Fast Models

Version 9.5

Reference Manual

ARM

ARM DUI0834F

Fast Models

Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
A	31 May 2014	Non-Confidential	New document for Fast Models v9.0, from DUI0423Q for v8.3.
B	30 November 2014	Non-Confidential	Update for v9.1.
C	28 February 2015	Non-Confidential	Update for v9.2.
D	31 May 2015	Non-Confidential	Update for v9.3.
E	31 August 2015	Non-Confidential	Update for v9.4.
F	30 November 2015	Non-Confidential	Update for v9.5.

Non-Confidential Proprietary Notice

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents
Fast Models Reference Manual

Preface

This preface introduces the *Fast Models Reference Manual*.

It contains the following:

About this book

This document provides a reference for signaling, clock, bus, generic peripheral, and processor components included in Fast Models.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces the document.

Chapter 2 Protocols
Components communicate through connected ports. Ports have protocols that define the function calls for different connections.

Chapter 3 Processor Components
This chapter describes the *Code Translation* (CT) processor components in Fast Models.

Chapter 4 Peripheral and Interface Components
This chapter describes the generic *Programmer's View* (PV) peripheral components in Fast Models.

Chapter 5 Base Platform
This chapter describes the Base Platform system model.

Chapter 6 Microcontroller Prototyping System 2
This chapter describes the model of the hardware platform.

Chapter 7 Versatile Express Model
This chapter describes the components of Fast Models that are specific to the *Versatile Express* (VE) *Fixed Virtual Platform* (FVP) model of the hardware platform.

Glossary

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

See the *ARM Glossary* for more information.

Typographic conventions

italic
: Introduces special terminology, denotes cross-references, and citations.

bold
: Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`
: Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`mono`space
: Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`
: Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`
: Denotes language keywords when used outside example code.

`<and>`
: Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS
: Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title *Fast Models Reference Manual*.
- The number ARM DUI0834F.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Other information

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

Chapter 1
Introduction

This chapter introduces the document.

It contains the following sections:

1.1 About the models

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 hardware.

They sacrifice timing accuracy to achieve fast simulation execution speeds: 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.

1.2 Model capabilities

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

Fast Models can:

- Accurately model instructions.
- Correctly execute architecturally-correct code.
- Model some unpredictable behavior.

Fast Models cannot:

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

Related concepts

1.3.1 How accurate are Fast Models? on page 1-12.

Related references

1.3.6 Caches in Fast Models on page 1-14.

1.3 Fast Models accuracy

This section describes the aspects of Fast Models accuracy, and specific areas where models might differ from hardware.

This section contains the following subsections:

1.3.1 How accurate are Fast Models?

Fast Models aim to be accurate to the view of the system programmer. Architecturally correct software cannot tell 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.

1.3.2 Timing accuracy of Fast Models

Fast Models do not model instruction timing accurately. The simulation as a whole has a very accurate concept of timing, but the *Code Translation* (CT) processors do not claim to dispatch instructions with 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 a significant 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.

  ———————————

1.3.3 Bus traffic in Fast Models

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:

Timing differences
: Re-ordering and buffering of memory accesses, out-of-order execution, speculative prefetch and drain-buffers can cause timing differences. They are not modeled, since they are not visible to the programmer except in situations where a cluster 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.

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.

Related references

1.3.4 Instruction prefetch in Fast Models on page 1-13.

1.3.4 Instruction prefetch in Fast Models

The CT engine in the processor models relies on Fast Models PVBus optimizations. It only performs code-translation if it has been able to prefetch and snoop the underlying memory. It then need not 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).

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.

Related references

1.3.3 Bus traffic in Fast Models on page 1-12.

1.3.5 Out-of-order execution and write-buffers in Fast Models

Hardware memory is Weakly Ordered, but Fast Models memory is Strongly Ordered.

The CT implementation executes instructions sequentially. One instruction is retired before the next starts to execute. In a real processor, multiple memory accesses can be outstanding, 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 cluster code.

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

1.3.6 Caches in Fast Models

Fast Models with cache-state modeling enabled can replicate some types of failure-case, but not all types.

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

1.3.7 Global exclusive monitor in Fast Models

This section describes the behavior of the global exclusive monitor, when `cache-state-modelled` is `false` and when it is `true`.

Whether `cache-state-modelled` is `false` or `true`, when the system enables the data caches, they implement the monitor for cacheable nonshared and shared memory using the cache coherency protocol. The monitor includes other clusters that are connected with a cache coherent interconnect. Most global monitor behavior, like exclusive granule size, is the same whether `cache-state-modelled` is `false` or `true`. In particular, the data caches follow behavior that the ARM® architecture prescribes in both cases. However, some behaviors do rely on the full modeling of cache functionality. An example is that the clearing of a monitor as a result of a cache line eviction does not happen with `cache-state-modelled` as `false`.

The RAMDevice component and PVBus to AMBAPV bridges do not contain global monitors. As a result, exclusive stores that reach a RAMDevice fail unless they go through an exclusive monitor. Some platforms might need a global monitor outside of the cache coherency domains. These platforms must include a system level monitor in the same place in the bus hierarchy as in the hardware. See the FVP_VE and FVP_Base example platforms for examples of how to use the PVBusExclusiveMonitor component.

1.4 Processor implementation

This section outlines the differences between the code translation models and the hardware.

This section contains the following subsections:

1.4.1 Caches in PV models

Some processor models have PV-accurate caches, but others do not model Level 1 or Level 2 caches.

Cores that have PV-accurate cache implementation provide a functionally-accurate model. For more information, see the processor component descriptions.

Other PV models do not model Level 1 or Level 2 caches. The system coprocessor registers related to cache operations permit cache aware software to work, but in most cases they only 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.

Functional caches in Fast Models - about

Fast Models implement two types of cache model: register model and 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. Incorrect cache management might return incorrect data, as it would on real hardware.

Fast Models provide:

- System IPs that support caches.
- Register models of caches on all processors that support caches and also the PL310 cache controller (PL310_L2CC).
- Functional models of caches integrated into Cortex cores.

For a core with no L2 cache the configuration parameters are:

`icache-state-modelled`
: Set whether the I-cache has stateful implementation.

`dcache-state-modelled`
: Set whether the D-cache has stateful implementation.

For cores with an L2 cache the configuration parameters are:

`l1_icache-state-modelled`
: Set whether L1 D-cache has stateful implementation.

`l1_dcache-state-modelled`
: Set whether L1 I-cache has stateful implementation.

`l2_cache-state-modelled`
: Enable unified Level 2 cache state model.

Functional caches in Fast Models - performance

Enabling functional cache modeling is likely to reduce performance.

Enable the L1 and L2 functional caches together. For consistent system operation, ARM recommends that you either disable functional behavior completely or enable it for both I and D L1 caches and the L2 cache.

Cache enablement must be system wide. If you enable cache state modelling in any component then you must enable it in all components in the system, including all cores (L1 and L2) and any external cache controller (such as the PL310_L2CC) and any interconnect that has caches.

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

1.4.2 GICv3 in PV models

This section describes the model of the GICv3.

GICv3 - about

The PV models implement the *Generic Interrupt Controller architecture version 3* (GICv3). GICv3 is an *Early Access Candidate* (EAC).

The GICv3 specification and the associated models are subject to minor refinement. The models are of version 18 of the specification.

The GICv3 architecture defines two parts: a core interface (integrated into the core) and a distributor. You can configure all ARMv8-A cores to include a GICv3 interface. You can integrate a separate GICv3Distributor component into your platforms. Communication between the core and the distributor is over an architected packet interface. An internal communication protocol represents the packets that pass over this interface.

You can configure the GICv3 models in some platforms to act as though they were GICv2 or GICv2-M models. Even in this mode, you need a GICv3Distributor component and supported core. Configure them to comply with the same standard. The GICv3Distributor component, by default, includes a single *Interrupt Translation Service* (ITS) component. You can configure it with 0 or more such components. If using GICv2/GICv2-M compatibility modes, configure the number of ITS components to be 0.

Related references

4.4.21 GICv3Distributor component on page 4-312.

GICv3 - functionality

Some features differ in the model.

- Support of the GITS_CTLR.Quiesce bit is not complete.
- Support of ITS save/restore is not complete. Configuration stays within the model and it does not use allocated memory.
- GICD_CTLR.RWP does not perform adequately. This difference is only an issue if you use the distributor in systems with delaying interface between the distributor and the cores. DO NOT use this version of the model for simulation of the GIC in a setup where interfaces are not instantly reactive.
- Monolithic implementations do not correctly share the GICR_WAKER state between redistributors. Some GICR_WAKER behaviors better suit distributed implementations.

GICv4 - functionality

GICv4 is an extension of the GICv3 architecture allowing the direct injection of LPI to a virtualized system. Enable it with the `virtual-lpi-support` parameter of the GICv3Distributor component.

1.4.3 CP14 Debug coprocessor

Some models fully implement the CP14 Debug coprocessor registers. Other models only implement the DSCR register. This register reads as 0 and ignores writes.

External debugging must be used to debug systems containing PV models.

1.4.4 MicroTLBs

The models do not implement device accurate MicroTLBs, or system coprocessor registers related to MicroTLB state.

1.4.5 TLBs in PV models

The PV models implement *Translation Lookaside Buffers* (TLBs) and model most aspects of TLB behavior.

——— **Note** ———

If the `device-accurate-tlb` parameter is set to `false`, the simulation uses 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. Set `device-accurate-tlb` parameter to `true` if you require device accuracy.

———————————

These TLB registers do not have working implementations:

- Primary memory remap register.
- Normal memory remap register.

In addition, the simulation does not distinguish peripheral accesses from data accesses, so it ignores configuration of the peripheral port memory remap register.

1.4.6 Memory access in PV models

This section describes the kinds of memory access in PV models.

About memory access in PV models

PV models use a PVBusMaster subcomponent to communicate with slaves in a System Canvas generated system. This provides 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 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 access in PV models

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. ARM recommends that you 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 access in PV models

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 type `pv::MEMORY` provide the fastest possible memory access for PV processors.

Slaves declared as type `pv::DEVICE` are normally used for peripheral access.

1.4.7 Timing in PV models

Programmer's View (PV) models are loosely timed.

- 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 difference in behavior can affect sequential access of device registers, where devices are expecting time to move on between accesses.
- DMA to and from *Tightly Coupled Memory* (TCM) is atomic.

1.4.8 VIC ports in PV models

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 signals the vectored interrupt address from the external VIC.
- The vic_ack port signals the VIC that an interrupt has been detected and is being serviced.

The expected interrupt sequence is:

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.

1.5 Processor CADI implementation

Processor models implement a subset of CADI functionality.

Table 1-1 CADI broker implementation

Features	Implemented	Remarks
Factories	Yes	Model DLL provides a single factory.
Instances	Yes	Factory can only instantiate one model at a time.

Table 1-2 CADI target implementation

Features	Implemented	Remarks
# breakpoints	Yes	No intrinsic limit.
Breakpoint disabling	Yes	You can disable global breakpoints, but not user breakpoints set through `CADIBptSet()`. Debuggers usually disable breakpoints with `CADIBptClear()` and re-enable them with `CADIBptSet()`, so this does not affect debugger GUIs, but it does affect the CADI C++ interface. Global breakpoints are built-in and permanent, so `CADIBptSet()/Clear()` cannot set or delete them. With `CADIBptConfigure()`, you can only enable/ disable these global breakpoints to stop on certain processor events.
Breakpoint ignore count	No	-
Breakpoint formal conditions	No	-
Breakpoint free-form conditions	No	-
Breakpoint setting	Yes[a]	-
Breakpoint types	Yes: `CADI_BPT_MEMORY`, `CADI_BPT_PROGRAM`, `CADI_BPT_PROGRAM_RANGE`	All registers in the Vector register group have disabled register breakpoints. You cannot create new register breakpoints.
Breakpoint triggers	Yes: `CADI_BPT_TRIGGER_ON_MODIFY`, `CADI_BPT_TRIGGER_ON_READ`, `CADI_BPT_TRIGGER_ON_WRITE`	`CADI_BPT_TRIGGER_ON_WRITE` approximates `CADI_BPT_TRIGGER_ON_MODIFY`. If either is set, the simulation stops when a value is written, even when it does not change.
`CADIExecAssertException()`	No	-
CADICache API	No	-
`CADIExecSingleStep()`	Yes	Fast Models ignores the `stepCycle` and `stepOver` arguments.
Intrusive debug	Yes	Stop/start can affect processor scheduling: single stepping and multiple stepping both change scheduling, and so are intrusive.
Memory overlays	No	-
Memory read/write	Yes[a]	-

[a] Not permitted while target is running.

Table 1-2 CADI target implementation (continued)

Features	Implemented	Remarks
Reset levels	Yes: 1	-
Runnable	Yes	-
Stop simulation	Yes	-
Virtual to physical	No	-

1.6 CADI interactions with processor behavior

Architecturally, M series processors have different reset behavior to that of A and R series processors.

For both hardware and model processors, a reset consists of asserting, and then deasserting the reset pin. However:

M series
: The architecture documentation specifies 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.

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 processors interacts differently with CADI debugging functionality than it does for A and R series processors.

On A and R series processors, after the reset pin of the processor is asserted, a new application can be loaded using the `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 begins to fetch and execute from the PC.

On M series processors, these techniques do not work because after reset is deasserted the processor updates SP_main and the PC, overwriting the settings made by the CADI calls.

For these techniques to be possible, 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+4. Similarly, if the SP has been modified using CADI, SP_main retains the value written to it, otherwise it reads it from address VTOR+0.

1.7 CADI sync watchpoints

The CADI/*Synchronous CADI* (SCADI) interfaces of processor components support watchpoints. Memory components like RAMDevice do not.

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.

You can read additional information about the last of the hit watchpoints from these 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. `memoryBptPC` and `memoryBptAccessVA` are 32 or 64 bit depending on the architecture. `memoryBptAccessSize` and `memoryBptAccessRW` 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.

1.8 Non-CADI sync watchpoints

This section describes syncLevel enum values, semantics, and behaviors, with syncLevel values listed from fast to slow simulation, from fewer to more requirements.

This section contains the following subsections:

1.8.1 syncLevel definitions

Definitions for 0 - OFF, 1 - SYNC_STATE, 2 - POST_INSN_IO, and 3 - POST_INSN_ALL.

`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.

Quantum end detection guarantees that a quantum is not overshot indefinitely. Quantum end detection applies to (but is not limited to) backward branches, indirect jumps, exceptions, and atomic operation retries. In addition to temporal quantum end detection, some events may end a quantum, like executing a barrier, entering a low power state, or accessing a peripheral. Target software and simulation controllers must not rely on a specific scheduling pattern based on these quantum end check points.

Use cases: 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. SCADI can read the up-to-date values of the processor registers, including PC and instruction count. You cannot stop the simulation synchronously.

Quantum end detection is as for syncLevel 0.

Use cases: 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, except that you can stop the simulation 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).

Quantum end detection is as for syncLevel 1, plus it includes the end of LD/ST instructions.

Use cases: CADI watchpoints, external breakpoints, stopping from within LD/ST-related MTI callbacks.

`syncLevel 3: POST_INSN_ALL`

As for syncLevel 2, except that you can stop the simulation synchronously from within any instruction. The simulation stops immediately after the current instruction has been completely executed (post instruction).

Quantum end detection is as for syncLevel 2. This allows switching between syncLevels 2 and 3 without changing the simulation scheduling.

Use cases: a `Stop` from within arbitrary MTI callbacks such as the `INST` callback. This syncLevel is a fallback for all use cases that do not fall into syncLevels 0-2.

1.8.2 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 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.

These registers are only for debugging and visibility in the debugger, and syncLevel users do not usually access them at all:

`syncLevel`
: Current syncLevel. Read-only.

`syncLevelSyncStateCount`
: User counter. Read-write (use as read-only).

`syncLevelPostInsnIOCount`
: User counter. Read-write (use as read-only).

`syncLevelPostInsnAllCount`
: User counter. Read-write (use as read-only).

`minSyncLevel`
: Same as `min_sync_level` parameter (as follows). Read-write.

The *Register and *Unregister registers are for syncLevel users to register and unregister themselves, by writing the value 0 to them. Changes to the syncLevel become effective at the next stop event checkpoint. In addition, syncLevel users can write to these registers any time before simulation is running, for example from the `init()` simulation phase. The change then takes effect immediately when the simulation is run.

The 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 you can expect from the model. You must treat access to these registers as read-only. You may writing to them, 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 primarily for debugging purposes and for situations where a single global setting of syncLevel is sufficient. You may change this parameter at runtime, and changes 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.

1.9 SCADI

This section describes *Synchronous CADI* (SCADI) for components.

This section contains the following subsections:

1.9.1 About SCADI

A SCADI interface enables you to request a synchronous stop of the simulation. It offers direct synchronous, but unsynchronized, register and memory read access.

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 of code that could use this interface are SystemC and 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. It is an additional interface with clear semantics that differ slightly from CADI.

1.9.2 Intended uses of CADI and SCADI

Component Architecture Debug Interface (CADI) and *Synchronous CADI* (SCADI) have separate design principles.

CADI
: is 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. 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 guidelines are:

- Use CADI for execution control.
- Use CADI or SCADI for read/write registers and memory, depending on the situation:
  - 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.

1.9.3 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.

1.9.4 SCADI interface access

This section describes the `CAInterface::ObtainInterface()` method and how to use it.

About SCADI interface access

You can obtain the SCADI interface from the same CAInterface pointers that provide 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 pass 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 access the SCADI interface of the LISA component itself, use the `getSCADI()` function. This returns the SCADI interface of the LISA component itself.

To access the SCADI interfaces of other components in the system, use the `getGlobalInterface()` function. This returns a CAInterface pointer to a simulation-wide Component Registry.

You then have to use this registry pointer with the `obtainComponentInterfacePointer()` function to obtain the SystemTraceInterface. This interface provides a list of all components in the system that provide trace data. You can use this list to access the SCADI interfaces of all the components.

Access to SCADI from within SystemC

To access the SCADI interfaces from within SystemC, use the `scx_get_global_interface()` function. This returns a CAInterface pointer to a simulation-wide Component Registry.

You then have to use this registry pointer with the `obtainComponentInterfacePointer()` function to obtain the SystemTraceInterface. This interface provides a list of all components in the system that provide trace data. You can use this list to access the SCADI interfaces of all the components.

Access to SCADI from MTI plug-ins

MTI plug-ins 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 SystemC or LISA use cases.

Example code for retrieving a SCADI interface pointer of a component

This code demonstrates how to use `getGlobalInterface()` to retrieve a SCADI interface pointer of a specific component.

From within LISA, use `getGlobalInterface()`. From within SystemC, it is assumed that you have a pointer to the Fast Models platform called `platform`. From within an MTI plug-in, you do not have to call `getGlobalInterface()`, and you can use the CAInterface pointer passed to `RegisterSimulation()`.

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 SystemC peripheral
  // - 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 = scx::scx_get_global_interface();
  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));
}
```

1.9.5 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 that these read access functions return might differ from the values returned by the corresponding CADI functions because some of the accessed resource values might change 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`.

1.10 Checkpoints

Checkpointing allows you to save the state of components in a simulation, and restore the simulation when you start a new simulation. Saved states are called checkpoints.

Core components that support checkpointing are:

- ARMCortexA57x*n*CT.
- ARMCortexA53x*n*CT.
- ARMCortexA72x*n*CT.

Checkpointing limitations are:

- Support for AArch32 state is not complete. Issues include interworking with AArch64 state.
- No support for the GICv3 interface.
- No support when using the functional model of the caches (when `cache-state-modelled` is on).
- No preservation of TLB contents.
- No preservation of exclusive monitor states.
- No preservation of PMU states.
- No preservation of architectural debug state.

These peripheral components pass checkpointing support tests, some with limits to their support:

PL011_Uart
: No support of the use of input and output files (`in_file` and `out_file` parameters).

PL110_CLCD and PL111_CLCD
: No known limits.

CCI400
: No support when using the functional model of the caches (when `cache-state-modelled` is on).

GIC_400
: No known limits.

MemoryMappedCounterModule
: No known limits.

Model Shell supports checkpointing with the `--save` and `--restore` command-line options:

`--save dirname`
: On simulation exit, save checkpoint to the directory `dirname`.

`--restore dirname`
: On simulation startup, restore checkpoint to the directory `dirname`.

Use `--save` and `--restore` to restore a checkpoint on simulation startup and save a checkpoint on exit. For example:

```
model_shell -m $MODEL --restore checkpoint_dir --save checkpoint_dir
```

You can control checkpointing from a custom CADI client, from a SystemC application, with PyCADIShell, and with LISA+ components.

See `ARM/FastModelsPortfolio/examples/CADIClient/checkpointing` for an example of how to save and restore a checkpoint using CADI.

See `ARM/FastModelsPortfolio/examples/SystemCExport/EVS_Checkpointing` for a SystemC example.

See `ARM/FastModelsPortfolio/examples/python/checkpointing.py` for an example of how to perform checkpointing with PyCADIShell.

See `ARM/FastModelsPortfolio/examples/LISAPlus/Checkpointing` for an example of how to provide LISA+ components with checkpointing support.

The checkpointing feature is under development. It has limitations:

- The checkpointing API will change in future releases of Fast Models.
- Checkpoints are not compatible between different versions of Fast Models. Do not use a checkpoint from this version of Fast Models to restore the simulation state with a new version of Fast Models.
- Many components do not support checkpointing. None of the Fixed Virtual Platforms that shipped with Fast Models support checkpointing, because they include components that do not have checkpointing support.
- Remote clients that are connected to a CADI server cannot perform checkpointing.
- LISA+ checkpointing elements are likely to change in future releases of Fast Models.

1.11 Use of the TelnetTerminal

This virtual component permits the transfer of UART data between a TCP/IP socket on the host and a serial port on the target.

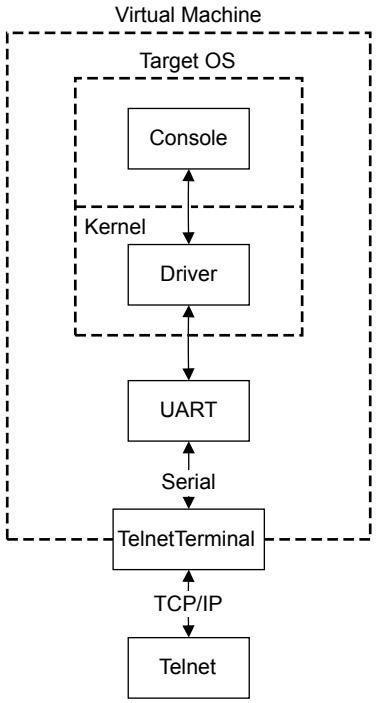

Figure 1-1 TelnetTerminal block diagram of one possible relationship between the target and host through the TelnetTerminal component

The TelnetTerminal block is what you configure when you define TelnetTerminal component parameters. The Virtual Machine is your model system.

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 TelnetTerminals by modifying the corresponding component parameters.

If the TelnetTerminal 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 TelnetTerminal instance can be defined when your model system starts. The actual value of the port used by each TelnetTerminal 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 TelnetTerminal component in telnet mode or raw mode.

Telnet mode

Support a subset of the RFC 854 protocol: the terminal participates in negotiations between the host and client concerning what is and is not supported, but there is no flow control.

Raw mode

The byte stream passes unmodified between the host and the target. The terminal does not participate in initial capability negotiations between the host and client, and instead acts as a TCP/IP port. You can use this feature to directly connect to your target through the TelnetTerminal component.

Related concepts

TelnetTerminal - about on page 4-390.

Related references

4.4.60 TelnetTerminal component on page 4-389.
TelnetTerminal - parameters on page 4-390.

1.12 Installing Telnet on Microsoft Windows

This section describes how to install the Telnet application on Microsoft Windows 7.

Microsoft Windows 7 does not include the Telnet application by default.

Prerequisites

Download the Telnet application by following the instructions on the Microsoft web site.

Procedure

1. Select **Start** > **Control Panel** > **Programs and Features**.
   A window opens that lets you uninstall or change programs.
2. Select **Turn Windows features on or off** in the left-hand side bar.
   The Windows Features dialog opens.
3. Select the **Telnet Client** check box and click **OK**.
   The installation might take several minutes to complete.

Related information

Microsoft Windows: 'Telnet: frequently asked questions'.

1.13 Network set up

This section describes how to set up networks using the serial interface that the TelnetTerminal component exposes.

This section contains the following subsections:

1.13.1 User mode networking

This section describes 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.

Note

- You can use TCP and UDP over IP, but not ICMP (ping).
- You can only use DHCP 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.

1.13.2 TAP/TUN networking

This section describes TAP/TUN networking.

TAP/TUN networking - limitations

This section describes general limitations of TAP/TUN networking for models.

DHCP

If the host uses Dynamic DNS, it inserts records into DNS. If you manage this host with DHCP, then installing TAP networking can cause failure to register in the DNS. After the physical device attaches to the bridge device, the DHCP client reruns, but the DHCP request does not have the correct hostname.

WiFi

Most WiFi adaptors do not implement the required support for TAP networking to work.

Setting up a network connection for Microsoft Windows

This section describes how to set up a network connection.

Prerequisites

Install the third-party IP package in the Fast Models networking environment.

Procedure

1. Close all non-essential applications.
2. Install the TAP driver and configure it:
   a. Locate Fast Model Portfolio, default `C:\Program Files\ARM\FastModelPortfolio_X.Y`.
   b. Execute the `ModelNetworking\add_adapter_64.bat` file.
3. Select **Start** > **Control Panel** > **Network Connections** and locate the newly installed TAP device.

4. Select at least one real Ethernet adapter connection. Press the **Ctrl** key to multi-select.
5. Right-click and select **Bridge Connections**.

Configuring the networking environment for Microsoft Windows

This section describes how to set the parameters to make a network connection.

Note

- This procedure has not been tested with wireless network adapters.
- 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.

Prerequisites

Use System Canvas or a related Fast Models tool to load a project and build a model, or use Model Shell or Model Debugger to start a prebuilt model. Then, select a component.

Procedure

1. Set the appropriate parameters on the `HostBridge` and `SMSC_91C111` components.

   ```
   hostbridge.interfaceName="ARM<x>"
   smsc_91c111.enabled=1
   ```

   `ARM<x>` is an adapter that is built into the network bridge, and is 0 by default. Use `x+1` if `ARM<x>` exists.
2. Optional: Set another parameter on the device, if you have to enable promiscuous mode for the TAP device.
   a. Select **Start** > **Run** > **cmd.exe**.
   b. Enter the show bridge and enable commands.

      ```
      netsh bridge show adapter
      netsh bridge set adapter <N> forcecompatmode=enable
      ```

      `<N>` is the id number of the TAP adapter that the first command lists.

Usage of tap_setup_32.exe or tap_setup_64.exe for Microsoft Windows

This section describes some 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 for Microsoft Windows

This section describes how to uninstall TAP adapters from the base location of the Fast Models Portfolio, default `C:\Program Files\ARM\FastModelPortfolio_X.Y`.

Procedure

1. Run the `remove_all_adapters_xx.bat` file.
   - `ModelNetworking\remove_all_adapters_32.bat` for 32-bit Microsoft Windows.
   - `ModelNetworking\remove_all_adapters_64.bat` for 64-bit Microsoft Windows.
2. If the uninstallation does not delete the bridge, delete it manually.
   a. Close all non-essential applications.
   b. Select **Network Connections**.
   c. Right-click on the bridge and select **Delete**.

Setting up a network connection for Red Hat Enterprise Linux

This section describes how to set up a network connection.

——— Note ———

Perform this procedure once for each host machine.

——— Restriction ———

The set-up and configuration instructions assume that your network provides IP addresses by DHCP. Otherwise, consult your network administrator.

Prerequisites

Ensure that the `brctl` utility is on your system. This utility is in the installation package, but ARM recommends that you use the standard Linux bridge utilities, which are in the Linux distribution.

Procedure

1. In a shell, change to the `bin` directory of your Fast Models installation.
   For example, `cd /FastModelPortfolio_X.Y/ModelNetworking`.
2. Run these scripts from the directory in which they are installed, because they do not work correctly if run from any other location:

   32-bit operating system
   Run `add_adapter_32.sh` as root. For example, `sudo ./add_adapter_32.sh`.
   64-bit operating system
   Run `add_adapter_64.sh` as root. For example, `sudo ./add_adapter_64.sh`.
3. A prompt appears: **Specify the TAP device prefix:(ARM)**. Select **Enter** to accept the default.
4. A prompt appears: **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.
5. A prompt appears: **Enter the network adapter which connects to the network:(eth0)**. Select **Enter** to accept the default, or input the name of a network adapter that connects to your network.
6. A prompt appears: **Enter a name for the network bridge to create:(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.
7. A prompt appears: **Enter the location to write the init script to:(/etc/init.d/FMNetwork)**. Select **Enter** to accept the default, or input another path with a new filename in an existing directory.
8. A prompt appears: **WARNING: the script creates 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.

9. A prompt appears: it 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.

   ————————————

Postrequisites

The network bridge is disabled after the host system is reset. To configure the host system to support bridged networking, you might have to create links to the `init` script (FMNetwork). The script suggests some appropriate links for Red Hat Enterprise Linux 5.

The default firewall configuration on Red Hat Enterprise Linux 5 blocks packet transmission across the TAP networking bridge device. You can disable the firewall. If the context makes this unwise, then add firewall rules to allow transmission. These iptables commands configure the firewall to allow packets across the bridge device:

```
iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
service iptables save
service iptables restart
```

Setting up a network connection for Ubuntu Linux

This section describes how to set up a network connection.

The model networking scripts do not work reliably on Ubuntu 12.04. To use TAP networking with Fast Models on Ubuntu 12.04, set up a TAP device manually by following these steps. This guide uses a network interface `eth0` and a username `fmuser`. Replace these values as appropriate.

———— **Note** ————

Typographic errors when modifying network configuration can cause the computer to fail to connect to the network. ARM recommends performing these steps on a machine that you have physical access to.

————————————

Procedure

1. ———— **Note** ————

   Edit the interfaces file with root privileges.

   ————————————

   Add `eth0` to `/etc/network/interfaces` if it is not there.

   This step stops network-manager from managing `eth0`. It can result in network-manager indicating there is no network connection even if there is.

Option	Description

For an interface using DHCP	Add:
	`auto eth0`
	`iface eth0 inet dhcp`

Configure a static IP address	Add the static information, for example:
	`auto eth0`
	`iface eth0 inet static`
	`address 192.168.0.2`
	`netmask 255.255.255.0`
	`gateway 192.168.0.1`

The network notifier applet launches the GUI tool network-manager. It automatically configures network devices that `/etc/network/interfaces` does not manage, and sets up devices in a way that is incompatible with bridging. This step makes sure that network-manager does not manage the network interface that you want to bridge to. If you are unsure how to configure your network interface, ask your network administrator.

2. Install the bridge-utils package.
   Run:

   ```
   sudo apt-get install bridge-utils
   ```

   This step provides the `brctl` command, for creating and managing the network bridge.
3. Create a bridge device by adding an entry to `/etc/network/interfaces`.

   ```
   auto armbr0
   iface armbr0 inet dhcp
   pre-up ifconfig eth0 0.0.0.0 promisc
   post-down ifconfig eth0 0.0.0.0 -promisc
   bridge_ports eth0
   ```

   The pre-up and post-down lines give commands to execute before bringing up `armbr0` and after bringing it down. These commands put `eth0` into promiscuous mode at pre-up and take it out of promiscuous mode at post-down. Promiscuous mode makes sure that the hardware does not filter out packets for the virtual ethernet device.

   This step creates a bridge device that is called `armbr0` from Fast Models TAP devices to the physical network.
4. Create the TAP devices.

   TAP devices need permission for specific users, so create one for each user who is to run the model with the virtual ethernet device. For each user, add lines like the following example to the armbr0 section of `/etc/network/interfaces`.

   ```
   pre-up ip tuntap add dev ARMfmuser mode tap user fmuser
   pre-up ifconfig ARMfmuser 0.0.0.0 promisc
   post-down ip tuntap del dev ARMfmuser mode tap
   ```

   This step creates TAP devices for users.
5. Add the TAP devices to the bridge by adding them to the bridge_ports line in the `armbr0` section.

   ```
   bridge_ports eth0 ARMfmuser
   ```

   The added `/etc/network/interfaces` code now looks like:

   ```
   auto eth0
   iface eth0 inet dhcp
   ```
6. Bridge the Fast Models TAP devices to the physical network.

   ```
   auto armbr0
   iface armbr0 inet dhcp
   pre-up ifconfig eth0 0.0.0.0 promisc
   post-down ifconfig eth0 0.0.0.0 -promisc
   pre-up ip tuntap add dev ARMfmuser mode tap user fmuser
   pre-up ifconfig ARMfmuser 0.0.0.0 promisc
   post-down ip tuntap del dev ARMfmuser mode tap
   bridge_ports eth0 ARMfmuser
   ```
7. Restart network services.
   - Run:

     ```
     sudo /etc/init.d/networking restart
     sudo service network-manager restart
     ```
   - Restart the computer to restart the network.

   Both choices disconnect and reconnect all network interfaces.

Configuring the networking environment for Linux

This section describes how to set the parameters to make a network connection.

Note

Firewall software might block network traffic in the network bridge, and result in a networking failure. If the model does not work after configuration, check the firewall settings.

Prerequisites

Use System Canvas or a related Fast Models tool to load a project or model, and then select a component.

Procedure

1. Set the parameters on the `HostBridge` and `SMSC_91C111` components.

```
hostbridge.interfaceName="ARM<username>"
smsc_91c111.enabled=1
```

`ARM<username>` is an adapter that is built into the network bridge.

Related information

Fast Models User Guide.

Solutions to networking issues on Linux

This section describes how to solve networking issues.

The model networking works after initial setup, but stops working after reboot

Set the correct access permissions for the `/dev/net/tun` device, by executing `chmod 666 /dev/net/tun` as root. To preserve the change across reboots, modify the udev rules of the TAP device: open `/etc/udev/rules.d/50-udev.rules` as root, and find the line:

```
KERNEL=="tun", NAME="net/%k"
```

If it does not have `MODE="0666"` at the end of the line, append `MODE="0666"`:

```
KERNEL=="tun", NAME="net/%k", MODE="0666"
```

Model networking installs correctly, but when a model starts up, the model cannot receive packets

Disable the firewall on the host machine, or add the TAP device to `trusted devices`.

Note

Refer to the vendor documentation manual.

Disabling and re-enabling networking for Linux

This section describes how to enable and disable networking with an `init` script.

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.

Procedure

- To disable networking without uninstalling it, invoke the installed `init` script (by default, `/etc/init.d/FMNetwork`) as root with the parameter `stop`.
  `sudo /etc/init.d/FMNetwork stop`
- To re-enable networking, invoke the `init` script as root with the parameter `start`.
  `sudo /etc/init.d/FMNetwork start`

Uninstalling networking for Linux

This section describes the steps to uninstall a network.

Procedure

1. In a shell window, change to the `bin` directory of your Fast Models installation.
   `cd /FastModelPortfolio_X.Y/ModelNetworking`
2. Run `uninstall.sh` as root, passing the location of the `init` script (FMNetwork) as an argument.
   `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.

   ———————————

Postrequisites

To uninstall networking:

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.

Chapter 2
Protocols

Components communicate through connected ports. Ports have protocols that define the function calls for different connections.

It contains the following sections:

2.1 AMBA-PV protocols

This section describes the AMBA-PV protocols.

This section contains the following subsections:

2.1.1 AMBA-PV protocols - about

The AMBA-PV components and protocols permit you to model a platform that interfaces with an ARM AMBA-based system.

The protocols are:

AMBAPV
: Models the AMBA protocols AXI4, AXI3, AHB and APB.

AMBAPVACE
: Models the AMBA ACE and DVM protocols.

AMBAPVSignal
: Models interrupts.

AMBAPVSignalState
: Transfers and receives signal states.

AMBAPVValue
: Models propagation of 32-bit integer values between components.

AMBAPVValue64
: Models propagation of 64-bit integer values between components.

AMBAPVValueState
: Permits a master to retrieve the current value from a slave, using 32-bit integer values.

AMBAPVValueState64
: Permits a master to retrieve the current value from a slave, using 64-bit integer values.

There are ready-to-use components that provide you with conversions between protocols.

Related concepts

4.2.1 AMBA-PV components - about on page 4-272.

Related information

AMBA-PV Extensions to TLM 2.0 Developer Guide.

2.1.2 AMBAPV protocol

The AMBAPV protocol defines behaviors for single read and single write transactions. This covers ARM AMBA AXI4, AXI3, AHB and APB bus protocol families, at the PV level.

In addition, the AMBAPV protocol provides support for AMBA protocol additional control information:

- 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)
```

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 must write the bytes of a word in increasing significance as the array index increases and a big endian master must 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.

2.1.3 AMBAPVACE protocol

This protocol defines behaviors for bus transactions. This covers ARM AMBA ACE and DVM bus protocol families, all at the PV level.

In addition, this protocol provides support for AMBA protocol additional extension information:
- 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. You must add an `amba_pv::amba_pv_extension` object to the transaction before calling this behavior, setting the `socket_id` parameter 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. You must add an `amba_pv::amba_pv_extension` object to the transaction before calling this behavior, setting the `socket_id` parameter 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. You must add an `amba_pv::amba_pv_extension` object to the transaction before calling this behavior, setting the `socket_id` parameter 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. You must add an `amba_pv::amba_pv_extension` object to the transaction before calling this behavior, setting the `socket_id` parameter 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()`
: Use this 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)
```

The generic payload data is in the format of 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 must write the bytes of a word in increasing significance as the array index increases and a big endian master must 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

To maintain memory coherency, apply these rules for debug transactions.

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 maintain memory coherency, apply these rules 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 must be broadcast back upstream as debug snoop transactions to all ports other than the one the request came in on. Incoming debug snoops must propagate upwards. Debug reads can terminate as soon as they hit a cache.

Debug writes must continue until they propagate 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 must observe a weak memory-order model. Any component that can block a thread whilst responsible for the payload of an in-flight transaction must take particular care. 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.

Only use DMI when you can guarantee that subsequent transactions do not result in any state transitions. This means, in general, do not use DMI for ACE coherent cacheable transactions.

2.1.4 AMBAPVSignal protocol

This protocol defines a single behavior to permit masters to change the state of signals such as interrupt. Strictly speaking, AMBA3 does not cover this behavior, but the AMBA-PV components do provide it.

`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.

2.1.5 AMBAPVSignalState protocol

This 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.

2.1.6 AMBAPVValue protocol

This 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.

2.1.7 AMBAPVValue64 protocol

This 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.

2.1.8 AMBAPVValueState protocol

This protocol permits propagation of 32-bit integer values between components and their retrieval 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.

2.1.9 AMBAPVValueState64 protocol

This protocol permits propagation of 64-bit integer values between components and their retrieval 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.

2.2 Clocking protocols

This section describes the clocking protocols.

This section contains the following subsections:

2.2.1 Clocking protocols - about

The clocking components and protocols provide a mechanism for systems to regulate the execution rate of components.

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

Related concepts

4.3.1 Clocking components - about on page 4-282.

2.2.2 ClockRateControl protocol

This protocol sets the ratio of this component.

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

Set the multiplier/divider that determines the clock divider ratio.

$$clk\_out_{freq} = clk\_in_{freq} * mul / div$$

Related references

ClockDivider - ports on page 4-282.

2.2.3 TimerCallback protocol

This protocol invokes a behavior on the component that set the timer. Do not use the `timer_control` port of the timer 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.

Related references

ClockTimer - ports on page 4-283.

2.2.4 TimerCallback64 protocol

This protocol invokes a behavior on the component that set the timer. Do not use the `timer_control` port of the timer 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.

Related references

ClockTimer64 - ports on page 4-284.

2.2.5 TimerControl protocol

This protocol controls the actions of the 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.

`set(uint32_t ticks) : void`
: Set the timer to countdown the given number of ticks.

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

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

`remaining() : uint32_t`
: Return how many ticks remain before the callback is invoked.

Related references

ClockTimer - ports on page 4-283.

2.2.6 TimerControl64 protocol

This protocol controls the actions of the 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. Canceling a timer when it is not active has no effect.

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

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

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

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

Related references

ClockTimer64 - ports on page 4-284.

2.3 Debug interface protocols

This section describes the debug interface protocols.

This section contains the following subsections:

2.3.1 Debug interface protocols - about

LISA components can expose aspects of their internal state 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 LISA+ resource REGISTER declaration, especially the `read_function` and `write_function` attributes.

Memories and other memory-like objects
: See the LISA+ resource MEMORY declaration, especially the `read_function` and `write_function` attributes.

Disassembly
: See CADIProtocol and CADIDisassemblerProtocol.

Instruction count
: See CADIProtocol.

Setting, configuring and clearing of breakpoints
: For example, code breakpoints, register breakpoints, watchpoints. See CADIProtocol.

Single stepping and instruction stepping support.
: See CADIProtocol.

If displaying and editing memory and registers are sufficient, you need not implement CADIProtocol and CADIDisassemblerProtocol. The component must implement them to enable the other debug features.

CADIProtocol and CADIDisassemblerProtocol permit you to implement the features on a *Component Architecture Debug Interface* (CADI) interface level, where the component code takes responsibility for the implementation of these interfaces.

Related information

Component Architecture Debug Interface v2.0 Developer Guide.
LISA+ Language for Fast Models Reference Manual.

2.3.2 CADIDisassemblerProtocol protocol

To support disassembly, implement all of these functions. None of them is optional.

These 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. The functionality of this port is then exposed by `CADIProtocol::CADIGetDisassembler()`. See CADIProtocol 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 must return `insn_type = eslapi::CADI_DISASSEMBLER_INSTRUCTION_TYPE_NOCALL;` and return `eslapi::CADI_DISASSEMBLER_STATUS_OK;`

Related references

2.3.3 CADIProtocol protocol on page 2-52.

Related information

Component Architecture Debug Interface v2.0 Developer Guide.

2.3.3 CADIProtocol protocol

This protocol supports debugging. To add breakpoint support, implement `CADIBpt...()` functions.

Note

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

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

```
    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;
```

You need to implement all of these functions 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()` add and remove breakpoints.
- `CADIBptConfigure()` 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 System Generator provides for this component 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, you can add breakpoint support. 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 automatically follows 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.

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 finishes.

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

The following function is for debugging purposes only. Do not implement it. 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.

2.4 Peripheral protocols

This section describes the peripheral protocols.

This section contains the following subsections:

2.4.1 AudioControl protocol

This protocol has get and release audio buffer behaviors.

`getPVAudioBuffer`
: Get an underlying host buffer for audio output.

`releasePVAudioBuffer`
: Release an underlying host buffer.

Related references

AudioOut_File - ports on page 4-288.

AudioOut_SDL - ports on page 4-288.

PL041_AACI - ports on page 4-346.

2.4.2 CharacterLCD protocol

This protocol has the behaviors `setLayoutInfo` and `draw`.

`setLayoutInfo`
: sets the width and height of the touchscreen.

`draw`
: sets the character information.

2.4.3 FlashLoaderPort protocol

This protocol initializes the flash contents at model startup and saves flash contents to a file when the model terminates.

`loadFlashFile(FlashLoader *) : uint32`
: Initiate loading of the flash contents.

`saveFlashFile(FlashLoader *) : uint32`
: Save the flash contents to a file.

Related references

IntelStrataFlashJ3 - ports on page 4-324.

2.4.4 GUIPollCallback protocol

This protocol defines one method that signals to the visualization component the end of the update period. You can invoke this callback even when the simulation is stopped.

`gui_callback() : void`
: This is sent by the GUIPoll component, at the configured period.

Related references

GUIPoll - ports on page 4-412.

2.4.5 ICS307Configuration protocol

This protocol sets the divider ratio of an ICS307 component at runtime. The output clock rate alters accordingly and any dependent components react to the clock rate change according to their defined behavior.

`setConfiguration(uint32_t vdw, uint32_t rdw,uint32_t od): void`
: Set the parameters for deriving the clock divider ratio.

vdw
: Range: 0-255.

rdw
: Range: 0-255.

od
: Range: 0-7.

Related references

ICS307 - ports on page 4-323.

2.4.6 KeyboardStatus protocol

This protocol passes keyboard events to a component such as the PS2Keyboard component.

Events are only sent when the visualization window is in focus. Keyboard combinations that 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.

Related references

PS2Keyboard - ports on page 4-380.

7.5.2 VEVisualisation - ports on page 7-457.

2.4.7 LCD protocol

This Visualisation Library signaling 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 behaviors are:

`lock() : VisRasterLayout *`
: Lock the raster region of the LCD in preparation for rendering.

`unlock()`
: Unlock the raster region, ready to be updated on the screen.

`update(int x, int y, unsigned int w, unsigned int h)`
: Update the selected rectangular area on screen from the raster buffer.

`setPreferredLayout(unsigned int width, unsigned int height, unsigned int depth)`
: 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.

Related references

7.5.2 VEVisualisation - ports on page 7-457.

2.4.8 LCDLayoutInfo protocol

This protocol has the behavior `setLayoutInfo`.

`setLayoutInfo`
: sets the width and height of the touchscreen.

2.4.9 MMC_Protocol protocol

This protocol describes an abstract, untimed interface between an MMC controller and an MMC or SD card.

The protocol contains 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. for further information on the protocol implementation, see the source file, `%PVLIB_HOME%\LISA\MMC_Protocol.lisa`.

Use of this protocol assumes knowledge of the MultiMediaCard specification, available from the MultiMediaCard Association, `www.mmca.org`.

The protocol has these 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:

`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:

`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.

Related references

MMC - ports on page 4-331.
PL180_MCI - ports on page 4-357.

2.4.10 MouseStatus protocol

This protocol passes 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.

Related references

7.5.2 VEVisualisation - ports on page 7-457.

2.4.11 PL080_DMAC_DmaPortProtocol protocol

This 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.

2.4.12 PS2Data protocol

This protocol is for communication between the KMI and a PS/2-like device.

For efficiency, the interface is 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.

Related references

PL050_KMI - ports on page 4-348.

2.4.13 PVBusSlaveControl protocol

The PVBusSlaveControl protocol enables you to access and modify the underlying memory that PVBusSlave controls.

`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.

`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 to handle bus accesses. It can be one of:

- `pv::ACCESSMODE_MEMORY`
- `pv::ACCESSMODE_DEVICE`
- `pv::ACCESSMODE_IGNORE`
- `pv::ACCESSMODE_ABORT`

`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`.

The limit pointer returns the device address for the limit of the accessible memory.

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 device_base, pv::bus_addr_t device_limit, const uint8_t *storage)`
`provideWriteStorage(pv::bus_addr_t device_base, pv::bus_addr_t device_limit, uint8_t *storage)`
`provideReadWriteStorage(pv::bus_addr_t device_base, pv::bus_addr_t device_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.

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.

`provideReadStorageEx(pv::bus_addr_t device_base, pv::bus_addr_t device_limit, const uint8_t *storage, double latency)`
`provideWriteStorageEx(pv::bus_addr_t device_base, pv::bus_addr_t device_limit, const uint8_t *storage, double latency)`
`provideReadWriteStorageEx(pv::bus_addr_t device_base, pv::bus_addr_t device_limit, uint8_t *storage, double read_latency, double write_latency)`

These methods take additional parameters to specify average latencies (in seconds) per byte, used when Timing Annotation is enabled.

In all other aspects they behave the same as `provideReadStorage()`, `provideWriteStorage()` and `provideReadWriteStorage()`, respectively.

