for that processor of the memory hierarchy. If more than one processor has access to the same cache unit, for example, a shared Level 2 cache, then:

- the cache must be described with all the same parameter settings in every case
- all caches downstream of a shared cache must also be shared, and in the same order for every observer
- the `[cache]-shared_key` parameter is set to an arbitrary nonzero value. Any cache in the system that has this value is considered to be one cache block.

You can describe nonlegal cache layouts using the `shared_key` mechanism. Not all bad cases can be easily detected during initialization, so take care to ensure correct cache configuration. The model might behave erratically if the cache layout cannot be rationalized.

See Figure 4-14 and Figure 4-15 for examples of processor-cache architecture configurations.

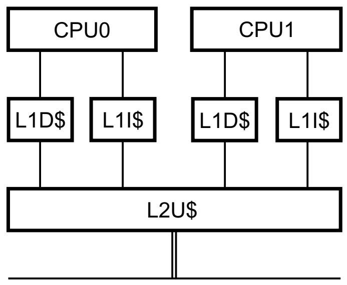


Figure 4-14 Processor-cache architecture configuration - Example 1

```
cpu0.cache-unification_level=2
cpu0.l2dcache-size_bytes=32768
cpu0.l2dcache-shared_key=1
cpu1.cache-unification_level=2
cpu1.l2dcache-size_bytes=32768
cpu1.l2dcache-shared_key=1
```

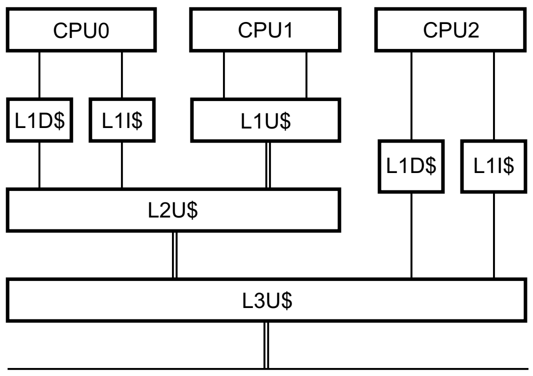


Figure 4-15 Processor-cache architecture configuration - Example 2

```
cpu0.cache-unification_level=2
cpu0.l2dcache-size_bytes=32768
cpu0.l2dcache-shared_key=1
cpu0.l3dcache-size_bytes=65536
cpu0.l3dcache-shared_key=2
cpu1.cache-unification_level=1
cpu1.l2dcache-size_bytes=32768
cpu1.l2dcache-shared_key=1
cpu1.l3dcache-size_bytes=65536
cpu1.l3dcache-shared_key=2
```

```
cpu2.cache-unification_level=2
cpu2.l2dcache-size_bytes=65536
cpu2.l2dcache-shared_key=2
```

Note

In the view of CPU2, the shared cache block marked L3U$ is at Level 2 in the memory system hierarchy.

Debug architecture configuration

The ARMv7 Debug architecture contains a number of optional features. The parameters listed in Table 4-39 control which of these features are implemented by the model.

Table 4-39 Debug architecture configuration parameters

Parameter	Description	Default
`implements_OSSaveAndRestore`	Add support for the OS Save and Restore mechanism implemented by DBGOSSRR and other registers.	`true`
`DBGOSLOCKINIT`	Initial value for the Locked bit in DBGOSLSR. When this bit is set, software access to debug registers is restricted.	`0x1`
`implements_secure_user_halting_debug`	Permit debug events in Secure User mode when invasive debug is not permitted in Secure privileged modes. (Deprecated in ARM v7.)	`false`
`DBGPID`	Value for CP14 DBGPID registers	`0x8000bb000`
`DBGCID`	Value for CP14 DBGCID registers	`0x0`
`DBGDSCCR_mask`	Implemented bits of DBGDSCCR	`0x7`
`cpu[n].DBGDRAR`	Value for Debug ROM address register	`0x0`
`cpu[n].DBGSRAR`	Value for Debug Self address register	`0x0`

Processor configuration

These parameters are repeated in groups `cpu0`-`cpu3` for each processor in the multiprocessor. See Table 4-40.

Table 4-40 Processor configuration parameters

Parameter	Description	Default
`cpu[n].CFGEND0`	Starts the processor in big endian BE8 mode	`false`
`cpu[n].CFGNMFI`	Sets the NMFI bit in the *System Control Register* (SCTLR) that prevents the FIQ interrupt from being masked in APSR.	`false`
`cpu[n].CFGTE`	Starts the processor in T32 mode	`false`
`cpu[n].CP15SDISABLE`	Disables access to some CP15 registers	`false`
`cpu[n].VINITHI`	Starts the processor with high vectors enabled, the vector base address is `0xFFFF0000`	`false`
`cpu[n].implements_neon`	Support NEON in this processor	`true`

Table 4-40 Processor configuration parameters (continued)

Parameter	Description	Default
`cpu[n].implements_thumbEE`	Support T32EE in this processor	`true`
`cpu[n].implements_trustzone`	Support TrustZone in this processor	`true`
`cpu[n].implements_vfp`	Support VFP in this processor	`true`
`cpu[n].fpsID`	Value for Floating-point System ID Register	`0x41033091`
`cpu[n].implements_vfpd16-d31`	If VFP is implemented, support 32 double-precision registers. Otherwise 16 are supported. If NEON is implemented, 32 registers are always supported and this parameter is ignored.	`true`
`cpu[n].implements_vfp_short_vectors`	Enable support for vfp short vector operations, as indicated by `MVFR0[27:24]`	`true`
`cpu[n].implements_fused_mac`	Implement the vfp fused multiply accumulate operations	`false`
`cpu[n].implements_sdiv_udiv`	Implement the integer divide operations	`false`
`cpu[n].vfp-enable_at_reset`	VFP registers are enabled without a requirement to write the corresponding access enable bits first	`false`
`cpu[n].use_IR`	Enable operation reordering in conjunction with `delayed_read_buffer`. See *Memory operation reordering* on page A-5.	0

Semihosting configuration

Semihosting is a method of target software running on the model to communicate with the host environment. This model permits the target C library to access I/O facilities of the host computer; file system, keyboard input, clock and so on. For more information see the *RealView Compilation Tools Developer Guide*.

These parameters are repeated in groups `cpu0-cpu3` for each processor in the multiprocessor. See Table 4-41.

Table 4-41 Processor configuration parameters

Parameter	Description	Default
`cpu[n].semihosting-ARM_SVC`	A32 SVC number to be treated as a semihosted call	`0x123456`
`cpu[n].semihosting-Thumb_SVC`	T32 SVC number to be treated as a semihosted call	`0xab`
`cpu[n].semihosting-cmd_line`	Program name and arguments to be passed as `argc`, `argv` to target programs using the semihosted c library.	
`cpu[n].semihosting-enable`	Enable semihosting of SVC instructions	`true`
`cpu[n].semihosting-heap_base`	Virtual address of heap base	`0x00000000`
`cpu[n].semihosting-heap_limit`	Virtual address of top of heap	`0x0f000000`
`cpu[n].semihosting-stack_base`	Virtual address of base of descending stack	`0x10000000`
`cpu[n].semihosting-stack_limit`	Virtual address of stack limit	`0x0f000000`

Message configuration

The parameters listed in Table 4-42 control how warning and error messages from the architectural checkers are generated.

Table 4-42 Message severity levels

Parameter	Description	Default
`messages.break_warning_level`	The simulation stops in the debugger after emitting a message at this level or higher.	5
`messages.ignore_warning_level`	Messages below this level are ignored and not printed.	1
`messages.suppress_repeated_messages`	The simulation does not emit more than one copy of a message when it is generated from a given point in the target program.	`true`
`messages.output_file`	The path[a] of the file to which messages are written. If blank, messages are sent to `stderr`.	

a. The format of the string follows the normal file path conventions for the host platform. File paths without a leading root are written into the current working directory, which might vary.

Except for fatal errors, the severity level of each message can be reconfigured in parameters `messages.severity_level_[*]`, permitting you to concentrate only on those warnings that are appropriate to your task. See Table 4-43.

Table 4-43 Message configuration parameters

Level	Name	Description
0	Minor Warning	Suspect, but plausibly correct
1	Warning	A likely bug
2	Severe Warning	Technically legal, but believed certain to be a bug
3	Error	A definite architectural violation
4	Severe Error	Target code unlikely to be able to recover
5	Fatal	From which the simulation is unable to continue

4.14.4 Registers

The ARMAEMv7AMPCT model provides registers with the following exceptions:

- integration and test registers are not implemented.

4.14.5 Caches

The ARMAEMv7AMPCT model implements L1 and L2 cache as architecturally defined.

4.14.6 Debug Features

The ARMAEMv7AMPCT model exports a CADI debug interface.

Registers

All processor, VFP, CP14 and CP15 registers, apart from performance counter registers, are visible in the debugger.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The Secure and Normal views show the contents of memory that are read by the processor when it performs a data-side memory access in the secure and non-secure modes in TrustZone. The Level 1 cache is examined first, and in the case of a cache miss the Level 2 cache, if present, is examined. This continues until external memory or peripherals are examined. This view is indexed by modified virtual address.

The L1-DCache, L1-DCacheNS, L2-DCache, ... views show the contents of an individual cache block. If there is a cache miss at this level, no data is shown. These views cause no side-effects in the simulation. This view is indexed by physical address.

The External and ExternalNS views show the contents of memory available from the external bus interface of the processor. Memory is shown only if it can be read without causing side effects. In most cases, the contents of the RAM are available but peripheral registers are not. Availability of non-idempotent memory such as flash units might depend on the state of the flash unit and code that has already been executed in the model. This view is indexed by physical address.

TLB

Each active processor in the multiprocessor exports a CADI interface that describes the current contents of its TLB.

Note

This view is subject to change in future versions.

You can examine TLB entries by opening a Disassembly view in the debugger window for the TLB CADI interface. You can select two display modes from the Memory Space menus:

- *Entries by index*
- *Entries by MVA* on page 4-86.

Entries by index

The Address field is an index into a linear, ordered list of TLB entries.

All entries in the TLB are shown, regardless of security state or ASID. Entries outside the current state, or that do not match the current ASID, are not be used by the processor for physical address translations.

This view is available only for TLBs with a limited, finite size. It is not supported when the `tlb_prefetch` option is enabled, because there is no linear indexing of TLB entries in the infinite pre-fetching TLB model.

Entries by MVA

The Address field corresponds to the *Modified Virtual Address* (MVA) from which a TLB entry is translated.

This view is specific to a particular security state and ASID. You can control this by opening a Register view in the debugger window. You can change the registers NS and ASID to view the corresponding TLB entries.

4.14.7 Verification and Testing

The ARMAEMv7AMPCT has been tested using the ARM *Architecture Verification Suite* (AVS).

4.14.8 Performance

The ARMAEMv7AMPCT model provides high performance in all areas except VFP and NEON instruction set execution which currently does not use code translation technology.

4.14.9 Library dependencies

The ARMAEMv7AMPCT model has no dependencies on external libraries.

4.14.10 Boundary features and architectural checkers

Boundary features and architectural checkers are model capabilities that help your development and testing process by exposing latent problems in the target code. For more information on these capabilities, see Appendix A *AEM ARMv7-A specifics*.

4.15 ARM1176CT

Figure 4-16 shows a view of the ARM1176CT component in System Canvas. This component is based on r0p4 of the ARM1176JZF-S™ processor.

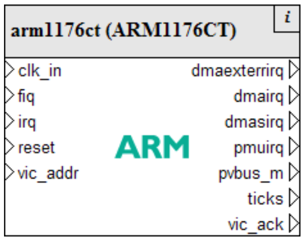


Figure 4-16 ARM1176CT in System Canvas

This component is written in C++.

4.15.1 Ports

Table 4-44 provides a brief description of the ports for the ARM1176CT component. For more information, see the processor technical reference manual.

Table 4-44 ARM1176CT ports

Name	Port protocol	Type	Description
`clk_in`	ClockSignal	slave	clock input
`pvbus_m`	PVBus	master	master port for all memory accesses
`reset`	Signal	slave	asynchronous reset signal input
`irq`	Signal	slave	asynchronous IRQ signal input
`fiq`	Signal	slave	asynchronous FIQ signal input
`pmuirq`	Signal	master	performance monitoring unit IRQ output
`dmairq`	Signal	master	normal DMA interrupt output
`dmasirq`	Signal	master	secure DMA interrupt output
`dmaexterrirq`	Signal	master	DMA error interrupt output
`vic_addr`	ValueState	slave	address input for connection to PL192 VIC
`vic_ack`	Signal	master	acknowledge signal output for PL192 VIC
`ticks`	InstructionCount	master	output that can be connected to a visualization component

4.15.2 Additional protocols

The ARM1176CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.15.3 Parameters

Table 4-45 provides a description of the configuration parameters for the ARM1176CT component.

Table 4-45 ARM1176CT parameters

Parameter	Description	Type	Values	Default
`BIGENDINIT`	initialize to ARMv5 big endian mode	Boolean	true/false	false
`CP15SDISABLE`	initialize to disable access to some CP15 registers	Boolean	true/false	false
`INITRAM`	initialize with ITCM0 enabled at address `0x0`	Boolean	true/false	false
`UBITINIT`	initialize to ARMv6 unaligned behavior	Boolean	true/false	false
`VINITHI`	initialize with high vectors enabled	Boolean	true/false	false
`itcm0_size`	Size of ITCM in KB.	Integer	`0x00 - 0x40`	`0x10`
`dtcm0_size`	Size of DTCM in KB.	Integer	`0x00 - 0x40`	`0x10`
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	true/false	false[a]
`semihosting-cmd_line`[b]	command line available to semihosting SVC calls	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	true/false	true
`semihosting-ARM_SVC`	A32 SVC number for semihosting	Integer	uint24_t	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting	Integer	uint8_t	`0xAB`
`semihosting-heap_base`	virtual address of heap base	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	virtual address of top of heap	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	virtual address of base of descending stack	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	virtual address of stack limit	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F0000000`
`vfp-enable_at_reset`[c]	enable coprocessor access and VFP at reset	Boolean	true/false	false
`vfp-present`	configure processor as VFP enabled[d]	Boolean	true/false	true

a. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

b. The value of argv[0] points to the first command line argument, not to the name of an image.

c. This is model specific behavior with no hardware equivalent.

d. This parameter lets you disable the VFP features of the model. However the model has not been validated as a true ARM1176JZ-S processor.

4.15.4 Registers

The ARM1176CT component provides the registers specified by the technical reference manual for the ARM1176JZF-S with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.15.5 Debug features

The ARM1176CT component exports a CADI debug interface.

Registers

All processor, VFP, and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single address unconditional instruction breakpoints
- unconditional instruction address range breakpoints
- single address unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARM1176CT component presents two 4GB views of virtual memory, one as seen from secure mode and one as seen from normal mode.

4.15.6 Verification and testing

The ARM1176CT component has been tested using:

- the architecture validation suite tests for the ARM1176JZF-S
- booting of Linux on an example system containing an ARM1176CT component.

4.15.7 Performance

The ARM1176CT component provides high performance in all areas except VFP instruction set execution which currently does not use code translation technology.

4.15.8 Library dependencies

The ARM1176CT component has no dependencies on external libraries.

4.15.9 Differences between the CT model and RTL implementations

The ARM1176PCT component differs from the corresponding revision of the ARM 1176 RTL implementation in the following ways:

- There is a single memory port combining instruction, data, and peripheral access.

4.16 ARM1136CT

Figure 4-17 shows a view of the ARM1136CT component in System Canvas. This component is based on r1p1 of the ARM1136JF-S™ processor.

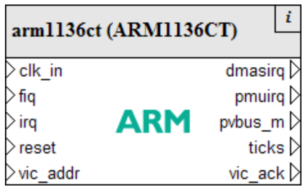


Figure 4-17 ARM1136CT in System Canvas

This component is written in C++.

4.16.1 Ports

Table 4-46 provides a brief description of the ports of the ARM1136CT component. For more information, see the processor technical reference manual.

Table 4-46 ARM1136CT ports

Name	Port protocol	Type	Description
`clk_in`	ClockSignal	slave	clock input
`pvbus_m`	PVBus	master	master port for all memory accesses
`reset`	Signal	slave	asynchronous reset signal input
`irq`	Signal	slave	asynchronous IRQ signal input
`fiq`	Signal	slave	asynchronous FIQ signal input
`pmuirq`	Signal	master	performance monitoring unit IRQ output
`dmasirq`[a]	Signal	master	normal DMA interrupt output
`vic_addr`	ValueState	slave	address input for connection to PL192 VIC
`vic_ack`	Signal	master	acknowledge signal output for PL192 VIC
`ticks`	InstructionCount	master	output that can be connected to a visualization component

a. This signal is currently misnamed and is to be named dmairq.

4.16.2 Additional protocols

The ARM1136CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.16.3 Parameters

Table 4-47 provides a description of the configuration parameters for the ARM1136CT component.

Table 4-47 ARM1136CT parameters

Parameter	Description	Type	Values	Default
`BIGENDINIT`	initialize to ARMv5 big endian mode	Boolean	true/false	false
`INITRAM`	initialize with ITCM0 enabled at address `0x0`	Boolean	true/false	false
`UBITINIT`	initialize to ARMv6 unaligned behavior	Boolean	true/false	false
`VINITHI`	initialize with high vectors enabled	Boolean	true/false	false
`itcm0_size`	Size of ITCM in KB.	Integer	`0x00 - 0x40`	`0x10`
`dtcm0_size`	Size of DTCM in KB.	Integer	`0x00 - 0x40`	`0x10`
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	true/false	false[a]
`semihosting-cmd_line`[b]	command line available to semihosting SVC calls	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	true/false	true
`semihosting-ARM_SVC`	A32 SVC number for semihosting	Integer	uint24_t	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting	Integer	uint8_t	`0xAB`
`semihosting-heap_base`	virtual address of heap base	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	virtual address of top of heap	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	virtual address of base of descending stack	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	virtual address of stack limit	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F0000000`
`vfp-enable_at_reset`[c]	enable coprocessor access and VFP at reset	Boolean	true/false	false
`vfp-present`	configure processor as VFP enabled[d]	Boolean	true/false	true

a. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

b. The value of argv[0] points to the first command line argument, not to the name of an image.

c. This is model specific behavior with no hardware equivalent.

d. This parameter lets you disable the VFP features of the model. However the model has not been validated as a true ARM1136J-S processor.

4.16.4 Registers

The ARM1136CT component provides the registers specified by the technical reference manual for the ARM1136JF-S with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.16.5 Debug features

The ARM1136CT component exports a CADI debug interface.

Registers

All processor, VFP and CP15 registers, apart from performance counter registers, are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single unconditional instruction breakpoints
- unconditional instruction range breakpoints
- single unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARM1136CT component presents a single flat 4GB view of virtual memory as seen by the model processor.

4.16.6 Verification and testing

The ARM1136CT component has been tested using:

- the architecture validation suite tests for the ARM1136JZF-S
- booting of Linux and other operating systems on an example system containing an ARM1136CT component.

4.16.7 Performance

The ARM1136CT component provides high performance in all areas except VFP instruction set execution which currently does not use code translation technology.

4.16.8 Library dependencies

The ARM1136CT component has no dependencies on external libraries.

4.16.9 Differences between the CT model and RTL implementations

The ARM1136CT component differs from the corresponding revision of the ARM 1136 RTL implementation in the following ways:

- There is a single memory port combining instruction, data, and peripheral access.

4.17 ARM968CT

Figure 4-18 shows a view of the ARM968CT component in System Canvas. This component is based on r0p1 of the ARM968E-S™ processor.

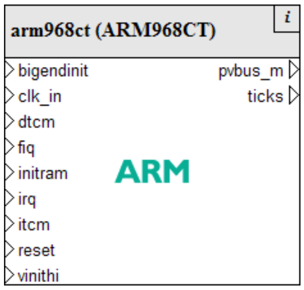


Figure 4-18 ARM968CT in System Canvas

This component is written in C++.

4.17.1 Ports

Table 4-48 provides a brief description of the ports of the ARM968CT component. For more information, see the processor technical reference manual.

Table 4-48 ARM968CT ports

Name	Port protocol	Type	Description
`clk_in`	ClockSignal	slave	clock input
`pvbus_m`	PVBus	master	master port for all memory accesses
`reset`	Signal	slave	asynchronous reset signal input
`irq`	Signal	slave	asynchronous IRQ signal input
`fiq`	Signal	slave	asynchronous FIQ signal input
`ticks`	InstructionCount	master	output that can be connected to a visualization component
`vinithi`	Signal	slave	initialize with high vectors enabled after a reset
`initram`	Signal	slave	initialize with ITCM enabled after reset
`itcm`	PVBus	slave	slave access to ITCM
`dtcm`	PVBus	slave	slave access to DTCM
`bigendinit`	Signal	slave	enable BE32 endianness after reset

4.17.2 Additional protocols

The ARM968CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.17.3 Parameters

Table 4-49 provides a description of the configuration parameters for the ARM968CT component.

Table 4-49 ARM968CT parameters

Parameter	Description	Type	Values	Default
`BIGENDINIT`	initialize to ARMv5 big endian mode	Boolean	true/false	false
`INITRAM`	initialize with ITCM0 enabled at address `0x0`	Boolean	true/false	false
`VINITHI`	initialize with high vectors enabled	Boolean	true/false	false
`dtcm0_size`	size of DTCM in KB, 0 disables	Integer	`0x0000 - 0x1000`	`0x8`
`itcm0_size`	size of ITCM in KB, 0 disables	Integer	`0x0000 - 0x1000`	`0x8`
`master_id`	master ID presented in bus transactions	Integer	`0x0000 - 0xFFFF`	`0x0`
`semihosting-ARM_SVC`	A32 SVC number for semihosting	Integer	uint24_t	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting	Integer	uint8_t	`0xAB`
`semihosting-cmd_line`[a]	command line available to semihosting SVC calls	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	true/false	true
`semihosting-heap_base`	virtual address of heap base	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	virtual address of top of heap	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	virtual address of base of descending stack	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	virtual address of stack limit	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F0000000`

a. The value of argv[0] points to the first command line argument, not to the name of an image.

4.17.4 Registers

The ARM968CT component provides the registers specified by the technical reference manual for the ARM968E-S with the following exceptions:

- coprocessor 14 registers are not implemented

- integration and test registers are not implemented.

4.17.5 Debug features

The ARM968CT component exports a CADI debug interface.

Registers

All processor, CP15 registers are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single unconditional instruction breakpoints
- unconditional instruction range breakpoints
- single unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints. The current models do not support processor exception breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARM968CT component presents a single flat 4GB view of virtual memory as seen by the model processor.

4.17.6 Verification and testing

The ARM968CT component has been tested using:

- the architecture validation suite tests for the ARM968E-S
- booting of uClinux and ThreadX OS images on a VE ARM968 simulation platform.

4.17.7 Performance

The ARM968CT component provides high performance in all areas except when protection regions are configured with regions or subregions less than 1KB in size. Any execution of instructions within the aligned 1KB of memory containing that region runs slower than expected.

4.17.8 Library dependencies

The ARM968CT component has no dependencies on external libraries.

4.17.9 Differences between the CT model and RTL implementations

The ARM968CT component differs from the corresponding revision of the ARM 968 RTL implementation in the following ways:

- No specific differences are listed.

4.17.10 DMA

DMA to or from TCMs can be enabled by connecting the TCM ports to a standard PVBusDecoder. The ARM968CT or any other master can access the TCMs.

The TCM regions do not behave exactly as described in the ARM968E-S TRM:

- DTCM1 is not supported.
- TCM memory does not alias throughout the 4MB TCM regions, only the lowest mapping can be used to access the TCMs. Aliasing can be implemented by appropriate mapping on the PVBusDecoder.

4.18 ARM926CT

Figure 4-19 shows a view of the ARM926CT component in System Canvas. This component is based on r0p5 the ARM926EJ-S™ processor.

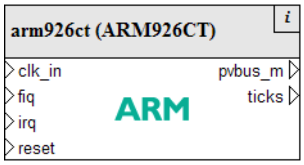


Figure 4-19 ARM926CT in System Canvas

This component is written in C++.

4.18.1 Ports

Table 4-50 provides a brief description of the ports of the ARM926CT component. For more information, see the processor technical reference manual.

Table 4-50 ARM926CT ports

Name	Port protocol	Type	Description
`clk_in`	ClockSignal	slave	clock input
`pvbus_m`	PVBus	master	master port for all memory accesses
`reset`	Signal	slave	asynchronous reset signal input
`irq`	Signal	slave	asynchronous IRQ signal input
`fiq`	Signal	slave	asynchronous FIQ signal input
`ticks`	InstructionCount	master	output that can be connected to a visualization component

4.18.2 Additional protocols

The ARM926CT component has two additional protocols. See *Additional protocols* on page 4-109.

4.18.3 Parameters

Table 4-51 provides a description of the configuration parameters for the ARM926CT component.

Table 4-51 ARM926CT parameters

Parameter	Description	Type	Values	Default
`BIGENDINIT`	initialize to ARMv5 big endian mode	Boolean	true/false	false
`INITRAM`	initialize with ITCM0 enabled at address `0x0`	Boolean	true/false	false
`VINITHI`	initialize with high vectors enabled	Boolean	true/false	false

Table 4-51 ARM926CT parameters (continued)

Parameter	Description	Type	Values	Default
`itcm0_size`	Size of ITCM in KB.	Integer	`0x000 - 0x400`	`0x8`
`dtcm0_size`	Size of DTCM in KB.	Integer	`0x000 - 0x400`	`0x8`
`device-accurate-tlb`	Specify whether all TLBs are modeled.	Boolean	true/false	false[a]
`semihosting-cmd_line`[b]	command line available to semihosting SVC calls	String	no limit except memory	[empty string]
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	true/false	true
`semihosting-ARM_SVC`	A32 SVC number for semihosting	Integer	uint24_t	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting	Integer	uint8_t	`0xAB`
`semihosting-heap_base`	virtual address of heap base	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	virtual address of top of heap	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	virtual address of base of descending stack	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	virtual address of stack limit	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F0000000`

a. Specifying `false` models enables modeling a different number of TLBs if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Specify `true` if device accuracy is required.

b. The value of argv[0] points to the first command line argument, not to the name of an image.

4.18.4 Registers

The ARM926CT component provides the registers specified by the technical reference manual for the ARM926EJ-S with the following exceptions:

- coprocessor 14 registers are not implemented
- integration and test registers are not implemented.

4.18.5 Debug features

The ARM926CT component exports a CADI debug interface.

Registers

All processor, CP15 registers are visible in the debugger. See the processor technical reference manual for a detailed description of available registers.

The CP14 DSCR register is visible for compatibility reasons with some debuggers. This register has no defined behavior.

Breakpoints

There is direct support for:

- single unconditional instruction breakpoints
- unconditional instruction range breakpoints
- single unconditional data breakpoints.

The debugger might augment these with more complex combinations of breakpoints. The current models do not support processor exception breakpoints.

The current models support processor exception breakpoints by the use of pseudoregisters available in the debugger register window. When debugger support is added to directly support processor exceptions, these pseudoregisters are removed.

Setting an exception register to a nonzero value causes execution to stop on entry to the associated exception vector.

Memory

The ARM926CT component presents a single flat 4GB view of virtual memory as seen by the model processor.

4.18.6 Verification and testing

The ARM926CT component has been tested using

- the architecture validation suite tests for the ARM926EJ-S
- booting of Linux on an example system containing an ARM926CT component.

4.18.7 Performance

The ARM926CT component provides high performance in all areas.

4.18.8 Library dependencies

The ARM926CT component has no dependencies on external libraries.

4.18.9 Differences between the CT model and RTL implementations

The ARM926CT component differs from the corresponding revision of the ARM 926 RTL implementation in the following ways:

- There is a single memory port combining instruction, data, and peripheral access.

4.19 Implementation differences

This section outlines the differences between the code translation model implementations and the actual hardware. It contains the following sections:

- *Caches*
- *CP14 Debug coprocessor* on page 4-104
- *MicroTLB* on page 4-104
- *TLB* on page 4-104
- *Memory Access* on page 4-105
- *Timing* on page 4-106
- *VIC Port* on page 4-106.

4.19.1 Caches

ARMv7 processor PV models, such as the Cortex-A9, Cortex-A8, and Cortex-R4, have PV-accurate cache implementation. This means that the cache implementation is sufficient to provide a functionally-accurate model. See the relevant processor component descriptions to find out what specific features are implemented.

All other PV models do not model Level 1 or Level 2 caches. The system coprocessor registers related to cache operations are modeled to permit cache aware software to work, but in most cases they perform no operation other than to check register access permissions.

The registers affected on all code translation processor models are:

- Invalidate and/or Clean Entire ICache/DCache
- Invalidate and/or Clean ICache/DCache by MVA
- Invalidate and/or Clean ICache/DCache by Index
- Invalidate and/or Clean Both Caches
- Cache Dirty Status
- Data Write Barrier
- Data Memory Barrier
- Prefetch ICache Line
- ICache/DCache lockdown
- ICache/DCache master valid.

Additional registers affected on the ARMCortexA8CT model are:

- Preload Engine registers
- Level 1 System array debug registers
- Level 2 System array debug registers
- Level 2 Cache Lockdown
- Level 2 Cache Auxiliary control.

Additional registers affected on the ARMCortexR4CT model are:

- Cache Size Override
- Validation registers.

One additional register is affected on the ARM1176CT model:

- Cache behavior override.

Additional registers affected on the ARM1136CT model are:

- Read Block Transfer Status Register
- Stop Prefetch Range

- Data/Instruction Debug Cache
- Level 1 System Debug registers.

4.19.2 CP14 Debug coprocessor

The CP14 Debug coprocessor registers are fully implemented for Cortex-A15, Cortex-A7 and AEMv7. For all other models, the CP14 Debug coprocessor registers are not implemented apart from the DSCR register. This register reads as 0 and ignores writes. External debugging must be used to debug systems containing PV models.

4.19.3 MicroTLB

The models do not implement a MicroTLB. System coprocessor registers related to MicroTLB state read as 0 and ignore writes.

The registers affected are only in the ARM1136CT component:

- Data/Instruction MicroTLB Index
- Data/Instruction MicroTLB PA
- Data/Instruction MicroTLB VA
- Data/Instruction MicroTlb Attr.

4.19.4 TLB

Translation Lookaside Buffers (TLBs) are implemented in the PV models and most aspects of TLB behavior are fully modeled. Memory attribute settings in TLB entries are currently ignored by PV models of processors because they relate to cache and write buffer behavior.

Note

If the `device-accurate-tlb` parameter is set to `false`, a different number of TLBs is used for the simulation if this improves simulation performance. The simulation is architecturally accurate, but not device accurate. Architectural accuracy is almost always sufficient. Set `device-accurate-tlb` parameter to `true` if device accuracy is required.

TLB registers which have implementations that read as 0 and ignore writes are:

- primary memory remap register
- normal memory remap register.

Additional registers affected on the ARMCortexA9 model are:

- Read Main TLB Entry
- Write Main TLB Entry
- Main TLB VA
- Main TLB PA
- Main TLB Attr.

Additional registers affected on the ARMCortexA8 model are:

- D-TLB CAM read/write
- D-TLB ATTR read/write
- D-TLB PA read/write.

Additional registers affected on the ARM1176CT model are:

- TLB Lockdown Index
- TLB Lockdown VA
- TLB Lockdown PA

- TLB Lockdown Attr.

Additional registers affected on the ARM1136CT model are:

- Read Main TLB Entry
- Read Main TLB VA
- Read Main TLB PA
- Read Main TLB Attr
- Main TLB Master Valid
- TLB Debug Control
- Data memory remap register
- Instruction memory remap register
- DMA memory remap register.

In addition peripheral accesses are not distinguished from data accesses so configuration of the peripheral port memory remap register is ignored.

4.19.5 Memory Access

PV models use a PVBusMaster subcomponent to communicate with slaves in a System Canvas generated system. This provides very efficient access to memory-like slaves and relatively efficient access to device-like slaves.

Memory access in PV models differs from real hardware to enable fast modeling of the processor:

- all memory accesses are performed in programmer view order
- unaligned accesses, where permitted, are always performed as byte transfers.

In addition, some PV models do not currently use all the transaction states available in a PVBus transaction. The Privileged and Instruction flags are set correctly for ARMv7 processors but might not be set correctly in earlier architectures. However all memory accesses are atomic so SWP instructions behave as expected.

I-side memory access

PV models cache translations of instructions fetched from memory-like slaves. The models might not perform further access to those slaves for significant periods. A slave can force the model to reread the memory by declaring that the memory has changed.

PV models do not model a prefetch queue but the code translation mechanism effectively acts as a prefetch queue of variable depth. You are advised to follow the standards in the *ARM® Architecture Reference Manual* for dealing with prefetch issues, such as self modifying code, and use appropriate cache flushing and synchronization barriers.

Translation of instructions only occurs for memory-like slaves, which are those declared by devices as having type `pv::MEMORY`. Instructions fetched from device-like slaves are repeatedly fetched, decoded and executed, significantly slowing down model performance.

D-side memory access

PV models cache references to the underlying memory of memory-like slaves, and might not perform further accesses to those slaves over the bus for significant periods. Slaves declared as having type `pv::MEMORY` provide the fastest possible memory access for PV processors.

DMA

DMA is currently modeled as for D-side memory access.

Peripheral access

Peripheral access is currently modeled as for D-side memory access.

4.19.6 Timing

Programmer's View (PV) models are loosely timed. The models have the following timing behavior:

- Caches and write buffers are not modeled, so all memory access timing is effectively zero wait state.
- All instructions execute, in aggregate, in one cycle of the component master clock input.
- Interrupts are not taken at every instruction boundary.
- Some sequences of instructions are executed atomically, ahead of the master clock of a component, so that system time does not advance during execution. This can sometimes have an effect in sequential access of device registers where devices are expecting time to move on between each access.
- DMA to and from *Tightly Coupled Memory* (TCM) is currently atomic.

4.19.7 VIC Port

The ARMCortexR4CT, ARM1176CT and ARM1136CT models implement a simplified model of the *Vectored Interrupt Controller* (VIC) port. The protocol consists of two ports:

- the vic_addr port used to signal the vectored interrupt address from the external VIC
- the vic_ack port used to signal the VIC that an interrupt has been detected and is being serviced.

The interrupt sequence is expected to be as follows:

1. The software enables the VIC interface by setting the VE bit in the CP15 control register and setting up suitable interrupt routines.
2. The VIC asserts IRQ.
3. Some time later, the processor detects and responds to the IRQ by asserting vic_ack.
4. The VIC writes the vector address to the processor using vic_addr.
5. The processor de-asserts vic_ack.
6. The processor transfers control to the vectored address returned from the VIC.

The interaction between the processor and the VIC is untimed after the processor acknowledges the interrupt, so certain interrupt sequences cannot occur in the code translation processor models.

4.20 Processor CADI implementation of optional functionality

Each processor model implements a subset of full CADI functionality. This section states which elements of CADI functionality the processor components implement.

4.20.1 CADI implementation

Table 4-52 shows the CADI broker implementation.

Table 4-52 CADI broker implementation

Simulation broker	Implemented	Remarks
Factories	1	Model DLL provides a single factory
Instances	1	Factory can only instantiate one model at a time

Table 4-53 shows the CADI target implementation.

Table 4-53 CADI target implementation

Processor target	Implemented	Remarks
Runnable	yes	
Intrusive Debug	yes	Processor scheduling is affected by stop/start
Stop simulation	yes	
Reset Levels	1	
Memory read/write	yes[a]	
# Breakpoints	32	
Breakpoint setting	yes[a]	
Breakpoint disabling	yes	Global breakpoints can be disabled, not processor breakpoints
Breakpoint types	`CADI_BPT_PROGRAM` `CADI_BPT_PROGRAM_RANGE` `CADI_BPT_MEMORY`	
Breakpoint triggers	`CADI_BPT_TRIGGER_ON_READ` `CADI_BPT_TRIGGER_ON_WRITE` `CADI_BPT_TRIGGER_ON_MODIFY`	
Breakpoint ignore count	no	
Breakpoint free-form conditions	no	
Breakpoint formal conditions	no	
Virtual To Physical	no	
CADICache API	no	

a. Not permitted while target is running.

4.21 CADI interactions with processor behavior

M-series processors are architecturally defined to have different reset behavior to that of A and R series processors.

For all processors, that is, in hardware and in the models, a reset consists of asserting, and then deasserting the reset pin. However:

- The M-series architecture documentation specify that on asserting the reset pin, the processor stops executing instructions. On deasserting the reset pin, the VTOR is given its reset value, SP_main is read from address VTOR+0, and the PC is read from address VTOR+4. See the Reset behavior section and the Vector Table Offset Register, VTOR section in the *ARMv7-M Architecture Reference Manual*. Also, the processor begins fetching and executing instructions.
- For the A and R series, asserting the reset pin causes the processor to stop executing instructions and the PC to be given its reset value, which depends on VINITHI, and the SP is `UNKNOWN`. See the Reset section in the *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition*. On deasserting the reset pin, the processor begins fetching and executing instructions.

The behavior of reset on M series interacts differently with CADI debugging functionality than it does for A and R series. On A and R series, after the reset pin of the processor is asserted, a new application can be loaded using CADIExecLoadApplication() call in CADI. This loads an application into memory, and sets the PC if a start address is specified in the application. When reset is deasserted, the processor begins fetching and executing from the start address as expected. Similarly, if a debugger asserts reset on the processor, it can modify memory with a sequence of CADIMemWrite() calls, update the PC with a CADIRegWrite() call, and when reset is deasserted the processor will begin fetching and executing from PC.

On M series processors the above techniques do not normally work, because the processor updates SP_main and the PC after reset is deasserted, and so overwrites the SP_main and PC set by the CADI calls.

To make it possible for these techniques to be used, the M series processor models differ slightly from the architectural reset behavior. When reset is asserted, the M series makes note of whether the PC or SP are modified before reset is next deasserted. If there are any CADI writes to the PC or SP registers, either directly through CADIRegWrite or indirectly through CADIExecLoadApplication(), this is tracked. When reset is deasserted, if the PC has been modified using CADI, the PC retains the value written to it, otherwise it reads it from address VTOR+0. Similarly, if the SP has been modified using CADI, SP_main retains the value written to it, otherwise it reads it from address VTOR+4.

4.22 Additional protocols

All processor components have two additional protocols.

4.22.1 InstructionCount

The InstructionCount protocol has two behaviors:

`getValue() : uint64_t`

Obtain the number of instructions executed by the processor

`getRunState() : uint32_t`

Obtain the power/run status of the processor.

The possible run state values are shown in Table 4-54:

Table 4-54 Possible run state values

Value	State label	Description
`0x0`	`UNKNOWN`	Run status unknown, that is, simulation has not started.
`0x1`	`RUNNING`	Processor running, is not idle and is executing instructions.
`0x2`	`HALTED`	External halt signal asserted.
`0x3`	`STANDBY_WFE`	Last instruction executed was WFE and standby mode has been entered.
`0x4`	`STANDBY_WFI`	Last instruction executed was WFI and standby mode has been entered.
`0x5`	`IN_RESET`	External reset signal asserted.
`0x6`	`DORMANT`	Partial processor power down.
`0x7`	`SHUTDOWN`	Complete processor power down.

Chapter 5
Peripheral and Interface Components

This chapter describes generic peripheral components included with Fast Models Portfolio. It contains the following sections:

5.1 About the components

The *Programmer's View* (PV) models of processors and devices work at a level where functional behavior is equivalent to what you would see if you were using real hardware. Timing accuracy is sacrificed to achieve fast simulation execution speeds. This means that you can use the PV models for confirming software functionality, but you must not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior.

The major components are:

- bus components, including conversions between AMBA and PV Model components for use with the SystemC export feature of Fast Models
- input/output devices
- memory, including flash
- Ethernet controller
- interrupt controllers
- static and dynamic memory controllers
- audio interface
- programmable clock generators.

Some components are software implementations of specific hardware functionality. Components based on PrimeCells are an example. Other components are necessary as part of the modeling environment and do not represent hardware. Examples of the latter category of component are the bus, clock, telnet and visualization components.

Components based on specific hardware functionality can be provided as either:

- Validated Peripheral Components: components that are functionally complete, or have clearly specified limitations. These components can be used to create virtual platforms and have been validated.

- Example Peripheral Components: components that have been developed as examples to illustrate ways to model different component types and to enable the creation of specific example virtual platforms. These components can be partially validated.

5.2 PV Bus components

This section introduces the bus components that are provided with Fast Models. It contains the following sections:

- *About the PVBus components*
- *PVBusDecoder component* on page 5-4
- *PVBusMaster component* on page 5-6
- *PVBusRange component* on page 5-7
- *PVBusSlave component* on page 5-8
- *TZSwitch component* on page 5-12
- *C++ classes* on page 5-14.

5.2.1 About the PVBus components

The PVBus components and protocols provide mechanisms for implementing functionally-accurate communication between bus masters and slaves. There is no modeling of handshaking or cycle counts. By removing this level of detail, and by using efficient internal communication mechanisms, PVBus components can provide very fast modeling of bus accesses.

PVBus components are not software implementations of specific hardware, but are instead abstract components required for the software model environment.

The PVBus API is accessed through a number of components, which abstract the internal details. These components must be used correctly to achieve high simulation speeds.

PVBus system components

A bus system consists of a number of bus masters, some infrastructure and a number of bus slaves. Each bus master must contain a PVBusMaster subcomponent, and each bus slave must contain a PVBusSlave subcomponent. These subcomponents provide PVBus master and slave ports. Each PVBus master port can only be connected to one slave, but any number of other masters can be connected to the same slave. PVBusDecoder, PVBusMaster and PVBusSlave components communicate using the PVBus protocol.

PVBusDecoder components can be added to the bus system. Each of these permits its masters to be connected to multiple slaves, each associated with a different bus address range.

PVBusSlave subcomponents provide built-in support for declaring memory-like, device-like, abort or ignore address ranges. PVBus has support for dealing efficiently with memory-like devices such as RAM, ROM and Flash.

Figure 5-1 on page 5-4 demonstrates a sample bus topology:

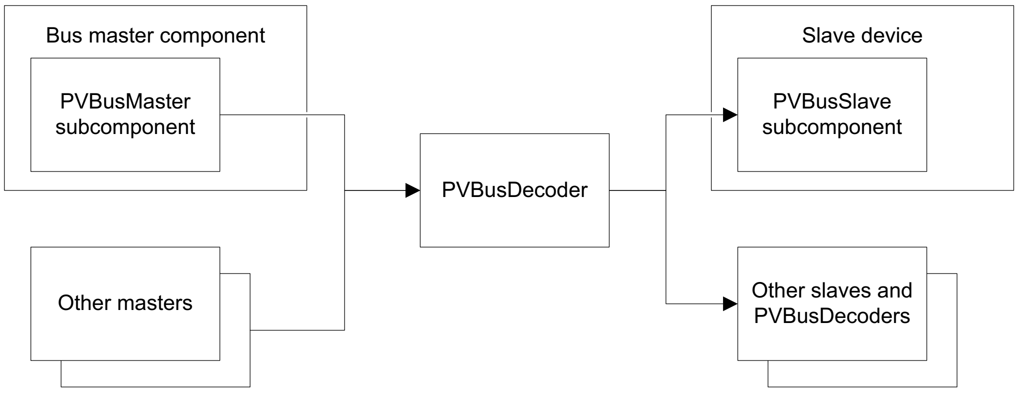


Figure 5-1 Sample bus topology

All communication over the PVBus is performed using transactions that are generated by PVBusMaster subcomponents and fulfilled by PVBusSlave components. Transactions can be routed to the slave device through its PVBusSlave subcomponent. When configured, the PVBusSlave can handle memory-like transactions efficiently without having to route these transactions to the slave device. All transactions are atomic and untimed, in keeping with the programmer's view abstraction.

PVBus examples

This document contains some examples of PVBus components in use. Additional examples can be found in Fast Models. Paths to these examples are provided in Table 5-1. For paths on Linux, substitute `$PVLIB_HOME` for the Microsoft Windows environment variable `%PVLIB_HOME%`.

Table 5-1 PVBus examples

Path	Description
`%PVLIB_HOME%\LISA\CounterModule.lisa`	A full implementation of a bus slave device (CounterTimer module).
`%PVLIB_HOME%\examples\FVP_EB\LISA\FVP_EB_ARM1176.lisa`	Example of moderately complex bus topology, using the PVBusDecoder component.
`%PVLIB_HOME%\examples\Common\LISA\RemapDecoder.lisa`	An example that dynamically modifies routing of requests based on a remap signal, using the TZSwitch component.

5.2.2 PVBusDecoder component

The PVBusDecoder provides support for mapping slave devices to different address ranges on a bus. Incoming bus requests are routed to the appropriate slave device, based on the transaction address.

Figure 5-2 shows a view of the component in System Canvas.

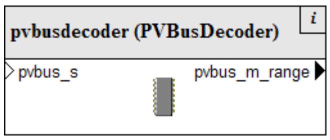


Figure 5-2 PVBusDecoder in System Canvas

This component is written in C++.

Ports

Table 5-2 provides a brief description of the PVBusDecoder component ports.

Table 5-2 PVBusDecoder ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Accepts incoming transactions. Connect this port to a bus master, or to the output of another bus decoder.
pvbus_m_range	addressable PVBus	Master	Specifies the address range for the bus master. The range must be 4KB aligned and a multiple of 4KB in size. If the address range is larger than the size of the slave device, the slave is aliased.[a]

a. Each slave connection is associated with a specific address range on the `pvbus_m_range` port. In LISA, the syntax for this is `decoder.pvbus_m_range[start..end] = slave.pvbus.` The values for `start` (inclusive) and `end` (inclusive) must specify a 4KB aligned region of a multiple of 4K bytes. You can specify an address range for the slave, where the decoder remaps addresses into the appropriate range.

Additional protocols

The PVBusDecoder component has no additional protocols.

Parameters

The PVBusDecoder component has no parameters.

Registers

The PVBusDecoder component has no registers.

Debug features

The PVBusDecoder component has no debug features.

Verification and testing

The PVBusDecoder component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PVBusDecoder component is does not significantly affect the performance of a PV system.

Library dependencies

The PVBusDecoder component has no dependencies on external libraries.

5.2.3 PVBusMaster component

The PVBusMaster component acts as a source for all transactions over the PVBus. If you want a component to act as a bus master, instantiate a PVBusMaster subcomponent and then use its control port to create one or more transaction generators to generate transactions on the bus.

Figure 5-3 shows a view of the component in System Canvas.

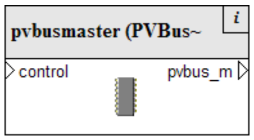


Figure 5-3 PVBusMaster in System Canvas

This component is written in C++.

Ports

Table 5-3 provides a brief description of the PVBusMaster ports.

Table 5-3 PVBusMaster ports

Name	Port protocol	Type	Description
pvbus_m	PVBus	Master	Sends out generated transactions.
control	PVTransactionMaster	Slave	Enables the owning component to instantiate `pv::TransactionGenerator` objects. See *PVTransactionMaster protocol*.

PVTransactionMaster protocol

The PVBusMaster component has one additional protocol.

This protocol is used to instantiate a `pv::TransactionGenerator` object from a PVBusMaster:

`createTransactionGenerator( ) : pv::TransactionGenerator *`

Instantiates a new transaction generator to control this bus master. A caller can allocate as many TransactionGenerators as it wants. It is up to the caller to delete TransactionGenerators when they are no longer required. For example:

```
behavior init() {
    tg = master.createTransactionGenerator();
}

behavior destroy() {
    delete tg;
}
```

Parameters

The PVBusMaster component has no parameters.

Registers

The PVBusMaster component has no registers.

Debug features

The PVBusMaster component has no debug features.

Verification and testing

The PVBusMaster component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PVBusMaster component is not expected to significantly affect the performance of a PV system. However, the way in which you implement the component can influence performance. See *TransactionGenerator efficiency considerations* on page 5-14.

Library dependencies

The PVBusMaster component has no dependencies on external libraries.

5.2.4 PVBusRange component

PVBusRange is used to divide and remap the memory range into two output ports, a and b. This component is currently used to implement the *TrustZone Memory Adaptor* (TZMA).

You must only use PVBusRange components if the routing decisions change infrequently, for example as part of a memory remap.

Figure 5-4 shows a view of the component in System Canvas.

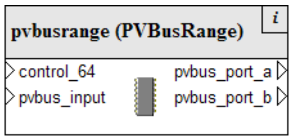


Figure 5-4 PVBusRange in System Canvas

This component is written in C++.

Ports

Table 5-4 provides a brief description of the PVBusRange ports.

Table 5-4 PVBusRange ports

Name	Port protocol	Type	Description
control_64	Value_64	Slave	A 64 bit value port used to signal the remapped range, X.
pvbus_input	PVBus	Slave	PVBus input.
pvbus_port_a	PVBus	Master	PVBus output, with remapped address range of [0, X].
pvbus_port_b	PVBus	Master	PVBus output, with remapped address range of [X+1, msize].

Value_64 protocol

The PVBusRange component has one additional protocol.

The Value_64 protocol provides the rules to communicate with the TZMA to signal the remapped range X.

`setValue(uint64_t value)`

Sets a 64-bit wide address to the slave port.

Parameters

The PVBusRange component has no parameters

Registers

The PVBusRange component has no registers.

Debug features

The PVBusRange component has no debug features.

Verification and testing

The PVBusRange component has been tested by directed component tests.

Performance

The PVBusRange component is optimized to have negligible impact on transaction performance except when memory remap settings are changed when there might be a significant effect.

Library dependencies

The PVBusRange component has no dependencies on external libraries.

5.2.5 PVBusSlave component

The PVBusSlave component handles decoding the slave side of the PVBus. Any component that acts as a bus slave must:

- provide a PVBus slave port
- instantiate a PVBusSlave subcomponent, with the size parameter configured for the address range covered by the device
- connect the slave port to the `pvbus_s` port on the PVBusSlave.

By default, the PVBusSlave translates all transactions into `read()` and `write()` requests on the device port.

The control port enables you to configure the PVBusSlave for a device. This lets you define the behavior of the different regions of the address space on the device. You can configure regions separately for read and write, at a 4k byte granularity. This permits you to set memory-like, device-like, abort or ignore access address regions.

Transactions to memory-like regions are handled internally by the PVBusSlave subcomponent. Abort and ignore regions are also handled by the PVBusSlave.

Transactions to device-like regions are forwarded to the device port. Your device must implement the `read()` and `write()` methods on the device port if any regions are configured as device-like.

Figure 5-5 shows a view of the component in System Canvas.

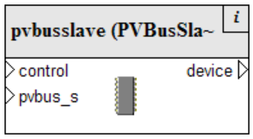


Figure 5-5 PVBusSlave in System Canvas

This component is written in C++.

Ports

Table 5-5 provides a brief description of the ports in the PVBus component.

Table 5-5 PVBus ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Handles incoming requests from bus masters.
device	PVDevice	Master	Passes on requests for peripheral register accesses to permit the owning component to handle the request.
control	PVBusSlave-Control	Slave	Enables the owning component to control which regions of the device memory are to be handled as RAM/ROM/Device. These settings can be changed dynamically. For example, when a Flash component is being programmed, it can switch to treating reads as Device requests instead of ROM requests.

PVDevice protocol

The PVDevice protocol enables you to implement support for memory-mapped device registers. The two methods are called through the device port on the PVBusSlave to handle bus read/write transactions:

`read(pv::ReadTransaction) : pv::Tx_Result`

This method permits a device to handle a bus read transaction.

`write(pv::WriteTransaction) : pv::Tx_Result`

This method permits a device to handle a bus write transaction.

The C++ transaction and `Tx_Result` classes are described in further detail. See *C++ classes* on page 5-14.

The PVDevice protocol uses two optional behaviors to differentiate between transaction originating from the processor (loads and stores) and transactions originating from an attached debugger:

`optional slave behavior debugRead(pv::ReadTransaction tx) : pv::Tx_Result`

If implemented, this method enables the device to handle a debug read transaction.

`optional slave behavior debugWrite(pv::WriteTransaction tx) : pv::Tx_Result`

If implemented, this method enables the device to handle a debug write transaction.

If the `debugRead` and `debugWrite` behaviors are implemented, they are called for all debug transactions. If they are not implemented, debug transactions are handled by the `read` and `write` behaviors.

PVBusSlaveControl

`setFillPattern(uint32_t fill1, uint32_t fill2)`

This sets a two-word alternating fill pattern to be used by uninitialized memory.

`setAccess(pv::bus_addr_t base, pv::bus_addr_t top, pv::accessType type, pv::accessMode mode)`

This reconfigures handling for a region of memory.

`base` (inclusive value) and `top` (exclusive value) specify the address range to configure. Both `base` and `top` values must be 4KB page aligned.

`type` selects what types of bus access must be reconfigured. It can be one of:

- `pv::ACCESSTYPE_READ`
- `pv::ACCESSTYPE_WRITE`
- `pv::ACCESSTYPE_RW`

`mode` defines how bus accesses must be handled. See *pv::accessMode values* on page 5-11.

`getReadStorage(pv::bus_addr_t address, pv::bus_addr_t *limit) : const uint8_t*`
`getWriteStorage(pv::bus_addr_t address, pv::bus_addr_t *limit) : uint8_t *`

These two methods permit you to access the underlying storage that PVBusSlave allocates to implement a region of memory.

The return value is a pointer to the byte that represents the storage corresponding to the address of `base`.

Memory is stored in blocks of at least 4KB, so the returned pointer can access all memory within a block. If a `limit` pointer is provided, it is used to return the device address corresponding to the limit of the current block.

The pointer value returned is not guaranteed to remain valid indefinitely. Bus activities, or other calls to the control port, might invalidate the pointer.

`provideReadStorage(pv::bus_addr_t base, pv::bus_addr_t limit, const uint8_t * storage)`,
`provideReadWriteStorage(pv::bus_addr_t base,pv::bus_addr_t limit, uint8_t * storage)`

These methods enable you to allocate blocks of memory that the PVBusSlave can use to manage regions of RAM/ROM. Only use these methods when you require a high degree of control over memory, such as when you require a device to map specific regions of host memory into the simulation.

You must align `base` (inclusive) and `limit` (exclusive) values to 4KB page addresses within device address space. The memory region pointed to by `storage` must be large enough to contain `(limit - base)` bytes.

After these calls, PVBusSlave controls access to the underlying memory. The owner must call `getWriteStorage()` before modifying the memory contents and `getReadStorage()` before reading the memory contents.

pv::accessMode values

The values assigned to `pv::accessMode` control what happens when an address is accessed. Legal values for the enumeration are:

`pv::ACCESSMODE_MEMORY`

Act as memory. The PVBusSlave manages the underlying storage to provide 4KB of memory, which can be ROM or RAM, depending on how you configure it to handle bus write transactions.