```
slave behavior getRegionIterHandle(): uint32_t;
slave behavior getNextRegionInfo(uint32_t iter_handle, pv::PVBusSlaveRegionInfo *info) : bool;
slave behavior closeRegionIterHandle(uint32_t iter_handle);
```

These methods form an iterator-like API that allows a PVBusSlave providing storage to report all the regions of the address space that have backing store.

The iteration begins by calling `getRegionIterHandle()`. This allocates an iterator and if successful returns a nonzero `iter_handle` to identify it.

The caller can then repeatedly call `getNextRegionInfo()` with `iter_handle`. If it finds a region, the behavior returns `true` and writes to the `info` struct if the pointer is non-null. Access the data in the region using `getReadStorage()` or `getWriteStorage()`.

The implementation can return regions in any order. They can be of any size or alignment, but must not overlap.

The implementation need not report allocated regions that are filled entirely with the default fill pattern, or allocated regions that contain only the data they had at simulation start.

On reaching the last region, the iterator closes automatically. If the handle is invalid or there are no further regions, the behavior returns `false`.

A caller can close an iterator opened by `getRegionIterHandle()` at any time using `closeRegionIterHandle()`. This deallocates the iterator, and further uses of the handle are invalid.

Related references

PVBusSlave - ports on page 4-405.
pv::accessMode values on page 4-405.

2.4.14 PVDevice protocol

The PVDevice protocol enables you to implement support for memory-mapped device registers. Call the two methods 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 PVDevice protocol uses two behaviors to differentiate between transactions originating from the processor (loads and stores) and transactions originating from an attached debugger:

`slave behavior debugRead(pv::ReadTransaction tx) : pv::Tx_Result`
: This method enables the device to handle a debug read transaction.

`slave behavior debugWrite(pv::WriteTransaction tx) : pv::Tx_Result`
: This method enables the device to handle a debug write transaction.

The `debugRead` and `debugWrite` behaviors are called for all debug transactions.

Related references

PVBusSlave - ports on page 4-405.

4.5.12 PVBus C++ transaction and Tx_Result classes on page 4-408.

2.4.15 PVTransactionMaster protocol

This protocol instantiates a `pv::TransactionGenerator` object from a PVBusMaster.

`createTransactionGenerator( ) : pv::TransactionGenerator *`
: This behavior instantiates a new transaction generator to control the 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 terminate() {
delete tg;
}
```

Related references

PVBusMaster - ports on page 4-403.

2.4.16 SerialData protocol

This protocol is implemented as a parallel interface for efficiency. All communication is driven by the master port.

This protocol has behaviors:

`dataTransmit(uint16_t data) : void`
: Used by the master to send data to the slave.

Table 2-1 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.

Table 2-2 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.

Table 2-3 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.

Table 2-4 Bits for signalsGet()

Bits	Function
7:4	Reserved
3	DCD
2	DSR
1	CTS
0	RI

Related references

PL011_Uart - ports on page 4-340.

2.4.17 TZFilterControl protocol

This protocol controls the communication between filter units and control registers in the APB control block.

`checkPermission(const pv::TransactionAttributes *attributes_, pv::bus_addr_t page_base_, bool is_read_, pv::RemapRequest &req_, bool &abort_on_error_) : bool`
Check the permission of the transactions filtered by the filter unit. Optional slave behavior.

`isEnabled() : bool`
: Check if the filter unit is enabled or not. The APB control block controls the unit. Slave behavior.

`isSecureSlave() : bool`
: Check if the connected slave is secure or not. Optional slave behavior.

`setConfig(bool rd_spec_enable, bool wr_spec_enable, uint32_t action) : void`
: Pass the configurations to the filter. Optional master behavior.

Related references

TZC_400 - ports on page 4-391.

2.4.18 VirtualEthernet protocol

This protocol has the sendToSlave and sendToMaster 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 `$PVLIB_HOME/include/components/VirtualEthernet/Protocol`.

Related references

HostBridge - ports on page 4-321.

SMSC_91C111 - ports on page 4-383.

VirtualEthernetCrossover - ports on page 4-399.

2.5 Processor protocols

This section describes the processor protocols.

This section contains the following subsections:

2.5.1 CounterInterface protocol

This protocol connects the `cntvalueb` port on Generic Timer components to MemoryMappedCounterModule components.

This semi-opaque protocol is exportable across a SystemC interface using a custom bridge.

It is a LISA+ protocol.

Related references

ARM926CT - ports on page 3-267.
ARM968CT - ports on page 3-264.
ARM1136CT - ports on page 3-262.
ARM1176CT - ports on page 3-259.
ARMCortexA5CT - ports on page 3-229.
ARMCortexA5MPxnCT - ports on page 3-225.
ARMCortexA7xnCT - ports on page 3-220.
ARMCortexA8CT - ports on page 3-217.
ARMCortexA9MPxnCT - ports on page 3-208.
ARMCortexA9UPCT - ports on page 3-213.
ARMCortexA12xnCT - ports on page 3-204.
ARMCortexA15xnCT - ports on page 3-199.
ARMCortexA53xnCT - ports on page 3-187.
ARMCortexA57xnCT - ports on page 3-89.
ARMCortexM3CT - ports on page 3-251.
ARMCortexM4CT - ports on page 3-248.
ARMCortexR4CT - ports on page 3-242.
ARMCortexR5CT - ports on page 3-237.
ARMCortexR7MPxnCT - ports on page 3-233.

2.5.2 GICv3Comms protocol

This protocol connects the `gicv3_redistributor_s` port on components with GICv3 Core Interfaces to platform level GICv3Distributor components.

This semi-opaque protocol is exportable across a SystemC interface using the bridges GICv3CommsPVBus and PVBusGICv3Comms.

It is a LISA+ protocol.

Related references

ARMCortexA53xnCT - ports on page 3-187.
ARMCortexA57xnCT - ports on page 3-89.

2.5.3 InstructionCount protocol

This protocol has the behaviors `getValue()` and `getRunState()`.

`getValue() : uint64_t`
Obtain the number of instructions executed by the processor.

`getRunState() : uint32_t`
Obtain the power/run status of the processor.

Table 2-5 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

Related references

ARM926CT - ports on page 3-267.
ARM968CT - ports on page 3-264.
ARM1136CT - ports on page 3-262.
ARM1176CT - ports on page 3-259.
ARMCortexA5CT - ports on page 3-229.
ARMCortexA5MPxnCT - ports on page 3-225.
ARMCortexA7xnCT - ports on page 3-220.
ARMCortexA8CT - ports on page 3-217.
ARMCortexA9MPxnCT - ports on page 3-208.
ARMCortexA9UPCT - ports on page 3-213.
ARMCortexA12xnCT - ports on page 3-204.
ARMCortexA15xnCT - ports on page 3-199.
ARMCortexA53xnCT - ports on page 3-187.
ARMCortexA57xnCT - ports on page 3-89.
ARMCortexM3CT - ports on page 3-251.
ARMCortexM4CT - ports on page 3-248.
ARMCortexR4CT - ports on page 3-242.
ARMCortexR5CT - ports on page 3-237.
ARMCortexR7MPxnCT - ports on page 3-233.

2.5.4 v8EmbeddedCrossTrigger_controlprotocol protocol

This protocol connects the *Cross Trigger Interface* (CTI) on ARMv8 processor components to platform level *Cross Trigger Matrix* (CTM) components.

This opaque protocol is not exportable across a SystemC interface.

Related references

ARMCortexA53xnCT - ports on page 3-187.
ARMCortexA57xnCT - ports on page 3-89.

2.6 Signaling protocols

This section describes the signaling protocols.

This section contains the following subsections:

2.6.1 Signaling protocols - about

Many components use the signaling protocols to indicate changes in state for signals such as interrupt and reset.

Each signaling protocol has two variants:

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

2.6.2 Signal protocol

The Signal protocol has the setValue behavior.

`setValue(enum sg::Signal::State) : void`
: indicates a state change.

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

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

2.6.3 StateSignal protocol

The StateSignal protocol has the setValue and getValue behaviors.

`setValue(enum sg::Signal::State) : void`
: indicates a value change.

`getValue() : sg::Signal::State`
: reads the current value.

2.6.4 Value protocol

The Value protocol has the setValue behavior.

`setValue(uint32_t value)`
: indicates a state change.

2.6.5 Value_64 protocol

This protocol provides the rules to communicate with the *TrustZone Memory Adapter* (TZMA) to signal the remapped range X.

`setValue(uint64_t value)`
: Sets a 64-bit wide address to the slave port.

Related references

PVBusRange - ports on page 4-404.

2.6.6 ValueState protocol

The ValueState protocol has the setValue and getValue behaviors.

`setValue(uint32_t value) : void`
: indicates a value change.

`getValue() : uint32_t`
: reads the current value state.

Chapter 3
Processor Components

This chapter describes the *Code Translation* (CT) processor components in Fast Models.

It contains the following sections:

3.1 About the processor components

These are *Code Translation* (CT) components, which match functional behavior with what you can observe from software. However, they sacrifice accuracy in timing to achieve fast simulation speeds.

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

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

- Timing is approximate.
- Selected processor models implement caches, including smartcache, although all processor models implement cache control registers.
- They do not implement write buffers.
- They do not implement micro-architectural features, such as MicroTLB or branch cache.
- Device-accurate modeling of multiple TLBs is off by default.
- They use 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 and Cortex-A5 models have two memory access ports, but use only one.
- Processors that support Jazelle only have trivial implementations.
- Cluster models do not simulate all cores at the same time: they run through each core in turn, executing a number of instructions on each core. There can be a bias in the order in which cores run after a restart (for example, core 0 always runs first), so the simulation might hit breakpoints on the favored core more often.

All model *Performance Monitor Unit* (PMU) registers allow software to program the PMU without aborting. Models implement the cycle count (CCNT) but not the programmable timers. Software can program registers of the PMU but the count value returned is not valid.

For information about the hardware, see the technical reference manuals.

3.2 Cortex-A processor components

This section describes the Cortex-A processor components.

This section contains the following subsections:

3.2.1 ARMCortexA72xnCT component

This section describes the ARMCortexA72x*n*CT component.

ARMCortexA72xnCT - about

This C++ component is a model of r0p0 of an ARMv8-A Cortex-A72 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA72xnCT - ports

This section describes the ports.

Table 3-1 ARMCortexA72xnCT ports

Name	Protocol	Type	Description
`clk_in`	ClockSignal	Slave	The clock signal connected to the `clk_in` port is used to determine the rate at which the core executes instructions.
`pvbus_m0`	PVBus	Master	The core will generate bus requests on this port.
`acp_s`	PVBus	Slave	AXI ACP slave port.
`virtio_s`	PVBus	Slave	The virtio coherent port, hooks directly into the L2 system and becomes coherent (assuming attributes are set correctly).
`reset[4]`	Signal	Slave	Raising this signal will put the core into reset mode.
`cpuporeset[4]`	Signal	Slave	Raising this signal will put the core into reset mode.
`presetdbg`	Signal	Slave	Raising this signal will put the core into reset mode.
`l2reset`	Signal	Slave	Raising this signal will put the core into reset mode.
`romaddr`	Value_64	Slave	Debug ROM base address.

Table 3-1 ARMCortexA72xnCT ports (continued)

Name	Protocol	Type	Description
romaddrv	Signal	Slave	Debug ROM base address valid.
event	Signal	Peer	This peer port of event input (and output) is for wakeup from WFE.
standbywfe[4]	Signal	Master	This signal indicates if a core is in WFE state.
standbywfi[4]	Signal	Master	This signal indicates if a core is in WFI state.
standbywfil2	Signal	Master	This signal indicates if a core is in WFI state.
dbgnopwrdwn[4]	Signal	Master	This signal relates to core power down.
dbgpwrupreq[4]	Signal	Master	This signal relates to core power down.
cfgsdisable	Signal	Slave	This signal disables write access to some secure Interrupt Controller registers.
smpen[4]	Signal	Master	This signals AMP or SMP mode for each core.
irq[4]	Signal	Slave	This signal drives the core's interrupt handling.
fiq[4]	Signal	Slave	This signal drives the core's fast-interrupt handling.
virq[4]	Signal	Slave	This signal is a virtualized version of the previous one.
vfiq[4]	Signal	Slave	This signal is a virtualized version of the previous one.
sei[4]	Signal	Slave	Per core System Error physical pins.
rei[4]	Signal	Slave	Per core System Error physical pins.
vsei[4]	Signal	Slave	Per core System Error physical pins.
pmuirq[4]	Signal	Master	Interrupt signal from performance monitoring unit.
commirq[4]	Signal	Master	Interrupt signal from debug communications channel.
commrx[4]	Signal	Master	Receive portion of Data Transfer Register full.
commtx[4]	Signal	Master	Transmit portion of Data Transfer Register empty.
vcpumntirq[4]	Signal	Master	Interrupt signal for virtual CPU maintenance IRQ.
clusterid	Value	Slave	The port reads the value in CPU ID register field, bits[11:8] of the MPIDR.
cp15sdisable[4]	Signal	Slave	This signal disables write access to some system control processor registers.
cfgte[4]	Signal	Slave	This signal provides default exception handling state.
ticks[4]	InstructionCount	Master	This port should be connected to one of the two ticks ports on a 'visualisation' component, in order to display a running instruction count.

Table 3-1 ARMCortexA72xnCT ports (continued)

Name	Protocol	Type	Description
vinithi[4]	Signal	Slave	This signal controls of the location of the exception vectors at reset.
cfgend[4]	Signal	Slave	This signal is for EE bit initialization.
periphbase	Value_64	Slave	This port sets the base address of private peripheral region.
rvbaraddr[4]	Value_64	Slave	Reset vector base address.
aa64naa32[4]	Signal	Slave	Register width after reset.
cryptodisable[4]	Signal	Slave	Disable cryptography extensions after reset.
cntvalueb	CounterInterface	Slave	Interface to SoC level counter module.
dbgen[4]	Signal	Slave	External debug interface.
spiden[4]	Signal	Slave	External debug interface.
niden[4]	Signal	Slave	External debug interface.
spniden[4]	Signal	Slave	External debug interface.
dbgack[4]	Signal	Master	External debug interface.
edbgrq[4]	Signal	Slave	External debug interface.
dev_debug_s	PVBus	Slave	External debug interface.
memorymapped_debug_s	PVBus	Slave	External debug interface.
cti[4]	v8EmbeddedCrossTrigger_controlprotocol	Master	Cross trigger matrix port.
ctidbgirq[4]	Signal	Master	Cross trigger matrix port.
CNTPNSIRQ[4]	Signal	Master	The per-EL counter signals master port.
CNTPSIRQ[4]	Signal	Master	The per-EL counter signals master port.
CNTVIRQ[4]	Signal	Master	The per-EL counter signals master port.
CNTHPIRQ[4]	Signal	Master	The per-EL counter signals master port.
broadcastinner	Signal	Slave	ACE defined pins.
broadcastouter	Signal	Slave	ACE defined pins.
broadcastcachemaint	Signal	Slave	ACE defined pins.
gicv3_redistributor_s[1]	GICv3Comms	Slave	GICv3 AXI-stream port.

ARMCortexA72xnCT - parameters

This section describes the parameters.

ARMCortexA72xnCT cluster parameters

Table 3-2 ARMCortexA72xnCT cluster parameters

Name	Type	Allowed values	Default value	Description
CLUSTER_ID	uint32_t	0x0-0xFFFF	0x0	-
dcache-state_modelled	bool	true, false	false	-

Table 3-2 ARMCortexA72xnCT cluster parameters (continued)

Name	Type	Allowed values	Default value	Description
icache-state_modelled	bool	true, false	false	-
PERIPHBASE	uint64_t	0x0-0xFFFFFFFFFFF	0x13080000	-
l2cache-size	uint32_t	0x80000-0x200000	0x80000	L2 cache size in bytes.
GICDISABLE	bool	true, false	true	-
is_skyros	bool	true, false	false	Cosmetic change that changes reset value of L2ACTLR register.

ARMCortexA72xnCT cache latency cluster parameters

Note

- These latencies are only effective when you enable cache-state modeling.
- Timing annotation for transactions downstream of the cache models propagates through the cache models.

Table 3-3 ARMCortexA72xnCT cache latency cluster parameters

Parameter	Type	Allowed values	Default value	Description
dcache-read_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-snoop_data_transfer_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for received snoop accesses that perfom a data transfer given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-write_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for write accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
icache-read_latency	uint64_t	-	0x0	L1 I-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when icache-state_modelled=true.
l2cache-read_latency	uint64_t	-	0x0	L2 cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_data_transfer_latency	uint64_t	-	0x0	L2 cache timing annotation latency for received snoop accesses that perfom a data transfer given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_issue_latency	uint64_t	-	0x0	L2 cache timing annotation latency for snoop accesses issued by this cache in total ticks. For use when dcache-state_modelled=true.
l2cache-write_latency	uint64_t	-	0x0	L2 cache timing annotation latency for write accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.

ARMCortexA72xnCT core parameters

Table 3-4 ARMCortexA72xnCT core parameters

Name	Type	Allowed values	Default value	Description
CFGEND	bool	true, false	false	-
CP15SDISABLE	bool	true, false	false	-
CFGTE	bool	true, false	false	-
VINITHI	bool	true, false	false	-
vfp-enable_at_reset	bool	true, false	false	-
semihosting-enable	bool	true, false	true	-
semihosting-ARM_SVC	uint32_t	0x0-0xFFFFFF	0x123456	-
semihosting-Thumb_SVC	uint32_t	0x0-0xFF	0xAB	-
semihosting-cmd_line	string	-	""	-
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0x0	-
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	-
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10000000	-
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	-
semihosting-cwd	string	-	""	-
RVBARADDR	uint64_t	0x0-0xFFFFFFFFFF	0x0	-
min_sync_level	uint32_t	0x0-0x3	0x0	Runtime parameter. Force minimum syncLevel (0=off=default, 1=syncState, 2=postInsnIO, 3=postInsnAll).
cpi_mul	uint32_t	0x1-0x7FFFFFFF	0x1	Runtime parameter. Multiplier for calculating CPI (Cycles Per Instruction).
cpi_div	uint32_t	0x1-0x7FFFFFFF	0x1	Runtime parameter. Divider for calculating CPI (Cycles Per Instruction).
AA64nAA32	bool	true, false	true	-
CRYPTODISABLE	bool	true, false	false	-
max_code_cache	uint32_t	0x0-0xFFFFFFFFFFFFFFF	67108864	Maximum number of bytes for caching code translations.
DBGROMADDRV	bool	true, false	true	-
DBGROMADDR	uint64_t	0x0-0xfffffffffff	0x22000000	-

ARMCortexA72xnCT TLB latency core parameters

Note

- The walk_cache_latency parameter is only available for AEMv8 clusters and only effective when you configure the walk cache with a non-zero size.
- Timing annotation for transactions downstream of the TLB model propagates through the TLB model.

Table 3-5 ARMCortexA72xnCT TLB latency core parameters

Parameter	Type	Allowed values	Default value	Description
ptw_latency	uint64_t	-	0x0	Page table walker latency for *Timing Annotation* (TA), in simulation ticks.
tlb_latency	uint64_t	-	0x0	TLB latency for TA, in simulation ticks.
walk_cache_latency	uint64_t	-	0x0	Walk cache latency for TA, in simulation ticks.

3.2.2 ARMAEMv8AMPCT component

This section describes the ARMv8-A AEMv8-A *Architecture Envelope Model* (AEM) component.

ARMAEMv8AMPCT - parameters

This section describes the parameters.

ARMAEMv8AMPCT cache parameters

Table 3-6 ARMAEMv8AMPCT cache parameters

Parameter	Allowed values	Default value	Description
BPIMVA_causes_translation_lookup	true, false	false	Performs a translation when BPIMVA is executed, that can cause both a translation table walk, or a translation fault.
cache-log2linelen	0x4-0x8	0x6	Log_2(cache-line length, in bytes).
cache_maintenance_hits_watchpoints	true, false	false	Enable AArch32 cache maintenance by DCIMVAC to trigger watchpoints.[b]
CTR-L1Ip-override	0-3	0	If nonzero, overrides the L1Ip bits in the CTR/CTR_EL0 system register. This parameter does not change the behavior of the cache, only what is present in the CTR register.
dcache-size	0x4000-0x100000	0x8000	L1 D-cache size, in bytes.
dcache-ways	0x1-0x40	0x2	The number of L1 D-cache ways.[c]*#rob1395750478883/CHDHAGAF*
DCZID-log2-block-size	0x0-0x9	0x8	Log_2(block size) cleared by DC ZVA instruction.[d]
icache-size	0x4000-0x100000	0x8000	L1 I-cache size, in bytes.
icache-ways	0x1-0x40	0x2	The number of L1 I-cache ways.*#rob1395750478883/CHDHAGAF*
l2cache-size	0x0-0x1000000	0x80000	L2 cache size, in bytes.

[b] UNPREDICTABLE.
[c] Sets are implicit from size.
[d] As read from DCZID_EL0.

Table 3-6 ARMAEMv8AMPCT cache parameters (continued)

Parameter	Allowed values	Default value	Description
l2cache-ways	0x1-0x40	0x10	The number of L2 cache ways.#rob1395750478883/CHDHAGAF
memory.l2_cache.is_inner_cacheable	true, false	true	L2 cache is inner cacheable, not outer cacheable.
memory.l2_cache.is_inner_shareable	true, false	true	L2 cache is inner shareable, not outer shareable.
treat-dcache-invalidate-as-clean-invalidate	true, false	false	Treats data cache invalidate operations as clean and invalidate.

ARMAEMv8AMPCT cluster parameters

Note

The cluster identifier range is 0-1.

Table 3-7 ARMAEMv8AMPCT cluster parameters

Parameter	Allowed values	Default value	Description
abort_execution_from_device_memory	true, false	false	Abort on execution from device memory.
ADFSR-AIFSR-implemented	true, false	false	Implements ADFSR and AIFSR.
advsimd_overread	true, false	false	AdvSIMD element load operations access all bytes of a 16-byte aligned window, even in Device memory.
apsr_read_restrict	true, false	false	At EL0, UNKNOWN bits of APSR are RAZ.
auxilliary_feature_register0	0x0-0xFFFFFFFF	0x0	Value for *Auxiliary Feature Register 0* (ID_AFR0).
branch-predictor-clear-policy	0x0-0x4	0x2	Sets the branch prediction policy as defined for MMFR1[31:28]. This parameter does not change the behavior of the branch predictor, only what is reported in MMFR1.BPred.

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
branch-predictor-supported-ops	0x0-0x2	0x1	Sets the branch prediction policy as defined for MMFR3[11:8]. This parameter does not change the behavior of the branch predictor, only what is reported in MMFR3.BPMaint.
clear_reg_top_eret	0x0-0x2	0x1	Clears or preserves the top 32 bits of general purpose registers on exception return. 0, preserve. 1, clear to zero. 2, random choice of preserve or clear to zero.
cpacr_trcdis_behaviour	0-2	2	If there is an ETM CP14 interface, how CPACR.TRCDIS/ NSACR.NSTRCDIS behaves. 0, RAZ/WI. 1, reserved. 2, implemented.
CTIPIDR	0x0-0xFFFFFFFFFF	0x0	If nonzero, override the CTI Peripheral Identification Register.
DBGBCR_BT_applies_RES0_before_valid_check	true, false	true	If set, RES0 behavior is applied to DBGBCR(_EL1).BT before checking for reserved values for this field.
dbgitr_buffer_size	0x0-0xFFFFFFFF	0x0	Number of instructions that can be buffered before EDSCR.ITE is cleared.
DBGPIDR	0x0-0xFFFFFFFFFF	0x0	If zero, build a value for the *DeBuG Peripheral Identification Register*, DBGPIDR. If nonzero, override DBGPIDR with this value.
debug_rom_is_flat	true, false	false	If true, present a debug ROM table recommended by ARMv8 Debug Architecture. Otherwise, use nested ROM tables.

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
delay_serror	0x0-0xFFFFFFFF	0x0	Minimum propagation delay of the *System Error* (**SERR**) signal into the cluster. Accurate in low-latency mode, `-C cpu.scheduler_mode=1`, but otherwise, any delay might be larger.
el0_el1_only_non_secure	true, false	false	Controls the security state of EL0 and EL1 if neither EL2 nor EL3 are implemented. `true` means non-secure.
enable_tlb_contig_check	true, false	true	Check consistency of TLB entries in regions with the Contiguous Bit set.
exception_return_treats_SPSR_J_as_0	true, false	false	Exception return treats SPSR.J as RAZ/WI.
exercise_stxr_fail	true, false	false	When `true`, returns a pseudorandom majority of *Store Exclusive Register* (STXR) instructions as Failed.
ext_abort_device_read_is_sync	true, false	true	Synchronous reporting of device read external aborts.
ext_abort_device_write_is_sync	true, false	false	Synchronous reporting of device write external aborts.
ext_abort_fill_data	-	0xFDFDFDFCFCFDFDFD	Returned data, if external aborts are asynchronous.
ext_abort_normal_cacheable_read_is_sync	true, false	true	Synchronous reporting of normal cacheable-read external aborts.
ext_abort_normal_cacheable_write_is_sync	true, false	false	Synchronous reporting of normal cacheable write external aborts.
ext_abort_normal_noncacheable_read_is_sync	true, false	true	Synchronous reporting of normal non-cacheable-read external aborts.
ext_abort_normal_noncacheable_write_is_sync	true, false	false	Synchronous reporting of normal non-cacheable write external aborts.
ext_abort_prefetch_is_sync	true, false	true	Synchronous reporting of instruction-fetch external aborts.

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
ext_abort_so_read_is_sync	true, false	true	Synchronous reporting of strongly ordered read external aborts.
ext_abort_so_write_is_sync	true, false	true	Synchronous reporting of strongly ordered write external aborts.
ext_abort_ttw_cacheable_read_is_sync	true, false	true	Synchronous reporting of TTW cacheable read external aborts.
ext_abort_ttw_noncacheable_read_is_sync	true, false	true	Synchronous reporting of TTW non-cacheable read external aborts.
exception_catch_type	0-2	0	Type of exception catch. 0, exception trapping. 1, non-exception trapping, higher priority than step. 2, non-exception-trapping, lower priority than step.
force_align_pc	true, false	false	UNPREDICTABLE branch to non-word-aligned address in ARM state is forced to be aligned.
fpcr_short_vector_raz	true, false	false	FPSCR and FPCR fields LEN and STRIDE are hardwired to 0.
has_aarch32_dbgdidr_etc	true, false	true	DBGDIDR, DBGDRAR, DBGDSAR exist even if EL1 does not implement AArch32.
has_exception_trapping_form_of_vector_catch	true, false	true	Implement the exception trapping form of a vector catch debug event.
has_pmu	true, false	true	Implement the optional Performance Monitors Extension.
has_tlb_conflict_abort	true, false	false	Inconsistent TLB content generates aborts.
has_16k_granule	true, false	false	Implements a 16k LPAE translation granule.
has_4k_granule	true, false	true	Implements a 4k LPAE translation granule.
has_64k_granule	true, false	true	Implements a 64k LPAE translation granule.
has_16bit_asids	true, false	true	Enables 16-bit *Address Space IDentifiers* (ASIDs).

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
has_delayed_sysreg	true, false	false	Delays the functional effect of system register writes until ISB or implicit barrier.
has_el2	true, false	true	Enables EL2.
has_el3	true, false	true	Enables EL3.
has_large_system_ext	true, false	false	Implement the ARMv8-A Large System Extensions. This feature is at Alpha release quality.
has_hardware_translation_table_update	0x0-0x2	0x2	Perform updates to page table access flag and dirty bit if the Large System Extensions are enabled. 0, no hardware updates. 1, support bit access updates. 2, support bit access updates, and dirty bit mechanism.
has_writebuffer	true, false	false	Implements write access buffering before L1 cache. This parameter might affect ext_abort behavior.
hcptr_tta_behaviour	0-2	2	If there is no ETM CP14 interface, how HCPTR.TTA behaves. 0, RAZ/WI. 1, RAO/WI. 2, stateful.
hcr_swio_res1	true, false	false	Determines whether HCR.SWIO and/or HCR_EL2.SWIO are RES1.
hsr_uncond_cc	true, false	false	Condition codes reported in HSR as AL if it passes.
icache-log2linelen	0x4-0x8	0x0	If nonzero, log2 of the instruction cache line length in bytes (valid values in the range 4 to 8). Otherwise use cache - log2linelen.
instruction_tlb_size	0x0-0xFFFFFFFF	0x0	Number of stage 1 and stage 2 ITLB entries. 0x0 for unified ITLB + DTLB.
is_uniprocessor	true, false	false	true for a single core implementation. When true, NUM_CORES must be 0x1.

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
itd_conditional_instructions_are_32bit	true, false	false	With ITD set, an IT instruction plus a T16 instruction are considered to be a single 32-bit conditional instruction.
max_32bit_el	-0x1-0x3	0x3	Maximum exception level supporting AArch32 modes. -0x1 means no support.
MIDR	0x0-0xFFFFFFFF	0x410FD0F0	Value for *Main ID Register* (MIDR).
mixed_endian	0x0-0x2	0x1	Enables the core to change the endianness at runtime. 0x0, not supported. 0x1, supported at all exception levels. 0x2, supported at EL0 only.
mvbar_reset_is_rvbar	true, false	true	If true then the reset value of MVBAR is RVBAR. If false then the rest value is unknown.
NUM_CORES	0x1-0x4	0x4	Number of cores implemented.
PA_SIZE	0x0-0x30	0x28	Physical address size, in bits.
pmu-num_counters	0x0-0x1F	0x8	Number of PMU counters implemented.
PMUPIDR	0x0-0xFFFFFFFFF	0x0	If nonzero, override the PMU Peripheral Identification Register.
register_reset_data	-	0x0	Fills data for register bits when they become UNKNOWN at reset.
report_iside_cmo_ifsr	true, false	true	Fault information for an iside cache maintenance operation is reported in the IFSR.
scheduler_mode	0x0-0x2	0x0	Controls instruction interleaving. 0, default long quantum. 1, low latency mode, short quantum, and signal checking. 2, lock-breaking mode, long quantum with additional context switches near load-exclusive instructions.

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
scr_nET_writeable	true, false	false	Whether SCR.nET is writable. Writing to it is purely cosmetic. Note: it does not implement nET behavior.
scramble_unknowns_at_reset	true, false	true	Fills in UNKNOWN bits in registers at reset with register_reset_data.
spsr_el3_is_mapped_to_spsr_mon	true, false	true	Defines whether SPSR_EL3 is mapped to AArch32 register SPSR_mon.
stage12_tlb_size	0x1-0xFFFFFFFF	0x80	Number of stage 1 and stage 2 TLB entries.
stage1_tlb_size	0x0-0xFFFFFFFF	0x0	Number of stage 1 TLB entries.
stage2_tlb_size	0x0-0xFFFFFFFF	0x0	Number of stage 2 TLB entries.
stage1_walkcache_size	0x0-0xFFFFFFFF	0x0	Number of stage 1 TLB walk-cache entries.
stage2_walkcache_size	0x0-0xFFFFFFFF	0x0	Number of stage 2 TLB walk-cache entries.
treat_wfi_wfe_as_nop	true, false	false	If set, never goes into wait state for WFI or WFE instructions.
take_ccfail_undef	true, false	true	In AArch32, take an Undefined Instruction exception even if the instruction fails its condition-codes check.
tidcp_traps_el0_undef_imp_def	true, false	true	The TIDCP bit traps, in EL0, undefined IMPLEMENTATION DEFINED instructions accessing coprocessor registers.
treat_pld_as_nop	true, false	false	If set, treat PLD as NOP.
treat_pli_as_nop	true, false	false	If set, treat PLI as NOP.

Table 3-7 ARMAEMv8AMPCT cluster parameters (continued)

Parameter	Allowed values	Default value	Description
`unpredictable_exclusive_abort_memtype`	`0x0-0x2`	`0x0`	MMU abort if exclusive access is not supported. `0x0`, none. Exclusives are permitted in all memory. `0x1`, exclusives abort in Device memory. `0x2`, exclusives abort in any memory type other than WB inner cacheable.
`unpredictable_hvc_behaviour`	`0x0-0x1`	`0x0`	Defines `HVC` UNPREDICTABLE behavior in HYP mode, EL2, when the SCR.HCE bit is clear. `0x0`, Undefined Instruction. `0x1`, `NOP` instruction.
`unpredictable_smc_behaviour`	`0x0-0x1`	`0x0`	Defines `SMC` UNPREDICTABLE behavior in Secure mode, EL3, when the SCR.SCD bit is clear. `0x0`, Undefined Instruction. `0x1`, `NOP` instruction.
`use_tlb_contig_hint`	`true`, `false`	`false`	Page table entries with the Contiguous Bit set generate large TLB entries.
`warn_unpredictable_in_v7`	`true`, `false`	`false`	Warns of UNPREDICTABLE behavior in ARMv7.
`watchpoint-log2secondary_restriction`	`0x0-0x3F`	`0x0`	Log_2(secondary restriction of FAR/EDWAR) on watchpoint hit for load and store operations.

ARMAEMv8AMPCT cache latency cluster parameters

Note

- These latencies are only effective when you enable cache-state modeling.
- Timing annotation for transactions downstream of the cache models propagates through the cache models.

Table 3-8 ARMAEMv8AMPCT cache latency cluster parameters

Parameter	Type	Allowed values	Default value	Description
dcache-read_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-snoop_data_transfer_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for received snoop accesses that perform a data transfer. It is given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-write_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for write accesses. It is given in ticks per byte accessed. For use when dcache-state_modelled=true.
icache-read_latency	uint64_t	-	0x0	L1 I-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when icache-state_modelled=true.
l2cache-read_latency	uint64_t	-	0x0	L2 cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_data_transfer_latency	uint64_t	-	0x0	L2 cache timing annotation latency for received snoop accesses that perform a data transfer. It is given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_issue_latency	uint64_t	-	0x0	L2 cache timing annotation latency for snoop accesses issued by this cache in total ticks. For use when dcache-state_modelled=true.
l2cache-write_latency	uint64_t	-	0x0	L2 cache timing annotation latency for write accesses. It is given in ticks per byte accessed. For use when dcache-state_modelled=true.

ARMAEMv8AMPCT core parameters

The models use the parameters for cores in sequence, from cpu0 onwards. If there are fewer cores than the maximum number, models ignore parameters for uninstantiated cores.

Note

- The cluster identifier range is 0-1. The core identifier range is 0-3.
- The model needs the separate cryptography plug-in to enable the cryptographic instructions.

Table 3-9 ARMAEMv8AMPCT core parameters

Parameter	Allowed values	Default value	Description
ase-present	true, false	true	Enables NEON™.
CFGEND	true, false	false	Uses big-endian order.
clock_divider	0x0-0xFFFFFFFFFF	0x1	Clock divider ratio for asymmetric MP clocking.
CONFIG64	true, false	true	Enables AArch64.

Table 3-9 ARMAEMv8AMPCT core parameters (continued)

Parameter	Allowed values	Default value	Description
CP15SDISABLE	true, false	false	Disables access to some CP15 registers in AArch32.
cpi_div	-	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	-	1	Multiplier for calculating *Cycles Per Instruction* (CPI).
crypto_aes	0x0-0x2	0x2	AES hash level. 0x0, AES-128. 0x1, AES-192. 0x2, AES-256.
crypto_sha1	0x0-0x1	0x1	Enable SHA1.
crypto_sha256	0x0-0x1	0x1	Enable SHA256.
CRYPTODISPOSABLE	true, false	false	Disable cryptographic features.
cti-intack_mask	0x0-0xFF	0x1	Set bits mean that the corresponding triggers require software acknowledgment through CTIINTACK. One bit per trigger.
cti-number_of_claim_bits	0-31	0	Number of implemented bits in CTICLAIMSET.
cti-number_of_triggers	0x0-0x8	0x8	Number of CTI event triggers.
enable_crc32	true, false	false	Enables the optional CRC32 instruction.
etm-present	true, false	true	Enables *Embedded Trace Macrocell* (ETM).
force-fpsid	true, false	false	Overrides the FPSID value.
force-fpsid-value	0x0-0xFFFFFFFF	0x0	Value for the overridden FPSID.
has_hcptr_tase	true, false	true	If false, HCPTR.TASE is RES0.
max_code_cache	-	-	Maximum cache size for code translations, in bytes.
min_sync_level	0x0-0x3	0x0	Minimum CADI syncLevel. 0x0, Off. 0x1, SyncState. 0x2, PostInsnIO. 0x3, PostInsnAll.
MPIDR-override	0x0-0xFFFFFFFFF	0x0	Overrides the MPIDR value. If this parameter is nonzero, it overrides the cluster and core ID bits in MPIDR.
number-of-breakpoints	0x2-0x10	0x10	Number of breakpoints.
number-of-context-breakpoints	0x0-0x10	0x10	Number of context-aware breakpoints.
number-of-watchpoints	0x2-0x10	0x10	Number of watchpoints.
POWERCTLI	0x0-0xFFFFFFFF	0x0	Default power control state.
RVBAR	0x0-0xFFFFFFFFFFFC	0x0	Resets the Vector Base Address when resetting into AArch64.
semihosting-ARM_SVC	0x0-0xFFFFFFFF	0x123456	A32 SVC number for semihosted calls
semihosting-cmd_line	-	-	Program name and arguments, argc, argv, for target programs using the semihosted C library.
semihosting-cwd	-	-	Virtual address of CWD.

Table 3-9 ARMAEMv8AMPCT core parameters (continued)

Parameter	Allowed values	Default value	Description
semihosting-enable	true, false	true	Enable semihosting of SVC instructions.[e]
semihosting-heap_base	-	0x00000000	Virtual address of heap base.
semihosting-heap_limit	-	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	-	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	-	0x0F000000	Virtual address of stack limit.
semihosting-Thumb_SVC	0x0-0xFFFFFFFF	0xAB	T32 SVC number for semihosted calls.
SMPnAMP	true, false	true	This core is in the inner shared domain, and uses its cache coherency protocol. This parameter is a model only parameter, not a synthesize option or a configuration port. In hardware, it is a design choice.
TEINIT	true, false	false	Controls the initial state of SCTLR.TE in AArch32. When set, causes AArch32 exceptions, including reset, to be taken in T32 mode.
unpredictable_non-contigous_BAS	true, false	true	Treat non-contiguous BAS field in the watchpoint control register as all ones.
unpredictable_WPMASKANDBAS	0-3	1	Constrained unpredictable handling of watchpoints when mask and BAS fields specified. 0, IGNOREMASK. 1, IGNOREBAS. 2, REPEATBAS8. 3, REPEATBAS.
vfp-enable_at_reset	true, false	false	Enables coprocessor access and VFP at reset. This behavior is model-specific. It has no hardware equivalent.
vfp-present	true, false	true	Enables floating-point arithmetic.
vfp-traps	true, false	true	Enables hardware trapping of VFP exceptions for VFPv4U.
VINITHI	true, false	false	Enables high vectors. The base address is 0xFFFF0000.

ARMAEMv8AMPCT TLB latency core parameters

Note

- The walk_cache_latency parameter is only available for AEMv8 clusters and only effective when you configure the walk cache with a non-zero size.
- Timing annotation for transactions downstream of the TLB model propagates through the TLB model.

[e] Semihosting is a method of running your target software on the model to communicate with the host environment. The AEM models permit the target C library to access the I/O facilities of the host computer, such as the filesystem, keyboard input, and clock.

Table 3-10 ARMAEMv8AMPCT TLB latency core parameters

Parameter	Type	Allowed values	Default value	Description
ptw_latency	uint64_t	-	0x0	Page table walker latency for *Timing Annotation* (TA), in simulation ticks.
tlb_latency	uint64_t	-	0x0	TLB latency for TA, in simulation ticks.
walk_cache_latency	uint64_t	-	0x0	Walk cache latency for TA, in simulation ticks.

ARMAEMv8AMPCT GIC parameters

Table 3-11 ARMAEMv8AMPCT GIC parameters

Parameter	Allowed values	Default value	Description
dic-spi_count	0x0-0xE0	0x40	Number of *Shared Peripheral Interrupts* (SPIs) supported.
gic.GICC-offset	0x0-0xFF000	0x0	Offset from PERIPHBASE for GICC registers.
gic.GICD-offset	0x0-0xFF000	0x40000	Offset from PERIPHBASE for GICD registers. Ignored when the GICv3 CPU interface is enabled, because the distributor is then external to the cluster.
gic.GICH-offset	0x0-0xFF000	0x10000	Offset from PERIPHBASE for GICH registers.
gic.GICH-other-CPU-offset	0x0-0xFF000	0x40000	From PERIPHBASE for GICH registers for accessing other cores in the cluster. Set to 0 to disable.
gic.GICV-offset	0x0-0x100000	0x40000	Size of registers that are based at PERIPHBASE that are considered to be owned by the GIC. Any accesses in the range PERIPHBASE-PERIPHBASE+gic. (PERIPH - size - 1) that do not match GIC registers are treated as RAZ/WI.
gicv3.A3-affinity-supported	true, false	false	Determines whether a nonzero value for affinity at level 3 is supported.
gicv3.BPR-min	0x0-0x3	0x2	Minimum value for GICC_BPR. The Non-secure version is this value + 1.
gicv3.cpuintf-mmap-access-level	0x0-0x2	0x0	Defines memory mapped access level. 0x0, GICC, GICH, and GICV registers. 0x1, only GICV registers. 0x2, not supported.
gicv3.EOI-check-CPUID	true, false	false	Check CPU ID specified for accesses to EOI registers instead of ending the highest priority active interrupt.
gicv3.EOI-check-ID	true, false	false	Check the Interrupt ID specified for accesses to EOI registers instead of ending the highest priority active interrupt.

Table 3-11 ARMAEMv8AMPCT GIC parameters (continued)

Parameter	Allowed values	Default value	Description
gicv3.EOI-deactivate-any-interrupt	true, false	false	Permit an EOI to deactivate interrupts that are not the highest priority active interrupt. EOI-ignore-out-of-order must be false otherwise this parameter is ignored.
gicv3.EOI-ignore-out-of-order	true, false	true	Ignore EOI writes that cannot end the highest priority active interrupt.
gicv3.HCR-VARE-is-raz	true, false	false	Virtual ARE bit in ICH_HCR_EL2 register is RAZ/WI.
gicv3.ignore-DIR-write-when-EOImode-not-set	true, false	true	Ignore UNPREDICTABLE access to the GICC_DIR register.
gicv3.IIDR_base	0x0-0xFFFFFFFF	0x43B	Base value for calculating the GICC_IIDR value.
gicv3.LR-count	0x0-0x40	0x10	Number of implemented list registers.
gicv3.physical-ID-bits	0x10-0x18	0x10	Number of physical ID bits implemented.
gicv3.PMHE-RAO-WI	true, false	false	ICC_CTLR_EL*.PHME is read as one, write ignored.
gicv3.priority-bits	0x4-0x8	0x5	Number of priority bits implemented.
gicv3.SRE-enable-action-on-mmap	0x0-0x2	0x0	Defines mmap access. 0x0, SRE one allows mmap access. 0x1, SRE one disables mmap access. 0x2, SRE one makes map access RAZ/WI.
gicv3.SRE-EL3-set-once	true, false	false	Restrict SRE EL3 to be set only once.
gicv3.STATUSR-implemented	true, false	true	If the GICv3 core interface is enabled, enable STATUS registers.
gicv3.VBPR-min	0x0-0x3	0x2	Minimum value for the GICV_BPR register. The Non-secure version is this value + 1.
gicv3.virtual-ID-bits	0x10-0x18	0x10	Number of virtual ID bits implemented.
gicv3.virtual-priority-bits	0x4-0x8	0x5	Number of virtual priority bits implemented.
non_secure_vgic_alias_when_ns_only	0x0-0xFFFFFFFFFFFF	0x0	If there is no EL3 and no Secure state, the VGIC has a Secure alias. If this parameter is nonzero, the model forms a Non-secure alias from its value for the VGIC, aligned to 32KB.

3.2.3 ARMCortexA57xnCT component

This section describes the ARMCortexA57x*n*CT component.

ARMCortexA57xnCT - about

This component is a model of r0p0 of an ARMv8-A Cortex-A57 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA57xnCT - ports

This section describes the ports.

Table 3-12 ARMCortexA57xnCT ports

Name	Protocol	Type	Description
aa64naa32[0-3]	Signal	Slave	Register width state after reset.
acp_s	PVBus	Slave	Bus slave that the processor receives coherency transactions on. It is a *Programmer's View* (PV) of the *Advanced Extensible Interface* (AXI) *Accelerator Coherency Port* (ACP) slave port.
broadcastinner	Signal	Slave	Enable broadcasting of inner shareable transactions.
broadcastouter	Signal	Slave	Enable broadcasting of outer shareable transactions.
broadcastcachemaint	Signal	Slave	Enable broadcasting of cache maintenance operations to downstream caches.
cfgend[0-3]	Signal	Slave	Configure endianness at reset: set the value of the EE bits in the CP15 SCTLR_EL3 and SCTR_S registers.
cfgte[0-3]	Signal	Slave	Initialize to take exceptions in T32 state after reset.
clk_in	ClockSignal	Slave	Processor clock input, for determining the rate of instruction execution relative to other system components.
clusterid	Value	Slave	Master ID, bits[23:8] of the MPIDR using bits[15:0] of the CLUSTERID value.
CNTHPIRQ[0-3]	Signal	Master	Hypervisor physical timer interrupt.
CNTPNSIRQ[0-3]	Signal	Master	Secure physical timer interrupt.
CNTPSIRQ[0-3]	Signal	Master	Non-secure physical timer interrupt.
cntvalueb	CounterInterface	Slave	Interface to a platform level MemoryMappedCounter module.
CNTVIRQ[0-3]	Signal	Master	Virtual timer interrupt.
commirq[0-3]	Signal	Master	Communications channel receive or transmit interrupt request.
commrx[0-3]	Signal	Master	Communications channel receive.
commtx[0-3]	Signal	Master	Communications channel transmit.
cp15sdisable[0-3]	Signal	Slave	Disables write access to some Secure CP15 registers.

Table 3-12 ARMCortexA57xnCT ports (continued)

Name	Protocol	Type	Description
cpuporeset[0-3]	Signal	Slave	Power on reset. Initializes all the processor logic, including debug logic.
cryptodisable[0-3]	Signal	Slave	Cryptography engine disable.[f]
ctdbgirq[0-3]	Signal	Master	*Cross Trigger Interface* (CTI) interrupt trigger output.
cti[0-3]	v8EmbeddedCrossTrigger_controlprotocol	Master	Interface to the CTI to a platform level *Cross Trigger Matrix* (CTM).
dbgack[0-3]	Signal	Master	Debug acknowledge.
dbgen[0-3]	Signal	Slave	Invasive debug enable.
dbgromaddr	Value_64	Slave	Specify bits[43:12] of the top-level ROM table physical address.
dbgromaddrv	Signal	Slave	Valid signal for dbgromaddr.
dev_debug_s	PVBus	Slave	Debug *Advanced Peripheral Bus* (APB) as exposed to external debug agents.[g]
edbgrq[0-3]	Signal	Slave	External debug request.
event	Signal	Peer	Event input and output for wakeup from WFE. This port amalgamates the hardware EVENTI and EVENT0 signals.
fiq[0-3]	Signal	Slave	Processor FIQ signal input.
gicv3_redistributor_s[0-3]	GICv3Comms	Slave	Interface to a platform-level GICv3Distributor component.
irq[0-3]	Signal	Slave	Processor IRQ signal input.
l2reset	Signal	Slave	Reset the shared L2 memory system controller.
memory_mapped_debug_s	PVBus	Slave	Debug APB as exposed to other system agents.[g]
niden[0-3]	Signal	Slave	Non-invasive debug enable.
periphbase	Value_64	Slave	Base address of peripheral memory space, the base address for the GIC CPU Interface registers, which are sampled into the *Configuration Base Address Register* (CBAR) at reset.
pmuirq[0-3]	Signal	Master	*Performance Monitoring Unit* interrupt signal.
presetdbg	Signal	Slave	Initialize the shared debug APB, *Cross Trigger Interface* (CTI), and *Cross Trigger Matrix* (CTM) logic.

[f] ARMv8 Cryptography Extensions require a separate package, which is subject to export license conditions. Contact ARM for details.
[g] The system designer decides whether a debug APB ties the external debug view with other system views. In the model, use a PVBusDecoder to direct traffic to the correct port.

Table 3-12 ARMCortexA57xnCT ports (continued)

Name	Protocol	Type	Description
pvbus_m0	PVBus	Master	Bus master that the processor generates transactions on. This port is a PV representation of the AXI master port.
rei[4]	Signal	Slave	Individual processor *RAM Error Interrupt* signal input.
reset[0-3]	Signal	Slave	Individual processor reset.
rvbaraddr[0-3]	Value_64	Slave	Reset Vector Base Address for executing in AArch64 state. Only sampled at reset.
sei[0-3]	Signal	Slave	Individual processor *System Error Interrupt* signal input.
smpen[4]	Signal	Master	Status of the CPUECTLR.SMPEN bit, whether the processor is SMP enabled.
spiden[0-3]	Signal	Slave	Secure privileged invasive debug enable.
spniden[0-3]	Signal	Slave	Secure privileged non-invasive debug enable.
standbywfe[0-3]	Signal	Master	Indicates if a processor is in *Wait For Event* state.
standbywfi[0-3]	Signal	Master	Indicates if a processor is in *Wait For Interrupt* state.
standbywfil2	Signal	Master	Indicate that all the individual processors and the L2 systems are in a WFI state.
ticks[0-3]	InstructionCount	Master	Instruction count for visualization.
vcpumntirq[0-3]	Signal	Master	Virtual processor interface maintenance interrupt request.
vinithi[0-3]	Signal	Slave	Initialize with high vectors enabled after reset.
virtio_s	PVBus	Slave	A model-specific port that connects to virtio peripherals in the L2 system. It ensures coherency, with the correct attributes.
vfiq[0-3]	Signal	Slave	Processor *Virtual FIQ* signal input.
virq[0-3]	Signal	Slave	Processor *Virtual IRQ* signal input.
vsei[0-3]	Signal	Slave	Processor *Virtual System Error Interrupt* request.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.2 GICv3Comms protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.
2.5.4 v8EmbeddedCrossTrigger_controlprotocol protocol on page 2-65.

ARMCortexA57xnCT - parameters

This section describes the parameters.

ARMCortexA57xnCT parameters

Table 3-13 ARMCortexA57xnCT parameters

Name	Type	Allowed values	Default value	Description
CLUSTER_ID	uint32_t	0x0-0xFFFF	0x0	Master ID, bits[23:8] of the MPIDR using bits[15:0] of the CLUSTER_ID value.
dcache-state_modelled	bool	true, false	false	L1 D-cache has stateful implementation.[h]
GICDISABLE	bool	true, false	true	Disable the GIC CPU interface.
icache-state_modelled	bool	true, false	false	L1 I-cache has stateful implementation.[h]
l2cache-size	uint32_t	0x80000-0x200000	0x80000	L2 cache size in bytes.
PERIPHBASE	uint64_t	0-0xFFFFFFFFFFFF	0x13080000	Base address of peripheral memory space, the base address for the GIC CPU Interface registers, sampled into the *Configuration Base Address Register* (CBAR) at reset.

ARMCortexA57xnCT cache latency cluster parameters

Note

- These latencies are only effective when you enable cache-state modeling.
- Timing annotation for transactions downstream of the cache models propagates through the cache models.

Table 3-14 ARMCortexA57xnCT cache latency cluster parameters

Parameter	Type	Allowed values	Default value	Description
dcache-read_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-snoop_data_transfer_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for received snoop accesses that perfom a data transfer given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-write_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for write accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
icache-read_latency	uint64_t	-	0x0	L1 I-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when icache-state_modelled=true.
l2cache-read_latency	uint64_t	-	0x0	L2 cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.

[h] If either L1 cache is stateful, then the L2 cache is stateful.

Table 3-14 ARMCortexA57xnCT cache latency cluster parameters (continued)

Parameter	Type	Allowed values	Default value	Description
l2cache-snoop_data_transfer_latency	uint64_t	-	0x0	L2 cache timing annotation latency for received snoop accesses that perfom a data transfer given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_issue_latency	uint64_t	-	0x0	L2 cache timing annotation latency for snoop accesses issued by this cache in total ticks. For use when dcache-state_modelled=true.
l2cache-write_latency	uint64_t	-	0x0	L2 cache timing annotation latency for write accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.

ARMCortexA57xnCT core parameters

Table 3-15 ARMCortexA57xnCT core parameters

Name	Type	Allowed values	Default value	Description
AA64nAA32	bool	true, false	true	Register width state after reset
CFGEND	bool	true, false	false	Endianness configuration at reset. It sets the initial value of the EE bits in the CP15 SCTLR_EL3 and SCTR_S registers after a reset.
CFGTE	bool	true, false	false	Initialize to take exceptions in T32 state after reset.
CP15SDISABLE	bool	true, false	false	Disable write access to some Secure CP15 registers.
cpi_div	uint32_t	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	uint32_t	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
CRYPTODISABLE	bool	true, false	false	Disable ARMv8 Cryptography Extensions.[i]
DBGROMADDR	uint64_t	0x0-0xFFFFFFFFFFFFFFFF	0x22000000	Specify bits[43:12] of the top-level ROM table Physical Address.
DBGROMADDRV	bool	true, false	true	System samples DBGROMADDR.
max_code_cache	uint32_t	0x0-0xFFFFFFFFFFFFFFFF	0x16777216	Maximum number of bytes used for caching code translations.
min_sync_level	uint32_t	0-3	0	The minimum syncLevel.
RVBARADDR	uint64_t	0x0-0xFFFFFFFFFFFFFFFF	0x0	Reset Vector Base Address for executing in AArch64 state.
semihosting-ARM_SVC	uint32_t	0x0-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-cmd_line	string	No limit except memory	""	Command line available to semihosting.

[i] ARMv8 Cryptography Extensions require a separate package, which is subject to export license conditions. Contact ARM for details.

Table 3-15 ARMCortexA57xnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
semihosting-cwd	string	No limit except memory	""	Default working directory for semihosting.[jk]
semihosting-enable	bool	true, false	true	Enable semihosting traps.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0x0	Virtual address of semihosting heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of semihosting top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10000000	Virtual address of semihosting base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of semihosting stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFF	0xAB	T32 SVC number for semihosting.
vfp-enable_at_reset	bool	true, false	false	All implementation operations required in order to enable FP and ASE support are performed implicitly by the model.[l]
VINITHI	bool	true, false	false	Initialize with high vectors enabled after reset.

ARMCortexA57xnCT TLB latency core parameters

Note

- The walk_cache_latency parameter is only available for AEMv8 clusters and only effective when you configure the walk cache with a non-zero size.
- Timing annotation for transactions downstream of the TLB model propagates through the TLB model.

Table 3-16 ARMCortexA57xnCT TLB latency core parameters

Parameter	Type	Allowed values	Default value	Description
ptw_latency	uint64_t	-	0x0	Page table walker latency for *Timing Annotation* (TA), in simulation ticks.
tlb_latency	uint64_t	-	0x0	TLB latency for TA, in simulation ticks.
walk_cache_latency	uint64_t	-	0x0	Walk cache latency for TA, in simulation ticks.

Related references

1.8 Non-CADI sync watchpoints on page 1-24.

ARMCortexA57xnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the ETM registers and the integration and test registers.

j The host operating system limits the maximum path length.
k The semihosting-cwd option does not provide any security. Software running on the model can access files outside this directory using relative paths containing "../" or using absolute paths.
l This is a model-specific option that has no hardware equivalent. ARM recommends that it is only used in test systems and tied off to false in production systems.

ARMCortexA57xnCT - caches

This component implements representative L1 and L2 caches.

ARMCortexA57xnCT - debug features

This component exports a CADI debug interface.

ARMCortexA57xnCT - debug - registers

All modeled registers are visible in the debugger.

ARMCortexA57xnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA57xnCT - debug - memory

This component presents virtual and physical views of memory.

The virtual views are:

- Secure Monitor.
- NS Hyp.
- Guest.

These views are 2^{64} bytes in size.

The physical views are:

- Physical Memory (Secure).
- Physical Memory (Non-secure).

These views are 2^{64} bytes in size, however the maximum physical address permissible by the Cortex-A57 processor is 2^{44} — 1.

ARMCortexA57xnCT - trace sources

This section describes the trace sources for the ARM_Cortex-A57.

ASYNC_MEMORY_FAULT

Context ID Register write.. Fields:

FAULT `unsigned int`
: Fault status in ESR format.

VADDR `signed int`
: Virtual Address (or 0 if unavailable).

PADDR `unsigned int`
: Physical Address (or 0 if unavailable).

ArchMsg.Warning.Unpredictably Indexed PM Event Register

DISPLAY Event %{IsDirect:(Selector value|Register)} %{SEL} is >= %{N}. Fields:

SEL `unsigned int`
: Selector.

IsDirect `unsigned int`
: Direct Access rather than indirect.

N `unsigned int`
: Number of Accessible Registers.

ArchMsg.Warning.branch_to_unaligned_address

DISPLAY Branch to unaligned_address %{ADDR}. Fields:

ADDR unsigned int
: Unaligned branch address.

ArchMsg.Warning.cache_contents_unknown

DISPLAY %{SIDE:(D|I)}-Cache was enabled in %{REGIME:(SECURE|NON_SECURE|HYP)} regime but was not invalidated since power-on: cache lines could contain UNKNOWN content%{SIDE:(Note: D-side caches must invalidate SECURE lines even if the cache is only used by NON_SECURE code in order to ensure that unexpected dirty lines tagged SECURE are not naturally evicted|). Fields:

SIDE unsigned int
: 0 d-side cache or unified cache, 1 i-side.

REGIME unsigned int
: Current translation regime.

ArchMsg.Warning.dcimvac_matches_watchpoint

DISPLAY Watchpoints matching an AArch32 DCIMVA are implementation defined.

ArchMsg.Warning.decode_fracbitsnegative

DISPLAY VCVT Instruction is unpredictable with a negative number of fraction bits.

ArchMsg.Warning.decode_initblock

DISPLAY Instruction is unpredictable in an IT block.

ArchMsg.Warning.decode_initblocknotlast

DISPLAY Instruction is unpredictable in an IT block when not the last.

ArchMsg.Warning.decode_invalid_condition

DISPLAY Instruction is unpredictable when the condition is not AL.

ArchMsg.Warning.decode_invalidfieldcombination

DISPLAY Instruction is unpredictable with the values in %{FIELD1} and %{FIELD2}. Fields:

FIELD1 enum
: Field name.

  0x0
  : UNUSED_FIELDID

  0x1
  : RN

  0x2
  : RM

  0x3
  : RA

  0x4
  : RS

  0x5
  : RD

  0x6
  : RDLO

  0x7
  : RDHI

  0x8
  : RT

`0x9`
RT2
`0xa`
MSB
`0xb`
LSB
`0xc`
WIDTHM1
`0xd`
IMM12
`0xe`
W
`0xf`
P
`0x10`
MASK
`0x11`
SZ

FIELD2 `enum`
Field name.
`0x0`
UNUSED_FIELDID
`0x1`
RN
`0x2`
RM
`0x3`
RA
`0x4`
RS
`0x5`
RD
`0x6`
RDLO
`0x7`
RDHI
`0x8`
RT
`0x9`
RT2
`0xa`
MSB
`0xb`
LSB
`0xc`
WIDTHM1
`0xd`
IMM12
`0xe`
W
`0xf`
P
`0x10`
MASK
`0x11`
SZ

ArchMsg.Warning.decode_invalidvalue

DISPLAY Instruction is unpredictable with the value in %{FIELD}. Fields:

FIELD enum
Field name.
0x0
UNUSED_FIELDID
0x1
RN
0x2
RM
0x3
RA
0x4
RS
0x5
RD
0x6
RDLO
0x7
RDHI
0x8
RT
0x9
RT2
0xa
MSB
0xb
LSB
0xc
WIDTHM1
0xd
IMM12
0xe
W
0xf
P
0x10
MASK
0x11
SZ

ArchMsg.Warning.decode_mem_hint_unallocated

DISPLAY There is no memory hint allocated to this bit pattern.

ArchMsg.Warning.decode_registermatch

DISPLAY Instruction is unpredictable with %{FIELD1} and %{FIELD2} as the same register. Fields:

FIELD1 enum
Field name.
0x0
UNUSED_FIELDID
0x1
RN
0x2
RM

0x3
RA
0x4
RS
0x5
RD
0x6
RDLO
0x7
RDHI
0x8
RT
0x9
RT2
0xa
MSB
0xb
LSB
0xc
WIDTHM1
0xd
IMM12
0xe
W
0xf
P
0x10
MASK
0x11
SZ

FIELD2 enum
Field name.
0x0
UNUSED_FIELDID
0x1
RN
0x2
RM
0x3
RA
0x4
RS
0x5
RD
0x6
RDLO
0x7
RDHI
0x8
RT
0x9
RT2
0xa
MSB
0xb
LSB

`0xc`
: WIDTHM1

`0xd`
: IMM12

`0xe`
: W

`0xf`
: P

`0x10`
: MASK

`0x11`
: SZ

ArchMsg.Warning.decode_registermismatch

DISPLAY Instruction is unpredictable with %{FIELD1} and %{FIELD2} as different registers. Fields:

FIELD1 enum
: Field name.

`0x0`
: UNUSED_FIELDID

`0x1`
: RN

`0x2`
: RM

`0x3`
: RA

`0x4`
: RS

`0x5`
: RD

`0x6`
: RDLO

`0x7`
: RDHI

`0x8`
: RT

`0x9`
: RT2

`0xa`
: MSB

`0xb`
: LSB

`0xc`
: WIDTHM1

`0xd`
: IMM12

`0xe`
: W

`0xf`
: P

`0x10`
: MASK

`0x11`
: SZ

FIELD2 enum
: Field name.

`0x0`
: UNUSED_FIELDID

`0x1`
: RN

`0x2`
: RM

`0x3`
: RA

`0x4`
: RS

`0x5`
: RD

`0x6`
: RDLO

`0x7`
: RDHI

`0x8`
: RT

`0x9`
: RT2

`0xa`
: MSB

`0xb`
: LSB

`0xc`
: WIDTHM1

`0xd`
: IMM12

`0xe`
: W

`0xf`
: P

`0x10`
: MASK

`0x11`
: SZ

ArchMsg.Warning.decode_registeroutofrange

DISPLAY Float point instruction is unpredictable indexing beyond the end of the register bank.

ArchMsg.Warning.decode_sbzsbo

DISPLAY Reserved bits in the instruction are not canonical.

ArchMsg.Warning.decode_transactiontoobig

DISPLAY Attempt to perform a memory transaction of more than 128 bytes.

ArchMsg.Warning.decode_unpred_other

DISPLAY Instruction is unpredictable.

ArchMsg.Warning.decode_unpreduseofpc

DISPLAY Instruction is unpredictable when %{FIELD} is PC. Fields:

`FIELD` `enum`
: Field name.

`0x0`
: UNUSED_FIELDID

`0x1`
: RN

`0x2`
: RM

`0x3`
: RA

`0x4`
: RS

`0x5`
: RD

`0x6`
: RDLO

`0x7`
: RDHI

`0x8`
: RT

`0x9`
: RT2

`0xa`
: MSB

`0xb`
: LSB

`0xc`
: WIDTHM1

`0xd`
: IMM12

`0xe`
: W

`0xf`
: P

`0x10`
: MASK

`0x11`
: SZ

ArchMsg.Warning.decode_unpreduseofr13

DISPLAY Instruction is unpredictable when %{FIELD} is R13. Fields:

`FIELD` `enum`
: Field name.

`0x0`
: UNUSED_FIELDID

`0x1`
: RN

`0x2`
: RM

`0x3`
: RA

`0x4`
: RS

`0x5`
: RD

`0x6`
: RDLO

`0x7`
: RDHI

`0x8`
: RT

`0x9`
: RT2

`0xa`
: MSB

`0xb`
: LSB

`0xc`
: WIDTHM1

`0xd`
: IMM12

`0xe`
: W

`0xf`
: P

`0x10`
: MASK

`0x11`
: SZ

ArchMsg.Warning.decode_unpreduseofr13inreglist

DISPLAY Instruction is unpredictable with R13 in register-list.

ArchMsg.Warning.decode_writebackandbaseinlist

DISPLAY Instruction is unpredictable with write back when the base register is in the transfer list.

ArchMsg.Warning.decode_zeroregistersinlist

DISPLAY Instruction is unpredictable when no registers are to be transfered.

ArchMsg.Warning.recursive_branch

An instruction is performing a branch that targets the same instruction. If this was intended then a WFI might be more effective DISPLAY recursive branch detected at PC=%{PC} PRIMARY KEY PC . Fields:

PC `unsigned int`
: Address of instruction causing the branch.

ArchMsg.Warning.recursive_exception

An instruction has generated an exception that targets the same instruction. DISPLAY recursive %{TYPE} exception detected at PC=%{PC} (CPSR=%{CPSR}) PRIMARY KEY PC . Fields:

PC `unsigned int`
: Address of instruction causing the exception.

CPSR `unsigned int`
: Processor state.

TYPE `enum`
: Type of exception.

`0x4`
: undefined instruction

`0xc`
: prefetch abort

ArchMsg.Warning.reentrant_vector_catch

DISPLAY Vector catch debug event on Prefetch or Data abort vector %{ADDR}. Fields:

ADDR unsigned int
: vector address.

ArchMsg.Warning.reserved_it_state

DISPLAY Execution with reserved IT state %{ITSTATE}. Fields:

ITSTATE unsigned int
: reserved IT state.

ArchMsg.Warning.tlb_contents_unknown

DISPLAY Translation was enabled in %{REGIME:(SECURE|NON_SECURE|HYP)} regime (vmid=%{VMID:x}) without invalidating %{INVALIDITY:(|d-side |i-side |)}entries since power-on: %{INVALIDITY:(|D|I|)}TLB could contain UNKNOWN content. Fields:

VMID unsigned int
: Current VMID if applicable.

REGIME unsigned int
: Current translation regime.

INVALIDITY unsigned int
: Which TLB may be invalid (if TLBs are separate).

ArchMsg.Warning.unknown_DLR_DSPSR

Exiting debug state with unknown DLR or DSPSR..

ArchMsg.Warning.unknown_ELR_SPSR

Returning from debug exception with unknown ELR or SPSR..

ArchMsg.Warning.unpredictable_banked_register_access

DISPLAY Access to banked register (R=%{R:d} SYSm=%{SYSM:2x}) is unpredictable. Fields:

R unsigned int
: R bit.

SYSM unsigned int
: SYSm.

ArchMsg.Warning.unpredictable_ccfail_undef

DISPLAY An instruction %{OPCODE} at VA=%{VADDR} failed its condition codes check but would otherwise have caused a non-data-dependent trap or undefined exception. Behaviour is implementation-defined choice of exception or nop.. Fields:

VADDR unsigned int
: Virtual address of the instruction being executed.

OPCODE unsigned int
: opcode of the instruction.

ArchMsg.Warning.unpredictable_t32_breakpoint_bas

DISPLAY Unpredictable breakpoint byte address select on a T32 instruction at %{ADDR} with DBGBCR%{N:d}=%{DBGBCR}, DBGBVR%{N:d}=%{DBGBVR}. Fields:

ADDR unsigned int
: Address.

N unsigned int
: Breakpoint number.

DBGBCR unsigned int
: Breakpoint control register.

DBGBVR unsigned int
: Breakpoint value register.

ADDR unsigned int
: Address.

N unsigned int
: Breakpoint number.

DBGBCR unsigned int
: Breakpoint control register.

DBGBVR unsigned int
: Breakpoint value register.

ArchMsg.Warning.unpredictable_t32_breakpoint_bas

DISPLAY Unpredictable breakpoint byte address select on a T32 instruction at %{ADDR} with DBGBCR%{N:d}=%{DBGBCR}, DBGBVR%{N:d}=%{DBGBVR}. Fields:

ADDR unsigned int
: Address.

N unsigned int
: Breakpoint number.

DBGBCR unsigned int
: Breakpoint control register.

DBGBVR unsigned int
: Breakpoint value register.

ADDR unsigned int
: Address.

N unsigned int
: Breakpoint number.

DBGBCR unsigned int
: Breakpoint control register.

DBGBVR unsigned int
: Breakpoint value register.

ArchMsg.Warning.unpredictable_watchpoint_far

DISPLAY The value for FAR/EDWAR for hitting this watchpoint may be anywhere in the range %{LOWER_BOUND} to %{UPPER_BOUND}. Fields:

LOWER_BOUND unsigned int
: the lowest address accessed by the instruction that triggered the watchpoint.

UPPER_BOUND unsigned int
: the highest watchpointed address accessed by that instruction.

ArchMsg.Warning.warning_AdvSIMDExpandImmUnexpectedZero

DISPLAY AdvSIMDExpandImm may treat this immediate value as UNPREDICTABLE.

ArchMsg.Warning.warning_ConditionalSMC

DISPLAY SMC instruction has UNPREDICTABLE effects when conditional when combined with traps.

ArchMsg.Warning.warning_access_crosses_page_boundary

DISPLAY Access crosses page boundary.

ArchMsg.Warning.warning_access_crosses_page_boundary_has_fault

DISPLAY Access crosses page boundary has fault(s).

ArchMsg.Warning.warning_access_crosses_page_boundary_spanning_different_memory

DISPLAY Access at address %{ADDR} crosses page from MEM_TYPE %{MEMTYPE_PAGE1} to %{MEMTYPE_PAGE2}. Fields:

ADDR unsigned int
: address of access crossing page.

MEMTYPE_PAGE1 signed int
: First Page's memory type (0-Normal, 1-Non-Normal).

MEMTYPE_PAGE2 signed int
: Second Page's memory type (0-Normal, 1-Non-Normal).

ArchMsg.Warning.warning_access_wraps_around_memory

DISPLAY Acceess warps around memory in %{WIDTH:d}bit mode. Fields:

WIDTH unsigned int
: Address width.

ArchMsg.Warning.warning_bx_from_thumbee

DISPLAY BX from ThumbEE to ARM at %{ADDR}. Fields:

ADDR unsigned int
: Destination address.

ArchMsg.Warning.warning_ccsidr_unimplemented_level

DISPLAY CCSIDR read while CSSELR %{CSSELR:x} points to an unimplemented cache level. Fields:

CSSELR unsigned int
: current effective value of CSSELR.

ArchMsg.Warning.warning_change_to_ns_when_tge_set

DISPLAY Attempting to change NS to 1 when HCR.TGE = 1.

ArchMsg.Warning.warning_contiguous_bit_check_abort

DISPLAY An abort occurred whilst attempting to check TLB entry is contiguous at %{ENTRY_ADDR}. Fields:

ENTRY_ADDR unsigned int
: address of conflicting TLB entry.

ArchMsg.Warning.warning_contiguous_bit_error

DISPLAY TLB entry at %{ENTRY_ADDR} was not contiguous with entry at %{CONTIG_ADDR} Contents are %{ENTRY_DATA} but expected %{CONTIG_DATA}. Fields:

ENTRY_ADDR unsigned int
: address of conflicting TLB entry.

ENTRY_DATA unsigned int
: contents of conflicting TLB entry.

CONTIG_ADDR unsigned int
: address of the TLB first read, with which this entry is expected to be contiguous.

CONTIG_DATA unsigned int
: expected contents based on the entry at CONTIG_ADDR.

ArchMsg.Warning.warning_cp10_cp11_mismatch

DISPLAY Unpredictable configuration of CP10 and CP11 controls at EL%{EL:d}. Fields:

EL unsigned int
EL owning the control.

ArchMsg.Warning.warning_cp10_cp11_reserved_value

DISPLAY Unpredictable configuration of CPACR CP10 (%{CP10}) or CP11 (%{CP11}). Fields:

CP10 unsigned int
CP10 access.
CP11 unsigned int
CP11 access.

ArchMsg.Warning.warning_csselr_level_out_of_range

DISPLAY CSSELR.Level out of range, written value %{LEVEL} with only %{IMPLEMENTED:d} levels implemented. Fields:

LEVEL unsigned int
written value of CSSELR.Level.
IMPLEMENTED unsigned int
number of cache levels implemented.

ArchMsg.Warning.warning_debug_flow_control_bits_not_obeyed

DISPLAY Flow control bits for %{REGISTER} were not obeyed, causing %{ERROR}. Fields:

REGISTER enum
register under consideration.
0x0
DTRRX
0x1
DTRTX
0x2
ITR
ERROR enum
type of error resulting.
0x0
overflow
0x1
underflow

ArchMsg.Warning.warning_debug_register_access_during_reset

DISPLAY Unpredictable %{IS_WRITE:(read|write)} access to debug register offset %{OFFSET} during reset. Fields:

IS_WRITE unsigned int
Write Not Read.
OFFSET unsigned int
Register Offset.

ArchMsg.Warning.warning_decode_cps_inconsistent_fields

DISPLAY The mode setting fields in the CPS instruction are inconsistent.

ArchMsg.Warning.warning_decode_invalid_state

DISPLAY Instruction is unpredictable in current exception-level/security state.

ArchMsg.Warning.warning_default_cacheable_mmu_on

DISPLAY Default cacheable is enabled (HCR.DC = 1) while MMU is enabled.

ArchMsg.Warning.warning_default_cacheable_vmmu_off

DISPLAY Default cacheable is enabled (HCR.DC = 1) while HCR.VM = 0.

ArchMsg.Warning.warning_deprecated_wvr_bit_2_set

DISPLAY DBGWVR%{N:d}_EL1=%{WVR:h} has bit 2 set, deprecated in ARMv8. Fields:

N unsigned int
: index.

WVR unsigned int
: DBGWVR.

ArchMsg.Warning.warning_eret_while_software_step_active_pending

Missing ISB between setting MDSCR_EL1.SS and ERET.

ArchMsg.Warning.warning_exclusive_to_non_normal

DISPLAY Exclusive Access to Non-Normal Memory %{ADDR}. Fields:

ADDR unsigned int
: Address.

ArchMsg.Warning.warning_exclusive_to_non_writeback

DISPLAY Exclusive Access to Non-Writeback-Cacheable Memory %{ADDR}. Fields:

ADDR unsigned int
: Address.

ArchMsg.Warning.warning_execute_from_device_memory

DISPLAY An attempt was made to execute from Device memory.

ArchMsg.Warning.warning_illegal_cpsr_mode

DISPLAY Writing an illegal or unimplement mode (%{NEW_MODE:h}) to CPSR was unpredictable. Fields:

NEW_MODE unsigned int
: New value of CPSR.M.

ArchMsg.Warning.warning_illegal_srs_mode

DISPLAY Illegal or UNPREDICTABLE mode (%{MODE:h}) used for SRS instruction. Fields:

MODE unsigned int
: mode for Banked SP.

ArchMsg.Warning.warning_implementation_defined_read_debug_register_in_SCS

Read Access to Debug registers from the processor is IMPLEMENTATION DEFINED. Fields:

PPB_OFFSET unsigned int
: debug register offset.

ArchMsg.Warning.warning_implementation_defined_sequential_security_transitions_supported

The behaviour of a sequential instruction fetches that cross from non-secure to secure memory and contain SG instruction is CONSTRAINED UNPREDICTABLE.

ArchMsg.Warning.warning_implementation_defined_stack_limit_check_supported

It is IMPLEMENTATION DEFINED whether stack pointer limit checking is performed for the this instructions. Fields:

ADDRESS `unsigned int`
: Address of instruction.

OPCODE `unsigned int`
: instruction opcode.

ArchMsg.Warning.warning_implementation_defined_write_debug_register_in_SCS

Write Access to Debug registers from the processor is IMPLEMENTATION DEFINED. Fields:

PPB_OFFSET `unsigned int`
: debug register offset.

DATA `unsigned int`
: data attempted to be written.

ArchMsg.Warning.warning_invalid_tcr_granule

DISPLAY TCR.TG%{TG_ID:(0|1)} bits have been set to a granule size %{REQUEST:d}K not implemented - using %{SUBSTITUTE:d}K. Fields:

REQUEST `signed int`
: page size requested (or 0 for reserved).

TG_ID `bool`
: bits TG0 or TG1.

SUBSTITUTE `signed int`
: best guess available page size.

ArchMsg.Warning.warning_load_multiple_user_registers_from_user_mode

DISPLAY An LDM(user registers) instruction executed from user mode.

ArchMsg.Warning.warning_load_pc_from_unaligned

DISPLAY PC loaded from an unaligned location.

ArchMsg.Warning.warning_reserved_breakpoint_state_match

DISPLAY %{IS_BREAKPOINT:(Watchpoint|Breakpoint)} programmed with a reserved combination of HMC, SSC and %{IS_BREAKPOINT:(PAC|PMC)}. Fields:

IS_BREAKPOINT `unsigned int`
: Is breakpoint, not watchpoint.

ArchMsg.Warning.warning_shareability

DISPLAY Unpredictable: combination of the 1st and 2nd stages of translation is Normal Inner Non-Cacheable, Outer Non-Cacheable.

ArchMsg.Warning.warning_software_step_set_while_enabled

DISPLAY MDSCR_EL1.SS set to 1 while software step debug exceptions are enabled.

ArchMsg.Warning.warning_thumb_instruction_wraps_around_memory

DISPLAY Thumb instruction warps around memory.

ArchMsg.Warning.warning_ttbr_sbz_bits_are_not_zero

DISPLAY %{TTBR}[%{MSB:d}:0] should be zero, but are not. Fields:

TTBR `enum`

which TTBR register.

`0x0`
: TTBR0

`0x1`
: TTBR1

`0x2`
: VTTBR

`0x3`
: TTBCR

`0x4`
: VTCR

`0x10`
: TTBR0_ELx

`0x11`
: TTBR1_ELx

`0x12`
: VTTBR_EL2

`0x13`
: TCR_ELx

`0x14`
: VTCR_EL2

MSB `unsigned int`
: x-1.

ArchMsg.Warning.warning_unaligned_address_dbgdtrrx_write

DISPLAY Unaligned address %{ADDR} in X0/R0 when DBGDTRRX written in memory access mode. Fields:

ADDR `unsigned int`
: Address.

ArchMsg.Warning.warning_unaligned_address_dbgdtrtx_read

DISPLAY Unaligned address %{ADDR} in X0/R0 when DBGDTRTX read in memory access mode. Fields:

ADDR `unsigned int`
: Address.

ArchMsg.Warning.warning_unaligned_to_device

DISPLAY Unaligned Access to Device Memory %{ADDR}. Fields:

ADDR `unsigned int`
: Address.

ArchMsg.Warning.warning_unaligned_to_strongly_ordered

DISPLAY Unaligned Access to Strongly Ordered Memory %{ADDR}. Fields:

ADDR `unsigned int`
: Address.

ArchMsg.Warning.warning_unknown_sau_rnr

DISPLAY SAU_RNR was set to an unsupported value.

ArchMsg.Warning.warning_unpredictable_AIRCR_PRIP_and_BFHFNMINP

DISPLAY The effect of setting both AIRCR.BFHFNMINP and AIRCR.PRIP to 1 is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_AIRCR_VECTCLRACTIVE_when_not_in_debug

DISPLAY The effect of writing a 1 to AIRCR.VECTCLRACTIVE if the processor is not halted in Debug state is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_AIRCR_VECTRESET_and_SYSRESETREQ

DISPLAY When the processor is halted in Debug state, if a write to the register writes a 1 to both VECTRESET and SYSRESETREQ, the behavior is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_AIRCR_VECTRESET_when_not_in_debug

DISPLAY The effect of writing a 1 to AIRCR.VECTRESET if the processor is not halted in Debug state is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_AIRCR_incorrect_VKEY

DISPLAY The value 0x05FA must be written to AIRCR.VKEY, otherwise the register write is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_EXE_RETURN_Reserved_Bit

EXC_RETURN[23:7] are reserved with the special condition that all bits should be written as one. Values other than all 1s are UNPREDICTABLE..

ArchMsg.Warning.warning_unpredictable_change_to_DEMCR_MON_STEP_at_insufficient_priority

DISPLAY The effect of changing DEMCR.MON_STEP at an execution priority that is lower than the priority of the DebugMonitor exception is UNPREDICTABLE..

ArchMsg.Warning.warning_unpredictable_change_to_priority_of_active_exception

DISPLAY Changing the priority of an active exception is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_clear_lspact

DISPLAY UNPREDICTABLE state on context switch. Fields:

DRVEXC_TAKEN `unsigned int`
: derived exception taken not pended.

REACHED_LAST_STORE `unsigned int`
: only the last faulted.

ArchMsg.Warning.warning_unpredictable_exception_catch

DISPLAY Generation of an exception catch event is unpredictable.

ArchMsg.Warning.warning_unpredictable_exception_return_inconsistent_state

DISPLAY state on exception return is unpredictable.

ArchMsg.Warning.warning_unpredictable_exception_return_instruction

DISPLAY exception return instruction is unpredictable.

ArchMsg.Warning.warning_unpredictable_in_debug_state

DISPLAY Instruction is UNPREDICTABLE when executed in debug state.

ArchMsg.Warning.warning_unpredictable_pmu_counter_access

DISPLAY Access to PMU counters from non-secure EL0 or EL1 is UNPREDICTABLE with MDCR_EL2.HPMN set to 0.

ArchMsg.Warning.warning_unpredictable_prioritization_breakpoint_match_vector_catch

DISPLAY Two events with the same priority occurred at the same instruction: Address Matching Vector Catch debug event, and Breakpoint debug event It is constrained unpredictable which is taken..

ArchMsg.Warning.warning_unpredictable_stack_selection

DISPLAY UNPREDICTABLE state on context switch.

ArchMsg.Warning.warning_unpredictable_tsize_out_of_range

DISPLAY %{TCR}.T%{TTBR:d}SZ (%{TSIZE:d} bits) is out of range. Must be between 25 and %{TSIZE_MAX:d}.. Fields:

TCR enum
: Which TCR.
  0x0
  : TTBR0
  0x1
  : TTBR1
  0x2
  : VTTBR
  0x3
  : TTBCR
  0x4
  : VTCR
  0x10
  : TTBR0_ELx
  0x11
  : TTBR1_ELx
  0x12
  : VTTBR_EL2
  0x13
  : TCR_ELx
  0x14
  : VTCR_EL2

TTBR unsigned int
: Which TTBR region.

TSIZE signed int
: Value of TSize.

TSIZE_MAX signed int
: The maximum allowed value.

ArchMsg.Warning.warning_unpredictable_unaligned_pc_as_base_register

DISPLAY The PC value must be word-aligned, otherwise the behavior of the instruction is UNPREDICTABLE.

ArchMsg.Warning.warning_unpredictable_vmsa_memattrib

DISPLAY Unpredictable vmsa memory attribute with texcb value programmed to %{TEXCB} when tex remap is %{TEX_REMAP}. Fields:

TEXCB unsigned int
: texcb value.

TEX_REMAP bool
: use tex remap.

ArchMsg.Warning.warning_unpredictable_vtcr_t0sz_sl0

DISPLAY The combination of VTCR.T0SZ=%{T0SZ:d} and VTCR.SL0=%{SL0:d} is unpredictable. Fields:

T0SZ signed int
: VTCR.T0SZ, required input address range.

SL0 signed int
: VTCR.SL0, starting level for stage 2 translation table walks.

ArchMsg.Warning.warning_unpredictable_write_DHCSR

DISPLAY UNPREDICTABLE set of bit-changes in DHCSR. Fields:

OLD unsigned int
: previous value.

NEW unsigned int
: data attempted to be written.

ArchMsg.Warning.warning_unsupported_access_to_memory_mapped_register

DISPLAY Access to memory mapped register is unsupported.

ArchMsg.Warning.warning_user_jmcr_access

DISPLAY User %{WRITE:(write|read)} access to JMCR is unpredictable. Fields:

WRITE unsigned int
: access is write.

ArchMsg.Warning.warning_virt_ext_secure_privileged_access

DISPLAY Access to Virtualization Extensions from a non-Monitor Secure Privileged mode.

ArchMsg.Warning.warning_wcr_mask_and_bas

DISPLAY DBGWCR_EL1.MASK is non-zero and BAS is not 0xff.

ArchMsg.Warning.warning_wcr_mask_reserved

DISPLAY DBGWCR_ELn.MASK is set to a reserved value.

ArchMsg.Warning.warning_wcr_non_configuous_bas

DISPLAY DBGWCR_EL1.BAS has non-contiguous set of ones.

ArchMsg.Warning.warning_wvr_masked_bit_not_zero

DISPLAY DBGWVRn_EL1.VA contains a bit masked by DBGWCRn_EL1.MASK that is not zero.

ArchMsg.Why.why_illegal_state

DISPLAY illegal mode change: %{MESSAGE}. Fields:

MESSAGE string
: The message.

BRA_DIR

Direct branches, to immediate address.. Fields:

PC unsigned int
: The address of the branch instruction..

ISET enum
: The instructions set of the branch instruction..

0x0
ARM
0x1
Thumb
0x2
Jazelle
0x4
AArch64

TARGET_PC unsigned int
The address the instruction branches to..

TARGET_ISET enum
The instructions set after the branch..