`pv::ACCESSMODE_DEVICE`

Act as a device. Requests to the select pages are routed to the PVBusSlave `device` port, where the necessary behavior can be implemented by the component.

`pv::ACCESSMODE_ABORT`

Generate bus abort signals for any accesses to this page.

`pv::ACCESSMODE_IGNORE`

Ignore accesses to this page. Bus read requests return 0.

Parameters

Table 5-6 provides a description of the configuration parameters for the PVBusSlave component.

Table 5-6 PVBusSlave configuration parameters

Parameter name	Description	Type	Allowed value	Default value
size	Addressable size of the device in bytes.	uint64_t	Must be a multiple of `0x1000` (4KB).	`0x10000000` (256MB)

Registers

The PVBusSlave component has no registers.

Debug features

The PVBusSlave component has no debug features.

Verification and testing

The PVBusSlave component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PVBusSlave component, typically, does not significantly affect the performance of a PV system. However, correct implementation of the PVBusSlave component is critical to the performance of the overall PV system. For example, routing a request to a PVDevice port is slower than letting the PVBusSlave handle the request internally, so you are strongly encouraged to use the PVBusSlave component support for memory-like regions where possible.

Library dependencies

The PVBusSlave component has no dependencies on external libraries.

5.2.6 TZSwitch component

The TZSwitch component permits you to control transaction routing based on the TrustZone security state of the transaction. The routing decisions can be changed dynamically. Transactions received on the pvbus_input port can be routed to either output port, or can generate an abort or be ignored.

The default behavior is to forward secure transactions to pvbus_port_a, and normal transactions to pvbus_port_b.

You must only use TZSwitches if the routing decisions change infrequently, for example as part of a memory remap.

Figure 5-6 shows a view of the component in System Canvas.

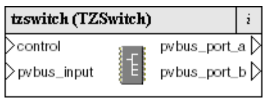


Figure 5-6 TZSwitch in System Canvas

The TZSwitch component is written in C++.

Ports

Table 5-7 provides a brief description of the TZSwitch component ports.

Table 5-7 TZSwitch ports

Name	Port protocol	Type	Description
pvbus_input	PVBus	Slave	Slave port for connection to PVBus master/decoder.
pvbus_port_a	PVBus	Master	Output port a.
pvbus_port_b	PVBus	Master	Output port b.
control	TZSwitchControl	Slave	Controls routing of transactions.

TZSwitchControl

The TZSwitch component has one additional protocol.

`routeAccesses (TZSwitch_InputFilter input, TZSwitch_RouteOption destination) : void`

Controls the routing of transactions on a TZSwitch component. Select the type of signals for reconfiguring by setting the value of `input` and `destination`. The possible values of `input` are:

TZINPUT_SECURE
changes the routing for secure transactions

TZINPUT_NORMAL
changes the routing for normal transactions

TZINPUT_ANY
changes the routing for all transactions.

The possible values of `destination` are:

TZROUTE_ABORT
causes transactions to generate an abort

TZROUTE_IGNORE
causes transactions to be ignored, and reads return 0

TZROUTE_TO_PORT_A
routes transactions to pvbus_port_a

TZROUTE_TO_PORT_B
routes transactions to pvbus_port_b.

Parameters

Table 5-8 lists the configuration parameters of the TZSwitch component.

Table 5-8 TZSwitch configuration parameters

Parameter name	Description	Type	Allowed value	Default value
secure	Default routing for secure transactions.	Integer	0, 1, 2, 3	1
normal	Default routing for normal transactions.	Integer	0, 1, 2, 3	2

——— **Note** ———

The secure and normal parameter values control the initial state of the TZSwitch component. Possible values for these parameters are:

0	ignore these transactions
1	forward the transactions to pvbus_port_a
2	forward the transactions to pvbus_port_b
3	generate an abort for these transactions.

The numbers used for initial configuration are not the same as the enumeration constants used to control routing at runtime.

Registers

The TZSwitch component has no registers.

Debug features

The TZSwitch component has no debug features.

Verification and testing

The TZSwitch component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The TZSwitch component is optimized to have negligible impact on transaction performance.

Note

When memory remap settings are changed when there might be a significant effect on performance.

Library dependencies

The TZSwitch component has no dependencies on external libraries.

5.2.7 C++ classes

This section describes:

- *Class pv::TransactionGenerator*
- *Class pv::Transaction* on page 5-15
- *Class pv::ReadTransaction* on page 5-16
- *Class pv::WriteTransaction* on page 5-16.

Class pv::TransactionGenerator

The `pv::TransactionGenerator` class provides efficient mechanisms for bus masters to generate transactions that are transmitted over the pvbus_m port of the associated PVBusMaster subcomponent.

You can produce `pv::TransactionGenerator` objects by invoking the `createTransactionGenerator()` method on the control port of a PVBusMaster component.

TransactionGenerator efficiency considerations

Each Transaction Generator caches connection information internally. This improves efficiency for multiple accesses to a single 4KB page. If a component requires repeated access data from a number of different pages, for example when streaming from one location to another, you are advised to create one transaction generator for each location being accessed.

You can dynamically create and destroy Transaction Generators, but it is better to allocate them once at initialization and destroy them at shutdown. See the DMA example in the examples directory of the System Canvas Model Library distribution.

TransactionGenerator class definition

```
class pv::TransactionGenerator
{
    // Tidy up when TransactionGenerator is deleted.
    ~TransactionGenerator()

    // Control AXI-specific signal generation for future transactions.
    // Privileged processing mode.
    void setPrivileged(bool priv = true);
    // Instruction access (vs data).
    void setInstruction(bool instr = true);
    // Normal-world access (vs secure).
    void setNonSecure(bool ns = true);
    // Locked atomic access.
    void setLocked(bool locked = true);
    // Exclusive atomic access access.
    void setExclusive(bool excl = true);

    // Generate transactions
    // Generate a read transaction.
    bool read(bus_addr_t, pv::AccessWidth width, uint32_t *data);
    // Generate a write transaction.
    bool write(bus_addr_t, pv::AccessWidth width, uint32_t const *data);

    // Generate read transactions.
    bool read8(bus_addr_t, uint8_t *data);
    bool read16(bus_addr_t, uint16_t *data);
    bool read32(bus_addr_t, uint32_t *data);
    bool read64(bus_addr_t, uint64_t *data);

    // Generate write transactions.
    bool write8(bus_addr_t, uint8_t const *data);
    bool write16(bus_addr_t, uint16_t const *data);
    bool write32(bus_addr_t, uint32_t const *data);
    bool write64(bus_addr_t, uint64_t const *data);
};
```

Enum pv::AccessWidth

Selects the required bus width for a transaction. Defined values are:

- `pv::ACCESS_8_BITS`
- `pv::ACCESS_16_BITS`
- `pv::ACCESS_32_BITS`
- `pv::ACCESS_64_BITS`.

Class pv::Transaction

The `pv::Transaction` class is a base class for read and write transactions that are visible in the PVBusSlave subcomponent. The class contains functionality common to both types of transaction.

The `pv::Transaction` class provides an interface that permits bus slaves to access the details of the transaction. Do not instantiate these classes manually. The classes are generated internally by the PVBus infrastructure.

This base class provides access methods to get the transaction address, access width, and bus signals. It also provides a method to signal that the transaction has been aborted.

```
class pv::Transaction
{
    public:
    // Accessors
    bus_addr_t getAddress() const;          // Transaction address.
    pv::AccessWidth getAccessWidth() const; // Request width.
    int getAccessByteWidth() const;         // Request width in bytes.
    int getAccessBitWidth() const;          // Request width in bits.

    bool isPrivileged() const;  // Privileged process mode?
    bool isInstruction() const; // Instruction request vs data?
    bool isNonSecure() const;   // Normal-world vs secure-world?
    bool isLocked() const;      // Atomic locked access?
    bool isExclusive() const;   // Atomic exclusive access?
    uint32_t getMasterID() const;
    bool hasSideEffect() const;

    // Generate transaction returns

    Tx_Result  generateAbort();          // Cause the transaction to abort
    Tx_Result  generateSlaveAbort();     // Cause the transaction to abort
    Tx_Result  generateDecodeAbort();    // Cause the transaction to abort
    Tx_Result  generateExclusiveAbort(); // Cause the transaction to abort
    Tx_Result  generateIgnore();
};
```

Class pv::ReadTransaction

The `pv::ReadTransaction` extends the `pv::Transaction` class to provide methods for returning data from a bus read request.

```
class ReadTransaction : public Transaction
{
    public:
    /*! Return a 64-bit value on the bus. */
    Tx_Result setReturnData64(uint64_t);

    /*! Return a 32-bit value on the bus. */
    Tx_Result setReturnData32(uint32_t);

    /*! Return a 16-bit value on the bus. */
    Tx_Result setReturnData16(uint16_t);

    /*! Return an 8-bit value on the bus. */
    Tx_Result setReturnData8(uint8_t);

    /*! This method provides an alternative way of returning a Tx_Result
    * success value (instead of just using the value returned from
    * setReturnData<n>()).
    *
    * This method can only be called if one of the setReturnData<n>
    * methods has already been called for the current transaction.
    */
    Tx_Result readComplete();
};
```

Class pv::WriteTransaction

The `pv::WriteTransaction` extends the `pv::Transaction` class to provide methods for returning data from a bus write request.

```
class WriteTransaction : public Transaction
{
    public:
    /*! Get bottom 64-bits of data from the bus. If the transaction width
    * is less than 64-bits, the data will be extended as appropriate.
    */
    uint64_t getData64() const;

    /*! Get bottom 32-bits of data from the bus. If the transaction width
    * is less than 32-bits, the data will be extended as appropriate.
    */
    uint32_t getData32() const;

    /*! Get bottom 16-bits of data from the bus. If the transaction width
    * is less than 16-bits, the data will be extended as appropriate.
    */
    uint16_t getData16() const;

    /*! Get bottom 8-bits of data from the bus. If the transaction width
    * is less than 8-bits, the data will be extended as appropriate.
    */
    uint8_t getData8() const;

    /*! Signal that the slave has handled the write successfully.
    */
    Tx_Result writeComplete();
};
```

5.3 AMBA-PV Components

This section contains information about AMBA components of Fast Models. It contains the following sections:

5.3.1 About the AMBA-PV components

The AMBA-PV Components are part of Fast Models.

The AMBA-PV Components provide you with components and protocols that permit you to model a platform that interfaces with an AMBA-based system. The system itself is modeled using *Open SystemC Initiative Transaction Level Modeling* (OSCI TLM) at *Programmer's View* (PV) level using the SystemC Export functionality of System Canvas. For more information about SystemC Export usage, see the *Fast Models User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html`.

The protocols provided with the AMBA-PV Components include:

AMBAPV Models the AMBA protocols AXI4™, AXI3™, AHB™ and APB™. See *AMBA-PV component protocols* on page 5-19.

AMBAPVACE

Models the AMBA ACE and DVM protocols. See *AMBA-PV component protocols* on page 5-19.

AMBAPVSignal

Models interrupts. See *AMBAPVSignal protocol* on page 5-23.

AMBAPVSignalState

Transfers and receives signal states. See *AMBAPVSignalState protocol* on page 5-23.

AMBAPVValue

Models propagation of 32 bit integer values between components. See *AMBAPVValue protocol* on page 5-24.

AMBAPVValue64

Models propagation of 64 bit integer values between components. See *AMBAPVValue64 protocol* on page 5-24.

AMBAPVValueState

Permits a master to retrieve the current value from a slave, using 32 bit integer values. See *AMBAPVValueState protocol* on page 5-24.

AMBAPVValueState64

Permits a master to retrieve the current value from a slave, using 64 bit integer values. See *AMBAPVValueState64 protocol* on page 5-24.

There are ready-to-use components that provide you with conversions between PVBus and AMBAPV protocols, and between Signal and AMBAPVSignal protocols, StateSignal and AMBAPVSignalState, Value(_64) and AMBAPVValue(64), and ValueState(_64) and AMBAPVValueState(64) protocols. There are examples of use of the AMBA-PV components in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, the examples can be found in `$PVLIB_HOME/examples/SystemCExport`.

For more information about the PVBus protocols and components, see *PV Bus components* on page 5-3.

The protocols and components are designed to interface with the AMBA TLM PV library for OSCI TLM 2.0. This library is delivered with Fast Models and is intended to providing a standard way of mapping the AMBA protocol on top of OSCI TLM 2.0.1 kit at PV level.

For more information about the AMBA TLM PV library for OSCI TLM 2.0.1 kit, see the documentation provided with Fast Models in `%MAXCORE_HOME%\AMBA-PV\doc`. On Linux, use the `$MAXCORE_HOME` environment variable instead.

For more information about the OSCI TLM 2.0, see the documentation provided with the kit. This can be downloaded from the OSCI, `http://www.systemc.org` web site.

5.3.2 AMBA-PV component protocols

The AMBA-PV component protocols are:

- *AMBAPV protocol*
- *AMBAPVACE protocol* on page 5-21
- *AMBAPVSignal protocol* on page 5-23
- *AMBAPVSignalState protocol* on page 5-23
- *AMBAPVValue protocol* on page 5-24
- *AMBAPVValue64 protocol* on page 5-24
- *AMBAPVValueState protocol* on page 5-24
- *AMBAPVValueState64 protocol* on page 5-24.

AMBAPV protocol

The AMBAPV protocol defines behaviors for single read and single write transactions. This covers AMBA AXI4, AXI3, AHB and APB bus protocol families, all at the PV level. In addition, the AMBAPV protocol provides support for AMBA protocol additional control information, including:

- protection units
- exclusive access and locked access mechanisms
- system-level caches.

The behaviors of the protocol are:

`read()`
: Completes a single read transaction at the given address for the given size in bytes. Additional AMBA protocol control information can be specified using the ctrl parameter. The `socket_id` parameter must be set to 0 in this context.

```
read(int socket_id, const sc_dt::uint64 & addr,
    unsigned char * data, unsigned int size,
    const amba_pv::amba_pv_control * ctrl,
    sc_core::sc_time & t):
    amba_pv::amba_pv_resp_t
```

`write()`
: Completes a single write transaction at the given address with specified data and write strobes. The size of the data is specified in bytes. Additional AMBA protocol control information can be specified using the ctrl parameter. The `socket_id` parameter must be set to 0 in this context.

```
write(int socket_id, const sc_dt::uint64 & addr,
    unsigned char * data, unsigned int size,
    const amba_pv::amba_pv_control * ctrl,
    unsigned char * strb, sc_core::sc_time & t):
    amba_pv::amba_pv_resp_t
```

`debug_read()`
: Completes a debug read transaction from a given address without causing any side effects. Specify the number of bytes to read in the `length` parameter. The number of successfully read values is returned. Additional AMBA protocol control information can be specified in the `ctrl` parameter. The `socket_id` parameter must be set to 0 in this context. This behavior is empty by default and returns 0.

```
debug_read(int socket_id, const sc_dt::uint64 & addr,
    unsigned char * data, unsigned int length,
    const amba_pv::amba_pv_control * ctrl):
    unsigned int
```

`debug_write()`

Completes a debug write transaction to a given address without causing any side effects. Specify the number of bytes to write in the `length` parameter. The number of successfully written values is returned. Additional AMBA protocol control information can be specified in the `ctrl` parameter. The `socket_id` parameter must be set to 0 in this context. This behavior is empty by default and returns 0.

```
debug_write(int socket_id, const sc_dt::uint64 & addr,
    unsigned char * data, unsigned int length,
    const amba_pv::amba_pv_control * ctrl):
    unsigned int
```

`b_transport()`

This is an optional slave behavior for blocking transport. It completes a single transaction using the blocking transport interface. The `amba_pv::amba_pv_extension` must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
b_transport(int socket_id,
    amba_pv::amba_pv_transaction & trans, sc_core::sctime & t)
```

`transport_dbg()`

This optional slave behavior implements the TLM debug transport interface. An `amba_pv::amba_pv_extension` object must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
transport_dbg(int socket_id,
        amba_pv::amba_pv_transaction & trans, sc_core::sctime & t):
```

```
unsigned int
```

`get_direct_mem_ptr()`

This is an optional slave behavior for requesting a DMI access to a given address. It returns a reference to a DMI descriptor that contains the bounds of the granted DMI region. Returns `true` if a DMI region is granted, `false` otherwise.

The `amba_pv::amba_pv_extension` must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
get_direct_mem_ptr(int socket_id,
    amba_pv::amba_pv_transaction & trans,
    tlm::tlm_dmi & dmi_data):
    bool
```

`invalidate_direct_mem_ptr()`

This is an optional master behavior to invalidate a DMI request. It invalidates DMI pointers that were previously established for the given DMI region. The `socket_id` parameter is 0 in this context.

```
invalidate_direct_mem_ptr(int socket_id,
    sc_dt::uint64 start_range, sc_dt::uint64 end_range)
```

For more information about the SystemC/TLM `amba_pv::amba_pv_control` and `amba_pv::amba_pv_status` classes, see the *AMBA-PV Extensions to OSCI TLM 2.0 Developer Guide*.

The contents of the generic payload data is formatted as an array of bytes in order of ascending bus address. This means that irrespective of the host machine endianness or modeled bus width, a little endian master should write the bytes of a word in increasing significance as the array index increases and a big endian master should write the bytes of a word in decreasing significance as the array index increases. A master or slave whose endianness does not match the endianness of the host machine must endian swap any access to the payload data that is wider than one byte. The same byte ordering rule applies to memory accesses using DMI pointers.

AMBAPVACE protocol

The AMBAPVACE protocol defines behaviors for bus transactions. This covers AMBA ACE and DVM bus protocol families, all at the PV level. In addition, the AMBAPV protocol provides support for AMBA protocol additional extension information, including:

- secure and privileged accesses
- atomic operations
- system-level caching and buffering control
- cache coherency transactions (ACE-Lite)
- bi-directional cache coherency transactions (ACE)
- distributed virtual memory transactions (DVM).

The behaviors of the protocol are:

`b_transport()`

This slave behavior implements the TLM blocking transport interface. An `amba_pv::amba_pv_extension` object must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
b_transport(int socket_id,
        amba_pv::amba_pv_transaction & trans, sc_core::sctime & t)
```

`transport_dbg()`

This optional slave behavior implements the TLM debug transport interface. An `amba_pv::amba_pv_extension` object must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
transport_dbg(int socket_id,
        amba_pv::amba_pv_transaction & trans, sc_core::sctime & t):
        unsigned int
```

`get_direct_mem_ptr()`

This optional slave behavior is for requesting a DMI access to a given address. It returns a reference to a DMI descriptor that contains the bounds of the granted DMI region. Returns true if a DMI region is granted, false otherwise. An `amba_pv::amba_pv_extension` object must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
get_direct_mem_ptr(int socket_id,
        amba_pv::amba_pv_transaction & trans,
        tlm::tlm_dmi & dmi_data):
        bool
```

`b_snoop()`

This master behavior implements an upstream snooping TLM blocking transport interface. An `amba_pv::amba_pv_extension` object must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
b_snoop(int socket_id,
         amba_pv::amba_pv_transaction & trans, sc_core::sctime & t)
```

`snoop_dbg()`

This optional master behavior implements an upstream snooping TLM debug transport interface. An `amba_pv::amba_pv_extension` object must be added to the transaction before calling this behavior. The `socket_id` parameter must be set to 0 in this context.

```
snoop_dbg(int socket_id,
        amba_pv::amba_pv_transaction & trans, sc_core::sctime & t):
        unsigned int
```

`invalidate_direct_mem_ptr()`

This optional master behavior is used to invalidate a DMI request. It invalidates DMI pointers that were previously established for the given DMI region. The `socket_id` parameter is 0 in this context.