0x0
ARM
0x1
Thumb
0x2
Jazelle
0x4
AArch64

IS_COND bool
Indicates if this is a conditional branch..

CORE_NUM unsigned int
Core number in a multi processor..

BRA_INDIR

Indirect branches, perhaps to a register.. Fields:

PC unsigned int
The address of the branch instruction..

ISET enum
The instructions set of the branch instruction..

0x0
ARM
0x1
Thumb
0x2
Jazelle
0x4
AArch64

TARGET_PC unsigned int
The address the instruction branches to..

TARGET_ISET enum
The instructions set after the branch..

0x0
ARM
0x1
Thumb
0x2
Jazelle
0x4
AArch64

IS_COND bool
Indicates if this is a conditional branch..

CORE_NUM unsigned int
Core number in a multi processor..

CACHE_MAINTENANCE_OP

Cache Maintenance Operation. Fields:

CORE_NUM unsigned int
: Core number in a multi processor..

SIDE enum
: Inst / Data..

  0x0
  : Instruction

  0x1
  : Data

  0x2
  : Instruction and Data

FUNCTION enum
: Clean / Invalidate.

  0x0
  : Clean

  0x1
  : Invalidate

  0x2
  : Clean and Invalidate

SCOPE enum
: Affected region.

  0x0
  : By Set/Way

  0x1
  : By MVA to PoC

  0x2
  : By MVA to PoU

  0x3
  : All to PoU

  0x4
  : All Inner Shareable to PoU

DATA unsigned int
: Specified MVA or Set/way.

CCFAIL

Conditional instruction condition check fail.. Fields:

COND enum
: The condition of the conditional instruction..

  0x0
  : EQ

  0x1
  : NE

  0x2
  : CS

  0x3
  : CC

  0x4
  : MI

  0x5
  : PL

  0x6
  : VS

`0x7`
VC
`0x8`
HI
`0x9`
LS
`0xa`
GE
`0xb`
LT
`0xc`
GT
`0xd`
LE
`0xe`
AL

PC `unsigned int`
The address of the conditional instruction..

CORE_NUM `unsigned int`
Core number in a multi processor..

CCFAIL_UNC

Conditional instruction condition check fail.. Fields:

COND `enum`
The condition of the conditional instruction..

`0x0`
EQ
`0x1`
NE
`0x2`
CS
`0x3`
CC
`0x4`
MI
`0x5`
PL
`0x6`
VS
`0x7`
VC
`0x8`
HI
`0x9`
LS
`0xa`
GE
`0xb`
LT
`0xc`
GT
`0xd`
LE
`0xe`
AL

PC unsigned int
: The address of the conditional instruction..

CORE_NUM unsigned int
: Core number in a multi processor..

CCPASS

Conditional instruction condition check pass.. Fields:

COND enum
: The condition of the conditional instruction..

  0x0
  : EQ

  0x1
  : NE

  0x2
  : CS

  0x3
  : CC

  0x4
  : MI

  0x5
  : PL

  0x6
  : VS

  0x7
  : VC

  0x8
  : HI

  0x9
  : LS

  0xa
  : GE

  0xb
  : LT

  0xc
  : GT

  0xd
  : LE

  0xe
  : AL

PC unsigned int
: The address of the conditional instruction..

CORE_NUM unsigned int
: Core number in a multi processor..

CCPASS_UNC

Conditional instruction condition check pass.. Fields:

COND enum
: The condition of the conditional instruction..

  0x0
  : EQ

  0x1
  : NE

  0x2
  : CS

`0x3`
: CC

`0x4`
: MI

`0x5`
: PL

`0x6`
: VS

`0x7`
: VC

`0x8`
: HI

`0x9`
: LS

`0xa`
: GE

`0xb`
: LT

`0xc`
: GT

`0xd`
: LE

`0xe`
: AL

PC `unsigned int`
: The address of the conditional instruction..

CORE_NUM `unsigned int`
: Core number in a multi processor..

CHECKPOINT_MESSAGE

Report error messages from the checkpointing process. Fields:

message `string`
: Message contents..

CHECKPOINT_RESTORE_END

Checkpoint restore completed.

CHECKPOINT_RESTORE_START

Checkpoint restore about to start.

CHECKPOINT_SAVE_END

Checkpoint save completed.

CHECKPOINT_SAVE_START

Checkpoint save about to start.

CODE_CACHE_FLUSH

Code cache flushed..

CODE_CACHE_FULL

Code cache full..

COMPILE_BLOCK_END

Last instruction of basic block translated.. Fields:

VADDR unsigned int
: Address of next instruction after this basic block.

COMPILE_BLOCK_START

First instruction of basic block translated.. Fields:

VADDR unsigned int
: Address of first instruction in basic block.

COMPILE_INST

ARM instruction compiled. Fields:

PC unsigned int
: The address of the instruction..

OPCODE unsigned int
: The opcode of the instruction..

SIZE unsigned int
: The size of the instruction in bytes..

ISET enum
: The instruction set of this instruction..

  0x0
  : ARM

  0x1
  : Thumb

  0x2
  : Jazelle

  0x4
  : AArch64

ITSTATE unsigned int
: The ITSTATE current for the instruction..

DISASS string
: Disassembly of the instruction..

CONTEXTIDR

Context ID Register write.. Fields:

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The new value written..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

CORE_ENDIAN

Core BE8 Big-Endian state changed. Fields:

BE8 bool
: Core BE8 Big-Endian state.

CORE_INFO

Static processor attributes. Only triggered by a call to DumpState().. Fields:

NUM_CORES unsigned int
: The number of cores in this MP processor..

CORE_NUM unsigned int
: The number of this core in an MP processor..

CLUSTER_ID unsigned int
: The cluster ID of this processor..

ARCH_PROFILE enum
: The architecture profile of the core..

0x0
: AR

0x1
: M

0x2
: A64

MEM_ARCH enum
: The memory architecture of the core..

0x0
: VMSA

0x1
: PMSA

0x2
: FLAT

QUANTUM_SIZE unsigned int
: The default quantum size of the core..

SECURITY_FEATURES bool
: Does the core have security features?.

FPU_VERSION unsigned int
: The VFP version implemented by the core..

CORE_LOADS

Processor load accesses.. Fields:

VADDR unsigned int
: The virtual address of the access..

RESPONSE enum
: 0=Aborted, 1=OK, 2=Exclusive Failed.

0x0
: Aborted

0x1
: OK

0x2
: Failed

LOCK enum
: Normal, exclusive or locked access..

0x0
: Normal

0x1
: Exclusive

0x2
: Locked

TRANS bool
: Is this a translated access..

ACQREL enum
: Is this an acquire/release.

0x0
: None

`0x1`
: Global

`0x2`
: Local

SIZE `unsigned int`
: Width of the access in bytes. Only required if DATA is not traced..

ELEMENT_SIZE `unsigned int`
: Width of each element..

PADDR `unsigned int`
: The physical (translated) address..

NSDESC `unsigned int`
: The physical address non-secure bit..

PADDR2 `unsigned int`
: If different from PADDR, the physical address of the second page of the access..

NSDESC2 `unsigned int`
: The second page physical address non-secure bit..

DATA `unsigned int`
: The data read or written..

CORE_REGS

Changes of the core registers R0 to R14.. Fields:

ID `unsigned int`
: The register number, 0 to 14..

PHYS_ID `enum`
: The physical register accessed..

`0x0`
: R0_usr

`0x1`
: R1_usr

`0x2`
: R2_usr

`0x3`
: R3_usr

`0x4`
: R4_usr

`0x5`
: R5_usr

`0x6`
: R6_usr

`0x7`
: R7_usr

`0x8`
: R8_usr

`0x9`
: R9_usr

`0xa`
: R10_usr

`0xb`
: R11_usr

`0xc`
: R12_usr

`0xd`
: SP_usr

`0xe`
: LR_usr

`0xf`
: R8_fiq

`0x10`
: R9_fiq

`0x11`
: R10_fiq

`0x12`
: R11_fiq

`0x13`
: R12_fiq

`0x14`
: SP_fiq

`0x15`
: LR_fiq

`0x16`
: SP_irq

`0x17`
: LR_irq

`0x18`
: SP_svc

`0x19`
: LR_svc

`0x1a`
: SP_abt

`0x1b`
: LR_abt

`0x1c`
: SP_und

`0x1d`
: LR_und

`0x1e`
: SP_hyp

`0x1f`
: ELR_hyp

`0x20`
: SP_mon

`0x21`
: LR_mon

`0x22`
: SPSR_fiq

`0x23`
: SPSR_irq

`0x24`
: SPSR_svc

`0x25`
: SPSR_und

`0x26`
: SPSR_abt

`0x27`
: SPSR_hyp

`0x28`
: SPSR_mon

`VALUE` `unsigned int`
: The new value written to the register..

`OLD_VALUE` `unsigned int`
: The old value overwritten..

MODE enum
: Bank of the register accessed..

  0x0
  : EL0t

  0x4
  : EL1t

  0x5
  : EL1h

  0x8
  : EL2t

  0x9
  : EL2h

  0xc
  : EL3t

  0xd
  : EL3h

  0x10
  : usr

  0x11
  : fiq

  0x12
  : irq

  0x13
  : svc

  0x16
  : mon

  0x17
  : abt

  0x1a
  : hyp

  0x1b
  : und

  0x1f
  : sys

CORE_NUM unsigned int
: Core number in a multi processor..

CORE_REGS64

Changes of the core registers X0..X30, SP_ELn.. Fields:

ID enum
: The register number, 0..30 for X0..X30, >=32 for SP_ELn..

  0x0
  : X0

  0x1
  : X1

  0x2
  : X2

  0x3
  : X3

  0x4
  : X4

  0x5
  : X5

  0x6
  : X6

`0x7`
: X7

`0x8`
: X8

`0x9`
: X9

`0xa`
: X10

`0xb`
: X11

`0xc`
: X12

`0xd`
: X13

`0xe`
: X14

`0xf`
: X15

`0x10`
: X16

`0x11`
: X17

`0x12`
: X18

`0x13`
: X19

`0x14`
: X20

`0x15`
: X21

`0x16`
: X22

`0x17`
: X23

`0x18`
: X24

`0x19`
: X25

`0x1a`
: X26

`0x1b`
: X27

`0x1c`
: X28

`0x1d`
: X29

`0x1e`
: X30

`0x20`
: SP_EL0

`0x21`
: SP_EL1

`0x22`
: SP_EL2

`0x23`
: SP_EL3

VALUE unsigned int
: The new value written to the register..

OLD_VALUE unsigned int
: The old value overwritten..

CORE_NUM unsigned int
: Core number in a multi processor..

CORE_STORES

Processor store accesses.. Fields:

VADDR unsigned int
: The virtual address of the access..

RESPONSE enum
: 0=Aborted, 1=OK, 2=Exclusive Failed.

  0x0
  : Aborted

  0x1
  : OK

  0x2
  : Failed

LOCK enum
: Normal, exclusive or locked access..

  0x0
  : Normal

  0x1
  : Exclusive

  0x2
  : Locked

TRANS bool
: Is this a translated access..

ACQREL enum
: Is this an acquire/release.

  0x0
  : None

  0x1
  : Global

  0x2
  : Local

SIZE unsigned int
: Width of the access in bytes. Only required if DATA is not traced..

ELEMENT_SIZE unsigned int
: Width of each element..

PADDR unsigned int
: The physical (translated) address..

NSDESC unsigned int
: The physical address non-secure bit..

PADDR2 unsigned int
: If different from PADDR, the physical address of the second page of the access..

NSDESC2 unsigned int
: The second page physical address non-secure bit..

DATA unsigned int
: The data read or written..

CP14_READ

System Coprocessor register read.. Fields:

CRn unsigned int
: CRn.

opc1 unsigned int
: opcode 1.

CRm unsigned int
: CRm.

opc2 unsigned int
: opcode 2.

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The value read..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the CP14 register accessed..

CP14_WRITE

System Coprocessor register write.. Fields:

CRn unsigned int
: CRn.

opc1 unsigned int
: opcode 1.

CRm unsigned int
: CRm.

opc2 unsigned int
: opcode 2.

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The new value written..

UPDATED_VALUE unsigned int
: Updated value of the register now it has been written..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the CP14 register accessed..

CP15_READ

System Coprocessor register read.. Fields:

CRn unsigned int
: CRn.

opc1 unsigned int
: opcode 1.

CRm unsigned int
: CRm.

opc2 unsigned int
: opcode 2.

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The value read..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the CP15 register accessed..

CP15_READ64

System Coprocessor register read.. Fields:

CRm unsigned int
: CRm.

opc unsigned int
: opcode 1.

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The value read..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the CP15 register accessed..

CP15_WRITE

System Control Coprocessor register write.. Fields:

CRn unsigned int
: CRn.

opc1 unsigned int
: opcode 1.

CRm unsigned int
: CRm.

opc2 unsigned int
: opcode 2.

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The new value written..

UPDATED_VALUE unsigned int
: Updated value of the register now it has been written..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the CP15 register accessed..

CP15_WRITE64

System Coprocessor register write.. Fields:

CRm unsigned int
: CRm.

opc unsigned int
: opcode 1.

NS bool
: Secure or nonsecure banked register is accessed..

VALUE unsigned int
: The new value written..

UPDATED_VALUE unsigned int
: Updated value of the register now it has been written..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the CP15 register accessed..

CPSR

CPSR change.. Fields:

VALUE unsigned int
: The new CPSR value.

OLD_VALUE unsigned int
: The old CPSR value.

CORE_NUM unsigned int
: Core number in a multi processor..

UNKNOWN unsigned int
: Bits within the register that have unknown value..

CRYPTO_SPEC

Every crypto instruction speculatively executed..

DEBUG_EVENT

Hardware debug support event.. Fields:

EVENT enum
: Description of event.
  - 0x0: HaltingDebugState
  - 0x1: HitWatchPoint
  - 0x2: HitBreakPoint
  - 0x3: HitVectorCatch
  - 0x4: DBGEN
  - 0x5: EDBGREQ
  - 0x6: SPIDEN
  - 0x7: NIDEN
  - 0x8: PIDEN
  - 0x9: PNIDEN

`0xa`
SPNIDEN
`0xb`
PADDRDBG31
`0xc`
DBGSWENABLE
`0xd`
DBGRESTART
`0xe`
HIDEN
`0xf`
HNIDEN
`0x10`
HitExceptionCatch

VALUE `unsigned int`
data value.

END_COMPILE

Compilation end.. Fields:

INST_COUNT `unsigned int`
Number of instructions compiled since START_COMPILE..
FETCHFAIL_COUNT `unsigned int`
Number of basic blocks exited due to ifetch failure START_COMPILE..
END_OF_PAGE_COUNT `unsigned int`
Number of basic blocks exited due to page boundary since START_COMPILE..
PAGE_STRADDLE_COUNT `unsigned int`
Number of basic blocks exited due to unaligned instructions crossing page since START_COMPILE..
NONSEQ_COUNT `unsigned int`
Number of basic blocks exited due to non-sequential instructions since START_COMPILE..

EXCEPTION

Exception taken.. Fields:

PC `unsigned int`
The location where the exception occurred..
LR `unsigned int`
The value assigned to the link register..
PREFERRED_RETURN `unsigned int`
The preferred return address for the exception..
TARGET_PC `unsigned int`
The address the exception branches to..
VECTOR `enum`
The exception vector..
`0x0`
Reset
`0x4`
UndefinedInstr
`0x8`
SWI
`0xc`
PrefetchAbort
`0x10`
handleDataAbort

`0x14`
: Hyp

`0x18`
: IRQ

`0x1c`
: FIQ

`0x80`
: CURRENT_SP0_SYNC

`0x81`
: CURRENT_SP0_IRQ

`0x82`
: CURRENT_SP0_FIQ

`0x83`
: CURRENT_SP0_ABORT

`0x84`
: CURRENT_SPx_SYNC

`0x85`
: CURRENT_SPx_IRQ

`0x86`
: CURRENT_SPx_FIQ

`0x87`
: CURRENT_SPx_ABORT

`0x88`
: LOWER_64_SYNC

`0x89`
: LOWER_64_IRQ

`0x8a`
: LOWER_64_FIQ

`0x8b`
: LOWER_64_ABORT

`0x8c`
: LOWER_32_SYNC

`0x8d`
: LOWER_32_IRQ

`0x8e`
: LOWER_32_FIQ

`0x8f`
: LOWER_32_ABORT

`0xf8`
: Thumb2EE Check Array

`0xfc`
: Thumb2EE Null Check

`TARGET_ISET` `enum`
: The instruction set of the exception handler code..

`0x0`
: ARM

`0x1`
: Thumb

`0x2`
: Jazelle

`0x4`
: AArch64

`CORE_NUM` `unsigned int`
: Core number in a multi processor..

`IS_PHYSICAL` `bool`
: Physical or Virtual exception.

EXCEPTION_END

Every exception completed..

EXCEPTION_RAISE

Every exception raised..

EXCEPTION_RETURN

Branches on leaving exception.. Fields:

PC `unsigned int`
: The address of the branch instruction..

ISET `enum`
: The instructions set of the branch instruction..
  `0x0`
  : ARM
  `0x1`
  : Thumb
  `0x2`
  : Jazelle
  `0x4`
  : AArch64

TARGET_PC `unsigned int`
: The address the instruction branches to..

TARGET_ISET `enum`
: The instructions set after the branch..
  `0x0`
  : ARM
  `0x1`
  : Thumb
  `0x2`
  : Jazelle
  `0x4`
  : AArch64

IS_COND `bool`
: Indicates if this is a conditional branch..

CORE_NUM `unsigned int`
: Core number in a multi processor..

EXCEPTION_START

Every exception started..

EXTERNAL_END

External input-change completed..

EXTERNAL_START

External input-change started..

FIQ_TAKEN

FIQ taken exception.

INST

Every instruction executed.. Fields:

PC unsigned int
The address of the instruction..
OPCODE unsigned int
The opcode of the instruction..
SIZE unsigned int
The size of the instruction in bytes..
MODE enum
The mode the core is in..
0x0
EL0t
0x4
EL1t
0x5
EL1h
0x8
EL2t
0x9
EL2h
0xc
EL3t
0xd
EL3h
0x10
usr
0x11
fiq
0x12
irq
0x13
svc
0x16
mon
0x17
abt
0x1a
hyp
0x1b
und
0x1f
sys
ISET enum
The current instruction set..
0x0
ARM
0x1
Thumb
0x2
Jazelle
0x4
AArch64
PADDR unsigned int
The physical address of the instruction..
NSDESC unsigned int
The physical address non-secure bit..
PADDR2 unsigned int
If different from PADDR, the physical address of the second page of the instruction..

NSDESC2 `unsigned int`
The second page physical address non-secure bit..
NS `unsigned int`
The core's non-secure bit..
ITSTATE `unsigned int`
The current ITSTATE..
INST_COUNT `unsigned int`
The core's instruction counter, starting at 1 for the first instruction..
LOCAL_TIME `unsigned int`
The core's local time..
CORE_NUM `unsigned int`
Core number in a multi processor..
DISASS `string`
Disassembly of instruction..

INST_END

Every instruction completed..

INST_START

Every instruction started.. Fields:

PC `unsigned int`
The address of the conditional instruction..
MODE `enum`
The mode the core is in..
`0x0`
EL0t
`0x4`
EL1t
`0x5`
EL1h
`0x8`
EL2t
`0x9`
EL2h
`0xc`
EL3t
`0xd`
EL3h
`0x10`
usr
`0x11`
fiq
`0x12`
irq
`0x13`
svc
`0x16`
mon
`0x17`
abt
`0x1a`
hyp
`0x1b`
und

0x1f
: sys

ISET enum
: The current instruction set..

0x0
: ARM

0x1
: Thumb

0x2
: Jazelle

0x4
: AArch64

NS enum
: The current Secure State..

0x0
: SEC

0x1
: N_SEC

IRQ_TAKEN

IRQ taken exception.

LOCAL_MONITOR

Local monitor activity. Fields:

State enum
: State of the monitor (Open/Exclusive).

0x0
: Open

0x1
: Exclusive

PADDR unsigned int
: Local Monitor Address.

MEMMAP_DEBUG_READ

Memory mapped reads to the debug registers.. Fields:

ADDR unsigned int
: address.

EXT bool
: Whether access is from an external device (such as the DAP).

VALUE unsigned int
: The value read..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the debug register accessed..

MEMMAP_DEBUG_WRITE

Memory mapped writes to the debug registers.. Fields:

ADDR unsigned int
: address.

EXT bool
: Whether access is from an external device (such as the DAP).

VALUE unsigned int
: The new value written..

UNDEF bool
: The register accessed is undefined..

CORE_NUM unsigned int
: Core number in a multi processor..

REG_NAME string
: Name of the debug register accessed..

MMU_TRANS

Address translation information.. Fields:

CORE_NUM unsigned int
: Core number in a multi processor..

SIDE enum
: Inst / Data..

  0x0
  : Inst

  0x1
  : Data

ASID unsigned int
: Address space identifier..

VMID unsigned int
: Virtual machine identifier..

VADDR unsigned int
: Virtual address of the access..

Hyp bool
: Entry matches in Hyp state only.

nG enum
: Flag indicating whether ASID will be matched.

  0x0
  : Global

  0x1
  : Non-Global

NSDESC enum
: Is secure side supposed to access secure or nonseure memory.

  0x0
  : Secure

  0x1
  : NonSecure

PADDR unsigned int
: Physical address of the access..

PAGESIZE unsigned int
: Page size as log2(size)..

MEMTYPE enum
: Memory type..

  0x0
  : Device-nGnRnE (StronglyOrdered)

  0x1
  : Device-nGnRE (Device)

  0x2
  : Device-nGRE

  0x3
  : Device-GRE

0x4
Normal

OUTERCACHE_TYPE enum
Outer Caching scheme (NC/MB/WA)..
0x0
NonCacheable
0x1
WriteThrough
0x2
WriteBack

OUTERCACHE_RA bool
Is the outer cache allocate on read.

OUTERCACHE_WA bool
Is the outer cache allocate on write.

OUTERCACHE_TRANSIENT bool
Is the outer write-through transient.

INNERCACHE_TYPE enum
Inner Caching scheme (NC/MB/WA)..
0x0
NonCacheable
0x1
WriteThrough
0x2
WriteBack

INNERCACHE_RA bool
Is the inner cache allocate on read.

INNERCACHE_WA bool
Is the inner cache allocate on write.

INNERCACHE_TRANSIENT bool
Is the inner write-through transient.

SH enum
Shareability.
0x0
NonShareable
0x1
InnerShareable
0x2
OuterShareable

MMU_TTB_READ

This event is triggered by reads caused by a translation table walk.. Fields:

SIDE enum
Inst / Data..
0x0
Inst
0x1
Data

LPAE bool
Is this for an LPAE translation.

STAGE unsigned int
Translation stage.

LEVEL unsigned int
Translation table level.

NSreq bool
Non secure request.

NSmem bool
: Non secure memory access.

IPA unsigned int
: For stage 1, the IPA of the read..

PADDR unsigned int
: The physical address of the read..

DATA unsigned int
: The data read..

ABORTED bool
: Was the walk successful..

MMU_TTB_WRITE

This event is triggered by writes caused by a translation table walk.. Fields:

SIDE enum
: Inst / Data..

  0x0
  : Inst

  0x1
  : Data

LPAE bool
: Is this for an LPAE translation.

STAGE unsigned int
: Translation stage.

LEVEL unsigned int
: Translation table level.

NSreq bool
: Non secure request.

NSmem bool
: Non secure memory access.

IPA unsigned int
: For stage 1, the IPA of the write..

PADDR unsigned int
: The physical address of the write..

DATA unsigned int
: The data written..

ABORTED bool
: Was the walk successful..

MODE_CHANGE

Mode change.. Fields:

MODE enum
: The new mode..

  0x0
  : EL0t

  0x4
  : EL1t

  0x5
  : EL1h

  0x8
  : EL2t

  0x9
  : EL2h

  0xc
  : EL3t

`0xd`
: EL3h

`0x10`
: usr

`0x11`
: fiq

`0x12`
: irq

`0x13`
: svc

`0x16`
: mon

`0x17`
: abt

`0x1a`
: hyp

`0x1b`
: und

`0x1f`
: sys

OLD_MODE `enum`
: The old mode..

`0x0`
: EL0t

`0x4`
: EL1t

`0x5`
: EL1h

`0x8`
: EL2t

`0x9`
: EL2h

`0xc`
: EL3t

`0xd`
: EL3h

`0x10`
: usr

`0x11`
: fiq

`0x12`
: irq

`0x13`
: svc

`0x16`
: mon

`0x17`
: abt

`0x1a`
: hyp

`0x1b`
: und

`0x1f`
: sys

CORE_NUM `unsigned int`
: Core number in a multi processor..

NON_SECURE enum
: The new security state.
  0x0
  : SECURE
  0x1
  : NON_SECURE

PERIODIC

Called for every quantum.. Fields:

INST_COUNT unsigned int
: The instruction count of this CPU..

PC unsigned int
: The address of the next instruction to be executed on this CPU..

CORE_NUM unsigned int
: Core number in a multi processor..

PMU_COUNTER_OVERFLOW

PMU counter overflow.. Fields:

INDEX unsigned int
: Counter index (as selected by PMSELR)..

EVENT_ID unsigned int
: Event Identifier.

INTERRUPT bool
: Is interrupt enabled on overflow for this counter.

PREFETCH_MEMORY64

Prefetch from PRFM or PRFUM instructions.. Fields:

PRFOP unsigned int
: Prefetch hint.

VADDR unsigned int
: Virtual address of the location that should be prefetched..

PRELOAD_DATA

Data preload from PLD instruction.. Fields:

VADDR unsigned int
: Virtual address of the data that should be preloaded..

PRE_TTB_READ

This event is triggered before reads caused by a translation table walk.. Fields:

ADDR unsigned int
: The physical address of the read..

RUN_STATE

Run state transition.. Fields:

INST_COUNT unsigned int
: Ticks count at point of transition..

NEW enum
: New run state..
  0x0
  : UNKNOWN

0x1
RUNNING
0x2
HALTED
0x3
STANDBY_WFE
0x4
STANDBY_WFI
0x5
IN_RESET
0x6
DORMANT
0x7
SHUTDOWN
0x8
BARRIER_WAIT

OLD enum
Old run state..
0x0
UNKNOWN
0x1
RUNNING
0x2
HALTED
0x3
STANDBY_WFE
0x4
STANDBY_WFI
0x5
IN_RESET
0x6
DORMANT
0x7
SHUTDOWN
0x8
BARRIER_WAIT

SIGNAL

External signal state change.. Fields:

SIGNAL enum
Signal that changed.
0x0
FIQ
0x1
IRQ
0x2
Reset
0x3
SystemError
0x4
Abort
0x5
Debug
0x6
Halt

`0x7`
: OSUnlockCatch

`0x8`
: VFIQ

`0x9`
: VIRQ

`0xa`
: POReset

`0xb`
: SysPOReset

`0xc`
: DebugReset

`0xd`
: ResetHold

`0xf`
: CP15SDisable

STATE `bool`
: Signal asserted state.

SPSR

SPSR change.. Fields:

VALUE `unsigned int`
: The new SPSR value.

OLD_VALUE `unsigned int`
: The old SPSR value.

MODE `enum`
: Which of the banked SPSR registers is written..

`0x0`
: EL0t

`0x4`
: EL1t

`0x5`
: EL1h

`0x8`
: EL2t

`0x9`
: EL2h

`0xc`
: EL3t

`0xd`
: EL3h

`0x11`
: fiq

`0x12`
: irq

`0x13`
: svc

`0x16`
: mon

`0x17`
: abt

`0x1a`
: hyp

`0x1b`
: und

CORE_NUM unsigned int
: Core number in a multi processor..

START_COMPILE

Compilation started.. Fields:

VADDR unsigned int
: Instruction where compilation begins..

SYNC

Called for every synchronization.. Fields:

INST_COUNT unsigned int
: The instruction count of this CPU..

LOCAL_TIME unsigned int
: The local time of this CPU..

LOCAL_QUANTUM unsigned int
: The local quantum of this CPU..

CORE_NUM unsigned int
: Core number in a multi processor..

SYSCALL

System call instruction executed.. Fields:

VADDR unsigned int
: Instruction that caused the system call..

TYPE enum
: System call type..
  0x0
  : SVC
  0x1
  : HVC
  0x2
  : SMC

IMM unsigned int
: Immediate value of the system call instruction..

SYSREG_READ64

System Coprocessor register read.. Fields:

REGNUM enum
: Internal register number.
  0x4071
  : IC IALLUIS
  0x4075
  : IC IALLU
  0x4078
  : AT S1E1R
  0x4083
  : TLBI VMALLE1IS
  0x4087
  : TLBI VMALLE1
  0x40b0-0x40bf 0x40f0-0x40f3
  : IMP DEF
  0x40f4
  : RAMIDX

0x40f5-0x40ff
IMP DEF
0x4176
DC IVAC
0x4178
AT S1E1W
0x4183
TLBI VAE1IS
0x4187
TLBI VAE1
0x41b0-0x41bf 0x41f0-0x41ff
IMP DEF
0x4276
DC ISW
0x4278
AT S1E0R
0x427a
DC CSW
0x427e
DC CISW
0x4283
TLBI ASIDE1IS
0x4287
TLBI ASIDE1
0x42b0-0x42bf 0x42f0-0x42ff
IMP DEF
0x4378
AT S1E0W
0x4383
TLBI VAAE1IS
0x4387
TLBI VAAE1
0x43b0-0x43bf 0x43f0-0x43ff 0x44b0-0x44bf 0x44f0-0x44ff
IMP DEF
0x4583
TLBI VALE1IS
0x4587
TLBI VALE1
0x45b0-0x45bf 0x45f0-0x45ff 0x46b0-0x46bf 0x46f0-0x46ff
IMP DEF
0x4783
TLBI VAALE1IS
0x4787
TLBI VAALE1
0x47b0-0x47bf 0x47f0-0x47ff 0x48b0-0x48bf 0x48f0-0x48ff 0x49b0-0x49bf
0x49f0-0x49ff 0x4ab0-0x4abf 0x4af0-0x4aff 0x4bb0-0x4bbf 0x4bf0-0x4bff
0x4cb0-0x4cbf 0x4cf0-0x4cff 0x4db0-0x4dbf 0x4df0-0x4dff 0x4eb0-0x4ebf
0x4ef0-0x4eff 0x4fb0-0x4fbf 0x4ff0-0x4fff 0x50b0-0x50bf 0x50f0-0x50ff
0x51b0-0x51bf 0x51f0-0x51ff 0x52b0-0x52bf 0x52f0-0x52ff 0x53b0-0x53bf
0x53f0-0x53ff 0x54b0-0x54bf 0x54f0-0x54ff 0x55b0-0x55bf 0x55f0-0x55ff
0x56b0-0x56bf 0x56f0-0x56ff 0x57b0-0x57bf 0x57f0-0x57ff 0x58b0-0x58bf
0x58f0-0x58ff
IMP DEF
0x5974
DC ZVA

0x5975
IC IVAU
0x597a
DC CVAC
0x597b
DC CVAU
0x597e
DC CIVAC
0x59b0-0x59bf 0x59f0-0x59ff 0x5ab0-0x5abf 0x5af0-0x5aff 0x5bb0-0x5bbf
0x5bf0-0x5bff 0x5cb0-0x5cbf 0x5cf0-0x5cff 0x5db0-0x5dbf 0x5df0-0x5dff
0x5eb0-0x5ebf 0x5ef0-0x5eff 0x5fb0-0x5fbf 0x5ff0-0x5fff
IMP DEF
0x6078
AT S1E2R
0x6083
TLBI ALLE2IS
0x6087
TLBI ALLE2
0x60b0-0x60bf 0x60f0-0x60ff
IMP DEF
0x6178
AT S1E2W
0x6180
TLBI IPAS2E1IS
0x6183
TLBI VAE2IS
0x6184
TLBI IPAS2E1
0x6187
TLBI VAE2
0x61b0-0x61bf 0x61f0-0x61ff 0x62b0-0x62bf 0x62f0-0x62ff 0x63b0-0x63bf
0x63f0-0x63ff
IMP DEF
0x6478
AT S12E1R
0x6483
TLBI ALLE1IS
0x6487
TLBI ALLE1
0x64b0-0x64bf 0x64f0-0x64ff
IMP DEF
0x6578
AT S12E1W
0x6580
TLBI IPAS2LE1IS
0x6583
TLBI VALE2IS
0x6584
TLBI IPAS2LE1
0x6587
TLBI VALE2
0x65b0-0x65bf 0x65f0-0x65ff
IMP DEF
0x6678
AT S12E0R

0x6683
TLBI VMALLS12E1IS
0x6687
TLBI VMALLS12E1
0x66b0-0x66bf 0x66f0-0x66ff
IMP DEF
0x6778
AT S12E0W
0x67b0-0x67bf 0x67f0-0x67ff 0x68b0-0x68bf 0x68f0-0x68ff 0x69b0-0x69bf
0x69f0-0x69ff 0x6ab0-0x6abf 0x6af0-0x6aff 0x6bb0-0x6bbf 0x6bf0-0x6bff
0x6cb0-0x6cbf 0x6cf0-0x6cff 0x6db0-0x6dbf 0x6df0-0x6dff 0x6eb0-0x6ebf
0x6ef0-0x6eff 0x6fb0-0x6fbf 0x6ff0-0x6fff
IMP DEF
0x7078
AT S1E3R
0x7083
TLBI ALLE3IS
0x7087
TLBI ALLE3
0x70b0-0x70bf 0x70f0-0x70ff
IMP DEF
0x7178
AT S1E3W
0x7183
TLBI VAE3IS
0x7187
TLBI VAE3
0x71b0-0x71bf 0x71f0-0x71ff 0x72b0-0x72bf 0x72f0-0x72ff 0x73b0-0x73bf
0x73f0-0x73ff 0x74b0-0x74bf 0x74f0-0x74ff
IMP DEF
0x7583
TLBI VALE3IS
0x7587
TLBI VALE3
0x75b0-0x75bf 0x75f0-0x75ff 0x76b0-0x76bf 0x76f0-0x76ff 0x77b0-0x77bf
0x77f0-0x77ff 0x78b0-0x78bf 0x78f0-0x78ff 0x79b0-0x79bf 0x79f0-0x79ff
0x7ab0-0x7abf 0x7af0-0x7aff 0x7bb0-0x7bbf 0x7bf0-0x7bff 0x7cb0-0x7cbf
0x7cf0-0x7cff 0x7db0-0x7dbf 0x7df0-0x7dff 0x7eb0-0x7ebf 0x7ef0-0x7eff
0x7fb0-0x7fbf 0x7ff0-0x7fff
IMP DEF
0x8002
MDCCINT_EL1
0x8010
MDRAR_EL1
0x8200
OSDTRRX_EL1
0x8202
MDSCR_EL1
0x8203
OSDTRTX_EL1
0x8206
OSECCR_EL1
0x8400
DBGBVR0_EL1
0x8401
DBGBVR1_EL1

0x8402
DBGBVR2_EL1
0x8403
DBGBVR3_EL1
0x8404
DBGBVR4_EL1
0x8405
DBGBVR5_EL1
0x8410
OSLAR_EL1
0x8411
OSLSR_EL1
0x8413
OSDLR_EL1
0x8414
DBGPRCR_EL1
0x8500
DBGBCR0_EL1
0x8501
DBGBCR1_EL1
0x8502
DBGBCR2_EL1
0x8503
DBGBCR3_EL1
0x8504
DBGBCR4_EL1
0x8505
DBGBCR5_EL1
0x8600
DBGWVR0_EL1
0x8601
DBGWVR1_EL1
0x8602
DBGWVR2_EL1
0x8603
DBGWVR3_EL1
0x8678
DBGCLAIMSET_EL1
0x8679
DBGCLAIMCLR_EL1
0x867e
DBGAUTHSTATUS_EL1
0x8700
DBGWCR0_EL1
0x8701
DBGWCR1_EL1
0x8702
DBGWCR2_EL1
0x8703
DBGWCR3_EL1
0x9801
MDCCSR_EL0
0x9804
DBGDTR_EL0
0x9805
DBGDTRxX_EL0

0xa007
DBGVCR32_EL2
0xc000
MIDR_EL1
0xc001
ID_PFR0_EL1
0xc002
ID_ISAR0_EL1
0xc003
MVFR0_EL1
0xc004
ID_AA64PFR0_EL1
0xc005
ID_AA64DFR0_EL1
0xc006
ID_AA64ISAR0_EL1
0xc007
ID_AA64MMFR0_EL1
0xc010
SCTLR_EL1
0xc020
TTBR0_EL1
0xc040
SPSR_EL1
0xc041
SP_EL0
0xc042
SPSel
0xc051
AFSR0_EL1
0xc052
ESR_EL1
0xc060
FAR_EL1
0xc074
PAR_EL1
0xc0a2
MAIR_EL1
0xc0a3
AMAIR_EL1
0xc0b0-0xc0bf
IMP DEF
0xc0c0
VBAR_EL1
0xc0c1
ISR_EL1
0xc0e1
CNTKCTL_EL1
0xc0f0-0xc0f1
IL1D0
0xc0f2-0xc0ff
IMP DEF
0xc101
ID_PFR1_EL1
0xc102
ID_ISAR1_EL1

0xc103
MVFR1_EL1
0xc104
ID_AA64PFR1_EL1
0xc105
ID_AA64DFR1_EL1
0xc106
ID_AA64ISAR1_EL1
0xc107
ID_AA64MMFR1_EL1
0xc110
ACTLR_EL1
0xc120
TTBR1_EL1
0xc140
ELR_EL1
0xc151
AFSR1_EL1
0xc19e
PMINTENSET_EL1
0xc1b0-0xc1bf
IMP DEF
0xc1d0
CONTEXTIDR_EL1
0xc1f0
IL1D1
0xc1f1
DL1D1
0xc1f2-0xc1ff
IMP DEF
0xc201
ID_DFR0_EL1
0xc202
ID_ISAR2_EL1
0xc203
MVFR2_EL1
0xc204-0xc207
Reserved
0xc210
CPACR_EL1
0xc220
TCR_EL1
0xc242
CURREL
0xc29e
PMINTENCLR_EL1
0xc2b0-0xc2bf
IMP DEF
0xc2f0
IL1D2
0xc2f1
DL1D2
0xc2f2-0xc2ff
IMP DEF
0xc301
ID_AFR0_EL1

0xc302
ID_ISAR3_EL1
0xc303-0xc307
Reserved
0xc342
PAN
0xc3b0-0xc3bf
IMP DEF
0xc3f0
IL1D3
0xc3f1
DL1D3
0xc3f2-0xc3ff
IMP DEF
0xc401
ID_MMFR0_EL1
0xc402
ID_ISAR4_EL1
0xc403-0xc404
Reserved
0xc405
ID_AA64AFR0_EL1
0xc406-0xc407
Reserved
0xc4b0-0xc4bf
IMP DEF
0xc4d0
TPIDR_EL1
0xc4f0
IMP DEF
0xc4f1
DL1D4
0xc4f2-0xc4ff
IMP DEF
0xc500
MPIDR_EL1
0xc501
ID_MMFR1_EL1
0xc502
ID_ISAR5_EL1
0xc503-0xc504
Reserved
0xc505
ID_AA64AFR1_EL1
0xc506-0xc507
Reserved
0xc5b0-0xc5bf 0xc5f0-0xc5ff
IMP DEF
0xc600
REVIDR_EL1
0xc601
ID_MMFR2_EL1
0xc602-0xc607
Reserved
0xc6b0-0xc6bf 0xc6f0-0xc6ff
IMP DEF

0xc701
ID_MMFR3_EL1
0xc702-0xc707
Reserved
0xc7b0-0xc7bf 0xc7f0-0xc7ff
IMP DEF
0xc800
CCSIDR_EL1
0xc8b0-0xc8bf
IMP DEF
0xc8f0
L2ACTLR
0xc8f1
IMP DEF
0xc8f2
CPUACTLR_EL1
0xc8f3
CBAR_EL1
0xc8f4-0xc8ff
IMP DEF
0xc900
CLIDR_EL1
0xc9b0-0xc9bf 0xc9f0-0xc9f1
IMP DEF
0xc9f2
CPUECTLR_EL1
0xc9f3-0xc9ff
IMP DEF
0xcab0
L2CTLR
0xcab1-0xcabf 0xcaf0-0xcaf1
IMP DEF
0xcaf2
CPUMERRSR_EL1
0xcaf3-0xcaff
IMP DEF
0xcbb0
L2ECTLR
0xcbb1-0xcbbf 0xcbf0-0xcbf1
IMP DEF
0xcbf2
L2MERRSR_EL1
0xcbf3-0xcbff 0xccb0-0xccbf 0xccf0-0xccff 0xcdb0-0xcdbf 0xcdf0-0xcdff
0xceb0-0xcebf 0xcef0-0xceff
IMP DEF
0xcf00
AIDR_EL1
0xcfb0-0xcfbf 0xcff0-0xcfff
IMP DEF
0xd000
CSSELR_EL1
0xd0b0-0xd0bf 0xd0f0-0xd0ff 0xd1b0-0xd1bf 0xd1f0-0xd1ff 0xd2b0-0xd2bf
0xd2f0-0xd2ff 0xd3b0-0xd3bf 0xd3f0-0xd3ff 0xd4b0-0xd4bf 0xd4f0-0xd4ff
0xd5b0-0xd5bf 0xd5f0-0xd5ff 0xd6b0-0xd6bf 0xd6f0-0xd6ff 0xd7b0-0xd7bf
0xd7f0-0xd7ff
IMP DEF

0xd842
NZCV
0xd844
FPCR
0xd845
DSPSR_EL0
0xd89c
PMCR_EL0
0xd89d
PMCCNTR_EL0
0xd89e
PMUSERENR_EL0
0xd8b0-0xd8bf
IMP DEF
0xd8e0
CNTFRQ_EL0
0xd8e2
CNTP_TVAL_EL0
0xd8e3
CNTV_TVAL_EL0
0xd8e8
PMEVCNTR0_EL0
0xd8ec
PMEVTYPER0_EL0
0xd8f0-0xd8ff
IMP DEF
0xd900
CTR_EL0
0xd942
DAIF
0xd944
FPSR
0xd945
DLR_EL0
0xd99c
PMCNTENSET_EL0
0xd99d
PMXEVTYPER_EL0
0xd9b0-0xd9bf
IMP DEF
0xd9e0
CNTPCT_EL0
0xd9e2
CNTP_CTL_EL0
0xd9e3
CNTV_CTL_EL0
0xd9e8
PMEVCNTR1_EL0
0xd9ec
PMEVTYPER1_EL0
0xd9f0-0xd9ff
IMP DEF
0xda9c
PMCNTENCLR_EL0
0xda9d
PMXEVCNTR_EL0

0xdab0-0xdabf
IMP DEF
0xdad0
TPIDR_EL0
0xdae0
CNTVCT_EL0
0xdae2
CNTP_CVAL_EL0
0xdae3
CNTV_CVAL_EL0
0xdae8
PMEVCNTR2_EL0
0xdaec
PMEVTYPER2_EL0
0xdaf0-0xdaff
IMP DEF
0xdb9c
PMOVSCLR_EL0
0xdb9e
PMOVSSET_EL0
0xdbb0-0xdbbf
IMP DEF
0xdbd0
TPIDRRO_EL0
0xdbe8
PMEVCNTR3_EL0
0xdbec
PMEVTYPER3_EL0
0xdbf0-0xdbff
IMP DEF
0xdc9c
PMSWINC_EL0
0xdcb0-0xdcbf
IMP DEF
0xdce8
PMEVCNTR4_EL0
0xdcec
PMEVTYPER4_EL0
0xdcf0-0xdcff
IMP DEF
0xdd9c
PMSELR_EL0
0xddb0-0xddbf
IMP DEF
0xdde8
PMEVCNTR5_EL0
0xddec
PMEVTYPER5_EL0
0xddf0-0xddff
IMP DEF
0xde9c
PMCEID0_EL0
0xdeb0-0xdebf 0xdef0-0xdeff
IMP DEF
0xdf00
DCZID_EL0

`0xdf9c`
: PMCEID1_EL0

`0xdfb0-0xdfbf`
: IMP DEF

`0xdfef`
: PMCCFILTR_EL0

`0xdff0-0xdfff`
: IMP DEF

`0xe000`
: VPIDR_EL2

`0xe010`
: SCTLR_EL2

`0xe011`
: HCR_EL2

`0xe020`
: TTBR0_EL2

`0xe021`
: VTTBR_EL2

`0xe030`
: DACR32_EL2

`0xe040`
: SPSR_EL2

`0xe041`
: SP_EL1

`0xe043`
: SPSR_IRQ

`0xe051`
: AFSR0_EL2

`0xe052`
: ESR_EL2

`0xe053`
: FPEXC32_EL2

`0xe060`
: FAR_EL2

`0xe0a2`
: MAIR_EL2

`0xe0a3`
: AMAIR_EL2

`0xe0b0-0xe0bf`
: IMP DEF

`0xe0c0`
: VBAR_EL2

`0xe0e1`
: CNTHCTL_EL2

`0xe0e2`
: CNTHP_TVAL_EL2

`0xe0f0-0xe0ff`
: IMP DEF

`0xe110`
: ACTLR_EL2

`0xe111`
: MDCR_EL2

`0xe140`
: ELR_EL2

`0xe143`
: SPSR_ABT

0xe150
IFSR32_EL2
0xe151
AFSR1_EL2
0xe1b0-0xe1bf
IMP DEF
0xe1e2
CNTHP_CTL_EL2
0xe1f0-0xe1ff
IMP DEF
0xe211
CPTR_EL2
0xe220
TCR_EL2
0xe221
VTCR_EL2
0xe243
SPSR_UND
0xe2b0-0xe2bf
IMP DEF
0xe2d0
TPIDR_EL2
0xe2e2
CNTHP_CVAL_EL2
0xe2f0-0xe2ff
IMP DEF
0xe311
HSTR_EL2
0xe343
SPSR_FIQ
0xe3b0-0xe3bf
IMP DEF
0xe3e0
CNTVOFF_EL2
0xe3f0-0xe3ff
IMP DEF
0xe460
HPFAR_EL2
0xe4b0-0xe4bf 0xe4f0-0xe4ff
IMP DEF
0xe500
VMPIDR_EL2
0xe5b0-0xe5bf 0xe5f0-0xe5ff 0xe6b0-0xe6bf 0xe6f0-0xe6ff
IMP DEF
0xe711
HACR_EL2
0xe7b0-0xe7bf 0xe7f0-0xe7ff 0xe8b0-0xe8bf 0xe8f0-0xe8ff 0xe9b0-0xe9bf
0xe9f0-0xe9ff 0xeab0-0xeabf 0xeaf0-0xeaff 0xebb0-0xebbf 0xebf0-0xebff
0xecb0-0xecbf 0xecf0-0xecff 0xedb0-0xedbf 0xedf0-0xedff 0xeeb0-0xeebf
0xeef0-0xeeff 0xefb0-0xefbf 0xeff0-0xefff
IMP DEF
0xf010
SCTLR_EL3
0xf011
SCR_EL3

0xf020
TTBR0_EL3
0xf040
SPSR_EL3
0xf041
SP_EL2
0xf051
AFSR0_EL3
0xf052
ESR_EL3
0xf060
FAR_EL3
0xf0a2
MAIR_EL3
0xf0a3
AMAIR_EL3
0xf0b0-0xf0bf
IMP DEF
0xf0c0
VBAR_EL3
0xf0f0-0xf0ff
IMP DEF
0xf110
ACTLR_EL3
0xf111
SDER32_EL3
0xf113
MDCR_EL3
0xf140
ELR_EL3
0xf151
AFSR1_EL3
0xf1b0-0xf1bf
IMP DEF
0xf1c0
RVBAR_EL3
0xf1f0-0xf1ff
IMP DEF
0xf211
CPTR_EL3
0xf220
TCR_EL3
0xf2b0-0xf2bf
IMP DEF
0xf2c0
RMR_EL3
0xf2d0
TPIDR_EL3
0xf2f0-0xf2ff 0xf3b0-0xf3bf 0xf3f0-0xf3ff 0xf4b0-0xf4bf 0xf4f0-0xf4ff
0xf5b0-0xf5bf 0xf5f0-0xf5ff 0xf6b0-0xf6bf 0xf6f0-0xf6ff 0xf7b0-0xf7bf
0xf7f0-0xf7ff 0xf8b0-0xf8bf
IMP DEF
0xf8e2
CNTPS_TVAL_EL1
0xf8f0-0xf8ff 0xf9b0-0xf9bf
IMP DEF

`0xf9e2`
CNTPS_CTL_EL1
`0xf9f0-0xf9ff 0xfab0-0xfabf`
IMP DEF
`0xfae2`
CNTPS_CVAL_EL1
`0xfaf0-0xfaff 0xfbb0-0xfbbf 0xfbf0-0xfbff 0xfcb0-0xfcbf 0xfcf0-0xfcff 0xfdb0-0xfdbf 0xfdf0-0xfdff 0xfeb0-0xfebf 0xfef0-0xfeff 0xffb0-0xffbf 0xfff0-0xffff`
IMP DEF

VALUE `unsigned int`
The value read..

UNDEF `bool`
The register accessed is undefined..

CORE_NUM `unsigned int`
Core number in a multi processor..

SYSREG_UPDATE32

Triggers when the system updates a register.. Fields:

REG `enum`
Register number..
`0x0`
SCR
`0x1`
NSACR
`0x2`
MVBAR
`0x3`
DFSR_S
`0x4`
DFAR_S
`0x5`
IFSR_S
`0x6`
IFAR_S
`0x7`
AIFSR_S
`0x8`
ADFSR_S
`0x9`
SCTLR_S
`0xa`
TPIDRURW_S
`0xb`
TPIDRUR0_S
`0xc`
TPIDRPRW_S
`0xd`
VBAR_S
`0xe`
ACTLR_S
`0xf`
DFSR_NS
`0x10`
DFAR_NS

0x11
IFSR_NS
0x12
IFAR_NS
0x13
AIFSR_NS
0x14
ADFSR_NS
0x15
SCTLR_NS
0x16
TPIDRURW_NS
0x17
TPIDRURO_NS
0x18
TPIDRPRW_NS
0x19
VBAR_NS
0x1a
ACTLR_NS
0x1b
ELR
0x1c
HCR
0x1d
HDCR
0x1e
HCPTR
0x1f
HSTR
0x20
HDFAR
0x21
HIFAR
0x22
HPFAR
0x23
HSCTLR
0x24
HSR
0x25
HTPIDR
0x26
HVBAR
0x27
VPIDR
0x28
VMPIDR
0x29
HCR2
0x2a
HACTLR
0x2b
MPIDR
0x2c
CPACR

0x2d
: TEECR

0x2e
: TEEHBR

0x2f
: WFAR

0x30
: ISR

0x31
: SPSR_svc

0x32
: SPSR_irq

0x33
: SPSR_fiq

0x34
: SPSR_abt

0x35
: SPSR_und

0x36
: SPSR_mon

0x37
: SPSR_hyp

0x38
: VDFSR

0x39
: SDER

0x3a
: CNTFRQ_EL0

0x3b
: CNTKCTL_EL1

0x3c
: CNTHCTL_EL2

0x3d
: CNTPS_CTL_EL1

0x3e
: CNTP_CTL_EL0

0x3f
: CNTV_CTL_EL0

0x40
: CNTHP_CTL_EL2

0x41
: DBGDSCRint

0x42
: DBGDSCRext

0x43
: DBGDTRRXext

0x44
: DBGDTRRXintorDBGDTRTXint

0x45
: DBGDTRTXext

0x46
: DBGWFAR

0x47
: DBGVCR

0x48
: EDECR

`0x49`
: EDITR

`0x4a`
: EDPCSRlo

`0x4b`
: EDCIDSR

`0x4c`
: EDVIDSR

`0x4d`
: EDRCR

`0x4e`
: DBGBVR0

`0x4f`
: DBGBVR1

`0x50`
: DBGBVR2

`0x51`
: DBGBVR3

`0x52`
: DBGBVR4

`0x53`
: DBGBVR5

`0x54`
: DBGBCR0

`0x55`
: DBGBCR1

`0x56`
: DBGBCR2

`0x57`
: DBGBCR3

`0x58`
: DBGBCR4

`0x59`
: DBGBCR5

`0x5a`
: DBGWVR0

`0x5b`
: DBGWVR1

`0x5c`
: DBGWVR2

`0x5d`
: DBGWVR3

`0x5e`
: DBGWCR0

`0x5f`
: DBGWCR1

`0x60`
: DBGWCR2

`0x61`
: DBGWCR3

`0x62`
: DBGDRAR

`0x63`
: DBGDSAR

`0x64`
: DBGOSLAR

0x65
OSLSR_EL1
0x66
EDPRCR
0x67
EDPRSR
0x68
MIDR
0x69
RESERVED (MIDR1)
0x6a
RESERVED (MIDR2)
0x6b
RESERVED (MIDR3)
0x6c
RESERVED (MIDR4)
0x6d
RESERVED (MIDR5)
0x6e
RESERVED (MIDR6)
0x6f
RESERVED (MIDR7)
0x70
ID_AA64PFR0[31:0]
0x71
ID_AA64PFR0[63:32]
0x72
ID_AA64DFR0[31:0]
0x73
ID_AA64DFR0[63:32]
0x74
ID_AA64ISAR0[31:0]
0x75
ID_AA64ISAR0[63:32]
0x76
ID_AA64MFR0[31:0]
0x77
ID_AA64MFR0[63:32]
0x78
ID_AA64PFR1[31:0]
0x79
ID_AA64PFR1[63:32]
0x7a
ID_AA64DFR1[31:0]
0x7b
ID_AA64DFR1[63:32]
0x7c
ID_AA64ISAR1[31:0]
0x7d
ID_AA64ISAR1[63:32]
0x7e
ID_AA64MMFR1[31:0]
0x7f
ID_AA64MMFR1[63:32]
0x80
DBGITCTRL

0x81
: DBGCLAIMSET

0x82
: DBGCLAIMCLR

0x83
: EDLAR

0x84
: EDLSR

0x85
: DBGAUTHSTATUS

0x86
: EDDEVTYPE

0x87
: EDPID0

0x88
: EDPID1

0x89
: EDPID2

0x8a
: EDPID3

0x8b
: EDPID4

0x8c
: RESERVED (EDPID5)

0x8d
: RESERVED (EDPID6)

0x8e
: RESERVED (EDPID7)

0x8f
: EDCID0

0x90
: EDCID1

0x91
: EDCID2

0x92
: EDCID3

0x93
: TEECR

0x94
: TEEHBR

0x95
: DBGBXVR0

0x96
: DBGBXVR1

0x97
: DBGBXVR2

0x98
: DBGBXVR3

0x99
: DBGBXVR4

0x9a
: DBGBXVR5

0x9b
: DBGDIDR

0x9c
: EDDEVID

`0x9d`
: EDDEVID1

`0x9e`
: EDDEVID2

`0x9f`
: DBGOSDLR

`0xa0`
: DBGOSSRR

`0xa1`
: EDPCSRhi

`0xa2`
: DCCINT

`0xa3`
: EDESR

`0xa4`
: EDWARlo

`0xa5`
: EDWARhi

`0xa6`
: EDACR

`0xa7`
: EDECCR

`0xa8`
: DBGWXVR0

`0xa9`
: DBGWXVR1

`0xaa`
: DBGWXVR2

`0xab`
: DBGWXVR3

`0xac`
: EDDEVAFF0

`0xad`
: EDDEVAFF1

`0xae`
: EDDEVARCH

`0xaf`
: EDSCR

`0xb0`
: MDCCSR_EL0

`0xb1`
: MDSCR_EL1

`0xb2`
: PMCID1SR

`0xb3`
: PMVIDSR

`0xb4`
: PMCID1SR

`0xb5`
: PRRR_S

`0xb6`
: NMRR_S

`0xb7`
: DACR_S

`0xb8`
: FCSE_S

0xb9
: TTBCR_S

0xba
: TTBR0_S

0xbb
: TTBR1_S

0xbc
: CONTEXTIDR_S

0xbd
: PAR_S

0xbe
: PRRR_NS

0xbf
: NMRR_NS

0xc0
: DACR_NS

0xc1
: FCSE_NS

0xc2
: TTBCR_NS

0xc3
: TTBR0_NS

0xc4
: TTBR1_NS

0xc5
: CONTEXTIDR_NS

0xc6
: PAR_NS

0xc7
: AMAIR0_S

0xc8
: AMAIR1_S

0xc9
: AMAIR0_NS

0xca
: AMAIR1_NS

0xcb
: HMAIR0

0xcc
: HMAIR1

0xcd
: HAMAIR0

0xce
: HAMAIR1

0xcf
: HTCR

0xd0
: VTCR

0xd1
: TCR_EL3

0xd2
: TCR_EL2

0xd3
: VTCR_EL2

0xd4
: DACR32_EL2

0xd5
: CTICONTROL

0xd6
: CTIINTACK

0xd7
: CTIAPPSET

0xd8
: CTICHANPULSE

0xd9
: CTITRIGINSTATUS

0xda
: CTITRIGOUTSTATUS

0xdb
: CTICHINSTATUS

0xdc
: CTICHOUTSTATUS

0xdd
: CTIGATE

0xde
: CTIITCTRL

0xdf
: CTILAR

0xe0
: UnknownCTI

0xe1
: CTIINEN0

0xe2
: CTIINEN1

0xe3
: CTIINEN2

0xe4
: CTIINEN3

0xe5
: CTIINEN4

0xe6
: CTIINEN5

0xe7
: CTIINEN6

0xe8
: CTIINEN7

0xe9
: CTIOUTEN0

0xea
: CTIOUTEN1

0xeb
: CTIOUTEN2

0xec
: CTIOUTEN3

0xed
: CTIOUTEN4

0xee
: CTIOUTEN5

0xef
: CTIOUTEN6

0xf0
: CTIOUTEN7

0xf1
: CCSELR_EL1

0xf2
: VPIDR_EL2

0xf3
: SCTLR_EL1

0xf4
: SCTLR_EL2

0xf5
: SCTLR_EL3

0xf6
: ACTLR_EL1

0xf7
: ACTLR_EL2

0xf8
: ACTLR_EL3

0xf9
: CPACR_EL1

0xfa
: CPTR_EL2

0xfb
: CPTR_EL3

0xfc
: SCR_EL3

0xfd
: MDCR_EL2

0xfe
: MDCR_EL3

0xff
: HSTR_EL2

0x100
: HACR_EL2

0x101
: AFSR0_EL1

0x102
: AFSR1_EL1

0x103
: AFSR0_EL2

0x104
: AFSR1_EL2

0x105
: AFSR0_EL3

0x106
: AFSR1_EL3

0x107
: ESR_EL1

0x108
: ESR_EL2

0x109
: ESR_EL3

0x10a
: CONTEXTIDR_EL1

0x10b
: TEECR32_EL1

0x10c
: DACR32_EL2

`0x10d`
IFSR32_EL2
`0x10e`
TEEHBR32_EL1
`0x10f`
SDER32_EL3
`0x110`
SPSR_EL1
`0x111`
SPSR_EL2
`0x112`
SPSR_EL3
`0x113`
FPSR
`0x114`
FPCR
`0x115`
DSPSR_EL0
`0x116`
DSPSR
`0x117`
DLR
`0x118`
DISR_EL1
`0x119`
VDISR_EL2
`0x11a`
VSESR_EL2
`0x11b`
ERRSELR_EL1
`0x11c`
ERXSTATUS_EL1

VALUE `unsigned int`
Value written to the register..

UNKNOWN `unsigned int`
Bits of the register which became unknown..

SYSREG_UPDATE64

Triggers when the system updates a register.. Fields:

REG `enum`
Register number..
`0x0`
PMPCSR
`0x1`
TTBR0_64_S
`0x2`
TTBR1_64_S
`0x3`
PAR_64_S
`0x4`
TTBR0_64_NS
`0x5`
TTBR1_64_NS
`0x6`
PAR_64_NS

`0x7`
: HTTBR

`0x8`
: VTTBR

`0x9`
: TTBR0_EL3

`0xa`
: MAIR_EL3

`0xb`
: AMAIR_EL3

`0xc`
: TTBR0_EL2

`0xd`
: MAIR_EL2

`0xe`
: AMAIR_EL2

`0xf`
: VTTBR_EL2

`0x10`
: TCR_EL1

`0x11`
: TTBR0_EL1

`0x12`
: TTBR1_EL1

`0x13`
: MAIR_EL1

`0x14`
: AMAIR_EL1

`0x15`
: PAR_EL1

`0x16`
: CONTEXTIDR_EL1

`0x17`
: VMPIDR_EL2

`0x18`
: HCR_EL2

`0x19`
: FAR_EL1

`0x1a`
: FAR_EL2

`0x1b`
: FAR_EL3

`0x1c`
: HPFAR_EL2

`0x1d`
: MAIR_EL1

`0x1e`
: MAIR_EL2

`0x1f`
: MAIR_EL3

`0x20`
: VBAR_EL1

`0x21`
: VBAR_EL2

`0x22`
: VBAR_EL3

0x23
: TPIDR_EL0

0x24
: TPIDRRO_EL0

0x25
: TPIDR_EL1

0x26
: TPIDR_EL2

0x27
: TPIDR_EL3

0x28
: ELR_EL1

0x29
: ELR_EL2

0x2a
: ELR_EL3

0x2b
: DLR_EL0

0x2c
: MPIDR_EL1

0x2d
: ERXFR_EL1

0x2e
: ERXCTLR_EL1

0x2f
: ERXADDR_EL1

0x30
: ERXMISC0_EL1

0x31
: ERXMISC1_EL1

VALUE unsigned int
: Value written to the register..

UNKNOWN unsigned int
: Bits of the register which became unknown..

SYSREG_WRITE64

System Coprocessor register write.. Fields:

REGNUM enum
: Internal register number.

0x4071
: IC IALLUIS

0x4075
: IC IALLU

0x4078
: AT S1E1R

0x4083
: TLBI VMALLE1IS

0x4087
: TLBI VMALLE1

0x40b0-0x40bf 0x40f0-0x40f3
: IMP DEF

0x40f4
: RAMIDX

0x40f5-0x40ff
: IMP DEF

0x4176
DC IVAC
0x4178
AT S1E1W
0x4183
TLBI VAE1IS
0x4187
TLBI VAE1
0x41b0-0x41bf 0x41f0-0x41ff
IMP DEF
0x4276
DC ISW
0x4278
AT S1E0R
0x427a
DC CSW
0x427e
DC CISW
0x4283
TLBI ASIDE1IS
0x4287
TLBI ASIDE1
0x42b0-0x42bf 0x42f0-0x42ff
IMP DEF
0x4378
AT S1E0W
0x4383
TLBI VAAE1IS
0x4387
TLBI VAAE1
0x43b0-0x43bf 0x43f0-0x43ff 0x44b0-0x44bf 0x44f0-0x44ff
IMP DEF
0x4583
TLBI VALE1IS
0x4587
TLBI VALE1
0x45b0-0x45bf 0x45f0-0x45ff 0x46b0-0x46bf 0x46f0-0x46ff
IMP DEF
0x4783
TLBI VAALE1IS
0x4787
TLBI VAALE1
0x47b0-0x47bf 0x47f0-0x47ff 0x48b0-0x48bf 0x48f0-0x48ff 0x49b0-0x49bf
0x49f0-0x49ff 0x4ab0-0x4abf 0x4af0-0x4aff 0x4bb0-0x4bbf 0x4bf0-0x4bff
0x4cb0-0x4cbf 0x4cf0-0x4cff 0x4db0-0x4dbf 0x4df0-0x4dff 0x4eb0-0x4ebf
0x4ef0-0x4eff 0x4fb0-0x4fbf 0x4ff0-0x4fff 0x50b0-0x50bf 0x50f0-0x50ff
0x51b0-0x51bf 0x51f0-0x51ff 0x52b0-0x52bf 0x52f0-0x52ff 0x53b0-0x53bf
0x53f0-0x53ff 0x54b0-0x54bf 0x54f0-0x54ff 0x55b0-0x55bf 0x55f0-0x55ff
0x56b0-0x56bf 0x56f0-0x56ff 0x57b0-0x57bf 0x57f0-0x57ff 0x58b0-0x58bf
0x58f0-0x58ff
IMP DEF
0x5974
DC ZVA
0x5975
IC IVAU

0x597a
DC CVAC
0x597b
DC CVAU
0x597e
DC CIVAC
0x59b0-0x59bf 0x59f0-0x59ff 0x5ab0-0x5abf 0x5af0-0x5aff 0x5bb0-0x5bbf 0x5bf0-0x5bff 0x5cb0-0x5cbf 0x5cf0-0x5cff 0x5db0-0x5dbf 0x5df0-0x5dff 0x5eb0-0x5ebf 0x5ef0-0x5eff 0x5fb0-0x5fbf 0x5ff0-0x5fff
IMP DEF
0x6078
AT S1E2R
0x6083
TLBI ALLE2IS
0x6087
TLBI ALLE2
0x60b0-0x60bf 0x60f0-0x60ff
IMP DEF
0x6178
AT S1E2W
0x6180
TLBI IPAS2E1IS
0x6183
TLBI VAE2IS
0x6184
TLBI IPAS2E1
0x6187
TLBI VAE2
0x61b0-0x61bf 0x61f0-0x61ff 0x62b0-0x62bf 0x62f0-0x62ff 0x63b0-0x63bf 0x63f0-0x63ff
IMP DEF
0x6478
AT S12E1R
0x6483
TLBI ALLE1IS
0x6487
TLBI ALLE1
0x64b0-0x64bf 0x64f0-0x64ff
IMP DEF
0x6578
AT S12E1W
0x6580
TLBI IPAS2LE1IS
0x6583
TLBI VALE2IS
0x6584
TLBI IPAS2LE1
0x6587
TLBI VALE2
0x65b0-0x65bf 0x65f0-0x65ff
IMP DEF
0x6678
AT S12E0R
0x6683
TLBI VMALLS12E1IS

0x6687
TLBI VMALLS12E1
0x66b0-0x66bf 0x66f0-0x66ff
IMP DEF
0x6778
AT S12E0W
0x67b0-0x67bf 0x67f0-0x67ff 0x68b0-0x68bf 0x68f0-0x68ff 0x69b0-0x69bf
0x69f0-0x69ff 0x6ab0-0x6abf 0x6af0-0x6aff 0x6bb0-0x6bbf 0x6bf0-0x6bff
0x6cb0-0x6cbf 0x6cf0-0x6cff 0x6db0-0x6dbf 0x6df0-0x6dff 0x6eb0-0x6ebf
0x6ef0-0x6eff 0x6fb0-0x6fbf 0x6ff0-0x6fff
IMP DEF
0x7078
AT S1E3R
0x7083
TLBI ALLE3IS
0x7087
TLBI ALLE3
0x70b0-0x70bf 0x70f0-0x70ff
IMP DEF
0x7178
AT S1E3W
0x7183
TLBI VAE3IS
0x7187
TLBI VAE3
0x71b0-0x71bf 0x71f0-0x71ff 0x72b0-0x72bf 0x72f0-0x72ff 0x73b0-0x73bf
0x73f0-0x73ff 0x74b0-0x74bf 0x74f0-0x74ff
IMP DEF
0x7583
TLBI VALE3IS
0x7587
TLBI VALE3
0x75b0-0x75bf 0x75f0-0x75ff 0x76b0-0x76bf 0x76f0-0x76ff 0x77b0-0x77bf
0x77f0-0x77ff 0x78b0-0x78bf 0x78f0-0x78ff 0x79b0-0x79bf 0x79f0-0x79ff
0x7ab0-0x7abf 0x7af0-0x7aff 0x7bb0-0x7bbf 0x7bf0-0x7bff 0x7cb0-0x7cbf
0x7cf0-0x7cff 0x7db0-0x7dbf 0x7df0-0x7dff 0x7eb0-0x7ebf 0x7ef0-0x7eff
0x7fb0-0x7fbf 0x7ff0-0x7fff
IMP DEF
0x8002
MDCCINT_EL1
0x8010
MDRAR_EL1
0x8200
OSDTRRX_EL1
0x8202
MDSCR_EL1
0x8203
OSDTRTX_EL1
0x8206
OSECCR_EL1
0x8400
DBGBVR0_EL1
0x8401
DBGBVR1_EL1
0x8402
DBGBVR2_EL1

0x8403
DBGBVR3_EL1
0x8404
DBGBVR4_EL1
0x8405
DBGBVR5_EL1
0x8410
OSLAR_EL1
0x8411
OSLSR_EL1
0x8413
OSDLR_EL1
0x8414
DBGPRCR_EL1
0x8500
DBGBCR0_EL1
0x8501
DBGBCR1_EL1
0x8502
DBGBCR2_EL1
0x8503
DBGBCR3_EL1
0x8504
DBGBCR4_EL1
0x8505
DBGBCR5_EL1
0x8600
DBGWVR0_EL1
0x8601
DBGWVR1_EL1
0x8602
DBGWVR2_EL1
0x8603
DBGWVR3_EL1
0x8678
DBGCLAIMSET_EL1
0x8679
DBGCLAIMCLR_EL1
0x867e
DBGAUTHSTATUS_EL1
0x8700
DBGWCR0_EL1
0x8701
DBGWCR1_EL1
0x8702
DBGWCR2_EL1
0x8703
DBGWCR3_EL1
0x9801
MDCCSR_EL0
0x9804
DBGDTR_EL0
0x9805
DBGDTRxX_EL0
0xa007
DBGVCR32_EL2

0xc000
MIDR_EL1
0xc001
ID_PFR0_EL1
0xc002
ID_ISAR0_EL1
0xc003
MVFR0_EL1
0xc004
ID_AA64PFR0_EL1
0xc005
ID_AA64DFR0_EL1
0xc006
ID_AA64ISAR0_EL1
0xc007
ID_AA64MMFR0_EL1
0xc010
SCTLR_EL1
0xc020
TTBR0_EL1
0xc040
SPSR_EL1
0xc041
SP_EL0
0xc042
SPSel
0xc051
AFSR0_EL1
0xc052
ESR_EL1
0xc060
FAR_EL1
0xc074
PAR_EL1
0xc0a2
MAIR_EL1
0xc0a3
AMAIR_EL1
0xc0b0-0xc0bf
IMP DEF
0xc0c0
VBAR_EL1
0xc0c1
ISR_EL1
0xc0e1
CNTKCTL_EL1
0xc0f0-0xc0f1
IL1D0
0xc0f2-0xc0ff
IMP DEF
0xc101
ID_PFR1_EL1
0xc102
ID_ISAR1_EL1
0xc103
MVFR1_EL1

0xc104
ID_AA64PFR1_EL1
0xc105
ID_AA64DFR1_EL1
0xc106
ID_AA64ISAR1_EL1
0xc107
ID_AA64MMFR1_EL1
0xc110
ACTLR_EL1
0xc120
TTBR1_EL1
0xc140
ELR_EL1
0xc151
AFSR1_EL1
0xc19e
PMINTENSET_EL1
0xc1b0-0xc1bf
IMP DEF
0xc1d0
CONTEXTIDR_EL1
0xc1f0
IL1D1
0xc1f1
DL1D1
0xc1f2-0xc1ff
IMP DEF
0xc201
ID_DFR0_EL1
0xc202
ID_ISAR2_EL1
0xc203
MVFR2_EL1
0xc204-0xc207
Reserved
0xc210
CPACR_EL1
0xc220
TCR_EL1
0xc242
CURREL
0xc29e
PMINTENCLR_EL1
0xc2b0-0xc2bf
IMP DEF
0xc2f0
IL1D2
0xc2f1
DL1D2
0xc2f2-0xc2ff
IMP DEF
0xc301
ID_AFR0_EL1
0xc302
ID_ISAR3_EL1

0xc303-0xc307
: Reserved

0xc342
: PAN

0xc3b0-0xc3bf
: IMP DEF

0xc3f0
: IL1D3

0xc3f1
: DL1D3

0xc3f2-0xc3ff
: IMP DEF

0xc401
: ID_MMFR0_EL1

0xc402
: ID_ISAR4_EL1

0xc403-0xc404
: Reserved

0xc405
: ID_AA64AFR0_EL1

0xc406-0xc407
: Reserved

0xc4b0-0xc4bf
: IMP DEF

0xc4d0
: TPIDR_EL1

0xc4f0
: IMP DEF

0xc4f1
: DL1D4

0xc4f2-0xc4ff
: IMP DEF

0xc500
: MPIDR_EL1

0xc501
: ID_MMFR1_EL1

0xc502
: ID_ISAR5_EL1

0xc503-0xc504
: Reserved

0xc505
: ID_AA64AFR1_EL1

0xc506-0xc507
: Reserved

0xc5b0-0xc5bf 0xc5f0-0xc5ff
: IMP DEF

0xc600
: REVIDR_EL1

0xc601
: ID_MMFR2_EL1

0xc602-0xc607
: Reserved

0xc6b0-0xc6bf 0xc6f0-0xc6ff
: IMP DEF

0xc701
: ID_MMFR3_EL1

0xc702-0xc707
Reserved
0xc7b0-0xc7bf 0xc7f0-0xc7ff
IMP DEF
0xc800
CCSIDR_EL1
0xc8b0-0xc8bf
IMP DEF
0xc8f0
L2ACTLR
0xc8f1
IMP DEF
0xc8f2
CPUACTLR_EL1
0xc8f3
CBAR_EL1
0xc8f4-0xc8ff
IMP DEF
0xc900
CLIDR_EL1
0xc9b0-0xc9bf 0xc9f0-0xc9f1
IMP DEF
0xc9f2
CPUECTLR_EL1
0xc9f3-0xc9ff
IMP DEF
0xcab0
L2CTLR
0xcab1-0xcabf 0xcaf0-0xcaf1
IMP DEF
0xcaf2
CPUMERRSR_EL1
0xcaf3-0xcaff
IMP DEF
0xcbb0
L2ECTLR
0xcbb1-0xcbbf 0xcbf0-0xcbf1
IMP DEF
0xcbf2
L2MERRSR_EL1
0xcbf3-0xcbff 0xccb0-0xccbf 0xccf0-0xccff 0xcdb0-0xcdbf 0xcdf0-0xcdff
0xceb0-0xcebf 0xcef0-0xceff
IMP DEF
0xcf00
AIDR_EL1
0xcfb0-0xcfbf 0xcff0-0xcfff
IMP DEF
0xd000
CSSELR_EL1
0xd0b0-0xd0bf 0xd0f0-0xd0ff 0xd1b0-0xd1bf 0xd1f0-0xd1ff 0xd2b0-0xd2bf
0xd2f0-0xd2ff 0xd3b0-0xd3bf 0xd3f0-0xd3ff 0xd4b0-0xd4bf 0xd4f0-0xd4ff
0xd5b0-0xd5bf 0xd5f0-0xd5ff 0xd6b0-0xd6bf 0xd6f0-0xd6ff 0xd7b0-0xd7bf
0xd7f0-0xd7ff
IMP DEF
0xd842
NZCV

0xd844
FPCR
0xd845
DSPSR_EL0
0xd89c
PMCR_EL0
0xd89d
PMCCNTR_EL0
0xd89e
PMUSERENR_EL0
0xd8b0-0xd8bf
IMP DEF
0xd8e0
CNTFRQ_EL0
0xd8e2
CNTP_TVAL_EL0
0xd8e3
CNTV_TVAL_EL0
0xd8e8
PMEVCNTR0_EL0
0xd8ec
PMEVTYPER0_EL0
0xd8f0-0xd8ff
IMP DEF
0xd900
CTR_EL0
0xd942
DAIF
0xd944
FPSR
0xd945
DLR_EL0
0xd99c
PMCNTENSET_EL0
0xd99d
PMXEVTYPER_EL0
0xd9b0-0xd9bf
IMP DEF
0xd9e0
CNTPCT_EL0
0xd9e2
CNTP_CTL_EL0
0xd9e3
CNTV_CTL_EL0
0xd9e8
PMEVCNTR1_EL0
0xd9ec
PMEVTYPER1_EL0
0xd9f0-0xd9ff
IMP DEF
0xda9c
PMCNTENCLR_EL0
0xda9d
PMXEVCNTR_EL0
0xdab0-0xdabf
IMP DEF

`0xdad0`
: TPIDR_EL0

`0xdae0`
: CNTVCT_EL0

`0xdae2`
: CNTP_CVAL_EL0

`0xdae3`
: CNTV_CVAL_EL0

`0xdae8`
: PMEVCNTR2_EL0

`0xdaec`
: PMEVTYPER2_EL0

`0xdaf0-0xdaff`
: IMP DEF

`0xdb9c`
: PMOVSCLR_EL0

`0xdb9e`
: PMOVSSET_EL0

`0xdbb0-0xdbbf`
: IMP DEF

`0xdbd0`
: TPIDRRO_EL0

`0xdbe8`
: PMEVCNTR3_EL0

`0xdbec`
: PMEVTYPER3_EL0

`0xdbf0-0xdbff`
: IMP DEF

`0xdc9c`
: PMSWINC_EL0

`0xdcb0-0xdcbf`
: IMP DEF

`0xdce8`
: PMEVCNTR4_EL0

`0xdcec`
: PMEVTYPER4_EL0

`0xdcf0-0xdcff`
: IMP DEF

`0xdd9c`
: PMSELR_EL0

`0xddb0-0xddbf`
: IMP DEF

`0xdde8`
: PMEVCNTR5_EL0

`0xddec`
: PMEVTYPER5_EL0

`0xddf0-0xddff`
: IMP DEF

`0xde9c`
: PMCEID0_EL0

`0xdeb0-0xdebf 0xdef0-0xdeff`
: IMP DEF

`0xdf00`
: DCZID_EL0

`0xdf9c`
: PMCEID1_EL0

`0xdfb0-0xdfbf`
: IMP DEF

`0xdfef`
: PMCCFILTR_EL0

`0xdff0-0xdfff`
: IMP DEF

`0xe000`
: VPIDR_EL2

`0xe010`
: SCTLR_EL2

`0xe011`
: HCR_EL2

`0xe020`
: TTBR0_EL2

`0xe021`
: VTTBR_EL2

`0xe030`
: DACR32_EL2

`0xe040`
: SPSR_EL2

`0xe041`
: SP_EL1

`0xe043`
: SPSR_IRQ

`0xe051`
: AFSR0_EL2

`0xe052`
: ESR_EL2

`0xe053`
: FPEXC32_EL2

`0xe060`
: FAR_EL2

`0xe0a2`
: MAIR_EL2

`0xe0a3`
: AMAIR_EL2

`0xe0b0-0xe0bf`
: IMP DEF

`0xe0c0`
: VBAR_EL2

`0xe0e1`
: CNTHCTL_EL2

`0xe0e2`
: CNTHP_TVAL_EL2

`0xe0f0-0xe0ff`
: IMP DEF

`0xe110`
: ACTLR_EL2

`0xe111`
: MDCR_EL2

`0xe140`
: ELR_EL2

`0xe143`
: SPSR_ABT

`0xe150`
: IFSR32_EL2

```
0xe151
       AFSR1_EL2
0xe1b0-0xe1bf
       IMP DEF
0xe1e2
       CNTHP_CTL_EL2
0xe1f0-0xe1ff
       IMP DEF
0xe211
       CPTR_EL2
0xe220
       TCR_EL2
0xe221
       VTCR_EL2
0xe243
       SPSR_UND
0xe2b0-0xe2bf
       IMP DEF
0xe2d0
       TPIDR_EL2
0xe2e2
       CNTHP_CVAL_EL2
0xe2f0-0xe2ff
       IMP DEF
0xe311
       HSTR_EL2
0xe343
       SPSR_FIQ
0xe3b0-0xe3bf
       IMP DEF
0xe3e0
       CNTVOFF_EL2
0xe3f0-0xe3ff
       IMP DEF
0xe460
       HPFAR_EL2
0xe4b0-0xe4bf 0xe4f0-0xe4ff
       IMP DEF
0xe500
       VMPIDR_EL2
0xe5b0-0xe5bf 0xe5f0-0xe5ff 0xe6b0-0xe6bf 0xe6f0-0xe6ff
       IMP DEF
0xe711
       HACR_EL2
0xe7b0-0xe7bf 0xe7f0-0xe7ff 0xe8b0-0xe8bf 0xe8f0-0xe8ff 0xe9b0-0xe9bf
0xe9f0-0xe9ff 0xeab0-0xeabf 0xeaf0-0xeaff 0xebb0-0xebbf 0xebf0-0xebff
0xecb0-0xecbf 0xecf0-0xecff 0xedb0-0xedbf 0xedf0-0xedff 0xeeb0-0xeebf
0xeef0-0xeeff 0xefb0-0xefbf 0xeff0-0xefff
       IMP DEF
0xf010
       SCTLR_EL3
0xf011
       SCR_EL3
0xf020
       TTBR0_EL3
```

0xf040
SPSR_EL3
0xf041
SP_EL2
0xf051
AFSR0_EL3
0xf052
ESR_EL3
0xf060
FAR_EL3
0xf0a2
MAIR_EL3
0xf0a3
AMAIR_EL3
0xf0b0-0xf0bf
IMP DEF
0xf0c0
VBAR_EL3
0xf0f0-0xf0ff
IMP DEF
0xf110
ACTLR_EL3
0xf111
SDER32_EL3
0xf113
MDCR_EL3
0xf140
ELR_EL3
0xf151
AFSR1_EL3
0xf1b0-0xf1bf
IMP DEF
0xf1c0
RVBAR_EL3
0xf1f0-0xf1ff
IMP DEF
0xf211
CPTR_EL3
0xf220
TCR_EL3
0xf2b0-0xf2bf
IMP DEF
0xf2c0
RMR_EL3
0xf2d0
TPIDR_EL3
0xf2f0-0xf2ff 0xf3b0-0xf3bf 0xf3f0-0xf3ff 0xf4b0-0xf4bf 0xf4f0-0xf4ff
0xf5b0-0xf5bf 0xf5f0-0xf5ff 0xf6b0-0xf6bf 0xf6f0-0xf6ff 0xf7b0-0xf7bf
0xf7f0-0xf7ff 0xf8b0-0xf8bf
IMP DEF
0xf8e2
CNTPS_TVAL_EL1
0xf8f0-0xf8ff 0xf9b0-0xf9bf
IMP DEF
0xf9e2
CNTPS_CTL_EL1

`0xf9f0-0xf9ff 0xfab0-0xfabf`
IMP DEF
`0xfae2`
CNTPS_CVAL_EL1
`0xfaf0-0xfaff 0xfbb0-0xfbbf 0xfbf0-0xfbff 0xfcb0-0xfcbf 0xfcf0-0xfcff`
`0xfdb0-0xfdbf 0xfdf0-0xfdff 0xfeb0-0xfebf 0xfef0-0xfeff 0xffb0-0xffbf`
`0xfff0-0xffff`
IMP DEF

VALUE `unsigned int`
The new value written..

UPDATED_VALUE `unsigned int`
Updated value of the register now it has been written..

UNDEF `bool`
The register accessed is undefined..

CORE_NUM `unsigned int`
Core number in a multi processor..

UNALIGNED_LDST_RETIRED

Processor unaligned load/store.. Fields:

VADDR `unsigned int`
The virtual address of the access..

RESPONSE `enum`
0=Aborted, 1=OK, 2=Exclusive Failed.
`0x0`
Aborted
`0x1`
OK
`0x2`
Failed

LOCK `enum`
Normal, exclusive or locked access..
`0x0`
Normal
`0x1`
Exclusive
`0x2`
Locked

TRANS `bool`
Is this a translated access..

ACQREL `enum`
Is this an acquire/release.
`0x0`
None
`0x1`
Global
`0x2`
Local

SIZE `unsigned int`
Width of the access in bytes. Only required if DATA is not traced..

ELEMENT_SIZE `unsigned int`
Width of each element..

PADDR `unsigned int`
The physical (translated) address..

NSDESC `unsigned int`
The physical address non-secure bit..

PADDR2 unsigned int
: If different from PADDR, the physical address of the second page of the access..

NSDESC2 unsigned int
: The second page physical address non-secure bit..

DATA unsigned int
: The data read or written..

VFP_D_REGS

VFP/NEON D 64 bit register write.. Fields:

ID unsigned int
: The register number..

VALUE unsigned int
: The new value written to the register..

OLD_VALUE unsigned int
: The old value overwritten..

CORE_NUM unsigned int
: Core number in a multi processor..

ALIASING enum
: Type of register aliasing used.
  0x1
    AArch32
  0x2
    AArch64

MASK unsigned int
: Mask for partial register update..

VFP_Q_REGS

VFP/NEON Q 128 bit register write.. Fields:

ID unsigned int
: The register number..

VALUE unsigned int
: The new value written to the register..

OLD_VALUE unsigned int
: The old value overwritten..

CORE_NUM unsigned int
: Core number in a multi processor..

ALIASING enum
: Type of register aliasing used.
  0x1
    AArch32
  0x2
    AArch64

MASK unsigned int
: Mask for partial register update..

VFP_SYS_REGS

Writes to the VFP/NEON units system registers.. Fields:

ID enum
: Which VFP system register is written..
  0x1
    FPSCR
  0x8
    FPEXC

VALUE unsigned int
: The new value written to the VFP system register..

OLD_VALUE unsigned int
: The register's old value overwritten..

CORE_NUM unsigned int
: Core number in a multi processor..

VFP_S_REGS

VFP/NEON S 32 bit register write.. Fields:

ID unsigned int
: The register number..

VALUE unsigned int
: The new value written to the register..

OLD_VALUE unsigned int
: The old value overwritten..

CORE_NUM unsigned int
: Core number in a multi processor..

ALIASING enum
: Type of register aliasing used..

  0x1
  : AArch32

  0x2
  : AArch64

WFE_END

WFE ended.. Fields:

INST_COUNT unsigned int
: Ticks count when leaving WFE..

WFE_EVENT_REGISTER

WFE event register status: set/clear, reason.. Fields:

INST_COUNT unsigned int
: Ticks count..

REASON enum
: Reason for set/clear. Only REASON==0 (Cleared by WFE) clears the bit, all other reasons set it..

  0x0
  : Cleared by WFE

  0x1
  : Set by SEV instruction (this or another core)

  0x2
  : Set by Exception Return

  0x3
  : Set by external FIQ

  0x4
  : Set by external IRQ

  0x5
  : Set by external Abort

  0x6
  : Set by virtual FIQ

  0x7
  : Set by virtual IRQ

`0x8`
: Set by virtual Abort

`0x10`
: Set by debug request

`0x11`
: Set by debug halt

`0x12`
: Set by debug OS unlock

`0x20`
: Set by pending exception (M class only)

`0x21`
: Set by exception taken (M class only)

`0x40`
: Set by SEVL instruction

`0x80`
: Set by Global Timer event

`0x100`
: Set by Global Virtual Timer event

`0x200`
: Set by loss of monitor reservation

WFE_IGNORED

WFE ignored.. Fields:

INST_COUNT `unsigned int`
: Ticks count when ignoring WFE..

TRAPPED `bool`
: This WFE was trapped..

EVENT `bool`
: This WFE was ignored because the event register was set..

WFE_START

WFE entered.. Fields:

INST_COUNT `unsigned int`
: Ticks count when entering WFE..

WFI_END

WFI ended.. Fields:

INST_COUNT `unsigned int`
: Ticks count when leaving WFI..

WFI_IGNORED

WFI ignored.. Fields:

INST_COUNT `unsigned int`
: Ticks count when ignoring WFI..

TRAPPED `bool`
: This WFI was trapped..

DISABLED `bool`
: This WFI was ignored because WFI is disabled..

WFI_START

WFI entered.. Fields:

INST_COUNT unsigned int
Ticks count when entering WFI..

WFI_WAKEUP

WFI wakeup.. Fields:

INST_COUNT unsigned int
Ticks count when WFI wakeup occurred..

REASON enum
Reason for wakeup..