```
invalidate_direct_mem_ptr(int socket_id,
        sc_dt::uint64 start_range, sc_dt::uint64 end_range)
```

For more information about the SystemC/TLM `amba_pv::amba_pv_extension` class, see the *AMBA-PV Extensions to OSCI TLM 2.0 Developer Guide.*

The contents of the generic payload data is formatted as an array of bytes in order of ascending bus address. This means that irrespective of the host machine endianness or modeled bus width, a little endian master should write the bytes of a word in increasing significance as the array index increases and a big endian master should write the bytes of a word in decreasing significance as the array index increases. A master or slave whose endianness does not match the endianness of the host machine must endian swap any access to the payload data that is wider than one byte. The same byte ordering rule applies to memory accesses using DMI pointers.

Special considerations for ACE and cache coherent interconnects

An ACE interconnect model must be able to cope with concurrent transactions in accordance with the hazard avoidance and prioritization rules in the ACE specification. Any external bus request, downstream transaction or upstream snoop transaction, can potentially cause a transaction to stall and the calling thread to be blocked, resulting in any number of other threads being scheduled.

To ensure that memory coherency is maintained, the following rules must be applied for debug transactions:

`debug reads` The bus must return data that represents the values that the bus master expects to observe if it issues a bus read. This must not modify the state of any bus components.

`debug writes` Must modify the contents of all copies of the location being accessed, so that a subsequent read from this location returns the data in the debug-write request. The debug write must not modify any other state, for example such as cache tags, clean/dirty/shared/unique MOESI state.

The implications for a coherent interconnect are that incoming debug transactions should be broadcast back upstream as debug snoop transactions to all ports other than the one the request came in on. Incoming debug snoops should propagate upwards. Debug reads can terminate as soon as they hit a cache. Debug writes must continue until they have propagated to all possible copies of the location, including downstream to main memory.

For cases where a debug transaction hazards with non-debug transactions that are in-flight, the debug transaction should observe a weak memory-order model. Particular care, however, must be taken by any component that can cause a thread to be blocked whilst that component is responsible for the payload of an in-flight transaction. In these cases the debug transaction must be hazarded against the in-flight payload to ensure that debug reads do not return stale data and debug writes do not cause cache incoherency.

DMI should only be used when it can be guaranteed that subsequent transactions do not result in any state transitions. This means that in the general case DMI should not be used for ACE coherent cacheable transactions.

AMBAPVSignal protocol

The AMBAPVSignal protocol defines a single behavior to permit masters to change the state changes of signals such as interrupt. Strictly speaking this behavior is not covered by AMBA3, but is provided with the AMBA-PV Components.

`set_state(int export_id, const bool &state): void`

Transfers a signal state. The `export_id` parameter must be set to 0 in this context.

AMBAPVSignalState protocol

The AMBAPVSignalState protocol defines two behaviors that permit a master to change the state of signals such as interrupt and to retrieve the state of such signals from slaves. Strictly speaking this behavior is not covered by AMBA3, but is provided with the AMBA-PV Components.

`set_state(int export_id, const bool &state): void`

Transfers a signal state. The `export_id` parameter must be set to 0 in this context.

`get_state(int export_id, tlm::tlm_tag<bool> * t): bool`

Retrieves a signal state. The `export_id` parameter must be set to 0, and the `t` parameter must be set to `NULL`, in this context.

AMBAPVValue protocol

The AMBAPVValue protocol models propagation of 32 bit integer values between components. Strictly speaking this behavior is not covered by AMBA3, but is provided with the AMBA-PV Components.

`set_state(int export_id, const uint32_t & value): void`

Transfers a value. The `export_id` parameter must be set to 0 in this context.

AMBAPVValue64 protocol

The AMBAPVValue64 protocol models propagation of 64 bit integer values between components. Strictly speaking this behavior is not covered by AMBA3, but is provided with the AMBA-PV Components.

`set_state(int export_id, const uint64_t & value): void`

Transfers a value. The `export_id` parameter must be set to 0 in this context.

AMBAPVValueState protocol

The AMBAPVValueState protocol defines two behaviors that permit propagation of 32 bit integer values between components and to retrieve such values from slaves. Strictly speaking this behavior is not covered by AMBA3, but is provided with the AMBA-PV Components.

`set_state(int export_id, const uint32_t & value):void`

Transfers a value. The `export_id` parameter must be set to 0 in this context.

`get_state(unsigned int export_id, tlm::tlm_tag<uint32_t> * t): uint32_t`

Retrieves a value. The `export_id` parameter must be set to 0, and the `t` parameter must be set to `NULL`, in this context.

AMBAPVValueState64 protocol

The AMBAPVValueState64 protocol defines two behaviors that permit propagation of 64 bit integer values between components and to retrieve such values from slaves. Strictly speaking this behavior is not covered by AMBA3, but is provided with the AMBA-PV Components.

`set_state(int export_id, const uint64_t & value): void`

Transfers a value. The `export_id` parameter must be set to 0 in this context.

`get_state(int export_id, tlm::tlm_tag<uint64_t> * t): uint64_t`

Retrieves a value. The `export_id` parameter must be set to 0, and the `t` parameter must be set to `NULL`, in this context.

5.3.3 PVBus2AMBAPV component

The PVBus2AMBAPV component converts from PVBus to AMBAPV protocols.

Figure 5-7 on page 5-25 shows a view of the component in System Canvas.

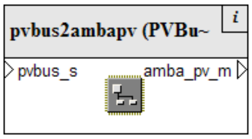


Figure 5-7 PVBus2AMBAPV in System Canvas

This component is written in LISA+.

Ports

Table 5-9 describes the ports in the PVBus2AMBAPV component.

Table 5-9 PVBus2AMBAPV ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Handles incoming transactions from PVBus masters.
amba_pv_m	AMBAPV	Master	Output master port for connection to top-level AMBAPV master port. Converted transactions are sent out through this port.

Additional protocols

The PVBus2AMBAPV component has no additional protocols.

Parameters

Table 5-10 provides a description of the configuration parameters for the PVBus2AMBAPV component.

Table 5-10 PVBus2AMBAPV configuration parameters

Parameter name	Description	Type	Allowed value	Default value
size	Addressable size of the device in bytes. Must be a multiple of 0x1000 (4KB).	uint64_t	0 to 4GB	`0x100000000` (4GB)

Registers

The PVBus2AMBAPV component has no registers.

Debug features

The PVBus2AMBAPV component has no debug features.

Verification and testing

The PVBus2AMBAPV component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

There is typically no significant performance impact because of using the PVBus2AMBAPV component.

Library dependencies

The PVBus2AMBAPV component has no dependencies on external libraries.

5.3.4 AMBAPV2PVBus component

The AMBAPV2PVBus component converts from AMBAPV to PVBus protocols.

Figure 5-8 shows a view of the component in System Canvas.

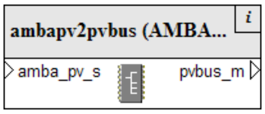


Figure 5-8 AMBAPV2PVBus in System Canvas

This component is written in LISA+.

Ports

Table 5-11 provides a description of the ports in the AMBAPV2PVBus component.

Table 5-11 AMBAPV2PVBus component ports

Name	Port protocol	Type	Description
amba_pv_s	AMBAPV	Slave	Input slave port for connection from top-level AMBAPV slave port.
pvbus_m	PVBus	Master	Handles outgoing PVBus transactions. Converted transactions are sent out through this port.

Additional protocols

The AMBAPV2PVBus component has no additional protocols.

Parameters

Table 5-12 describes the configuration parameters for the AMBAPV2PVBus component.

Table 5-12 AMBAPV2PVBus configuration parameters

Parameter name	Description	Type	Allowed value	Default value
base_addr	Base address of the component. Defines an offset to be added to the address of outgoing transactions.	uint64_t	-	0

Registers

The AMBAPV2PVBus component has no registers.

Debug features

The AMBAPV2PVBus component has no debug features.

Verification and testing

The AMBAPV2PVBus component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPV2PVBus component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPV2PVBus component has no dependencies on external libraries.

5.3.5 PVBus2AMBAPVACE component

The PVBus2AMBAPVACE component converts from PVBus to AMBAPVACE protocols.

Figure 5-9 shows a view of the component in System Canvas.

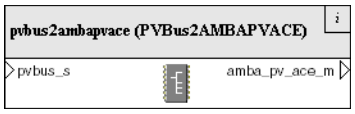


Figure 5-9 PVBus2AMBAPVACE in System Canvas

This component is written in LISA+.

Ports

Table 5-13 describes the ports in the PVBus2AMBAPVACE component.

Table 5-13 PVBus2AMBAPVACE ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Handles incoming transactions from PVBus masters. Converted upstream ACE snoop and DVM transactions are sent out through this port.
amba_pv_ace_m	AMBAPVACE	Master	Master port for connection to top-level AMBAPVACE master port. Converted transactions are sent out through this port. Handles incoming ACE snoop and DVM transactions from AMBA-PV ACE slaves.

Additional protocols

The PVBus2AMBAPVACE component has no additional protocols.

Parameters

Table 5-14 provides a description of the configuration parameters for the PVBus2AMBAP component.

Table 5-14 PVBus2AMBAPVACE configuration parameters

Parameter name	Description	Type	Allowed value	Default value
size	Addressable size of the device in bytes	uint64_t	Must be a multiple of `0x1000` (4KB)	`0x1000000000000`

Registers

The PVBus2AMBAPVACE component has no registers.

Debug features

The PVBus2AMBAPVACE component supports debug bus transactions but has no specific debug features.

Verification and testing

The PVBus2AMBAPVACE component has been tested using system level tests that included booting Linux on a bigLITTLE VE type platform.

Performance

The translation of bus transactions by the bridge has some impact on performance. Bus masters that cache memory transactions avoid much of this impact.

Library dependencies

The PVBus2AMBAPVACEcomponent is dependent on the AMBA-PV API which must be at least version 1.4.

5.3.6 AMBAPVACE2PVBus component

The AMBAPVACE2PVBus component converts from AMBAPVACE to PVBus protocols.

Figure 5-10 shows a view of the component in System Canvas.

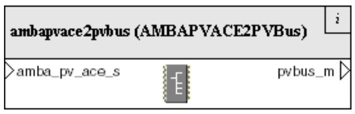


Figure 5-10 AMBAPV2PVBus in System Canvas

This component is written in LISA+.

Ports

Table 5-15 describes the ports in the AMBAPVACE2PVBus component.

Table 5-15 PVBus2AMBAPV ports

Name	Port protocol	Type	Description
amba_pv_ace_s	AMBAPVACE	Slave	Slave port for connection from top-level AMBAPVACE slave port. Handles incoming transactions from AMBA-PV ACE masters. Converted upstream ACE snoop and DVM transactions are sent out through this port.
pvbus_m	PVBus	Master	Converted downstream transactions are sent out though this port. Handles incoming ACE snoop and DVM transactions from PVBus slaves.

Additional protocols

The AMBAPVACE2PVBus component has no additional protocols.

Parameters

The AMBAPVACE2PVBus component has no parameters.

Registers

The AMBAPVACE2PVBus component has no registers.

Debug features

The AMBAPVACE2PVBus component supports debug bus transactions but has no specific debug features.

Verification and testing

The AMBAPVACE2PVBus component has been tested using system level tests that included booting Linux on a bigLITTLE VE type platform.

Performance

The translation of bus transactions by the bridge has some impact on performance. Bus masters that cache memory transactions avoid much of this impact. The bridge does not support DMI.

Library dependencies

The AMBAPVACE2PVBus component is dependent on the AMBA-PV API which must be at least version 1.4.

5.3.7 SGSignal2AMBAPVSignal component

The SGSignal2AMBAPVSignal component converts from Signal to AMBAPVSignal protocols.

Figure 5-11 shows a view of the component in System Canvas.

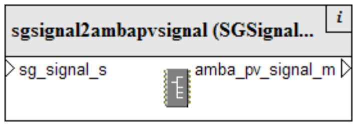


Figure 5-11 SGSignal2AMBAPVSignal in System Canvas

This component is written in LISA+.

Ports

Table 5-16 describes the SGSignal2AMBAPVSignal ports.

Table 5-16 SGSignal2AMBAPVSignal ports

Name	Port protocol	Type	Description
sg_signal_s	Signal	Slave	Handles incoming signal state changes.
amba_pv_signal_m	AMBAPVSignal	Master	Output master port for connection to top-level AMBAPVSignal master port. Converted signal state changes are sent out through this port.

Additional protocols

The SGSignal2AMBAPVSignal component has no additional protocols.

Parameters

The SGSignal2AMBAPVSignal component has no configuration parameters.

Registers

The SGSignal2AMBAPVSignal component has no registers.

Debug features

The SGSignal2AMBAPVSignal component has no debug features.

Verification and testing

The SGSignal2AMBAPVSignal component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The SGSignal2AMBAPVSignal component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SGSignal2AMBAPVSignal component has no dependencies on external libraries.

5.3.8 AMBAPVSignal2SGSignal component

The AMBAPVSignal2SGSignal component converts from AMBAPVSignal to Signal protocols.

Figure 5-12 shows a view of the component in System Canvas.

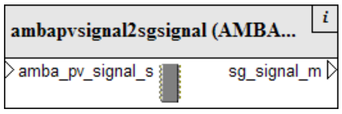


Figure 5-12 AMBAPVSignal2SGSignal in System Canvas

This component is written in LISA+.

Ports

Table 5-17 provides a description of the ports in the AMBAPVSignal2SGSignal component.

Table 5-17 AMBAPVSignal2SGSignal component ports

Name	Port protocol	Type	Description
amba_pv_signal_s	AMBAPVSignal	Slave	Input slave port for connection from top-level AMBAPVSignal slave port.
sg_signal_m	Signal	Master	Handles outgoing signal state changes. Converted signal state changes are sent out through this port.

Additional protocols

The AMBAPVSignal2SGSignal component has no additional protocols.

Parameters

The AMBAPVSignal2SGSignal component has no configuration parameters.

Registers

The AMBAPVSignal2SGSignal component has no registers.

Debug features

The AMBAPVSignal2SGSignal component has no debug features.

Verification and testing

The AMBAPVSignal2SGSignal component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPVSignal2SGSignal component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPVSignal2SGSignal component has no dependencies on external libraries.

5.3.9 SGStateSignal2AMBAPVSignalState component

The SGStateSignal2AMBAPVSignalState component converts from StateSignal to AMBAPVSignalState protocols.

Figure 5-13 shows a view of the component in System Canvas.

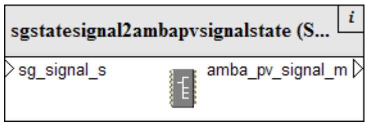


Figure 5-13 SGStateSignal2AMBAPVSignalState in System Canvas

This component is written in LISA+.

Ports

Table 5-18 describes the SGStateSignal2AMBAPVSignalState ports.

Table 5-18 SGStateSignal2AMBAPVSignalState ports

Name	Port protocol	Type	Description
sg_signal_s	Signal	Slave	Handles incoming signal state changes.
amba_pv_signal_m	AMBAPVSignal State	Master	Output master port for connection to top-level AMBAPVSignal master port. Converted signal state changes are sent out through this port.

Additional protocols

The SGStateSignal2AMBAPVSignalState component has no additional protocols.

Parameters

The SGStateSignal2AMBAPVSignalState component has no configuration parameters.

Registers

The SGStateSignal2AMBAPVSignalState component has no registers.

Debug features

The SGStateSignal2AMBAPVSignalState component has no debug features.

Verification and testing

The SGStateSignal2AMBAPVSignalState component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The SGStateSignal2AMBAPVSignalState component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SGStateSignal2AMBAPVSignalState component has no dependencies on external libraries.

5.3.10 AMBAPVSignalState2SGStateSignal component

The AMBAPVSignalState2SGStateSignal component converts from AMBAPVSignalState to StateSignal protocols.

Figure 5-14 shows a view of the component in System Canvas.

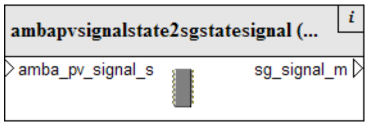


Figure 5-14 AMBAPVSignalState2SGStateSignal in System Canvas

This component is written in LISA+.

Ports

Table 5-19 lists the ports in the AMBAPVSignalState2SGStateSignal component.

Table 5-19 AMBAPVSignalState2SGStateSignal component ports

Name	Port protocol	Type	Description
amba_pv_signal_s	AMBAPVSignal State	Slave	Input slave port for connection from top-level AMBAPVSignal slave port.
sg_signal_m	StateSignal	Master	Handles outgoing signal state changes. Converted signal state changes are sent out through this port.

Additional protocols

The AMBAPVSignalState2SGStateSignal component has no additional protocols.

Parameters

The AMBAPVSignalState2SGStateSignal component has no configuration parameters.

Registers

The AMBAPVSignalState2SGStateSignal component has no registers.

Debug features

The AMBAPVSignalState2SGStateSignal component has no debug features.

Verification and testing

The AMBAPVSignalState2SGStateSignal component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPVSignalState2SGStateSignal component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPVSignalState2SGStateSignal component has no dependencies on external libraries.

5.3.11 SGValue2AMBAPVValue component

The SGValue2AMBAPVValue component converts from Value to AMBAPVValue protocols, using 32 bit integer values between components.

Figure 5-15 shows a view of the component in System Canvas.

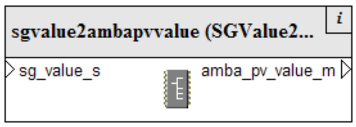


Figure 5-15 SGValue2AMBAPVValue in System Canvas

This component is written in LISA+.

Ports

Table 5-20 lists the SGValue2AMBAPVValue ports.

Table 5-20 SGValue2AMBAPVValue ports

Name	Port protocol	Type	Description
sg_value_s	Value	Slave	Handles incoming value changes.
amba_pv_value_m	AMBAPVValue	Master	Output master port for connection to top-level AMBAPVValue master port. Converted value changes are sent out through this port.

Additional protocols

The SGValue2AMBAPVValue component has no additional protocols.

Parameters

The SGValue2AMBAPVValue component has no configuration parameters.

Registers

The SGValue2AMBAPVValue component has no registers.

Debug features

The SGValue2AMBAPVValue component has no debug features.

Verification and testing

The SGValue2AMBAPVValue component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The SGValue2AMBAPVValue component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SGValue2AMBAPVValue component has no dependencies on external libraries.

5.3.12 SGValue2AMBAPVValue64 component

The SGValue2AMBAPVValue64 component converts from Value_64 to AMBAPVValue protocols. 64 bit integer values are used between components.

Figure 5-16 shows a view of the component in System Canvas.

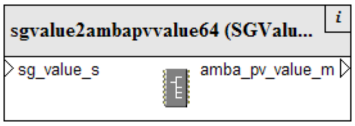


Figure 5-16 SGValue2AMBAPVValue64 in System Canvas

This component is written in LISA+.

Ports

Table 5-21 describes the SGValue2AMBAPVValue64 ports.

Table 5-21 SGValue2AMBAPVValue64 ports

Name	Port protocol	Type	Description
sg_value_s	Value_64	Slave	Handles incoming value changes.
amba_pv_value_m	AMBAPVValue64	Master	Output master port for connection to top-level AMBAPVValue64 master port. Converted value changes are sent out through this port.

Additional protocols

The SGValue2AMBAPVValue64 component has no additional protocols.

Parameters

The SGValue2AMBAPVValue64 component has no configuration parameters.

Registers

The SGValue2AMBAPVValue64 component has no registers.

Debug features

The SGValue2AMBAPVValue64 component has no debug features.

Verification and testing

The SGValue2AMBAPVValue64 component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The SGValue2AMBAPVValue64 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SGValue2AMBAPVValue64 component has no dependencies on external libraries.

5.3.13 AMBAPVValue2SGValue component

The AMBAPVValue2SGValue component converts from AMBAPVValue to Value protocols, using 32 bit integer values between components.

Figure 5-17 shows a view of the component in System Canvas.

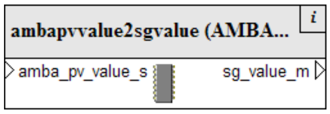


Figure 5-17 AMBAPVValue2SGValue in System Canvas

This component is written in LISA+.

Ports

Table 5-22 lists the ports in the AMBAPVValue2SGValue component.

Table 5-22 AMBAPVValue2SGValue component ports

Name	Port protocol	Type	Description
amba_pv_value_s	AMBAPVValue	Slave	Input slave port for connection from top-level AMBAPVValue slave port.
sg_value_m	Value	Master	Handles outgoing value changes. Converted value changes are sent out through this port.

Additional protocols

The AMBAPVValue2SGValue component has no additional protocols.

Parameters

The AMBAPVValue2SGValue component has no configuration parameters.

Registers

The AMBAPVValue2SGValue component has no registers.

Debug features

The AMBAPVValue2SGValue component has no debug features.

Verification and testing

The AMBAPVValue2SGValue component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPVValue2SGValue component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPVValue2SGValue component has no dependencies on external libraries.

5.3.14 AMBAPVValue2SGValue64 component

The AMBAPVValue2SGValue64 component converts from AMBAPVValue to Value_64 protocols. 64 bit integer values are used between components.

Figure 5-18 shows a view of the component in System Canvas.

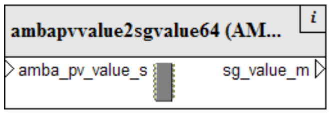


Figure 5-18 AMBAPVValue2SGValue64 in System Canvas

This component is written in LISA+.

Ports

Table 5-23 lists the ports in the AMBAPVValue2SGValue64 component.

Table 5-23 AMBAPVValue2SGValue64 component ports

Name	Port protocol	Type	Description
amba_pv_value_s	AMBAPVValue64	Slave	Input slave port for connection from top-level AMBAPVValue64 slave port.
sg_value_m	Value	Master	Handles outgoing value changes. Converted value changes are sent out through this port.

Additional protocols

The AMBAPVValue2SGValue64 component has no additional protocols.

Parameters

The AMBAPVValue2SGValue64 component has no configuration parameters.

Registers

The AMBAPVValue2SGValue64 component has no registers.

Debug features

The AMBAPVValue2SGValue64 component has no debug features.

Verification and testing

The AMBAPVValue2SGValue64 component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPVValue2SGValue64 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPVValue2SGValue64 component has no dependencies on external libraries.

5.3.15 SGValueState2AMBAPVValueState component

The SGValueState2AMBAPVValueState component converts from ValueState to AMBAPVValueState protocols. 32 bit integer values are used between components.

Figure 5-19 shows a view of the component in System Canvas.

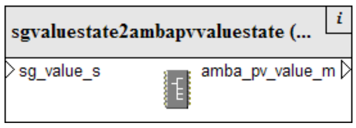


Figure 5-19 SGValueState2AMBAPVValueState in System Canvas

This component is written in LISA+.

Ports

Table 5-24 describes the SGValueState2AMBAPVValueState ports.

Table 5-24 SGValueState2AMBAPVValueState ports

Name	Port protocol	Type	Description
sg_value_s	Value_64	Slave	Handles incoming value changes.
amba_pv_value_m	AMBAPVValue64	Master	Output master port for connection to top-level AMBAPVValue64 master port. Converted value changes are sent out through this port.

Additional protocols

The SGValueState2AMBAPVValueState component has no additional protocols.

Parameters

The SGValueState2AMBAPVValueState component has no configuration parameters.

Registers

The SGValueState2AMBAPVValueState component has no registers.

Debug features

The SGValueState2AMBAPVValueState component has no debug features.

Verification and testing

The SGValueState2AMBAPVValueState component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The SGValueState2AMBAPVValueState component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SGValueState2AMBAPVValueState component has no dependencies on external libraries.

5.3.16 SGValueState2AMBAPVValueState64 component

The SGValueState2AMBAPVValueState64 component converts from ValueState_64 to AMBAPVValueState protocols, using 64 bit integer values between components.

Figure 5-20 on page 5-40 shows a view of the component in System Canvas.

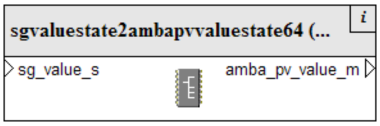


Figure 5-20 SGValueState2AMBAPVValueState64 in System Canvas

This component is written in LISA+.

Ports

Table 5-25 describes the SGValueState2AMBAPVValueState64 ports.

Table 5-25 SGValueState2AMBAPVValueState64 ports

Name	Port protocol	Type	Description
sg_value_s	Value_64	Slave	Handles incoming value changes.
amba_pv_value_m	AMBAPVValue64	Master	Output master port for connection to top-level AMBAPVValue64 master port. Converted value changes are sent out through this port.

Additional protocols

The SGValueState2AMBAPVValueState64 component has no additional protocols.

Parameters

The SGValueState2AMBAPVValueState64 component has no configuration parameters.

Registers

The SGValueState2AMBAPVValueState64 component has no registers.

Debug features

The SGValueState2AMBAPVValueState64 component has no debug features.

Verification and testing

The SGValueState2AMBAPVValueState64 component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The SGValueState2AMBAPVValueState64 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SGValueState2AMBAPVValueState64 component has no dependencies on external libraries.

5.3.17 AMBAPVValueState2SGValueState component

The AMBAPVValueState2SGValueState component converts from AMBAPVValueState to ValueState protocols, using 32 bit integer values between components.

Figure 5-21 shows a view of the component in System Canvas.

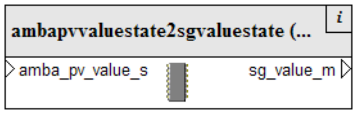


Figure 5-21 AMBAPVValue2SGValue in System Canvas

This component is written in LISA+.

Ports

Table 5-26 provides a description of the ports in the AMBAPVValueState2SGValueState component.

Table 5-26 AMBAPVValueState2SGValueState component ports

Name	Port protocol	Type	Description
amba_pv_value_s	AMBAPVValue64	Slave	Input slave port for connection from top-level AMBAPVValue64 slave port.
sg_value_m	Value_64	Master	Handles outgoing value changes. Converted value changes are sent out through this port.

Additional protocols

The AMBAPVValueState2SGValueState component has no additional protocols.

Parameters

The AMBAPVValueState2SGValueState component has no configuration parameters.

Registers

The AMBAPVValueState2SGValueState component has no registers.

Debug features

The AMBAPVValueState2SGValueState component has no debug features.

Verification and testing

The AMBAPVValueState2SGValueState component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPVValueState2SGValueState component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPVValueState2SGValueState component has no dependencies on external libraries.

5.3.18 AMBAPVValueState2SGValueState64 component

The AMBAPVValueState2SGValueState64 component converts from AMBAPVValueState64 to ValueState_64 protocols, using 64 bit integer values between components.

Figure 5-22 shows a view of the component in System Canvas.

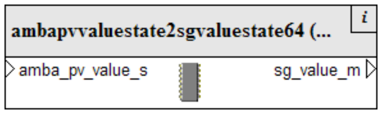


Figure 5-22 AMBAPVValue2SGValue64 in System Canvas

This component is written in LISA+.

Ports

Table 5-27 provides a description of the ports in the AMBAPVValueState2SGValueState64 component.

Table 5-27 AMBAPVValueState2SGValueState64 component ports

Name	Port protocol	Type	Description
amba_pv_value_s	AMBAPVValue64	Slave	Input slave port for connection from top-level AMBAPVValue64 slave port.
sg_value_m	Value_64	Master	Handles outgoing value changes. Converted value changes are sent out through this port.

Additional protocols

The AMBAPVValueState2SGValueState64 component has no additional protocols.

Parameters

The AMBAPVValueState2SGValueState64 component has no configuration parameters.

Registers

The AMBAPVValueState2SGValueState64 component has no registers.

Debug features

The AMBAPVValueState2SGValueState64 component has no debug features.

Verification and testing

The AMBAPVValueState2SGValueState64 component has been tested as part of the SystemC Export example systems. These systems can be found in `%PVLIB_HOME%\examples\SystemCExport`. On Linux, look in the `$PVLIB_HOME/examples/SystemCExport` directory.

Performance

The AMBAPVValueState2SGValueState64 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AMBAPVValueState2SGValueState64 component has no dependencies on external libraries.

5.4 Example Peripheral Components

This section describes generic examples of peripheral components included with Fast Models. It contains the following sections:

- *HostBridge component* on page 5-158
- *VirtualEthernetCrossover component* on page 5-159
- *CCI400 component* on page 5-160
- *GIC400 component* on page 5-163
- *MemoryMappedCounterModule component* on page 5-168.

5.4.1 About the Example Peripheral Components Model Library

The Example Peripheral Components Model Library is a component of Fast Models. Generic peripheral components as implemented in Fast Models are described in this section. Ports, functionality, parameters and registers for the model components are provided.

These example peripheral components are provided to demonstrate how to model different types of components, and to provide a set of examples that permits the development of example platforms. They might not model all aspects of a component, and they might be only partially validated.

For more information about the functionality of the hardware that the model simulates, see the relevant technical reference manual.

5.4.2 PL011_Uart component

The PL011_Uart component provides a programmer's view model of the PL011_UART PrimeCell component as described in its technical reference manual. See *PrimeCell UART (PL011) r1p4 Technical Reference Manual*.

Note

The PL011_Uart component does not implement the DMA functionality of the PL011 PrimeCell.

Figure 5-23 shows a view of the component in System Canvas.

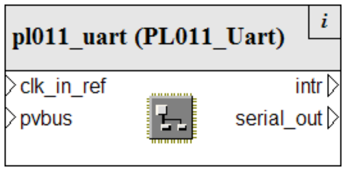


Figure 5-23 PL011_Uart in System Canvas

This component is written in LISA+.

Ports

Table 5-28 provides a brief description of the PL011_Uart component ports. For more information, see the PL011 technical reference manual.

Table 5-28 PL011_Uart ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Interrupt signaling.
clk_in_ref	ClockSignal	Slave	Clock input, typically 14.745MHz, which sets the master transmit/receive rate.
serial_out	SerialData	Master	Used to communicate with a serial device, such as a terminal.

SerialData protocol for PL011_Uart

The PL011_Uart component has one additional protocol, SerialData. The serial data protocol is implemented as a parallel interface for efficiency. All communication is driven by the master port.

`dataTransmit(uint16_t data) : void`

Used by the master to send data to the slave. See Table 5-29 for bit definitions.

Table 5-29 Bits for dataTransmit()

Bits	Function
15:8	reserved
7:0	transmit data

`dataReceive(void) : uint16_t`

Used by the master to receive data from the slave. See Table 5-30 for bit definitions.

Table 5-30 Bits for dataReceive()

Bits	Function
15:13	reserved
12	set when no data available for reading
11	reserved
10	break error
9:8	reserved
7:0	receive data

`signalsSet(uint8_t signal) : void`

Used by the master to get the current signal status. See Table 5-31 for bit definitions.

Table 5-31 Bits for signalsSet()

Bits	Function
7	Out1
6	Out2
5	RTS
4	DTR
3:0	reserved

`signalsGet() : uint8_t`

Used by the master to get the current signal status. See Table 5-32 for bit definitions.

Table 5-32 Bits for signalsGet()

Bits	Function
7:4	Reserved
3	DCD
2	DSR
1	CTS
0	RI

Parameters

Table 5-33 lists the parameters used by the PL011_Uart component.

Table 5-33 PL011_Uart configuration parameters

Parameter name	Description	Type	Allowed value	Default value
baud_rate	Baud rate	Integer	-	38400
clock_rate	Clock rate for PL011	Integer	-	14745600
in_file	Input file	String	-	[empty string]
in_file_escape_sequence	Input file escape sequence	String	-	##
out_file	Output file	String	-	[empty string]
shutdown_on_eot	Shutdown simulation when an EOT (ASCII 4) character is transmitted.	Boolean	true/false	false

Table 5-33 PL011_Uart configuration parameters (continued)

Parameter name	Description	Type	Allowed value	Default value
uart_enable	Enables UART when the system starts	Boolean	true/false	false
unbuffered_output	Unbuffered output	Boolean	true/false	false
untimed_fifos[a]	Controls the rate at which serial data is transferred between the Tx/Rx FIFOs of the UART and the SerialData port.	Boolean	true/false	false

a. When false, characters of serial data are clocked to/from the SerialData port at a rate controlled by the clk_in_ref clock rate and the baud-rate-divider configuration of the UART clock. Enabling untimed_fifos permits serial data to be sent/received as fast as it can be generated/consumed. The modem control signals are still generated correctly, so the UART is not able to transmit data faster than the receiving end can handle. (For example, TelnetTerminal uses the CTS signal to avoid overflowing its TCP/IP buffer).

Registers

Table 5-34 gives a description of the configuration registers for the PL011_Uart component.

Table 5-34 PL011_Uart registers

Register name	Offset	Access	Description
UARTDR	`0x00`	read/write	Data register.
UARTRSR	`0x04`	read only	Receive status register.
UARTECR	`0x04`	write only	Error clear register.
UARTFR	`0x18`	read only	Flag register.
UARTILPR	`0x20`	read/write	IrDA low-power counter[a].
UARTIBRD	`0x24`	read/write	Integer baud rate divisor.
UARTFBRD	`0x28`	read/write	Fractional baud rate divisor.
UARTLCR_H	`0x2C`	read/write	Line control register, high byte.
UARTCR	`0x30`	read/write	Control register.
UARTFLS	`0x34`	read/write	Interrupt FIFO level select.
UARTMSC	`0x38`	read/write	Interrupt mask set/clear.
UARTRIS	`0x3C`	read only	Raw interrupt status.
UARTMIS	`0x40`	read only	Masked interrupt register.
UARTICR	`0x44`	write only	Interrupt clear register.
UARTDMACR	`0x48`	read/write	DMA control register[a].

a. Not implemented.

Debug features

The PL011_Uart component has no debug features.

Verification and testing

The PL011_Uart component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PL011_Uart component is not expected to significantly affect the performance of a PV system. However, at very high baud rates such as in excess of 1MHz, simulation performance might be reduced.

Library dependencies

The PL011_Uart component has no dependencies on external libraries.

5.4.3 SerialCrossover component

The SerialCrossover component provides the functionality of a serial crossover cable. It implements two SerialData slave ports and permits two SerialData master ports, such as from PL011_Uart components, to be connected.

Data received on one port is buffered in a FIFO until it is read from the other port.

Signals received on one port are latched and available to be read by the other port.

Figure 5-24 shows a view of the component in System Canvas.

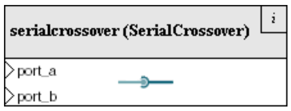


Figure 5-24 SerialCrossover in System Canvas

This component is written in C++.

Ports

Table 5-35 provides a brief description of the ports in the SerialCrossover component.

Table 5-35 SerialCrossover ports

Name	Port protocol	Type	Description
port_a	SerialData	Slave	Slave port for connecting to a SerialData master.
port_b	SerialData	Slave	Slave port for connecting to a SerialData master.

Additional protocols

The SerialCrossover component has no additional protocols.

Parameters

The SerialCrossover component has no parameters.

Registers

The SerialCrossover component has no registers.

Debug features

The SerialCrossover component has no debug features.

Verification and testing

The SerialCrossover component has been tested as part of the Dual Processor example system using the test suites and by booting operating systems.

Performance

The SerialCrossover component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SerialCrossover component has no dependencies on external libraries.

5.4.4 TelnetTerminal component

The TelnetTerminal component is a virtual component that permits UART data to be transferred between a SerialData port and a TCP/IP socket on the host.

When the simulation is started and the TelnetTerminal component is enabled, the component opens a server (listening) socket on a TCP/IP port. This is port 5000 by default.

Data written to the SerialData port is transmitted over the network socket.When data becomes available on the network socket, the TelnetTerminal component buffers the data. The data can then be read from SerialData.

If there is no connection to the network socket when the first data access is made, a host telnet session is automatically started. Prior to this first access, you can connect a client of your choice to the network socket. If the connection between the TelnetTerminal component and the client is broken at any time, the port is re-opened, permitting you to make another client connection.

Further information on how to use the TelnetTerminal component is provided later in this book. See *Terminal* on page 5-178.

Figure 5-25 shows a view of the component in System Canvas.

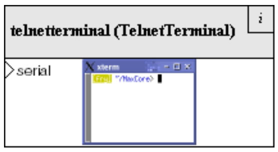


Figure 5-25 TelnetTerminal in System Canvas

This component is written in C++.

Ports

Table 5-36 provides a brief description of the TelnetTerminal component ports.

Table 5-36 TelnetTerminal ports

Name	Port protocol	Type	Description
serial	SerialData	Slave	Slave port for connecting to a SerialData master.

Additional protocols

The TelnetTerminal component has no additional protocols.

Parameters

Table 5-37 lists the parameters used by the TelnetTerminal component.

Table 5-37 TelnetTerminal configuration parameters

Parameter name	Description	Type	Allowed value	Default value
mode	Terminal operation mode.	String	telnet[a], raw[b]	telnet
start_telnet	Enable terminal when the system starts.	Boolean	true/false	true
start_port	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.	Integer	valid port number	5000

a. In telnet mode, the TelnetTerminal component supports a subset of the telnet protocol defined in RFC 854.

b. In raw mode, the TelnetTerminal component does not interpret or modify the byte stream contents. This permits you to make a debugger connection, for example, to connect a gdb client to a gdbserver running on the target operating system.

Registers

The TelnetTerminal component has no registers.

Debug features

The TelnetTerminal component has no debug features.

Verification and testing

The TelnetTerminal component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The TelnetTerminal component is not expected to significantly affect the performance of a PV system.

Library dependencies

The performance of the TelnetTerminal component depends on the host terminal.

5.4.5 PL022_SSP component

Note

The PL022_SSP component is a preliminary release. It is provided as-is with the VE reference platform model, and is not yet a fully supported peripheral.

The PL022_SSP component is a programmer's view model of the ARM PL022 *Synchronous Serial Port* (SSP) PrimeCell. Although the PL022_SSP component has clock input, it is not internally clock-driven. This is different to the equivalent hardware.

For more information, see the component documentation. See the *ARM PrimeCell Synchronous Serial Port (PL022) Technical Reference Manual*.

Figure 5-26 shows a view of the component in System Canvas.

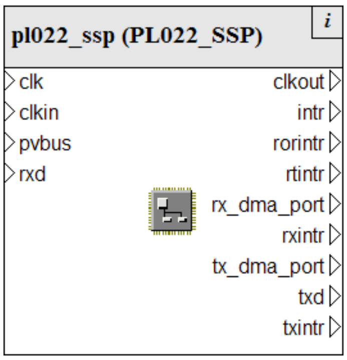


Figure 5-26 PL022_SSP in System Canvas

This component is written in LISA+.

Ports

Table 5-38 provides a brief description of the PL022_SSP ports. For more information, see the component documentation.

Table 5-38 PL022_SSP ports

Name	Port protocol	Type	Description
clk	ClockSignal	Slave	Main PrimeCell SSP clock input.
clkin	ClockSignal	Slave	PrimeCell SSP clock input.
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
rxd	ValueState	Slave	PrimeCell SSP receive data.

Table 5-38 PL022_SSP ports (continued)

Name	Port protocol	Type	Description
clkout	ClockSignal	Master	Clock output.
intr	Signal	Master	Interrupt signaling.
rorintr	Signal	Master	Receive overrun interrupt.
rtintr	Signal	Master	Receive timeout interrupt[a].
rx_dma_port	PL080_DMAC_DmaPortProtocol[b]	Master	PrimeCell SSP receive DMA port.
rxintr	Signal	Master	Receive FIFO service request port.
tx_dma_port	PL080_DMAC_DmaPortProtocol[b]	Master	PrimeCell SSP transmit DMA port.
txd	ValueState	Master	PrimeCell SSP transmit data.
txintr	Signal	Master	Transmit FIFO service request.

a. Not supported.

b. See *PL080_DMAC_DmaPortProtocol* on page 5-71.

Additional protocols

The PL022_SSP component has no additional protocols.

Parameters

The PL022_SSP component has no parameters.

Registers

Table 5-39 provides a description of the configuration registers for the PL022_SSP component.

Table 5-39 PL022_SSP registers

Register name	Offset	Access	Description
SSPCR0	`0x000`	read/write	Control register 0.
SSPCR1	`0x004`	read/write	Control register 1.
SSPDR	`0x008`	read/write	FIFO data.
SSPSR	`0x00C`	read only	Status.
SSPCPSR	`0x010`	read/write	Clock prescale.
SSPIMSC	`0x014`	read/write	Interrupt mask set/clear.
SSPRIS	`0x018`	read only	Raw interrupt status.
SSPMIS	`0x01C`	read only	Masked interrupt status.
SSPICR	`0x020`	write only	Interrupt clear.
SSPDMACR	`0x024`	read/write	DMA control.
SSPeriphID0	`0xFE0`	read only	Peripheral ID bits 7:0.

Table 5-39 PL022_SSP registers (continued)

Register name	Offset	Access	Description
SSPeriphID1	`0xFE4`	read only	Peripheral ID bits 15:8.
SSPeriphID2	`0xFE8`	read only	Peripheral ID bits 23:16.
SSPeriphID3	`0xFEC`	read only	Peripheral ID bits 31:24.
SSPPCellID0	`0xFF0`	read only	PrimeCell ID bits 7:0.
SSPPCellID01	`0xFF4`	read only	PrimeCell ID bits 15:8.
SSPPCellID	`0xFF8`	read only	PrimeCell ID bits 23:16.
SSPPCellID3	`0xFFC`	read only	PrimeCell ID bits 31:24.

Debug features

The PL022_SSP component has no debug features.

Verification and testing

The functions of the PL022_SSP component have been tested individually using a tailored test suite. The component has not been validated against a target operating system, but improved support is expected in the next release.

Performance

The PL022_SSP component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL022_SSP component has no dependencies on external libraries.

5.4.6 PL030_RTC component

The PL030_RTC component is a programmer's view model of the PL030_RTC PrimeCell. For a detailed description of the behavior of the PrimeCell, read the technical reference manual. See the *ARM PrimeCell Real Time Clock (PL030) Technical Reference Manual*.

Figure 5-27 shows a view of the component in System Canvas.

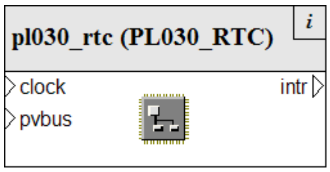


Figure 5-27 PL030_RTC in System Canvas

This component is written in LISA+.

Ports

Table 5-40 provides a brief description of the ports. For more information, see the PL030 documentation.

Table 5-40 PL030_RTC ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Interrupt signaling.
clock	ClockSignal	Slave	Clock input, typically 1MHz, driving master count rate.

Additional protocols

The PL030_RTC component has no additional protocols.

Parameters

The PL030_RTC component has no parameters.

Registers

Table 5-41 provides a description of the configuration registers for the PL030_RTC component. See the PL030 documentation for further details.

Table 5-41 PL030_RTC registers

Register name	Offset	Access	Description
RTCDR	`0x00`	read only	Data register.
RTCMR	`0x04`	read/write	Match register.
RTCSTAT	`0x08`	read only	Interrupt status register.
RTCEOI	`0x08`	write only	Interrupt clear register.
RTCLR	`0x0C`	read/write	Counter load register.
RTCCR	`0x10`	read/write	Counter register.

Debug features

The PL030_RTC component has no debug features.

Verification and testing

The PL030_RTC component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PL030_RTC component has no impact on the performance of a PV system when idle or counting down. The component only executes code when the counter expires or during bus accesses.

Library dependencies

The PL030_RTC component has no dependencies on external libraries.

5.4.7 PL031_RTC component

The PL031_RTC component is a programmer's view model of the PL031 Real Time Clock. It can be used to provide a basic alarm function or long time base counter. For a detailed description of the behavior of the PL031_RTC, read the component documentation. See the *ARM PrimeCell Real Time Clock (PL031) Technical Reference Manual*.

Figure 5-28 shows a view of the component in System Canvas.

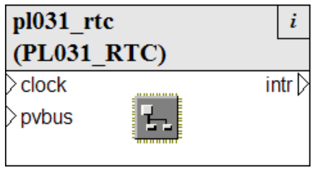


Figure 5-28 PL031_RTC in System Canvas

This component is written in LISA+.

Ports

Table 5-42 provides a brief description of the ports. See the PL031 RTC documentation for further details.

Table 5-42 PL031_RTC ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Interrupt signaling.
clock	ClockSignal	Slave	Clock input, typically 1MHz, driving master count rate.

Additional protocols

The PL031_RTC component has no additional protocols.

Parameters

The PL031_RTC component has no parameters.

Registers

Table 5-43 provides a description of the configuration registers for the PL031_RTC component.

Table 5-43 PL031_RTC registers

Register name	Offset	Access	Description
RTCDR	`0x000`	read only	Data register.
RTCMR	`0x004`	read/write	Match register.
RTCLR	`0x008`	read/write	Load register.
RTCCR	`0x00C`	read/write	Control register.
RTCIMSC	`0x010`	read/write	Interrupt mask set and clear register.
RTCRIS	`0x014`	read only	Raw interrupt status register.
RTCMIS	`0x018`	read only	Masked interrupt status register.
RTCIRC	`0x01C`	write only	Interrupt clear register[a].
RTCPeriphID0	`0xFE0`	read only	Peripheral ID register[a].
RTCPeriphID1	`0xFE4`	read only	Peripheral ID register[a].
RTCPeriphID2	`0xFE8`	read only	Peripheral ID register[a].
RTCPeriphID3	`0xFEC`	read only	Peripheral ID register[a].
RTCPCellID0	`0xFF0`	read only	PrimeCell ID register[a].
RTCPCellID1	`0xFF4`	read only	PrimeCell ID register[a].
RTCPCellID2	`0xFF8`	read only	PrimeCell ID register[a].
RTCPCellID3	`0xFFC`	read only	PrimeCell ID register[a].

a. This register has no CADI interface.

Debug features

The PL031_RTC component has no debug features.

Verification and testing

The PL031_RTC component has been tested as part of the VE example system using VE test suites and by booting operating systems.

The PL031_RTC component has been tested as part of the SMLT component.

Performance

The PL031_RTC component has no impact on the performance of a PV system when idle or counting down. The component only executes code when the counter expires or during bus accesses.

Library dependencies

The PL031_RTC component has no dependencies on external libraries.

5.4.8 PL041_AACI component

The PL041_AACI component is a programmer's view model of the PL041 *Advanced Audio CODEC Interface* (AACI). This component also contains a minimal register model of the LM4529 secondary codec as implemented on development boards supplied by ARM. For more information, see the component documentation. See the *ARM PrimeCell Advanced Audio CODEC Interface (PL041) Technical Reference Manual*.

The PL041_AACI component is not a complete implementation of the AACI because the following functionality is not currently implemented:

- audio input
- DMA access to FIFOs, rather than Programmed I/O
- programming of the secondary codec through FIFOs rather than slot registers.

The PL041_AACI component is designed to connect to an audio output component such as AudioOutFile or AudioOut_SDL. See *AudioOut_File component* on page 5-60. Also see *AudioOut_SDL component* on page 5-62.

Figure 5-29 shows a view of the component in System Canvas.

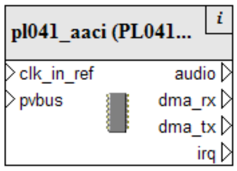


Figure 5-29 PL041_AACI in System Canvas

This component is written in LISA+.

Ports

Table 5-44 provides a brief description of the ports on the PL041_AACI component. For more information, see the component documentation.

Table 5-44 PL041_AACI ports

Name	Port protocol	Type	Description
clk_ref_in	ClockSignal	Slave	Reference clock input, typically 25MHz.
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
audio	AudioControl	Master	Used to communicate with an audio out device.
irq	Signal	Master	Single IRQ output port.

AudioControl protocol

The PL041_AACI component has one additional protocol.

The AudioControl protocol has the following behaviors:

`getPVAudioBuffer`

get an underlying host buffer for audio output

`releasePVAudioBuffer`

release an underlying host buffer.

Parameters

The PL041_AACI component has no parameters.

Registers

Table 5-45 provides a description of the registers for the PL041_AACI component.

Table 5-45 PL041_AACI registers

Register name	Offset	Access	Description
RXCR1	`0x00`	read/write	FIFO1 receive control
TXCR1	`0x04`	read/write	FIFO1 transmit control
SR1	`0x08`	read/write	Channel 1 status
ISR1	`0x0C`	read/write	Channel 1 interrupt status
IE1	`0x10`	read/write	Channel 1 interrupt enable
RXCR2	`0x14`	read/write	FIFO2 receive control
TXCR2	`0x18`	read/write	FIFO2 transmit control
SR2	`0x1C`	read/write	Channel 2 status
ISR2	`0x20`	read/write	Channel 2 interrupt status
IE2	`0x24`	read/write	Channel 2 interrupt enable
RXCR3	`0x28`	read/write	FIFO3 receive control
TXCR3	`0x2C`	read/write	FIFO3 transmit control
SR3	`0x30`	read/write	Channel 3 status
ISR3	`0x34`	read/write	Channel 3 interrupt status
IE3	`0x38`	read/write	Channel 3 interrupt enable
RXCR4	`0x3C`	read/write	FIFO4 receive control
TXCR4	`0x40`	read/write	FIFO4 transmit control
SR4	`0x44`	read/write	Channel 4 status
ISR4	`0x48`	read/write	Channel 4 interrupt status
IE4	`0x4C`	read/write	Channel 4 interrupt enable
SL1RX	`0x50`	read/write	Slot 1 receive data
SL1TX	`0x54`	read/write	Slot 1 transmit data

Table 5-45 PL041_AACI registers (continued)

Register name	Offset	Access	Description
SL2RX	`0x58`	read/write	Slot 2 receive data
SL2TX	`0x5C`	read/write	Slot 2 transmit data
SL12RX	`0x60`	read/write	Slot 12 receive data
SL12TX	`0x64`	read/write	Slot 12 transmit data
LSFR	`0x68`	read/write	Slot flag register
SLISTAT	`0x6C`	read/write	Slot interrupt status
SLIEN	`0x70`	read/write	Slot interrupt enable
ALLINTCLR	`0x74`	write only	All interrupts clear
MAINCR	`0x78`	read/write	Main control
RESET	`0x7C`	read/write	Reset control
SYNC	`0x80`	read/write	Sync control
ALLINTS	`0x84`	read/write	All FIFO interrupts status
MAINFR	`0x88`	read/write	Main flags register

Debug features

The PL041_AACI component has no debug features.

Verification and testing

The PL041_AACI component has been tested using the ALSA driver for this component under Linux.

Performance

The PL041_AACI component relies on a timed callback from the simulation so might have a small impact on simulation performance. The ability to play audio through this component depends on the AudioOut Component in use and on the performance requirements of the software running on the simulated system. The rate of FIFO draining is controlled by the audio output to which the component is connected. This might not correspond to the rate that would be expected from the reference clock.

Library dependencies

The PL041_AACI component has no dependencies on external libraries.

5.4.9 AudioOut_File component

The AudioOut_File component implements and audio output suitable for use with the PL041_AACI component. It writes raw 16 bit 48KHz stereo audio data to a user specified file.

Figure 5-30 on page 5-61 shows a view of the component in System Canvas.

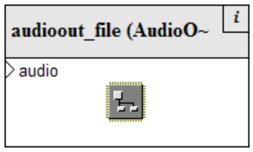


Figure 5-30 AudioOut in System Canvas

This component is written in LISA+.

Ports

Table 5-46 provides a brief description of the ports for the AudioOut_File component.

Table 5-46 AudioOut_File ports

Name	Port protocol	Type	Description
audio	AudioControl	Slave	Audio input for a connection to a component such as the PL041_AACI.

Additional protocols

The AudioControl protocol is described elsewhere in this document. See *AudioControl protocol* on page 5-58.

Parameters

Table 5-47 provides a description of the configuration parameters for the AudioOut_File component.

Table 5-47 AudioOut_File configuration parameters

Parameter name	Description	Type	Allowed value	Default value
fname	Filename for output.	String	legal filename	[empty string]

Registers

The AudioOut_File component has no registers.

Debug features

The AudioOut_File component has no debug features.

Verification and testing

The AudioOut_File component has been tested as part of a system containing a PL041.

Performance

The AudioOut_File component is not expected to significantly affect the performance of a PV system. AudioOut_File drains audio data at the rate that would be expected by software running in the simulation.

Library dependencies

The AudioOut_File component has no dependencies on external libraries.

5.4.10 AudioOut_SDL component

The AudioOut_SDL component outputs audio using the host audio output features through the Simple Direct Media abstraction layer.

Figure 5-31 shows a view of the component in System Canvas.

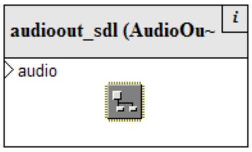


Figure 5-31 AudioOut_SDL in System Canvas

This component is written in LISA+ but relies on an external C++ class.

Ports

Table 5-48 provides a brief description of the AudioOut_SDL component ports.

Table 5-48 AudioOut_SDL ports

Name	Port protocol	Type	Description
audio	AudioControl	Slave	Audio input for a connection to a component such as the PL041_AACI.

Additional protocols

The AudioControl protocol is described elsewhere in this document. See *AudioControl protocol* on page 5-58.

Parameters

The AudioOut_SDL component has no parameters.

Registers

The AudioOut_SDL component has no registers.

Debug features

The AudioOut_SDL component has no debug features.

Verification and testing

The AudioOut_SDL component has been tested as part of a system containing a PL041.

Performance

The AudioOut_SDL component results in SDL audio callbacks and might have a small impact on PV systems containing the component. AudioOut_SDL attempts to drain audio data at whatever rate is required to maintain smooth sound playback on the host PC. This might not match the data rate expected by applications running on the simulation.

Library dependencies

The AudioOut_SDL component depends on the Simple DirectMedia library by C++ class.

5.4.11 PL050_KMI_component

The PL050_KMI component is a programmer's view model of the PL050 Keyboard/Mouse Interface PrimeCell as described in the *ARM PrimeCell PS2 Keyboard/Mouse Interface (PL050) Technical Reference Manual*.

The component is designed to communicate with models of PS/2-like devices such as a PS2Keyboard or PS2Mouse.

Figure 5-32 shows a view of the component in System Canvas.

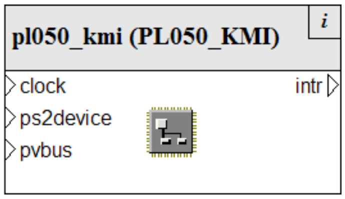


Figure 5-32 Keyboard/Mouse controller in System Canvas

This component is written in LISA+.

Ports

Table 5-49 provides a brief description of the ports. See *the ARM PrimeCell PS2 Keyboard/Mouse Interface (PL050) Technical Reference Manual*.

Table 5-49 PL050_KMI ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Master port signalling completion of transmit or receive.
clock	ClockSignal	Slave	Clock input, typically 1MHz, which sets the master transmit/receive rate.
ps2device	PS2Data	Slave	Used to communicate with a PS/2-like device.

PS2Data

The PL050_KMI component has one additional protocol.

The PS2Data protocol is used for communication between the KMI and a PS/2-like device. For efficiency reasons the interface is implemented as a parallel byte interface rather than a serial clock/data interface. The behaviors are:

`setClockData(enum ps2clockdata) : void`

Used by the KMI to simulate forcing the state of the data/clock lines. to indicate whether it is able to receive data, wants to send a command, or is inhibiting communication.

`getData() : uint8`

Used by the PS/2 device to get command data from the KMI.

`putData(uint8 data) : void`

Used by the PS/2 device to send device data to the KMI.

Parameters

The PL050_KMI component has no parameters.

Registers

Table 5-50 provides a description of the configuration registers for the component.

Table 5-50 PL050_KMI registers

Register name	Offset	Access	Description
KMICR	`0x00`	read/write	Control register
KMISTAT	`0x04`	read only	Status register
KMIDATA	`0x08`	read/write	Data register
KMICLKDIV	`0x0C`	read/write	Internal clock divider
KMIR	`0x10`	read only	Interrupt status register
KMIPeriphID0	`0xfe0`	read only	Peripheral ID register
KMIPeriphID1	`0xfe4`	read only	Peripheral ID register
KMIPeriphID2	`0xfe8`	read only	Peripheral ID register
KMIPeriphID3	`0xfec`	read only	Peripheral ID register
KMIPCellID0	`0xff0`	read only	PrimeCell ID register
KMIPCellID1	`0xff4`	read only	PrimeCell ID register
KMIPCellID2	`0xff8`	read only	PrimeCell ID register
KMIPCellID3	`0xffc`	read only	PrimeCell ID register

Debug features

The PL050_KMI component has no debug features.

Verification and testing

The PL050_KMI component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PL050_KMI component is not expected to have a significant impact on performance of a PV system. However if it is connected to the Visualisation component, then the performance of the component is dependent on that of the visualization.

Library dependencies

The PL050_KMI component has no dependencies on external libraries.

5.4.12 PS2Keyboard component

The PS2Keyboard component is a virtual component that translates a stream of key press information into appropriate PS/2 serial data. The key press data stream must be provided from another component such as a visualization component.

Figure 5-33 shows a view of the component in System Canvas.

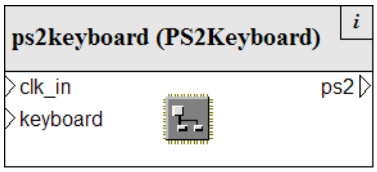


Figure 5-33 PS2Keyboard in System Canvas

This component is written in LISA+.

Ports

Table 5-51 provides a brief description of the ports in the PS2Keyboard component. For more information, see the technical reference manual for your hardware baseboard.

Table 5-51 PS2Keyboard ports

Name	Port protocol	Type	Description
keyboard	KeyboardStatus	Slave	Receives keyboard input from, for example, the Visualisation component.
clk_in	ClockSignal	Slave	Drives the PS/2 clocking rate, typically 1MHz.
ps2	PS2Data	Master	Connection to the PS/2 controller, for example, the PL050_KMI.

KeyboardStatus

The PS2Keyboard component has one additional protocol.

The KeyboardStatus protocol is used to pass keyboard events to a component such as the PS2Keyboard component. Events are only sent when the visualization window is in focus. Keyboard combinations which are filtered by the host OS such as **Ctrl+Alt+Del** are not detected by the visualization. See `components/KeyCode.h` for a list of ATKeyCode `code` values.

The protocol behaviors are:

`keyDown(ATKeyCode code) : void`

This is sent when a key on the host keyboard is pressed.

`keyUp(ATKeyCode code) : void`

This is sent when a key on the host keyboard is released.

Parameters

The PS2Keyboard component has no parameters.

Registers

The PS2Keyboard component has no registers.

Debug features

The PS2Keyboard component has no debug features.

Verification and testing

The PS2Keyboard component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PS2Keyboard component is not expected to significantly affect the performance of a PV system. However if it is connected to the Visualisation component, then the performance of the component is dependent on that of the visualization.

Library dependencies

The PS2Keyboard component has no dependencies on external libraries.

5.4.13 PS2Mouse component

The PS2Mouse component implements the PS/2 register interface of a PS/2 style mouse. The mouse movement and button press data must be provided from another component such as the Visualisation component.

Figure 5-34 shows a view of the component in System Canvas.

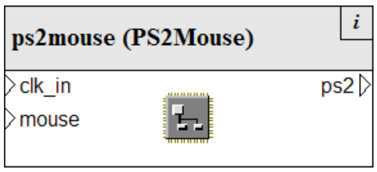


Figure 5-34 PS2Mouse component in System Canvas

This component is written in LISA+.

Ports

Table 5-52 provides a brief description of the ports in the PS2Mouse component. See the *RealView Emulation Baseboard User Guide (Lead Free)*.

Table 5-52 PS2Mouse ports

Name	Port protocol	Type	Description
mouse	MouseStatus[a]	Slave	Receives keyboard input from, for example, the Visualisation component.
clk_in	ClockSignal	Slave	Drives the PS/2 clocking rate, typically 1MHz.
ps2	PS2Data	Master	Connection to the PS/2 controller, for example the PL050_KMI.

a. The MouseStatus protocol is documented in *Visualisation Library* on page 5-170.

MouseStatus

The PS2Mouse component has one additional protocol.

The MouseStatus protocol is used to pass mouse movement and button events to another component such as the PS2Mouse component. Events are only sent when the visualization window is in focus.

The protocol behaviors are:

`mouseMove(int dx, int dy) : void`

This is sent when the host mouse is moved. Mouse movement events are always relative.

`mouseButton(uint8_t button, bool down) : void`

This is sent when a button on the host mouse is pressed or released.

`button` indicates which button has been pressed or released and is typically 0,1 or 2 but can be anything up to 7 depending on the OS and attached mouse.

`down` is true if a button is pressed and false if released.

Parameters

The PS2Mouse component has no parameters.

Registers

The PS2Mouse component has no registers.

Debug features

The PS2Mouse component has no debug features.

Verification and testing

The PS2Mouse component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PS2Mouse component is not expected to significantly affect the performance of a PV system. However if it is connected to the Visualisation component, then the performance of the component is dependent on that of the visualization.

Library dependencies

The PS2Mouse component has no dependencies on external libraries.

5.4.14 PL061_GPIO component

The *General Purpose Input/Output* (GPIO) component is a programmer's view model of the ARM PL061 PrimeCell. It provides eight programmable inputs or outputs. Ports of different widths can be created by multiple instantiation. In addition, an interrupt interface is provided to configure any number of pins as interrupt sources. For more information, see the component documentation. See *ARM PrimeCell General Purpose Input/Output (PL061) Technical Reference Manual*.

Figure 5-35 shows a view of the component in System Canvas.

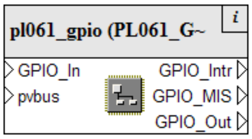


Figure 5-35 PL061_GPIO in System Canvas

This component is written in LISA+.

Ports

Table 5-53 provides a brief description of the PL061_GPIO ports. For more information, see the component documentation.

Table 5-53 PL061_GPIO ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
GPIO_In	Value	Slave	Input lines[a].
GPIO_Out	Value	Master	Output lines[a].
GPIO_Intr	Signal	Master	Interrupt signal indicating to an interrupt controller that an interrupt occurred in one or more of the GPIO_In lines.
GPIO_MIS	Value	Master	Indicates the masked interrupt status[a].

a. Only the lower end eight bits [7:0] are used.

Additional protocols

The PL061_GPIO component has no additional protocols.

Parameters

The PL061_GPIO component has no parameters.

Registers

Table 5-54 provides a description of the configuration registers for the PL061_GPIO component.

Table 5-54 PL061_GPIO registers

Register name	Offset	Access	Description
GPIODATA	`0x000 - 0x3FC`	read/write	GPIO prime data register. The address offsets serve as a mask. Only bits [11:2] are valid as the mask.[a]
GPIODIR	`0x400`	read/write	Data direction register. Set for output, clear for input.
GPIOIS	`0x404`	read/write	Interrupt sense register. Set for level trigger, clear for edge trigger.
GPIOIBE	`0x408`	read/write	Bits set, both edges on corresponding pin trigger and interrupt.
GPIOIEV	`0x40C`	read/write	Interrupt event register. Bit set for rising edge or high level trigger.
GPIOIE	`0x410`	read/write	Interrupt mask register.
GPIORIS	`0x414`	read	Raw interrupt status register.
GPIOMIS	`0x418`	read	Masked interrupt status register.
GPIOIC	`0x41C`	write	Interrupt clear register.
GPIOAFSEL	`0x420`	read/write	Mode control select
GPIOPeriphID0	`0xfe0`	read	Peripheral ID register
GPIOPeriphID1	`0xfe4`	read	Peripheral ID register
GPIOPeriphID2	`0xfe8`	read	Peripheral ID register
GPIOPeriphID3	`0xfec`	read	Peripheral ID register
GPIOPCellID0	`0xff0`	read	PrimeCell ID register

Table 5-54 PL061_GPIO registers (continued)

Register name	Offset	Access	Description
GPIOPCellID1	`0xff4`	read	PrimeCell ID register
GPIOPCellID2	`0xff8`	read	PrimeCell ID register
GPIOPCellID3	`0xffc`	read	PrimeCell ID register

a. For writes, values written to the registers are transferred onto the GPOIT pins if the respective pins have been configured as output ports. Set certain pins in GPIO_Mask to high to enable writing. A similar process applies to reads. Details of how to use this register are covered elsewhere. See the *ARM PrimeCell General Purpose Input/Output (PL061) Technical Reference Manual*.

Debug features

The PL061_GPIO component has no debug features.

Verification and testing

The functions of the PL061_GPIO component have been tested individually using a tailored test suite.

Performance

The PL061_GPIO component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL061_GPIO component has no dependencies on external libraries.

5.4.15 PL080_DMAC component

The PL080_DMAC component is a programmer's view model of the ARM PL080 DMA Controller. It provides eight configurable DMA channels, and 16 DMA ports for handshaking with peripherals. You can configure each channel to operate in one of eight flow control modes either under DMA control or the control of the source or destination peripheral. Transfers can occur on either master channel and can optionally be endian converted on both source and destination transfers. See external documentation for additional details on the PL080. See the *ARM PrimeCell DMA Controller (PL080) Technical Reference Manual*.

Figure 5-36 shows a view of the component in System Canvas.

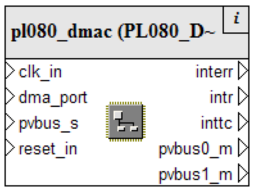


Figure 5-36 PL080_DMAC in System Canvas

This component is written in LISA+.

Ports

Table 5-55 provides a brief description of the ports in the PL080_DMAC component. For more information, see the component documentation.

Table 5-55 PL080_DMAC ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Slave bus for register accesses
clk_in	ClockSignal	Slave	Clock signal to control DMA transfer rate
reset_in	Signal	Slave	Reset signal
pvbus0_m	PVBus	Master	Master bus interface 0 for DMA transfers
pvbus1_m	PVBus	Master	Master bus interface 1 for DMA transfers
interr	Signal	Master	DMA error interrupt signal
inttc	Signal	Master	DMA terminal count signal
intr	Signal	Master	Combined DMA error and terminal count signal
dma_port[16]	PL080_DMAC_DmaPortProtocol[a]	Slave	Peripheral handshake ports

a. See *PL080_DMAC_DmaPortProtocol*.

PL080_DMAC_DmaPortProtocol

The PL080_DMAC component has one additional protocol.

The PL080_DMAC_DmaPortProtocol protocol provides methods to permit handshaking between peripherals and the DMA controller.

`request(uint32 request) : void`

Passes requests from a peripheral to the DMA controller. The request is a bitfield with the low four bits defined. The request is level sensitive and latched internally by the DMA controller. It is sampled and interpreted in a manner dependent on the target channel and configured flow control.

`0: PL080_REQ_BURST`

burst transfer request

`1: PL080_REQ_SINGLE`

single transfer request

`2: PL080_REQ_LBURST`

last burst request

`3: PL080_REQ_LSINGLE`

last single request.

`response(uint32 response) : void`

Passes responses from the DMA controller to peripherals. The response is a bitfield with the low two bits defined. The response is transient rather than level sensitive.

`0: PL080_RES_TC`

terminal count response

`1: PL080_RES_CLR`

clear request response.

Parameters

Table 5-56 provides a description of the configuration parameters for the PL080_DMAC component.

Table 5-56 PL080_DMAC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
fifo_size	Controls the size of channel FIFOs in bytes	Integer	0-1024	16
max_transfer	Limits the number of transfers that can be made atomically	Integer	1-1024	256
generate_clear	Controls whether completion of a burst/single transfer generates a clear response to peripherals	Boolean	true/false	false
activate_delay	Sets the minimum number of cycles after a request or channel enable	Integer	0-256	0

Registers

Table 5-57 provides a description of the configuration registers for the PL080_DMAC component.

Table 5-57 PL080_DMAC registers

Register name	Offset	Access	Description
IrqStatus	`0x000`	read only	Combined interrupt status
IrqTCStatus	`0x004`	read only	Masked terminal count status
IrqTCClear	`0x008`	write only	Terminal count clear
IrqErrStatus	`0x00C`	read only	Masked error status
IrqErrClear	`0x010`	write only	Error clear

Table 5-57 PL080_DMAC registers (continued)

Register name	Offset	Access	Description
RawIrqTCStatus	`0x014`	read only	Raw terminal count status
RawIrqErrStatus	`0x018`	read only	Raw error status
EnabledChannels	`0x01C`	read only	Enabled channels
SoftBReq	`0x020`	read/write	Soft burst request/status
SoftSReq	`0x024`	read/write	Soft single request/status
SoftLBReq	`0x028`	read/write	Soft last burst request/status
SoftLSReq	`0x02C`	read/write	Soft last single request/status
Configuration	`0x030`	read/write	Master configuration
Sync	`0x034`	read/write	Synchronization control
C0SrcAddr	`0x100`	read/write	Channel source address
C0DstAddr	`0x104`	read/write	Channel destination address
C0LLI	`0x108`	read/write	Channel linked list item
C0Control	`0x10C`	read/write	Channel control
C0Config	`0x110`	read/write	Channel configuration
C1SrcAddr	`0x120`	read/write	Channel source address
C1DstAddr	`0x124`	read/write	Channel destination address
C1LLI	`0x128`	read/write	Channel linked list item
C1Control	`0x12C`	read/write	Channel control
C1Config	`0x130`	read/write	Channel configuration
C2SrcAddr	`0x140`	read/write	Channel source address
C2DstAddr	`0x144`	read/write	Channel destination address
C2LLI	`0x148`	read/write	Channel linked list item
C2Control	`0x14C`	read/write	Channel control
C2Config	`0x150`	read/write	Channel configuration
C3SrcAddr	`0x160`	read/write	Channel source address
C3DstAddr	`0x164`	read/write	Channel destination address
C3LLI	`0x168`	read/write	Channel linked list item
C3Control	`0x16C`	read/write	Channel control
C3Config	`0x170`	read/write	Channel configuration
C4SrcAddr	`0x180`	read/write	Channel source address
C4DstAddr	`0x184`	read/write	Channel destination address
C4LLI	`0x188`	read/write	Channel linked list item

Table 5-57 PL080_DMAC registers (continued)

Register name	Offset	Access	Description
C4Control	`0x18C`	read/write	Channel control
C4Config	`0x190`	read/write	Channel configuration
C5SrcAddr	`0x1A0`	read/write	Channel source address
C5DstAddr	`0x1A4`	read/write	Channel destination address
C5LLI	`0x1A8`	read/write	Channel linked list item
C5Control	`0x1AC`	read/write	Channel control
C5Config	`0x1B0`	read/write	Channel configuration
C6SrcAddr	`0x1C0`	read/write	Channel source address
C6DstAddr	`0x1C4`	read/write	Channel destination address
C6LLI	`0x1C8`	read/write	Channel linked list item
C6Control	`0x1CC`	read/write	Channel control
C6Config	`0x1D0`	read/write	Channel configuration
C7SrcAddr	`0x1E0`	read/write	Channel source address
C7DstAddr	`0x1E4`	read/write	Channel destination address
C7LLI	`0x1E8`	read/write	Channel linked list item
C7Control	`0x1EC`	read/write	Channel control
C7Config	`0x1F0`	read/write	Channel configuration
PeriphID0	`0xFE0`	read only	PrimeCell peripheral ID
PeriphID1	`0xFE4`	read only	PrimeCell peripheral ID
PeriphID2	`0xFE8`	read only	PrimeCell peripheral ID
PeriphID3	`0xFEC`	read only	PrimeCell peripheral ID
PCellID0	`0xFF0`	read only	PrimeCell ID
PCellID1	`0xFF4`	read only	PrimeCell ID
PCellID2	`0xFF8`	read only	PrimeCell ID
PCellID3	`0xFFC`	read only	PrimeCell ID

Debug features

The PL080_DMAC component has no debug features.

Verification and testing

The functions of the PL080_DMAC component have been tested individually using a tailored test suite.

Performance

The PL080_DMAC component might have a significant impact on system performance in certain flow control modes. Channels configured for small bursts, or using single bursts, and with peripheral DMA handshaking could add significant overheads. The peripheral has not been fully optimized to make use of the advanced features of the PVBus model.

Library dependencies

The PL080_DMAC component has no dependencies on external libraries.

5.4.16 PL110_CLCD component

The PL110_CLCD component provides a programmer's view model of the PL110 *Color LCD* (CLCD) controller PrimeCell. The model can be connected through a framebuffer port to, for instance, a visualization component, so that LCD output can be viewed.

The implementation is intended to provide a register model of the LCD controller and both timing and bus utilization models are made deliberately inaccurate to favor efficiency of implementation and model speed.

Figure 5-37 shows a view of the component in System Canvas.

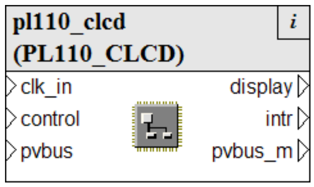


Figure 5-37 PL110_CLCD in System Canvas

This component is written in LISA+.

Ports

Table 5-58 provides a brief description of the ports. See also the *ARM PrimeCell Color LCD Controller (PL110) Technical Reference Manual*.

Table 5-58 PL110_CLCD ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Interrupt signaling for flyback events.
clk_in	ClockSignal	Slave	Master clock input, typically 24MHz, to drive pixel clock timing.
display	LCD	Master	Connection to visualization component. See *Visualisation Library* on page 5-170.
control	Value	Slave	Auxiliary control register 1.
pvbus_m	PVBus	Master	DMA port for video data.

Additional protocols

The PL110_CLCD component has no additional protocols.

Parameters

Table 5-59 provides a description of the configuration parameters for the PL110_CLCD component.

Table 5-59 PL110_CLCD configuration parameters

Parameter name	Description	Type	Allowed value	Default value
pixel_double_limit	Sets a threshold in horizontal pixels, below which pixels sent to the framebuffer are doubled in size horizontally and vertically.	Integer	-	300

Registers

Table 5-60 provides a description of the configuration registers for the PL110_CLCD component.

Table 5-60 PL110_CLCD registers

Register name	Offset	Access	Description
LCDTiming0	`0x000`	read/write	Horizontal timing
LCDTiming1	`0x004`	read/write	Vertical timing
LCDTiming2	`0x008`	read/write	Clock and polarity control
LCDTiming3	`0x00C`	read/write	Line end control
LCDUPBASE	`0x010`	read/write	Upper panel frame base address
LCDLPBASE	`0x014`	read/write	Lower panel frame base address
LCDIMSC	`0x018`	read/write	Interrupt mask
LCDControl	`0x01C`	read/write	Control
LCDRIS	`0x020`	read only	Raw interrupt status
LCDMIS	`0x024`	read only	Masked interrupt status
LCDICR	`0x028`	write only	Interrupt clear
LCDIPCURR	`0x02C`	read only	Upper panel current address
LCDLPCURR	`0x030`	read only	Lower panel current address
LCDPalette	`0x200 - 0x400`	read/write	Palette registers

Table 5-60 PL110_CLCD registers (continued)

Register name	Offset	Access	Description
LCDPeriphID0	`0xfe0`	read	Peripheral ID register
LCDPeriphID1	`0xfe4`	read	Peripheral ID register
LCDPeriphID2	`0xfe8`	read	Peripheral ID register
LCDPeriphID3	`0xfec`	read	Peripheral ID register
LCDPCellID0	`0xff0`	read	PrimeCell ID register
LCDPCellID1	`0xff4`	read	PrimeCell ID register
LCDPCellID2	`0xff8`	read	PrimeCell ID register
LCDPCellID3	`0xffc`	read	PrimeCell ID register

Debug features

The PL110_CLCD component has no debug features.

Verification and testing

The PL110_CLCD component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The PL110_CLCD component might affect the performance of a PV system. The implementation is optimized for situations where the majority of the framebuffer does not change. For instance, displaying full screen video results in significantly reduced performance. Rendering pixel data into an appropriate form for the framebuffer port (rasterization) can also take a significant amount of simulation time. If the pixel data are coming from a PVBusSlave region that has been configured as memory-like, rasterization only occurs in regions where memory contents are modified.

Library dependencies

The PL110_CLCD component has no dependencies on external libraries.

5.4.17 PL111_CLCD component

The PL111_CLCD component provides all features of the programmer's view model of the PL110 CLCD. The new hardware cursor feature of the PL111_CLCD is implemented as the major change compared with PL110.

The view of the component in System Canvas is shown in Figure 5-38 on page 5-78.

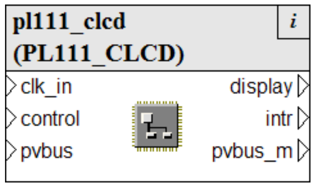


Figure 5-38 PL111_CLCD in System Canvas

This component is written in LISA+.

Ports

Table 5-61 provides a brief description of the ports. See also the *ARM PrimeCell Color LCD Controller (PL111) Technical Reference Manual*.

Table 5-61 PL111_CLCD ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Interrupt signaling for flyback events.
clk_in	ClockSignal	Slave	Master clock input, typically 24MHz, to drive pixel clock timing.
display	LCD	Master	Connection to visualization component. See *Visualisation Library* on page 5-170.
control	Value	Slave	Auxiliary control register 1.
pvbus_m	PVBus	Master	DMA port for video data.

Additional protocols

The PL111_CLCD component has no additional protocols.

Parameters

Table 5-62 provides a description of the configuration parameters for the PL111_CLCD component.

Table 5-62 PL111_CLCD configuration parameters

Parameter name	Description	Type	Allowed value	Default value
pixel_double_limit	Sets a threshold in horizontal pixels, below which pixels sent to the framebuffer are doubled in size horizontally and vertically.	Integer	-	300

Registers

Table 5-63 provides a description of the configuration registers for the PL111_CLCD component.

Table 5-63 PL111_CLCD registers

Register name	Offset	Access	Description
LCDTiming0	`0x0`	read/write	Horizontal timing
LCDTiming1	`0x4`	read/write	Vertical timing
LCDTiming2	`0x8`	read/write	Clock and polarity control
LCDTiming3	`0xC`	read/write	Line end control
LCDUPBASE	`0x10`	read/write	Upper panel frame base address
LCDLPBASE	`0x14`	read/write	Lower panel frame base address
LCDControl	`0x18`	read/write	Control
LCDIMSC	`0x1C`	read/write	Interrupt mask
LCDRIS	`0x20`	read only	Raw interrupt status
LCDMIS	`0x24`	read only	Masked interrupt status
LCDICR	`0x28`	write only	Interrupt clear
LCDIPCURR	`0x2C`	read only	Upper panel current address
LCDLPCURR	`0x30`	read only	Lower panel current address
LCDPalette	`0x200 - 0x3FC`	read/write	Palette registers
CursorImage	`0x800-0xBFC`	read/write	Cursor image RAM register
ClcdCrsCtrl	`0xC00`	read/write	Cursor control
ClcdCrsrConfig	`0xC04`	read/write	Cursor configuration
ClcdCrsrPalette0	`0xC08`	read/write	Cursor palette
ClcdCrsrPalette1	`0xC0C`	read/write	Cursor palette
ClcdCrsrXY	`0XC10`	read/write	Cursor XY position
ClcdCrsrClip	`0xC14`	read/write	Cursor clip position
ClcdCrsrIMSC	`0xc20`	read/write	Cursor interrupt mask set/clear
ClcdCrsrICR	`0xc24`	read/write	Cursor interrupt clear
ClcdCrsrRIS	`0xc28`	read/write	Cursor raw interrupt status
ClcdCrsrMIS	`0xc2c`	read/write	Cursor masked interrupt status
CLCDPeriphID0	`0xfe0`	read	Peripheral ID register
CLCDPeriphID1	`0xfe4`	read	Peripheral ID register

Table 5-63 PL111_CLCD registers (continued)

Register name	Offset	Access	Description
CLCDPeriphID2	`0xfe8`	read	Peripheral ID register
CLCDPeriphID3	`0xfec`	read	Peripheral ID register
CLCDPCellID0	`0xff0`	read	PrimeCell ID register
CLCDPCellID1	`0xff4`	read	PrimeCell ID register
CLCDPCellID2	`0xff8`	read	PrimeCell ID register
CLCDPCellID3	`0xffc`	read	PrimeCell ID register

Debug features

The PL111_CLCD component has no debug features.

Verification and testing

The PL111_CLCD component has been tested as part of the PL111 test system using PL11x test suites.

Performance

The PL111_CLCD component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL111_CLCD component has no dependencies on external libraries.

5.4.18 PL180_MCI component

The PL180_MCI component provides a programmer's view model of the PL180 *Multimedia Card Interface* (MCI). See the *ARM PrimeCell Multimedia Card Interface (PL180) Technical Reference Manual*.

When paired with an MMC card model, the PL180_MCI component provides emulation of a flexible, persistent storage mechanism. The PL180_MMC component fully models the registers of the corresponding PrimeCell, but supports a subset of the functionality of the PL180. Specifically:

- The controller supports block mode transfers, but does not currently support streaming data transfer.
- The controller can be attached to a single MMC device. The MMC bus mode and SDIO modes of the PL180 PrimeCell are not supported. See *MMC component* on page 5-83.
- Command and Data timeouts are not simulated.
- Payload CRC errors are not simulated.
- The DMA interface present in the PL180 PrimeCell is not modeled.
- Minimal timing is implemented within the model.

The view of the component in System Canvas is shown in Figure 5-39 on page 5-81.

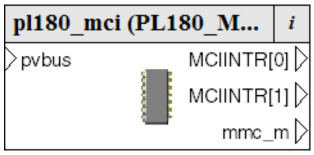


Figure 5-39 PL180_MCI in System Canvas

This component is written in LISA+.

Ports

Table 5-64 provides a brief description of the ports.

Table 5-64 PL180_MCI ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
MCIINTR[0-1]	Signal	Master	Interrupt request ports.
mmc_m	MMC_Protocol	Master	The *Multi Media Card* (MMC) master port.

MMC_Protocol

The PL180_MCI component has one additional protocol.

MMC_Protocol describes an abstract, untimed interface between an MMC controller and an MMC or SD card. The protocol contains a number of methods that must be implemented by the master (controller) and some that must be implemented by the slave (card). This protocol is used by the reference PL180 MCI and MMC models. Further information on the protocol implementation can be found in the source file. See `%PVLIB_HOME%\LISA\MMC_Protocol.lisa`.

Use of this protocol assumes some knowledge of the MultiMediaCard specification, available from the MultiMediaCard Association, `www.mmca.org`.

MMC_Protocol has the following behaviors:

`cmd` Commands are sent from the controller to the card using this behavior, which is implemented by the card model. The MMC command is sent with an optional argument. The card responds as defined by the MMC specification. The controller model checks that the response type matches expectations, and updates its state appropriately. The transaction-level protocol does not model start/stop bits or CRCs on the command/response payload.

For data transfer in the card to controller direction, the behaviors are:

`Rx` After the host and controller have initiated a read through the command interface, the card calls the `Rx` behavior on the controller to provide the read data. The call provides a pointer and a length. The ARM MMC reference model simulates device read latency by waiting a number of clock cycles prior to calling this behavior. If the controller is unable to accept the data, or wants to force a protocol error, it can return false in response to this behavior.

`Rx_rdy` A handshake, used by the controller to inform the card that the controller is ready to receive more data. The ARM MMC reference model does not time out, so waits indefinitely for this handshake in a multiple block data transfer.

For data transfer in the controller to card direction, the behaviors are:

`Tx`	After the host and controller have initiated a write through the command interface, the card calls the `Tx` behavior on the controller. The call provides a pointer to an empty buffer to be written, and a length. The ARM MMC reference model simulates device write latency by waiting a number of clock cycles prior to each buffer being offered.
`Tx_done`	The controller calls this behavior on the card when the block has been written. The card model can then commit the data to its persistent storage.

The card model must also implement:

`cmd_name`	This behavior returns the name of the command issued. A card must implement this behavior, but is free to return an empty string for all requests. Only call this behavior for diagnostic messages.

Parameters

The PL180_MCI component has no parameters.

Registers

Table 5-65 provides a description of the configuration registers for the PL180_MCI component.

Table 5-65 PL180_MCI registers

Register name	Offset	Access	Description
MCIPower	`0x000`	read/write	Power control register
MCIClock	`0x004`	read/write	Clock control register
MCIArgument	`0x008`	read/write	Argument register
MCICommand	`0x00C`	read/write	Command register
MCIRespCmd	`0x010`	read only	Response command register
MCIResponse0	`0x014`	read only	Response register
MCIResponse1	`0x018`	read only	response registerR
MCIResponse2	`0x01C`	read only	Response register
MCIResponse3	`0x020`	read only	Response register
MCIDataTimer	`0x024`	read/write	Data timer
MCIDataLength	`0x028`	read/write	Data length register
MCIDataCtrl	`0x02C`	read/write	Data control register
MCIDataCnt	`0x030`	read only	Data counter
MCIStatus	`0x034`	read only	Status register
MCIClear	`0x038`	write only	Clear register
MCIMask0	`0x03C`	read/write	Interrupt 0 mask register
MCIMask1	`0x040`	read/write	Interrupt 1 mask register

Table 5-65 PL180_MCI registers (continued)

Register name	Offset	Access	Description
MCISelect	`0x044`	read/write	Secure Digital card select register
MCIFifoCnt	`0x048`	read only	FIFO counter
MCIFIFO	`0x080`	read/write	data FIFO register
MCIPeriphID0	`0xFE0`	read only	Peripheral ID bits 7:0
MCIPeriphID1	`0xFE4`	read only	Peripheral ID bits 15:8
MCIPeriphID2	`0xFE8`	read only	Peripheral ID bits 23:16
MCIPeriphID3	`0xFEC`	read only	Peripheral ID bits 31:24
MCIPCellID0	`0xFF0`	read only	PrimeCell ID bits 7:0
MCIPCellID1	`0xFF4`	read only	PrimeCell ID bits 15:8
MCIPCellID2	`0xFF8`	read only	PrimeCell ID bits 23:16
MCIPCellID3	`0xFFC`	read only	PrimeCell ID bits 31:24

Debug features

At compile time, command tracing can be enabled within the PL180_MCI component by modifying the `PL180_TRACE` macro in the `MMC.lisa` file. This sends command and event trace to standard output. You can use this output to help diagnose device driver and controller-to-card protocol issues.

Verification and testing

The PL180_MCI component has been tested in conjunction with the ARM MMC reference model. The component has been tested in the VE example with Boot Monitor and Linux drivers.

Performance

The PL180_MCI component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL180_MCI component has no dependencies on external libraries.

5.4.19 MMC component

The MMC component models a *MultiMedia Card* (MMC) device. The MMC component implementation is a simple MMC, compatible with MMCA specification version 3.31. The MMC component is provided as source code, permitting you to extend functionality if required. Further information about the MMC device can be obtained from the MultiMedia Card Association, `www.mmca.org`.

When paired with a PL180_MCI component, the MMC device model provides emulation of a flexible, persistent storage mechanism. The MMC component uses a file on the host PC to simulate the storage device. The size of this backing store file determines the reported size of the MMC device. As small sections of this file are paged in by the model, large filesystems can be modeled while making efficient use of host PC memory. The backing store file contains a FAT filesystem. The file format is a direct bit copy of an SD card file system image.

The MMC component does not model card insertion or removal. Instead the card is modeled as having already been inserted at system instantiation time.

The following commands are supported by the MMC component:
- MMC_GO_IDLE_STATE
- MMC_SEND_OP_COND
- MMC_ALL_SEND_CID
- MMC_SET_RELATIVE_ADDR
- MMC_SET_DSR
- MMC_SELDESL_CARD
- MMC_SEND_CSD
- MMC_SEND_CID
- MMC_STOP_TRANSMISSION
- MMC_SEND_STATUS
- MMC_GO_INACTIVE_STATE
- MMC_READ_SINGLE_BLOCK
- MMC_READ_MULTIPLE_BLOCK
- MMC_SET_BLOCK_COUNT
- MMC_WRITE_BLOCK
- MMC_WRITE_MULTIPLE_BLOCK.

The MMC_SET_BLOCKLEN command is supported, but the implementation is based on 512 byte blocks.

The following erase commands (Class 5) are supported, but have no effect on the disk backing storage:
- MMC_ERASE_GROUP_START
- MMC_ERASE_GROUP_END
- MMC_ERASE.

The following commands are not supported by the MMC component:
- MMC_SWITCH
- MMC_SEND_EXT_CSD
- MMC_BUSTEST_R
- MMC_BUSTEST_W.

Stream read and write commands (Classes 1 and 3) are not supported:
- MMC_READ_DAT_UNTIL_STOP
- MMC_WRITE_DAT_UNTIL_STOP
- MMC_PROGRAM_CID
- MMC_PROGRAM_CSD.

Block oriented write protection commands (Class 6) are unsupported:
- MMC_SET_WRITE_PROT
- MMC_CLR_WRITE_PROT
- MMC_SEND_WRITE_PROT.

Lock card commands (Class 7) and application-specific commands (Class 8) are unsupported:
- MMC_LOCK_UNLOCK

- MMC_APP_CMD
- MMC_GEN_CMD.

I/O mode commands (Class 9) are unsupported:

- MMC_FAST_IO
- MMC_GO_IRQ_STATE.

Reserved commands are not supported. If a reserved command is used, the MMC ST_ER_B_ILLEGAL_COMMAND bit is set.

The view of the component in System Canvas is shown in Figure 5-40.

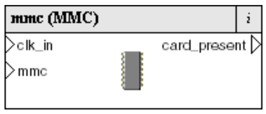


Figure 5-40 MMC in System Canvas

This component is written in LISA+.

Ports

Table 5-66 provides a brief description of the ports.

Table 5-66 MMC ports

Name	Port protocol	Type	Description
card_present	StateSignal	Master	Used to signal whether an MMC image is loaded. It is set if an image is loaded, and is clear if no image is loaded.
clk_in	ClockSignal	Slave	Master clock input.
mmc	MMC_Protocol	Slave	The MMC slave port.

Additional protocols

The MMC component uses the MMC_Protocol. See *MMC_Protocol* on page 5-81.

Parameters

Table 5-67 provides a description of the configuration parameters for the MMC component. Parameters are provided to permit configuration of a number of attributes reflected in the CID and CSD registers. See *Registers*. You can customize the component further by modifying the MMC model source code directly.

Table 5-67 MMC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
p_fast_access	Controls whether MMC card block reads and writes have a simulated delay, or are completed as fast as possible.	Boolean	true/false	true
p_mmc_file	File used for the MMC component backing store	String	valid filename	`mmc.dat`
p_prodName	Card ID product name	String	six character string	`ARMmmc`
p_prodRev	Card ID product revision	Integer	-	`0x1`
p_manid	Card ID manufacturer ID	Integer	-	`0x2`[a]
p_OEMid	Card ID OEM ID	Integer	-	`0x0000`
p_sernum	Card serial number	Integer	-	`0xCA4D0001`

a. This is equivalent to “Sandisk”.

Registers

Table 5-68 on page 5-87 provides a description of the configuration registers for the MMC component. For a full definition of MMC registers, see the MMCA System Summary documentation. Device-specific register information can also be obtained from MMC vendors.

The MMC component registers are not memory-mapped. Instead, they are accessed using relevant MMC commands. The MMC component model makes the registers available through a CADI interface. Modification of these registers through CADI is not recommended, but not prohibited. For example, modifying the card ID (CID) registers can be useful when experimenting with drivers, but direct modification of the STATUS_REG register is likely to put the card model into an indeterminate state.

Table 5-68 MMC registers

Register name	Offset	Description
OCR_REG	`0x000`	Operating conditions register
CID_REG0	`0x004`	Card ID bits 127:96
CID_REG1	`0x005`	Card ID bits 95:64
CID_REG2	`0x006`	Card ID bits 63:32
CID_REG3	`0x007`	Card ID bits 31:0
CSD_REG0	`0x008`	Card specific data bits 127:96
CSD_REG1	`0x009`	Card specific data bits 95:64
CSD_REG2	`0x00a`	Card specific data bits 63:32
CSD_REG3	`0x00b`	Card specific data bit 31:0
RCA_REG	`0x00c`	Relative card address register
DSR_REG	`0x00d`	Driver stage register
BLOCKLEN_REG	`0x00e`	Block length
STATUS_REG	`0x00f`	Card status
BLOCK_COUNT_REG	`0x010`	Block count

Debug features

The MMC component provides a CADI interface for viewing internal registers. See *Registers* on page 5-86. Command tracing can be enabled within the component at compile time by modifying the `MMC_TRACE` macro in the `MMC.lisa` file. This sends command and event trace to standard output. You can use this trace output to help with debugging device driver and controller-to-card protocol issues.

Verification and testing

The MMC component has been tested in conjunction with the ARM MMC reference model. The component has been tested in the VE example with Boot Monitor and Linux drivers.

Performance

The MMC component is not expected to significantly affect the performance of a PV system.

Library dependencies

The MMC component has no dependencies on external libraries.

5.4.20 PL192_VIC component

The PL192 is a *Vectored Interrupt Controller* (VIC) used to aggregate interrupts and generate interrupt signals to the ARM processor. When coupled to an ARM processor that provides a VIC port, routing to the appropriate interrupt handler can be optionally performed in hardware, reducing interrupt latency. The PL192_VIC can also be daisy-chained with other PL192 VICs to permit more than 32 interrupts. The VIC supports hardware and software prioritization of interrupts. For more information, see the VIC documentation. See *ARM PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual*.

Figure 5-41 shows a view of the component in System Canvas.

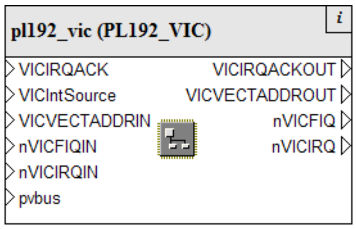


Figure 5-41 PL192_VIC in System Canvas

This component is written in LISA+.

Ports

Table 5-69 provides a brief description of the PL192_VIC component ports. For more information, see the component documentation.

Table 5-69 PL192_VIC ports

Name	Port protocol	Type	Description
VICIntSource [32]	Signal	Slave	Interrupt source input sources
VICVECTADDRIN	Value	Slave	Used to receive vector address when daisy chained
nVICFIQIN	Signal	Slave	Used to receive FIQ signal when daisy chained
nVICIRQIN	Signal	Slave	Used to receive IRQ signal when daisy chained
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
VICIRQACK	Signal	Slave	Receive acknowledge signal from next level VIC or processor
VICIRQACKOUT	Signal	Master	Used to send out acknowledge signals when daisy chained
VICVECTADDROUT	Value	Master	Used to send vector address to next level VIC or processor
nVICFIQ	Signal	Master	Send out FIQ signal to the next level VIC or CPI
nVICIRQ	Signal	Master	Send out IRQ signal to the next level VIC or processor

Additional protocols

The PL192_VIC component has no additional protocols.

Parameters

The PL192_VIC component has no parameters.

Registers

Table 5-70 provides a description of the configuration registers for the PL192_VIC component.

Table 5-70 PL192_VIC registers

Register name	Offset	Access	Description
IRQSTATUS	`0x000`	read only	IRQ status register
FIQSTATUS	`0x004`	read only	FIQ status register
RAWINTR	`0x008`	read only	Raw interrupt status register
INTSELECT	`0x00C`	read/write	Interrupt select register
INTENABLE	`0x010`	read/write	Interrupt enable register
INTENCLEAR	`0x014`	write only	Interrupt enable clear register
SOFTINT	`0x018`	read/write	Software interrupt register
SOFTINTCLEAR	`0x01C`	write only	Software interrupt clear register
PROTECTION	`0x020`	read/write	Protection enable register
SWPRIORITY	`0x024`	read/write	Software priority mask
PRIORITYDAISY	`0x028`	read/write	Vector priority register for daisy chain
VECTADDR[0:31]	`0x100 - 0x17C`	read/write	32 vector addresses
VECTPRIORITY[0:31]	`0x200 - 0x27C`	read/write	32 priority registers
VICADDRESS	`0xF00`	read/write	Vector address register

Debug features

The PL192_VIC component has no debug features.

Verification and testing

The PL192_VIC has been run against the RTL validation suite and has been successfully used in validation platforms.

Performance

The PL192_VIC component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL192_VIC component has no dependencies on external libraries.

5.4.21 PL310_L2CC component

The PL310_L2CC provides a model of an *Level 2 Cache Controller* (L2CC). The presence of additional on-chip secondary cache can improve performance when significant memory traffic is generated by the processor. A secondary cache assumes the existence of a Level 1, or primary, cache, which is closely coupled or internal to the processor. For more information, see the component documentation. See the *ARM PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual*. For more information about the accuracy and functionality of Fast Models and cache models, see Chapter 2 *Accuracy and Functionality*.

The PL310_L2CC component has two modes of operation:

- Register view: Cache control registers are present but the cache behavior is not modeled.
- Functional model: Cache behavior is modeled.

The mode of operation is controlled by a parameter to the component. See *Parameters* on page 5-92, parameter `cache-state_modelled`.

Figure 5-42 shows a view of the component in System Canvas.

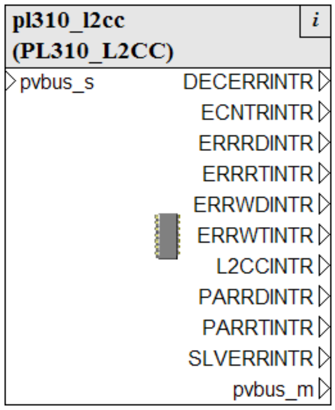


Figure 5-42 PL310_L2CC in System Canvas

This component is written in LISA+.

The PL310 is only supported when connected to the Cortex-A5 or Cortex-A9 processor. An example system is shown in Figure 5-43 on page 5-91.

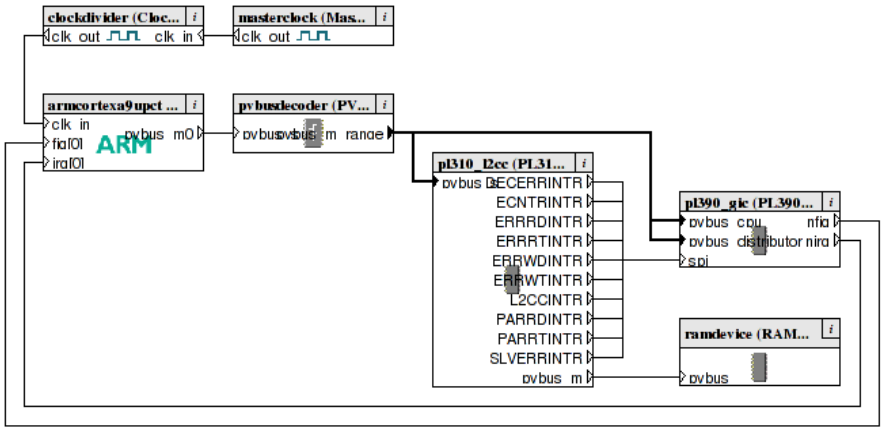


Figure 5-43 PL310_L2CC in an example system

Ports

Table 5-71 provides a brief description of the PL310_L2CC component ports. For more information, see the component documentation.

Table 5-71 PL310_L2CC ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Slave port for connection to PV bus master/decoder
pvbus_m	PVBus	Master	Master port for connection to PV bus master/decoder
DECERRINTR	Signal	Master	Decode error received on master port from L3
ECNTRINTR	Signal	Master	Event counter overflow / increment
ERRRDINTR	Signal	Master	Error on L2 data RAM read
ERRRTINTR	Signal	Master	Error on L2 tag RAM read
ERRWDINTR	Signal	Master	Error on L2 data RAM write
ERRWTINTR	Signal	Master	Error on L2 tag RAM write
L2CCINTR	Signal	Master	Combined interrupt output
PARRDINTR	Signal	Master	Parity error on L2 data RAM read
PARRTINTR	Signal	Master	Parity error on L2 tag RAM read
SLVERRINTR	Signal	Master	Slave error on master port from L3

Additional protocols

The PL310_L2CC component has no additional protocols.

Parameters

Table 5-72 lists the parameters in the PL310_L2CC.

Table 5-72 PL310_L2CC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
ASSOCIATIVITY	Associativity for auxiliary control register	Integer	0 (8-way), 1 (16-way)	0
CACHEID	Cache controller cache ID	Integer	0 - 63	0
cache-state_modelled[a]	Specifies whether real cache state is modeled (vs. register model)	Boolean	true/false	false
CFGBIGEND	Big-endian mode for accessing configuration registers out of reset	Integer	0,1	0
LOCKDOWN_BY_LINE[b]	Lockdown by line	Integer	0,1	0
LOCKDOWN_BY_MASTER[c]	Lockdown by master	Integer	0,1	0
REGFILEBASE	Base address for accessing configuration registers	Integer	0 - 0xfffff000	0x1f002000
WAYSIZE	Size of ways for auxiliary control register	Integer	0 - 7	1

a. To understand the performance impact of enabling this parameter, see section *Performance* on page 5-94.

b. Value is reflected in CacheType register bit 25, but the feature is not switched off when the parameter is 0.

c. Value is reflected in CacheType register bit 26, but the feature is not switched off when the parameter is 0.

Registers

Table 5-73 provides a description of the configuration registers for the PL310_L2CC component.

Table 5-73 PL310_L2CC registers

Register name	Offset	Access	Description
CacheID	`0x000`	read only	r0 cache ID
CacheType	`0x004`	read only	r0 cache type
Ctrl	`0x100`	read/write	r1 control
AuxCtrl	`0x104`	read/write	r1 auxiliary control

Table 5-73 PL310_L2CC registers (continued)

Register name	Offset	Access	Description
TagLatencyCtrl[a]	`0x108`	read/write	r1 tag RAM latency control
DataLatencyCtrl[a]	`0x10C`	write only	r1 data RAM latency control
EventCounterCtrl[a]	`0x200`	read/write	r2 event counter control
EventCounter1Cfg[a]	`0x204`	write only	r2 event counter 1 configuration
EventCounter0Cfg[a]	`0x208`	read/write	r2 event counter0 configuration
EventCounter1[a]	`0x20C`	read/write	r2 event counter 1 value
EventCounter0[a]	`0x210`	read/write	r2 event counter 0 value
InterruptMask	`0x214`	read/write	r2 interrupt mask
MaskedInterruptStatus	`0x218`	read only	r2 masked interrupt status
RawInterruptStatus	`0x21C`	read only	r2 raw interrupt status
InterruptClear	`0x220`	write only	r2 interrupt clear
CacheSync[a]	`0x730`	read/write	r7 cache sync
InvalidateByPA	`0x770`	read/write	r7 invalidate line by PA
InvalidateByWay	`0x77C`	read/write	r7 invalidate by way
CleanByPA	`0x7B0`	read/write	r7 clean line by PA
CleanByIdxWay	`0x7B8`	read/write	r7 clean line by index or way
CleanByWay	`0x7BC`	read/write	r7 clean by way
CleanInvalByPA	`0x7F0`	read/write	r7 clean and invalidate line by PA
CleanInvalByIdxWay	`0x7F8`	read/write	r7 clean and invalidate line by index or way
CleanInvalByWay	`0x7FC`	read/write	r7 clean and invalidate by way
DataLockdown0	`0x900`	read/write	r9 data lockdown 0 by way
InstructionLockdown0	`0x904`	read/write	r9 instruction lockdown 0 by way
DataLockdown1	`0x908`	read/write	r9 data lockdown 1 by way
InstructionLockdown1	`0x90C`	read/write	r9 instruction lockdown 1 by way
DataLockdown2	`0x910`	read/write	r9 data lockdown 2 by way
InstructionLockdown2	`0x914`	read/write	r9 instruction lockdown 2 by way
DataLockdown3	`0x918`	read/write	r9 data lockdown 3 by way

Table 5-73 PL310_L2CC registers (continued)

Register name	Offset	Access	Description
InstructionLockdown3	`0x91C`	read/write	r9 instruction lockdown 3 by way
DataLockdown4	`0x920`	read/write	r9 data lockdown 4 by way
InstructionLockdown4	`0x924`	read/write	r9 instruction lockdown 4 by way
DataLockdown5	`0x928`	read/write	r9 data lockdown 5 by way
InstructionLockdown5	`0x92C`	read/write	r9 instruction lockdown 5 by way
DataLockdown6	`0x930`	read/write	r9 data lockdown 6 by way
InstructionLockdown6	`0x934`	read/write	r9 instruction lockdown 6 by way
DataLockdown7	`0x938`	read/write	r9 data lockdown 7 by way
InstructionLockdown7	`0x93C`	read/write	r9 instruction lockdown 7 by way
LockdownByLineEnable	`0x950`	read/write	r9 lockdown by line enable
UnlockAll	`0x954`	read/write	r9 unlock all lines by way
AFilterStart[a]	`0xC00`	read/write	r12 address filtering start
AFilterEnd[a]	`0xC04`	read/write	r12 address filtering end
DebugControl[a]	`0xF40`	read/write	r15 debug control register

a. Operation of this register is not functionally modeled.

Debug features

The PL310_L2CC component exports the PL310 registers by CADI.

Verification and testing

The PL310_L2CC has been run against the RTL validation suite and passes for supported features. It has also been tested with operating system booting in both normal and exclusive modes, and has successfully used in validation platforms.

Performance

The performance of the PL310_L2CC component depends on the configuration of the associated L1 caches and the mode it is in.

- register mode: The PL310_L2CC model does not significantly affect performance.
- functional mode with functional-mode L1: the addition of a functional L2 cache has minimal further impact on performance when running applications that are cache-bound.
- functional mode with a register-mode L1: the PL310_L2CC has a significant impact on system performance in this case.

Library dependencies

The PL310_L2CC component has no dependencies on external libraries.

Functionality

The PL310 implements the programmer visible functionality of the PL310, and excludes some non-programmer visible features.

Implemented hardware features

The following features of the PL310 hardware are implemented in the PL310 model:

- Physically addressed and physically tagged.
- Lockdown format C supported, for data and instructions. Lockdown format C is also known as way locking.
- Lockdown by line supported.
- Lockdown by master id supported.
- Direct mapped to 16-way associativity, depending on the configuration and the use of lockdown registers. The associativity is configurable as 8 or 16.
- L2 cache available size can be 16KB to 8MB, depending on configuration and the use of the lockdown registers.
- Fixed line length of 32 bytes (eight words or 256 bits).
- Supports all of the AXI cache modes:
  - write-through and write-back.
  - read allocate, write allocate, read and write allocate.
- Force write allocate option to always have cacheable writes allocated to L2 cache, for processors not supporting this mode.
- Normal memory non-cacheable shared reads are treated as cacheable non-allocatable. Normal memory non-cacheable shared writes are treated as cacheable write-through no write-allocate. There is an option, Shared Override, to override this behavior.
- TrustZone support, with the following features:
  - Non-Secure (NS) tag bit added in tag RAM and used for lookup in the same way as an address bit.
  - NS bit in Tag RAM used to determine security level of evictions to L3.
  - Restrictions for NS accesses for control, configuration, and maintenance registers to restrict access to secure data.
- Pseudo-Random victim selection policy. You can make this deterministic with use of lockdown registers.
- Software option to enable exclusive cache configuration.
- Configuration registers accessible using address decoding in the component.
- Interrupt triggering in case of an error response when accessing L3.
- Maintenance operations.
- Prefetching capability.

Hardware features not implemented

The following features of the PL310 hardware are not implemented in the PL310 model, most of them are not relevant from a PV modeling point of view:

- There is no interface to the data and tag RAM as they are embedded to the model.
- Critical word first linefill not supported, as this is not relevant for PV modeling.
- Buffers are not modeled.
- Outstanding accesses on slave and master ports cannot occur by design in a PV model as all transactions are atomic.
- Option to select one or two master ports and option to select one or two slave ports is not supported. Only one master port and one slave port is supported.
- Clock management and power modes are not supported, as they is not relevant for PV modeling.
- Wait, latency, clock enable, parity, and error support for data and tag RAMs not included, as this is not relevant for PV modeling, and the data and tag RAMs embedded in the model cannot generate error responses.
- MBIST support is not included.
- Debug mode and debug registers are not supported.
- Test mode and scan chains are not supported.
- L2 cache event monitoring is not supported.
- Address filtering in the master ports is not supported.
- Performance counters are not supported.
- Specific Cortex-A9 related optimizations are not supported: Prefetch hints, Full line of zero and Early write response.
- Hazard detection is not required because of the atomic nature of the accesses at PV modeling and the fact that buffers are not modeled, thus hazards cannot occur.

Registers belonging to features not implemented are accessible but do not have a functionality.

Features additional to the Hardware

- Data RAM and Tag RAM are embedded to the model.

Features different to the Hardware

- Error handling. DECERR from the master port is mapped to SLVERR. Internal errors in cache RAM (like parity errors) cannot happen in the model.
- Background cache operations do not occur in the background. They occur atomically.
- The LOCKDOWN_BY_LINE and LOCKDOWN_BY_MASTER parameter values are reflected in the CacheType register, but the feature is not switched off when the parameter is 0.

5.4.22 PL330_DMAC component

The PL330_DMAC component is a programmer's view model of the ARM PL330 *Direct Memory Access Controller* (DMAC). The DMA controller is modeled using a single LISA component but with a C++ model for each of the channels included in the LISA file. Enabled channels are kept on an enabled_channels stack in priority order. When a channel state changes, rearbitration takes place to make the highest (topmost) channel active. See the *ARM PrimeCell DMA Controller (PL330) Technical Reference Manual*.

Figure 5-44 shows a view of the component in System Canvas.

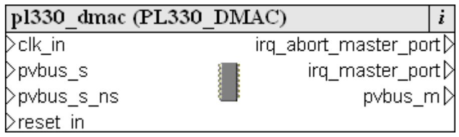


Figure 5-44 PL330_DMAC in System canvas

This component is written in LISA+.

Ports

Table 5-74 provides a brief description of the ports for the PL330_DMAC component. For more information, see the component documentation.

Table 5-74 PL330_DMAC ports

Name	Port protocol	Type	Description
clk_in	ClockSignal	Slave	Main processor clock input.
irq_abort_master_port	Signal	Master	Undefined instruction or instruction error.
irq_master_port	Signal	Master	Sets when DMASEV.
pvbus_m	PVBus	Master	Master port for all memory accesses.
pvbus_s_ns	PVBus	Slave	Slave port for all register accesses (non-secure).
reset_in	Signal	Slave	Reset signal.

Additional protocols

The PL330_DMAC component has no additional protocols.

Parameters

Table 5-75 provides a description of the configuration parameters for the PL330_DMAC component.

Table 5-75 PL330_DMAC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
p_max_irqs	Number of interrupts	Integer	0-32	32
p_max_channels	Virtual channels	Integer	≤8	8
p_controller_nsecure	Controller non-secure at reset	Boolean	`true` or `false`	`false`
p_controller_boots	DMA boots from reset	Boolean	`true` or `false`	`true`
p_reset_pc	DMA PC at reset	Integer	Any valid address	`0x60000000`

Registers

Table 5-76 provides a description of the configuration registers for the PL330_DMAC component.

Table 5-76 PL330_DMAC registers

Register name	Offset	Access	Description
DS	0x000	read only	DMA status register
DPC	0x004	read only	DMA program counter register
INTEN	0x020	read/write	Interrupt enable register
ES	0x024	read only	Event status register
INTSTATUS	0x028	read only	Interrupt status register
INTCLR	0x02c	write only	Interrupt clear register
FSM	0x030	read only	Fault status DMA manager register
FSC	0x034	read only	Fault status DMA channel register
FTM	0x038	read only	Fault type DMA manager register
FTC0	0x040	read only	Fault type for DMA channel 0
FTC1	0x044	read only	Fault type for DMA channel 1
FTC2	0x048	read only	Fault type for DMA channel 2
FTC3	0x04c	read only	Fault type for DMA channel 3

Table 5-76 PL330_DMAC registers (continued)

Register name	Offset	Access	Description
FTC4	0x050	read only	Fault type for DMA channel 4
FTC5	0x054	read only	Fault type for DMA channel 5
FTC6	0x058	read only	Fault type for DMA channel 6
FTC7	0x05c	read only	Fault type for DMA channel 7
CS0	0x100	read only	Channel status for DMA channel 0
CS1	0x108	read only	Channel status for DMA channel 1
CS2	0x110	read only	Channel status for DMA channel 2
CS3	0x118	read only	Channel status for DMA channel 3
CS4	0x120	read only	Channel status for DMA channel 4
CS5	0x128	read only	Channel status for DMA channel 5
CS6	0x130	read only	Channel status for DMA channel 6
CS7	0x138	read only	Channel status for DMA channel 7
CPC0	0x104	read only	Channel PC for DMA channel 0
CPC1	0x10c	read only	Channel PC for DMA channel 1
CPC2	0x114	read only	Channel PC for DMA channel 2
CPC3	0x11c	read only	Channel PC for DMA channel 3
CPC4	0x124	read only	Channel PC for DMA channel 4
CPC5	0x12c	read only	Channel PC for DMA channel 5
CPC6	0x134	read only	Channel PC for DMA channel 6
CPC7	0x13c	read only	Channel PC for DMA channel 7
SA_0	0x400	read only	Source address for DMA channel 0
SA_1	0x420	read only	Source address for DMA channel 1

Table 5-76 PL330_DMAC registers (continued)

Register name	Offset	Access	Description
SA_2	0x440	read only	Source address for DMA channel 2
SA_3	0x460	read only	Source address for DMA channel 3
SA_4	0x480	read only	Source address for DMA channel 4
SA_5	0x4A0	read only	Source address for DMA channel 5
SA_6	0x4C0	read only	Source address for DMA channel 6
SA_7	0x4E0	read only	Source address for DMA channel 7
DA_0	0x404	read only	Destination address for DMA channel 0
DA_1	0x424	read only	Destination address for DMA channel 1
DA_2	0x444	read only	Destination address for DMA channel 2
DA_3	0x464	read only	Destination address for DMA channel 3
DA_4	0x484	read only	Destination address for DMA channel 4
DA_5	0x4A4	read only	Destination address for DMA channel 5
DA_6	0x4C4	read only	Destination address for DMA channel 6
DA_7	0x4E4	read only	Destination address for DMA channel 7
CC_0	0x408	read only	Channel control for DMA channel 0
CC_1	0x428	read only	Channel control for DMA channel 1
CC_2	0x448	read only	Channel control for DMA channel 2
CC_3	0x468	read only	Channel control for DMA channel 3
CC_4	0x488	read only	Channel control for DMA channel 4
CC_5	0x4A8	read only	Channel control for DMA channel 5

Table 5-76 PL330_DMAC registers (continued)

Register name	Offset	Access	Description
CC_6	0x4C8	read only	Channel control for DMA channel 6
CC_7	0x4E8	read only	Channel control for DMA channel 7
LC0_0	0x40C	read only	Loop counter for DMA channel 0
LC0_1	0x42C	read only	Loop counter for DMA channel 1
LC0_2	0x44C	read only	Loop counter for DMA channel 2
LC0_3	0x46C	read only	Loop counter for DMA channel 3
LC0_4	0x48C	read only	Loop counter for DMA channel 4
LC0_5	0x4AC	read only	Loop counter for DMA channel 5
LC0_6	0x4CC	read only	Loop counter for DMA channel 6
LC0_7	0x4EC	read only	Loop counter for DMA channel 7
LC1_0	0x410	read only	Loop counter 1 for DMA channel 0
LC1_1	0x430	read only	Loop counter 1 for DMA channel 1
LC1_2	0x450	read only	Loop counter 1 for DMA channel 2
LC1_3	0x470	read only	Loop counter 1 for DMA channel 3
LC1_4	0x490	read only	Loop counter 1 for DMA channel 4
LC1_5	0x4B0	read only	Loop counter 1 for DMA channel 5
LC1_6	0x4D0	read only	Loop counter 1 for DMA channel 6
LC1_7	0x4F0	read only	Loop counter 1 for DMA channel 7
DBGSTATUS	0xD00	read only	Debug status register
DBGCMD	0xD04	write only	Debug command register
DBGINST0	0xD08	write only	Debug instruction-0 register
DBGINST1	0xD0C	write only	Debug instruction-1 register

Table 5-76 PL330_DMAC registers (continued)

Register name	Offset	Access	Description
periph_id_0	0xFE0	read only	Peripheral ID register 0
periph_id_1	0xFE4	read only	Peripheral ID register 1
periph_id_2	0xFE8	read only	Peripheral ID register 2
periph_id_3	0xFEC	read only	Peripheral ID register 3
pcell_id_0	0xFF0	read only	PrimeCell ID register 0
pcell_id_1	0xFF4	read only	PrimeCell ID register 1
pcell_id_2	0xFF8	read only	PrimeCell ID register 2
pcell_id_3	0xFFC	read only	PrimeCell ID register 3

Debug features

The PL330_DMAC component has no debug features.

Verification and testing

The functions of the PL330_DMAC component have been tested individually using a tailored test suite.

Performance

The PL330_DMAC component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL330_DMAC component has no dependencies on external libraries.

5.4.23 PL340_DMC component

The PL340_DMC component is a programmer's view model of the ARM PL340 *Dynamic Memory Controller* (DMC). It provides an interface for up to four DRAM chips. The implementation also provides an apb_interface to configure the controller behavior. Further information is available in the component documentation. See the *ARM PrimeCell Dynamic Memory Controller (PL340) Technical Reference Manual*.

Figure 5-45 shows a view of the component in System Canvas.

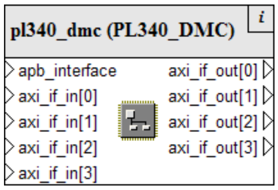


Figure 5-45 PL340_DMC in System Canvas

This component is written in LISA+.

Ports

Table 5-77 provides a brief description of the ports for the PL340_DMC component. For more information, see the component documentation. See the *ARM PrimeCell Dynamic Memory Controller (PL340) Technical Reference Manual*.

Table 5-77 PL340_DMC ports

Name	Port protocol	Type	Description
axi_chip_if_in[4]	PVBus	Slave	Slave bus for connecting to bus decoder
apb_interface	PVBus	Slave	Slave bus interface for register access
axi_chip_if_out[4]	PVBus	Master	Master to connect to DRAM

Additional protocols

The PL340_DMC component has no additional protocols.

Parameters

Table 5-78 provides a description of the configuration parameters for the PL340_DMC component.

Table 5-78 PL340_DMC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
MEMORY_WIDTH	Indicates the width, in bits, of connected memory	Integer	16, 32, 64	32
IF_CHIP_0 to IF_CHIP_3	Indicates whether memory is connected	Integer	-1, 0[a]	-1

a. The permitted values have the following meanings:

-1	nothing connected to the interface
0	DRAM connected.

Registers

Table 5-79 provides a description of the configuration registers for the PL340_DMC component. These are accessible through the APB interface.

Table 5-79 PL340_DMC registers

Register name	Offset	Access	Description
memc_status	`0x000`	read only	Memory controller status register
memc_cmd	`0x004`	write only	Used to modify the state machine of the controller
direct_cmd	`0x008`	write only	Used to set the memory controller configurations

Table 5-79 PL340_DMC registers (continued)

Register name	Offset	Access	Description
memory_cfg	`0x00C`	read/write	Set/read the configuration of the controller
refresh_prd	`0x010`	read/write	Refresh period register
cas_latency	`0x014`	read/write	CAS latency register
t_dqss	`0x018`	read/write	t_dqss register
t_mrd	`0x01C`	read/write	t_mrd register
t_ras	`0x020`	read/write	t_ras register
t_rc	`0x024`	read/write	t_rc register
t_rcd	`0x028`	read/write	t_rcd register
t_rfc	`0x02C`	read/write	t_rfc register
t_rp	`0x030`	read/write	t_rp register
t_rrd	`0x034`	read/write	t_rrd register
t_wr	`0x038`	read/write	t_wr register
t_wtr	`0x03C`	read/write	t_wtr register
t_xp	`0x040`	read/write	t_xp register
t_xsr	`0x044`	read/write	t_xsr register
t_esr	`0x048`	read/write	t_esr register
id_00_cfg	`0x100`	read/write	Sets the QOS
id_01_cfg	`0x104`	read/write	Sets the QOS
id_02_cfg	`0x108`	read/write	Sets the QOS
id_03_cfg	`0x10C`	read/write	Sets the QOS
id_04_cfg	`0x110`	read/write	Sets the QOS
id_05_cfg	`0x114`	read/write	Sets the QOS
id_06_cfg	`0x118`	read/write	Sets the QOS
id_07_cfg	`0x11C`	read/write	Sets the QOS
id_08_cfg	`0x120`	read/write	Sets the QOS
id_09_cfg	`0x124`	read/write	Sets the QOS
id_10_cfg	`0x128`	read/write	Sets the QOS
id_11_cfg	`0x12C`	read/write	Sets the QOS
id_12_cfg	`0x130`	read/write	Sets the QOS
id_13_cfg	`0x134`	read/write	Sets the QOS
id_14_cfg	`0x138`	read/write	Sets the QOS

Table 5-79 PL340_DMC registers (continued)

Register name	Offset	Access	Description
id_15_cfg	`0x13C`	read/write	Sets the QOS
chip_0_cfg	`0x200`	read/write	Set up the external memory device configuration
chip_1_cfg	`0x204`	read/write	Set up the external memory device configuration
chip_2_cfg	`0x208`	read/write	Set up the external memory device configuration
chip_3_cfg	`0x20C`	read/write	Set up the external memory device configuration
user_status	`0x300`	read only	User status register
user_config	`0x304`	write only	User configuration register
periph_id_0	`0xFE0`	read only	Peripheral ID register 0[a]
periph_id_1	`0xFE4`	read only	Peripheral ID register 1[a]
periph_id_2	`0xFE8`	read only	Peripheral ID register 2[a]
periph_id_3	`0xFEC`	read only	Peripheral ID register 3[a]
pcell_id_0	`0xFF0`	read only	PrimeCell ID register 0[a]
pcell_id_1	`0xFF4`	read only	PrimeCell ID register 1[a]
pcell_id_2	`0xFF8`	read only	PrimeCell ID register 2[a]
pcell_id_3	`0xFFC`	read only	PrimeCell ID register 3[a]

a. This register has no CADI interface.

Debug features

The PL340_DMC component has no debug features.

Verification and testing

The PL340_DMC functions of the component have been tested individually using a tailored test suite.

Performance

The PL340_DMC component has negligible impact on performance.

Library dependencies

The PL340_DMC component has no dependencies on external libraries.

5.4.24 PL350_SMC component

The PL350_SMC component is a programmer's view model of the ARM PL350 *Static Memory Controller* (SMC). It provides two memory interfaces. Each interface can be connected to a maximum of four memory devices, giving a total of eight inputs from the PVBusDecoder and eight outputs to either SRAM or NAND devices. Only one kind of memory can be connected to a particular interface, either SRAM or NAND. Further technical details on the SMC are described elsewhere. See the *ARM PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual*.

The PL350_SMC component implementation provides a PVBus slave to control the device behavior. A remap port is also provided to assist in remapping particular memory regions.

Figure 5-46 shows a view of the component in System Canvas.

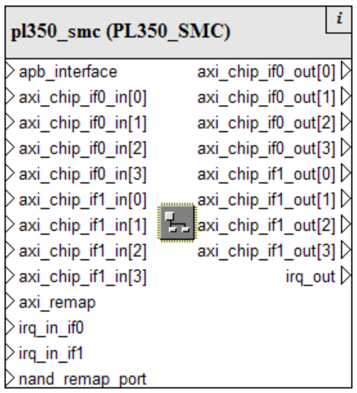


Figure 5-46 PL350_SMC in System Canvas

This component is written in LISA+.

Ports

Table 5-80 provides a brief description of the ports for the PL350_SMC component. For more information, see the component documentation. See *ARM PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual*.

Table 5-80 PL350_SMC ports

Name	Port protocol	Type	Description
axi_chip_if0_in[4]	PVBus	Slave	Slave bus for interface 0 connecting to memory
axi_chip_if1_in[4]	PVBus	Slave	Slave bus for interface 1 PVBus connecting to memory
apb_interface	PVBus	Slave	Slave bus interface for register access
axi_chip_if0_out[4]	PVBus	Master	Master interface 0 to connect to SRAM/NAND
axi_chip_if1_out[4]	PVBus	Master	Master interface 1 to connect to SRAM/NAND
axi_remap	PVBus	Slave	Remaps the device to `0x0`
irq_in_if0	Signal	Slave	Interface 0 interrupt connection from the device

Table 5-80 PL350_SMC ports (continued)

Name	Port protocol	Type	Description
irq_in_if1	Signal	Slave	Interface 1 interrupt connection from the device
nand_remap_port	PVBus	Slave	Remaps the connected NAND port to `0x0`
irq_out	Signal	Master	Interrupt port

Additional protocols

The PL350_SMC component has no additional protocols.

Parameters

Table 5-81 provides a description of the configuration parameters for the PL350_SMC component.

Table 5-81 PL350_SMC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
IF0_MEM_TYPE_PARAMETER IF1_MEM_TYPE_PARAMETER	Indicates memory type for interfaces 0 and 1: **0** SRAM **1** NAND.	Integer	0, 1	0
REMAP	Indicates if a particular interface has been remapped: **-1** remap not enabled **0 to 7** identifies device that gets address `0x0`.	Integer	-1, 0-7	-1
IF0_CHIP_0 to IF0_CHIP_3 IF1_CHIP_0 to IF1_CHIP_3	Indicates memory connected to chip slots for interfaces 0 and 1: **True** nothing connected to the interface **False** memory connected.	Boolean	True, False	False
IF0_CHIP0_BASE to IF0_CHIP3_BASE IF1_CHIP0_BASE to IF1_CHIP3_BASE	Chip *y* base address for interfaces 0 and 1.	Integer	address where chips connected	0
IF0_CHIP0_SIZE to IF0_CHIP3_SIZE IF1_CHIP0_SIZE to IF1_CHIP3_SIZE	Chip *y* size for interfaces 0 and 1.	Integer	device size	0

Registers

Table 5-82 provides a description of the configuration registers for the PL350_SMC component. These are accessible through the APB interface.

Table 5-82 PL350_SMC registers

Register name	Offset	Access	Description
memc_status	`0x000`	read only	Memory controller status register
memif_cfg	`0x004`	read only	Memory interface configuration register
memc_cfg_set	`0x008`	write only	Used to set memory controller configurations
memc_cfg_clr	`0x00C`	write only	Clear the configuration register
direct_cmd	`0x010`	write only	Commands sent to the device
set_cycles	`0x014`	write only	Holding register for cycle settings
set_opmode	`0x018`	write only	Holding register for opmode settings
refresh_period_0	`0x020`	read/write	Insert idle cycles on interface 0
refresh_period_1	`0x024`	read/write	Insert idle cycles on interface 1
device_cycles0_0	`0x100`	read only	Device cycle configuration
device_cycles0_1	`0x120`	read only	Device cycle configuration
device_cycles0_2	`0x140`	read only	Device cycle configuration
device_cycles0_3	`0x160`	read only	Device cycle configuration
device_cycles1_0	`0x180`	read only	Device cycle configuration
device_cycles1_1	`0x1A0`	read only	Device cycle configuration
device_cycles1_2	`0x1C0`	read only	Device cycle configuration
device_cycles1_3	`0x1E0`	read only	Device cycle configuration
opmode0_0	`0x104`	read only	Opmode configuration
opmode0_1	`0x124`	read only	Opmode configuration
opmode0_2	`0x144`	read only	Opmode configuration
opmode0_3	`0x164`	read only	Opmode configuration
opmode1_0	`0x184`	read only	Opmode configuration
opmode1_1	`0x1A4`	read only	Opmode configuration
opmode1_2	`0x1C4`	read only	Opmode configuration
opmode1_3	`0x1E4`	read only	Opmode configuration

Table 5-82 PL350_SMC registers (continued)

Register name	Offset	Access	Description
user_status	`0x200`	read/write	User status register
user_config	`0x204`	read/write	User configuration register
periph_id_0	`0xFE0`	read only	Peripheral ID register 0[a]
periph_id_1	`0xFE4`	read only	Peripheral ID register 1[a]
periph_id_2	`0xFE8`	read only	Peripheral ID register 2[a]
periph_id_3	`0xFEC`	read only	Peripheral ID register 3[a]
pcell_id_0	`0xFF0`	read only	PrimeCell ID register 0[a]
pcell_id_1	`0xFF4`	read only	PrimeCell ID register 1[a]
pcell_id_2	`0xFF8`	read only	PrimeCell ID register 2[a]
pcell_id_3	`0xFFC`	read only	PrimeCell ID register 3[a]

a. This register has no CADI interface.

Debug features

The PL350_SMC component has no debug features.

Verification and testing

The functions of the PL350_SMC component have been tested individually using a tailored test suite.

Performance

The PL350_SMC component is optimized to have negligible impact on transaction performance, except when memory remap settings are changed when there might be a significant effect.

Library dependencies

The PL350_SMC component has no dependencies on external libraries.

5.4.25 PL350_SMC_NAND_FLASH component

The PL350_SMC_NAND_FLASH component is a programmer's view model of the flash that must be connected to the PL350_SMC component. Information on how to program the PL350_SMC_NAND_FLASH component is available in the component documentation. See *ARM PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual*.

Figure 5-47 shows a view of the component in System Canvas.

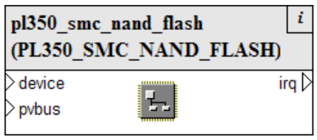


Figure 5-47 PL350_SMC_NAND_FLASH in System Canvas

This component is written in LISA+.

Ports

Table 5-83 provides a brief description of the PL350_SMC_NAND_FLASH component ports. For more information, see the static memory controller documentation.

Table 5-83 PL350_SMC_NAND_FLASH ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
irq	Signal	Master	Interrupt signaling
device	PVDevice	Slave	Port used to define device behavior

Additional protocols

The PL350_SMC_NAND_FLASH component has no additional protocols.

Parameters

Table 5-84 provides a description of the configuration parameters for the PL350_SMC_NAND_FLASH component.

Table 5-84 PL350_SMC_NAND_FLASH configuration parameters

Parameter name	Description	Type	Allowed value	Default value
DEVICE_NAME	Name of the device	String	-	"Samsung K9F1G08U0M"

DEVICE_1
DEVICE_2
DEVICE_3
DEVICE_4
	The device ID	Integer	-	`default(0xEC)`
				`default(0xDA)`
				`default(0x80)`
				`default(0x15)`

NAND_FLASH_SIZE	Size of the flash device in bytes	Integer	-	`0x1080000`
NAND_PAGE_SIZE	Page size	Integer	`0x2112` or `0x528`	`0x2112`
NAND_SPARE_SIZE_PER_PAGE	Extra bits	Integer	64 or 16	64
NAND_VALID_SIZE_PER_PAGE	Valid page size	Integer	-	2048
NAND_PAGE_COUNT_PER_BLOCK	Number of pages in each block	Integer	-	64
NAND_BLOCK_COUNT	Number of blocks in the flash device	Integer	-	2048

Registers

The PL350_SMC_NAND_FLASH component has no registers.

Debug features

The PL350_SMC_NAND_FLASH component has no debug features.

Verification and testing

The PL350_SMC_NAND_FLASH component has been tested as part of an integrated platform.

Performance

The PL350_SMC_NAND_FLASH component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL350_SMC_NAND_FLASH component has no dependencies on external libraries.

5.4.26 PL390_GIC component

The PL390 is a Generic Interrupt Controller (GIC) which implements the ARM Generic Interrupt Controller Architecture. The component is based on r0p0 of the PL390 Interrupt Controller.

The GIC provides support for three interrupt types:

- Software Generated Interrupt (SGI)
- Private Peripheral Interrupt (PPI)
- Shared Peripheral Interrupt (SPI).

Interrupts are programmable so that the following can be set:

- security state for an interrupt
- priority state for an interrupt
- enabling or disabling state for an interrupt
- processors that receive an interrupt.

For a detailed description of the behavior of the PL390 Interrupt Controller, see the component documentation. See *ARM PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual* and the Generic Interrupt Controller Architecture Specification.

Figure 5-41 on page 5-88 shows a view of the component in System Canvas.

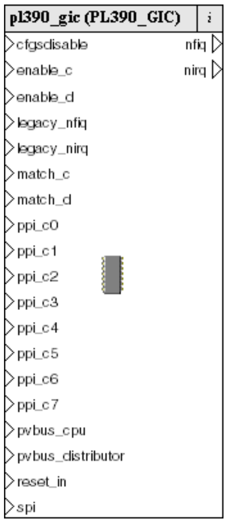


Figure 5-48 PL390_GIC in System Canvas

This component is written in LISA+ and C++.

Ports

Table 5-85 provides a brief description of the PL390_GIC component ports. For more information, see the component documentation.

Table 5-85 PL390_GIC ports

Name	Port protocol	Type	Description
nfiq [8]	Signal	Master	Send out FIQ signal to processor <n>
nirq [8]	Signal	Master	Send out IRQ signal to processor <n>
ppi_c0 [16]	Signal	Slave	Private peripheral interrupt for processor 0 (num_cpus>=1)
ppi_c1 [16]	Signal	Slave	Private peripheral interrupt for processor 1 (num_cpus>=2)
ppi_c2 [16]	Signal	Slave	Private peripheral interrupt for processor 2 (num_cpus>=3)
ppi_c3 [16]	Signal	Slave	Private peripheral interrupt for processor 3 (num_cpus>=4)
ppi_c4 [16]	Signal	Slave	Private peripheral interrupt for processor 4 (num_cpus>=5)
ppi_c5 [16]	Signal	Slave	Private peripheral interrupt for processor 5 (num_cpus>=6)
ppi_c6 [16]	Signal	Slave	Private peripheral interrupt for processor 6 (num_cpus>=7)
ppi_c7 [16]	Signal	Slave	Private peripheral interrupt for processor 7 (num_cpus>=8)
pvbus_cpu	PVBus	Slave	Slave port for connection to processor interface

Table 5-85 PL390_GIC ports (continued)

Name	Port protocol	Type	Description
pvbus_distributor	PVBus	Slave	Slave port for connection to distributor interface
enable_c [8]	Value	Slave	Compared with masked PVBus master id to select processor interface: (master_id & enable_c<n>) == match_c<n>
match_c [8]	Value	Slave	Mask on the PVBus master id to select processor interface: (master_id & enable_c<n>) == match_c<n>
enable_d [8]	Value	Slave	Compared with masked PVBus master id to select distributor interface: (master_id & enable_d<n>) == match_d<n>
match_d [8]	Value	Slave	Mask on the PVBus master id to select distributor interface: (master_id & enable_d<n>) == match_d<n>
legacy_nfiq [8]	Signal	Slave	Legacy FIQ interrupt for processor Interface <n>
legacy_nirq [8]	Signal	Slave	Legacy IRQ interrupt for processor Interface <n>
cfgsdisable	Signal	Slave	Set preventing write accesses to security-critical configuration registers
reset_in	Signal	Slave	Reset signal
spi [988]	Signal	Slave	Shared peripheral interrupt inputs

Additional protocols

The PL390_GIC component has no additional protocols.

Parameters

Table 5-86 provides a description of the configuration parameters for the PL390_GIC component.

Table 5-86 PL390_GIC configuration parameters

Parameter name	Description	Type	Allowed value	Default value
ARCHITECTURE_VERSION	Set architecture version in periph_id register	Integer	0 - 1	1
AXI_IF		Boolean	true/false	true
C_ID_WIDTH	Width of the processor interface master id	Integer	0 - 32	32

Table 5-86 PL390_GIC configuration parameters (continued)

Parameter name	Description	Type	Allowed value	Default value
D_ID_WIDTH	Width of the distributor interface master id	Integer	0 - 32	32
ENABLE_LEGACY_FIQ	Provide legacy fiq interrupt inputs	Boolean	true/false	true
ENABLE_LEGACY_FIQ	Provide legacy irq interrupt inputs	Boolean	true/false	true
ENABLE_PPI_EDGE	ppi edge sensitive	Boolean	true/false	false
ENABLE_TRUSTZONE	Support trust zone	Boolean	true/false	true
INIT_ENABLE_C0 to INIT_ENABLE_C7	init value of ENABLE_C<n>	Integer	-	`0xFFFFFFFF`
INIT_ENABLE_D0 to INIT_ENABLE_D7	init value of ENABLE_D<n>	Integer	-	`0xFFFFFFFF`
INIT_MATCH_C0 to INIT_MATCH_C7	init value of MATCH_C<n>	Integer	-	`0xFFFFFFFF`
INIT_MATCH_D0 to INIT_MATCH_D7	init value of MATCH_D<n>	Integer	-	`0xFFFFFFFF`
NUM_CPU	Number of processor interfaces	Integer	1 - 8	8
NUM_LSPI	Number of lockable SPIs	Integer	0 - 31	31
NUM_PPI	Number of private peripheral interrupts	Integer	0 - 16	16
NUM_PRIORITY_LEVELS	Number of priority levels	Integer	16, 32, 64, 128, 256	256
NUM_SGI	Number of software generated interrupts	Integer	0 - 16	16
NUM_SPI	Number of shared peripheral interrupts	Integer	0 - 988	988

Registers

Table 5-87 on page 5-115 and Table 5-88 on page 5-119 provide a description of the configuration registers for the PL390_GIC component. In these tables <n> corresponds to the number of a processor interface. A processor interface consists of a pair of interfaces: pvbus_cpu and pvbus_distributor. The enable_c<n> and match_c<n> signals identify the originator of a transaction on pvbus_cpu. Similarly, enable_d<n> and match_d<n> signals are used to identify the originator of a transaction on pvbus_distributor.

Note

To reduce compile time, the registers are not available by default. To activate them uncomment one of the following statements in PL390_GIC.lisa:

```
// #define FEW_CADI_REGISTER
// #define ALL_CADI_REGISTER
```

Table 5-87 PL 390_GIC registers: Distributor Interface

Register name	Offset	Access	Description
enable	`0xD0000`	read/write	ICDICR [S]: Interrupt Control Register
enable_ns	`0xD0001`	read/write	ICDICR [NS]: Interrupt Control Register
ic_type	`0xD0008`	read only	ICDDIIR: Distributor Implementer Identification Register
sgi_security_if<n>	`0xDn080`	read/write	ICDISR: SGI Interrupt Security Register Interrupt ID 0-15
ppi_security_if<n>	`0xDn080`	read/write	ICDISR: PPI Interrupt Security Register Interrupt ID 16-31
spi_security_0-31	`0xD0084`	read/write	ICDISR: SPI Interrupt Security Register Interrupt ID 32-63
spi_security_32-63	`0xD0088`	read/write	ICDISR: SPI Interrupt Security Register Interrupt ID 64-95
...			
spi_security_960-987	`0xD00FC`	read/write	ICDISR: SPI Interrupt Security Register Interrupt ID 992-1019
sgi_enable_set_if<n>	`0xDn100`	read only	ICDISER: SGI Enable Set Register Interrupt ID 0-15
ppi_enable_set_if<n>	`0xDn100`	read only	ICDISER: PPI Enable Set Register Interrupt ID 16-31
spi_enable_set_0-31	`0xD0104`	read only	ICDISER: SPI Enable Set Register Interrupt ID 32-63
spi_enable_set_32-63	`0xD0108`	read only	ICDISER: SPI Enable Set Register Interrupt ID 64-95
...			
spi_enable_set_960-987	`0xD017C`	read only	ICDISER: SPI Enable Set Register Interrupt ID 922-1019
sgi_enable_clear_if<n>	`0xDn180`	read only	ICDICER: SGI Enable Clear Register Interrupt ID 0-15
ppi_enable_clear_if<n>	`0xDn182`	read only	ICDICER: SGI Enable Clear Register Interrupt ID 16-31
spi_enable_clear_0-31	`0xDn182`	read only	ICDICER: SGI Enable Clear Register Interrupt ID 32-63
spi_enable_clear_32-63	`0xD0188`	read only	ICDICER: SGI Enable Clear Register Interrupt ID 32-63

Table 5-87 PL 390_GIC registers: Distributor Interface (continued)

Register name	Offset	Access	Description
...			
spi_enable_clear_960-987	`0xD01FC`	read only	ICDICER: SGI Enable Clear Register Interrupt ID 32-63
sgi_pending_set_if<n>	`0xDn200`	read only	ICDISPR: SGI Pending Set Register Interrupt ID 0-15
ppi_pending_set_if<n>	`0xDn202`	read only	ICDISPR: PPI Pending Set Register Interrupt ID 16-31
spi_pending_set_0-31	`0xD0204`	read only	ICDISPR: SPI Pending Set Register Interrupt ID 32-63
spi_pending_set_32-63	`0xD0208`	read only	ICDISPR: SPI Pending Set Register Interrupt ID 64-95
...			
spi_pending_set_960-987	`0xD027C`	read only	ICDISPR: SPI Pending Set Register Interrupt ID 992-1019
sgi_pending_clear_if<n>	`0xDn280`	read only	ICDICPR: SGI Pending Clear Register Interrupt ID 0-15
ppi_pending_clear_if<n>	`0xDn282`	read only	ICDICPR: SGI Pending Clear Register Interrupt ID 16-31
spi_pending_clear_0-31	`0xDn284`	read only	ICDICPR: SGI Pending Clear Register Interrupt ID 32-63
spi_pending_clear_32-63	`0xD0288`	read only	ICDICPR: SGI Pending Clear Register Interrupt ID 64-95
...			
spi_pending_clear_960- 987	`0xD037C`	read only	ICDICPR: SGI Pending Clear Register Interrupt ID 992-1019
priority_sgi_if<n>_0-3	`0xDn400`	read/write	ICDIPR: SGI Priority Level Register Interrupt ID 0-3
...			
priority_sgi_if<n>_12-15	`0xDn40C`	read/write	ICDIPR: SGI Priority Level Register Interrupt ID 12-15
priority_ppi_if<n>_0-3	`0xDn410`	read/write	ICDIPR: PPI Priority Level Register Interrupt ID 16-19
...			
priority_ppi_if<n>_12-15	`0xDn41C`	read/write	ICDIPR: PPI Priority Level Register Interrupt ID 28-31
priority_spi_0-3	`0xD0420`	read/write	ICDIPR: PPI Priority Level Register Interrupt ID 28-31
...			
priority_spi_984-987	`0xD07F8`	read/write	ICDIPR: PPI Priority Level Register Interrupt ID 28-31

Table 5-87 PL 390_GIC registers: Distributor Interface (continued)

Register name	Offset	Access	Description
target_sgi_i<n>0_0-3	`0xDn800`	read only	ICDIPTR: SGI Target Register Interrupt ID 0-3
...			
target_sgi_i<n>0_12-15	`0xDn80C`	read only	ICDIPTR: SGI Target Register Interrupt ID 12-15
target_ppi_i<n>0_0-3	`0xDn810`	read only	ICDIPTR: PPI Target Register Interrupt ID 16-19
...			
target_ppi_i<n>0_12-15	`0xDn81C`	read only	ICDIPTR: PPI Target Register Interrupt ID 28-31
target_spi_0-3	`0xD0820`	read/write	ICDIPTR: SPI Target Register Interrupt ID 32-35
...			
target_spi_984-987	`0xD0BF8`	read/write	ICDIPTR: SPI Target Register Interrupt ID 32-35
sgi_config_if<n>_0-15	`0xDnC00`	read only	ICDICR: SGI Interrupt Configuration Register Interrupt ID 0-15
ppi_config_if<n>_0-15	`0xDnC04`	read only	ICDICR: SGI Interrupt Configuration Register Interrupt ID 0-15
spi_config_0-15	`0xD0C08`	read/write	ICDICR: SPI Interrupt Configuration Register Interrupt ID 32-47
...			
spi_config_976-987	`0xD0CFC`	read/write	ICDICR: SPI Interrupt Configuration Register Interrupt ID 1008-1019
ppi_if<n>	`0xDnD00`	read only	ICDICR: SPI Interrupt Configuration Register Interrupt ID 1008-1019
spi_0-31	`0xD0D04`	read only	ICDICR: SPI Interrupt Configuration Register Interrupt ID 1008-1019
...			
spi_960-987	`0xD0D7C`	read only	SPI Status Register Interrupt ID 992-1019
legacy_int<n>	`0xDnDD0`	read only	Legacy Interrupt Register
match_d<n>	`0xDnDE0`	read only	Match Register
enable_d<n>	`0xDnDE4`	read only	Enable Register

Table 5-87 PL 390_GIC registers: Distributor Interface (continued)

Register name	Offset	Access	Description
sgi_control	`0xD0F00`	read/write	ICDSGIR: Software Generated Interrupt Register
periph_id_d_8	`0xD0FC0`	read only	Peripheral Identification Register 8
periph_id_d_4-7	`0xD0FD0`	read only	Peripheral Identification Register [7:4]
periph_id_d_0-3	`0xD0FE0`	read only	Peripheral Identification Register [7:4]
component_id	`0xD0FF0`	read only	PrimeCell Identification Register

Debug features

The PL390_GIC component provides some registers for functional verification and integration testing.

For more information see the Technical Reference Manual.

Verification and testing

The PL390_GIC has been run against the RTL validation suite and has been successfully used in validation platforms.

Performance

The PL390_GIC component is not expected to significantly affect the performance of a PV system.

Library dependencies

The PL390_GIC component has no dependencies on external libraries.

5.4.27 SP804_Timer component

The SP804_Timer component is a programmer's view model of the ARM Dual-Timer module. For a detailed description of the behavior of the SP804 timer, see other documentation. See the *ARM Dual-Timer Module (SP804) Technical Reference Manual*.

Figure 5-49 shows a view of the component in System Canvas.

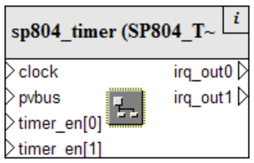


Figure 5-49 SP804_Timer in System Canvas

This component is written in LISA+.

Ports

Table 5-88 provides a brief description of the ports. For more information, see the component documentation.

Table 5-88 SP804_Timer ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
irq_out0	Signal	Master	Interrupt signaling
irq_out1	Signal	Master	Interrupt signaling
clock	ClockSignal	Slave	Clock input, typically 1MHz, driving master count rate
timer_en[0]	ClockRateControl	Slave	Port for changing the rate of timer 1
timer_en[1]	ClockRateControl	Slave	Port for changing the rate of timer 2

Additional protocols

The SP804_Timer component has no additional protocols.

Parameters

The SP804_Timer component has no parameters.

Registers

Table 5-89 provides a description of the configuration registers for the SP804_Timer component.

Table 5-89 SP804_Timer registers

Register name	Offset	Access	Description
Timer1Load	`0x000`	read/write	Data register
Timer1Value	`0x004`	read only	Value register
Timer1Control	`0x008`	read/write	Load register
Timer1IntClr	`0x00C`	write only	Interrupt clear register
Timer1RIS	`0x010`	read only	Raw interrupt status register
Timer1MIS	`0x014`	read only	Masked interrupt status register
Timer1BGLoad	`0x018`	read/write	Background load register
Timer2Load	`0x020`	read/write	Data register
Timer2Value	`0x024`	read only	Value register
Timer2Control	`0x028`	read/write	Load register
Timer2IntClr	`0x02C`	write only	Interrupt clear register

Table 5-89 SP804_Timer registers (continued)

Register name	Offset	Access	Description
Timer2RIS	`0x030`	read only	Raw interrupt status register
Timer2MIS	`0x034`	read only	Masked interrupt status register
Timer2BGLoad	`0x038`	read/write	Background load register

Debug features

The SP804_Timer component has no debug features.

Verification and testing

The SP804_Timer component has been tested as part of the SMLT component.

Performance

The SP804_Timer component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SP804_Timer component has no dependencies on external libraries.

5.4.28 SP805_Watchdog component

The SP805_Watchdog component is a programmer's view model of the ARM Watchdog timer module. For a detailed description of the behavior of the SP805 timer, see the component documentation. See the *ARM Watchdog Module (SP805) Technical Reference Manual*.

Figure 5-50 shows a view of the component in System Canvas.

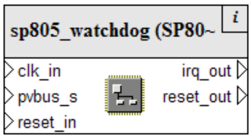


Figure 5-50 SP805_Watchdog in System Canvas

This component is written in LISA+.

Ports

Table 5-90 provides a brief description of the SP805_Watchdog component ports. For more information, see the component documentation. See the *ARM Watchdog Module (SP805) Technical Reference Manual*.

Table 5-90 SP805_Watchdog ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Slave port for connection to PV bus master/decoder
irq_out	Signal	Master	Interrupt signaling
reset_out	Signal	Master	Reset signaling
clk_in	ClockSignal	Slave	Clock input, typically 1MHz, driving master count rate
reset_in	Signal	Master	Master reset signal

Additional protocols

The SP805_Watchdog component has no additional protocols.

Parameters

Table 5-91 provides a description of the configuration parameter for the SP805_Watchdog component.

Table 5-91 SP805_Watchdog configuration parameters

Parameter name	Description	Type	Allowed value	Default value
simhalt	If true, Halt simulation instead of signalling reset	Boolean	true/false	false

Registers

Table 5-92 provides a description of the configuration registers for the SP805_Watchdog component.

Table 5-92 SP805_Watchdog registers

Register name	Offset	Access	Description
SP805_WDOG_Load	`0x000`	read/write	Load register
SP805_WDOG_VALUE	`0x004`	read only	Value register
SP805_WDOG_CONTROL	`0x008`	read/write	Control register
SP805_WDOG_INT_CLR	`0x00C`	write only	Clear interrupt register

Table 5-92 SP805_Watchdog registers (continued)

Register name	Offset	Access	Description
SP805_WDOG_RAW_INT_STATUS	`0x00C`	read only	Raw interrupt status register
SP805_WDOG_MASKED_INT_STATUS	`0x010`	read only	Masked interrupt status register
SP805_WDOG_LOCK	`0xC00`	read/write	Register access lock register

Debug features

The SP805_Watchdog component has no debug features.

Verification and testing

The SP805_Watchdog component has been tested as part of an integrated platform.

Performance

The SP805_Watchdog component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SP805_Watchdog component has no dependencies on external libraries.

5.4.29 SP810_SysCtrl component

The SP810_SysCtrl component is a programmer's view model of the System Controller in the PrimeXsys® subsystem. For a detailed description of the behavior of the SP810, see other documentation. See the *PrimeXsys System Controller (SP810) Technical Reference Manual*.

Figure 5-51 shows a view of the component in System Canvas.

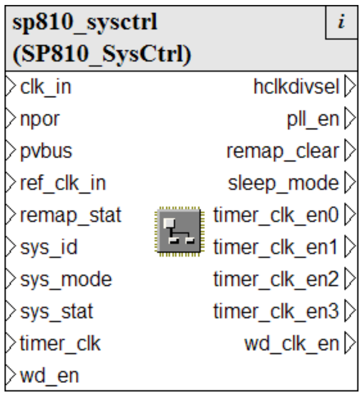


Figure 5-51 SP810_SysCtrl in System Canvas

This component is written in LISA+.

Ports

Table 5-93 provides a brief description of the SP810_SysCtrl component ports. For more information, see the component documentation.

Table 5-93 SP810_SysCtrl ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
clk_in	ClockSignal	Slave	Clock input
ref_clk_in	ClockSignal	Slave	Clock source used by the Timer and Watchdog modules
timer_clk_en[0]	ClockRateControl	Master	Timer clock enable 0
timer_clk_en[1]	ClockRateControl	Master	Timer clock enable 1
timer_clk_en[2]	ClockRateControl	Master	Timer clock enable 2
timer_clk_en[3]	ClockRateControl	Master	Timer clock enable 3
remap_clear	StateSignal	Master	Remap clear request output
npor[a]	Signal	Slave	Power on reset
remap_stat[a]	StateSignal	Slave	Remap status input
sys_mode[a]	ValueState	Slave	Present system mode
sys_stat[a]	ValueState	Slave	System status input
wd_en[a]	Signal	Slave	Watchdog module enable input
hclkdivsel[a]	ValueState	Master	Define the processor clock/bus clock ratio
pll_en[a]	Signal	Master	PLL enable output
sleep_mode[a]	Signal	Master	Control clocks for SLEEP mode
wd_clk_en[a]	Signal	Master	Watchdog module clock enable output

a. Not fully implemented. Using this port has unpredictable results.

Additional protocols

The SP810_SysCtrl component has no additional protocols.

Parameters

Table 5-94 provides a description of the configuration parameters for the SP810_SysCtrl component.

Table 5-94 SP810_SysCtrl configuration parameters

Parameter name	Description	Type	Allowed value	Default value
sysid	System identification register	Integer	-	`0x00000000`
use_s8	Toggles whether switch S8 is enabled	Boolean	true/false	false

Registers

Table 5-95 provides a description of the configuration registers for the SP810_SysCtrl component. See the component documentation for further details.

Table 5-95 SP810_SysCtrl registers

Register name	Offset	Access	Description
SCCTRL	`0x0`	read/write	System control
SCSYSSTAT	`0x4`	read/write	System status
SCIMCTRL	`0x8`	read/write	Interrupt mode control
SCIMSTAT	`0xC`	read/write	Interrupt mode status
SCXTALCTRL	`0x10`	read/write	Crystal control
SCPLLCTRL	`0x14`	read/write	PLL control
SCPLLFCTRL	`0x18`	read/write	PLL frequency control
SCPERCTRL0	`0x1C`	read/write	Peripheral control
SCPERCTRL1	`0x20`	read/write	Peripheral control
SCPEREN	`0x24`	write only	Peripheral clock enable
SCPERDIS	`0x28`	write only	Peripheral clock disable
SCPERCLKEN	`0x2C`	read only	Peripheral clock enable status
SCPERSTAT	`0x30`	read only	Peripheral clock status
SCSysID0	`0xEE0`	read only	System identification 0
SCSysID1	`0xEE4`	read only	System identification 1
SCSysID2	`0xEE8`	read only	System identification 2
SCSysID3	`0xEEC`	read only	System identification 3
SCITCR	`0xF00`	read/write	Integration test control
SCITIR0	`0xF04`	read/write	Integration test input 0

Table 5-95 SP810_SysCtrl registers (continued)

Register name	Offset	Access	Description
SCITIR1	`0xF08`	read/write	Integration test input 1
SCITOR	`0xF0C`	read/write	Integration test output
SCCNTCTRL	`0xF10`	read/write	Counter test control
SCCNTDATA	`0xF14`	read/write	Counter data
SCCNTSTEP	`0xF18`	write only	Counter step
SCPeriphID0	`0xFE0`	read only	Peripheral identification 0
SCPeriphID1	`0xFE4`	read only	Peripheral identification 1
SCPeriphID2	`0xFE8`	read only	Peripheral identification 2
SCPeriphID3	`0xFEC`	read only	Peripheral identification 3
SPCellID0	`0xFF0`	read only	PrimeCell identification 0
SPCellID1	`0xFF4`	read only	PrimeCell identification 1
SPCellID2	`0xFF8`	read only	PrimeCell identification 2
SPCellID3	`0xFFC`	read only	PrimeCell identification 3

Debug features

The SP810_SysCtrl component has no debug features.

Verification and testing

The SP810_SysCtrl component has been tested as part of the Emulation Board model.

Performance

The SP810_SysCtrl component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SP810_SysCtrl component has no dependencies on external libraries.

5.4.30 TZIC component

The TZIC component is a programmer's view model of the *TrustZone Interrupt Controller* (TZIC) as described elsewhere. See *AMBA 3 TrustZone Interrupt Controller (SP890) Revision: r0p0 Technical Overview (ARM DTO 0013)*.

The TZIC provides a software interface to the secure interrupt system in a TrustZone design. It provides secure control of the nFIQ and masks out the interrupt sources chosen for nFIQ from the interrupts that are passed onto a non-secure interrupt controller.

Figure 5-52 on page 5-126 shows a view of the component in System Canvas.

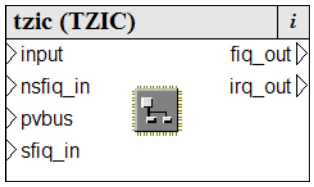


Figure 5-52 TZIC in System Canvas

This component is written in LISA+.

Ports

Table 5-96 provides a brief description of the ports. More details are in the Technical Overview. See the *AMBA 3 TrustZone Interrupt Controller (SP890) Revision: r0p0 Technical Overview*.

Table 5-96 TZIC ports

Name	Port protocol	Type	Description
nsfiq_in	Signal	Slave	Connects to the nFIQ output of the non-secure interrupt controller
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
sfiq_in	Signal	Slave	Daisy chaining secure FIQ input, otherwise connects to logic 1 if interrupt controller not daisy chained
input[32]	Signal	Slave	32 interrupt input sources
fiq_out	Signal	Master	FIQ interrupt to processor
irq_out[32]	Signal	Master	32 IRQ output ports

Additional protocols

The TZIC component has no additional protocols.

Parameters

The TZIC component has no parameters.

Registers

Table 5-97 provides a description of the configuration registers for the TZIC component.

Table 5-97 TZIC registers

Register name	Offset	Access	Description
FIQStatus	`0x000`	read only	Provide the status of the interrupts after FIQ masking.
RawIntr	`0x004`	read only	Provide the status of the source interrupts and software interrupts to the interrupt controller.
IntSelect	`0x008`	read/write	Select whether the corresponding input source can be used to generate an FIQ or whether it passes through to TZICIRQOUT.
FIQEnable	`0x00C`	read/write	Enable the corresponding FIQ-selected input source, which can then generate an FIQ.
FIQEnClear	`0x010`	write only	Clear bits in the TZICFIQEnable register.
Bypass	`0x014`	read/write	Enable nNSFIQIN to be routed directly to FAQ, bypassing all TZIC logic. Only the least significant bit is used.
Protection	`0x018`	read/write	Enable or disable protected register access, stopping register accesses when the processor is in user mode.
Lock	`0x01C`	write only	Enable or disable all other register write access.
LockStatus	`0x020`	read only	Provide the lock status of the TZIC registers.

Debug features

The TZIC component has no debug features.

Verification and testing

The TZIC component has been tested separately using its own test suite.

The SP890 TZIC component has been tested inside the SMLT component.

The FIQ has been tested under the secure environment.

Performance

The TZIC component is not expected to significantly affect the performance of a PV system.

Library dependencies

The TZIC component has no dependencies on external libraries.

5.4.31 TZMPU component

The TZMPU component is a programmer's view model of the *TrustZone Memory Protection Unit* (TZMPU). The TZMPU has been superseded by the *TrustZone Address Space Controller* (TZASC), so other documentation on the component is limited to internal specifications.

The TZMPU component is conceptualized as an address filter. You can program a configurable number of regions to grant different access rights for secure and non-secure AXI transactions to different areas of memory.

Figure 5-53 shows a view of the component in System Canvas.

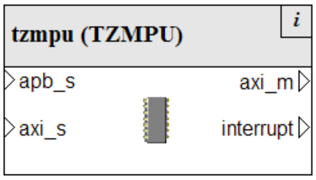


Figure 5-53 TZMPU in System Canvas

This component is written in LISA+.

Ports

Table 5-98 provides a brief description of the ports.

Table 5-98 TZMPU ports

Name	Port protocol	Type	Description
apb_s	PVBus	Slave	APB slave port
axi_s	PVBus	Slave	AXI slave port
axi_m	PVBus	Master	AXI master port
interrupt	Signal	Master	Interrupt signalling port

Additional protocols

The TZMPU component has no additional protocols.

Parameters

Table 5-99 provides a description of the configuration parameters for the TZMPU component.

Table 5-99 TZMPU configuration parameters

Parameter name	Description	Type	Allowed value	Default value
numRegions	Number of memory regions	Integer	2 to 16, in increments of 1	16
axiAddrWidth	AXI address width, in bits	Integer	32 to 64, in increments of 1 bit	32

Registers

Table 5-100 provides a description of the configuration registers for the TZMPU component.

Table 5-100 TZMPU registers

Register name	Offset	Access	Description
configuration	`0x000`	read only	Defines AXI address width and number of regions.
action	`0x004`	read/write	Action for access permission failures.
lockdown_range	`0x008`	read/write	Region lockdown range.
lockdown_enable	`0x00C`	read/write	Region lockdown select.
int_status	`0x010`	read only	Indicates status of permission failures.
int_clear	`0x014`	write only	Clear unserviced interrupts.
fail_address_low	`0x020`	read only	Low order 32 bits of address failed.
fail_address_high	`0x024`	read only	High order 32 bits of address failed.
fail_control	`0x028`	read only	Access type of failed accesses.
fail_id	`0x02C`	read only	Master AXI ID of failed access.
region_base_lo_0	`0x100`	read only, or read/write[a]	Protection region 0 base low address.
region_base_hi_0	`0x104`	read only, or read/write[a]	Protection region 0 base high address.
region_attr_sz_0	`0x108`	read only, or read/write[a]	Protection region 0 size.
region_base_lo_1	`0x110`	read only, or read/write[a]	Protection region 1 base low address.

Table 5-100 TZMPU registers (continued)

Register name	Offset	Access	Description
region_base_hi_1	`0x114`	read only, or read/write[a]	Protection region 1 base high address.
region_attr_sz_1	`0x118`	read only, or read/write[a]	Protection region 1 size.
…	…	…	…
region_base_lo_15	`0x1F0`	read only, or read/write[a]	Protection region 15 base low address.
region_base_hi_15	`0x1F4`	read only, or read/write[a]	Protection region 15 base high address.
region_attr_sz_15	`0x1F8`	read only, or read/write[a]	Protection region 15 size.
itcr	`0xE00`	read/write	Integration test control register.
itip	`0xE04`	read only	Integration test input register.
itop	`0xE08`	read/write	Integration test output register.
periph_id_0	`0xFE0`	read only	Peripheral identification register.
periph_id_1	`0xFE4`	read only	Peripheral identification register.
periph_id_2	`0xFE8`	read only	Peripheral identification register.
periph_id_3	`0xFEC`	read only	Peripheral identification register.
pcell_id_0	`0xFF0`	read only	PrimeCell identification register 0.
pcell_id_1	`0xFF4`	read only	PrimeCell identification register 1.
pcell_id_2	`0xFF8`	read only	PrimeCell identification register 2.
pcell_id_3	`0xFFC`	read only	PrimeCell identification register 3.

a. Regions specified in the lockdown_range register are locked down and are read only, otherwise they are read/write.

Debug features

The TZMPU component has no debug features.

Verification and testing

The TZMPU component has been tested separately using its own test suite.

Performance

The TZMPU component is not expected to significantly affect the performance of a PV system.

Library dependencies

The TZMPU component has no dependencies on external libraries.

5.4.32 RemapDecoder

The RemapDecoder component provides the low memory flash/RAM remap behavior of the example systems.

Figure 5-54 shows a view of the component in System Canvas.

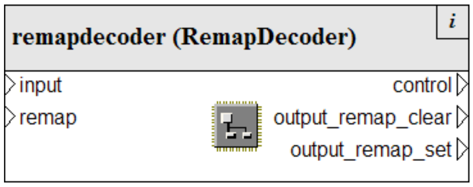


Figure 5-54 RemapDecoder in System Canvas

This model is written in LISA+.

Ports

Table 5-101 provides a brief description of the ports of the RemapDecoder component. For more information, see the VE hardware documentation.

Table 5-101 RemapDecoder ports

Name	Port protocol	Type	Description
input	PVBus	Slave	For connection to PV bus master/decoder
output_remap_set	PVBus	Master	For connection to a component addressable with remap set
output_remap_clear	PVBus	Master	For connection to a component addressable with remap clear
remap	StateSignal	Slave	Input permitting control of remap state
control	TZSwitchControl		Internal port. Not for use.

Additional protocols

The RemapDecoder component has no additional protocols.

Parameters

The RemapDecoder component has no parameters.

Registers

The RemapDecoder component has no registers

Debug features

The RemapDecoder component has no debug features.

Verification and testing

The RemapDecoder component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The RemapDecoder component is not expected to significantly affect the performance of a PV system.

Library dependencies

The RemapDecoder component has no dependencies on external libraries.

5.4.33 BP135_AXI2APB component

The BP135_AXI2APB component is a programmer's view model of the ARM AXI2APB Bridge peripheral. The component, when configured by another component such as the BP141_TZPC, permits control of secure access to up to 16 PrimeCell peripherals. For further information, see the component documentation. See *PrimeCell Infrastructure AMBA 3 AXI to AMBA 3 APB Bridge (BP135) Revision: r0p0 Technical Overview*.

The PVBus makes no distinction between AXI and APB bus protocols. For simple models, you do not have to use a bridge peripheral when connecting devices to a processor.

Figure 5-55 shows a view of the component in System Canvas.

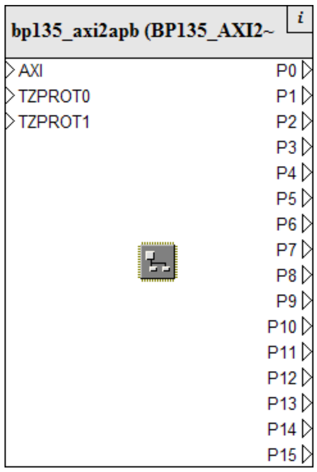


Figure 5-55 BP135_AXI2APB in System Canvas

This component is written in LISA+.

Ports

Table 5-102 provides a brief description of the ports for the BP135_AXI2APB component. For more information, see the component documentation.

Table 5-102 BP135_AXI2APB ports

Name	Port protocol	Type	Description
AXI	PVBus	Slave	Slave port for connection to a PVBus master/decoder
P0 - P15	PVBus	Master	Master ports for connection to PVBus slaves
TZPROT0 TZPROT1	Value	Slave	Control ports for selecting secure state of slaves

Additional protocols

The BP135_AXI2APB component has no additional protocols.

Parameters

The BP135_AXI2APB component has no parameters.

Registers

The BP135_AXI2APB component has no registers.

Debug features

The BP135_AXI2APB component has no debug features.

Verification and testing

The BP135_AXI2APB component has been tested as part of an integrated platform.

Performance

The BP135_AXI2APB component should not significantly impact performance of a PV system.

Library dependencies

The BP135_AXI2APB component is dependent on the BP135TZSwitchControl component which is used to decode the protection selection bits provided by the TZPROT ports and control the protection routing of the embedded TZSwitch routing components. See *TZSwitch component* on page 5-12.

5.4.34 BP141_TZMA component

The BP141_TZMA component provides a programmer's view of the BP141 *TrustZone Memory Adapter* (TZMA). This permits a single physical memory cell of up to 2MB to be shared between a secure and non-secure storage area. The partitioning between these areas is flexible.

The BP141_TZMA routes transactions according to:

- the memory region that they are attempting to access

- the security mode of the transaction.

The BP141_TZMA fixes the base address of the secure region to the base address of the decode space. It uses the R0SIZE[9:0] input to configure the size of the secure region in 4KB increments up to a maximum of 2MB.

Table 5-103 lists the response of the BP141_TZMA for all combinations of address range and transfer security types. `TZMEMSIZE` is the maximum addressing range of the memory as defined by that parameter. See Table 5-105 on page 5-135. By default, `TZMEMSIZE` is set to 2MB. AxADDR is the offset address that the transactions want to access.

Table 5-103 BP141_TZMA security control

AxADDR	Memory Region	Non-secure Transfer	Secure Transfer
AxADDR < R0Size	secure, R0	illegal	legal
R0SIZE <= AxADDR and AxADDR < `TZMEMSIZE`	non-secure, R1	legal	legal
AxADDR => `TZMEMSIZE`	no access	illegal	illegal

Figure 5-56 shows a view of the BP141_TZMA component in System Canvas.

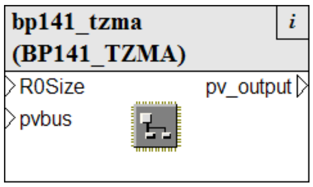


Figure 5-56 BP141_TZMA in System Canvas

This component is written in LISA+.

Ports

Table 5-104 provides a brief description of the BP141_TZMA ports. Further information is available in separate documentation. See the *ARM PrimeCell Infrastructure AMBA 3 AXI TrustZone Memory Adapter (BP141) Technical Overview*.

Table 5-104 BP141_TZMA ports

Name	Port protocol	Type	Description
R0Size	Value	Slave	A software interface that is driven from the *TrustZone Protection Controller* (TZPC), setting the secure region size by bits [9:0]
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
pv_output	PVBus	Master	Routed PVBus output

Parameters

Table 5-105 provides a description of the configuration parameters for the BP141_TZMA component.

Table 5-105 BP141_TZMA configuration parameters

Parameter name	Description	Type	Allowed value	Default value
TZMEMSIZE	Sets the maximum size of the addressable memory[a]	uint32_t	-	`0x200000`
TZSEGSIZE	Configures the size of the region allocated for secure access for each increment of the R0Size value	uint32_t	multiples of 4096	4096
TZSECROMSIZE	Configures the initial region configuration value so that an external secure control is not required	uint32_t	-	`0x200`

a. This parameter is deprecated and is provided for backwards compatibility.

Registers

The BP141_TZMA component has no registers.

Debug features

The BP141_TZMA component has no debug features.

Verification and testing

The BP141_TZMA component has been tested separately using its own test suite and as part of the VE example system using VE test suites and by booting operating systems.

Performance

The BP141_TZMA component is not expected to significantly affect the performance of a PV system.

Library dependencies

The BP141_TZMA component has no dependencies on external libraries.

5.4.35 BP147_TZPC component

The BP147_TZPC component is a programmer's view model of the ARM *TrustZone Protection Controller* (TZPC) peripheral. The TZPC provides a software interface to the protection bits in a secure system in a TrustZone design. For a detailed description of the behavior of the BP147 TZPC, see the component documentation. See the *PrimeCell Infrastructure AMBA 3 TrustZone Protection Controller (BP147) Revision: r0p0 Technical Overview*.

Figure 5-57 shows a view of the component in System Canvas.

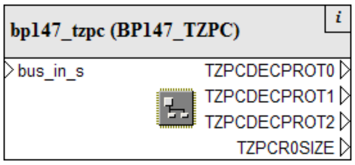


Figure 5-57 BP147_TZPC in System Canvas

This component is written in LISA+.

Ports

Table 5-106 provides a brief description of the BP147_TZPC ports. For more information, see the component documentation.

Table 5-106 BP147_TZPC ports

Name	Port protocol	Type	Description
bus_in_s	PVBus	Slave	Slave port for connection to PV bus master/decoder
TZPCDECPROT0	Value	Master	Output decode protection 0 status
TZPCDECPROT1	Value	Master	Output decode protection 1 status
TZPCDECPROT2	Value	Master	Output decode protection 2 status
TZPCR0SIZE	Value	Master	Output secure RAM region size

Additional protocols

The BP147_TZPC component has no additional protocols.

Parameters

The BP147_TZPC component has no parameters.

Registers

Table 5-107 provides a description of the configuration registers for the BP147_TZPC component.

Table 5-107 BP147_TZPC registers

Register name	Offset	Access	Description
R0SIZE	`0x000`	read/write	Secure RAM region size register
DECPROT0Stat	`0x800`	read only	Decode protection 0 status register
DECPROT0Set	`0x804`	write only	Decode protection 0 set register
DECPROT0Clr	`0x808`	write only	Decode protection 0 clear register
DECPROT1Stat	`0x80C`	read only	Decode protection 1 status register
DECPROT1Set	`0x810`	write only	Decode protection 1 set register
DECPROT1Clr	`0x814`	write only	Decode protection 1 clear register
DECPROT2Stat	`0x818`	read only	Decode protection 2 status register
DECPROT2Set	`0x81C`	write only	Decode protection 2 set register
DECPROT2Clr	`0x820`	write only	Decode protection 2 clear register

Debug features

The BP147_TZPC component has no debug features.

Verification and testing

The BP147_TZPC has been tested using a unit test suite.

Performance

The BP147_TZPC component is not expected to significantly affect the performance of a PV system.

Library dependencies

The BP147_TZPC component has no dependencies on external libraries.

5.4.36 AndGate component

The AndGate component implements a logical AND of two Signal input ports to generate a single output Signal. You can use this, for example, to combine two interrupt signals.

Figure 5-58 shows a view of the component in System Canvas.

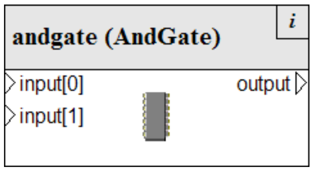


Figure 5-58 AndGate in System Canvas

This component is written in LISA+.

Ports

Table 5-108 provides a brief description of the AndGate component ports.

Table 5-108 AndGate ports

Name	Port protocol	Type	Description
input[0]	Signal	Slave	First input signal
input[1]	Signal	Slave	Second input signal
output	Signal	Master	Combined output signal

Additional protocols

The AndGate component has no additional protocols.

Parameters

The AndGate component has no parameters.

Registers

The AndGate component has no registers.

Debug features

The AndGate component has no debug features.

Verification and testing

The AndGate component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The AndGate component is not expected to significantly affect the performance of a PV system.

Library dependencies

The AndGate component has no dependencies on external libraries.

5.4.37 OrGate component

The OrGate component implements a logical OR of two Signal input ports to generate a single output Signal. For example, you can use this component to combine two interrupt signals.

Figure 5-59 shows a view of the component in System Canvas.

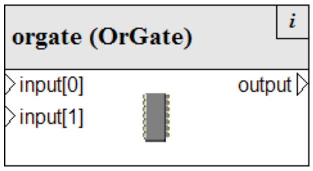


Figure 5-59 OrGate in System Canvas

This component is written in LISA+.

Ports

Table 5-109 provides a brief description of the ports in the OrGate component.

Table 5-109 OrGate ports

Name	Port protocol	Type	Description
input[0]	Signal	Slave	First input signal
input[1]	Signal	Slave	Second input signal
output	Signal	Master	Combined output signal

Additional protocols

The OrGate component has no additional protocols.

Parameters

The OrGate component has no parameters.

Registers

The OrGate component has no registers.

Debug features

The OrGate component has no debug features.

Verification and testing

The OrGate component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The OrGate component is not expected to significantly affect the performance of a PV system.

Library dependencies

The OrGate component has no dependencies on external libraries.

5.4.38 ICS307 component

The ICS307 component implements a programmer's view model of an ICS307 clock divider. It can be used to convert the rate of one ClockSignal to another ClockSignal by application of configurable multiplier, divider and scale values.

The Divider ratio can be set by startup parameters or at runtime by a configuration port. Changes to the input ClockSignal rate and divider ratio are reflected immediately by the output ClockSignal ports.

The divisor ratio is determined by three values:

- `vdw`
- `rdw`
- `od`.

To calculate the divisor ratio, use:

```
Divisor = ((rdw+2) * scale) / (2 * (vdw+8))
```

where `scale` is derived from a table indexed by `od`, shown in Table 5-110:

Table 5-110 od to scale conversion

od	scale
0	10
1	2
2	8
3	4
4	5
5	7
6	3
7	6

The default values of `vdw`, `rdw` and `od` are 4, 6 and 3 to give a default divisor rate of:

```
((6+2) * 4) / (2 * (4+8)) = 4/3
```

Figure 5-60 shows a view of the component in System Canvas.

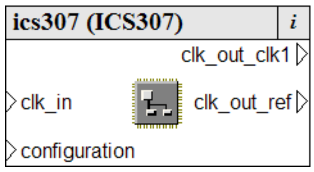


Figure 5-60 ICS307 in System Canvas

This component is written in LISA+.

Ports

Table 5-111 provides a brief description of the ports.

Table 5-111 ICS307 ports

Name	Port protocol	Type	Description
clk_in	ClockSignal	Slave	Master clock rate
clk_out_clk1	ClockSignal	Master	Modified clock rate
clk_out_ref	ClockSignal	Master	Pass through of master clock rate for divider chaining
configuration	ICS307Configuration	Slave	Configuration port for setting divider ratio dynamically

ICS307Configuration

The ICS307 has one additional protocol.

The ICS307Configuration protocol permits you to set the divider ratio of an ICS307 component at runtime. The output clock rate is altered accordingly and any dependent components should react to the clock rate change according to their defined behavior.

`setConfiguration(uint32_t vdw, uint32_t rdw,uint32_t od): void` Sets the parameters used to derive the clock divider ratio.

vdw	must be in the range 0-255
rdw	must be in the range 0-255
od	must be in the range 0-7.

Parameters

Table 5-112 provides a description of the configuration parameters for the ICS307 component.

Table 5-112 ICS307 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
vdw	Used to calculate clock divider ratio	Integer	0-255	4
rdr	Used to calculate clock divider ratio	Integer	0-255	6
od	Used to calculate clock divider ratio	Integer	0-7	3

Registers

The ICS307 component has no registers.

Debug features

The ICS307 component has no debug features.

Verification and testing

The ICS307 component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The ICS307 component is not expected to significantly affect the performance of a PV system. However, modifying the ICS307 timing parameters is relatively slow, so you are discouraged from doing so too often.

Library dependencies

The ICS307 component has no dependencies on external libraries.

5.4.39 ElfLoader component

The ElfLoader component provides an alternative method of loading an elf file into arbitrary locations in the system.

Figure 5-61 shows a view of the component in System Canvas.

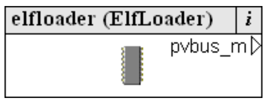


Figure 5-61 ElfLoader in System Canvas

This component is written in LISA+.

Ports

Table 5-113 provides a brief description of the ports in the ElfLoader component.

Table 5-113 ElfLoader ports

Name	Port protocol	Type	Description
pvbus_m	PVBus	Master	Master port for all memory accesses

Additional protocols

The ElfLoader component has no additional protocols.

Parameters

Table 5-116 on page 5-144 provides a description of the configuration parameters for the ElfLoader component.

Table 5-114 ElfLoader configuration parameters

Parameter name	Description	Type	Allowed value	Default value
elf	ELF file	String	-	[empty string]
lfile	Load file for large address mapping	String	-	[empty string]
ns_copy	Copy whole file to NS memory space	Boolean	`true` or `false`	`true`

Registers

The ElfLoader component has no registers.

Debug features

The ElfLoader component has no debug features.

Verification and testing

The ElfLoader component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The ElfLoader component is not expected to significantly affect the performance of a PV system.

Library dependencies

The ElfLoader component has no dependencies on external libraries.

5.4.40 FlashLoader component

The FlashLoader component complements the IntelStrataFlashJ3 component by providing a means to initialize the contents of up to four Flash components in sequence from a single host flash image file. See also *IntelStrataFlashJ3 component* on page 5-145.

Figure 5-62 shows a view of the component in System Canvas.

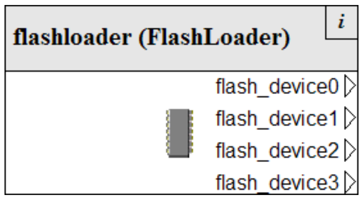


Figure 5-62 FlashLoader in System Canvas

This component is written in LISA+.

Ports

Table 5-115 provides a brief description of the ports in the FlashLoader component.

Table 5-115 FlashLoader ports

Name	Port protocol	Type	Description
flash_device0 flash_device1 flash_device2 flash_device3	FlashLoaderPort	Master	Used to program a flash device

FlashLoaderPort

The FlashLoader component has one additional protocol.

This protocol can be used to initialize the flash contents at model startup and save flash contents to a file when the model terminates. It is typically connected to a FlashLoader component. See *FlashLoader component* on page 5-143. The behaviors are:

`loadFlashFile(FlashLoader*) : uint32`

Initiate loading of the flash contents.

`saveFlashFile(FlashLoader*) : uint32`

Save the flash contents to a file.

Parameters

Table 5-116 provides a description of the configuration parameters for the FlashLoader component.

Table 5-116 FlashLoader configuration parameters

Parameter name	Description	Type	Allowed value	Default value
fname		String	-	[empty string]
fnameWrite		String	-	[empty string]

Registers

The FlashLoader component has no registers.

Debug features

The FlashLoader component has no debug features.

Verification and testing

The FlashLoader component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The FlashLoader component is not expected to significantly affect the performance of a PV system.

Library dependencies

The FlashLoader component has no dependencies on external libraries.

5.4.41 IntelStrataFlashJ3 component

The IntelStrataFlashJ3 component provides an efficient implementation of a NOR Flash memory type device. In normal usage, the device acts as *Read Only Memory* (ROM) whose contents can be determined either by programming using the flashloader port or by using standard flash programming software run on the model, such as the ARM Firmware Suite.

The size of the flash is determined by a startup parameter, size.

The IntelStrataFlashJ3 component implementation is approximately that of the Intel part used in the VE development board. The component is effectively organized as a bank of two 16bit Intel Flash components forming a 32bit component that can be read or programmed in parallel. The component supports all hardware behavior except for the following:

- protection register
- enhanced configuration register
- unique device identifier
- one time programmable cells
- block locking, which is silently ignored
- suspend/resume, which is silently ignored
- status interrupt line.

All block operations are atomic. This means that the status register state machine status bit always reads 1, ready.

For further information on the behavior of the hardware, see the Intel datasheet for the Intel StrataFlash Memory (J3). See Download Center, `http://downloadcenter.intel.com/`.

Figure 5-63 shows a view of the component in System Canvas.

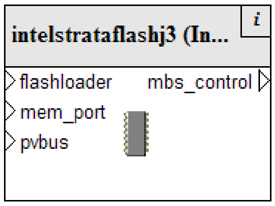


Figure 5-63 IntelStrataFlashJ3 in System Canvas

This component is written in LISA+.

Ports

Table 5-117 provides a brief description of the IntelStrataFlashJ3 component ports.

Table 5-117 IntelStrataFlashJ3 ports

Name	Port protocol	Type	Description
flashloader	FlashLoaderPort	Slave	Permits a FlashLoader component to initialize the flash contents from a binary file
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
mbs_control	PVBusSlaveControl	Master	Internal control
mem_port	PVDevice	Slave	

Additional protocols

The IntelStrataFlashJ3 component has one additional protocol, FlashLoaderPort. See *FlashLoaderPort* on page 5-144.

Parameters

Table 5-118 lists the parameters for the IntelStrataFlashJ3 component.

Table 5-118 IntelStrataFlashJ3 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
size	Set the size of the component in bytes	Integer	must be a power of 2	64MB

Registers

In normal operation the IntelStrataFlashJ3 component has no user visible registers. The device responds to the Common Flash Interface protocol which permits the device to be programmed at runtime. See general flash programming documentation on how to use the interface.

Debug features

The IntelStrataFlashJ3 component can be read/written using normal debugWrites.

Verification and testing

The IntelStrataFlashJ3 component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The IntelStrataFlashJ3 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The IntelStrataFlashJ3 component has no dependencies on external libraries.

5.4.42 VFS2 component

The *Virtual File System* (VFS), implemented in the VFS2 component, is a virtual device that provides access to parts of the underlying host filesystem through a target OS-specific driver. The component is virtual, so you might have to write your own driver. Example drivers are provided in Fast Models, or are available from ARM. These drivers are for bare metal, written in C++, and for Linux, written in C. Much of the driver code can be reused for porting to other operating systems.

There are two use cases for the VFS component, which are more fully described in the text files that accompany the examples:

- Using a Linux kernel patch to add the supplied VFS driver to Linux. This example can be requested from ARM.
- Writing a driver for a different OS. See the `WritingADriver.txt` file in the `VFS2/docs` directory.

The VFS device implementation consists of the following parts:

VFS LISA component

Coordinates device activity.

MessageBox component

Handles bus activity and interrupts. See *MessageBox component* on page 5-149.

`MBoxTypes.h`

Defines types shared between target and OS.

`VFS.cpp/h` Implements the VFS class/function interface.

`VFSFileSystem.cpp/h`

Provides a host filesystem abstraction layer.

`MessageCodec.h`

Provides utility classes for packing and unpacking messages.

`VFSOps.h` Defines the VFS operations.

`VFSTypes.h`

Defines types shared between target and OS.

To use the VFS component in a platform, add the `vfs.sgrepo` repository file to your project. You must also add the path to the C++ header files in the VFS implementation parts list to your project.

Figure 5-64 shows a view of the component in System Canvas.

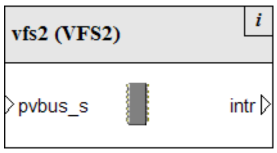


Figure 5-64 VFS2 in System Canvas

This component is written in LISA+.

Ports

Table 5-119 provides a brief description of the VFS2 component ports.

Table 5-119 VFS2 ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Provides memory-mapped access to the VFS device.
intr	Signal	Master	Optional interrupt line used to indicate availability of incoming VFS data. If the intr port is not used, MessageBox registers can be used to poll for incoming VFS data. See Table 5-123 on page 5-150.

Additional protocols

The VFS2 component has no additional protocols.

Parameters

Table 5-120 provides a description of the configuration parameters for the VFS2 component.

Table 5-120 VFS2 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
mount	Specifies the path to the host folder that is accessible within the target OS when mounted.	String	valid path	[empty string]

Registers

The VFS2 component has no registers.

Debug features

The VFS2 component has no debug features.

Verification and testing

The VFS2 component has been tested through use with Linux operating systems.

Performance

The VFS2 component is not expected to significantly affect the performance of a PV system, but is itself dependent on the performance of the host filesystem.

Library dependencies

The VFS2 component has no dependencies on external libraries.

5.4.43 MessageBox component

The MessageBox component is designed to operate as a subcomponent to the VFS2 component. See *VFS2 component* on page 5-147. When used with a suitable driver, MessageBox permits passing of blocks of data, or messages, between the driver and the parent VFS2 device. The MessageBox component is not a hardware model, and is designed to operate efficiently within a Fast Models platform model.

The current MessageBox implementation is generic but is primarily designed to provide a transport layer for the VFS2 component.

A C header file, `MBoxTypes.h`, is supplied in the `VFS2/C` directory of Fast Models. This header file contains definitions of register offsets, control and status bits, and buffer sizes. The example MessageBox driver implementation in the `VFS2/cpptest` directory is a simple, polling implementation written in C++.

Figure 5-65 shows a view of the component in System Canvas.

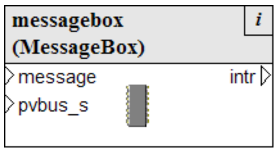


Figure 5-65 MessageBox component in System Canvas

This component is written in LISA+.

Ports

Table 5-121 provides a brief description of the MessageBox component ports.

Table 5-121 MessageBox ports

Name	Port protocol	Type	Description
pvbus_s	PVBus	Slave	Provides memory-mapped access to the MessageBox registers and shared buffer.
message	MessageBox	Slave	Delivers messages to the parent component, and receives messages from the parent for delivery to the target driver.
intr	Signal	Master	Optional interrupt line used to indicate availability of incoming message data. Alternatively the status register can be polled.

MessageBox

The MessageBox component has one additional protocol.

The message port implements half of a protocol for communication with a parent device. The other half is implemented by the parent device. When the CONTROL register is written to with the MBOX_CONTROL_END value, the data between the START and END offsets in the buffer are sent by reference to the parent device. Either during this call, or afterwards, the parent device is free to invoke the other message behaviors to set up an incoming packet and set the RXREADY signal.

The protocol behaviors are:

`begin_msg(void** buffer, uint32_t len)`

Returns a pointer to the buffer for sending a message. `len` is ignored.

`end_msg(uint32_t len)`

Sets the START/END registers to indicate the start and end of the message in the buffer and sets the RXREADY signal.

`message(const void* buffer, uint32_t len)`

Indicates that a message has been received, and the contents are in the buffer with a length of `len` bytes. This behavior must be implemented by the parent device.

`cancel_msg()`

Cancel a message.

Parameters

Table 5-122 provides a description of the configuration parameters for the MessageBox component.

Table 5-122 MessageBox configuration parameters

Parameter name	Description	Type	Allowed value	Default value
id	MessageBox ID	Integer	-	`0x01400400`

Registers

Table 5-123 provides a description of the configuration registers for the MessageBox component.

Table 5-123 MessageBox registers

Register name	Offset	Access	Description
MESSAGEBOX_ID	`0x00`	read only	Returns a user-configurable ID value.
MESSAGEBOX_DATA	`0x04`	read/write	See *DATA register* on page 5-151.
MESSAGEBOX_CONTROL	`0x08`	read/write	See *CONTROL register* on page 5-151.
MESSAGEBOX_STATUS	`0x0C`	read only	See *STATUS signal bits* on page 5-151.

Table 5-123 MessageBox registers (continued)

Register name	Offset	Access	Description
MESSAGEBOX_START	`0x10`	read/write	See *START register*.
MESSAGEBOX_END	`0x14`	read/write	See *END register* on page 5-152.
MESSAGEBOX_IRQMASK	`0x18`	read/write	Controls which of the status bits can cause the interrupt signal to be asserted.

DATA register

Writing to the DATA register writes to the buffer at the offset specified in the END register, and increments the register by 4.

Reading from the register reads from the buffer at the offset specified in the START register, and increments the register by 4.

CONTROL register

The CONTROL register issues commands to control message passing. The register can be programmed with one of two values:

1	Causes the START/END registers to be reset to 0 and clears the RXREADY bit in the STATUS register. This can also be done directly by programming the START/END registers to 0.
2	Causes the buffer memory between the START/END offsets to be sent to the parent component for processing. Typically the parent component performs some processing and at some point causes an incoming packet to be constructed in the buffer and the RXREADY signal to be set.

All other values for the CONTROL register are reserved.

STATUS signal bits

The STATUS signal returns status bits indicating whether more data can be transferred to or from the buffer and if there is a new incoming message available. The STATUS signal has three defined bits:

`0 RXEMPTY`	Set to 1 when START=END so no more receive data is available.
`1 TXFULL`	Set to 1 if END is incremented to the end of the buffer.
`2 RXREADY`	Set to 1 when an incoming packet is available for reading. This is reset to 0 when the DATA, START or END registers are accessed.

START register

When using the DATA register, the START register gives the offset of the next word to read from the buffer.

When accessing the buffer directly, for outgoing messages the START register is programmed to the start of the message. For incoming messages it indicates the start of the message. The offset is relative to the buffer start.

END register

When using the DATA register, the END register gives the offset to the first unused word after the end of the message.

When accessing the buffer directly, for outgoing messages the END register is programmed to 1 past the end of the message. For incoming messages it indicates 1 past the end of the message. The offset is relative to the buffer start.

Buffer

The data buffer is mapped starting at the device offset of `0x1000` and is currently 60KB in size, occupying the rest of the 64KB of device space. Do not use the first 4KB.

Interrupts

The interrupt line is asserted through `sg::Signal::Set` whenever `STATUS & IRQMASK` is non zero.

Debug features

The MessageBox component has no debug features.

Verification and testing

The MessageBox component has been tested through use with Linux operating systems.

Performance

The MessageBox component is not expected to significantly affect the performance of a PV system, but is itself dependent on the performance of the host filesystem.

Library dependencies

The MessageBox component has no dependencies on external libraries.

5.4.44 RAMDevice component

The RAMDevice component provides an efficient implementation of a generic memory device.

Figure 5-66 shows a view of the component in System Canvas.

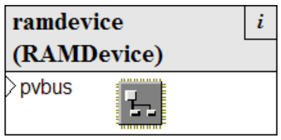


Figure 5-66 RAMDevice in System Canvas

This component is written in LISA+.

Ports

Table 5-124 provides a brief description of the ports of the RAMDevice component. For more information, see the technical reference manual for your hardware baseboard.

Table 5-124 RAMDevice ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Bus slave interface

Additional protocols

The RAMDevice component has no additional protocols.

Parameters

Table 5-125 lists the configuration parameters of the RAMDevice component.

Table 5-125 RAMDevice configuration parameters

Parameter name	Description	Type	Allowed value	Default value
size	Size of the memory in bytes	uint64_t	4KB to 4GB	`0x100000000` (4GB)

Registers

The RAMDevice component has no registers.

Debug features

The RAMDevice implements a CADI MEMORY view. You can connect a CADI client to the target and view the physical memory contents.

Verification and testing

The RAMDevice component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The RAMDevice component is not expected to significantly impact the performance of a PV system.

Library dependencies

The RAMDevice component has no dependencies on external libraries.

5.4.45 SMSC_91C111 component

The SMSC_91C111 component implements a model of the SMSC 91C111 Ethernet controller.

The model provides the register interface of the SMSC part and can be configured to act as an unconnected Ethernet port, or an Ethernet port connected to the host by an Ethernet bridge.

Information on how to install and configure the networking environment is described separately. See *Setting-up a TAP network connection and configuring the networking environment for Microsoft Windows* on page 5-180.

Figure 5-67 shows a view of the component in System Canvas.

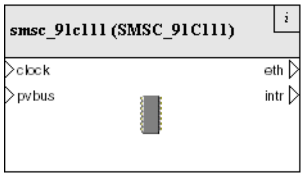


Figure 5-67 SMSC_91C111 in System Canvas

This component is written in C++.

Ports

Table 5-126 provides a brief description of the ports in the SMSC_91C111 component.

Table 5-126 SMSC_91C111 ports

Name	Port protocol	Type	Description
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder.
intr	Signal	Master	Interrupt signaling.
clock	ClockSignal	Slave	Clock input, typically 25MHz, which sets the master transmit/receive rate.
eth	VirtualEthernet	Master	Ethernet port.

Additional protocols

The SMSC_91C111 component has one additional protocol.

The VirtualEthernet protocol has the following behaviors:

`sendToSlave(EthernetFrame* frame)`

send an Ethernet frame to the slave port

`sendToMaster(EthernetFrame* frame)`

send an Ethernet frame to the master port.

The Ethernet frame class encapsulates an Ethernet frame in a broken-up format that is more accessible by components. For information on the class definition, see the `EthernetFrame.h` header file located in
`...\ARM\FastModelPortfolio_X.Y\include\components\VirtualEthernet\Protocol\*`

Parameters

Table 5-127 provides a description of the configuration parameters for the SMSC_91C111 component.

Table 5-127 SMSC_91C111 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
enabled	Used for sending Ethernet frames between components.	Boolean	true/false	false
mac_address	MAC address to use on host	String	see *mac_address parameter*	`00:02:f7:ef:00:02`
promiscuous	Puts host Ethernet controller into promiscuous mode, for instance when sharing the Ethernet controller with the host OS.	Boolean	true/false	true

mac_address parameter

There are two options for the mac_address parameter.

If a MAC address is not specified, when the simulator is run it takes the default MAC address, which is randomly-generated. This provides some degree of MAC address uniqueness when running models on multiple hosts on a local network.

Note

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

Registers

Table 5-128 on page 5-156, Table 5-129 on page 5-156, Table 5-130 on page 5-156 and Table 5-131 on page 5-157 provide descriptions of the configuration registers for the SMSC_91C111 component.

The SMSC_91C111 91C111 uses a banked register model of primarily 16 bit registers. There are also indirectly accessible registers for the PHY unit.

Bank 0 register

Table 5-128 lists the SMSC_91C111 bank 0 registers.

Table 5-128 SMSC_91C111 bank 0 registers

Register name	Offset	Access	Description
TCR	`0x0`	read/write	Transmit control
EPH	`0x2`	read only	Status of last transmitted frame
RCR	`0x4`	read/write	Receive control
COUNTER	`0x6`	read/write	MAC statistics
MIR	`0x8`	read/write	Memory information
RPCR	`0xA`	read/write	Receive/PHY control
BANK	`0xE`	read/write	Bank select

Bank 1 register

Table 5-129 lists the SMSC_91C111 bank 1 registers.

Table 5-129 SMSC_91C111 bank 1 registers

Register name	Offset	Access	Description
CONFIG	`0x0`	read/write	Configuration
BASE	`0x2`	read/write	Base address
IA0_1	`0x4`	read/write	MAC address 0, 1
IA2_3	`0x6`	read/write	MAC address 2, 3
IA4_5	`0x8`	read/write	MAC address 4, 5
GP	`0xA`	read/write	General purpose
CONTROL	`0xC`	read/write	Control
BANK	`0xE`	read/write	Bank select

Bank 2 register

Table 5-130 lists the SMSC_91C111 bank 2 registers.

Table 5-130 SMSC_91C111 bank 2 registers

Register name	Offset	Access	Description
MMU_COMMAND	`0x0`	read/write	MMU commands
PNR	`0x2`	read/write	Packet number
ALLOCATED	`0x3`	read/write	Allocated packet number

Table 5-130 SMSC_91C111 bank 2 registers (continued)

Register name	Offset	Access	Description
FIFO_PORTS	`0x4`	read/write	Tx/Rx FIFO packet number
POINTER	`0x6`	read/write	Address to access in Tx/Rx packet
DATA	`0x8`	read/write	Data register[a]
INTERRUPT	`0xC`	read/write	Interrupt status
INTERRUPT_MASK	`0xD`	read/write	Interrupt mask
BANK	`0xE`	read/write	Bank select

a. The data register can be accessed as 8, 16 or 32 bits and adjusts the pointer accordingly.

Bank 3 register

Table 5-131 lists the SMSC_91C111 bank 3 registers.

Table 5-131 SMSC_91C111 bank 3 registers

Register name	Offset	Access	Description
MT0_1	`0x0`	read/write	Multicast table 0, 1
MT2_3	`0x2`	read/write	Multicast table 2, 3
MT4_5	`0x4`	read/write	Multicast table 4, 5
MT6_7	`0x6`	read/write	Multicast table 6, 7
MGMT	`0x8`	read/write	Management interface
REVISION	`0xA`	read only	Chip revision ID
ERCV	`0xC`	read/write	Early receive
BANK	`0xE`	read/write	Bank select

Debug features

The SMSC_91C111 component has no debug features.

Verification and testing

The SMSC_91C111 component has been tested as part of the VE example system using VE test suites and by booting operating systems.

Performance

The SMSC_91C111 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The SMSC_91C111 component has no dependencies on external libraries.

5.4.46 HostBridge component

The HostBridge component is a virtual programmer's view model. It acts as a networking gateway to exchange Ethernet packets with a TAP device on the host, and to forward packets to NIC models. An alternative to this TAP/TUN method is user mode networking, which emulates a built-in IP router and DHCP server to route traffic by means of the host user mode socket layer. For more information, see *User mode networking* on page 5-180.

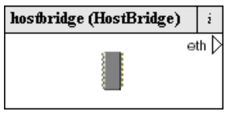


Figure 5-68 HostBridge in System Canvas

This component is written in LISA+.

Ports

Table 5-132 provides a brief description of the HostBridge component ports. For more information, see the hardware documentation.

Table 5-132 HostBridge ports

Name	Port Protocol	Type	Description
eth	VirtualEthernet	master	Send or receive Ethernet frame.

Additional protocols

The HostBridge component has one additional protocol, VirtualEthernet. See *VirtualEthernetCrossover component* on page 5-159.

Parameters

Table 5-133 provides a description of the configuration parameters for the HostBridge component.

Table 5-133 HostBridge configuration parameters

Parameter name	Description	Type	Allowed value	Default value
interfaceName	Host Interface	String	Valid string characters	ARM0
userNetPorts	Listening ports to expose in user-mode networking	String	Formatted string	[Empty string]
userNetSubnet	Virtual subnet for user-mode networking	String	Formatted string	172.20.51.0/24
userNetworking	Enable user-mode networking[a]	Boolean	`true` or `false`	`false`

a. TAP/TUN mode is enabled by default, and is disabled when user mode is in use.

Registers

The HostBridge component has no registers.

Debug Features

The HostBridge component has no debug features.

Verification and testing

The HostBridge component has been tested as part of a system with network functionalities.

Performance

It is not expected that normal networking usage of the HostBridge component significantly affects the performance of a PV system. However, heavy usage on the networking of the model, for example to run networking performance tests, might potentially slow down the simulation.

Library dependencies

The HostBridge component has no dependencies on external libraries.

See also

Reference

- *Fast Models User Guide*, *User mode networking* on page 2-31, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html`.

5.4.47 VirtualEthernetCrossover component

The VirtualEthernetCrossover component provides the functionality of an Ethernet crossover cable. It implements two VirtualEthernet slave ports and enables two VirtualEthernet master ports to be connected.

Data received on one port is forwarded to the other port without delay.

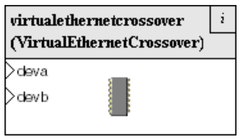


Figure 5-69 VirtualEthernetCrossover in System Canvas

This component is written in LISA+.

Ports

Table 5-134 provides a brief description of the VirtualEthernetCrossover component ports.

Table 5-134 VirtualEthernetCrossover ports

Name	Port Protocol	Type	Description
deva	VirtualEthernet	slave	Slave port for connecting to a VirtualEthernet master
devb	VirtualEthernet	slave	Slave port for connecting to a VirtualEthernet master

Additional protocols

The VirtualEthernetCrossover component has no additional protocols.

Parameters

The VirtualEthernetCrossover component has no parameters.

Registers

The VirtualEthernetCrossover component has no registers.

Debug Features

The VirtualEthernetCrossover component has no debug features.

Verification and testing

The VirtualEthernetCrossover component has been tested as part of the VE example system using the networking test suites and by booting operating systems.

Performance

The VirtualEthernetCrossover component is not expected to significantly affect the performance of a PV system.

Library dependencies

The VirtualEthernetCrossover component has no dependencies on external libraries.

5.4.48 CCI400 component

The CCI400 component is the *Cache Coherent Interconnect* for AXI 4. *AXI Coherency Extensions* (ACE) are extensions to AXI4 that provide support for system-level cache-coherency between multiple clusters. See Figure 5-70 on page 5-161:

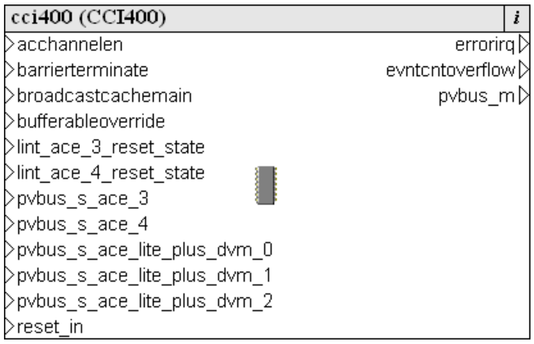


Figure 5-70 CCI400 in System Canvas

This component is written in LISA+.

Ports

Table 5-135 provides a brief description of the CCI400 component ports. For more information, see the hardware documentation.

Table 5-135 CCI400 ports

Name	Port Protocol	Type	Description
`acchannelen`	Value	slave	For each upstream port, determine if it is enabled or not with.respect to snoop requests.
`barrierterminate`	Value	slave	For each downstream port, determine if barriers are terminated at that port.
`broadcastcachemain`	Value	slave	For each downstream port, determine if broadcast cache maintenance operations are forwarded down that port.
`bufferableoverride`	Value	slave	For each downstream port, determine if all transactions are forced to non-bufferable.
`errorirq`	Signal	master	A signal stating that the imprecise error register is nonzero.
`evntcntoverflow[5]`	Signal	master	When an event counter overflows, it sets the corresponding signal.
`lint_ace_3_reset_state`, `lint_ace_4_reset_state`	Signal	slave	These ports can be connected to the reset signals of the system attached to the `pbvus_s_ace_3` and `pvbus_s_ace_4` ports.
`pvbus_m`	PVBus	master	Master port for all downstream memory accesses.
`pbvus_s_ace_3`, `pvbus_s_ace_4`	PVBus	slave	ACE-capable slave ports.
`pvbus_s_ace_lite_plus_dvm_0`, `pvbus_s_ace_lite_plus_dvm_1`, `pvbus_s_ace_lite_plus_dvm_2`	PVBus	slave	Memory bus interface that implements ACE lite and DVM protocol.
`reset_in`	Signal	slave	Signal to reset the CCI.

Additional protocols

The CCI400 component has no additional protocols.

Parameters

Table 5-136 provides a description of the configuration parameters for the CCI400 component.

Table 5-136 CCI400 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`acchannelen`	For each upstream port, determine if it is enabled or not with respect to snoop requests.	Integer	0-31	31
`barrierterminate`	For each downstream port, determine if barriers are terminated at that port.	Integer	0-7	7
`broadcastcachemain`	For each downstream port a bit determines if broadcast cache maintenance operations are forwarded down that port.	Integer	0-7	0
`bufferableoverride`	For each downstream port, determine if all transactions are forced to non-bufferable.	Integer	0-7	0
`cache_state_modelled`	Model the cache coherency operations. This must be enabled to correctly maintain coherency between ACE masters that model cache state.	Boolean	`true` or `false`	`true`
`force_on_from_start`	The CCI normally starts up with snooping disabled. However, using this permits the model to start up as enabled without having to program it. This is only set up at simulation reset and not at signal reset.	Boolean	`true` or `false`	`false`
`log_enabled`	Enable log messages from the CCI register file: • 0 means do not print anything • 1 means print only access violations • 2 means also print writes • 3 means also print reads.	Integer	0, 1, 2, 3	1

Registers

The CCI400 component provides the registers specified by the technical reference manual.

Debug Features

The CCI400 component exports a CADI debug interface.

Verification and testing

The CCI400 component has been tested:

- by running a switching hypervisor on an example system containing an ARMCortexA7x*n*CT component and an ARMCortexA15x*n*CT CCI400 model.

Performance

If `cache_state_modelled` is disabled, the CCI400 component has negligible performance impact. If `cache_state_modelled` is enabled, it adds significant cost to throughput for coherent transactions.

Library dependencies

The CCI400 component has no dependencies on external libraries.

5.4.49 GIC400 component

The GIC400 component represents the GIC-400 *Generic Interrupt Controller* (GIC), and includes a *Virtualized Generic Interrupt Controller* (VGIC). It is a wrapper that permits easier configuration of the v7_VGIC component that supports parameterized configuration. See Figure 5-71:

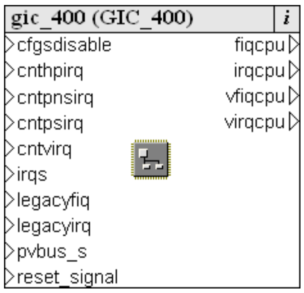


Figure 5-71 GIC400 in System Canvas

This component is written in LISA+.

The GIC-400 has several memory-mapped interfaces at the same address that are banked by the processor that is communicating with it. Consequently, the GIC-400 must be able to distinguish from which processor a transaction has originated. In the hardware, the GIC-400 is supplied this information in the AUSER fields on AXI. In Fast Models, there is no exact equivalent to this field. However, each transaction has a `master_id` that the model can choose to use to identify the originating processor.

ARM clusters assign the `master_id` as follows:

- bits[31:16] SBZ (ignored)
- bits[5:2] CLUSTERID
- bits[1:0] cpu_id within cluster

where CLUSTERID is the 4 bit field that is set either by a parameter on the processor or by driving a value on the clusterid port. CPUID is the processor number within the cluster. CLUSTERID appears in the CP15 register space as part of the MPIDR register.

The ARM architecture suggests that each cluster in the system is given a different CLUSTERID, and this is essential for the VGIC to identify the processor. The parameters in the GIC-400 component permit it to construct the map of `master_id` to interface number.

For each processor interface that the GIC-400 supports, there are the following parameters:

- `interfaceN.cluster_id`
- `interfaceN.core_id`
- `interfaceN.inout_port_number_to_use`

where N is the interface number (0-7). The `cluster_id` and `core_id` tell the GIC-400 to map that cluster or processor combination to interface N.

In using `inout_port_number_to_use`, the GIC-400 has a number of input and output ports that are intended to be paired with a particular processor interface. For example:

- the `irqcpu[]` pin is intended to be wired to the irq port of the corresponding processor
- the `cntpnsirq` pin from the processor is intended to be wired to a `cntpnsirq[]` pin on GIC-400 to transport a *Private Peripheral Interrupt* (PPI) from the processor to the GIC-400.

To easily support clusters that can have variable numbers of processors, the `interfaceN.inout_port_number_to_use` parameter tells the GIC-400 that if it wants to send or receive a signal to the processor attached to interface N, then it should use the following pins:

- `irqout[interfaceN.inout_port_number_to_use]`
- `fiqout[interfaceN.inout_port_number_to_use]`
- `virqout[interfaceN.inout_port_number_to_use]`
- `vfiqout[interfaceN.inout_port_number_to_use]`
- `legacyirq[interfaceN.inout_port_number_to_use]`
- `cntpnsirq[interfaceN.inout_port_number_to_use]`
- `cntpsirq[interfaceN.inout_port_number_to_use]`
- `legacyfiq[interfaceN.inout_port_number_to_use]`
- `cntvirq[interfaceN.inout_port_number_to_use]`
- `cnthpirq[interfaceN.inout_port_number_to_use]`
- ...

——— **Note** ———

`legacyirq` and `legacyfiq` are not signals from the processor but are signals into the GIC-400 from the legacy interrupt system. They are wired to PPIs and can also bypass the GIC-400 if control registers of the GIC-400 are set up in particular ways. See the *ARM Generic Interrupt Controller Architecture version 2.0 Architecture Specification* for more information.

———

——— **Note** ———

The fabric between the ARM clusters and the GIC might remap the `master_id` of a transaction, and if so then the GIC might lose the ability to identify the originating processor. The fabrics that ARM ships in Fast Models perform no such transformation.

The comparison that the GIC-400 performs on the `master_id` is only on the bottom 6 bits of the `master_id`, and the rest are ignored. If you are writing your own fabric and do not properly propagate the `master_id` or transform it, the GIC-400 might not be able to identify the processor correctly. The source code for the GIC_400 component can be examined to see how it might be adapted for it to understand different `master_id` schemes.

———

Ports

Table 5-137 provides a brief description of the GIC400 component ports. For more information, see the hardware documentation.

Table 5-137 GIC400 ports

Name	Port Protocol	Type	Description
`cfgsdisable`	Signal	slave	Disable write access to some GIC registers.
`cnthpirq`	Signal	slave	Hypervisor physical timer event.
`cntpnsirq`	Signal	slave	Non-secure physical timer event.
`cntpsirq`	Signal	slave	Secure physical timer event.
`cntvirq`	Signal	slave	Virtual timer event.
`fiqcpu`	Signal	master	FIQ signal to the corresponding processor.
`irqcpu`	Signal	master	IRQ signal to the corresponding processor.
`irqs`	Signal	slave	Interrupt request input lines for the GIC.
`legacyfiq`	Signal	slave	Signal into the GIC-400 from the legacy interrupt system.
`legacyirq`	Signal	slave	Signal into the GIC-400 from the legacy interrupt system.
`pvbus_s`	PVBus	slave	Handles incoming transactions from PVBus masters.
`reset_signal`	Signal	slave	Reset signal input.
`vfiqcpu`	Signal	master	Virtual FIQ signal to the processor.
`virqcpu`	Signal	master	Virtual IRQ signal to the processor.

Additional protocols

The GIC400 component has no additional protocols.

Parameters

Table 5-138 provides a description of the configuration parameters for the GIC400 component.

Table 5-138 GIC400 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
NUM_CPUS	Number of interfaces to support.	Integer	1-8	1
NUM_SPIS	Number of shared peripheral interrupt pins.	Integer	0-480	224
`interface0.cluster_id`	The CLUSTERID of the interface you want to appear as `interface0` in the VGIC.	Integer	0-15	0
`interface0.core_id`	The processor id of `interface0` in the cluster.	Integer	0-15	0
`interface0.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	0
`interface1.cluster_id`	The CLUSTERID of the processor you want to appear as `interface1` in the VGIC.	Integer	0-15	0
`interface1.core_id`	The processor id of `interface1` in the cluster.	Integer	0-15	0
`interface1.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	1
`interface2.cluster_id`	The CLUSTERID of the interface you want to appear as `core0` in the VGIC.	Integer	0-15	0
`interface2.core_id`	The processor id of `core0` in the cluster.	Integer	0-15	0
`interface2.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	2
`interface3.cluster_id`	The CLUSTERID of the interface you want to appear as `interface3` in the VGIC.	Integer	0-15	0
`interface3.core_id`	The processor id of `interface3` in the cluster.	Integer	0-15	0
`interface3.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	3
`interface4.cluster_id`	The CLUSTERID of the interface you want to appear as `interface4` in the VGIC.	Integer	0-15	0
`interface4.core_id`	The processor id of interface4 in the cluster.	Integer	0-15	0
`interface4.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	4

Table 5-138 GIC400 configuration parameters (continued)

Parameter name	Description	Type	Allowed value	Default value
`interface5.cluster_id`	The CLUSTERID of the interface you want to appear as `interface5` in the VGIC.	Integer	0-15	0
`interface5.core_id`	The processor id of `interface5` in the cluster.	Integer	0-15	0
`interface5.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	5
`interface6.cluster_id`	The CLUSTERID of the interface you want to appear as `interface6` in the VGIC.	Integer	0-15	0
`interface6.core_id`	The processor id of `interface6` in the cluster.	Integer	0-15	0
`interface6.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	6
`interface7.cluster_id`	The CLUSTERID of the interface you want to appear as `interface7` in the VGIC.	Integer	0-15	0
`interface7.core_id`	The processor id of `interface7` in the cluster.	Integer	0-15	0
`interface7.inout_port_number_to_use`	Which ppiN port is used for this interface.	Integer	0-7	7

Registers

The GIC400 component has no registers.

Debug Features

The GIC400 component has no debug features.

Verification and testing

The GIC400 component has been tested as part of a system with network functionalities.

Performance

The GIC400 component is not expected to significantly affect the performance of a PV system.

Library dependencies

The GIC400 component has no dependencies on external libraries.

5.4.50 MemoryMappedCounterModule component

The MemoryMappedCounterModule component implements a memory-mapped counter module, and is required for models containing multiple clusters of processors with Generic Timers. It must also be used to run a single processor system where the Generic Timer runs at a different rate to the input clock to the processor. See Figure 5-72:

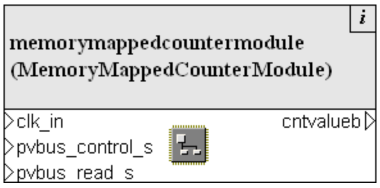


Figure 5-72 MemoryMappedCounterModule in System Canvas

This component is written in LISA+.

Ports

Table 5-139 provides a brief description of the MemoryMappedCounterModule component ports. For more information, see the hardware documentation.

Table 5-139 MemoryMappedCounterModule ports

Name	Port Protocol	Type	Description
`clk_in`	Clock signal	slave	This clock input is used to determine the frequency of the Physical Count provided to the clusters connected to the `cntvalueb` port.
`cntvalueb`	CompoundPortLISA	master	This implements a private protocol between the cluster and the MemoryMappedCounterModule. This must be connected to the `cntvalueb` port on each cluster in the system and to the MemoryMappedCounterModule component.
`pvbus_control_s`	PVBus[a]	slave	This slave port is used to provide memory-mapped read write access to the control registers of the module.
`pvbus_read_s`	PVBus[a]	slave	This slave port is used to provide memory-mapped read access to additional registers. It is currently not implemented.

a. Two bus slave ports are provided because the architecture specification permits each set of registers to be mapped at different base addresses that do not have to be contiguous.

Additional protocols

The MemoryMappedCounterModule component has no additional protocols.

Parameters

Table 5-140 provides a description of the configuration parameters for the MemoryMappedCounterModule component.

Table 5-140 MemoryMappedCounterModule configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`non_arch_start_at_default`	A model-specific way of enabling the counter module out of reset.	Boolean	`true` or `false`	`false`

Registers

The MemoryMappedCounterModule component has no registers.

Debug Features

The MemoryMappedCounterModule component has no debug features.

Verification and testing

The MemoryMappedCounterModule component has been tested as part of a system with network functionalities.

Performance

The MemoryMappedCounterModule component is not expected to significantly affect the performance of a PV system.

Library dependencies

The MemoryMappedCounterModule component has no dependencies on external libraries.

5.5 Visualisation Library

This section describes the components and protocols included in the Visualisation Library for Fast Models. It contains the following sections:

- *About the Visualisation Library*
- *LCD protocol* on page 5-171
- *Visualisation components* on page 5-171
- *GUIPoll component* on page 5-171
- *C++ classes* on page 5-173.

5.5.1 About the Visualisation Library

The Visualisation Library is a part of Fast Models.

The Visualisation Library does not model hardware directly but instead provides you with components, protocols, and a library. These permit your simulation to display a GUI that lets you interact with the external I/O from the platform being modeled.

The types of I/O handled include:

- LCD display, such as the output from the PL110_CLCD component display port
- LEDs representing values from a ValueState port as either single lights, or as segmented alphanumeric displays
- DIP switches, which can drive a ValueState port
- capture of keyboard and mouse input, using the KeyboardStatus and MouseStatus protocols to feed input to a PS2Keyboard or PS2Mouse component.
- background graphics, custom rendered graphics, and clickable push buttons, permitting the UI to provide display a skin representing the physical appearance of the device being modeled
- status information such as processor instruction counters, with values taken from the InstructionCount port of a processor.

The Visualisation Library provides a C++ API that enables you to write your own visualization components in LISA+. These custom components can display any combination of the supported I/O types. See *C++ classes* on page 5-173.

You can add the prebuilt GUITick component to your custom component. The GUITick component provides a LISA visualization component with a periodic signal which is used to keep the display updated, even when the simulation is stopped.

An example of how to use the Visualisation API is in `%PVLIB_HOME%\examples\VP_PhoneSkin`. On Linux, the equivalent directory is `$PVLIB_HOME/examples/VP_PhoneSkin`. The PhoneVisualisation subcomponent uses the Visualisation API to create a GUI consisting of a status panel, two LCD panels, and some push buttons.

A prebuilt component called Visualisation is included within the platform models. This component provides a window containing a representation of a single LCD display, DIP switches, LEDs, and one or two instruction counters. The window resizes to match the timing parameters configured in the LCD controller. See *Visualisation components* on page 5-171.

5.5.2 LCD protocol

The Visualisation Library supports one signaling protocol, the LCD protocol. The LCD protocol provides the interface between an LCD controller peripheral (for example the PL110) and a visualization component. This permits the LCD controller to render the framebuffer contents into a region of the visualization GUI.

LISA visualization components can provide any number of LCD ports. The implementations of these behaviors can delegate the calls to appropriate methods on the `VisRenderRegion` class.

See the source code of the `PhoneVisualisation.lisa` component for an example of implementing the LCD protocol.

The LCD protocol behaviors are:

`lock() : VisRasterLayout *`
: This locks the raster region of the LCD in preparation for rendering.

`unlock()`
: This unlocks the raster region, ready to be updated on the screen.

`update(int x, int y, unsigned int w, unsigned int h)`
: This updates the selected rectangular area on screen from the raster buffer.

`setPreferredLayout(unsigned int width, unsigned int height, unsigned int depth)`
: This is a request from the LCD controller to set the preferred size for the raster region, to match the timing parameters used by the LCD controller.

5.5.3 Visualisation components

The Visualisation components provide a host window to display status information in addition to a frame buffer. Each example platform model, such as the VE, contains its own LISA Visualisation model. You can use the model as the basis for your own visualization containing components such as a PL110_CLCD component. To use the Visualisation components in your own system, you must copy the component LISA files from the relevant platform model directory, as the components are not included in the generic Model Library.

Detailed information on the platform model Visualisation components is provided elsewhere. For more information about the VE Fixed Virtual Platform, see Chapter 6 *Versatile Express Model: Platform and Components* and the *RealView Development Suite Real-Time System Models User Guide*.

5.5.4 GUIPoll component

The GUIPoll component is a precompiled component that provides a periodic signal that can be used to drive a GUI refresh in a visualization component. It is intended to be used as a subcomponent inside of a LISA-based visualization component.

You can configure the period at which the GUIPoll component runs in milliseconds of real time, not simulation time. The component produces a `gui_callback()` signal at approximately this period, even when the simulation is paused.

You must implement the slave side of the `gui_callback()` signal in a LISA-based visualization component. Use this event to invoke the `Visualisation::poll()` method to keep the GUI updated.

Figure 5-73 on page 5-172 shows a view of the component in System Canvas.

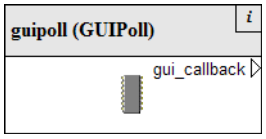


Figure 5-73 GUIPoll in System Canvas

This component is written in LISA+.

Ports

The GUIPoll component generates signals on the port in Table 5-141.

Table 5-141 GUIPoll component ports

Name	Port protocol	Type	Description
gui_callback	GUIPollCallback	Master	This port is used to send callback requests to the visualization component

Additional protocols

The GUIPoll component has one additional protocol.

GUIPollCallback

This protocol defines a single method that is used to signal to the visualization component that the requested update period has elapsed. You can invoke this callback even when the simulation is stopped. The behavior of the protocol is:

`gui_callback() : void`

This is sent by the GUIPoll component, at the configured period.

Parameters

Table 5-142 provides a description of the configuration parameters for the GUIPoll component.

Table 5-142 GUIPoll configuration parameters

Parameter name	Description	Type	Allowed value	Default value
delay_ms	Determines the period, in milliseconds of real-time, between `gui_callback()` calls	Integer	-	20

Registers

The GUIPoll component has no registers.

Debug features

The GUIPoll component has no debug features.

Verification and testing

The GUIPoll component has been tested as part of the PhoneSkin example platform, confirming that the GUI continues to be refreshed even when the simulation is stopped.

Performance

The GUIPoll component is not expected to significantly affect the performance of a PV system.

Library dependencies

The GUIPoll component has no dependencies on external libraries.

5.5.5 C++ classes

This section describes the C++ classes and structures that are made available to you through the Visualisation Library. To make use of these classes, you must ensure that your LISA component begins with:

```
includes
{
#include “components/Visualisation.h”
}
```

Classes and structures described in this section are:

- *Class Visualisation*
- *Class VisRegion* on page 5-175
- *Class VisPushButtonRegion : public VisRegion* on page 5-175
- *Class VisRenderRegion : public VisRegion* on page 5-175
- *Struct VisRasterLayout* on page 5-176
- *Struct VisPixelFormat* on page 5-177.

Class Visualisation

The Visualisation class provides access to the API for creating a custom LISA visualization component.

A component obtains an instance of this class by calling the global function `createVisualisation()`. The component can then use this instance to control the size and layout of the visualization window.

The Visualisation Library is currently implemented using the *Simple DirectMedia Layer* (SDL) cross-platform rendering library.

Global functions

`Visualisation *createVisualisation()`

This generates an instance of the Visualisation Library. You can only call this function once, as SDL only supports opening a single display window.

Class Visualisation

`~Visualisation()`

Destructor for the Visualisation Library. You must only call this method when your simulation is shutting down, after all allocated resources (VisRenderRegions, VisPushButtonRegions, VisBitmaps) have been deleted.

`void configureWindow(unsigned int width, unsigned int height, unsigned int bit_depth)`

This sets the visualization window to the requested size and bit depth. Depending on the display capabilities, the window might actually get a different bit depth from the size you requested.

`VisBitmap *loadImage(char const *filename)`

This allocates a new VisBitmap object, initialized by loading a Microsoft Windows Bitmap (`.BMP`) from the given file.

`VisBitmap *loadImageWithAlphaKey(char const *filename, unsigned int red, unsigned int green, unsigned int blue)`

This allocates a VisBitmap object, as with `loadImage()`. All pixels of the color specified by `red`, `green`, `blue` are converted into a transparent alpha channel.

`VisBitmap *cropImage(VisBitmap *source, int x, int y, unsigned int width, unsigned int height)`

This allocates a new VisBitmap object, by cropping a region from the source bitmap.

`void releaseImage(VisBitmap *)`

This releases the resources held by the given VisBitmap. The underlying bitmap is only be unloaded if it is not in use.

`void setBackground(VisBitmap *background, int x, int y)`

This sets the background image for the visualization window. This takes a copy of the data referenced by the VisBitmap, so it is safe for the client to call `releaseImage(background)` immediately after calling `setBackground()`. The background is not displayed until the first call to `poll()`.

`VisRenderRegion *createRenderRegion()`

This allocates a new VisRenderRegion object that can be used to display arbitrary graphics, including LCD contents, in a rectangular region.

`VisPushButtonRegion *createPushButtonRegion()`

This allocates a new VisPushButtonRegion, which can be placed at a location on the display to provide a clickable push button.

`bool poll(VisEvent *event)`

This call permits the Visualisation Library to poll for GUI events. The client passes a reference to a VisEvent structure, which receives details of a single mouse/keyboard event.

The method returns false if no events have occurred.

Your LISA visualization implementations should call this periodically by using a GUIPoll component. On each `gui_callback()` event, you must ensure that the visualization component repeatedly calls `poll()` until it returns false.

`void lockMouse(VisRegion *region)`

This method locks the mouse to the visualization window and hides the mouse pointer.

`void unlockMouse()`

This unlocks and redisplays the mouse pointer.

`bool hasQuit()`

This returns true if the user has clicked on the close icon of the visualization window.

Class VisRegion

The `VisRegion` class is the common base class for VisPushButtonRegion and VisRenderRegion, representing an arbitrary region of the visualization display.

`~VisRegion()` This permits clients to delete a VisPushButtonRegion when it is no longer required.

`void setId(void *id)`

This permits a client-defined identifier to be associated with the region.

`void *getId()`

This returns the client-defined identifier.

`void setVisible(bool vis)`

Specifies whether the region is to be displayed on the screen. This is currently ignored by the SDL implementation.

`void setLocation(int x, int y, unsigned int width, unsigned int height)`

This sets the location of this region relative to the visualization window.

Class VisPushButtonRegion : public VisRegion

The `VisPushButtonRegion : public VisRegion` class defines a region of the visualization window that represents a clickable button. Optionally, the button can provide different VisBitmap representations for a button-up and a button-down graphic, and a graphic to use when the mouse pointer rolls over the button.

In addition to the public method defined in VisRegion, this class defines the following:

`void setButtonUpImage(VisBitmap *bmpUp) : void`, `setButtonDownImage(VisBitmap *bmpDown) : void`, `setButtonRollOverImage(VisBitmap *bmpRollover)`

Sets the graphics to be used for each of the button states. If any image is not specified or is set to NULL, then the corresponding area of the visualization background image is used.

The VisPushButtonRegion takes a copy of the VisBitmap, so the client can safely call `Visualisation::releaseBitmap()` on its copy.

`void setKeyCode(int code)`

This sets the code for the keypress event that is generated when the button is pressed or released.

Class VisRenderRegion : public VisRegion

The `VisRenderRegion : public VisRegion` class defines a region of the visualization window that can render arbitrary client-drawn graphics, including a representation of the contents of an LCD.

In addition to the public method defined in VisRegion, this class defines the following:

`VisRasterLayout const *lock()`

Locks the region for client rendering. The VisRasterLayout structure is described elsewhere. See *Struct VisRasterLayout*. While the buffer is locked, the client can modify the pixel data for the buffer. You must not call the methods `writeText()` and `renderBitmap()` while the buffer is locked.

`void unlock()`

This releases the lock on the render buffer, permitting the buffer to be updated on screen.

`void update(int left, int top, unsigned int width, unsigned int height)`

This causes the specified rectangle to be drawn to the GUI.

`int writeText(const char *text, int x, int y)`

This renders the given ASCII text onto an unlocked VisRenderRegion. The return value is the x co-ordinate of the end of the string. The default font is 8 pixels high, and cannot be changed.

void `renderBitmap(VisBitmap *bitmap, int x, int y)`

Draws a bitmap onto an unlocked VisRenderRegion.

Struct VisRasterLayout

The `VisRasterLayout` struct defines the layout of the pixel data in a frame-buffer. The `lock()` method of the LCD protocol expects to be given a pointer to this structure. You can generate a suitable instance by calling `VisRasterRegion::lock()`.

The structure contains the following fields:

`uint8_t *`**buffer**

This points to the buffer for the rasterized pixel data. The controller can write pixels into this buffer, but must stick within the bounds specified by the width and height.

`uint32_t` **pitch**

The number of bytes between consecutive raster lines in the pixel data. This can be greater than the number of bytes per line.

`uint32_t` **width**

The width, in pixels, of the render area. This value can be less than the width requested by the LCD controller when it called `lock()`.

`uint32_t` **height**

The height, in pixels, of the render area. This value can be less than the height requested by the LCD controller when it called `lock()`.

`VisPixelFormat` **format**

This structure defines the format of the pixel data in the raster buffer.

`bool` **changed**

This is set to true if the pixel format or buffer size has changed since the previous call to `lock()`.

Pixel data is represented as a one-dimensional array of bytes. The top-left pixel is pointed to by the buffer member. Each pixel takes up a number of bytes, given by `format.pbytes`.

The pixel at location (x, y) is stored in the memory bytes starting at `buffer[y * pitch + x * format.pbytes]`.

Struct VisPixelFormat

The `VisPixelFormat` struct specifies the format of pixel data within the buffer.

The members are:

`uint32_t rbits, gbits, bbits`

The number of bits per color channel.

uint32_t `roff, goff, boff`

The offset within the pixel data value for the red/green/blue channels.

uint32_t `pbytes`

The size of a single pixel, in bytes.

`format.pbytes` specifies the number of bytes that make up the data for a single pixel. These bytes represent a single pixel value, stored in host-endian order. The pixel value contains a number of the form `(R<<format.roff) + (G<<format.goff) + (B<<format.boff)`, where `(R,G,B)` represents the values of the color channels for the pixel, containing values from 0 up to `(1<<format.rbits)`, `(1<<format.gbits)`, `(1<<format.bbits)`.

5.6 Using Component Features

This section describes how to use selected component features available in Fast Models. These components are:

- *Terminal*
- *User mode networking* on page 5-180
- *Setting-up a TAP network connection and configuring the networking environment for Microsoft Windows* on page 5-180
- *Setting-up a network connection and configuring the networking environment for Linux* on page 5-182.

5.6.1 Terminal

The Terminal component is a virtual component that permits UART data to be transferred between a TCP/IP socket on the host and a serial port on the target. The component itself is described in *TelnetTerminal component* on page 5-50.

——— **Note** ————

If you want to use the Terminal component with a Microsoft Windows 7 client you must first install Telnet. The Telnet application is not installed on Microsoft Windows 7 by default. You can download the application by following the instructions on the Microsoft web site. Run a search for “Windows 7 Telnet” to find the Telnet FAQ page, or see , `http://windows.microsoft.com/en-US/windows7/Telnet-frequently-asked-questions`. To install Telnet:

1. Select **Start** → **Control Panel** → **Programs and Features**. This opens a window that lets you uninstall or change programs.

2. Select **Turn Windows features on or off** in the left-hand side bar. This opens the Windows Features dialog. Select the **Telnet Client** check box.

3. Click **OK**. The installation of Telnet might take several minutes to complete.

————————

A block diagram of one possible relationship between the target and host through the Terminal component is shown in Figure 5-74 on page 5-179. The TelnetTerminal block is what you configure when you define Terminal component parameters. The Virtual Machine is your model system.

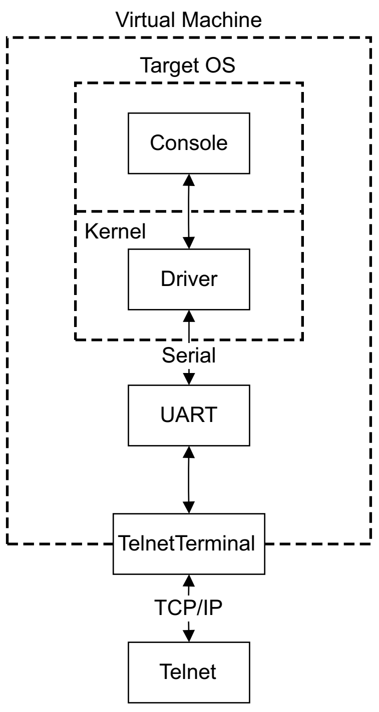


Figure 5-74 Terminal block diagram

On the target side, the console process invoked by your target OS relies on a suitable driver being present. Such drivers are normally part of the OS kernel. The driver passes serial data through a UART. The data is forwarded to the TelnetTerminal component, which exposes a TCP/IP port to the world outside of the model. This port can be connected to by, for example, a Telnet process on the host.

You can change the startup behavior for each of up to four Terminals by modifying the corresponding component parameters. See *Parameters* on page 5-51.

If the Terminal connection is broken, for example by closing a client telnet session, the port is re-opened on the host. This could have a different port number if the original one is no longer available. Before the first data access, you can connect a client of your choice to the network socket. If there is no existing connection when the first data access is made, and the start_telnet parameter is true, a host telnet session is started automatically.

The port number of a particular Terminal instance can be defined when your model system starts. The actual value of the port used by each Terminal is declared when it starts or restarts, and might not be the value you specified if the port is already in use. If you are using Model Shell, the port numbers are displayed in the host window in which you started the model.

You can start the Terminal component in one of two modes:

- *Telnet mode*
- *Raw mode* on page 5-180.

Telnet mode

In telnet mode, the Terminal component supports a subset of the RFC 854 protocol. This means that the Terminal participates in negotiations between the host and client concerning what is and is not supported, but flow control is not implemented.

Raw mode

Raw mode permits the byte stream to pass unmodified between the host and the target. This means that the Terminal component does not participate in initial capability negotiations between the host and client, and instead acts as a TCP/IP port. You can take advantage of this feature to directly connect to your target through the Terminal component.

5.6.2 User mode networking

User mode networking emulates a built-in IP router and DHCP server, and routes TCP and UDP traffic between the guest and host, and uses the user mode socket layer of the host to communicate with other hosts. This allows the use of a significant number of IP network services without requiring administrative privileges, or the installation of a separate driver on the host on which the model is running.

To set up and use user mode networking:

1. Install the Fast Models package to obtain the TPIP for the images.
2. Start a model with Linux from the TPIP package. For example, use:
   - Model: FVP_VE_Cortex-A15x1
   - Kernel: FVP_VE_CLI_VE_V7_LPAE/FVP_VE_CLI_VE_V7_LPAE.axf
   - Image: FVP_VE_CLI_VE_V7_LPAE/armv5t_min_VE_V7_LPAE.image.
3. To enable user mode networking, run the model with the following additional CADI parameters:
   ```
   -C motherboard.hostbridge.userNetworking=true
   -C motherboard.smsc_91c111.enabled=true
   ```
4. To map a host port to a model port, run the model with the following additional CADI parameters:
   ```
   -C motherboard.hostbridge.userNetPorts="8022=22"
   ```
   This example maps port 8022 on the host to port 22 on the model.

Note

You can use only TCP and UDP over IP. ICMP (ping) is not supported.

DHCP is only supported within the private network.

You can only make inward connections by mapping ports on the host to the model. This is common to all implementations that provide host connectivity using NAT.

Operations that require privileged source ports, for example NFS in its default configuration, do not work.

If setup fails, or the parameter syntax is incorrect, there is no error reporting.

5.6.3 Setting-up a TAP network connection and configuring the networking environment for Microsoft Windows

This section describes how to set up a network connection for your PC, and then to configure the networking environment for use by Fast Models on a Microsoft Windows platform.

——— **Note** ———

You must ensure that you first install the third-party IP package in the Fast Models networking environment.

Setting-up a network connection

To set up the network connection:

1. Close all non-essential applications.
2. Install the TAP driver and configure it:
   - Access the base location of Fast Model Portfolio, default `C:\Program Files\ARM\FastModelPortfolio_X.Y`
   - Run either:
     — `ModelNetworking\add_adapter_32.bat` for 32-bit Microsoft Windows
     — `ModelNetworking\add_adapter_64.bat` for 64-bit Microsoft Windows.
3. Select **Start** → **Control Panel** → **Network Connections** and locate the newly-installed TAP device.
4. Press the **Ctrl** key to multi-select at least one real Ethernet adapter connection.
5. Right-click and select **Bridge Connections**.

——— **Note** ———

You only have to perform this procedure once, at the beginning.

Configuring the networking environment

Use System generator to load the required project and build a model, or use Model Shell or Model Debugger to start a prebuilt model.

Set the following parameters on the **HostBridge** and **SMSC_91C111** components. Enter:

```
hostbridge.interfaceName="ARM<x>"
smsc_91c111.enabled=1
```

where **ARM<x>** is the adapter built into the network bridge, and is 0 by default. If ARM0 already exists, then use the next available integer, that is, 1, 2, 3...

——— **Note** ———

This procedure has not been tested with wireless network adapters.

——— **Note** ———

If you have to enable promiscuous mode for the TAP device, you must set an additional parameter on the device. To do this:

1. Select **Start** → **Run** → **cmd.exe**
2. Enter:

   ```
   netsh bridge show adapter
   netsh bridge set adapter <N> forcecompatmode=enable
   ```

   where **<N>** is the id number of the TAP adapter listed by the first command.

——— **Note** ———

Firewall software might block network traffic in the network bridge, and might result in a networking failure in the model. If the model does not work after configuration, check the firewall settings.

———

Usage of tap_setup_32.exe or tap_setup_64.exe

The following are example commands for use with `tap_setup_32.exe <commands> <params>...`:

help	Help information.
set_media	Configure the TAP devices to `Always Connected`.
set_perm	Configure the TAP devices to `Non-admin Accessible`.
restart	Restart the TAP devices.
list_dev	List available TAP devices and output to a file.
rename	Rename the device to **ARMx**.

install <inffile> <id> Install a TAP Win32 adapter.

remove <dev> Remove TAP Win32 adapters. Set **<dev>** to `All` to remove all tap devices.

setup <inffile> <id> Automated setup process.

Uninstalling

To uninstall all TAP adapters, run either:

- `ModelNetworking\remove_all_adapters_32.bat` for 32-bit Microsoft Windows
- `ModelNetworking\remove_all_adapters_64.bat` for 64-bit Microsoft Windows.

Do this from the base location of Fast Model Portfolio, default `C:\Program Files\ARM\FastModelPortfolio_X.Y`.

If the uninstallation does not delete the bridge, you can delete it manually:

1. Close all non-essential applications.
2. Select **Network Connections**.
3. Right-click on the bridge and select **Delete**.

5.6.4 Setting-up a network connection and configuring the networking environment for Linux

This section describes how to set up a network connection, and then to configure the networking environment for use by Fast Models on a Linux platform.

The following instructions assume that your network provides IP addresses by DHCP. If this is not the case, consult your network administrator.

Setting-up a network connection

To set up the network connection:

1. Ensure that you have installed the `brctl` utility on your system.

Note

This utility is included in the installation package, but you are recommended to use the standard Linux bridge utilities, which are included in the Linux distribution.

2. In a shell, change to the `bin` directory of your Fast Models installation. For example, `cd /FastModelPortfolio_X.Y/ModelNetworking`
3. You must run the following scripts from the directory in which they are installed, because they do not work correctly if run from any other location:
   - for a 32-bit operating system, run `add_adapter_32.sh` as root. For example, `sudo ./add_adapter_32.sh`
   - for a 64-bit operating system, run `add_adapter_64.sh` as root. For example, `sudo ./add_adapter_64.sh`
4. At the prompt **Please specify the TAP device prefix:(ARM)**, select **Enter** to accept the default.
5. At the prompt **Please specify the user list**, type a space-separated list of all users who are to use the model on the network, then select **Enter**. All entries in the list must be the names of existing user accounts on the host.
6. At the prompt **Please enter the network adapter which connect to the network:(eth0)**, select **Enter** to accept the default, or input the name of a network adapter that connects to your network.
7. At the prompt **Please enter a name for the network bridge that will be created:(armbr0)**, select **Enter** to accept the default, or input a name for the network bridge. You must not have an existing network interface on your system with the selected name.
8. At the prompt **Please enter the location where the init script should be written:(/etc/init.d/FMNetwork)**, select **Enter** to accept the default, or input another path with a new filename in an existing directory.
9. At the prompt **WARNING: The script will create a bridge which includes the local network adapter and tap devices. You may suffer temporary network loss. Do you want to proceed? (Yes or No)**, check all values input so far, and type **Yes** if you wish to proceed. If you type **No**, no changes are made to your system.
10. At the prompt that informs you of the changes that the script is to make to your system, input **Yes** if you are happy to accept these changes, or input **No** to leave your system unchanged.

    Note

    After entering **Yes**, you might temporarily lose network connectivity. Also, the IP address of the system might change.

11. The network bridge is disabled after the host system is reset. If you want the host system to be configured to support FVP bridged networking, you might have to create links to the `init` script (FMNetwork) in appropriate directories on your system. The script suggests some appropriate links for Red Hat Enterprise Linux 5.

Note

You only have to perform this procedure once, at the beginning.

Configuring the networking environment

Using System Canvas or a related Fast Models tool, load the required project or model and select your component. For more information, see the *Fast Models User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html`.

Set the following parameters on the **HostBridge** and **SMSC_91C111** components. Enter:

```
hostbridge.interfaceName="ARM<username>"
smsc_91c111.enabled=1
```

where **ARM<username>** is the adapter built into the network bridge.

Note

Firewall software might block network traffic in the network bridge, and might result in a networking failure in the model. If the model does not work after configuration, check the firewall settings.

Disabling and re-enabling networking

You can disable FVP networking without having to uninstall it. Use the installed `init` script (by default, `/etc/init.d/FMNetwork`) for this purpose.

To disable FVP networking, invoke the `init` script as root with the parameter `stop`. For example, `sudo /etc/init.d/FMNetwork stop`. To re-enable FVP networking, invoke the `init` script as root with the parameter `start`. For example, `sudo /etc/init.d/FMNetwork start`.

Warning

These operations remove/restore TAP devices and the network bridge. There is a temporary loss of network connectivity and your IP address might change.

Uninstalling networking

To uninstall networking:

1. In a shell window, change to the `bin` directory of your Fast Models installation. For example, `cd /FastModelPortfolio_X.Y/ModelNetworking`
2. Run `uninstall.sh` as root, passing the location of the `init` script (FMNetwork) as an argument. For example, `sudo ./uninstall.sh /etc/init.d/FMNetwork`

   You must run this script from the directory in which it is installed, because it does not work correctly if run from any other location.

   Warning

   There is a temporary loss of network connectivity and your IP address might change.

The uninstall script removes everything that can be safely removed. It does not remove:

- symlinks to the `init` script
- `/sbin/brctl`

You must remove any symlinks that you have created. Removing `brctl` is optional.

5.7 CADI sync watchpoints

Watchpoints are only supported by the CADI/Synchronous CADI (SCADI) interfaces of processor components. They are not supported by memory components like RAMDevice.

When the simulator hits a watchpoint, the CADI/SCADI interface emits a sequence, typically only one, of `modeChange(CADI_EXECMODE_Bpt, bptNumber)` callbacks, where `bptNumber` is the breakpoint ID of the watchpoint. After this sequence it sends a `modeChange(CADI_EXECMODE_Stop)` callback. Only after the debugger receives this `Stop` callback might it inspect the state of the model.

Additional information about the last of the hit watchpoints can be read from the following CADI registers declared by the processor in register group `Simulation`:

`memoryBptPC`

The PC of the instruction that caused the watchpoint to hit.

`memoryBptAccessVA`

Virtual address of the access or sub-access that caused the watchpoint to hit.

`memoryBptAccessSize`

Size in bytes of the access or sub-access that caused the watchpoint to hit.

`memoryBptAccessRW`

Type of access or sub-access that caused the watchpoint to hit: 1=read, 2=write, 3=both, 0=no access.

These registers are not memory (or CPnn-) mapped anywhere and are not accessible to target programs. These registers are read-only. All registers are 32-bit.

When multiple accesses have hit watchpoints at the same time, for example during the same instruction, the information contained in these registers is valid and consistent only for one of these accesses.

Whenever at least one CADI watchpoint is set on a processor, the syncLevel on that processor is at least 2.

5.8 Non-CADI sync watchpoints

This section describes syncLevel enum values, semantics and behaviors, with syncLevel values listed from fast to slow simulation, from less to more requirements:

`syncLevel 0: OFF`

The simulator runs as fast as possible. It does not permit inspection of the processor registers while the simulation is running, and does not stop synchronously when requested to do so.

Use case: normal fast simulation and normal CADI debugging while no CADI watchpoint is set.

`syncLevel 1: SYNC_STATE`

The simulation runs slightly slower than `syncLevel 0`. The up-to-date values of the processor registers, including PC and instruction count, can be read by SCADI. The simulation cannot be stopped synchronously.

Use case: external breakpoints that block the simulation, inspect state of processor/memory from within a peripheral or memory access.

`syncLevel 2: POST_INSN_IO`

As for `syncLevel 1`. In addition, the simulation can be stopped synchronously from within all LD/ST and similar instructions. The simulation stops immediately **after** the current LD/ST instruction has been completely executed (post instruction).

Main use cases: CADI watchpoints, external breakpoints, stopping from within LD/ST-related MTI callbacks.

`syncLevel 3: POST_INSN_ALL`

As for `syncLevel 2`. In addition, the simulation can be stopped synchronously from within any instruction. The simulation stops immediately **after** the current instruction has been completely executed (post instruction).

Main use case: a `Stop` from within arbitrary MTI callbacks such as the `INST` callback. A fallback for all use cases that do not fall into `syncLevel 0-2`.

5.8.1 Controlling and observing the syncLevel

CADI watchpoints automatically register and unregister for their required syncLevel (`POST_INSN_IO`). All other use cases must explicitly register and unregister for the specific syncLevel they require.

Users of syncLevel write to a set of non-architectural processor registers in the CADI and SCADI interface to register and unregister for specific syncLevels. Processor registers are more suitable than CADI parameters for exposing an interface that has side effects on writes and where values might change spontaneously.

This is the exposed interface to control and observe the value of syncLevel. All these registers are in the CADI/SCADI register group `Simulation` for each CT processor that contains non-architectural, simulator-specific registers. All are 32-bit integer registers. Users of syncLevel write to these registers to register and unregister for the syncLevel they require:

`syncLevelSyncStateRegister`

Users write to this register for `SYNC_STATE`. Write-only.

`syncLevelSyncStateUnregister`

Users write this to unregister for `SYNC_STATE`. Write-only.

`syncLevelPostInsnIORegister`

Users write to this register for `POST_INSN_LDST`. Write-only.

`syncLevelPostInsnIOUnregister`

Users write this to unregister for `POST_INSN_LDST`. Write-only.

`syncLevelPostInsnAllRegister`

Users write this to register for `POST_INSN_ALL`. Write-only.

`syncLevelPostInsnAllUnregister`

Users write this to unregister for `POST_INSN_ALL`. Write-only.

The following registers are only provided for debugging purposes and visibility in the debugger, and are usually not accessed by syncLevel users at all:

`syncLevel`

Current syncLevel. Read-only.

`syncLevelSyncStateCount`

User counter. Read-write (should be used read-only).

`syncLevelPostInsnIOCount`

User counter. Read-write (should be used read-only).

`syncLevelPostInsnAllCount`

User counter. Read-write (should be used read-only).

`minSyncLevel`

Same as `min_sync_level` parameter (as follows). Read-write.

The *Register and *Unregister registers are intended to be used, that is, written, by syncLevel users to register and unregister themselves. The value written is ignored and should be 0. Changes to the syncLevel become effective at the next *Stop event checkpoint*. In addition, these registers can be written any time before simulation is running, for example from the `init()` simulation phase. The change then takes effect immediately when the simulation is run.

All other registers are only present to make the debugging of these mechanisms and their users easier. The syncLevel enables you to see what kind of performance can currently be expected from the model. You must treat access to these registers as read-only. Writing to them is permitted, however, to permit debugging the syncLevel mechanisms.

These registers are not memory (or CPnn-) mapped anywhere, and are not accessible to target programs.

In addition to this debug register interface, there is a CADI parameter that can influence the syncLevel:

- `min_sync_level (default=0, type=int, runtime=true)`

This parameter enables you to control the minimum syncLevel by the CADI parameter interface. This is not the intended primary interface to control the syncLevel because it does not enable multiple independent syncLevel users to indicate their requirements to the simulator. It is intended primarily for debugging purposes and for situations where a single global setting of syncLevel is sufficient. Changes to this parameter at runtime are permitted and become effective on the next *Stop event checkpoint*. Reading this parameter value returns the `min_sync_level`, not the current syncLevel. This parameter is only an additional way of controlling the syncLevel and controls the same mechanisms as the register interface.

5.9 SCADI

A *Synchronous CADI* (SCADI) interface for components offers direct synchronous, but unsynchronized, register and memory read access. It enables you to request a *synchronous stop* of the simulation. The intention is that this interface is used by code that is inherently synchronized with the simulation, in particular from code that is part of the simulation itself (simulation thread). Examples for code that would be permitted to use this interface is SystemC or LISA peripheral device/bus access functions and MTI callback functions that are always called synchronously by the simulation itself.

Note

The SCADI interface is not a replacement for CADI in any way. It is an additional interface with clear semantics that differ slightly from CADI.

5.9.1 Intended use of CADI and SCADI

The design principles for CADI and SCADI are:

CADI Used by debuggers and simulation controllers to control the simulation. Execution control functions are always asynchronous and are usually called from a non-simulation thread, usually the debugger GUI thread. You can also use CADI to read/write registers, memory and so on, but only from non-simulation threads, for example only from the debugger thread.

SCADI Implements a specific subset of functions of CADI, that is, mainly read/write registers and memory, set and clear breakpoints, and get disassembly. You can only use SCADI from the simulation thread itself and from threads that can make sure on their own that the simulation is currently blocked, for example a debugger thread while it is sure that the simulation is in a stable state. SCADI is intended to be used from within peripheral read/write accesses while the simulation is running, or from within MTI callbacks that the simulation is running.

SCADI does not implement and execute control functions except `CADIExecStop()`, because only stop makes sense from within the simulation thread. SCADI is the Synchronous CADI interface. Asynchronous functions such as `CADIExecContinue()` and `CADIExecSingleStep()` are not supported by SCADI, so for these you must use CADI instead. The `SCADI::CADIExecStop()` is the exception, because for CADI this means “stop when it is convenient for the simulation”, which is asynchronous, but `CADIExecStop()` is synchronous and means “stop now”, which you enable with the appropriate syncLevel >=2.

The guideline is:

- use CADI for execution control
- use CADI or SCADI for read/write registers and memory, depending on the situation. That is:
  - use SCADI if the caller can make sure that the accesses are (potentially inherently) synchronized with the simulation
  - use CADI if called from a debugger thread that does not know exactly whether the simulation is running or is in a stable state.

5.9.2 Responsibilities of the SCADI caller

Synchronous interface means that the caller of SCADI functions is always responsible for ensuring that the simulation is in a stable state. Being in a stable state means that the simulation does not advance while the call is being made. The caller is also responsible for meeting all host thread and synchronization requirements of any external bus slaves that are directly or indirectly connected to the memory bus.

5.9.3 Obtaining the SCADI interface

You can obtain the SCADI interface from the same CAInterface pointers that are used to obtain the CADI and the MTI interfaces. This is done using the `CAInterface::ObtainInterface()` method. The interface names for the normal CADI and MTI interfaces are `CADI::IFNAME()` and `ComponentTraceInterface::IFNAME()` respectively.

The interface name for the SCADI interface to be passed into `ObtainInterface()` is "`eslapi.SCADI2`", the interface revision is 0. There are no `IFNAME()` and `IFREVISION()` static functions defined for SCADI, and a plain string constant and integer constant (0) has to be used instead.

The pointer returned by `CAInterface::ObtainInterface("eslapi.SCADI2")` must be cast into an `eslapi::CADI` class pointer, that is, the same class as for `eslapi.CADI2`, the normal CADI interface class.

Access to SCADI from within LISA components

To get access to the SCADI interface of the LISA component itself you can use the `getSCADI()` function. This returns the SCADI interface of the LISA component itself. To get access to the SCADI interfaces of other components in the system, you can use the `getGlobalInterface()` function. This returns a CAInteraface pointer. You then have to use the `ObtainInterface()` function on that pointer to obtain the SystemTraceInterface. This interface provides a list of all components in the system which provide trace data. This list can also be used to gain access to the SCADI interfaces of all these components. See *Example code*.

Access to SCADI from within System C

Access to the SCADI interfaces from within a System C system work similar to the LISA components: The generated System C platform (the generated System C wrapper around the Fast Model subsystem) provides a `getGlobalInterface()` function which returns a CAInterface pointer. This can be used to retrieve the SCADI interfaces in the same way as in the LISA case. See *Example code*.

Access to SCADI from MTI plugins

MTI plugins can gain access to SCADI interfaces by using the CAInterface pointer, which is passed as a parameter to the `RegisterSimulation()` function. This pointer has the same semantics as the CAInterface pointer returned by the `getGlobalInterface()` functions in the System C or LISA use cases.

Example code

This example code demonstrates how to use `getGlobalInterface()` to retrieve a SCADI interface pointer of a specific component. From LISA it is `getGlobalInterface()`, from within System C it is assumed that you have a pointer to the Fast Models platform called `platform`. From within a MTI plug-in you do not have to call `getGlobalInterface()`, and you can use the CAInterface pointer passed to `RegisterSimulation()`.

Example 5-1 Retrieving a SCADI interface pointer of a specific component

```
#include "sg/SGComponentRegistry.h"
eslapi::CADI *Peripheral::get_SCADI_interface(const char *instanceName)
{
  // get access to MTI interface (SystemTraceInterface)
  // - this code is for a System C peripheral which has access to the generated Fast
  //   Models 'platform'
  // - for LISA this would be just 'getGlobalInterface()'
  // - for MTI plugins this would be just using the CAInterface *caif pointer passed
  //   into RegisterSimulation()
  eslapi::CAInterface *globalInterface = platform->getGlobalInterface();
  const char *errorMessage = "(no error)";

  MTI::SystemTraceInterface *mti = 0;
  if (globalInterface)
     mti = sg::obtainComponentInterfacePointer<MTI::SystemTraceInterface>
                                        (globalInterface,
                                        "mtiRegistry",
                                        &errorMessage);
  if (mti == 0)
  {
     printf("ERROR:MTI interface not found! (%s)\n", errorMessage);
     return 0;
  }