0x0
Reset signal

0x1
FIQ signal

0x2
IRQ signal

0x3
Abort signal

0x4
VFIQ signal

0x5
VIRQ signal

0x6
VAbort signal

0x7
Debug request signal

0x8
Debug halt signal

0x9
Debug OS Unlock request

0xc
Pending exception

0xd
Global timer event stream

0xe
Global virtual timer event stream

0xf
loss of global monitor reservation

0x10
IMP DEF wakeup mechanism (CADI register write)

ARMCortexA57xnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA57xnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- The value of the AArch64 PMCEID0_EL0 register, and the AArch32 alias of this register, differs in the model from the TRM value. The model value reflects the model counters.
- The architecture version reported for the enabled GIC CPU interface in the model is GICv3. The RTL implements GICv4.
- The mechanisms for setting the affinity fields of the MPIDR. The RTL has two ports:
  — CLUSTERIDAFF1[7:0].
  — CLUSTERIDAFF2[7:0].

AFF1 sets the value of MPIDR bits[15:8] and AFF2 sets the value of MPIDR bits[23:16]. In contrast, the model has a single `CLUSTER_ID` port. This difference allows the setting of bits[23:8] of the MPIDR using bits[15:0] of the `CLUSTER_ID` value.

- The memory mapped debug registers have a view for cores and a view for external debug agents. In the model, these views require two PVBus ports. In hardware, the system designer decides how the implementation differentiates the views.
- In the model, a single `peer` event port combines the functionality of the `eventi` and `evento` signals in the RTL.
- The Generic Timers are *Programmer's View* (PV) level abstractions: a model-specific protocol connects the `cntvalueb` port to the MemoryMappedCounterModule.
- The GIC CPU Interface is a PV level abstraction: a model-specific protocol connects the GIC CPU Interface to the GIC Distributor.
- The CoreSight *Cross Trigger Interface* (CTI) is a PV level abstraction: the interface is a model-specific one.
- The model has no mechanism to read the internal memory that the Cache and TLB structures use, through the implementation defined region of the system coprocessor interface. This memory includes the RAM Index Register, IL1DATA Registers, DL1DATA Registers, and associated functionality.
- The model does not implement:
  - ETM registers.
  - The PMUEVENT bus.
  - The WARMRESETREQ signal. However, the warm reset code sequence (the ARMv8-A ARM ARM specifies it) makes the model simulate a warm reset of the core.
  - The PMUSNAPSHOTREQ and PMUSNAPSHOTACK signals.
  - The CLREXMONREQ, CLREXMONACK, L2FLUSHREQ, and L2FLUSHACK L2 power management signals.
  - The EXTERRIRQ and INTERRIRQ signals.
  - Processor dynamic-retention signals.
  - The SYSBARDISABLE signal.
  - The DBGPWRDUP, DBGPWRUPREQ, DBGNOPWRDWN, and DBGRSTREQ debug power management signals.

Related references

4.4.25 MemoryMappedCounterModule component on page 4-325.

Related information

Code sequence to request a Warm reset as a result of RMR_ELx.RR.

3.2.4 ARMCortexA53xnCT component

This section describes the ARMCortexA53x*n*CT component.

ARMCortexA53xnCT - about

This C++ component is a model of r0p1 of an ARMv8-A Cortex-A53 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA53xnCT - ports

This section describes the ports.

Table 3-17 ARMCortexA53xnCT ports

Name	Protocol	Type	Description
aa64naa32[0-3]	Signal	Slave	Register width state after reset.
acp_s	PVBus	Slave	Bus slave that the processor receives coherency transactions on. It is a *Programmer's View* (PV) of the *Advanced Extensible Interface* (AXI) *Accelerator Coherency Port* (ACP) slave port.
broadcastinner	Signal	Slave	Enable broadcasting of inner shareable transactions.
broadcastouter	Signal	Slave	Enable broadcasting of outer shareable transactions.
broadcastcachemaint	Signal	Slave	Enable broadcasting of cache maintenance operations to downstream caches.
cfgend[0-3]	Signal	Slave	Configure endianness at reset: set the value of the EE bits in the CP15 SCTLR_EL3 and SCTR_S registers.
cfgte[0-3]	Signal	Slave	Initialize to take exceptions in T32 state after reset.
clk_in	ClockSignal	Slave	Processor clock input, for determining the rate of instruction execution relative to other system components.
clusterid	Value	Slave	Master ID, bits[23:8] of the MPIDR using bits[15:0] of the CLUSTERID value.
CNTHPIRQ[0-3]	Signal	Master	Hypervisor physical timer interrupt.
CNTPNSIRQ[0-3]	Signal	Master	Secure physical timer interrupt.
CNTPSIRQ[0-3]	Signal	Master	Non-secure physical timer interrupt.
cntvalueb	CounterInterface	Slave	Interface to a platform level MemoryMappedCounter module.
CNTVIRQ[0-3]	Signal	Master	Virtual timer interrupt.
commirq[0-3]	Signal	Master	Communications channel receive or transmit interrupt request.
commrx[0-3]	Signal	Master	Communications channel receive.
commtx[0-3]	Signal	Master	Communications channel transmit.
cp15sdisable[0-3]	Signal	Slave	Disables write access to some Secure CP15 registers.
cpuporeset[0-3]	Signal	Slave	Power on reset. Initializes all the processor logic, including debug logic.
cryptodisable[0-3]	Signal	Slave	Cryptography engine disable.[m]
ctdbgirq[0-3]	Signal	Master	*Cross Trigger Interface* (CTI) interrupt trigger output.

[m] ARMv8 Cryptography Extensions require a separate package, which is subject to export license conditions. Contact ARM for details.

Table 3-17 ARMCortexA53xnCT ports (continued)

Name	Protocol	Type	Description
`cti[0-3]`	v8EmbeddedCrossTrigger_controlprotocol	Master	Interface to the CTI to a platform level *Cross Trigger Matrix* (CTM).
`dbgack[0-3]`	Signal	Master	Debug acknowledge.
`dbgen[0-3]`	Signal	Slave	Invasive debug enable.
`dbgromaddr`	Value_64	Slave	Specify bits[43:12] of the top-level ROM table physical address.
`dbgromaddrv`	Signal	Slave	Valid signal for `dbgromaddr`.
`dev_debug_s`	PVBus	Slave	Debug *Advanced Peripheral Bus* (APB) as exposed to external debug agents.[n]
`edbgrq[0-3]`	Signal	Slave	External debug request.
`event`	Signal	Peer	Event input and output for wakeup from WFE. This port amalgamates the hardware EVENTI and EVENT0 signals.
`fiq[0-3]`	Signal	Slave	Processor FIQ signal input.
`gicv3_redistributor_s[0-3]`	GICv3Comms	Slave	Interface to a platform-level GICv3Distributor component.
`irq[0-3]`	Signal	Slave	Processor IRQ signal input.
`l2reset`	Signal	Slave	Reset the shared L2 memory system controller.
`memory_mapped_debug_s`	PVBus	Slave	Debug APB as exposed to other system agents.[n]
`niden[0-3]`	Signal	Slave	Non-invasive debug enable.
`periphbase`	Value_64	Slave	Base address of peripheral memory space, the base address for the GIC CPU Interface registers, which are sampled into the *Configuration Base Address Register* (CBAR) at reset.
`pmuirq[0-3]`	Signal	Master	*Performance Monitoring Unit* interrupt signal.
`presetdbg`	Signal	Slave	Initialize the shared debug APB, *Cross Trigger Interface* (CTI), and *Cross Trigger Matrix* (CTM) logic.
`pvbus_m0`	PVBus	Master	Bus master that the processor generates transactions on. This port is a PV representation of the AXI master port.
`rei[4]`	Signal	Slave	Individual processor *RAM Error Interrupt* signal input.
`reset[0-3]`	Signal	Slave	Individual processor reset.

[n] The system designer decides whether a debug APB ties the external debug view with other system views. In the model, use a PVBusDecoder to direct traffic to the correct port.

Table 3-17 ARMCortexA53xnCT ports (continued)

Name	Protocol	Type	Description
rvbaraddr[0-3]	Value_64	Slave	Reset Vector Base Address for executing in AArch64 state. Only sampled at reset.
sei[0-3]	Signal	Slave	Individual processor *System Error Interrupt* signal input.
smpen[4]	Signal	Master	Status of the CPUECTLR.SMPEN bit, whether the processor is SMP enabled.
spiden[0-3]	Signal	Slave	Secure privileged invasive debug enable.
spniden[0-3]	Signal	Slave	Secure privileged non-invasive debug enable.
standbywfe[0-3]	Signal	Master	Indicates if a processor is in *Wait For Event* state.
standbywfi[0-3]	Signal	Master	Indicates if a processor is in *Wait For Interrupt* state.
standbywfil2	Signal	Master	Indicate that all the individual processors and the L2 systems are in a WFI state.
ticks[0-3]	InstructionCount	Master	Instruction count for visualization.
vcpumntirq[0-3]	Signal	Master	Virtual processor interface maintenance interrupt request.
vinithi[0-3]	Signal	Slave	Initialize with high vectors enabled after reset.
virtio_s	PVBus	Slave	A model-specific port that connects to virtio peripherals in the L2 system. It ensures coherency, with the correct attributes.
vfiq[0-3]	Signal	Slave	Processor *Virtual FIQ* signal input.
virq[0-3]	Signal	Slave	Processor *Virtual IRQ* signal input.
vsei[0-3]	Signal	Slave	Processor *Virtual System Error Interrupt* request.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.2 GICv3Comms protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.
2.5.4 v8EmbeddedCrossTrigger_controlprotocol protocol on page 2-65.

ARMCortexA53xnCT - parameters

This section describes the parameters.

ARMCortexA53xnCT parameters

Table 3-18 ARMCortexA53xnCT parameters

Name	Type	Allowed values	Default value	Description
CLUSTER_ID	uint32_t	0x0-0xFFFF	0x0	Master ID, bits[23:8] of the MPIDR using bits[15:0] of the CLUSTER_ID value.
dcache-size	uint32_t	0x4000-0x100000	0x8000	L1 D-cache size in bytes.
dcache-state_modelled	bool	true, false	false	L1 D-cache has stateful implementation.[o]
GICDISABLE	bool	true, false	true	Disable the GIC CPU interface.
icache-size	uint32_t	0x4000-0x100000	0x8000	L1 I-cache size in bytes.
icache-state_modelled	bool	true, false	false	L1 I-cache has stateful implementation.[o]
l2cache-size	uint32_t	0x80000-0x200000	0x80000	L2 cache size in bytes.
PERIPHBASE	uint64_t	0-0xFFFFFFFFFFFF	0x13080000	Base address of peripheral memory space, the base address for the GIC CPU Interface registers, sampled into the *Configuration Base Address Register* (CBAR) at reset.

ARMCortexA53xnCT cache latency cluster parameters

Note

- These latencies are only effective when you enable cache-state modeling.
- Timing annotation for transactions downstream of the cache models propagates through the cache models.

Table 3-19 ARMCortexA53xnCT cache latency cluster parameters

Parameter	Type	Allowed values	Default value	Description
dcache-read_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-snoop_data_transfer_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for received snoop accesses that perfom a data transfer given in ticks per byte accessed. For use when dcache-state_modelled=true.
dcache-write_latency	uint64_t	-	0x0	L1 D-cache timing annotation latency for write accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
icache-read_latency	uint64_t	-	0x0	L1 I-cache timing annotation latency for read accesses given in ticks per byte accessed. For use when icache-state_modelled=true.

[o] If either L1 cache is stateful, then the L2 cache is stateful.

Table 3-19 ARMCortexA53xnCT cache latency cluster parameters (continued)

Parameter	Type	Allowed values	Default value	Description
l2cache-read_latency	uint64_t	-	0x0	L2 cache timing annotation latency for read accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_data_transfer_latency	uint64_t	-	0x0	L2 cache timing annotation latency for received snoop accesses that perfom a data transfer given in ticks per byte accessed. For use when dcache-state_modelled=true.
l2cache-snoop_issue_latency	uint64_t	-	0x0	L2 cache timing annotation latency for snoop accesses issued by this cache in total ticks. For use when dcache-state_modelled=true.
l2cache-write_latency	uint64_t	-	0x0	L2 cache timing annotation latency for write accesses given in ticks per byte accessed. For use when dcache-state_modelled=true.

ARMCortexA53xnCT core parameters

Table 3-20 ARMCortexA53xnCT core parameters

Name	Type	Allowed values	Default value	Description
AA64nAA32	bool	true, false	true	Register width state after reset.
CFGEND	bool	true, false	false	Endianness configuration at reset. It sets the initial value of the EE bits in the CP15 SCTLR_EL3 and SCTR_S registers after a reset.
CFGTE	bool	true, false	false	Initialize to take exceptions in T32 state after reset.
CP15SDISABLE	bool	true, false	false	Disable write access to some Secure CP15 registers.
cpi_div	uint32_t	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	uint32_t	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
CRYPTODISABLE	bool	true, false	false	Disable ARMv8 Cryptography Extensions.[p]
DBGROMADDR	uint64_t	0x0-0xFFFFFFFFFFFFFFFF	0x22000000	Specify bits[43:12] of the top-level ROM table Physical Address.
DBGROMADDRV	bool	true, false	true	System samples DBGROMADDR.
max_code_cache	uint32_t	0x0-0xFFFFFFFFFFFFFFFF	0x16777216	Maximum number of bytes used for caching code translations.
min_sync_level	uint32_t	0-3	0	The minimum syncLevel.
RVBARADDR	uint64_t	0x0-0xFFFFFFFFFFFFFFFF	0x0	Reset Vector Base Address for executing in AArch64 state.
semihosting-ARM_SVC	uint32_t	0x0-0xFFFFFF	0x123456	A32 SVC number for semihosting.

[p] ARMv8 Cryptography Extensions require a separate package, which is subject to export license conditions. Contact ARM for details.

Table 3-20 ARMCortexA53xnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
semihosting-cmd_line	string	No limit except memory	""	Command line available to semihosting.
semihosting-cwd	string	No limit except memory	""	Default working directory for semihosting.[qr]
semihosting-enable	bool	true, false	true	Enable semihosting traps.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0x0	Virtual address of semihosting heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of semihosting top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10000000	Virtual address of semihosting base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of semihosting stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFF	0xAB	T32 SVC number for semihosting.
vfp-enable_at_reset	bool	true, false	false	All implementation operations required in order to enable FP and ASE support are performed implicitly by the model.[s]
VINITHI	bool	true, false	false	Initialize with high vectors enabled after reset.

ARMCortexA53xnCT TLB latency core parameters

Note

- The walk_cache_latency parameter is only available for AEMv8 clusters and only effective when you configure the walk cache with a non-zero size.
- Timing annotation for transactions downstream of the TLB model propagates through the TLB model.

Table 3-21 ARMCortexA53xnCT TLB latency core parameters

Parameter	Type	Allowed values	Default value	Description
ptw_latency	uint64_t	-	0x0	Page table walker latency for *Timing Annotation* (TA), in simulation ticks.
tlb_latency	uint64_t	-	0x0	TLB latency for TA, in simulation ticks.
walk_cache_latency	uint64_t	-	0x0	Walk cache latency for TA, in simulation ticks.

Related references

1.8 Non-CADI sync watchpoints on page 1-24.

[q] The host operating system limits the maximum path length.
[r] The semihosting-cwd option does not provide any security. Software running on the model can access files outside this directory using relative paths containing ".." or using absolute paths.
[s] This is a model-specific option that has no hardware equivalent. ARM recommends that it is only used in test systems and tied off to false in production systems.

ARMCortexA53xnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the ETM registers and the integration and test registers.

ARMCortexA53xnCT - caches

This component implements representative L1 and L2 caches.

ARMCortexA53xnCT - debug features

This component exports a CADI debug interface.

ARMCortexA53xnCT - debug - registers

All modeled registers are visible in the debugger.

ARMCortexA53xnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA53xnCT - debug - memory

This component presents virtual and physical views of memory.

The virtual views are:

- Secure Monitor.
- NS Hyp.
- Guest.

These views are 2^{64} bytes in size.

The physical views are:

- Physical Memory (Secure).
- Physical Memory (Non-secure).

These views are 2^{64} bytes in size, however the maximum physical address permissible by the Cortex-A53 processor is 2^{40} — 1.

ARMCortexA53xnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA53xnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- The value of the AArch64 PMCEID0_EL0 register, and the AArch32 alias of this register, differs in the model from the TRM value. The model value reflects the model counters.
- The architecture version reported for the enabled GIC CPU interface in the model is GICv3. The RTL implements GICv4.
- The mechanisms for setting the affinity fields of the MPIDR. The RTL has two ports:
  - CLUSTERIDAFF1[7:0].
  - CLUSTERIDAFF2[7:0].

AFF1 sets the value of MPIDR bits[15:8] and AFF2 sets the value of MPIDR bits[23:16]. In contrast, the model has a single `CLUSTER_ID` port. This difference allows the setting of bits[23:8] of the MPIDR using bits[15:0] of the `CLUSTER_ID` value.

- The memory mapped debug registers have a view for cores and a view for external debug agents. In the model, these views require two PVBus ports. In hardware, the system designer decides how the implementation differentiates the views.
- In the model, a single `peer` event port combines the functionality of the `eventi` and `evento` signals in the RTL.
- The Generic Timers are *Programmer's View* (PV) level abstractions: a model-specific protocol connects the `cntvalueb` port to the MemoryMappedCounterModule.
- The GIC CPU Interface is a PV level abstraction: a model-specific protocol connects the GIC CPU Interface to the GIC Distributor.
- The CoreSight *Cross Trigger Interface* (CTI) is a PV level abstraction: the interface is a model specific one.
- The model has no mechanism to read the internal memory that the Cache and TLB structures use, through the implementation defined region of the system coprocessor interface. This memory includes the RAM Index Register, IL1DATA Registers, DL1DATA Registers, and associated functionality.
- The model does not implement:
  - ETM registers.
  - The PMUEVENT bus.
  - The WARMRESETREQ signal. However, the warm reset code sequence (the ARMv8-A ARM ARM specifies it) makes the model simulate a warm reset of the core.
  - The PMUSNAPSHOTREQ and PMUSNAPSHOTACK signals.
  - The CLREXMONREQ, CLREXMONACK, L2FLUSHREQ, and L2FLUSHACK L2 power management signals.
  - The EXTERRIRQ and INTERRIRQ signals.
  - Processor dynamic-retention signals.
  - The SYSBARDISABLE signal.
  - The DBGPWRDUP, DBGPWRUPREQ, DBGNOPWRDWN, and DBGRSTREQ debug power management signals.
  - The RTL synthesis option to remove FP and ASE.
  - The RTL synthesis option for a Cortex-A15 style debug memory map.

Related references

4.4.25 MemoryMappedCounterModule component on page 4-325.

Related information

Code sequence to request a Warm reset as a result of RMR_ELx.RR.

3.2.5 ARMCortexA17xnCT component

This section describes the ARMCortexA17x*n*CT component.

ARMCortexA17xnCT - about

This C++ component is a model of r0p0 of a Cortex-A17 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA17xnCT - ports

This section describes the ports.

Table 3-22 ARMCortexA17xnCT ports

Name	Protocol	Type	Description
acp_s	PVBus	Slave	*Advanced eXtensible Interface* (AXI) *Accelerator Coherency Port* (ACP) slave port.
broadcastcachemaint	Signal	Slave	Controls issuing of cache maintenance transactions, such as CleanShared, CleanInvalid and MakeInvalid, on the coherent interconnect.[t]
broadcastinner	Signal	Slave	Controls issuing of coherent transactions targeting the Inner Shareable domain on the coherent interconnect.[t]
broadcastouter	Signal	Slave	Controls issuing of coherent transactions targeting the Outer Shareable domain on the coherent interconnect.[t]
CFGADDRFILTENDNS	Value_64	Slave	peripheral_m port NS end address, using only bits[39:20].
CFGADDRFILTENDS	Value_64	Slave	peripheral_m port S end address, using only bits[39:20].
CFGADDRFILTENNS	Signal	Slave	Enable peripheral_m port filtering for NS accesses.
CFGADDRFILTENS	Signal	Slave	Enable peripheral_m port filtering for S accesses.
CFGADDRFILTSTARTNS	Value_64	Slave	peripheral_m port NS start address, using only bits[39:20].
CFGADDRFILTSTARTS	Value_64	Slave	peripheral_m port S start address, using only bits[39:20].
cfgend[4]	Signal	Slave	Initialize to BE8 endianness after a reset. This signal controls the SCTLR.EE bit.
clk_in	ClockSignal	Slave	Main processor clock signal input, for setting the rate at which the core executes instructions.
clusterid	Value	Slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
CNTHPIRQ[4]	Signal	Master	Hypervisor physical timer interrupt.
CNTPNSIRQ[4]	Signal	Master	Non-secure physical timer interrupt.
CNTPSIRQ[4]	Signal	Master	Secure physical timer interrupt.
cntvalueb	CounterInterface	Slave	Synchronous counter value, the interface to the *System on Chip* (SoC) level counter module. Connect this to the MemoryMappedCounterModule component.
CNTVIRQ[4]	Signal	Master	Virtual timer interrupt.
cp15sdisable[4]	Signal	Slave	Disable write access to some secure system control processor registers.
cpuporeset[4]	Signal	Slave	Power-on reset. Initializes all the processor logic, including the NEON and VFP, 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 hardware EVENTI and EVENT0 signals.
fiq[4]	Signal	Slave	Core FIQ signal input, driving 'fast'-interrupt handling.
irq[4]	Signal	Slave	Core IRQ signal input, driving interrupt handling.
l2reset	Signal	Slave	Reset shared L2 memory system, interrupt controller and timer logic.
peripheral_m	PVBus	Master	Peripheral port of the core, in the control of the filter registers.
pmuirq[4]	Signal	Master	Interrupt signal from the *Performance Monitoring Unit* (PMU).

[t] *AXI Coherency Extensions* (ACE) defined pin.

Table 3-22 ARMCortexA17xnCT ports (continued)

Name	Protocol	Type	Description
presetdbg	Signal	Slave	Initializes the shared debug *Advanced Peripheral Bus* (APB), *Cross Trigger Interface* (CTI), and *Cross Trigger Matrix* (CTM) logic.
pvbus_m0	PVBus	Master	AXI bus master channel. The core generates bus requests on this port.
reset[4]	Signal	Slave	Core reset signal, which puts the core into reset mode..
standbywfe[4]	Signal	Master	Indicates if a core is in *Wait For Event* (WFE) state.
standbywfi[4]	Signal	Master	Indicates if a core is in *Wait For Interrupt* (WFI) state.
teinit[4]	Signal	Slave	Enable exceptions in T32 state after a reset. This signal controls the SCTLR.TE bit.
ticks[4]	InstructionCount	Master	Core instruction count for visualization. Connect it to one of the two ticks ports on a visualization component to display a running instruction count.
vfiq[4]	Signal	Slave	Virtual FIQ signal input.
vinithi[4]	Signal	Slave	Initialize with high vectors enabled after a reset. This signal controls the location of the exception vectors at reset.
virq[4]	Signal	Slave	Virtual IRQ signal input.
virtio_s	PVBus	Slave	Virtio coherent port, which hooks into the L2 system and becomes coherent, given correct attributes.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.
2.6.2 Signal protocol on page 2-66.
2.6.4 Value protocol on page 2-66.
2.6.5 Value_64 protocol on page 2-66.

ARMCortexA17xnCT - parameters

The parameters are set once, irrespective of the number of cores. If the cluster has multiple cores, then each core has its own parameters.

Table 3-23 ARMCortexA17xnCT cluster parameters

Name	Type	Allowed values	Default value	Description
CFGADDRFILTENDNS	uint64_t	0-0xFFFFF00000	0	peripheral_m port NS end address.
CFGADDRFILTENDS	uint64_t	0-0xFFFFF00000	0	peripheral_m port S end address.
CFGADDRFILTENNS	bool	true, false	false	peripheral_m port NS address filtering enabled.
CFGADDRFILTENS	bool	true, false	false	peripheral_m port S address filtering enabled.
CFGADDRFILTSTARTNS	uint64_t	0-0xFFFFF00000	0	peripheral_m port NS start address.
CFGADDRFILTSTARTS	uint64_t	0-0xFFFFF00000	0	peripheral_m port S start address.
CLUSTER_ID	uint32_t	0x0-0xF	0x0	Cluster ID value.

Table 3-23 ARMCortexA17xnCT cluster parameters (continued)

Name	Type	Allowed values	Default value	Description
IMINLN	bool	true, false	true	Instruction cache minimum line size. false, 32 bytes. true, 64 bytes.
l1_dcache-state_modelled	bool	true, false	false	Set whether L1 D-cache has stateful implementation.
l1_icache-state_modelled	bool	true, false	false	Set whether L1 I-cache has stateful implementation.
l2_cache-size	uint32_t	0x40000-0x800000	0x40000	Set L2 cache size in bytes.
l2_cache-state_modelled	bool	true, false	false	Set whether L2 cache has stateful implementation.

Table 3-24 ARMCortexA17xnCT core parameters

Name	Type	Allowed values	Default value	Description
ase-present[u]	bool	true, false	true	Model has NEON support.
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CP15SDISABLE	bool	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	uint32_t	0x1-0x7FFFFFFF	0x1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	uint32_t	0x1-0x7FFFFFFF	0x1	Multiplier for calculating CPI.
DBGROMADDR	uint32_t	0x0-0xFFFFFFFF	0x12000003	Initializes the CP15 **DBGDRAR** register. Bits[39:12] of this register specify the ROM table physical address.
DBGROMADDRV	bool	true, false	true	If true, this sets bits[1:0] of the CP15 **DBGDRAR** to indicate that the address is valid.
DBGSELFADDR	uint32_t	0x0-0xFFFFFFFF	0x00010003	Initializes the CP15 **DBGDSAR** register. Bits[39:17] of this register specify the ROM table physical address.
DBGSELFADDRV	bool	true, false	true	If true, this sets bits[1:0] of the CP15 **DBGDSAR** to indicate that the address is valid.
l1_icache-size	uint32_t	0x8000-0x10000	0x8000	Size of L1 I-cache.
min_sync_level	uint32_t	0x0-0x03	0x0	Controls the minimum syncLevel. 0x0, off = default. 0x1, syncState. 0x2, postInsnIO. 0x3, postInsnAll.
semihosting-ARM_SVC	uint32_t	0x0-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-cmd_line	string	-	""	Command line available to semihosting SVC calls.

[u] The ase-present and vfp-present parameters configure the synthesis options.

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.

Table 3-24 ARMCortexA17xnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
semihosting-cwd	string	-	""	Virtual address of CWD.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0	Virtual address of heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFF	0xAB	T32 SVC number for semihosting.
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[v]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[u]	bool	true, false	true	Model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled after a reset.

3.2.6 ARMCortexA15xnCT component

This section describes the ARMCortexA15x*n*CT component.

ARMCortexA15xnCT - about

This C++ component is a model of r2p0 of a Cortex-A15 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA15xnCT - ports

This section describes the ports.

Table 3-25 ARMCortexA15xnCT ports

Name	Protocol	Type	Description
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.
clusterid	Value	Slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
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.
cntvalueb	CounterInterface	Slave	Synchronous counter value. This must be connected to the MemoryMappedCounterModule component.

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

Table 3-25 ARMCortexA15xnCT ports (continued)

Name	Protocol	Type	Description
CNTVIRQ[0-3]	Signal	Master	Virtual timer event.
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.
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.
virtio_s	PVBus	Slave	Channel for coherency traffic from virtual model devices. Not for public use.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA15xnCT - parameters

The parameters are set once, irrespective of the number of cores. If the cluster has multiple cores, then each core has its own parameters.

Table 3-26 ARMCortexA15xnCT cluster parameters

Name	Type	Allowed values	Default value	Description
CFGSDISABLE	Boolean	true, false	false	Disable some accesses to GIC registers.
CLUSTER_ID	Integer	0-15	0	Cluster ID value.

Table 3-26 ARMCortexA15xnCT cluster parameters (continued)

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

Table 3-27 ARMCortexA15xnCT core parameters

Name	Type	Allowed values	Default value	Description
CFGEND	Boolean	true, false	false	Initialize to BE8 endianness.
CP15SDISABLE	Boolean	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
DBGROMADDR	Integer	0x00000000-0xFFFFFFFF	0x12000003	This value is used to initialize the CP15 DBGDRAR register. Bits[39:12] of this register specify the ROM table physical address.
DBGROMADDRV	Boolean	true, false	true	If true, this sets bits[1:0] of the CP15 DBGDRAR to indicate that the address is valid.
DBGSELFADDR	Integer	0x00000000-0xFFFFFFFF	0x00010003	This value is used to initialize the CP15 DBGDSAR register. Bits[39:17] of this register specify the ROM table physical address.
DBGSELFADDRV	Boolean	true, false	true	If true, this sets bits[1:0] of the CP15 DBGDSAR to indicate that the address is valid.

[w] 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 3-27 ARMCortexA15xnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
TEINIT	Boolean	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.
ase-present[x]	Boolean	true, false	true	Set whether CT model has been built with NEON support.
min_sync_level	Integer	0-3	0	Controls the minimum syncLevel.
semihosting-cmd_line	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	Integer	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	0x00-0xFF	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
vfp-enable_at_reset[y]	Boolean	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[x]	Boolean	true, false	true	Set whether CT model has been built with VFP support.

ARMCortexA15xnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the integration and test registers.

ARMCortexA15xnCT - caches

This component implements L1 and L2 caches as architecturally defined.

[x] The ase-present and vfp-present parameters configure the synthesis options.

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.

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

ACE limitation

AXI Coherency Extensions (ACE) are extensions to AXI4 that support system-level cache-coherency between multiple clusters. The ACE cache models in the Cortex-A15 and the Cortex-A7, and the ACE support in the CCI-400 have a limitation: these functional models process only one transaction at a time. Normally, the simulation processes each transaction to completion before allowing any master to generate another transaction. However, there is a situation in which the simulation might fail. Suppose a SystemC bus slave calls `wait()` while it is processing a transaction. This call might allow another master to issue another transaction that passes through the CCI-400 or the Cortex-A15/Cortex-A7 caches. This situation could happen if a SystemC bus master running in another thread is connected to one of the ACE-lite ports on the CCI-400.

Cache and TLB component visibility

If a core model has a cache model available, to create it and make it visible, enable it. To enable the cache model and be ready to use cache CADI, set these model parameters:

```
l1_icache-state_modelled
l1_dcache-state_modelled
l2_cache-state_modelled
```

To use cache and TLB viewers, connect the cache and TLB CADI components.

ARMCortexA15xnCT - debug features

This component exports a CADI debug interface.

ARMCortexA15xnCT - debug - registers

All core, VFP, CP14, and CP15 registers are visible in the debugger. All CP14 debug registers are implemented.

ARMCortexA15xnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA15xnCT - debug - memory

This component presents three 4GB views of virtual address space, that is, one in hypervisor, one in secure mode and one in non-secure mode.

ARMCortexA15xnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA15xnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- The GIC does not respect the CFGSDISABLE signal. This leads to some registers wrongly being accessible.
- The Broadcast *Translation Lookaside Buffer* (TLB) or cache operations in the model do not cause other cores in the cluster that are asleep because of *Wait For Interrupt* (WFI) to wake up.
- It ignores the RR bit in the SCTLR.
- It implements the Power Control Register in the system control coprocessor but writing to it does not change the behavior of the model.
- When modeling the SCU, coherency operations are by memory writes then reads to refill from memory, rather than cache-to-cache transfers.

- It does not implement ETM registers.
- It implements TLB bitmap registers as RAZ/WI.
- It does not support the Cortex-A15 mechanism to read the internal memory that the Cache and TLB structures use through the implementation defined region of the system coprocessor interface. This includes the RAM Index Register, IL1DATA Registers, DL1DATA Registers, and associated functionality.

3.2.7 ARMCortexA12xnCT component

This section describes the ARMCortexA12x*n*CT component.

ARMCortexA12xnCT - about

This C++ component is a model of r0p0 of a Cortex-A12 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA12xnCT - ports

This section describes the ports.

Table 3-28 ARMCortexA12xnCT ports

Name	Protocol	Type	Description
`acp_s`	PVBus	Slave	*Advanced eXtensible Interface* (AXI) ACP slave port.
`broadcastcachemaint`	Signal	Slave	Controls issuing of cache maintenance transactions, such as CleanShared, CleanInvalid and MakeInvalid, on the coherent interconnect.[z]
`broadcastinner`	Signal	Slave	Controls issuing of coherent transactions targeting the Inner Shareable domain on the coherent interconnect.[z]
`broadcastouter`	Signal	Slave	Controls issuing of coherent transactions targeting the Outer Shareable domain on the coherent interconnect.[z]
`cfgend[4]`	Signal	Slave	Initialize to BE8 endianness after a reset.
`CFGADDRFILTENDNS`	Value_64	Slave	`peripheral_m` port NS end address, using only bits[39:20].
`CFGADDRFILTENDS`	Value_64	Slave	`peripheral_m` port S end address, using only bits[39:20].
`CFGADDRFILTENNS`	Signal	Slave	Enable `peripheral_m` port filtering for NS accesses.
`CFGADDRFILTENS`	Signal	Slave	Enable `peripheral_m` port filtering for S accesses.
`CFGADDRFILTSTARTNS`	Value_64	Slave	`peripheral_m` port NS start address, using only bits[39:20].
`CFGADDRFILTSTARTS`	Value_64	Slave	`peripheral_m` port S start address, using only bits[39:20].
`clk_in`	ClockSignal	Slave	Main processor clock input.
`clusterid`	Value	Slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
`CNTHPIRQ[4]`	Signal	Master	Hypervisor physical timer event.
`CNTPNSIRQ[4]`	Signal	Master	Non-secure physical timer event.
`CNTPSIRQ[4]`	Signal	Master	Secure physical timer event.
`cntvalueb`	CounterInterface	Slave	Synchronous counter value. Connect this to the MemoryMappedCounterModule component.
`CNTVIRQ[4]`	Signal	Master	Virtual timer event.

[z] *AXI Coherency Extensions* (ACE) defined pin.

Table 3-28 ARMCortexA12xnCT ports (continued)

Name	Protocol	Type	Description
cpuporeset[4]	Signal	Slave	Power-on reset. Initializes all the processor logic, including the NEON and VFP, Debug, PTM, breakpoint, and watchpoint logic in the processor CLK domain.
cp15sdisable[4]	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[4]	Signal	Slave	Processor FIQ signal input.
irq[4]	Signal	Slave	Processor IRQ signal input.
l2reset	Signal	Slave	Reset shared L2 memory system, interrupt controller and timer logic.
peripheral_m	PVBus	Master	Peripheral port of the processor, under control by filter registers.
pmuirq[4]	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[4]	Signal	Slave	Individual processor reset signal.
standbywfe[4]	Signal	Master	Indicates if a processor is in *Wait For Event* (WFE) state.
standbywfi[4]	Signal	Master	Indicates if a processor is in *Wait For Interrupt* (WFI) state.
teinit[4]	Signal	Slave	Initialize to take exceptions in T32 state after a reset.
ticks[4]	InstructionCount	Master	Processor instruction count for visualization.
vfiq[4]	Signal	Slave	Processor virtual FIQ signal input.
vinithi[4]	Signal	Slave	Initialize with high vectors enabled after a reset.
virq[4]	Signal	Slave	Processor virtual IRQ signal input.
virtio_s	PVBus	Slave	Virtio coherent port, which hooks into the L2 system and becomes coherent, given correct attributes.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA12xnCT - parameters

This section describes the parameters.

Table 3-29 ARMCortexA12xnCT parameters

Name	Type	Allowed values	Default value	Description
CFGADDRFILTENDNS	uint64_t	0-0xFFFFF00000	0	peripheral_m port NS end address.
CFGADDRFILTENDS	uint64_t	0-0xFFFFF00000	0	peripheral_m port S end address.
CFGADDRFILTENNS	bool	true, false	false	peripheral_m port NS address filtering enabled.

Table 3-29 ARMCortexA12xnCT parameters (continued)

Name	Type	Allowed values	Default value	Description
CFGADDRFILTENS	bool	true, false	false	peripheral_m port S address filtering enabled.
CFGADDRFILTSTARTNS	uint64_t	0-0xFFFFF00000	0	peripheral_m port NS start address.
CFGADDRFILTSTARTS	uint64_t	0-0xFFFFF00000	0	peripheral_m port S start address.
CLUSTER_ID	uint32_t	0-0xF	0	Cluster ID value.
IMINLN	bool	true, false	true	Instruction cache minimum line size. false, 32 bytes. true, 64 bytes.
l1_dcache-state_modelled	bool	true, false	false	Set whether L1 D-cache has stateful implementation.
l1_icache-state_modelled	bool	true, false	false	Set whether L1 I-cache has stateful implementation.
l2_cache-size	uint32_t	512KB, 1MB, 2MB, 4MB	0x40000	Set L2 cache size in bytes.
l2_cache-state_modelled	bool	true, false	false	Set whether L2 cache has stateful implementation.

Table 3-30 ARMCortexA12xnCT core parameters

Name	Type	Allowed values	Default value	Description
ase-present[aa]	bool	true, false	true	Model has NEON support.
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CP15SDISABLE	bool	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	uint32_t	0x1-0x7FFFFFFF	0x1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	uint32_t	0x1-0x7FFFFFFF	0x1	Multiplier for calculating CPI.
DBGROMADDR	uint32_t	0x0-0xFFFFFFFF	0x12000003	Initializes the CP15 **DBGDRAR** register. Bits[39:12] of this register specify the ROM table physical address.
DBGROMADDRV	bool	true, false	true	If true, this sets bits[1:0] of the CP15 **DBGDRAR** to indicate that the address is valid.
DBGSELFADDR	uint32_t	0x0-0xFFFFFFFF	0x00010003	Initializes the CP15 **DBGDSAR** register. Bits[39:17] of this register specify the ROM table physical address.
DBGSELFADDRV	bool	true, false	true	If true, this sets bits[1:0] of the CP15 **DBGDSAR** to indicate that the address is valid.
l1_icache-size	uint32_t	0x8000-0x10000	0x8000	Size of L1 I-cache.

[aa] The ase-present and vfp-present parameters configure the synthesis options.

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.

Table 3-30 ARMCortexA12xnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
min_sync_level	uint32_t	0x0-0x03	0x0	Controls the minimum syncLevel. 0x0, off = default. 0x1, syncState. 0x2, postInsnIO. 0x3, postInsnAll.
semihosting-ARM_SVC	uint32_t	0x0-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-cmd_line	string	-	""	Command line available to semihosting SVC calls.
semihosting-cwd	string	-	""	Virtual address of CWD.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0	Virtual address of heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFF	0xAB	T32 SVC number for semihosting.
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[ab]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[aa]	bool	true, false	true	Model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled after a reset.

ARMCortexA12xnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the integration and test registers.

ARMCortexA12xnCT - caches

This component implements L1 and L2 caches as architecturally defined.

ARMCortexA12xnCT - debug features

The component exports a CADI debug interface.

ARMCortexA12xnCT - debug - registers

All core, VFP, CP14, and CP15 registers are visible in the debugger. All CP14 debug registers are implemented.

ARMCortexA12xnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

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

ARMCortexA12xnCT - debug - memory

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

ARMCortexA12xnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA12xnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- The Broadcast *Translation Lookaside Buffer* (TLB) or cache operations in this model do not cause other cores 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-A12 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.

3.2.8 ARMCortexA9MPxnCT component

This section describes the ARMCortexA9MPx*n*CT component.

ARMCortexA9MPxnCT - about

This C++ component is a model of r3p0 of a Cortex-A9 processor containing from one to four cores. The *n* shows the number of cores.

This component implements these 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).

ARMCortexA9MPxnCT - ports

This section describes the ports.

Table 3-31 ARMCortexA9MPxnCT ports

Name	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.

Table 3-31 ARMCortexA9MPxnCT ports (continued)

Name	Protocol	Type	Description
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.
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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA9MPxnCT - parameters

The parameters are set once, irrespective of the number of cores. Each core has its own parameters.

The first core might have the instance name coretile.core.cpu0, for example, where “coretile.core” and “cpu0” are the normal instance name and the first core, respectively.

Table 3-32 ARMCortexA9MPxnCT cluster parameters

Name	Type	Allowed values	Default value	Description
CLUSTER_ID	int	0-15	0	Cluster ID value.
CFGSDISABLE	bool	true, false	false	Disable some accesses to GIC registers.
dcache-state_modelled	bool	true, false	false	Set whether or not D-cache has stateful implementation.
device-accurate-tlb	bool	true, false	false[ac]	Specify whether or not all TLBs are modeled.
dic-spi_count	int	0-223, in increments of 32	64	Number of shared peripheral interrupts implemented.
FILTEREN	bool	true, false	false	Enable filtering of accesses through pvbus_m0.
FILTEREND	int	Must be aligned on 1MB boundary.	0x0	End of region filtered to pvbus_m0.
FILTERSTART	int	Must be aligned on 1MB boundary.	0x0	Base of region filtered to pvbus_m0.
icache-state_modelled	bool	true, false	false	Set whether or not I-cache has stateful implementation.
PERIPHBASE	int	-	0x13080000[ad]	Base address of peripheral memory space.

Table 3-33 ARMCortexA9MPxnCT core parameters

Name	Type	Allowed values	Default value	Description
ase-present[ae]	bool	true, false	true	Set whether or not the model has NEON support.
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CFGNMFI	bool	true, false	false	Enable nonmaskable FIQ interrupts on startup.
CP15SDISABLE	bool	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	int	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	int	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
dcache-size	int	16KB, 32KB, 64KB	0x8000	Set D-cache size in bytes.
icache-size	int	16KB, 32KB, 64KB	0x8000	Set I-cache size in bytes.
min_sync_level	int	0-3	0	Controls the minimum syncLevel.

ac 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.

ad 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.

ae The ase-present and vfp-present parameters configure the synthesis options.

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.

Table 3-33 ARMCortexA9MPxnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
POWERCTLI	int	0-3	0	Default power control state for core.
SMPnAMP	bool	true, false	false	Set whether or not the core is part of a coherent domain. This parameter is a model only parameter, not a synthesize option or a configuration port. In hardware, it is a design choice.
semihosting-cmd_line[af]	string	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	int	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	int	0x00-0xFF	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	int	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[ag]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[ae]	bool	true, false	true	Set whether or not model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled.

ARMCortexA9MPxnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the integration and test registers.

These TLB registers do not have working implementations:

- Main TLB Attr.
- Main TLB PA.
- Main TLB VA.
- Normal memory remap register.
- Primary memory remap register.
- Read Main TLB Entry.
- Write Main TLB Entry.

In addition, the simulation does not distinguish peripheral accesses from data accesses, so it ignores configuration of the peripheral port memory remap register.

[af] The value of argv[0] points to the first command line argument, not to the name of an image.
[ag] This is a model specific behavior with no hardware equivalent.

ARMCortexA9MPxnCT - caches

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

Cache and TLB component visibility

If a core model has a cache model available, to create it and make it visible, enable it. To enable the cache model and be ready to use cache CADI, set these model parameters:

```
l1_icache-state_modelled
l1_dcache-state_modelled
```

To use cache and TLB viewers, connect the cache and TLB CADI components.

ARMCortexA9MPxnCT - debug features

This component exports a CADI debug interface.

ARMCortexA9MPxnCT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger. This component also exports the SCU, Watchdog/Timer and GIC registers.

ARMCortexA9MPxnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA9MPxnCT - debug - memory

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

ARMCortexA9MPxnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA9MPxnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- This 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 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 this model do not cause other cores 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.

3.2.9 ARMCortexA9UPCT component

This section describes the ARMCortexA9UPCT component.

ARMCortexA9UPCT - about

This C++ component is a model of r3p0 of a Cortex-A9 processor.

ARMCortexA9UPCT - ports

This section describes the ports.

Table 3-34 ARMCortexA9UPCT ports

Name	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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA9UPCT - parameters

The parameters are set once.

The processor might have the instance name coretile.core.cpu0, for example.

Table 3-35 ARMCortexA9UPCT cluster parameters

Name	Type	Allowed values	Default value	Description
`CLUSTER_ID`	`int`	0-15	0	Cluster ID value.
`device-accurate-tlb`	`bool`	`true`, `false`	`false`[ah]	Specify whether or not all TLBs are modeled.

Table 3-35 ARMCortexA9UPCT cluster parameters (continued)

Name	Type	Allowed values	Default value	Description
dcache-state_modelled	bool	true, false	false	Set whether or not D-cache has stateful implementation.
icache-state_modelled	bool	true, false	false	Set whether or not I-cache has stateful implementation.

Table 3-36 ARMCortexA9UPCT core parameters

Name	Type	Allowed values	Default value	Description
ase-present[ai]	bool	true, false	true	Set whether or not model has NEON support.
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CFGNMFI	bool	true, false	false	Enable nonmaskable FIQ interrupts on startup.
CP15SDISABLE	bool	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	int	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	int	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
dcache-size	int	16KB, 32KB, 64KB	0x8000	Set D-cache size in bytes.
icache-size	int	16KB, 32KB, 64KB	0x8000	Set I-cache size in bytes.
min_sync_level	int	0-3	0	Controls the minimum syncLevel.
POWERCTLI	int	0-3	0	Default power control state for core.
semihosting-cmd_line[aj]	string	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	int	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	int	0x00-0xFF	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	int	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.

[ah] 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.

[ai] The ase-present and vfp-present parameters configure the synthesis options.

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.

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

Table 3-36 ARMCortexA9UPCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[ak]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[ai]	bool	true, false	true	Set whether or not the model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled.

[ak] This is a model specific behavior with no hardware equivalent.

ARMCortexA9UPCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the integration and test registers.

These TLB registers do not have working implementations:

- Normal memory remap register.
- Primary memory remap register.
- Read Main TLB Entry.
- Write Main TLB Entry.
- Main TLB VA.
- Main TLB PA.
- Main TLB Attr.

In addition, the simulation does not distinguish peripheral accesses from data accesses, so it ignores configuration of the peripheral port memory remap register.

ARMCortexA9UPCT - caches

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

Cache and TLB component visibility

If a core model has a cache model available, to create it and make it visible, enable it. To enable the cache model and be ready to use cache CADI, set these model parameters:

```
l1_icache-state_modelled
l1_dcache-state_modelled
```

To use cache and TLB viewers, connect the cache and TLB CADI components.

ARMCortexA9UPCT - debug features

This component exports a CADI debug interface.

ARMCortexA9UPCT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

ARMCortexA9UPCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA9UPCT - debug - memory

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

ARMCortexA9UPCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA9UPCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- 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.

3.2.10 ARMCortexA8CT component

This section describes the ARMCortexA8CT component.

ARMCortexA8CT - about

This C++ component is a model of r2p1 of the Cortex-A8 processor.

ARMCortexA8CT - ports

This section describes the ports.

Table 3-37 ARMCortexA8CT ports

Name	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

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA8CT - parameters

This section describes the parameters.

Table 3-38 ARMCortexA8CT parameters

Name	Type	Allowed values	Default value	Description
`CFGEND0`	Boolean	`true`, `false`	`false`	Initialize to BE8 endianness.
`CFGNMFI`	Boolean	`true`, `false`	`false`	Enable nonmaskable FIQ interrupts on startup.

Table 3-38 ARMCortexA8CT parameters (continued)

Name	Type	Allowed values	Default value	Description
CFGTE	Boolean	true, false	false	Initialize to take exceptions in T32 state. Model starts in T32 state.
CP15SDISABLE	Boolean	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
l1_dcache-state_modelled[al]	Boolean	true, false	false	Include Level 1 data cache state model.
l1_icache-state_modelled[al]	Boolean	true, false	false	Include Level 1 instruction cache state model.
l2_cache-state_modelled[al]	Boolean	true, false	false	Include unified Level 2 cache state model.
l1_dcache-size	Integer	16KB, 32KB	0x8000	Set L1 D-cache size in bytes.
l1_icache-size	Integer	16KB, 32KB	0x8000	Set L1 I-cache size in bytes.
l2_cache-size	Integer	128KB-1024KB	0x40000	Set L2 cache size in bytes.
device-accurate-tlb	Boolean	true, false	false[am]	Specify whether all TLBs are modeled.
implements_vfp	Boolean	true, false	true	Set whether the model has been built with VFP and NEON support.
master_id	Integer	0x0000-0xFFFF	0x0	master ID presented in bus transactions
min_sync_level	Integer	0-3	0	Controls the minimum syncLevel.
semihosting-ARM_SVC	Integer	uint24_t	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	uint8_t	0xAB	T32 SVC number for semihosting.
semihosting-cmd_line[an]	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F0000000	Virtual address of stack limit.
siliconID	Integer	uint32_t	0x41000000	Value as read by the system coprocessor siliconID register.

[al] If one cache is stateful, then the others must be too.
[am] 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.
[an] The value of argv[0] points to the first command line argument, not to the name of an image.

Table 3-38 ARMCortexA8CT parameters (continued)

Name	Type	Allowed values	Default value	Description
vfp-enable_at_reset[ao]	Boolean	true, false	false	Enable coprocessor access and VFP at reset.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.

ARMCortexA8CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers, the integration and test registers, and the PLE model, which is register based and has no implemented behavior.

This PV model does not model Level 1 or Level 2 caches. The system coprocessor registers related to cache operations permit cache aware software to work, but in most cases they only check register access permissions:

- Cache Dirty Status.
- Data Memory Barrier.
- Data Write Barrier.
- ICache/DCache lockdown.
- ICache/DCache master valid.
- Invalidate and/or Clean Both Caches.
- Invalidate and/or Clean Entire ICache/DCache.
- Invalidate and/or Clean ICache/DCache by Index.
- Invalidate and/or Clean ICache/DCache by MVA.
- Level 1 System array debug registers.
- Level 2 Cache Auxiliary control.
- Level 2 Cache Lockdown.
- Level 2 System array debug registers.
- Prefetch ICache Line.
- Preload Engine registers.

TLBs

These TLB registers do not have working implementations:

- D-TLB ATTR read/write.
- D-TLB CAM read/write.
- D-TLB PA read/write.
- Normal memory remap register.
- Primary memory remap register.

In addition, the simulation does not distinguish peripheral accesses from data accesses, so it ignores configuration of the peripheral port memory remap register.

ARMCortexA8CT - caches

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

ARMCortexA8CT - debug features

This component exports a CADI debug interface.

ARMCortexA8CT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

[ao] This is a model specific behavior with no hardware equivalent.

ARMCortexA8CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA8CT - debug - memory

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

ARMCortexA8CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA8CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- There is a single memory port combining instruction, data, DMA and peripheral access.
- The 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 processor do not cause data aborts.

3.2.11 ARMCortexA7xnCT component

This section describes the ARMCortexA7x*n*CT component.

ARMCortexA7xnCT - about

This C++ component is a model of r0p0 of the Cortex-A7 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexA7xnCT - ports

This section describes the ports.

Table 3-39 ARMCortexA7xnCT ports

Name	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 cluster clock input.
`clusterid`	Value	Slave	Sets the value in the CLUSTERID field (bits[11:8]) of the MPIDR.
`cntvalueb`	CounterInterface	Slave	Synchronous counter value. This must be connected to the MemoryMappedCounterModule component.
`cp15sdisable[0-3]`	Signal	Slave	Disable write access to some secure cp15 registers.

Table 3-39 ARMCortexA7xnCT ports (continued)

Name	Protocol	Type	Description
cpuporeset[0-3]	Signal	Slave	Power on reset. Initializes all the core logic, including the NEON and VFP logic, Debug, PTM, breakpoint and watchpoint logic in the core 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	Core FIQ signal input.
irq[0-3]	Signal	Slave	Core 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	Core reset signal.
standbywfe[0-3]	Signal	Master	Indicates if a core is in *Wait For Event* (WFE) state.
standbywfi[0-3]	Signal	Master	Indicates if a core 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	Core instruction count for visualization.
vfiq[0-3]	Signal	Slave	Core virtual FIQ signal input.
vinithi[0-3]	Signal	Slave	Initialize with high vectors enabled after a reset.
virq[0-3]	Signal	Slave	Core virtual IRQ signal input.
virtio_s	PVBus	Slave	Channel for coherency traffic from virtual model devices. Not for public use.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA7xnCT - parameters

The configuration parameters for this component are set once, irrespective of the number of cores. Each core has its own configuration parameters.

Table 3-40 ARMCortexA7xnCT cluster parameters

Name	Type	Allowed values	Default value	Description
CFGSDISABLE	Boolean	true, false	false	Disable some accesses to GIC registers.
CLUSTER_ID	Integer	0-15	0	Cluster ID value.
dic-spi_count	Integer	0-480, in increments of 32	64	Number of shared peripheral interrupts implemented.

Table 3-40 ARMCortexA7xnCT cluster parameters (continued)

Name	Type	Allowed values	Default value	Description
`internal_vgic`	Boolean	`true`, `false`	`true`	Configures whether or not the model of the cluster contains a VGIC.
`l1_dcache-state_modelled`	Boolean	`true`, `false`	`false`	Set whether or not L1 D-cache has stateful implementation.
`l1_icache-state_modelled`	Boolean	`true`, `false`	`false`	Set whether or not L1 I-cache has stateful implementation.
`l2_cache-size`	Integer	0 (no L2 cache), 128KB, 256KB, 512KB, 1024KB	`0x800000`	Set L2 cache size in bytes.
`l2_cache-state_modelled`	Boolean	`true`, `false`	`false`	Set whether or not L2 cache has stateful implementation.
`PERIPHBASE`	Integer	`0x0000000000-0xFFFFFFFFFF`	`0x13080000`[ap]	Base address of peripheral memory space.

Table 3-41 ARMCortexA7xnCT core parameters

Name	Type	Allowed values	Default value	Description
`ase-present`[aq]	Boolean	`true`, `false`	`true`	Set whether or not CT model has been built with NEON support.
`CFGEND`	Boolean	`true`, `false`	`false`	Initialize to BE8 endianness.
`CP15SDISABLE`	Boolean	`true`, `false`	`false`	Initialize to disable access to some CP15 registers.
`cpi_mul`	Integer	`1-0x7FFFFFFF`	`1`	Multiplier for calculating CPI.
`cpi_div`	Integer	`1-0x7FFFFFFF`	`1`	Divider for calculating *Cycles Per Instruction* (CPI).
`DBGROMADDR`	Integer	`0x00000000-0xFFFFFFFF`	`0x12000003`	This value is used to initialize the CP15 `DBGDRAR` register. Bits[39:12] of this register specify the ROM table physical address.
`DBGROMADDRV`	Boolean	`true`, `false`	`true`	If `true`, this sets bits[1:0] of the CP15 `DBGDRAR` to indicate that the address is valid.
`DBGSELFADDR`	Integer	`0x00010003`	`0x00010003`	This value is used to initialize the CP15 `DBGDSAR` register. Bits[39:17] of this register specify the ROM table physical address.
`DBGSELFADDRV`	Boolean	`true`, `false`	`true`	If `true`, this sets bits[1:0] of the CP15 `DBGDSAR` to indicate that the address is valid.

[ap] 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.

[aq] The ase-present and vfp-present parameters configure the synthesis options.

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.

Table 3-41 ARMCortexA7xnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
`l1_dcache-size`	Integer	`0x2000-0x10000`	`0x8000`	Size of L1 D-cache.
`l1_icache-size`	Integer	`0x2000-0x10000`	`0x8000`	Size of L1 I-cache.
`min_sync_level`	Integer	0-3	0	Controls the minimum syncLevel.
`semihosting-cmd_line`	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
`semihosting-enable`	Boolean	`true`, `false`	`true`	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to `false`.
`semihosting-ARM_SVC`	Integer	`0x000000-0xFFFFFF`	`0x123456`	A32 SVC number for semihosting.
`semihosting-Thumb_SVC`	Integer	`0x00-0xFF`	`0xAB`	T32 SVC number for semihosting.
`semihosting-heap_base`	Integer	`0x00000000-0xFFFFFFFF`	`0x0`	Virtual address of heap base.
`semihosting-heap_limit`	Integer	`0x00000000-0xFFFFFFFF`	`0x0F000000`	Virtual address of top of heap.
`semihosting-stack_base`	Integer	`0x00000000-0xFFFFFFFF`	`0x10000000`	Virtual address of base of descending stack.
`semihosting-stack_limit`	Integer	`0x00000000-0xFFFFFFFF`	`0x0F000000`	Virtual address of stack limit.
`TEINIT`	Boolean	`true`, `false`	`false`	T32 exception enable. The default has exceptions including reset handled in A32 state.
`vfp-enable_at_reset`[ar]	Boolean	`true`, `false`	`false`	Enable coprocessor access and VFP at reset.
`vfp-present`[aq]	Boolean	`true`, `false`	`true`	Set whether or not CT model has been built with VFP support.
`VINITHI`	Boolean	`true`, `false`	`false`	Initialize with high vectors enabled.

ARMCortexA7xnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

ARMCortexA7xnCT - caches

This component implements the L1 and L2 caches as architecturally defined.

ACE limitation

AXI Coherency Extensions (ACE) are extensions to AXI4 that support system-level cache-coherency between multiple clusters. The ACE cache models in the Cortex-A15 and the Cortex-A7, and the ACE support in the CCI-400 have a limitation: these functional models process only one transaction at a time. Normally, the simulation processes each transaction to completion before allowing any master to generate another transaction. However, there is a situation in which the simulation might fail. Suppose a SystemC bus slave calls `wait()` while it is processing a transaction. This call might allow another master to issue another transaction that passes through the CCI-400 or the Cortex-A15/Cortex-A7 caches. This situation could happen if a SystemC bus master running in another thread is connected to one of the ACE-lite ports on the CCI-400.

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

Cache and TLB component visibility

If a core model has a cache model available, to create it and make it visible, enable it. To enable the cache model and be ready to use cache CADI, set these model parameters:

```
l1_icache-state_modelled
l1_dcache-state_modelled
l2_cache-state_modelled
```

To use cache and TLB viewers, connect the cache and TLB CADI components.

ARMCortexA7xnCT - debug features

This component exports a CADI debug interface.

ARMCortexA7xnCT - debug - registers

All core, VFP, CP14, and CP15 registers are visible in the debugger. All CP14 debug registers are implemented.

ARMCortexA7xnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA7xnCT - debug - memory

This component presents three 4GB views of virtual address space: hypervisor, secure, and non-secure.

ARMCortexA7xnCT - verification and testing

This component passes tests using the architecture validation suite tests, booting Linux on an example system containing an ARMCortexA7x*n*CT component, and booting Linux on an example system containing an ARMCortexA7x*n*CT component and an ARMCortexA15x*n*CT *Cache Coherent Interconnect* (CCI400) model.

ARMCortexA7xnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

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

3.2.12 ARMCortexA5MPxnCT component

This section describes the ARMCortexA5MPx*n*CT component.

ARMCortexA5MPxnCT - about

This C++ component is a model of r0p0 of a Cortex-A5 processor. The *n* shows the number of cores.

This component implements 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).

ARMCortexA5MPxnCT - ports

This section describes the ports.

Table 3-42 ARMCortexA5MPxnCT ports

Name	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.
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.

Table 3-42 ARMCortexA5MPxnCT ports (continued)

Name	Protocol	Type	Description
`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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA5MPxnCT - parameters

The parameters are set once, irrespective of the number of cores. Each core has its own parameters.

Table 3-43 ARMCortexA5MPxnCT cluster parameters

Name	Type	Allowed values	Default value	Description
`CLUSTER_ID`	`int`	0-15	0	Cluster ID value.
`CFGSDISABLE`	`bool`	`true`, `false`	`false`	Disable some accesses to GIC registers.
`dcache-state_modelled`	`bool`	`true`, `false`	`false`	Set whether or not D-cache has stateful implementation.
`device-accurate-tlb`	`bool`	`true`, `false`	`false`[as]	Specify whether or not all TLBs are modeled.
`dic-spi_count`	`int`	0-223, in increments of 32	64	Number of shared peripheral interrupts implemented.
`FILTEREN`	`bool`	`true`, `false`	`false`	Enable filtering of accesses through pvbus_m0.
`FILTERSTART`	`int`	Align on 1MB boundary.	`0x0`	Base of region filtered to pvbus_m0.
`FILTEREND`	`int`	Align on 1MB boundary.	`0x0`	End of region filtered to pvbus_m0.
`icache-state_modelled`	`bool`	`true`, `false`	`false`	Set whether or not I-cache has stateful implementation.
`PERIPHBASE`	`int`	-	`0x13080000`[at]	Base address of peripheral memory space.

[as] 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.
[at] If you are using this component on a VE model platform, this parameter is set automatically to `0x1F000000` and is not visible in the parameter list.

Table 3-44 ARMCortexA5MPxnCT core parameters

Name	Type	Allowed values	Default value	Description
ase-present[au]	bool	true, false	true	Set whether or not the model has NEON support.
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CFGNMFI	bool	true, false	false	Enable nonmaskable FIQ interrupts on startup.
CP15SDISABLE	bool	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	int	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	int	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
dcache-size	int	4KB, 8KB, 16KB, 32KB, 64KB	0x8000	Set D-cache size in bytes.
icache-size	int	4KB, 8KB, 16KB, 32KB, 64KB	0x8000	Set I-cache size in bytes.
min_sync_level	int	0-3	0	Controls the minimum syncLevel.
POWERCTLI	int	0-3	0	Default power control state for core.
SMPnAMP	bool	true, false	false	Set whether or not the core is part of a coherent domain. This parameter is a model only parameter, not a synthesize option or a configuration port. In hardware, it is a design choice.
semihosting-cmd_line	string	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	int	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	int	0x00-0xFF	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	int	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.

[au] The ase-present and vfp-present parameters configure the synthesis options.

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.

Table 3-44 ARMCortexA5MPxnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[av]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[au]	bool	true, false	true	Set whether or not model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled.

ARMCortexA5MPxnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

ARMCortexA5MPxnCT - caches

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

Cache and TLB component visibility

If a core model has a cache model available, to create it and make it visible, enable it. To enable the cache model and be ready to use cache CADI, set these model parameters:

```
l1_icache-state_modelled
l1_dcache-state_modelled
```

To use cache and TLB viewers, connect the cache and TLB CADI components.

ARMCortexA5MPxnCT - debug features

This component exports a CADI debug interface.

ARMCortexA5MPxnCT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

This component also exports the SCU, Watchdog/Timer and GIC registers.

ARMCortexA5MPxnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA5MPxnCT - debug - memory

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

[av] This is a model specific behavior with no hardware equivalent.

ARMCortexA5MPxnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA5MPxnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- There is a single memory port combining instruction, data, DMA and peripheral access.
- This 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 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 this model do not cause other cores 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.

3.2.13 ARMCortexA5CT component

This section describes the ARMCortexA5CT component.

ARMCortexA5CT - about

This C++ component is a model of r0p0 of a Cortex-A5 processor.

ARMCortexA5CT - ports

This section describes the ports.

Table 3-45 ARMCortexA5CT ports

Name	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.

Table 3-45 ARMCortexA5CT ports (continued)

Name	Protocol	Type	Description
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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexA5CT - parameters

This section describes the parameters.

Table 3-46 ARMCortexA5CT cluster parameters

Name	Type	Allowed values	Default value	Description
CLUSTER_ID	int	0-15	0	Cluster ID value.
device-accurate-tlb	bool	true, false	false[aw]	Specify whether or not all TLBs are modeled.
dcache-state_modelled	bool	true, false	false	Set whether or not D-cache has stateful implementation.
icache-state_modelled	bool	true, false	false	Set whether or not I-cache has stateful implementation.

Table 3-47 ARMCortexA5CT core parameters

Name	Type	Allowed values	Default value	Description
ase-present[ax]	bool	true, false	true	Set whether or not model has NEON support.
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CFGNMFI	bool	true, false	false	Enable nonmaskable FIQ interrupts on startup.
CP15SDISABLE	bool	true, false	false	Initialize to disable access to some CP15 registers.
cpi_div	int	1-0x7FFFFFFF	1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	int	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
dcache-size	int	4KB, 8KB, 16KB, 32KB, 64KB	0x8000	Set D-cache size in bytes.
icache-size	int	4KB, 8KB, 16KB, 32KB, 64KB	0x8000	Set I-cache size in bytes.

[aw] 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.

[ax] The ase-present and vfp-present parameters configure the synthesis options.

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.

Table 3-47 ARMCortexA5CT core parameters (continued)

Name	Type	Allowed values	Default value	Description
min_sync_level	int	0-3	0	Controls the minimum syncLevel.
POWERCTLI	int	0-3	0	Default power control state for core.
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
semihosting-cmd_line	string	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	int	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-heap_base	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	int	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
semihosting-Thumb_SVC	int	0x00-0xFF	0xAB	T32 SVC number for semihosting.
vfp-enable_at_reset[ay]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[ax]	bool	true, false	true	Set whether or not the model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled.

[ay] This is a model specific behavior with no hardware equivalent.

ARMCortexA5CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

ARMCortexA5CT - caches

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

Cache and TLB component visibility

If a core model has a cache model available, to create it and make it visible, enable it. To enable the cache model and be ready to use cache CADI, set these model parameters:

```
l1_icache-state_modelled
l1_dcache-state_modelled
```

To use cache and TLB viewers, connect the cache and TLB CADI components.

ARMCortexA5CT - debug features

This component exports a CADI debug interface.

ARMCortexA5CT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

ARMCortexA5CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexA5CT - debug - memory

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

ARMCortexA5CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexA5CT - differences between the CT model and RTL implementations

This component differs from the corresponding RTL implementation.

- 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.

3.3 Cortex-R processor components

This section describes the Cortex-R processor components.

This section contains the following subsections:

3.3.1 ARMCortexR7MPxnCT component

This section describes the ARMCortexR7MPx*n*CT component.

ARMCortexR7MPxnCT - about

This C++ component is a model of r0p0 of a Cortex-R7 processor containing from one to four cores. The *n* shows the number of cores.

ARMCortexR7MPxnCT - ports

This section describes the ports.

Table 3-48 ARMCortexR7MPxnCT ports

Name	Protocol	Type	Description
`acp_s`	PVBus	Slave	Slave channel.
`cfgend`	Signal	Slave	Initialize to BE8 endianness after a reset.
`cfgnmfi`	Signal	Slave	Enable non-maskable FIQ interrupts after a reset.
`clk_in`	ClockSignal	Slave	Main cluster clock input.
`clusterid`	Value	Slave	Value in MPIDR register.
`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	Core FIQ signal input.
`fiqout[0-1]`	Signal	Master	Output of core 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	Core IRQ signal input.
`irqout[0-1]`	Signal	Master	Output of core nIRQ from the interrupt controller.
`periphbase`	Value	Slave	Base of private peripheral region.
`periphclk_in`	ClockSignal	Slave	Timer or 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 core.
`pvbus_m0`	PVBus	Master	AXI master 0 bus master channel.
`pvbus_s`	PVBus	Slave	TCM slave port.
`reset[0-1]`	Signal	Slave	Core reset signal.

Table 3-48 ARMCortexR7MPxnCT ports (continued)

Name	Protocol	Type	Description
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 cores are in SMP mode.
standbywfe[0-1]	Signal	Master	Indicates if a core is in WFE state.
standbywfi[0-1]	Signal	Master	Indicates if a core is in WFI state.
teinit	Signal	Slave	Initialize to take exceptions in T32 state after a reset.
ticks[0-1]	InstructionCount	Master	Core instruction count for visualization.
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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexR7MPxnCT - parameters

The parameters are set once, irrespective of the number of cores. Each core has its own parameters.

The first core might have the instance name coretile.core.cpu0, for example, where “coretile.core” and “cpu0” are the normal instance name and the first core, respectively.

Table 3-49 ARMCortexR7MPxnCT cluster parameters

Name	Type	Allowed values	Default value	Description
CLUSTER_ID	int	0x0-0xF	0x0	Cluster ID value.
dcache-state_modelled	bool	true, false	false	Set whether or not D-cache has stateful implementation.
dic-spi_count	int	0x0-0xE	0x40	Number of shared peripheral interrupts implemented.
ecc_on	bool	true, false	false	Enable Error Correcting Code.
icache-state_modelled	bool	true, false	false	Set whether or not I-cache has stateful implementation.
LOCK_STEP	int	0 - Disable. Set for two independent cores. 1 - Lock Step. Appears to the system as two cores but is internally modeled as a single core. 3 - Split Lock. Appears to the system as two cores but can be statically configured from reset either as two independent cores or two locked cores. 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 cluster.	0	Affects dual-core configurations only, and ignored by single-core configurations.

Table 3-49 ARMCortexR7MPxnCT cluster parameters (continued)

Name	Type	Allowed values	Default value	Description
MFILTEREN	bool	true, false	false	Enables filtering of address ranges.
MFILTEREND	int	0x00000000-0xFFFFFFFF	0x0	Specifies the end address for address filtering.
MFILTERSTART	int	0x00000000-0xFFFFFFFF	0x0	Specifies the start address for address filtering.
PERIPHBASE	int	0x00000000-0xFFFFFFFF	0xAE000000	Base address of peripheral memory space.

Table 3-50 ARMCortexR7MPxnCT core parameters

Name	Type	Allowed values	Default value	Description
CFGEND	bool	true, false	false	Initialize to BE8 endianness.
CFGNMFI	bool	true, false	false	Enable nonmaskable FIQ interrupts on startup.
cpi_div	int	0x1-0x7FFFFFFF	0x1	Divider for calculating *Cycles Per Instruction* (CPI).
cpi_mul	int	0x1-0x7FFFFFFF	0x1	Multiplier for calculating CPI.
dcache-size	int	0x00000000-0x10000000	0x8000	Set D-cache size in bytes.
DP_FLOAT	bool	true, false	true	Sets whether or not double-precision instructions are available.
icache-size	int	0x00000000-0x10000000	0x8000	Set I-cache size in bytes.
min_sync_level	int	0-3	0	Controls the minimum syncLevel.
NUM_MPU_REGION	int	12-16	16	Sets the number of MPU regions.
POWERCTLI	int	0-3	0	Default power control state for core.
semihosting-ARM_SVC	int	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-cmd_line[az]	string	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	int	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	int	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
semihosting-Thumb_SVC	int	0x00-0xFF	0xAB	T32 SVC number for semihosting.

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

Table 3-50 ARMCortexR7MPxnCT core parameters (continued)

Name	Type	Allowed values	Default value	Description
SMPnAMP	bool	true, false	false	Set whether or not the core is part of a coherent domain. This parameter is a model only parameter, not a synthesize option or a configuration port. In hardware, it is a design choice.
tcm-present	bool	true, false	true	Disables the DTCM and ITCM.
TEINIT	bool	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[ba]	bool	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present[az]	bool	true, false	true	Set whether or not model has VFP support.
VINITHI	bool	true, false	false	Initialize with high vectors enabled.

ARMCortexR7MPxnCT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the integration and test registers.

ARMCortexR7MPxnCT - caches

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

ARMCortexR7MPxnCT - debug features

This component exports a CADI debug interface.

ARMCortexR7MPxnCT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

This component also exports the SCU, Watchdog/Timer and GIC registers.

ARMCortexR7MPxnCT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexR7MPxnCT - debug - memory

This component presents two 4GB views of physical memory, one as seen from secure mode and one as seen from normal mode.

[ba] This is a model specific behavior with no hardware equivalent.

ARMCortexR7MPxnCT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARMCortexR7MPxnCT - differences between the CT model and RTL implementations

This component differs from the corresponding revision RTL implementation.

- This 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 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 this model do not cause other cores 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.

3.3.2 ARMCortexR5CT component

This section describes the ARMCortexR5CT component.

ARMCortexR5CT - about

This C++ component is a model of r1p2 of a Cortex-R5 processor containing from one to two cores. The *n* shows the number of cores.

ARMCortexR5CT - ports

This section describes the ports.

Table 3-51 ARMCortexR5CT ports

Name	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 non-maskable FIQ interrupts.
`clk_in`	ClockSignal	Slave	The clock signal that is 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.
`groupid`	Value	Slave	Group ID used for MPIDR.
`initrama[2]`	Signal	Slave	Usable if ATCM is enabled at reset.
`initramb[2]`	Signal	Slave	Usable if BTCM is enabled at reset.
`irq[2]`	Signal	Slave	This signal drives the interrupt handling of the processor.

Table 3-51 ARMCortexR5CT ports (continued)

Name	Protocol	Type	Description
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.
pvbus_s	PVBus	Slave	TCM slave 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	To display a running instruction count, connect this port to one of the two ticks ports on a visualization component.
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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexR5CT - parameters

The parameters are set once, irrespective of the number of cores. Each core has its own parameters.

Table 3-52 ARMCortexR5CT cluster parameters

Name	Type	Allowed values	Default value	Description
dcache-state_modelled	Boolean	true, false	false	Set whether D-cache has stateful implementation.
GROUP_ID	Integer	0-15	0	Value read in GROUP ID register field, bits[15:8] of the MPIDR.
icache-state_modelled	Boolean	true, false	false	Set whether I-cache has stateful implementation.
INST_ENDIAN	Boolean	true, false	true	Controls whether the model supports the instruction endianness bit.

Table 3-52 ARMCortexR5CT cluster parameters (continued)

Name	Type	Allowed values	Default value	Description
LOCK_STEP	Integer	0 - Disable. Set for two independent cores. 1 - Lock Step. Appears to the system as two cores but is internally modeled as a single core. 3 - Split Lock. Appears to the system as two cores but can be statically configured from reset either as two independent cores or two locked cores. 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 cluster.	0	Affects dual-core configurations only, and ignored by single-core configurations.
MICRO_SCU	Boolean	true, false	true	Controls whether the effects of the MicroSCU are modeled.
NUM_BREAKPOINTS	Integer	2-8	3	Controls with how many breakpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.
NUM_WATCHPOINTS	Integer	1-8	2	Controls with how many watchpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.
SLSPLIT	Boolean	true, 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	Sets whether the model starts in split mode or locked mode. Contact ARM for details.

Table 3-53 ARMCortexR5CT core parameters

Name	Type	Allowed values	Default value	Description
atcm_base[bb]	Integer	0x00000000-0xFFFFFFFF	0x40000000	Model-specific. Sets the base address of the ATCM.
btcm_base[bb]	Integer	0x00000000-0xFFFFFFFF	0x00000000	Model-specific. Sets the base address of the BTCM.
CFGATCMSZ	Integer	0x00000000-0xE	0xE	Sets the size of the ATCM.
CFGBTCMSZ	Integer	0x00000000-0xE	0xE	Sets the size of the BTCM.
CFGEND	Boolean	true, false	false	Initialize to BE8 endianness.

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

Table 3-53 ARMCortexR5CT core parameters (continued)

Name	Type	Allowed values	Default value	Description
CFGIE	Boolean	true, false	false	Set the reset value of the instruction endian bit.
CFGNMFI	Boolean	true, false	false	Enable nonmaskable FIQ interrupts on startup.
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating *Cycle Per Instruction* (CPI).
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating CPI.
dcache-size	Integer	0x1000-0x10000	0x10000	Set D-cache size in bytes.
DP_FLOAT	Boolean	true, false. If true, then double precision VFP is supported. If false, then the VFP is single precision only.	true	Sets whether double-precision instructions are available.
icache-size	Integer	0x1000-0x10000	0x10000	Set I-cache size in bytes.
min_sync_level	Integer	0-3	0	Controls the minimum syncLevel.
NUM_MPU_REGION	Integer	0x00, 0xC, 0x10. 0 = no MPU.	0xC	Sets the number of MPU regions.
semihosting-ARM_SVC	Integer	0x000000-0xFFFFFF	0x123456	A32 SVC number for semihosting.
semihosting-cmd_line	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of stack limit.
semihosting-Thumb_SVC	Integer	0x00-0xFF	0xAB	T32 SVC number for semihosting.
TEINIT	Boolean	true, false	false	T32 exception enable. The default has exceptions including reset handled in A32 state.
vfp-enable_at_reset[bb]	Boolean	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present	Boolean	true, false	true	Set whether model has VFP support.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.

ARMCortexR5CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

ARMCortexR5CT - caches

This component implements an L1 cache as architecturally defined, but does not implement an L2 cache.

ARMCortexR5CT - debug features

This component exports a CADI debug interface.

ARMCortexR5CT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

ARMCortexR5CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexR5CT - debug - memory

This component presents a single 4GB view of memory.

ARMCortexR5CT - verification and testing

This component passes tests using the architecture validation suite tests.

ARMCortexR5CT - performance

This component provides high performance in all areas except VFP and instruction set execution.

ARMCortexR5CT - differences between the CT model and RTL implementations

This component differs from the corresponding RTL implementation.

- 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 ARM does not provide the ability to set error schemes for the caches or TCMs. Contact ARM if you require a particular value in the Build Options registers.

3.3.3 ARMCortexR4CT component

This section describes the ARMCortexR4CT component.

ARMCortexR4CT - about

This C++ component is a model of r1p2 of a Cortex-R4 processor.

ARMCortexR4CT - ports

This section describes the ports.

Table 3-54 ARMCortexR4CT ports

Name	Protocol	Type	Description
`cfgie`[bc]	Signal	Slave	Configure instruction endianness after a reset.
`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.
`clk_in`	ClockSignal	Slave	Clock input.
`dtcm`	PVBus	Slave	Slave access to DTCM.
`fiq`	Signal	Slave	Asynchronous FIQ signal input.
`initrami`	Signal	Slave	Initialize with ITCM enabled after reset.
`initramd`	Signal	Slave	Initialize with DTCM enabled after reset.
`irq`	Signal	Slave	Asynchronous IRQ signal input.
`itcm`	PVBus	Slave	Slave access to ITCM.
`pmuirq`	Signal	Master	Performance monitoring unit IRQ output.
`pvbus_m`	PVBus	Master	Master port for all memory accesses.
`reset`	Signal	Slave	Asynchronous reset signal input.
`standbywfi`	Signal	Master	Signal that the processor is in standby waiting for interrupts.
`ticks`	InstructionCount	Master	Output that can be connected to a visualization component.
`vic_ack`	Signal	Master	Acknowledge signal output for PL192 VIC.
`vic_addr`	ValueState	Slave	Address input for connection to PL192 VIC. The port expects a full 32-bit address.
`vinithi`	Signal	Slave	Initialize with high vectors enabled after a reset.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexR4CT - parameters

This section describes the parameters.

Table 3-55 ARMCortexR4CT parameters

Name	Type	Allowed values	Default value	Description
`CFGEND0`	Boolean	`true`, `false`	`false`	Initialize to BE8 endianness.
`CFGIE`	Boolean	`true`, `false`	`false`	Configure instructions as big endian.
`CFGNMFI`	Boolean	`true`, `false`	`false`	Enable nonmaskable FIQ interrupts on startup.

[bc] The model implements this, although it is optional in hardware.

Table 3-55 ARMCortexR4CT parameters (continued)

Name	Type	Allowed values	Default value	Description
CFGTE	Boolean	true, false	false	Initialize to take exceptions in T32 state. Model starts in T32 state.
INITRAMD	Boolean	true, false	false	Set or reset the INITRAMD signal.
INITRAMI	Boolean	true, false	false	Set or reset the INITRAMI signal.
LOCZRAMI	Boolean	true, false	false	Set or reset the LOCZRAMI signal.
NUM_MPU_REGION	Integer	0, 8, 12	8	Number of MPU regions.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.
dcache-size	Integer	4KB, 8KB, 16KB, 32KB, 64KB	0x10000	Set D-cache size in bytes.
dcache-state_modelled	Boolean	true, false	false	Set whether D-cache has stateful implementation.
dtcm0_base	Integer	uint32_t	0x00800000	Base address of DTCM at startup.
dtcm0_size	Integer	0x0000 - 0x2000	0x8	Size of DTCM in KB.
icache-size	Integer	4KB, 8KB, 16KB, 32KB, 64KB	0x10000	Set I-cache size in bytes.
icache-state_modelled	Boolean	true, false	false	Set whether I-cache has stateful implementation.
implements_vfp	Boolean	true, false	true	Set whether the model has been built with VFP support.
itcm0_base	Integer	uint32_t	0x00000000	Base address of ITCM at startup.
itcm0_size	Integer	0x0000 - 0x2000	0x8	Size of ITCM in KB.
master_id	Integer	0x0000 - 0xFFFF	0x0	Master ID presented in bus transactions
min_sync_level	Integer	0-3	0	Controls the minimum syncLevel.
semihosting-ARM_SVC	Integer	24-bit integer	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	8-bit integer	0xAB	T32 SVC number for semihosting.
semihosting-cmd_line[bd]	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	Integer	0x00000000 - 0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000 - 0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000 - 0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000 - 0xFFFFFFFF	0x0F0000000	Virtual address of stack limit.

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

Table 3-55 ARMCortexR4CT parameters (continued)

Name	Type	Allowed values	Default value	Description
vfp-enable_at_reset[be]	Boolean	true, false	false	Enable coprocessor access and VFP at reset.
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating *Cycle Per Instruction* (CPI).
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating CPI.

ARMCortexR4CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

This PV model does not model Level 1 or Level 2 caches. The system coprocessor registers related to cache operations permit cache aware software to work, but in most cases they only check register access permissions:

- 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.
- Cache Size Override.
- Validation registers.

ARMCortexR4CT - caches

This component implements a PV-accurate cache view.

ARMCortexR4CT - debug features

This component exports a CADI debug interface.

ARMCortexR4CT - debug - registers

All core and implemented registers are visible in the debugger.

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

ARMCortexR4CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexR4CT - debug - memory

This component presents one 4GB view of virtual memory.

[be] This is a model specific behavior with no hardware equivalent.

ARMCortexR4CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of uClinux on an example system.

ARMCortexR4CT - performance

This component provides high performance in all areas except with instructions in protection regions smaller than 1KB, and VFP instruction set execution.

ARMCortexR4CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- 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 core 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.

3.4 Cortex-M processor components

This section describes the Cortex-M processor components.

This section contains the following subsections:

3.4.1 ARMCortexM7CT component

This C++ component is a model of an *Early Access Candidate* (EAC), r0p2, of a Cortex-M7 core.

ARMCortexM7CT - ports

This section describes the ports.

Table 3-56 ARMCortexM7CT ports

Name	Protocol	Type	Description
ahbd	PVBus	Slave	Debug AHB: core bus slave driven by the DAP.
ahbs	PVBus	Slave	External master (e.g. DMA) can write TCMs (whether or not enabled in xTCMCR).
auxfault	Value	Slave	Wired to the Auxiliary Fault Status Register.
bigend	Signal	Slave	Configure big endian data format.
clk_in	ClockSignal	Slave	The clock signal connected to the clk_in port is used to determine the rate at which the core executes instructions.
coreconfig	ValueState	Master	Validation system only: allow querying for the core's config.
cpuwait	Signal	Slave	-
currpri	Value	Master	Current execution priority.
dap_s	PVBus	Slave	Debug Access Port (DAP).
dbgen	Signal	Slave	Disallow (DAP) debugger access.
dbgrestart	Signal	Slave	-
dbgrestarted	Signal	Master	-
edbgrq	Signal	Slave	External debug request.
event	Signal	Peer	This peer port of event input (and output) is for wakeup from WFE and corresponds to the RTL TXEV and RXEV signals.
fpudisable	Signal	Slave	Configure core with no FPU on reset.
fpxxc	Value	Master	Port which sends the value of the FPXXC cumulative exception flags.
halted	Signal	Master	-
intisr[240]	Signal	Slave	This signal array delivers signals to the NVIC.
intnmi	Signal	Slave	Configure nonmaskable interrupt.
lockup	Signal	Master	Asserted when the processor is in lockup state.
mpudisable	Signal	Slave	Configure core with no MPU on reset.
niden	Signal	Slave	-
poreset	Signal	Slave	Raising this signal power-on resets the core.

Table 3-56 ARMCortexM7CT ports (continued)

Name	Protocol	Type	Description
pv_ppbus_m	PVBus	Master	The core generates External Private Peripheral Bus requests on this port.
pvbus_m	PVBus	Master	The core generates bus requests on this port.
sleepdeep	Signal	Master	Asserted when the processor is in deep sleep.
sleeping	Signal	Master	Asserted when the processor is in sleep.
stcalib	Value	Slave	Calibration value for the SysTick timer.
stclk	ClockSignal	Slave	Reference clock for the SysTick timer.
sysreset	Signal	Slave	Raising this signal will put the core into reset mode (but does not reset the debug logic).
sysresetreq	Signal	Master	Asserted to indicate that a reset is required.
ticks	InstructionCount	Master	Port allowing the number of instructions since startup to be read from the CPU.

ARMCortexM7CT - parameters

This section describes the parameters.

Table 3-57 ARMCortexM7CT parameters

Name	Type	Allowed values	Default value	Description
BIGENDINIT	bool	true, false	false	Initialize core to big-endian mode.
cpi_div	uint32_t	0x1-0x7FFFFFFF	0x1	Divider for calculating CPI (Cycle Per Instruction). Runtime parameter.
cpi_mul	uint32_t	0x1-0x7FFFFFFF	0x1	Multiplier for calculating CPI (Cycle Per Instruction). Runtime parameter.
DBGLVL	uint32_t	0x0-0x1	0x1	0: 2 DWT, 4 FPB; 1: 4 DWT, 8 FPB comparators.
dcache-size	uint32_t	0x0-0x100000	0x8000	L1 D-cache size in bytes.
dcache-state_modelled	bool	true, false	false	Set whether D-cache has stateful implementation.
dcache-ways	uint32_t	1-64	4	L1 D-cache ways (sets are implicit from size).
DP_FLOAT	bool	true, false	true	Support 64-bit floats in VFP.
dtcm_enable	bool	true, false	false	DTCM enabled on reset.
dtcm_size	uint32_t	0x1-0x4000	0x100	DTCM size in KB.
icache-size	uint32_t	0x0-0x100000	0x8000	L1 I-cache size in bytes.
icache-state_modelled	bool	true, false	false	Set whether I-cache has stateful implementation.
icache-ways	uint32_t	1-64	2	L1 I-cache ways (sets are implicit from size.
ignore_imprecise_aborts	bool	true, false	false	Suppress effects of imprecise data-aborts.
INITVTOR	uint32_t	0x0-0xFFFFFF80	0x0	Vector-table offset at reset.
itcm_enable	bool	true, false	false	ITCM enabled on reset.
itcm_size	uint32_t	0x1-0x4000	0x100	ITCM size in KB.
LVL_WIDTH	uint32_t	0x3-0x8	0x3	Number of bits of interrupt priority.
master_id	uint32_t	0x0-0xFFFFFFFF	0x0	Master ID presented in bus transactions.

Table 3-57 ARMCortexM7CT parameters (continued)

Name	Type	Allowed values	Default value	Description
min_sync_level	uint32_t	0x0-0x3	0x0	Force minimum syncLevel (0 = off = default, 1 = syncState, 2 = postInsnIO, 3 = postInsnAll). Runtime parameter.
NUM_IRQ	uint32_t	0x0-0xF0	0x10	Number of user interrupts.
NUM_MPU_REGION	uint32_t	0-16	16	Number of MPU regions.
scheduler_mode	uint32_t	0x0-0xFFFFFFFF	0x0	Control the interleaving of instructions in this processor (0 = default long quantum, 1 = low latency mode, short quantum and signal checking, 2 = lock-breaking mode, long quantum with additional context switches near load-exclusive instructions).
semihosting-cmd_line	string	-	""	Command line available to semihosting SVC calls.
semihosting-cwd	string	-	""	-
semihosting-enable	bool	true, false	true	Enable semihosting SVN traps.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x10700000	Virtual address of top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10700000	Virtual address of base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x10800000	Virtual address of stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFFFFFFFF	0xAB	Thumb SVC number for semihosting.
vfp-present	bool	true, false	true	Set whether model has VFP support.
WIC	bool	true, false	true	Include support for WIC-mode deep sleep.

3.4.2 ARMCortexM4CT component

This section describes the ARMCortexM4CT component.

ARMCortexM4CT - about

This C++ component is a model of r0p0 of a Cortex-M4 processor.

ARMCortexM4CT - ports

This section describes the ports.

Table 3-58 ARMCortexM4CT ports

Name	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	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 RXEV signals.

Table 3-58 ARMCortexM4CT ports (continued)

Name	Protocol	Type	Description
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.
reset	Signal	Slave	Asynchronous reset signal input (not debug components).
sleepdeep	Signal	Master	Processor is in deep sleep.
sleeping	Signal	Master	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[bf]	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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexM4CT - parameters

This section describes the parameters.