  // find component with instanceName
  eslapi::CAInterface *compif = 0;
  for (MTI::SystemTraceInterface::TraceComponentIndex i = 0;
      i < mti->GetNumOfTraceComponents();
      i++)
  {
      const char *name = mti->GetComponentTracePath(i);
      if (verbose)
         printf("TEST MTI component '%s'\n", name);
      if (strcmp(name, instanceName) == 0)
         compif = mti->GetComponentTrace(i);
  }
  if (!compif)
  {
      printf("ERROR:instance '%s' not found while trying to get SCADI! \n",
              instanceName);
      return 0;
  }

  // get and return SCADI interface
  return static_cast<eslapi::CADI *>(compif->ObtainInterface("eslapi.SCADI2", 0, 0));
}
```

5.9.4 SCADI semantics

This section lists the available subset of functionality and the differences in semantics from the CADI interface.

The SCADI interface provides a subset of functionality of the CADI interface. All functions in SCADI differ from the CADI functions, that is, the caller is responsible for satisfying the constraints and requirements in terms of synchronization with the simulation.

These functions have the same semantics as in CADI:

- `CADIRegGetGroups()`

- `CADIRegGetMap()`
- `CADIRegGetCompount()`
- `CADIMemGetSpaces()`
- `CADIMemGetBlocks()`
- `CADIMemGetOverlays()`
- `CADIGetParameters()`
- `CADIGetParameterInfo()`
- `CADIXfaceGetFeatures()`

The following debug read access functions have the same semantics as in CADI, but the values returned by these read access functions might differ from the values returned by the corresponding CADI functions because some of the accessed resources values might change over time during the execution of an instruction. The values of resources that are modified by the current instruction are UNKNOWN, except the PC which always reflects the address of the last instruction that was issued. Reading and writing registers and memory while the simulation is running is generally only useful for `syncLevel >= SL_SYNC_STATE`. For SL_OFF, all registers and all memory locations are always UNKNOWN while the simulation is running.

- `CADIRegRead()`
- `CADIMemRead()`
- `CADIGetParameterValues()`
- `CADIGetInstructionCount()`
- `CADIGetPC()`
- `CADIGetDisassembler()`

The following write access functions always provide write access to the register in the `Simulation` register group. They also might provide limited write access semantics to certain resources. A SCADI implementation might choose not to support any write access to core registers and memory:

- `CADIRegWrite()`
- `CADIMemWrite()`
- `CADISetParameters()`

The breakpoint functions have nearly all the same semantics as in the CADI interface, except that changes to breakpoints only become effective at the next *Stop event checkpoint* defined by the current syncLevel. Changes might also only become visible in `CADIBptRead()` and `CADIBptGetList()` at this next synchronization point.

- `CADIBptGetList()`
- `CADIBptRead()`
- `CADIBptSet()`
- `CADIBptClear()`
- `CADIBptConfigure()`

Execution control is limited to stopping the simulation. The simulation stops at the next synchronization point as defined by the current syncLevel:

- `CADIExecStop()`

All other functions are not supported, and return `CADI_STATUS_CmdNotSupported`.

Chapter 6
Versatile Express Model: Platform and Components

This chapter describes components of Fast Models that are specific to the *Versatile™ Express* (VE) system model of the hardware platform. It contains the following sections:

Note

Using the visualization component can reduce simulation performance on some workstations. The simulation speed is affected by the setting of the `rate_limit-enable` parameter as described in *Parameters* on page 6-10.

6.1 About the Versatile Express baseboard components

The *Versatile Express* (VE) components have been specifically developed to model some of the functionality of the VE hardware.

A complete model implementation of the VE platform includes both VE-specific components and generic ones such as buses and timers. This section describes components used only in the VE model. Generic components are documented in other chapters of this book.

The VE model uses platform-specific components that are mainly located in the `%PVLIB_HOME%\examples\FVP_VE\LISA` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_VE/LISA` directory instead.

6.1.1 See also

Reference

- *VE and MPS FVP Reference Guide*
  - *Getting Started with Fixed Virtual Platforms*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0575-/Cihcdfjf.html`.
  - *Programmer's Reference for the VE FVPs*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0575-/Chdbeibh.html`.
- The hardware documentation for information about the functionality that the VE components simulate.

6.2 VE model memory map

Table 6-1 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 6-1 Memory map

Peripheral	Modeled	Address range	Size
NOR FLASH0 (CS0)	Yes	`0x00_00000000-0x00_03FFFFFF`	64MB
Reserved	-	`0x00_04000000-0x00_07FFFFFF`	64MB
NOR FLASH0 alias (CS0)	Yes	`0x00_08000000-0x00_0BFFFFFF`	64MB
NOR FLASH1 (CS4)	Yes	`0x00_0C000000-0x00_0FFFFFFF`	64MB
Unused (CS5)	-	`0x00_10000000-0x00_13FFFFFF`	-
PSRAM (CS1) - unused	No	`0x00_14000000-0x00_17FFFFFF`	-
Peripherals (CS2) see Table 6-3 on page 6-4.	Yes	`0x00_18000000-0x00_1BFFFFFF`	64MB
Peripherals (CS3) see Table 6-4 on page 6-4.	Yes	`0x00_1C000000-0x00_1FFFFFFF`	64MB
CoreSight and peripherals	No	`0x00_20000000-0x00_2CFFFFFF`[a]	-
Graphics space	No	`0x00_2D000000-0x00_2D00FFFF`	-
System SRAM	Yes	`0x00_2E000000-0x00_2EFFFFFF`	64KB
Ext AXI	No	`0x00_2F000000-0x00_7FFFFFFF`	-
4GB DRAM (in 32-bit address space)[b]	Yes	`0x00_80000000-0x00_FFFFFFFF`	2GB
Unused	-	`0x01_00000000-0x07_FFFFFFFF`	-
4GB DRAM (in 36-bit address space)[b]	Yes	`0x08_00000000-0x08_FFFFFFFF`	4GB
Unused	-	`0x09_00000000-0x7F_FFFFFFFF`	-
4GB DRAM (in 40-bit address space)[b]	Yes	`0x80_00000000-0xFF_FFFFFFFF`	4GB

a. The private peripheral region address `0x2c000000` is mapped in this region. The parameter PERIPHBASE can be used to map the peripherals to a different address.

b. The model contains only 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.

The model has a `secure_memory` option. When you enable this option, the memory map is changed for a number of peripherals as shown in Table 6-2:

Table 6-2 CS2 peripheral memory map for secure_memory option

Peripheral	Address range	Functionality with secure_memory enabled
NOR FLASH0 (CS0)	`0x00_00000000-0x00_0001FFFF`	Secure RO, aborts on non-secure accesses.
Reserved	`0x00_04000000-0x00_0401FFFF`	Secure SRAM, aborts on non-secure accesses.
NOR FLASH0 alias (CS0)	`0x00_08000000-0x00_7DFFFFFF`	Normal memory map, aborts on secure accesses.
Ext AXI	`0x00_7e000000-0x00_7FFFFFFF`	Secure DRAM, aborts on non-secure accesses.
4GB DRAM (in 32-bit address space)	`0x00_80000000-0xFF_FFFFFFFF`	Normal memory mpa, aborts on secure accesses.

Table 6-3 shows details of the memory map for peripherals in the CS2 region.

Table 6-3 CS2 peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[a]
VRAM - aliased	Yes	`0x00_18000000-0x00_19FFFFFF`	32MB	-
Ethernet (SMSC 91C111)	Yes	`0x00_1A000000-0x00_1AFFFFFF`	16MB	47
USB - unused	No	`0x00_1B000000-0x00_1BFFFFFF`	16MB	-

a. The Interrupt signal column lists the value to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

Table 6-4 shows details of the memory map for peripherals in the CS3 region.

Table 6-4 CS3 peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[a]
Local DAP ROM	No	`0x00_1C000000-0x00_1C00FFFF`	64KB	-
VE System Registers	Yes	`0x00_1C010000-0x00_1C01FFFF`	64KB	-
System Controller (SP810)	Yes	`0x00_1C020000-0x00_1C02FFFF`	64KB	-
TwoWire serial interface (PCIe)	No	`0x00_1C030000-0x00_1C03FFFF`	64KB	-
AACI (PL041)	Yes	`0x00_1C040000-0x00_1C04FFFF`	64KB	43
MCI (PL180)	Yes	`0x00_1C050000-0x00_1C05FFFF`	64KB	41, 42
KMI - keyboard (PL050)	Yes	`0x00_1C060000-0x00_1C06FFFF`	64KB	44
KMI - mouse (PL050)	Yes	`0x00_1C070000-0x00_1C07FFFF`	64KB	45

Table 6-4 CS3 peripheral memory map (continued)

Peripheral	Modeled	Address range	Size	GIC Int[a]
Reserved	-	`0x00_1C080000-0x00_1C08FFFF`	64KB	-
UART0 (PL011)	Yes	`0x00_1C090000-0x00_1C09FFFF`	64KB	37
UART1 (PL011)	Yes	`0x00_1C0A0000-0x00_1C0AFFFF`	64KB	38
UART2 (PL011)	Yes	`0x00_1C0B0000-0x00_1C0BFFFF`	64KB	39
UART3 (PL011)	Yes	`0x00_1C0C0000-0x00_1C0CFFFF`	64KB	40
VFS2	Yes	`0x00_1C0D0000-0x00_1C0DFFFF`	64KB	73
Reserved	-	`0x00_1C0E0000-0x00_1C0EFFFF`	64KB	-
Watchdog (SP805)	Yes	`0x00_1C0F0000-0x00_1C0FFFFF`	64KB	32
Reserved	-	`0x00_1C100000-0x00_1C10FFFF`	64KB	-
Timer-0 (SP804)	Yes	`0x00_1C110000-0x00_1C11FFFF`	64KB	34
Timer-1 (SP804)	Yes	`0x00_1C120000-0x00_1C12FFFF`	64KB	35
Reserved	-	`0x00_1C130000-0x00_1C15FFFF`	192KB	-
TwoWire serial interface (DVI) - unused	No	`0x00_1C160000-0x00_1C16FFFF`	64KB	-
Real-time Clock (PL031)	Yes	`0x00_1C170000-0x00_1C17FFFF`	64KB	36
Reserved	-	`0x00_1C180000-0x00_1C19FFFF`	128KB	-
CF Card - unused	No	`0x00_1C1A0000-0x00_1C1AFFFF`	64KB	
Reserved	-	`0x00_1C1B0000-0x00_1C1EFFFF`	256KB	-
Color LCD Controller (PL111)	Yes	`0x00_1C1F0000-0x00_1C1FFFFF`	64KB	46
Reserved	-	`0x00_1C200000-0x00_1FFFFFFF`	64KB	-

a. The Interrupt signal column lists the value to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

——— **Note** ———

The VE FVP implementation of memory does not require programming the memory controller with the correct values.

This means you must ensure that the memory controller is set up properly if you run an application on actual hardware. If this is not done, applications that run on a FVP might fail on actual hardware.

6.3 VE model parameters

Table 6-5 lists the VE FVP instantiation time parameters that you can change when you start the model.

The syntax to use in a configuration file is:

```
motherboard.component_name.parameter=value
```

Table 6-5 VE model instantiation parameters

Component name	Parameter	Description	Type	Values	Default
`ve_sysregs`	`user_switches_value`	switch S6 setting	Integer	see *Switch S6*	`0`
`flashldr_0`	`fname`	path to flash image file	String	valid filename	`""`
`flashldr_1`	`fname`	path to flash image file	String	valid filename	`""`
`mmc`	`p_mmc_file`	multimedia card filename	String	valid filename	`mmc.dat`
`pl111_clcd_0`	`pixel_double_limit`	sets threshold in horizontal pixels below which pixels sent to framebuffer doubled in size in both dimensions	Integer	-	`0x12c`
`sp810_sysctrl`	`use_s8`	indicates whether to read `boot_switches_value`	Boolean	`true` or `false`	`false`
`vfs2`	`mount`	mount point for the host file system	String	directory path on host	`""`

6.3.1 Switch S6

Switch S6 is equivalent to the Boot Monitor configuration switch on the VE hardware. Default settings are listed in Table 6-6.

If you have the standard ARM Boot Monitor flash image loaded, the setting of switch S6-1 changes what happens on model reset. Otherwise, the function of switch S6 is implementation dependent.

To write the switch position directly to the S6 parameter in the model, you must convert the switch settings to an integer value from the equivalent binary, where 1 is on and 0 is off.

Table 6-6 Default positions for VE System Model switch

Switch	Default Position	Function in default position
S6-1	OFF	Displays prompt permitting Boot Monitor command entry after system start.

Table 6-6 Default positions for VE System Model switch (continued)

Switch	Default Position	Function in default position
S6-2	OFF	See Table 6-7.
S6-3	OFF	See Table 6-7.
S6-4 to S6-8	OFF	Reserved for application use.

If S6-1 is in the ON position, the Boot Monitor executes the boot script that was loaded into flash. If there is no script, the Boot Monitor prompt is displayed.

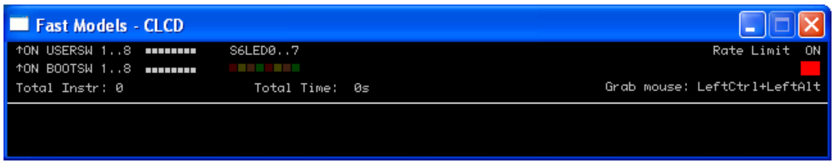
The settings of S6-2 and S6-3 affect STDIO source and destination on model reset as defined in Table 6-7.

Table 6-7 STDIO redirection

S6-2	S6-3	Output	Input	Description
OFF	OFF	UART0	UART0	STDIO autodetects whether to use semihosting I/O or a UART. If a debugger is connected, STDIO is redirected to the debugger output window, otherwise STDIO goes to UART0.
OFF	ON	UART0	UART0	STDIO is redirected to UART0, regardless of semihosting settings.
ON	OFF	CLCD	Keyboard	STDIO is redirected to the CLCD and keyboard, regardless of semihosting settings.
ON	ON	CLCD	UART0	STDIO output is redirected to the LCD and input is redirected to the keyboard, regardless of semihosting settings.

6.4 VEVisualisation component

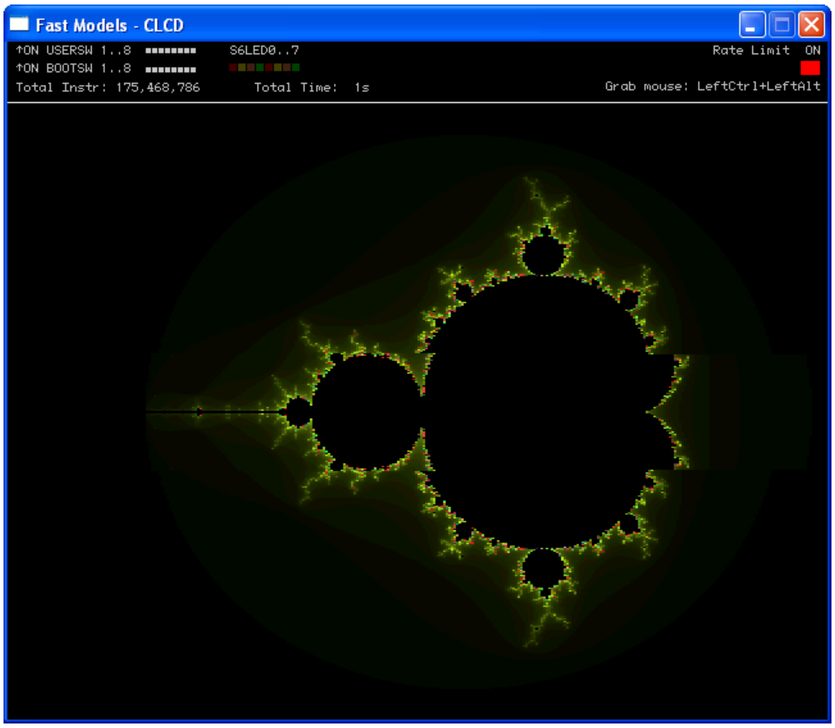
The VEVisualisation component can generate events when the host mouse or keyboard are used when the visualization window is in focus. For example, the switch elements can be toggled from the visualization window. Figure 6-1 shows the startup CLCD visualization window for the VE FVP.



```
↑ON USERSW 1..8 ■■■■■■■■    S6LED0..7                      Rate Limit  ON
↑ON BOOTSW 1..8 ■■■■■■■■
Total Instr: 0                 Total Time:  0s      Grab mouse: LeftCtrl+LeftAlt
```

Figure 6-1 VE Fixed Virtual Platform CLCD visualization window

When a suitable application or system image is loaded, and has configured the PL110_CLCD controller registers, the window expands to show the contents of the frame buffer. Figure 6-2 shows an example using the `brot.axf` image.



```
↑ON USERSW 1..8 ■■■■■■■■    S6LED0..7                      Rate Limit  ON
↑ON BOOTSW 1..8 ■■■■■■■■
Total Instr: 175,468,786       Total Time:  1s      Grab mouse: LeftCtrl+LeftAlt
```

Figure 6-2 VE FVP CLCD with brot.axf image

Further details on how to use the features provided in the CLCD visualization window are given elsewhere. See the *RealView Development Suite Real-Time System Models User Guide*.

Figure 6-3 on page 6-9 shows a view of the VEVisualisation component in System Canvas. This component can be found in the `%PVLIB_HOME%\examples\FVP_VE\LISA\` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_VE/LISA` directory.

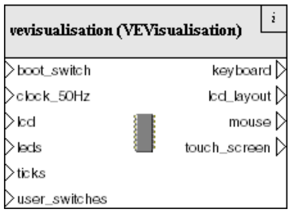


Figure 6-3 VEVisualisation in System Canvas

The VEVisualisation component is written in LISA+.

6.4.1 Ports

Table 6-8 gives a brief description of the VEVisualisation component ports. For more information, see the VE FVP documentation. See the *RealView Development Suite Real-Time System Models User Guide*.

Table 6-8 VEVisualisation component ports

Name	Port Protocol	Type	Description
`ticks`	InstructionCount	slave	Connection from a PV processor model to show current instruction count.
`lcd`	LCD	slave	Connection from a CLCD controller for visualization of the frame buffer.
`leds`	ValueState	slave	Displays state using the eight colored LEDs on the status bar.
`user_switches`	ValueState	slave	Provides state for the eight User DIP switches on the left side of the CLCD status bar, equivalent to switch S6 on VE hardware.
`boot_switch`	ValueState	slave	Provides state for the eight Boot DIP switches on the right side of the CLCD status bar.
`keyboard`	KeyboardStatus	master	Output port providing key change events when the visualization window is in focus.
`mouse`	MouseStatus	master	Output port providing mouse movement and button events when the visualization window is in focus.
`clock_50Hz`	ClockSignal	slave	50Hz clock input.
`touch_screen`	MouseStatus	master	Provides mouse events when the visualization window is in focus.

Table 6-8 VEVisualisation component ports (continued)

Name	Port Protocol	Type	Description
`lcd_layout`	LCDLayoutInfo	master	Layout information for alphanumeric LCD display.
`daughter_leds`	ValueState	slave	A read/write port to read and set the value of the LEDs. 1 bit per LED, LSB left-most, up to 32 LEDs available. The LEDs appear only when parameter `daughter_led_count` is set to nonzero.
`daughter_user_switches`	ValueState	slave	A read port to return the value of the daughter user switches. Write to this port to set the value of the switches, and use during reset only. LSB is left-most, up to 32 switches available.

6.4.2 Additional protocols

The VEVisualisation component has three additional protocols, all of which are described in a separate document. See Chapter 5 *Peripheral and Interface Components*.

The three protocols are:
- KeyboardStatus
- MouseStatus
- LCD.

6.4.3 Parameters

Table 6-9 provides a description of the configuration parameters for the VEVisualisation component.

Table 6-9 VEVisualisation configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`cluster0_name`	Label for cluster 0 performance values	String		Cluster0
`cluster1_name`	Label for cluster 1 performance values	String		Cluster1
`cpu_name`	Processor name displayed in window title	String		
`daughter_led_count`	Set to nonzero to display up to 32 LEDs. See `daughter_leds` in Table 6-8 on page 6-9.	Integer	`0-32`	`0`
`daughter_user_switch_count`	Set this parameter to display up to 32 switches. See `daughter_user_switches` in Table 6-8 on page 6-9.	Integer	`0-32`	`0`
`disable_visualisation`	Disable the VEVisualisation component on model startup	Boolean	`true/false`	`false`

Table 6-9 VEVisualisation configuration parameters (continued)

Parameter name	Description	Type	Allowed value	Default value
`rate_limit-enable`	Restrict simulation speed so that simulation time more closely matches real time rather than running as fast as possible	Boolean	`true`/`false`	`true`
`trap_key`	Trap key that works with left Ctrl key to toggle mouse display	Integer	valid ATKeyCode key value[a]	`74`[b]
`window_title`	Window title (`%cpu%` is replaced by `cpu_name`)	String		Fast Models - CLCD `%cpu%`

a. See the header file, `%PVLIB_HOME%\components\KeyCode.h`, for a list of ATKeyCode values. On Linux, use `$PVLIB_HOME/components/KeyCode.h`.

b. This is equivalent to the left **Alt** key, so pressing Left Alt and Left Ctrl simultaneously toggles the mouse display.

Note

Setting the `rate_limit-enable` parameter to `true` (the default) prevents the simulation from running too fast on fast workstations and enables timing loops and mouse actions to work correctly. The overall simulation speed, however, is reduced.

If your priority is high simulation speed, set `rate_limit-enable` to `false`.

6.4.4 Registers

The VEVisualisation component has no registers.

6.4.5 Debug Features

The VEVisualisation component has no debug features.

6.4.6 Verification and testing

The VEVisualisation component has been tested by use as an I/O device for booting Linux and other operating systems.

6.4.7 Performance

Use of elements in the status bar are not expected to significantly affect the performance of a PV system. However, applications that make heavy use of re-drawing the contents of the frame buffer incur overhead through GUI interactions on the host OS.

6.4.8 Library dependencies

The VEVisualisation component relies on the *Simple DirectMedia Layer* (SDL) libraries, specifically `libSDL-1.2.so.0.11.2`. This library is bundled with the Model Library and is also available as a rpm for Red Hat Enterprise Linux. For more information, see the SDL, `http://www.libsdl.org` web site.

6.5 VE_SysRegs component

The VE system registers component is a programmer's view model of the VE model status and system control registers. For a detailed description of the behavior of the component, see the hardware documentation.

Figure 6-4 shows a view of the component in System Canvas.

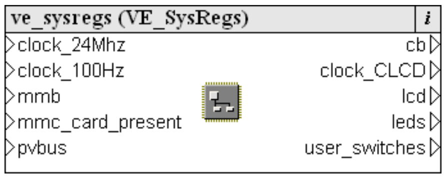


Figure 6-4 VE_SysRegs in System Canvas

This component is written in LISA+.

6.5.1 Ports

Table 6-10 provides a brief description of the VE_SysRegs component ports. For more information, see the hardware documentation.

Table 6-10 VE_SysRegs ports

Name	Port Protocol	Type	Description
`cb[0-1]`	VECBProtocol	master	The *Configuration Bus* (CB) controls the power and reset sequence.
`clk_24Mhz`	ClockSignal	slave	Reference clock for internal counter register.
`clk_100Hz`	ClockSignal	slave	Reference clock for internal counter register.
`clock_CLCD`	ClockRate Control	master	The clock for the LCD controller.
`lcd`	LCD	master	Multi-media bus interface output to the LCD.
`leds`	ValueState	master	Displays state of the SYS_LED register using the eight colored LEDs on the status bar.
`mmb[0-2]`	LCD	slave	Multi-media bus interface input.
`mmc_card_present`	StateSignal	slave	Indicates if an image representing an *MultiMedia Card* (MMC) has been configured.
`pvbus`	PVBus	slave	Slave port for connection to PV bus master/decoder.
`user_switches`	ValueState	master	Provides state for the eight User DIP switches on the left side of the CLCD status bar, equivalent to switch S6 on VE hardware.

6.5.2 Additional protocols

The VE_SysRegs component has no additional protocols.

6.5.3 Parameters

Table 6-11 provides a description of the configuration parameters for the VE_SysRegs component.

Table 6-11 VE_SysRegs configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`exit_on_shutdown`	Used to shut down the system. When `true`, if software uses the `SYS_CFGCTRL` function `SYS_CFG_SHUTDOWN`, then the simulator shuts down and exits[a].	Boolean	`true`/`false`	`false`
`sys_proc_id0`	Procesor ID register at CoreTile Express Site 1	Integer	`0x0c000000`	`0x0c000000`
`sys_proc_id1`	Processor ID at CoreTile Express Site 2	Integer	`0xff000000`	`0xff000000`
`tilePresent`	Tile fitted	Boolean	`true`/`false`	`true`
`user_switches_value`	User switches	Integer	`0`	`0`

a. For more information on the `SYS_CFGCTRL` function values, see the *Motherboard Express µATX V2M-P1 Technical Reference Manual*.

6.5.4 Registers

Table 6-12 provides a description of the configuration registers for the VE_SysRegs component.

Table 6-12 VE_SysRegs registers

Register name	Offset	Access	Description
SYS_ID	`0x00`	read/write	system identity
SYS_SW	`0x04`	read/write	bits[7:0] map to switch S6
SYS_LED	`0x08`	read/write	bits[7:0] map to user LEDs
SYS_100HZ	`0x24`	read only	100Hz counter
SYS_FLAGS	`0x30`	read/write	general purpose flags
SYS_FLAGSCLR	`0x34`	write only	clear bits in general purpose flags
SYS_NVFLAGS	`0x38`	read/write	general purpose non-volatile flags
SYS_NVFLAGSCLR	`0x3C`	write only	clear bits in general purpose non-volatile flags
SYS_MCI	`0x48`	read only	MCI

Table 6-12 VE_SysRegs registers (continued)

Register name	Offset	Access	Description
SYS_FLASH	`0x4C`	read/write	flash control
SYS_CFGSW	`0x58`	read/write	boot select switch
SYS_24MHZ	`0x5C`	read only	24MHz counter
SYS_MISC	`0x60`	read/write	miscellaneous control flags
SYS_DMA	`0x64`	read/write	DMA peripheral map
SYS_PROCID0	`0x84`	read/write	processor ID
SYS_PROCID1	`0x88`	read/write	processor ID
SYS_CFGDATA	0xA0	read/write	Data to be read/written from/to motherboard controller
SYS_CFGCTRL	0xA4	read/write	Control data transfer to motherboard controller
SYS_CFGSTAT	0xA8	read/write	Status of data transfer to motherboard

6.5.5 Debug Features

The VE_SysRegs component has no debug features.

6.5.6 Verification and testing

The VE_SysRegs component has been tested as part of the Emulation Board model.

6.5.7 Performance

The VE_SysRegs component is not expected to significantly affect the performance of a PV system.

6.5.8 Library dependencies

The VE_SysRegs component has no dependencies on external libraries.

6.6 Other VE model components

This section briefly describes the purpose of selected additional components included with the VE model platform model.

6.6.1 VEConnector and VESocket components

The VEConnector and VESocket components are used to forward Interrupts, CLK and PVBus transactions between the VEBaseboard and the CoreTile. As their names indicate, the two components emulate the connections in a CoreTile to the VE model.

6.6.2 VEInterruptForwarder component

The VEInterruptForwarder component is an assistant component for VESocket. VEInterruptForwarder receives all interrupts and is responsible for telling the socket which interrupt happened.

6.6.3 VERemapper component

The VERemapper component is a control component involved with remapping logic on the VE model. The component receives the **remap_clear** signal from the SP810_SysCtrl component and boot_switch value from the VE_SysReg component to determine which devices are to be remapped and unremapped. See *VE_SysRegs component* on page 6-12. The SP810_SysCtrl component is described in Chapter 5 *Peripheral and Interface Components*.

6.7 Differences between the VE hardware and the system model

This section describes features of the VE hardware that are not implemented in the models, or that have significant differences in implementation:

- *Memory map*
- *Memory aliasing*
- *Features not present in the model*
- *Features partially implemented in the model*
- *Restrictions on the processor models* on page 6-17
- *Timing considerations* on page 6-17.

6.7.1 Memory map

The model is based on the memory map of the hardware VE platform, but is not intended to be an accurate representation of 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, and those areas that are documented as reserved. Accessing these regions from the host processor results in the model presenting a warning.

6.7.2 Memory aliasing

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_0000000` and again at `0x80_00000000`. The aliasing of DRAM then repeats from `0x81_00000000` up to `0xFF_FFFFFFFF`.

6.7.3 Features not present in the model

The following features present on the hardware version of the VE motherboard are not implemented in the system models:

- two-wire serial bus interfaces
- USB interfaces
- PCI Express interfaces
- compact flash
- *Digital Visual Interface* (DVI)
- debug and test interfaces
- *Dynamic Memory Controller* (DMC)
- *Static Memory Controller* (SMC).

Note

For more information on memory-mapped peripherals, see *VE model memory map* on page 6-3.

6.7.4 Features partially implemented in the model

The following feature present on the hardware version of the VE motherboard is only partially implemented in the Fixed Virtual Platforms:

- *Sound* on page 6-17.

Partial implementation means that some of the components are present but the 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 as in the VE hardware, but with a limited number of sample rates.

6.7.5 Restrictions on the processor models

The following general restrictions apply to the Fixed Virtual Platform implementations of ARM processors:

- The simulator does not model cycle timing. In aggregate, all instructions execute in one processor master clock cycle, with the exception of Wait For Interrupt.
- Write buffers are not modeled on all processors.
- Most aspects of TLB behavior are implemented in the models. In ARMv7 models and later, the TLB memory attribute settings are used when stateful cache is enabled.
- No device-accurate MicroTLB is implemented.
- A single memory access port is implemented. The port combines accesses for instruction, data, DMA and peripherals. Configuration of the peripheral port memory map register is ignored.
- All memory accesses are atomic and are performed in programmer’s view order. Unaligned accesses are always performed as byte transfers.
- Some instruction sequences are executed atomically so that system time does advance during their execution. This can sometimes have an effect in sequential access of device registers where devices are expecting time to move on between each access.
- Interrupts are not taken at every instruction boundary.
- Integration and test registers are not implemented.
- Not all CP14 debug registers are implemented on all processors.
- Breakpoint types supported directly by the model are:
  - single address unconditional instruction breakpoints
  - single address unconditional data breakpoints
  - unconditional instruction address range breakpoints
- Processor exception breakpoints are supported by pseudoregisters in the debugger. Setting an exception register to a nonzero value stops execution on entry to the associated exception vector.
- Performance counters are not implemented on all models.

6.7.6 Timing considerations

The Fixed Virtual Platforms provide an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

When code interacts with real world devices like 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 keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

The VE FVPs provide the Rate Limit feature to match simulation time to match 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.

Chapter 7
Emulation Baseboard Model: Platform and Components

This chapter describes components of Fast Models that are specific to the EB system model of the hardware platform. It contains the following sections:

Note

Using the visualization component can reduce simulation performance on some workstations. The simulation speed is affected by the setting of the `rate_limit-enable` parameter as described in *Parameters* on page 7-11.

7.1 About the Emulation Baseboard components

The *Emulation Baseboard* (EB) components have been specifically developed to model in software some of the functionality of the RealView Emulation Baseboard hardware.

A complete model implementation of the EB platform includes both EB-specific components and generic ones such as buses and timers. This section describes components used only in the *RealView Emulation Baseboard* (EB) model. Generic components are documented in other chapters of this book.

The EB model uses platform-specific components that are mainly located in the `%PVLIB_HOME%\examples\FVP_EB\LISA` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_EB/LISA` directory instead. Further information on the Emulation Baseboard model, including its use, can be found elsewhere. See the *RealView Development Suite Real-Time System Models User Guide*. See the hardware documentation for information about the functionality that the EB components simulate. See *RealView Emulation Baseboard User Guide (Lead Free)*.

7.2 Emulation Baseboard memory map

Table 7-1 lists the locations and interrupts for memory, peripherals, and controllers used in the EB Fixed Virtual Platforms. For more information about the controllers and peripherals, see the *Emulation Baseboard User Guide*.

Table 7-1 Memory map and interrupts for standard peripherals

Peripheral	Modeled	Address range	Bus	Size	GIC Int[a]	DCCI Int[b]
Dynamic memory	Yes	`0x00000000-0x0FFFFFFF`	AHB	256MB	-	-
System registers (see *Status and system control registers* on page 7-24)	Yes	`0x10000000-0x10000FFF`	APB	4KB	-	-
SP810 System Controller	Yes	`0x10001000-0x10001FFF`	APB	4KB	-	-
Two-Wire Serial Bus Interface	No	`0x10002000-0x10002FFF`	APB	4KB	-	-
Reserved	-	`0x10003000-0x10003FFF`	APB	4KB	-	-
PL041 *Advanced Audio CODEC Interface* (AACI)	Partial[c]	`0x10004000-0x10004FFF`	APB	4KB	51	51
PL180 *MultiMedia Card Interface* (MCI)	Partial[d]	`0x10005000-0x10005FFF`	APB	4KB	49, 50	49, 50
Keyboard/Mouse Interface 0	Yes	`0x10006000-0x10006FFF`	APB	4KB	52	7
Keyboard/Mouse Interface 1	Yes	`0x10007000-0x10007FFF`	APB	4KB	53	8
Character LCD Interface	No	`0x10008000-0x10008FFF`	APB	4KB	55	55
UART 0 Interface	Yes	`0x10009000-0x10009FFF`	APB	4KB	44	4
UART 1 Interface	Yes	`0x1000A000-0x1000AFFF`	APB	4KB	45	5
UART 2 Interface	Yes	`0x1000B000-0x1000BFFF`	APB	4KB	46	46
UART 3 Interface	Yes	`0x1000C000-0x1000CFFF`	APB	4KB	47	47
Synchronous Serial Port Interface	Yes	`0x1000D000-0x1000DFFF`	APB	4KB	43	43
Smart Card Interface	No	`0x1000E000-0x1000EFFF`	APB	4KB	62	62
Reserved	-	`0x1000F000-0x1000FFFF`	APB	4KB	-	-
SP805 Watchdog Interface	Yes	`0x10010000-0x10010FFF`	APB	4KB	32	32
SP804 Timer modules 0 and 1 interface (Timer 1 starts at `0x10011020`)	Yes	`0x10011000-0x10011FFF`	APB	4KB	36	1
SP804 Timer modules 2 and 3 interface (Timer 3 starts at `0x10020020`)	Yes	`0x10012000-0x10012FFF`	APB	4KB	37	2
PL061 GPIO Interface 0	Yes	`0x10013000-0x10013FFF`	APB	4KB	38	38
PL061 GPIO Interface 1	Yes	`0x10014000-0x10014FFF`	APB	4KB	39	39
PL061 GPIO Interface 2 (miscellaneous onboard I/O)	Yes	`0x10015000-0x10015FFF`	APB	4KB	40	40
Reserved	-	`0x10016000-0x10016FFF`	APB	4KB	-	-
PL030 Real Time Clock Interface	Yes	`0x10017000-0x10017FFF`	APB	4KB	-	-

Table 7-1 Memory map and interrupts for standard peripherals (continued)

Peripheral	Modeled	Address range	Bus	Size	GIC Int[a]	DCCI Int[b]
Dynamic Memory Controller configuration	Partial[e]	`0x10018000-0x10018FFF`	APB	4KB	-	-
PCI controller configuration registers	No	`0x10019000-0x10019FFF`	AHB	4KB	-	-
Reserved	-	`0x1001A000-0x1001FFFF`	APB	24KB	-	-
PL111 Color LCD Controller	Yes	`0x10020000-0x1002FFFF`	AHB	64KB	55	55
DMA Controller configuration registers	Yes	`0x10030000-0x1003FFFF`	AHB	64KB	-	-
Generic Interrupt Controller 1	Yes[f]	`0x10040000-0x1004FFFF`	AHB	64KB	-	-
Generic Interrupt Controller 2 (nFIQ for tile 1)	No[g]	`0x10050000-0x1005FFFF`	AHB	64KB	-	-
Generic Interrupt Controller 3 (nIRQ for tile 2)	No[g]	`0x10060000-0x1006FFFF`	AHB	64KB	-	-
Generic Interrupt Controller 4 (nFIQ for tile 2)	No[g]	`0x10070000-0x1007FFFF`	AHB	64KB	-	-
PL350 Static Memory Controller configuration[h]	Yes	`0x10080000-0x1008FFFF`	AHB	64KB	-	-
Reserved	-	`0x10090000-0x100EFFFF`	AHB	448MB	-	-
Debug Access Port (DAP) ROM table Some debuggers read information on the target processor and the debug chain from the DAP table.	No	`0x100F0000-0x100FFFFF`	AHB	64KB	-	-
Reserved	-	`0x10100000-0x1FFFFFFF`	-	255MB	-	-
Reserved	-	`0x20000000-0x3FFFFFFF`	-	512MB	-	-
NOR Flash	Yes[i]	`0x40000000-0x43FFFFFF`	AXI	64MB	-	-
Disk on Chip	No	`0x44000000-0x47FFFFFF`	AXI	64MB	41	41
SRAM	Yes	`0x48000000-0x4BFFFFFF`	AXI	64MB	-	-
Configuration flash	No	`0x4C000000-0x4DFFFFFF`	AXI	32MB	-	-
Ethernet	Yes[j]	`0x4E000000-0x4EFFFFFF`	AXI	16MB	60	60
USB	No	`0x4F000000-0x4FFFFFFF`	AXI	16MB	-	-
PISMO expansion memory	No	`0x50000000-0x5FFFFFFF`	AXI	256MB	58	58
PCI interface bus windows	No	`0x60000000-0x6FFFFFFF`	AXI	256MB	-	-
Dynamic memory (mirror)	Yes	`0x70000000-0x7FFFFFFF`	AXI	256MB	-	-
Memory tile (2nd CoreTile)	Yes	`0x80000000-0xBFFFFFFF`	AXI	0-1GB	-	-

a. The Interrupt signal column lists the value to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

b. For the EB model that uses the Cortex-A9, a DIC is used as the interrupt controller instead of the GIC. The numbers in this column are the interrupt numbers used by the DCCI system.

c. See *Sound* on page 7-23.

d. The implementation of the PL180 is currently limited, so not all features are present.

e. See *Differences between the EB hardware and the system model* on page 7-23.

f. The Cortex™-A9 models implement an internal GIC. This is mapped at `0x1F000000 - 0x1F001FFF`.

g. The EB FVP GICs are not the same as those implemented on the EB hardware as the register map is different. See *Generic Interrupt Controller* on page 7-24.

h. Although the EB hardware uses the PL093 static memory controller, the model implements PL350. These are functionally equivalent.

i. This peripheral is implemented in the IntelStrataFlashJ3 component in the EB FVP.

j. This peripheral is implemented in the SC91C111 component in the EB FVP.

Note

The EB FVPs implementation of memory does not require programming the memory controller with the correct values. This means you must ensure that the memory controller is set up properly if run an application on actual hardware. If this is not done, applications that run on a FVP might fail on actual hardware.

7.3 Emulation Baseboard parameters

Table 7-2 lists the EB FVP instantiation time parameters that you can change when you start the model.

The syntax to use in a configuration file is:

```
baseboard.component_name.parameter=value
```

Table 7-2 EB Baseboard Model instantiation parameters

Component name	Parameter	Description	Type	Values	Default
`eb_sysregs_0`	`user_switches_value`	switch S6 setting	Integer	see *Switch S6*	`0`
`eb_sysregs_0`	`boot_switch_value`	switch S8 setting	Integer	see *Switch S8* on page 7-7	`0`
`flashldr_0`	`fname`	path to flash image file	String	valid filename	`""`
`flashldr_1`	`fname`	path to flash image file	String	valid filename	`""`
`mmc`	`p_mmc_file`	multimedia card filename	String	valid filename	`mmc.dat`
`pl111_clcd_0`	`pixel_double_limit`	sets threshold in horizontal pixels below which pixels sent to framebuffer doubled in size in both dimensions	Integer	-	`0x12c`
`smc`	`REMAP`	indicates which channel of the SMC is bootable	Integer	-1, 0-7	-1 (no remap)
`sp810_sysctrl`	`use_s8`	indicates whether to read `boot_switches_value`	Boolean	`true` or `false`	`false`
`vfs2`	`mount`	mount point for the host file system	String	directory path on host	`""`
`sdram_size`	`sdram_size`	size of memory logic tile in second installed CoreTile	Integer	`0x00000000` to `0x10000000` (1GB)	`0` (0GB)

7.3.1 Switch S6

Switch S6 is equivalent to the Boot Monitor configuration switch on the EB hardware. Default settings are listed in Table 7-3 on page 7-7.

If you have the standard ARM Boot Monitor flash image loaded, the setting of switch S6-1 changes what happens on model reset. Otherwise, the function of switch S6 is implementation dependent.

To write the switch position directly to the S6 parameter in the model, you must convert the switch settings to an integer value from the equivalent binary, where 1 is on and 0 is off.

Table 7-3 Default positions for EB System Model switch S6

Switch	Default Position	Function in default position
S6-1	OFF	Displays prompt permitting Boot Monitor command entry after system start.
S6-2	OFF	See Table 7-4.
S6-3	OFF	See Table 7-4.
S6-4 to S6-8	OFF	Reserved for application use.

If S6-1 is in the ON position, the Boot Monitor executes the boot script that was loaded into flash. If there is no script, the Boot Monitor prompt is displayed.

The settings of S6-2 and S6-3 affect STDIO source and destination on model reset as defined in Table 7-4.

Table 7-4 STDIO redirection

S6-2	S6-3	Output	Input	Description
OFF	OFF	UART0	UART0	STDIO autodetects whether to use semihosting I/O or a UART. If a debugger is connected, STDIO is redirected to the debugger output window, otherwise STDIO goes to UART0.
OFF	ON	UART0	UART0	STDIO is redirected to UART0, regardless of semihosting settings.
ON	OFF	CLCD	Keyboard	STDIO is redirected to the CLCD and keyboard, regardless of semihosting settings.
ON	ON	CLCD	UART0	STDIO output is redirected to the LCD and input is redirected to the keyboard, regardless of semihosting settings.

For more information on Boot Monitor configuration and commands, see the *Emulation Baseboard User Guide (Lead Free)*.

7.3.2 Switch S8

Switch S8 is disabled by default. To enable it, you must change the state of the parameter `baseboard.sp810_sysctrl.use_s8` to `true` before you start the model. See *Emulation Baseboard parameters* on page 7-6.

If you have a Boot Monitor flash image loaded, switch S8 enables you to remap boot memory.

On reset, the EB hardware starts to execute code at `0x0`, which is typically volatile DRAM. You can put the contents of non-volatile RAM at this location by setting the S8 switch in the EB FVP CLCD as shown in Table 7-5. The settings take effect on model reset.

Table 7-5 EB System Model switch S8 settings

Switch S8[4:1]	Memory Range	Description
0000	`0x40000000–0x4FEFFFFF`	NOR flash remapped to `0x0`
0001	`0x44000000–0x47FFFFFF`	NOR flash remapped to `0x0`
0010	`0x48000000–0x4BFFFFFF`	SRAM remapped to `0x0`

Note

Attempting to change switch S8 settings after the model has started, for example by using the CLCD DIP switches, might lead to unpredictable behavior.

7.4 EBVisualisation component

The EBVisualisation component can generate events when the host mouse or keyboard are used when the visualization window is in focus. For example, the switch elements can be toggled from the visualization window. Figure 7-1 shows the startup CLCD visualization window for the EB FVP.

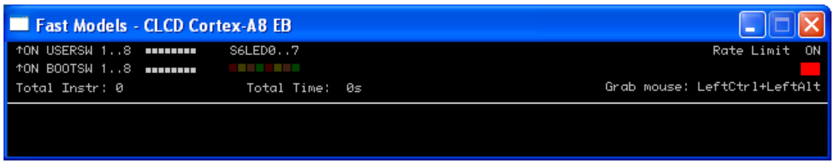


Figure 7-1 EB Fixed Virtual Platform CLCD visualization window

When a suitable application or system image is loaded, and has configured the PL110_CLCD controller registers, the window expands to show the contents of the frame buffer. Figure 7-2 shows an example using the `brot.axf` image.

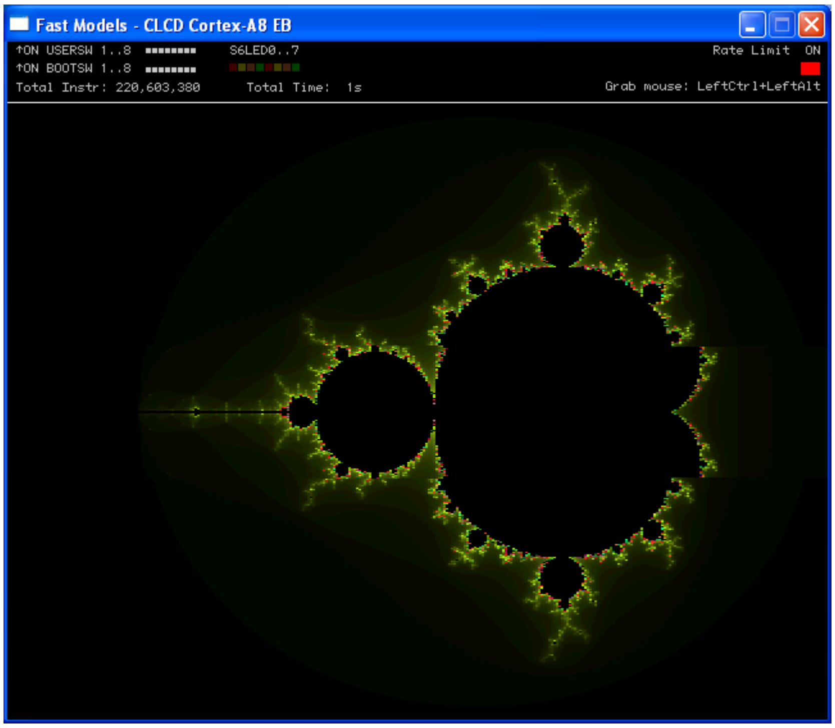


Figure 7-2 EB FVP CLCD with brot.axf image

Further details on how to use the features provided in the CLCD visualization window are given elsewhere. See the *RealView Development Suite Real-Time System Models User Guide*.

Figure 7-3 on page 7-10 shows a view of the EBVisualisation component in System Canvas. This component can be found in the `%PVLIB_HOME%\examples\FVP_EB\LISA\` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_EB/LISA` directory.

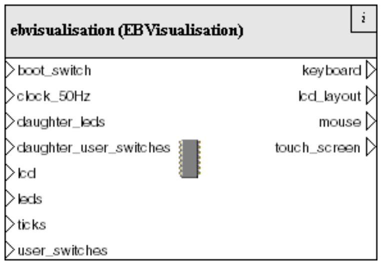


Figure 7-3 EBVisualisation in System Canvas

The EBVisualisation component is written in LISA+.

7.4.1 Ports

Table 7-6 gives a brief description of the EBVisualisation component ports. For more information, see the EB FVP documentation. See the *RealView Development Suite Real-Time System Models User Guide*.

Table 7-6 EBVisualisation component ports

Name	Port Protocol	Type	Description
`ticks`	InstructionCount	slave	Connection from a PV processor model to show current instruction count.
`lcd`	LCD	slave	Connection from a CLCD controller for visualization of the frame buffer.
`leds`	ValueState	slave	Displays state using the eight colored LEDs on the status bar.
`user_switches`	ValueState	slave	Provides state for the eight User DIP switches on the left side of the CLCD status bar, equivalent to switch S6 on EB hardware.
`boot_switch`	ValueState	slave	Provides state for the eight Boot DIP switches on the right side of the CLCD status bar, equivalent to switch S8 on EB hardware.
`keyboard`	KeyboardStatus	master	Output port providing key change events when the visualization window is in focus.
`mouse`	MouseStatus	master	Output port providing mouse movement and button events when the visualization window is in focus.
`clock_50Hz`	ClockSignal	slave	50Hz clock input.
`touch_screen`	MouseStatus	master	Provides mouse events when the visualization window is in focus.

Table 7-6 EBVisualisation component ports (continued)

Name	Port Protocol	Type	Description
`lcd_layout`	LCDLayoutInfo	master	Layout information for alphanumeric LCD display.
`daughter_leds`	ValueState	slave	A read/write port to read and set the value of the LEDs. 1 bit per LED, LSB left-most, up to 32 LEDs available. The LEDs appear only when parameter `daughter_led_count` is set to nonzero.
`daughter_user_switches`	ValueState	slave	A read port to return the value of the daughter user switches. Write to this port to set the value of the switches, and use during reset only. LSB is left-most, up to 32 switches available.

7.4.2 Additional protocols

The EBVisualisation component has three additional protocols, all of which are described in a separate document. See Chapter 5 *Peripheral and Interface Components*.

The three protocols are:
- KeyboardStatus
- MouseStatus
- LCD.

7.4.3 Parameters

Table 7-7 provides a description of the configuration parameters for the EBVisualisation component.

Table 7-7 EBVisualisation configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`cpu_name`	Processor name displayed in window title	String		
`daughter_led_count`	Set to nonzero to display up to 32 LEDs. See `daughter_leds` in Table 7-6 on page 7-10.	Integer	`0-32`	`0`
`daughter_user_switch_count`	Set this parameter to display up to 32 switches. See `daughter_user_switches` in Table 7-6 on page 7-10.	Integer	`0-32`	`0`
`disable_visualisation`	Disable the EBVisualisation component on model startup	Boolean	`true/false`	`false`

Table 7-7 EBVisualisation configuration parameters (continued)

Parameter name	Description	Type	Allowed value	Default value
`rate_limit-enable`	Restrict simulation speed so that simulation time more closely matches real time rather than running as fast as possible	Boolean	`true/false`	`true`
`trap_key`	Trap key that works with left Ctrl key to toggle mouse display	Integer	valid ATKeyCode key value[a]	`74`[b]
`window_title`	Window title (`%cpu%` is replaced by `cpu_name`)	String		Fast Models - CLCD `%cpu%`

a. See the header file, `%PVLIB_HOME%\components\KeyCode.h`, for a list of ATKeyCode values. On Linux, use `$PVLIB_HOME/components/KeyCode.h`.

b. This is equivalent to the left **Alt** key, so pressing Left Alt and Left Ctrl simultaneously toggles the mouse display.

Note

Setting the `rate_limit-enable` parameter to `true` (the default) prevents the simulation from running too fast on fast workstations and enables timing loops and mouse actions to work correctly. The overall simulation speed, however, is reduced.

If your priority is high simulation speed, set `rate_limit-enable` to `false`.

7.4.4 Registers

The EBVisualisation component has no registers.

7.4.5 Debug Features

The EBVisualisation component has no debug features.

7.4.6 Verification and testing

The EBVisualisation component has been tested by use as an I/O device for booting Linux and other operating systems.

7.4.7 Performance

Use of elements in the status bar are not expected to significantly affect the performance of a PV system. However, applications that make heavy use of re-drawing the contents of the frame buffer incur overhead through GUI interactions on the host OS.

7.4.8 Library dependencies

The MPSVisualisation component relies on the *Simple DirectMedia Layer* (SDL) libraries, specifically `libSDL-1.2.so.0.11.2`. This library is bundled with the Model Library and is also available as a rpm for Red Hat Enterprise Linux. For more information, see the SDL, `http://www.libsdl.org` web site.

7.5 EB_SysRegs component

The EB system registers component is a programmer's view model of the EB baseboard status and system control registers. For a detailed description of the behavior of the component, see the hardware documentation. See the *RealView Emulation Baseboard User Guide (Lead Free)*.

Figure 7-4 shows a view of the component in System Canvas.

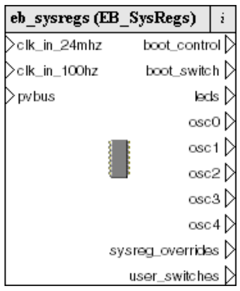


Figure 7-4 EB_SysRegs in System Canvas

This component is written in LISA+.

7.5.1 Ports

Table 7-8 provides a brief description of the EB_SysRegs component ports. For more information, see the hardware documentation.

Table 7-8 EB_SysRegs ports

Name	Port Protocol	Type	Description
`pvbus`	PVBus	slave	slave port for connection to PV bus master/decoder
`clk_in_24mhz`	ClockSignal	slave	reference clock for internal counter register
`clk_in_100hz`	ClockSignal	slave	reference clock for internal counter register
`osc0`	ICS307Configuration	master	settings for the ICS307 OSC0
`osc1`	ICS307Configuration	master	settings for the ICS307 OSC1
`osc2`	ICS307Configuration	master	settings for the ICS307 OSC2
`osc3`	ICS307Configuration	master	settings for the ICS307 OSC4
`osc4`	ICS307Configuration	master	settings for the ICS307 OSC5
`boot_control`	ValueState	master	passes the value of the boot switch to the EBRemapper component[a]

Table 7-8 EB_SysRegs ports (continued)

Name	Port Protocol	Type	Description
`boot_switch`	ValueState	master	provides state for the eight Boot DIP switches on the right side of the CLCD status bar, equivalent to switch S8 on EB hardware
`leds`	ValueState	master	displays state of the SYS_LED register using the eight colored LEDs on the status bar
`user_switches`	ValueState	master	provides state for the eight User DIP switches on the left side of the CLCD status bar, equivalent to switch S6 on EB hardware.

a. See *EBRemapper component* on page 7-22.

7.5.2 Additional protocols

The EB_SysRegs component has no additional protocols.

7.5.3 Parameters

Table 7-9 provides a description of the configuration parameters for the EB_SysRegs component.

Table 7-9 EB_SysRegs configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`boot_switches_Value`	boot select switch	Integer	as switch S8 on EB hardware[a]	`0`
`clcdid`	color LCD ID	Integer	-	`0x1`
`proc_id_tile_site1`	processor ID register	Integer	-	`0x04000000`
`proc_id_tile_site2`	processor ID register	Integer	-	`0x04000000`
`sys_id`	system identification register	Integer	-	`0x01400400`
`tile1`	tile 1 fitted	Boolean	`true/false`	`true`
`tile2`	tile 2 fitted	Boolean	`true/false`	`false`
`user_switches_value`	user switches	Integer	as switch S6 on EB hardware[a]	`0`

a. Full details on switch values and their effects are provided elsewhere. See the *RealView Development Suite Real-Time System Models User Guide*. See also the *RealView Emulation Baseboard User Guide (Lead Free)*.

7.5.4 Registers

Table 7-10 provides a description of the configuration registers for the EB_SysRegs component.

Table 7-10 EB_SysRegs registers

Register name	Offset	Access	Description
SYS_ID	0x00	read only	system identity
SYS_SW	0x04	read only	bits[7:0] map to switch S6
SYS_LED	0x08	read/write	bits[7:0] map to user LEDs
SYS_OSC0	0x0C	read/write	settings for the ICS307 OSC0
SYS_OSC1	0x10	read/write	settings for the ICS307 OSC1
SYS_OSC2	0x14	read/write	settings for the ICS307 OSC2
SYS_OSC3	0x18	read/write	settings for the ICS307 OSC3
SYS_OSC4	0x1C	read/write	settings for the ICS307 OSC4
SYS_LOCK	0x20	read/write	write 0xA05F to unlock
SYS_100HZ	0x24	read only	100Hz counter
SYS_CONFIGDATA1	0x28	-	reserved for configuring clock domain
SYS_CONFIGDATA2	0x2C	-	reserved for configuring clock domain
SYS_FLAGS	0x30	read/write	general purpose flags
SYS_FLAGSCLR	0x34	write only	clear bits in general purpose flags
SYS_NVFLAGS	0x38	read/write	general purpose non-volatile flags
SYS_NVFLAGSCLR	0x3C	write only	clear bits in general purpose non-volatile flags
SYS_PCICTL	0x44	read/write	PCI control
SYS_MCI	0x48	read only	MCI
SYS_FLASH	0x4C	read/write	flash control
SYS_CLCD	0x50	read/write	controls LCD power and multiplexing
SYS_CLCDSER	0x54	read/write	LCD control
SYS_BOOTCS	0x58	read only	boot select switch
SYS_24MHZ	0x5C	read only	24MHz counter
SYS_MISC	0x60	read only	miscellaneous control flags

Table 7-10 EB_SysRegs registers (continued)

Register name	Offset	Access	Description
SYS_DMAPSR0	`0x64`	read/write	DMA peripheral map
SYS_DMAPSR1	`0x68`	read/write	DMA peripheral map
SYS_DMAPSR2	`0x6C`	read/write	DMA peripheral map
SYS_IOSEL	`0x70`	read/write	peripheral I/O select
SYS_PLDCTL1	`0x74`	read/write	configure the attached CoreTiles
SYS_PLDCTL2	`0x78`	read/write	configure the attached CoreTiles
SYS_BUSID	`0x80`	read only	bus ID
SYS_PROCID0	`0x84`	read only	processor ID
SYS_PROCID1	`0x88`	read only	processor ID
SYS_OSCRESET0	`0x8C`	read/write	oscillator reset
SYS_OSCRESET1	`0x90`	read/write	oscillator reset
SYS_OSCRESET2	`0x94`	read/write	oscillator reset
SYS_OSCRESET3	`0x98`	read/write	oscillator reset
SYS_OSCRESET4	`0x9C`	read/write	oscillator reset
SYS_VOLTAGE0	`0xA0`	read/write	CoreTile configuration
SYS_VOLTAGE1	`0xA4`	read/write	CoreTile configuration
SYS_VOLTAGE2	`0xA8`	read/write	CoreTile configuration
SYS_VOLTAGE3	`0xAC`	read/write	CoreTile configuration
SYS_VOLTAGE4	`0xB0`	read/write	CoreTile configuration
SYS_VOLTAGE5	`0xB4`	read/write	CoreTile configuration
SYS_VOLTAGE6	`0xB8`	read/write	CoreTile configuration
SYS_VOLTAGE7	`0xBC`	read/write	CoreTile configuration
SYS_TEST_OSC0	`0xC0`	read only	oscillator test
SYS_TEST_OSC1	`0xC4`	read only	oscillator test
SYS_TEST_OSC2	`0xC8`	read only	oscillator test
SYS_TEST_OSC3	`0xCC`	read only	oscillator test
SYS_TEST_OSC4	`0xD0`	read only	oscillator test
SYS_T1_PLD_DATA	`0xE0`	read/write	PLD configuration
SYS_T2_PLD_DATA	`0xE4`	read/write	PLD configuration
SYS_GPIO	`0xE8`	read/write	GPIO

7.5.5 Debug Features

The EB_SysRegs component has no debug features.

7.5.6 Verification and testing

The EB_SysRegs component has been tested as part of the Emulation Board model.

7.5.7 Performance

The EB_SysRegs component is not expected to significantly affect the performance of a PV system.

7.5.8 Library dependencies

The EB_SysRegs component has no dependencies on external libraries.

7.6 EBCortexA9_SysRegs component

The EB Cortex-A9 system registers component is a programmer's view model of the Cortex-A9 specific status and system control registers. It overrides some of the system registers provided by the EB_SysRegs component. For a detailed description of the behavior of the component, see the hardware documentation. See the *RealView Emulation Baseboard User Guide (Lead Free)*.

Figure 7-4 on page 7-13 shows a view of the component in System Canvas.

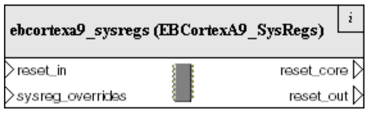


Figure 7-5 EBCortexA9_SysRegs in System Canvas

This component is written in LISA+.

7.6.1 Ports

Table 7-8 on page 7-13 provides a brief description of the EBCortexA9_SysRegs component ports. For more information, see the hardware documentation.

Table 7-11 EBCortexA9_SysRegs ports

Name	Port Protocol	Type	Description
reset_core[4]	Signal	master	signals to reset separate processors
reset_out	Signal	master	reset signal
sysreg_overrides	EBSysRegs	slave	port to communicate with EB_SysRegs
reset_in	Signal	slave	reset signal

7.6.2 Additional protocols

The EBCortexA9_SysRegs component has one additional protocol.

The EBSysRegs protocol provides the rules to communicate with the EB_SysRegs component to override some of the system registers provided by the EB_SysRegs component.

7.6.3 Parameters

The EBCortexA9_SysRegs component has no parameters.

7.6.4 Registers

Table 7-10 on page 7-15 provides a description of the configuration registers for the EBCortexA9_SysRegs component.

Table 7-12 EBCortexA9_SysRegs registers

Register name	Offset	Access	Description
SYS_RESETCTL	`0x40`	read/write	Controls the software reset level
SYS_PLDCTL1	`0x74`	read only	Configure the attached CoreTiles
SYS_PLDCTL2	`0x78`	read/write	Configure the attached CoreTiles

7.6.5 Debug Features

The EBCortexA9_SysRegs component has no debug features.

7.6.6 Verification and testing

The EBCortexA9_SysRegs component has been tested as part of the Emulation Board model.

7.6.7 Performance

The EBCortexA9_SysRegs component is not expected to significantly affect the performance of a PV system.

7.6.8 Library dependencies

The EBCortexA9_SysRegs component has no dependencies on external libraries.

7.7 TSC2200 component

The TSC2200 component is a programmer's view model of the EB baseboard touchscreen interface. This component is based on a TSC2200 PDA analog interface circuit, which is provided as standard on the EB baseboard. For a detailed description of the behavior of the component, see the hardware documentation. See the *RealView Emulation Baseboard User Guide (Lead Free)*.

Figure 7-6 shows a view of the component in System Canvas.

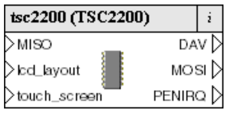


Figure 7-6 TSC2200 in System Canvas

This component is written in LISA+.

7.7.1 Ports

Table 7-13 provides a brief description of the TSC2200 component ports. For more information, see the hardware documentation.

Table 7-13 TSC2200 ports

Name	Port Protocol	Type	Description
DAV	Signal	master	Indicates if data available.
lcd_layout	LCDLayoutInfo	slave	Sets the initial horizontal and vertical size of the touchscreen.
MISO	Value	slave	Serial data output to PL022 *Synchronous Serial Port* (SSP) device.
MOSI	Value	master	Serial data input from PL022 SSP device.
PENIRQ	Signal	master	Sends IRQ signal to the host if the pen is down.
touch_screen	MouseStatus	slave	Provides pointer position and pen status.

7.7.2 Additional protocols

The TSC2200 component has one additional protocol.

LCDLayoutInfo

The LCDLayoutInfo protocol has the following behavior:

`setLayoutInfo` determines the width and height of the touchscreen.

7.7.3 Parameters

Table 7-14 provides a description of the configuration parameters for the TSC2200 component. The parameters are hardware specific, so might be different depending on the LCD hardware display being considered.

Table 7-14 TSC2200 configuration parameters

Parameter name	Description	Type	Allowed value	Default value
touch_max_x	Maximum x value that can be returned by the touchscreen.	Integer	-	3900
touch_min_x	Minimum x value that can be returned by the touchscreen.	Integer	-	250
touch_max_y	Maximum y value that can be returned by the touchscreen.	Integer	-	3550
touch_min_y	Minimum y value that could be returned by the touchscreen.	Integer	-	270

7.7.4 Registers

The TSC2200 component has no registers.

7.7.5 Debug Features

The TSC2200 component has no debug features.

7.7.6 Verification and testing

The TSC2200 component has been tested as part of the Emulation Board model.

7.7.7 Performance

The TSC2200 component is not expected to significantly affect the performance of a PV system.

7.7.8 Library dependencies

The TSC2200 component has no dependencies on external libraries.

7.7.9 Functionality

The TSC2200 component implements functional required to support a touch screen. It currently does not provide support for:

- 4x4 keypad
- A/D & D/A converters
- Temperature measurement
- Battery measurement.

7.8 Other Emulation Baseboard components

This section briefly describes the purpose of selected additional components included with the Emulation Baseboard platform model.

7.8.1 EBConnector and EBSocket components

The EBConnector and EBSocket components are used to forward Interrupts, CLK and PVBus transactions between the EBBaseboard and the CoreTile. As their names indicate, the two components emulate the connections in a CoreTile to the Emulation Baseboard.

7.8.2 EBInterruptForwarder component

The EBInterruptForwarder component is an assistant component for EBSocket. EBInterruptForwarder receives all interrupts and is responsible for telling the socket which interrupt happened.

7.8.3 EBRemapper component

The EBRemapper component is a control component involved with remapping logic on the Emulation Baseboard. The component receives the **remap_clear** signal from the SP810_SysCtrl component and boot_switch value from the EB_SysReg component to determine which devices are to be remapped and unremapped. The EB_SysReg component is described elsewhere in this document. See *EB_SysRegs component* on page 7-13. The SP810_SysCtrl component is described in Chapter 5 *Peripheral and Interface Components*.

7.9 Differences between the EB hardware and the system model

The following sections describe features of the Emulation Baseboard hardware that are not implemented in the models or have significant differences in implementation:

- *Features not present in the model*
- *Remapping and DRAM aliasing* on page 7-24
- *Dynamic memory characteristics* on page 7-24
- *Status and system control registers* on page 7-24
- *Generic Interrupt Controller* on page 7-24
- *GPIO2* on page 7-25
- *Timing considerations* on page 7-25.

7.9.1 Features not present in the model

The following features present on the hardware version of the Emulation Baseboard are not implemented in the system models:

- two wire serial bus interface
- character LCD interface
- smart card interface
- PCI controller configuration registers
- debug access port
- disk on chip
- configuration flash
- USB
- PISMO expansion memory
- PCI interface bus windows
- UART Modem handshake signals
- VGA support.

Note

For more information on memory-mapped peripherals, see *Emulation Baseboard memory map* on page 7-3.

The following features present on the hardware version of the Emulation Baseboard are only partially implemented in the Fixed Virtual Platforms:

- *Sound*
- *Dynamic memory controller*.

Partial implementation means that some of the components are present but the 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 EB FVPs implement the PL041 AACI PrimeCell and the audio CODEC as in the EB hardware, but with a limited number of sample rates.

Dynamic memory controller

The dynamic memory controller, though modeled in the EB FVPs, does not provide direct memory access to all peripherals. Only the audio and synchronous serial port interface components can be accessed through the DMC.

7.9.2 Remapping and DRAM aliasing

The EB hardware provides considerable memory remap functionality. During this boot remapping, the bottom 64MB of the physical address map can be:

- NOR flash
- Static expansion memory.

The hardware aliases all 256MB of system DRAM at `0x70000000`, and provides remap functionality.

Remapping does not typically apply to the system models. However, NOR flash is modeled and can be remapped. See *Switch S8* on page 7-7.

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, and those areas that are documented as reserved. Accessing these regions from the host processor results in the model presenting a warning.

7.9.3 Dynamic memory characteristics

The Emulation Baseboard hardware contains a PL340 DMC. This presents a configuration interface at address `0x10030000` in the memory map.

The system models configure a generic area of DRAM and does not model the PL340. This simplification helps speed the simulation.

7.9.4 Status and system control registers

For the hardware version of the Emulation Baseboard, the status and system control registers enable the processor to determine its environment and to control some on-board operations.

Note

Most of the EB FVP functionality is determined by its configuration on startup.

All EB system registers have been implemented in the system model, except for SYS_TEST_OSC[4:0], the oscillator test registers. Registers that are not implemented function as memory and the values written to them do not alter the behavior of the model.

In addition to the standard EB system control registers, an additional register has been modeled, based on the Platform Baseboard design. This is the SYS_RESETCTL register. For further information on this register see *RealView Platform Baseboard Explore for Cortex-A9 User Guide*.

7.9.5 Generic Interrupt Controller

The *Generic Interrupt Controller* (GIC) provided with the EB FVPs differs substantially from that in the current Emulation Board firmware. The programmer's model of the newer device is largely backwards compatible. The model GIC is an implementation of the PL390 PrimeCell, for which comprehensive documentation is provided elsewhere. See *PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual*.

7.9.6 GPIO2

On the EB hardware, GPIO2 is dedicated to USB, a push button, and MCI status signals. USB and MCI are not implemented in the EB FVPs, and no push button is modeled. The GPIO is therefore provided as another generic IO device.

7.9.7 Timing considerations

The Fixed Virtual Platforms provides an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

If your code interacts with real world devices like 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 keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

To work around this problem, the EB FVPs supply the Rate Limit feature. Enabling Rate Limit, either 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.

Chapter 8
Microcontroller Prototyping System: Platform and Components

This chapter introduces the *Microcontroller Prototyping System* (MPS) components provided with Fast Models. It contains the following sections:

8.1 About the Microcontroller Prototyping System components

The MPS components have been specifically developed to model in software some of the functionality of the Microcontroller Prototyping System hardware.

A complete model implementation of the MPS platform includes both MPS-specific components and generic ones such as buses and timers. This section describes components used only in the *Microcontroller Prototyping System Model*. Generic components are documented in other chapters of this book.

The MPS FVP uses platform-specific components that are located in the `%PVLIB_HOME%\examples\FVP_MPS\LISA` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_MPS/LISA` directory instead.

8.2 MPS model memory map

This section describes the MPS memory map. For standard ARM peripherals, see the TRM for that device.

Table 8-1 Overview of MPS memory map

Description	Modeled	Address range
4MB Remap region for SRAM0 overlay of Flash	Yes	0x00000000–0x003FFFFF
Non remapped Flash memory	Yes	0x00400000–0x03FFFFFF
SRAM for code and data storage (remap RAM)	Yes	0x10000000–0x103FFFFF
SRAM for code and data storage	Yes	0x10400000–0x107FFFFF
FLASH aliased for programming	Yes	0x18000000–0x1BFFFFFF
Processor system registers	Yes	0x1F000000–0x1F000FFF
Reserved for SMC configuration registers	N/A	0x1F001000–0x1F002FFF
I2C for DVI	Yes	0x1F003000–0x1F003FFF
PL022 SPI for Touch Screen	Yes	0x1F004000–0x1F004FFF
PL011 UART	Yes	0x1F005000–0x1F005FFF
Reserved	N/A	0x1F006000–0x1FFFFFFF
SP805 Watch Dog	Yes	0x40000000–0x4000FFFF
PL031 RTC	Yes	0x40001000–0x40001FFF
SP804 Timer (0)	Yes	0x40002000–0x40002FFF
SP804 Timer (1)	Yes	0x40003000–0x40003FFF
DUT system registers	Yes	0x40004000–0x40004FFF
PL181 SD/MMC controller	Yes	0x40005000–0x40005FFF
Reserved	N/A	0x40006000–0x40006FFF
PL011 UART (1)	Yes	0x40007000–0x40007FFF
PL011 UART (2)	Yes	0x40008000–0x40008FFF
PL011 UART (3)	Yes	0x40009000–0x40009FFF
PL041 AC97 controller	Yes	0x4000A000–0x4000AFFF
DS702 I2C (ADCDAC)	Partial[a]	0x4000B000–0x4000BFFF
DUT Character LCD	Yes	0x4000C000–0x4000CFFF
Reserved	N/A	0x4000D000–0x4000EFFF
Reserved	N/A	0x4FFA0000–0x4FFAFFFF
Flexray	Partial[a]	0x4FFB0000–0x4FFBFFFF
CAN	Partial[a]	0x4FFC0000–0x4FFCFFFF

Table 8-1 Overview of MPS memory map (continued)

Description	Modeled	Address range
LIN	Partial[a]	`0x4FFD0000–0x4FFDFFFF`
Ethernet	Partial[a]	`0x4FFE0000–0x4FFEFFFF`
Video	Yes	`0x4FFF0000–0x4FFFFFFF`
External AHB interface to DUT FPGA	Yes	`0x50000000–0x5FFFFFFF`
DMC	Yes	`0x60000000–0x9FFFFFFF`
SMC	Yes	`0xA0000000–0xAFFFFFFF`
Private Peripheral Bus	Yes	`0xE0000000–0xE00FFFFF`
System bus interface to DUT FPGA	Yes	`0xE0100000–0xFFFFFFFF`

a. This model is represented by a register bank and has no functionality beyond this.

Note

A Bus Error is generated for accesses to memory areas not shown in Table 8-1 on page 8-3.

Any memory device that does not occupy the total region is aliased within that region.

8.2.1 MPS registers

This section describes the MPS memory-mapped registers.

Processor system registers

Table 8-2 provides a description of the processor system registers.

Table 8-2 MPS processor system registers

Register name	Address	Access	Description
SYS_ID	`0x1F000000`	read/write	Board and FPGA identifier
SYS_MEMCFG	`0x1F000004`	read/write	Memory remap and alias
SYS_SW	`0x1F000008`	read/write	Indicates user switch settings
SYS_LED	`0x1F00000C`	read/write	Sets LED outputs
SYS_TS	`0x1F000010`	read/write	TouchScreen register

DUT system registers

Table 8-15 on page 8-14 provides a description of the DUT system registers.

Table 8-3 MPS DUT system registers

Register name	Address	Access	Description
SYS_ID	`0x40004000`	read/write	Board and FPGA identifier
SYS_PERCFG	`0x40004004`	read/write	Peripheral control signals
SYS_SW	`0x40004008`	read/write	Indicates user switch settings
SYS_LED	`0x4000400C`	read/write	Sets LED outputs
SYS_7SEG	`0x40004010`	read/write	Sets seven-segment LED outputs
SYS_CNT25MHZ	`0x40004014`	read/write	Free running counter incrementing at 25MHz
SYS_CNT100HZ	`0x40004018`	read/write	Free running counter incrementing at 100Hz

Character LCD registers

Table 8-12 on page 8-12 provides a description of the character LCD registers.

Table 8-4 MPS LCD registers

Register name	Address	Access	Description
CHAR_COM	`0x4000C000`	write	Command register. The command set is compatible with the commands of the Hitachi HD44780U controller.
CHAR_DAT	`0x4000C004`	write	Write data register.
CHAR_RD	`0x4000C008`	read	Read data register.
CHAR_RAW	`0x4000C00C`	read/write	Raw interrupt.
CHAR_MASK	`0x4000C010`	read/write	Interrupt mask.
CHAR_STAT	`0x4000C014`	read/write	Masked interrupt.

Memory configuration and remap

Table 8-5 provides a description of the memory configuration register.

Table 8-5 Memory configuration

Name	Bits	Access	Power On Reset	Description
Reserved	31:3	-	-	-
SWDPEN	2	RW	0b	Single Wire Debug Port Enable. 1 is SWD 0 JTAG
ALIAS	1	RW	1b	Alias FLASH. 1 is Aliased on. 0 is Aliased off
REMAP	0	RW	0b	Remap SSRAM. 1 is Remap on. 0 is Remap off

The ability to remap the Static memory into the bottom of memory (overlaying the Flash) is required for booting and code execution to permit the interrupt vector table to be modified. It is also used to permit boot code execution from SRAM for code development, rather than programming the FLASH each time.

The aliasing of the Flash memory into SRAM space is required to permit the Flash memory to be reprogrammed at this offset. It also permits full flash memory access when the Remap is enabled, otherwise only the Flash memory above 4MB would be accessible.

Switches

Table 8-6 lists the bits for the user switch inputs.

Table 8-6 User switches

Name	Bits	Access	Reset	Note
Reserved	31:8	-	-	-
USER_BUT[3:0]	7:4	RO	-	Always returns value of user buttons
USER_SW[3:0]	3:0	RO	-	Always returns value of user switches

Seven-segment display

Table 8-7 lists the bits that control the seven-segment display.

Table 8-7 Seven-segment register

Name	Bits	Access	Reset	Note
DISP3	31:24	RW	`0x00`	Segments for display 3
DISP2	23:16	RW	`0x00`	Segments for display 2
DISP1	15:8	RW	`0x00`	Segments for display 1
DISP0	7:0	RW	`0x00`	Segments for display 0

8.3 MPSVisualisation

The MPSVisualisation component provides:

- a host window to display status
- a frame buffer
- MPS front panel and internal switches.

The MPSVisualisation component can generate events when the touchscreen is used.

At startup, if connected to a suitable platform model, the visualization presents a window with a status bar containing many of the visual components of the platform.

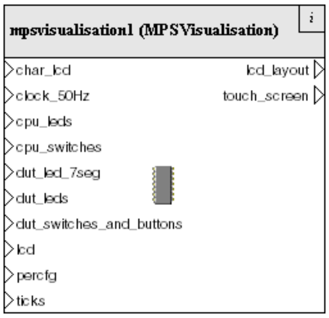


Figure 8-1 MPSVirtualisation component in sgcanvas

The Color LCD (CLCD) screen is divided into three sections:

- A representation of the Microcontroller Prototyping System front panel. This includes the character LCD, LEDs and push buttons present on the physical MPS platform. The buttons on the front panel of the MPSVisualisation component can be controlled by clicking on them with the mouse.
- A combined panel presenting the character LED display and dip-switches present inside the Microcontroller Prototyping System, control of the SD Card interface and the normal model visualization status and rate-limit control.
- A CLCD connected to the Microcontroller Prototyping System. This is currently not provided in the initial release of the Microcontroller Prototyping System hardware.

Table 8-8 MPSVisualisation interactive controls

Button	Description
SDCard	Inserts or removes the SD card.
DIP switches	Allow the state of the dip switches to be set in a similar manner to the hardware platform.
SDcard write-protect slider	Enables/disables read-only access to the SD card model.
statistics	Change the statistics display mode.
rate limit	Enable/disable rate limiter.
reset button	Non-functional. To reset a model the simulation must be restarted or reset by an attached debugger.

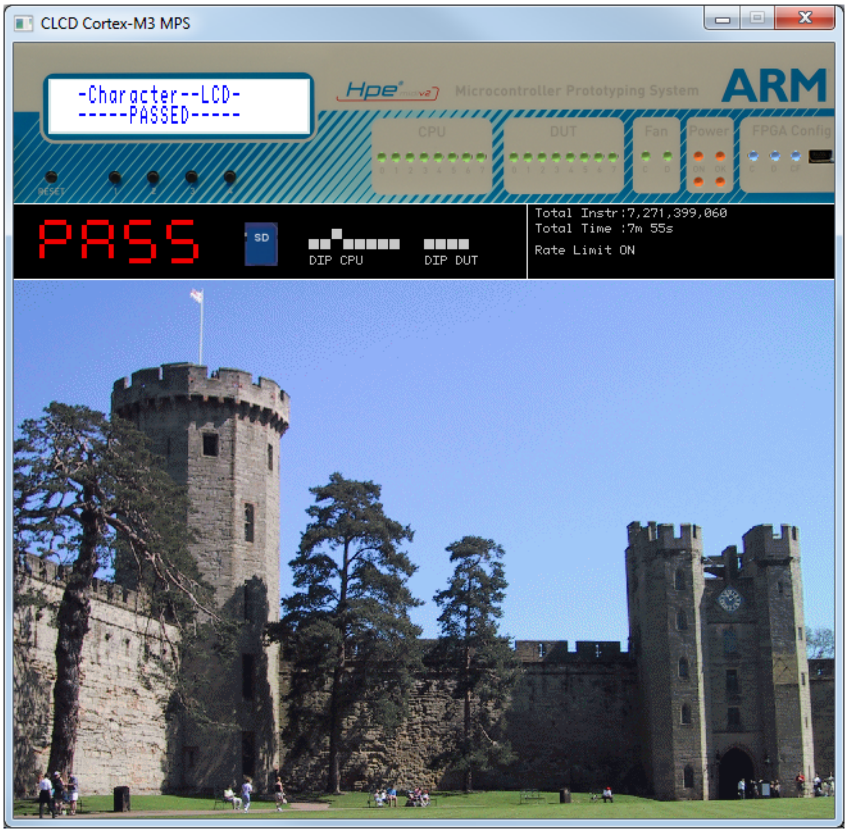


Figure 8-2 Microprocessor Prototyping System Fixed Virtual Platform CLCD

This component is written in LISA+.

8.3.1 Ports

Table 8-9 gives a brief description of the MPSVisualisation component ports. For more information, see the Microcontroller Prototyping System platform documentation. See the *Microcontroller Prototyping Board User Guide*.

Table 8-9 MPSVisualisation component ports

Name	Port protocol	Type	Description
`ticks`	InstructionCount	slave	Connection from a PV processor model to show current instruction count.
`clock_50Hz`	ClockSignal	slave	50Hz clock input
`touch_screen`	MouseStatus	master	Provides mouse events when the visualization window is in focus.
`lcd_layout`	LCDLayoutInfo	master	Layout information for alphanumeric LCD display.
`cpu_leds`	ValueState	slave	Displays state using the eight colored LEDs on the status bar. These LEDs are driven by the LED_LIGHTS register as on the MPS hardware.

Table 8-9 MPSVisualisation component ports (continued)

Name	Port protocol	Type	Description
`cpu_switches`	ValueState	slave	Provides state for the four DIP switches on the status bar.
`lcd`	LCD	slave	Displays state using LCD display.
`dut_switches_and_buttons`	ValueState	slave	Sets and displays state for four DIP switches and four buttons.
`dut_leds`	ValueState	slave	Displays state using the eight colored LEDs on the status bar.
`dut_led_7seg`	ValueState	slave	Displays state using four digit seven segment display.
`char_lcd`	CharacterLCD	slave	Displays state using the alphanumeric LCD display.
`percfg`	ValueState	slave	Sets and displays status of SD card.

8.3.2 Additional protocols

The MPSVisualisation component has four additional protocols, all of which are described in Chapter 5 *Peripheral and Interface Components*.

The four protocols are:
- LCDLayoutInfo
- MouseStatus
- LCD
- CharacterLCD

8.3.3 Parameters

Table 8-10 provides a description of the configuration parameters for the MPSVisualisation component.

Table 8-10 MPSVisualisation configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`trap_key`	trap key that works with left Ctrl key to toggle mouse pointer display	Integer	valid ATKeyCode key value[a]	`107`[b]
`rate_limit-enable`	rate limit simulation	Boolean	true/false	`true`
`disable-visualisation`	enable/disable visualization	Boolean	true/false	`false`

a. See the header file, `%PVLIB_HOME%\components\KeyCode.h`, for a list of ATKeyCode values. On Linux use `$PVLIB_HOME/components/KeyCode.h`.

b. This is equivalent to the left **Windows** key.

8.3.4 Registers

The MPSVisualisation component has no registers.

8.3.5 Debug features

The MPSVisualisation component has no debug features.

8.3.6 Verification and testing

The MPSVisualisation component has been tested as part of the MPS example system using MPS test suites.

8.3.7 Performance

Applications that make heavy use of re-drawing the contents of the frame buffer might incur overhead through GUI interactions on the host OS.

8.3.8 Library dependencies

The MPSVisualisation component relies on the *Simple DirectMedia Layer* (SDL) libraries, specifically `libSDL-1.2.so.0.11.2`. This library is bundled with the Model Library and is also available as a rpm for Red Hat Enterprise Linux. For more information, see the SDL, `http://www.libsdl.org` web site.

8.4 MPS_CharacterLCD

The MPS_CharacterLCD component provides a 2x80 character display window.

Figure 8-3 shows the MPS_CharacterLCD component in System Canvas.

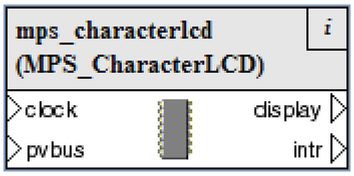


Figure 8-3 MPSVirtualisation component in sgcanvas

This component is written in LISA+.

8.4.1 Ports

Table 8-11 gives a brief description of the MPS_CharacterLCD component ports. For more information, see the Microcontroller Prototyping System platform documentation. See the *Microcontroller Prototyping Board User Guide*.

Table 8-11 MPS_CharacterLCD component ports

Name	Port protocol	Type	Description
`display`	CharacterLCD	master	Provides content of alphanumeric display.
`intr`	Signal	master	Interrupt signal driven for display update.
`clock`	ClockSignal	slave	Reference clock.
`pvbus`	PVBus	slave	Slave port for connection to PV bus master/decoder.

8.4.2 Additional protocols

The MPS_CharacterLCD component has one additional protocol, CharacterLCD.

8.4.3 Parameters

The MPS_CharacterLCD component has no parameters.

8.4.4 Registers

Table 8-12 provides a description of the configuration parameters for the MPS_CharacterLCD component.

Table 8-12 MPS_CharacterLCD registers

Register name	Offset	Access	Description
CHAR_COM	`0x00`	write	Command register. The command set is compatible to the commands of the Hitachi HD44780U controller.
CHAR_DAT	`0x04`	write	Write data register.
CHAR_RD	`0x08`	read	Read data register.
CHAR_RAW	`0x0C`	read/write	Raw interrupt.
CHAR_MASK	`0x10`	read/write	Interrupt mask.
CHAR_STAT	`0x14`	read/write	Masked interrupt.

8.4.5 Debug features

The MPS_CharacterLCD component has no debug features.

8.4.6 Verification and testing

The MPS_CharacterLCD component has been tested as part of the MPS example system using MPS test suites and by booting ucLinux.

8.4.7 Performance

Applications that make heavy use of re-drawing the contents of the character LCD might incur an overhead through GUI interactions on the host OS.

8.4.8 Library dependencies

The MPS_CharacterLCD component relies on the *Simple DirectMedia Layer* (SDL) libraries, specifically `libSDL-1.2.so.0.11.2`. This library is bundled with the Model Library and is also available as a rpm for Red Hat Enterprise Linux. For more information, see the SDL, `http://www.libsdl.org` web site.

8.5 MPS_DUTSysReg

The MPS_DUTSysReg component provides the MPS DUT system registers.

Figure 8-4 shows a view of the component with the input ports fully expanded, in System Canvas.

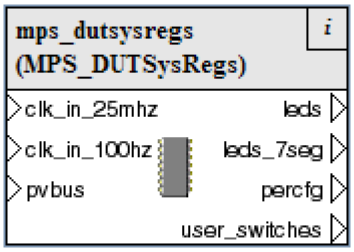


Figure 8-4 MPS_DUTSysReg component in sgcanvas

This component is written in LISA+.

8.5.1 Ports

Table 8-13 provides a brief description of the ports of the MPS_DUTSysReg component. For more information, see the Microcontroller Prototyping System hardware documentation.

Table 8-13 MPS_DUTSysReg ports

Name	Port protocol	Type	Description
`pvbus`	PVBus	slave	For connection to PV bus master/decoder.
`clk_in_25mhz`	ClockSignal	slave	Reference clock for internal counter register.
`clk_in_100hz`	ClockSignal	slave	Reference clock for internal counter register.
`percfg`	ValueState	master	Status of SD card.
`user_switches`	ValueState	master	Status of user switches.
`leds`	ValueState	master	Status of LEDs.
`leds_7seg`	ValueState	master	Status of seven segment display.

8.5.2 Additional protocols

The MPS_DUTSysReg component has no additional protocols.

8.5.3 Parameters

Table 8-14 provides a description of the configuration parameters for the MPS_DUTSysReg component.

Table 8-14 MPS_DUTSysReg configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`sys_id`	System Identification Register	Integer	`0x0 - 0xffffffff`	`0x01590500`
`percfg_value`	System peripheral configuration	Integer	`0x0 - 0x3`	`0`
`user_switches_value`	User switches	Integer	`0x0 - 0xff`	`0`

8.5.4 Registers

Table 8-15 provides a description of the registers for the MPS_DUTSysReg component.

Table 8-15 MPS_DUTSysReg registers

Register name	Offset	Access	Description
SYS_ID	`0x00`	read/write	Board and FPGA identifier
SYS_PERCFG	`0x04`	read/write	Peripheral control signals
SYS_SW	`0x08`	read/write	Indicates user switch settings
SYS_LED	`0x0C`	read/write	Sets LED outputs
SYS_7SEG	`0x10`	read/write	Sets LED outputs
SYS_CNT25MHZ	`0x14`	read/write	Free running counter incrementing at 25MHz.
SYS_CNT100HZ	`0x18`	read/write	Free running counter incrementing at 100Hz

8.5.5 Debug features

The MPS_DUTSysReg component has no debug features.

8.5.6 Verification and testing

The MPS_DUTSysReg component has been tested as part of the MPS example system using MPS test suites and by booting operating systems.

8.5.7 Performance

The MPS_DUTSysReg component is not expected to significantly affect the performance of a PV system.

8.5.8 Library dependencies

The MPS_DUTSysReg component has no dependencies on external libraries.

8.6 Other MPS virtual platform components

This section briefly describes the purpose of selected additional components included with the Microcontroller Prototyping System platform model.

8.6.1 MPSInterruptForwarder and MPSInterruptReceiver components

The MPSInterruptForwarder and MPSInterruptReceiver components are assistant components for the MPS_DUT and MPSCoreTile components. MPSInterruptForwarder receives all interrupts and is responsible for telling the MPSInterruptReceiver which interrupt happened. The MPSInterruptReceiver drives the corresponding interrupt output signal.

8.7 Differences between the MPS hardware and the system model

The following sections describe features of the Microcontroller Prototyping System hardware that are not implemented in the models or have significant differences in implementation:

- *Features not present in the model*
- *Timing considerations*.

8.7.1 Features not present in the model

The following features present on the hardware version of the Microcontroller Prototyping System are not implemented in the system models:

- I2C interface
- CAN interface
- LIN
- FlexRay.

These components are present in the Microcontroller Prototyping System model as dummy components and take parameters that have no effect on model operation.

The following components are currently not implemented in the model:

- Ethernet

The following features present on the hardware version of the Microcontroller Prototyping System are only partially implemented in the model:

- Sound

Partial implementation means that some of the components are present but the 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 MPS model implements the PL041 AACI PrimeCell and the audio CODEC as in the MPS hardware, but with a limited number of sample rates. AACI Audio Input is not supported.

8.7.2 Timing considerations

The Fixed Virtual Platforms provide you with an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

If your code interacts with real world devices like 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 keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

To work around this problem, the MPS FVP provides the Rate Limit feature. Enabling Rate Limit, either 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.

Appendix A
AEM ARMv7-A specifics

This appendix describes characteristics and capabilities that are specific to the *Architecture Envelope Model* (AEM) ARMv7-A. It contains the following sections:

- *Boundary features and architectural checkers* on page A-2
- *Debug architecture support* on page A-8
- *IMPLEMENTATION DEFINED features* on page A-10
- *Trace* on page A-12.

A.1 Boundary features and architectural checkers

Boundary features and architectural checkers are model capabilities that help your development and testing process by exposing latent problems in the target code. Certain boundary features or architectural checkers, however, might have an adverse effect on the overall running speed of target code. There are two reasons that this occurs:

- some checkers require the host processor to perform extra work, for example to keep track of historical information
- some features cause the simulated target processor to perform extra work, for example in the number of steps required to perform cache maintenance.

As a result, you can optionally enable or disable checkers in cases where there is an impact on performance. In the sections that follow, the effect of both the above factors on the total execution time of a typical target operating system boot is indicated as a relative rating on a scale from 1 to 5, where a larger number indicates a greater impact on speed.

A.1.1 Aggressively pre-fetching TLB

Rating: 2.

The architecture describes that a processor might at any time request the pagetable entry or entries corresponding to any virtual address. When the Boolean parameter `vmsa.tlb_prefetch` is set as `true`, the AEM actively monitors the memory for all pagetables, and immediately updates the *Translation Lookaside Buffer* (TLB) accounting for any changes.

Also, in this mode the TLB maintains state for old and new entries, and raises the message `E_StaleTLB` if there is any possible ambiguity as to which entry could have been used in a subsequent transaction.

Note

The aggressively pre-fetching TLB does not currently support the `IMPLEMENTATION DEFINED` TLB lockdown feature described in *TLB lockdown* on page A-10. Lockdown registers are non-operational when the pre-fetching TLB option is in use.

Note

The aggressively pre-fetching TLB does not currently support the Large Physical Address or Virtualization features. TLB pre-fetching does not occur when either lpae or virtualization is implemented in the processor.

A.1.2 Passive infinite TLB

Rating: 1.

You can safely set the parameters `vmsa.main_tlb_size` and `vmsa.instruction_tlb_size` to arbitrarily large values. Entries are fetched only when they are used, but remain in the TLB until no space remains or the entry is explicitly flushed out.

A.1.3 Infinite write buffer

Rating: 4.

When the parameter `vmsa.infinite_write_buffer` is set, write accesses performed by one processor are not visible to other processors, or outside the processor, unless a relevant barrier operation is performed or until a sufficiently long elapsed delay has occurred.

You can adjust the delay time with the parameter `vmsa.write_buffer_delay`, which is the approximate number of instructions between successive buffer drains. If you set this parameter to 0, then no time-based drains occur, and you must perform explicit barrier operations.

In regions marked as Device or Strongly Ordered memory, accesses complete in an order relative to other accesses of the same type, but can be interleaved with accesses to Normal memory. Writes to these regions are not combinable, and reads cannot be satisfied from the buffer, so an explicit read access invokes the completion of all pending write accesses to that memory type.

You can use this feature most effectively with the cache incoherence check. See *Cache incoherence check*.

A.1.4 Treat cache invalidate operation as clean and invalidate

Rating: 1.

With the parameter `vmsa.cache_treat_invalidate_as_clean` set as `true`, a cache invalidate operation cleans any dirty lines before they are invalidated.

Note

It is never appropriate to rely on incoherence effects of cached memory, because a cached view can be cleaned or evicted at any time.

A.1.5 Cache incoherence check

Rating: 5.

When the parameter `vmsa.cache_incoherence_check` is set, the model warns about any data-side memory read accesses whose result could have been ambiguous because of incoherency in the system. The warning contains the physical address being accessed, and at least two data values that could have been the result.

For example, consider an area of memory that has been copied into the Level 1 cache. If the memory is changed by an external agent, then a read to these addresses could return either the old value from the cache or the new value if that cache line is evicted.

Similar effects can occur in MP systems if newer values are pending in the writebuffer, or in a writeback cache block outside the inner shared domain.

A.1.6 Delayed operation of CP15 instructions

Rating: 2.

In general, the functional effect of any operation in the system control coprocessor, CP15 is not guaranteed to occur until after an *Instruction Synchronization Barrier* (ISB) is subsequently executed, or an exception entry or return occurs. There are several exceptions to this rule. Some guarantee earlier visibility in certain circumstances, and others require extra steps to guarantee that the operation takes place. These are described more fully in the *ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition*.

When parameter `delayed_CP15_operations` is set to `true`, the functional effects of the CP15 operations are postponed for as long as the model determines that it is legal to do so. That is, until the specified barrier is executed, or other ordering requirements force earlier completion.

Table A-1 shows the barrier type to be executed for the functional effect of specific cache and branch prediction operations to be seen in the model.

Table A-1 Delayed CP15 operations

CRn	op1	CRm	op2	Operation	Barrier
c7	0	c1	0	Invalidate all instruction caches to PoU inner shareable	DSB[a]
c7	0	c1	6	Invalidate entire branch predictor array inner shareable	ISB
c7	0	c5	0	Invalidate all instruction caches to PoU	DSB[a]
c7	0	c5	1	Invalidate instruction caches by MVA to PoU	DSB[a]
c7	0	c5	4	(Deprecated encoding) ISB	Implicit
c7	0	c5	6	Invalidate entire branch predictor array	ISB
c7	0	c5	7	Invalidate MVA from branch predictor	ISB
c7	0	c6	1	Invalidate data cache line by MVA to PoC	Implicit
c7	0	c6	2	Invalidate data cache line by set/way	DMB
c7	0	c10	1	Clean data cache line by MVA to PoC	Implicit
c7	0	c10	2	Clean data cache line by set/way	DMB
c7	0	c11	1	Clean data cache line by MVA to PoU	Implicit
c7	0	c14	1	Clean and invalidate data cache line by MVA to PoC	Implicit
c7	0	c14	2	Clean and invalidate data cache line by set/way	DMB

a. In general, an ISB would also be required in target code to guarantee that the following instruction fetch has not been pre-fetched from possibly stale data.

All other CP15 operations not listed in Table A-1 are postponed until an *Instruction Synchronization Barrier* (ISB) is executed or an exception entry or return occurs, subject to ordering requirements.

Where a *Data Memory Barrier* (DMB) is specified as the required barrier, a *Data Synchronization Barrier* (DSB) is a suitable alternative.

For more information on barrier requirements, see the sections relating to TLB maintenance, ordering of cache and branch predictor maintenance operations, and changes to CP15 registers and the memory order model in the *ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition*.

A.1.7 An undefined instruction failed its condition code check

Rating: 1.

If an instruction is conditional, and would cause an undefined instruction abort if the condition codes check passes, then the behavior when the condition code check fails can be `IMPLEMENTATION DEFINED`. A processor can choose to implement such instructions either as a NOP or as causing an undefined exception.

When one of these instructions fails its condition code check, a warning is emitted. Also, by setting option `take_ccfail_undef` to `true`, you can configure the model to take the undefined exception (instead of the usual behavior of treating the `ccfailed` instruction as a NOP).

Currently, the model detects only instructions that are guaranteed to cause an Undefined Instruction exception regardless of processor state. For example, unused encodings in the A32 or T32 instruction sets, missing coprocessors or coprocessor registers, and write accesses to read-only CP15 registers (and vice versa) are all detected. Conversely, instructions that might or might not cause an Undefined Instruction exception, depending on the internal state of the processor, are not detected. For example, the model currently does not detect CP15 instructions that are inaccessible because of privilege or security state. Similarly, the model currently does not detect VFP or NEON instructions that are used when that component has not been properly enabled.

A.1.8 Memory marking check

Rating: 1.

The architecture specifies that behavior can be unpredictable when the same physical memory location has any of the following contradictory combinations of memory marking:

- different memory types (Normal, Device, or Strongly-ordered)
- different shareable attributes (Outer shareable, Inner shareable and non-shareable) in Normal memory or in Device memory where supported
- different cacheability attributes.

Such memory marking contradictions can occur, for example:

- by the use of aliases in a virtual-to-physical address mapping
- by programming the incorrect memory attributes for TLB lookups
- at any time when the same physical address is used by more than one processor inside an inner shared domain.

When parameter `vmsa.memory_marking_check` is set `true`, the model emits a warning when the TLB of a processor, or of any two processors inside the inner shared domain, are loaded with multiple entries for one physical address that fail to obey the restrictions listed.

You can use this feature most effectively with the aggressively pre-fetching TLB. See *Aggressively pre-fetching TLB* on page A-2.

A.1.9 Memory operation reordering

Rating: 3.

The ARMv7 architecture employs what is called a weakly-ordered memory model. This means that, with certain limits, the order of load and store operations performed by the processor is not expected to be the same as the program order.

The model can reorder memory read operations (from LDR, LDM and LDD instructions) with respect to each other, store operations, and certain other instructions. To enable this functionality, set the parameter `vmsa.delayed_read_buffer=1` for the processor, and also set `cpu[n].use_IR=3` for each processor that is to participate in reordering.

When stepping through programs in Model Debugger, a reordered load is evident when the destination register of a load instruction is displayed as `0xAFAFAFAF` (or `0xAFAF` or `0xAF` for sub-word operations). The real value is loaded into the destination register when that value is required for an operation that cannot be deferred any further, for example when it is used to compute the address of a subsequent memory access.

If the value is never needed, for example when another value is written into the destination register before it is ever read, then the deferred operation is killed instead. In such cases, it is likely that the load operation is not visible from outside the processor, and it also does not cause its usual side-effects such as filling cache lines.

A.1.10 Other checks

The checks that are described in this section are always enabled. You can, however, suppress any message from being printed by adjusting the corresponding severity level parameter. See *Message configuration* on page 4-84.

Exclusive access into non-normal memory

`LDREX`, `STREX` and similar instructions are only guaranteed to operate correctly in normal memory. Their operation in other memory types such as Device or Strongly Ordered, is not architecturally defined and must not be relied on. For example, some implementations take an external abort.

BX or BLX to illegal addresses

An address ending `0b10` is not a legal branch target.

Non-normal pagetables should have XN bit set

The ARM architecture does not guarantee that a processor executes instructions from non-normal memory types such as Device or Strongly Ordered. Some implementations can take a pre-fetch abort. It is recommended that the XN bit is set in pagetables for non-normal memory to avoid seeing `IMPLEMENTATION DEFINED` effects in executed code.

Assumptions about cache geometry

Cache maintenance operations can be indexed by set and way. Any program that uses these operations must have read the cache ID registers to determine the number of sets and ways present in the system. Compatibility is not guaranteed if the target code attempts to infer this information from reading the Main ID register.

Overlapping pagetable entries

It is an error if the TLB is permitted to contain more than one physical address mapping for each combination of virtual address, *Application Space Identifier* (ASID) and security state. This might occur if entries in the TLB are not flushed before loading a different set of pagetables describing regions of a different size, or if pagetable entries are not correctly repeated for a Supersection or Large Page entry.

Pagetable properties remapped whilst MMU is enabled

The status of System Control Register bits EE, AFE and TRE affect the way that pagetables are interpreted, but it is `IMPLEMENTATION DEFINED` whether the interpretation takes place when the TLB is loaded or when it is used. Therefore, if these bits are changed while there are active TLB entries, any entries currently in the TLB might not be correctly interpreted.

AP==110 deprecated

Pagetable entries using the encoding `AP[2] = 1`, `AP[1:0] = 0b10` means read-only for both privileged mode and user mode accesses, but its use is deprecated in VMSAv7. Instead use `AP[2] = 1`, `AP[1:0] = 0b11`.

A dirty cache line was invalidated but not cleaned

When the invalidate cache maintenance operation is performed on a writeback cache, the data in any dirty lines might or might not have been flushed to the next cache level or backing RAM. Any subsequent data reads from that address are therefore ambiguous. It is legal to do this, but take care that it is not used improperly or else bugs can be introduced.

Reserved encoding

When extended addressing is in use, each 8-bit field of the `MAIR0` and `MAIR1` registers must be written with a valid memory type. Writing any field with values of the following form is unpredictable:

- `0xyy xxxx` for values of `yy` other than 0
- `xxxx 0xyy` for values of `yy` other than 0
- `yyyy 0000` for values of `yyyy` other than 0
- `0000 1xxx`.

Incoherency between S and NS memory

Some system platforms (such as the VE) do not distinguish between SECURE and NON_SECURE memory accesses, and the storage destination depends only on the physical address regardless of the NS bit. In these cases, using S and NS versions of the same physical address can have unpredictable cache incoherency effects. For example, if the cache contains dirty lines for the same address in each state, the final memory contents depend on random eviction order.

When the AEM, however, is used as a component in a system platform that distinguishes between S and NS accesses, this usage is perfectly legal and recommended, so this warning must be disabled by decreasing the value of the `messages.severity_level_E_ReservedEncoding` parameter.

Exception level change

Various unpredictable effects can occur if an MSR or CPS instruction is used to change privilege level, for example, leaving monitor mode with `SCR.NS==0`, leaving hyp mode, or entering user mode.

In preference, target code must use one of the exception return operations such as `RFE`, `ERET` or `SUBS PC`, `LR`.

A.2 Debug architecture support

In default configuration, the model implements ARMv7 debug architecture, as described in Part C of the *ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition*.

When the model is configured to include support for the Virtualization extension, then the model also implements ARM Debug Architecture v7.1.

A.2.1 Invasive debug

This section describes the invasive debug features that are available:

- *Debug events*
- *Debug exceptions*
- *Debug state*
- *Invasive debug authentication*.

Debug events

All software debug events are supported, that is, BKPT instruction, breakpoints, watchpoints and vector catch.

Only one halting debug event is currently handled. External debug request is triggered by means of the signal port `edgrq`. The halting debug events, Halt Request and OS Unlock Catch are not currently implemented.

The external signal port is not connected in the VE platform.

Debug exceptions

Monitor debug mode is available in the AEM when configured through the DBGDSCR. In this mode software debug events cause a debug exception resulting in either:

- a pre-fetch abort exception in the case of a BKPT instruction event, breakpoint event and vector catch event
- a data abort exception in the case of a watchpoint event.

Debug state

You can configure the AEM to enter debug state in the case of a software debug event through the DBGDCSR. Halting debug events always enter debug state if debug is both enabled and permitted.

Invasive debug authentication

Debug is enabled or disabled by two signals on the external debug interface, **dbgen** and **spiden**. Signal ports for non-invasive debug, `niden` and `spniden`, are present but are currently unused.

The external signal interface is not connected in the VE platform.

A.2.2 Non-invasive debug

Non-invasive debug is currently not supported.

A.2.3 Debug registers

Access to architectural debug features is available through the debug registers. The AEM supports software access to the registers by three interfaces:

- baseline CP14 interface
- extended CP14 interface
- memory-mapped interface.

The registers can also be accessed through an external bus interface. This interface is implemented as a pvbus type slave port.

A.3 IMPLEMENTATION DEFINED features

Some aspects of the behavior of the processor are IMPLEMENTATION DEFINED in the ARM architecture, meaning that they can legally vary between different processor implementations. Any code that is intended to be run portably across the multiple ARM implementations must take care when using any of these facilities, since they might or might not be present.

The AEM uses some of these features differently to existing implementations.

A.3.1 ACTLR

The *Auxiliary Control Register* (ACTLR) is an IMPLEMENTATION DEFINED register that usually contains control bits to enable or disable certain processor-specific features. It is normally written very early in the bootstrap sequence, and is often not changed afterwards.

In the AEM, the ACTLR has no functional bits implemented. Any changes to the configuration of the processor must instead be carried out using the parameters in the CADI interface of the model.

A.3.2 TLB lockdown

The ARM architecture defines the concept of TLB lockdown but does not specify the register format used in its implementation. Code that runs portably across ARMv7-A implementations cannot use TLB lockdown except by selecting processor-specific code segments.

The model supports TLB lockdown using the register scheme of the Cortex-A8, only if the size of the TLB is compatible with that format, that is, the TLB contains 32 entries or fewer. TLB lockdown is not available in cases where aggressive TLB pre-fetch is enabled.

A.3.3 Internal peripherals

Some ARM processors implement a set of internal peripheral components that are addressed by a range of programmer-visible physical addresses. The content and behavior of these peripherals are IMPLEMENTATION DEFINED.

In default configuration, the AEM implements the set of peripherals defined in Table A-2. These are similar to the devices described in the *Cortex-A9 MPCore Technical Reference Manual*.

The physical address of the peripherals is given as a fixed offset from the base address. The base address in the VE platform is `0x2C000000`. When using the AEM as a component in other platforms, you can configure the base address with the `PERIPHBASE` parameter.

Each component can also contain one or more interrupt sources, which are connected to the internal *Generic Interrupt Controller* (GIC) at the ID numbers specified.

Table A-2 Internal peripherals

Peripheral	Address range	Interrupt
SCU	`0x0000 - 0x00FF`	
GIC	`0x0100 - 0x01FF`	Legacy IRQ - 31 Legacy FIQ - 28

Table A-2 Internal peripherals (continued)

Peripheral	Address range	Interrupt
Global Timer	`0x0200` - `0x02FF`	27
Local Timer and Watchdog	`0x0600` - `0x06FF`	Watchdog - 30 Local Timer - 29
Interrupt Distributer	`0x1000` - `0x1FFF`	

use_Cortex-A15_peripherals mode

When the parameter `use_Cortex-A15_peripherals` is set to `true`, the AEM implements the set of peripherals defined in Table A-3. These are similar to the devices described in the *Cortex-A15 Technical Reference Manual*.

Table A-3 Internal peripherals (use_Cortex-A15_peripherals)

Peripheral	Address range
Reserved	`0x0000` - `0x0FFF`
Interrupt Controller Distributer	`0x1000` - `0x1FFF`
Interrupt Controller Physical CPU Interface	`0x2000` - `0x3FFF`
Interrupt Controller Virtual CPU Interface (Hypervisor view for requesting processor)	`0x4000` - `0x4FFF`
Interrupt Controller Virtual CPU Interface (Hypervisor view for all processors)	`0x5000` - `0x5FFF`
Interrupt Controller Virtual CPU Interface (Virtual Machine view)	`0x6000` - `0x7FFF`

The interrupt controller is a *Virtualized Generic Interrupt Controller* (VGIC).

Note

The generic timer becomes available by CP15 system control operations.

A.4 Trace

The AEM includes support for Model Trace Interface (MTI) trace plug-ins. This enables the generation of trace files that track the execution of instructions, memory accesses, and other related operations of the processor.

For more information about this interface and the usage of trace plug-ins, see the *Fast Models Tarmac Trace User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0532-/index.html`.

A.4.1 TarmacTraceAEM

The AEM model includes a trace plug-in with some enhanced capabilities, in addition to those described in the *Fast Models Tarmac Trace User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0532-/index.html`. It is loaded when specified by a command-line option to Model Debugger:

```
--trace-plugin <installation_path>/lib/TarmacTraceAEM.dll
```

Additional capabilities are enabled using the parameters listed in *trace_mmu*, which become available in the trace parameters hierarchy.

trace_mmu

Lines prefixed `TTW` describe the data that has been read by a translation table walk.

The line specifies the address from which the descriptor is read, the data returned from that address, and a summary of how that data is interpreted.

Lines prefixed `TLB` describe a translation entry that has been added to the TLB.

The line specifies whether the D-side or I-side TLB is being loaded, the size of the entry, and the mapping from VA to PA.