Table 3-59 ARMCortexM4CT parameters

Name	Type	Allowed values	Default value	Description
BB_PRESENT	Boolean	true, false	true	Enable bitbanding.
BIGENDINIT	Boolean	true, false	false	Initialize processor to big endian mode.
LVL_WIDTH	Integer	3-8	3	Number of bits of interrupt priority.
NUM_IRQ	Integer	1-240	16	Number of user interrupts.
NUM_MPU_REGION	Integer	0, 8	8	Number of MPU regions.
master_id	Integer	0x0000-0xFFFF	0x0	Master ID presented in bus transactions.

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

Table 3-59 ARMCortexM4CT parameters (continued)

Name	Type	Allowed values	Default value	Description
`min_sync_level`	Integer	0-3	0	Controls the minimum syncLevel.
`semihosting-Thumb_SVC`	Integer	8-bit integer	`0xAB`	T32 SVC number for semihosting.
`semihosting-cmd_line`	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
`semihosting-enable`	Boolean	`true`, `false`	`true`	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to `false`.
`semihosting-heap_base`	Integer	`0x00000000-0xFFFFFFFF`	`0x0`	Virtual address of heap base.
`semihosting-heap_limit`	Integer	`0x00000000-0xFFFFFFFF`	`0x10700000`	Virtual address of top of heap.
`semihosting-stack_base`	Integer	`0x00000000-0xFFFFFFFF`	`0x10700000`	Virtual address of base of descending stack.
`semihosting-stack_limit`	Integer	`0x00000000-0xFFFFFFFF`	`0x10800000`	Virtual address of stack limit.
`vfp-present`	Boolean	`true`, `false`	`true`	Set whether the model has VFP support.
`cpi_mul`	Integer	`1-0x7FFFFFFF`	`1`	Multiplier for calculating *Cycles Per Instruction* (CPI).
`cpi_div`	Integer	`1-0x7FFFFFFF`	`1`	Divider for calculating CPI.

ARMCortexM4CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the processor debug registers, system debug registers, debug interface port registers, TPIU registers, and ETM registers.

ARMCortexM4CT - caches

This component does not implement any caches.

ARMCortexM4CT - debug features

This component exports a CADI debug interface.

ARMCortexM4CT - debug - registers

All core and implemented registers are visible in the debugger.

ARMCortexM4CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexM4CT - debug - memory

This component presents one 4GB view of virtual memory.

ARMCortexM4CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of uClinux on an example system.

ARMCortexM4CT - performance

This component provides high performance in all areas except with instructions in protection regions smaller than 1KB, and FP instruction set execution.

ARMCortexM4CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- The *Wakeup Interrupt Controller* (WIC) is not 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).
- No debug-related components are implemented.
- The unimplemented registers are the processor debug registers, system debug registers, debug interface port registers, TPIU registers, and ETM registers.
- 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.

3.4.3 ARMCortexM3CT component

This section describes the ARMCortexM3CT component.

ARMCortexM3CT - about

This C++ component is a model of r2p1 of a Cortex-M3 processor.

ARMCortexM3CT - ports

This section describes the ports.

Table 3-60 ARMCortexM3CT ports

Name	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.

Table 3-60 ARMCortexM3CT ports (continued)

Name	Protocol	Type	Description
event	Signal	Peer	Event input and output for wakeup from WFE. This port combines the TXEV and RXEV signals.
intisr[0-239]	Signal	Slave	External interrupt signals.
intnmi	Signal	Save	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.
pv_ppbus_m	PVBus	Master	Master port for memory accesses on the External Private Peripheral Bus.
sleepdeep	Signal	Master	Processor is in deep sleep.
sleeping	Signal	Master	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.

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARMCortexM3CT - parameters

This section describes the parameters.

Table 3-61 ARMCortexM3CT parameters

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

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

Table 3-61 ARMCortexM3CT parameters (continued)

Name	Type	Allowed values	Default value	Description
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	Integer	0x00000000 - 0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000 - 0xFFFFFFFF	0x10700000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000 - 0xFFFFFFFF	0x10700000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000 - 0xFFFFFFFF	0x10800000	Virtual address of stack limit.
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating *Cycle Per Instruction* (CPI).
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating CPI.

ARMCortexM3CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the processor debug registers, system debug registers, debug interface port registers, TPIU registers, and ETM registers.

ARMCortexM3CT - caches

This component does not implement any caches.

ARMCortexM3CT - debug features

This component exports a CADI debug interface.

ARMCortexM3CT - debug - registers

All core and implemented registers are visible in the debugger.

ARMCortexM3CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARMCortexM3CT - debug - memory

This component presents one 4GB view of virtual memory.

ARMCortexM3CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of uClinux and RTX on an example system.

ARMCortexM3CT - performance

This component provides high performance except with instructions in protection regions smaller than 1KB.

ARMCortexM3CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

- 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.
- The unimplemented registers are the processor debug registers, system debug registers, debug interface port registers, TPIU registers, and ETM registers.
- 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.

3.4.4 ARMCortexM0PlusCT component

This section describes the ARMCortexM0PlusCT component.

ARMCortexM0PlusCT - about

This C++ component models r0p1 of a Cortex-M0+ core.

This model does not have a parameter that is equivalent to the RAR integration option. The architecturally required register state is reset.

ARM does not guarantee that all ARMv7-M behavior is absent from models of ARMv6-M cores. As a consequence, ARM does not guarantee that code that runs on ARMv7-M cores but fails on ARMv6-M cores fails on ARMv6-M Fast Models cores.

ARMCortexM0PlusCT - ports

This section describes the ports.

Table 3-62 ARMCortexM0PlusCT ports

Name	Protocol	Type	Description
`auxfault`	Value	Slave	This is wired to the Auxiliary Fault Status Register.
`bigend`	Signal	Slave	Configure big endian data format.
`clk_in`	ClockSignal	Slave	The clock signal connected to the clk_in port is used to determine the rate at which the core executes instructions.

Table 3-62 ARMCortexM0PlusCT ports (continued)

Name	Protocol	Type	Description
currpri	Value	Master	Current execution priority.
edbgrq	Signal	Slave	External debug request.
event	Signal	Peer	This peer port of event input (and output) is for wakeup from WFE and corresponds to the RTL TXEV and RXEV signals.
intisr[32]	Signal	Slave	This signal array delivers signals to the NVIC.
intnmi	Signal	Slave	Configure nonmaskable interrupt.
lockup	Signal	Master	Asserted when the processor is in lockup state.
poreset	Signal	Slave	Raising this signal will do a power-on reset of the core.
pv_ppbus_m	PVBus	Master	The core will generate External Private Peripheral Bus requests on this port.
pvbus_m	PVBus	Master	The core will generate bus requests on this port.
sleepdeep	Signal	Master	Asserted when the processor is in deep sleep.
sleeping	Signal	Master	Asserted when the processor is in sleep.
stcalib	Value	Slave	This is the calibration value for the SysTick timer.
stclk	ClockSignal	Slave	This is the reference clock for the SysTick timer.
sysreset	Signal	Slave	Raising this signal will put the core into reset mode (but does not reset the debug logic).
sysresetreq	Signal	Master	Asserted to indicate that a reset is required.
ticks	InstructionCount	Master	Port allowing the number of instructions since startup to be read from the CPU.

ARMCortexM0PlusCT - parameters

This section describes the parameters.

Table 3-63 ARMCortexM0PlusCT parameters

Name	Type	Allowed values	Default value	Description
BIGENDINIT	bool	true, false	false	Initialize processor to big endian mode.
BKPT	uint32_t	0x0-0x4	0x4	The number of breakpoint unit comparators. Runtime parameter.
cpi_div	uint32_t	0x1-0x7FFFFFFF	0x1	Divider for calculating CPI (Cycles Per Instruction). Runtime parameter.
cpi_mul	uint32_t	0x1-0x7FFFFFFF	0x1	Multiplier for calculating CPI (Cycles Per Instruction). Runtime parameter.
DBG	bool	true, false	true	Whether or not the debug extensions are implemented.
IRQDIS	uint32_t	-	0x0	IRQ line disable mask. Bit n of this 32-bit parameter disables IRQ[n] and WICLINES[n+2].
master_id	uint32_t	0x0-0xFFFFFFFF	0x0	Master ID presented in bus transactions.
min_sync_level	uint32_t	0x0-0x3	0x0	Force minimum syncLevel (0 = off = default, 1 = syncState, 2 = postInsnIO, 3 = postInsnAll). Runtime parameter.

Table 3-63 ARMCortexM0PlusCT parameters (continued)

Name	Type	Allowed values	Default value	Description
NUM_IRQ	uint32_t	0x0-0x20	0x20	Number of user interrupts.
NUM_MPU_REGION	uint32_t	0x0-0x8	0x0	Number of MPU regions.
semihosting-cmd_line	string	-	-	Command line available to semihosting SVC calls.
semihosting-cwd	string	-	-	Virtual address of CWD.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x10700000	Virtual address of top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10700000	Virtual address of base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x10800000	Virtual address of stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFFFFFFFF	0xAB	T32 SVC number for semihosting.
SYST	bool	true, false	true	Include SysTick timer functionality.
USER	bool	true, false	false	-
VTOR	bool	true, false	false	-
WIC	bool	true, false	true	Include support for WIC-mode deep sleep.
WPT	uint32_t	0x0-0x2	0x2	The number of watchpoint unit comparators. Runtime parameter.

3.4.5 ARMCortexM0CT component

This section describes the ARMCortexM0CT component.

ARMCortexM0CT - about

This C++ component models r0p0 of a Cortex-M0 core.

This model does not have a parameter that is equivalent to the RAR integration option. The architecturally required register state is reset.

ARM does not guarantee that all ARMv7-M behavior is absent from models of ARMv6-M cores. As a consequence, ARM does not guarantee that code that runs on ARMv7-M cores but fails on ARMv6-M cores fails on ARMv6-M Fast Models cores.

ARMCortexM0CT - ports

This section describes the ports.

Table 3-64 ARMCortexM0CT ports

Name	Protocol	Type	Description
auxfault	Value	Slave	This port is wired to the Auxiliary Fault Status Register.
bigend	Signal	Slave	Configure big endian data format.

Table 3-64 ARMCortexM0CT ports (continued)

Name	Protocol	Type	Description
clk_in	ClockSignal	Slave	The clock signal connected to the clk_in port is used to determine the rate at which the core executes instructions.
currpri	Value	Master	Current execution priority.
edbgrq	Signal	Slave	External debug request.
event	Signal	Peer	This peer port of event input (and output) is for wakeup from WFE and corresponds to the RTL TXEV and RXEV signals.
intisr[32]	Signal	Slave	This signal array delivers signals to the NVIC.
intnmi	Signal	Slave	Configure nonmaskable interrupt.
lockup	Signal	Master	Asserted when the processor is in lockup state.
poreset	Signal	Slave	Raising this signal will do a power-on reset of the core.
stcalib	Value	Slave	The calibration value for the SysTick timer.
stclk	ClockSignal	Slave	The reference clock for the SysTick timer.
pv_ppbus_m	PVBus	Master	The core will generate External Private Peripheral Bus requests on this port.
pvbus_m	PVBus	Master	The core will generate bus requests on this port.
sleepdeep	Signal	Master	Asserted when the processor is in deep sleep.
sleeping	Signal	Master	Asserted when the processor is in sleep.
sysreset	Signal	Slave	Raising this signal will put the core into reset mode (but does not reset the debug logic).
sysresetreq	Signal	Master	Asserted to indicate that a reset is required.
ticks	InstructionCount	Master	Port allowing the number of instructions since startup to be read from the CPU.

ARMCortexM0CT - parameters

This section describes the parameters.

Table 3-65 ARMCortexM0CT parameters

Name	Type	Allowed values	Default value	Description
BIGENDINIT	bool	true, false	false	Initialize processor to big endian mode.
BKPT	uint32_t	0x0-0x4	0x4	The number of breakpoint unit comparators. Runtime parameter.
cpi_div	uint32_t	0x1-0x7FFFFFFF	0x1	Divider for calculating CPI (Cycle Per Instruction). Runtime parameter.
cpi_mul	uint32_t	0x1-0x7FFFFFFF	0x1	Multiplier for calculating CPI (Cycle Per Instruction). Runtime parameter.
DBG	bool	true, false	true	Whether or not the debug extensions are implemented.
master_id	uint32_t	0x0-0xFFFFFFFF	0x0	Master ID presented in bus transactions.
min_sync_level	uint32_t	0x0-0x3	0x0	Force minimum syncLevel (0 = off = default,1 = syncState, 2 = postInsnIO, 3 = postInsnAll). Runtime parameter.

Table 3-65 ARMCortexM0CT parameters (continued)

Name	Type	Allowed values	Default value	Description
NUM_IRQ	uint32_t	0x1-0x20	0x20	Number of user interrupts.
semihosting-cmd_line	string	-	""	Command line available to semihosting SVC calls.
semihosting-cwd	string	-	""	Virtual address of CWD.
semihosting-enable	bool	true, false	true	Enable semihosting SVC traps. Applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	uint32_t	0x0-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	uint32_t	0x0-0xFFFFFFFF	0x10700000	Virtual address of top of heap.
semihosting-stack_base	uint32_t	0x0-0xFFFFFFFF	0x10700000	Virtual address of base of descending stack.
semihosting-stack_limit	uint32_t	0x0-0xFFFFFFFF	0x10800000	Virtual address of stack limit.
semihosting-Thumb_SVC	uint32_t	0x0-0xFFFFFFFF	0xAB	T32 SVC number for semihosting.
SYST	bool	true, false	true	Include SysTick timer functionality.
WIC	bool	true, false	true	Include support for WIC-mode deep sleep.
WPT	uint32_t	0x0-0x2	0x2	The number of watchpoint unit comparators. Runtime parameter.

3.5 Classic processor components

This section describes the classic processor components.

This section contains the following subsections:

3.5.1 ARM1176CT component

This section describes the ARM1176CT component.

ARM1176CT - about

This C++ component is a model of r0p4 of an ARM1176JZF-S™ processor.

ARM1176CT - ports

This section describes the ports.

Table 3-66 ARM1176CT ports

Name	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

Related references

2.5.1 CounterInterface protocol on page 2-64.

2.5.3 InstructionCount protocol on page 2-64.

ARM1176CT - parameters

This section describes the parameters.

Table 3-67 ARM1176CT parameters

Name	Type	Allowed values	Default value	Description
BIGENDINIT	Boolean	true, false	false	Initialize to ARMv5 big endian mode.
CP15SDISABLE	Boolean	true, false	false	Initialize to disable access to some CP15 registers.
INITRAM	Boolean	true, false	false	Initialize with ITCM0 enabled at address 0x0.
UBITINIT	Boolean	true, false	false	Initialize to ARMv6 unaligned behavior.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.
itcm0_size	Integer	0x00-0x40	0x10	Size of ITCM in KB.
dtcm0_size	Integer	0x00-0x40	0x10	Size of DTCM in KB.
device-accurate-tlb	Boolean	true, false	false[bh]	Specify whether all TLBs are modeled.
semihosting-cmd_line[bi]	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	Integer	uint24_t	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	uint8_t	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F0000000	Virtual address of stack limit.
vfp-enable_at_reset[bj]	Boolean	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present	Boolean	true, false	true	Configure processor as VFP enabled.[bk]
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating *Cycles Per Instruction* (CPI).
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating CPI.

ARM1176CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

This PV model does not model Level 1 or Level 2 caches. The system coprocessor registers related to cache operations permit cache aware software to work, but in most cases they only check register access permissions:

- Cache behavior override.
- Cache Dirty Status.
- Invalidate and/or Clean Both Caches.

bh 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.
bi The value of argv[0] points to the first command line argument, not to the name of an image.
bj This is model specific behavior with no hardware equivalent.
bk This parameter lets you disable the VFP features of the model. However the model has not been validated as a true ARM1176JZ-S processor.

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

TLBs

These TLB registers do not have working implementations:

- Normal memory remap register.
- Primary memory remap register.
- TLB Lockdown Attr.
- TLB Lockdown Index.
- TLB Lockdown PA.
- TLB Lockdown VA.

In addition, the simulation does not distinguish peripheral accesses from data accesses, so it ignores configuration of the peripheral port memory remap register.

ARM1176CT - debug features

This component exports a CADI debug interface.

ARM1176CT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

ARM1176CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARM1176CT - debug - memory

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

ARM1176CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARM1176CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

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

3.5.2 ARM1136CT component

This section describes the ARM1136CT component.

ARM1136CT - about

This C++ component is a model of r1p1 of an ARM1136JF-S™ processor.

ARM1136CT - ports

This section describes the ports.

Table 3-68 ARM1136CT ports

Name	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`[bl]	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

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARM1136CT - parameters

This section describes the parameters.

Table 3-69 ARM1136CT parameters

Name	Type	Allowed values	Default value	Description
`BIGENDINIT`	Boolean	`true`, `false`	`false`	Initialize to ARMv5 big endian mode.
`INITRAM`	Boolean	`true`, `false`	`false`	Initialize with ITCM0 enabled at address `0x0`.
`UBITINIT`	Boolean	`true`, `false`	`false`	Initialize to ARMv6 unaligned behavior.
`VINITHI`	Boolean	`true`, `false`	`false`	Initialize with high vectors enabled.
`itcm0_size`	Integer	`0x00-0x40`	`0x10`	Size of ITCM in KB.
`dtcm0_size`	Integer	`0x00-0x40`	`0x10`	Size of DTCM in KB.
`device-accurate-tlb`	Boolean	`true`, `false`	`false`[bm]	Specify whether all TLBs are modeled.
`semihosting-cmd_line`[bn]	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.

[bl] This signal is currently misnamed and is to be named dmairq.
[bm] 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.
[bn] The value of argv[0] points to the first command line argument, not to the name of an image.

Table 3-69 ARM1136CT parameters (continued)

Name	Type	Allowed values	Default value	Description
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	Integer	uint24_t	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	uint8_t	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F0000000	Virtual address of stack limit.
vfp-enable_at_reset[bo]	Boolean	true, false	false	Enable coprocessor access and VFP at reset.
vfp-present	Boolean	true, false	true	Configure processor as VFP enabled.[bp]
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating *Cycles Per Instruction* (CPI).
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating CPI.

ARM1136CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

This PV model does not model Level 1 or Level 2 caches. The system coprocessor registers related to cache operations permit cache aware software to work, but in most cases they only check register access permissions:

- Cache Dirty Status.
- Data Memory Barrier.
- Data Write Barrier.
- Data/Instruction Debug Cache.
- ICache/DCache lockdown.
- ICache/DCache master valid.
- Invalidate and/or Clean Both Caches.
- Invalidate and/or Clean Entire ICache/DCache.
- Invalidate and/or Clean ICache/DCache by Index.
- Invalidate and/or Clean ICache/DCache by MVA.
- Level 1 System Debug registers.
- Prefetch ICache Line.
- Read Block Transfer Status Register.
- Stop Prefetch Range.

MicroTLBs

This component does not implement device accurate MicroTLBs, or system coprocessor registers related to MicroTLB state. The registers affected are:

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

[bo] This is model specific behavior with no hardware equivalent.
[bp] This parameter lets you disable the VFP features of the model. However the model has not been validated as a true ARM1136J-S processor.

TLBs

These TLB registers do not have working implementations:

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

In addition, the simulation does not distinguish peripheral accesses from data accesses, so it ignores configuration of the peripheral port memory remap register.

ARM1136CT - debug features

This component exports a CADI debug interface.

ARM1136CT - debug - registers

All core, VFP, and CP15 registers are visible in the debugger.

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

ARM1136CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARM1136CT - debug - memory

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

ARM1136CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux and other operating systems on an example system.

ARM1136CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

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

3.5.3 ARM968CT component

This section describes the ARM968CT component.

ARM968CT - about

This C++ component is a model of r0p1 of an ARM968E-S™ processor.

ARM968CT - ports

This section describes the ports.

Table 3-70 ARM968CT ports

Name	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

Related references

2.5.1 CounterInterface protocol on page 2-64.
2.5.3 InstructionCount protocol on page 2-64.

ARM968CT - parameters

This section describes the parameters.

Table 3-71 ARM968CT parameters

Name	Type	Allowed values	Default value	Description
BIGENDINIT	Boolean	true, false	false	Initialize to ARMv5 big endian mode.
INITRAM	Boolean	true, false	false	Initialize with ITCM0 enabled at address 0x0.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.
dtcm0_size	Integer	0x0000-0x1000	0x8	Size of DTCM in KB, 0 disables.
itcm0_size	Integer	0x0000-0x1000	0x8	Size of ITCM in KB, 0 disables.
master_id	Integer	0x0000-0xFFFF	0x0	Master ID presented in bus transactions.
semihosting-ARM_SVC	Integer	uint24_t	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	uint8_t	0xAB	T32 SVC number for semihosting.
semihosting-cmd_line[bq]	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.

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

Table 3-71 ARM968CT parameters (continued)

Name	Type	Allowed values	Default value	Description
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F0000000	Virtual address of stack limit.
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating *Cycles Per Instruction* (CPI).
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating CPI.

ARM968CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

ARM968CT - debug features

This component exports a CADI debug interface.

ARM968CT - debug - registers

All core and CP15 registers are visible in the debugger.

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

ARM968CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARM968CT - debug - memory

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

ARM968CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of uClinux and ThreadX OS on an example system.

ARM968CT - performance

This 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.

ARM968CT - DMA

Enable the DMA to or from TCMs 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 technical reference manual.

- 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.

3.5.4 ARM926CT component

This section describes the ARM926CT component.

ARM926CT - about

This C++ component is a model of r0p5 of a ARM926EJ-S™ processor.

ARM926CT - ports

This section describes the ports.

Table 3-72 ARM926CT ports

Name	Protocol	Type	Description
`clk_in`	ClockSignal	Slave	Clock input
`pvbus_m`	PVBus	Master	Master port for all memory accesses

Table 3-72 ARM926CT ports (continued)

Name	Protocol	Type	Description
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

Related references

2.5.1 CounterInterface protocol on page 2-64.

2.5.3 InstructionCount protocol on page 2-64.

ARM926CT - parameters

This section describes the parameters.

Table 3-73 ARM926CT parameters

Name	Type	Allowed values	Default value	Description
BIGENDINIT	Boolean	true, false	false	Initialize to ARMv5 big endian mode.
INITRAM	Boolean	true, false	false	Initialize with ITCM0 enabled at address 0x0.
VINITHI	Boolean	true, false	false	Initialize with high vectors enabled.
itcm0_size	Integer	0x000-0x400	0x8	Size of ITCM in KB.
dtcm0_size	Integer	0x000-0x400	0x8	Size of DTCM in KB.
device-accurate-tlb	Boolean	true, false	false[br]	Specify whether all TLBs are modeled.
semihosting-cmd_line[bs]	String	No limit except memory	[Empty string]	Command line available to semihosting SVC calls.
semihosting-enable	Boolean	true, false	true	Enable semihosting SVC traps. Caution: applications that do not use semihosting must set this parameter to false.
semihosting-ARM_SVC	Integer	uint24_t	0x123456	A32 SVC number for semihosting.
semihosting-Thumb_SVC	Integer	uint8_t	0xAB	T32 SVC number for semihosting.
semihosting-heap_base	Integer	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	Integer	0x00000000-0xFFFFFFFF	0x0F000000	Virtual address of top of heap.
semihosting-stack_base	Integer	0x00000000-0xFFFFFFFF	0x10000000	Virtual address of base of descending stack.
semihosting-stack_limit	Integer	0x00000000-0xFFFFFFFF	0x0F0000000	Virtual address of stack limit.
cpi_mul	Integer	1-0x7FFFFFFF	1	Multiplier for calculating *Cycles Per Instruction* (CPI).
cpi_div	Integer	1-0x7FFFFFFF	1	Divider for calculating CPI.

[br] 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.

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

Table 3-73 ARM926CT parameters (continued)

Name	Type	Allowed values	Default value	Description
dcache-state_modelled	Boolean	true, false	false	Set whether D-cache has stateful implementation.
icache-state_modelled	Boolean	true, false	false	Set whether I-cache has stateful implementation.
dcache-size	Integer	0x1000-0x20000	0x20000	Set D-cache size in bytes.
icache-size	Integer	0x1000-0x20000	0x20000	Set I-cache size in bytes.

ARM926CT - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies except for the coprocessor 14 registers and the integration and test registers.

ARM926CT - debug features

This component exports a CADI debug interface.

ARM926CT - debug - registers

All core and CP15 registers are visible in the debugger.

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

ARM926CT - debug - breakpoints

This component directly supports single address unconditional instruction breakpoints, unconditional instruction address range breakpoints, and single address unconditional data breakpoints.

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

The model does not support CADI exception breakpoints. Instead, it implements exception breakpoints as register breakpoints on pseudoregisters, named after the exceptions, in the Vectors register group.

ARM926CT - debug - memory

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

ARM926CT - verification and testing

This component passes tests by using the architecture validation suite tests and booting of Linux on an example system.

ARM926CT - differences between the CT model and RTL implementations

This component differs from the corresponding revision of the RTL implementation.

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

Chapter 4
Peripheral and Interface Components

This chapter describes the generic *Programmer's View* (PV) peripheral components in Fast Models.

It contains the following sections:

4.1 Peripheral and interface components - about

The *Programmer's View* (PV) models show you what the hardware would show you. However, they sacrifice accuracy in timing to achieve fast simulation speeds.

You can test software functionality with PV models, but do not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior. For example, the AMBAPV2PVBus component rejects transactions with `byte_enable` set because PVBus does not support AXI write strobe.

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. PrimeCell based components 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 that model specific hardware functionality are 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.

4.2 AMBA-PV components

This section describes the AMBA components.

This section contains the following subsections:

4.2.1 AMBA-PV components - about

The AMBA-PV components and protocols permit you to model a platform that interfaces with an ARM AMBA-based system.

The system is modeled using *Accellera Systems Initiative Transaction Level Modeling* (ASI TLM) at *Programmer's View* (PV) level using the SystemC Export functionality of System Canvas.

These components allow conversion between protocols: PVBus and AMBAPV, Signal and AMBAPVSignal, StateSignal and AMBAPVSignalState, Value(_64) and AMBAPVValue(64), and ValueState(_64) and AMBAPVValueState(64). 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`.

The protocols and components are designed to interface with the AMBA TLM PV library for ASI TLM 2.0. Fast Models provides this library as a standard way of mapping the AMBA protocol on top of ASI TLM 2.0.2 kit at PV level.

For more information about the AMBA TLM PV library for ASI TLM 2.0.2 kit, see the Fast Models documentation in `%MAXCORE_HOME%\AMBA-PV\doc`. On Linux, use the `$MAXCORE_HOME` environment variable instead.

For more information about ASI TLM 2.0, see the Accellera documentation that is provided with the kit.

Related concepts

2.1.1 AMBA-PV protocols - about on page 2-43.

Related references

4.5 PVBus components on page 4-400.

Related information

AMBA-PV Extensions to TLM 2.0 Developer Guide.

Fast Models User Guide, SystemC Export with Multiple Instantiation.

Accellera Systems Initiative.

4.2.2 PVBus2AMBAPV component

This section describes the PVBus2AMBAPV component.

PVBus2AMBAPV - about

This component converts from PVBus to AMBAPV protocols.

This is a LISA+ component.

PVBus2AMBAPV - ports

This section describes the ports.

Table 4-1 PVBus2AMBAPV ports

Name	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.

PVBus2AMBAPV - parameters

This section describes the parameters.

Table 4-2 PVBus2AMBAPV parameters

Name	Type	Allowed values	Default value	Description
`force-dmi-size`	`bool`	`true`, `false`	`true`	Align DMI start and end addresses to 4kB. If `true`, DMI memory is faster, but DMI regions that are less than 4kB in size or not 4kB aligned are inaccessible.
`global-monitor`	`bool`	`true`, `false`	`false`	Enable built-in global monitor. This may be required if cache-state modeling is disabled in up-stream cores and the down-stream platform contains no global monitor.
`size`	`uint64_t`	`0` to 2^{64} `- 1`, where `0` represents 2^{64} bytes, but must also be a multiple of `0x1000` (4KB)	`0x1000000000000`	Addressable size of the device in bytes.

PVBus2AMBAPV - verification and testing

This component passes tests 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.

4.2.3 AMBAPV2PVBus component

This section describes the AMBAPV2PVBus component.

AMBAPV2PVBus - about

This LISA+ component converts from AMBAPV to PVBus protocols.

PVBus does not support transactions with `byte_enable` set (strobing transactions, in AXI terms). This bridge component rejects them.

AMBAPV2PVBus - ports

This section describes the ports.

Table 4-3 AMBAPV2PVBus ports

Name	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.

AMBAPV2PVBus - parameters

This section describes the parameters.

Table 4-4 AMBAPV2PVBus parameters

Name	Type	Allowed values	Default value	Description
base_addr	uint64_t	-	0	Base address of the component. Defines an offset to be added to the address of outgoing transactions.

AMBAPV2PVBus - verification and testing

This component passes tests 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.

4.2.4 PVBus2AMBAPVACE component

This section describes the PVBus2AMBAPVACE component.

PVBus2AMBAPVACE - about

This component converts from PVBus to AMBAPVACE protocols.

This is a LISA+ component.

PVBus2AMBAPVACE - ports

This section describes the ports.

Table 4-5 PVBus2AMBAPVACE ports

Name	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.

PVBus2AMBAPVACE - parameters

This section describes the parameters.

Table 4-6 PVBus2AMBAPVACE parameters

Name	Type	Allowed values	Default value	Description
`force-dmi-size`	`bool`	`true`, `false`	`true`	Align DMI start and end addresses to 4kB. If `true`, DMI memory is faster, but DMI regions that are less than 4kB in size or not 4kB aligned are inaccessible.
`global-monitor`	`bool`	`true`, `false`	`false`	Enable built-in global monitor. This may be required if cache-state modeling is disabled in up-stream cores and the down-stream platform contains no global monitor.
`size`	`uint64_t`	`0` to 2^{64} `- 1`, where `0` represents 2^{64} bytes, but must also be a multiple of `0x1000` (4KB)	`0x1000000000000`	Addressable size of the device in bytes.

PVBus2AMBAPVACE - debug features

This component supports debug bus transactions but has no specific debug features.

PVBus2AMBAPVACE - verification and testing

This component passes tests using system level tests that included booting Linux on an ARM big.LITTLE™ VE platform.

PVBus2AMBAPVACE - 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.

PVBus2AMBAPVACE - library dependencies

This component depends on the AMBA-PV API, which must be at least version 1.4.

4.2.5 AMBAPVACE2PVBus component

This section describes the AMBAPVACE2PVBus component.

AMBAPVACE2PVBus - about

This component converts from AMBAPVACE to PVBus protocols.

This is a LISA+ component.

AMBAPVACE2PVBus - ports

This section describes the ports.

Table 4-7 AMBAPVACE2PVBus ports

Name	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.

AMBAPVACE2PVBus - debug features

This component supports debug bus transactions but has no specific debug features.

AMBAPVACE2PVBus - verification and testing

This component passes system level tests that included booting Linux on an ARM big.LITTLE VE platform.

AMBAPVACE2PVBus - 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.

AMBAPVACE2PVBus - library dependencies

This component depends on the AMBA-PV API, which must be at least version 1.4.

4.2.6 SGSignal2AMBAPVSignal component

This section describes the SGSignal2AMBAPVSignal component.

SGSignal2AMBAPVSignal - about

This component converts from Signal to AMBAPVSignal protocols.

This is a LISA+ component.

SGSignal2AMBAPVSignal - ports

This section describes the ports.

Table 4-8 SGSignal2AMBAPVSignal ports

Name	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.

SGSignal2AMBAPVSignal - verification and testing

This component passes tests 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.

4.2.7 AMBAPVSignal2SGSignal component

This section describes the AMBAPVSignal2SGSignal component.

AMBAPVSignal2SGSignal - about

This component converts from AMBAPVSignal to Signal protocols.

This is a LISA+ component.

AMBAPVSignal2SGSignal - ports

This section describes the ports.

Table 4-9 AMBAPVSignal2SGSignal ports

Name	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.

AMBAPVSignal2SGSignal - verification and testing

This component passes tests 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.

4.2.8 SGStateSignal2AMBAPVSignalState component

This section describes the SGStateSignal2AMBAPVSignalState component.

SGStateSignal2AMBAPVSignalState - about

This component converts from StateSignal to AMBAPVSignalState protocols.

This is a LISA+ component.

SGStateSignal2AMBAPVSignalState - ports

This section describes the ports.

Table 4-10 SGStateSignal2AMBAPVSignalState ports

Name	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.

SGStateSignal2AMBAPVSignalState - verification and testing

This component passes tests 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.

4.2.9 AMBAPVSignalState2SGStateSignal component

This section describes the AMBAPVSignalState2SGStateSignal component.

AMBAPVSignalState2SGStateSignal - about

This component converts from AMBAPVSignalState to StateSignal protocols.

This is a LISA+ component.

AMBAPVSignalState2SGStateSignal - ports

This section describes the ports.

Table 4-11 AMBAPVSignalState2SGStateSignal ports

Name	Protocol	Type	Description
`amba_pv_signal_s`	AMBAPVSignalState	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.

AMBAPVSignalState2SGStateSignal - verification and testing

This component passes tests 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.

4.2.10 SGValue2AMBAPVValue component

This section describes the SGValue2AMBAPVValue component.

SGValue2AMBAPVValue - about

This component converts from Value to AMBAPVValue protocols, using 32-bit integer values between components.

This is a LISA+ component.

SGValue2AMBAPVValue - ports

This section describes the ports.

Table 4-12 SGValue2AMBAPVValue ports

Name	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.

SGValue2AMBAPVValue - verification and testing

This component passes tests 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.

4.2.11 SGValue2AMBAPVValue64 component

This section describes the SGValue2AMBAPVValue64 component.

SGValue2AMBAPVValue64 - about

The SGValue2AMBAPVValue64 component converts from Value_64 to AMBAPVValue protocols. 64-bit integer values are used between components.

This is a LISA+ component.

SGValue2AMBAPVValue64 - ports

This section describes the ports.

Table 4-13 SGValue2AMBAPVValue64 ports

Name	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.

SGValue2AMBAPVValue64 - verification and testing

This component passes tests 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.

4.2.12 AMBAPVValue2SGValue component

This section describes the AMBAPVValue2SGValue component.

AMBAPVValue2SGValue - about

This component converts from AMBAPVValue to Value protocols, using 32-bit integer values between components.

This is a LISA+ component.

AMBAPVValue2SGValue - ports

This section describes the ports.

Table 4-14 AMBAPVValue2SGValue ports

Name	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.

AMBAPVValue2SGValue - verification and testing

This component passes tests 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.

4.2.13 AMBAPVValue2SGValue64 component

This section describes the AMBAPVValue2SGValue64 component.

AMBAPVValue2SGValue64 - about

This component converts from AMBAPVValue to Value_64 protocols. 64-bit integer values are used between components.

This is a LISA+ component.

AMBAPVValue2SGValue64 - ports

This section describes the ports.

Table 4-15 AMBAPVValue2SGValue64 ports

Name	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.

AMBAPVValue2SGValue64 - verification and testing

This component passes tests 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.

4.2.14 SGValueState2AMBAPVValueState component

This section describes the SGValueState2AMBAPVValueState component.

SGValueState2AMBAPVValueState - about

This component converts from ValueState to AMBAPVValueState protocols. 32-bit integer values are used between components.

This is a LISA+ component.

SGValueState2AMBAPVValueState - ports

This section describes the ports.

Table 4-16 SGValueState2AMBAPVValueState ports

Name	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.

SGValueState2AMBAPVValueState - verification and testing

This component passes tests 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.

4.2.15 SGValueState2AMBAPVValueState64 component

This section describes the SGValueState2AMBAPVValueState64 component.

SGValueState2AMBAPVValueState64 - about

This component converts from ValueState_64 to AMBAPVValueState protocols, using 64-bit integer values between components.

This is a LISA+ component.

SGValueState2AMBAPVValueState64 - ports

This section describes the ports.

Table 4-17 SGValueState2AMBAPVValueState64 ports

Name	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.

SGValueState2AMBAPVValueState64 - verification and testing

This component passes tests 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.

4.2.16 AMBAPVValueState2SGValueState component

This section describes the AMBAPVValueState2SGValueState component.

AMBAPVValueState2SGValueState - about

This component converts from AMBAPVValueState to ValueState protocols, using 32-bit integer values between components.

This is a LISA+ component.

AMBAPVValueState2SGValueState - ports

This section describes the ports.

Table 4-18 AMBAPVValueState2SGValueState ports

Name	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.

AMBAPVValueState2SGValueState - verification and testing

This component passes tests 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.

4.2.17 AMBAPVValueState2SGValueState64 component

This component describes the AMBAPVValueState2SGValueState64 component.

AMBAPVValueState2SGValueState64 - about

This component converts from AMBAPVValueState64 to ValueState_64 protocols, using 64-bit integer values between components.

This is a LISA+ component.

AMBAPVValueState2SGValueState64 - ports

This section describes the ports.

Table 4-19 AMBAPVValueState2SGValueState64 ports

Name	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.

AMBAPVValueState2SGValueState64 - verification and testing

This component passes tests 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.

4.3 Clocking components

This section describes the clocking components.

This section contains the following subsections:

4.3.1 Clocking components - about

The clocking components and protocols provide a mechanism for systems to regulate the execution rate of components.

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. ClockTimers can generate callbacks after a given number of ticks of that clock. ClockTimers 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.

Related concepts

2.2.1 Clocking protocols - about on page 2-49.

Related references

4.3.2 ClockDivider component on page 4-282.

4.3.3 ClockTimer component on page 4-283.

4.3.4 ClockTimer64 component on page 4-284.

4.3.5 MasterClock component on page 4-284.

4.3.2 ClockDivider component

This section describes the ClockDivider component.

ClockDivider component - about

This 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.

This is a C++ component.

ClockDivider - ports

This section describes the ports.

Table 4-20 ClockDivider ports

Name	Protocol	Type	Description
clk_in	ClockSignal	Slave	Source clock rate
clk_out	ClockSignal	Master	Output clock rate
rate	ClockRateControl	Slave	Permits you to dynamically change the clock divider ratio.

Related references

2.2.2 ClockRateControl protocol on page 2-49.

ClockDivider - parameters

This section describes the parameters.

Note

You can set these parameters from LISA but not SystemC.

Table 4-21 ClockDivider parameters

Name	Type	Allowed values	Default value	Description
mul	uint_32	-	1	Clock rate multiplier
div	uint_32	-	1	Clock rate divider

ClockDivider - verification and testing

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

ClockDivider - performance

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

4.3.3 ClockTimer component

This section describes the ClockTimer component.

ClockTimer component - about

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

This is a C++ component.

ClockTimer - ports

This section describes the ports.

Table 4-22 ClockTimer ports

Name	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

Related references

2.2.5 TimerControl protocol on page 2-50.

2.2.3 TimerCallback protocol on page 2-49.

ClockTimer - verification and testing

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

ClockTimer - 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.

4.3.4 ClockTimer64 component

This section describes the ClockTimer64 component.

ClockTimer64 component - about

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

This is a C++ component.

ClockTimer64 - ports

This section describes the ports.

Table 4-23 ClockTimer ports

Name	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

Related references

2.2.6 TimerControl64 protocol on page 2-50.

2.2.4 TimerCallback64 protocol on page 2-49.

ClockTimer64 - verification and testing

This component passes internal unit tests.

ClockTimer64 - 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.

4.3.5 MasterClock component

This section describes the MasterClock component.

MasterClock component - about

This 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.

This is a C++ component.

MasterClock - ports

This section describes the ports.

Table 4-24 MasterClock ports

Name	Protocol	Type	Description
`clk_out`	ClockSignal	Master	Master clock rate

For more information, see the hardware documentation.

MasterClock - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4 Peripheral components

This section describes the peripheral components. They are examples of different types of component, and permit the development of example platforms. They might not model all aspects of a component, and they might be only partially validated.

Note

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

This section contains the following subsections:

4.4.1 AndGate component

This section describes the AndGate component.

AndGate - about

This component implements a logical AND of two Signal input ports to generate a single output Signal. For example, you can use this component to combine two interrupt signals.

This is a LISA+ component.

AndGate - ports

This section describes the ports.

Table 4-25 AndGate ports

Name	Protocol	Type	Description
`input[0]`	Signal	Slave	First input signal
`input[1]`	Signal	Slave	Second input signal
`output`	Signal	Master	Combined output signal

AndGate - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.2 AudioOut_File component

This section describes the AudioOut_File component.

AudioOut_File - about

This 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.

This is a LISA+ component.

AudioOut_File - ports

This section describes the ports.

Table 4-26 AudioOut_File ports

Name	Protocol	Type	Description
`audio`	AudioControl	Slave	Audio input for a connection to a component such as the PL041_AACI

Related references

2.4.1 AudioControl protocol on page 2-55.

AudioOut_File - parameters

This section describes the parameters.

Table 4-27 AudioOut_File parameters

Name	Type	Allowed values	Default value	Description
`fname`	String	Legal filename	[Empty string]	Filename for output

AudioOut_File - verification and testing

This component passes tests as part of a system containing a PL041.

AudioOut_File - performance

ARM expects this component to have little effect on the performance of PV systems. AudioOut_File drains audio data at the rate that would be expected by software running in the simulation.

4.4.3 AudioOut_SDL component

This section describes the AudioOut_SDL component.

AudioOut_SDL - about

This component outputs audio using the host features of the Simple DirectMedia Layer library.

This is a LISA+ component that relies on an external C++ class.

AudioOut_SDL - ports

This section describes the ports.

Table 4-28 AudioOut_SDL ports

Name	Protocol	Type	Description
`audio`	AudioControl	Slave	Audio input for a connection to a component such as the PL041_AACI

Related references

2.4.1 AudioControl protocol on page 2-55.

AudioOut_SDL - verification and testing

This component passes tests as part of a system containing a PL041.

AudioOut_SDL - performance

This 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.

AudioOut_SDL - library dependencies

This component depends on the Simple DirectMedia Layer library C++ class.

4.4.4 BP135_AXI2APB component

This section describes the BP135_AXI2APB component.

BP135_AXI2APB - about

This LISA+ component is a model of r0p0 of the PrimeCell Infrastructure AMBA 3 AXI to AMBA 3 APB Bridge (BP135).

The component, when configured by another component such as the BP141_TZPC, permits control of secure access to up to 16 PrimeCell peripherals.

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.

BP135_AXI2APB - ports

This section describes the ports.

Table 4-29 BP135_AXI2APB ports

Name	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

BP135_AXI2APB - verification and testing

This component passes tests as part of an integrated platform.

BP135_AXI2APB - library dependencies

This component depends on the BP135TZSwitchControl component, which decodes the protection selection bits provided by the TZPROT ports and controls the protection routing of the embedded TZSwitch routing components.

Related references

4.5.10 TZSwitch component on page 4-406.

4.4.5 BP141_TZMA component

This section describes the BP141_TZMA component.

BP141_TZMA - about

This LISA+ component is a model of the ARM PrimeCell Infrastructure AMBA 3 AXI TrustZone Memory Adapter (BP141).

It 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.

This component routes transactions according to the:

- Memory region that they are attempting to access.
- 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.

`TZMEMSIZE` is the maximum addressing range of the memory as defined by that parameter. By default, `TZMEMSIZE` is set to 2MB. AxADDR is the offset address that the transactions want to access.

Table 4-30 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

BP141_TZMA - ports

This section describes the ports.

Table 4-31 BP141_TZMA ports

Name	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

BP141_TZMA - parameters

This section describes the parameters.

Table 4-32 BP141_TZMA parameters

Name	Type	Allowed values	Default value	Description
`TZMEMSIZE`	`uint32_t`	-	`0x200000`	Sets the maximum size of the addressable memory. ARM deprecates this parameter.
`TZSEGSIZE`	`uint32_t`	Multiples of 4096	4096	Configures the size of the region allocated for secure access for each increment of the R0Size value.
`TZSECROMSIZE`	`uint32_t`	-	`0x200`	Configures the initial region configuration value so that an external secure control is not required.

BP141_TZMA - verification and testing

This component passes tests separately by using its own test suite and as part of the VE example system by using VE test suites and by booting operating systems.

4.4.6 BP147_TZPC component

This section describes the BP147_TZPC component.

BP147_TZPC - about

This LISA+ component is a model of r0p0 of the ARM AMBA 3 TrustZone Protection Controller (BP147).

It provides a software interface to the protection bits in a secure system in a TrustZone design.

BP147_TZPC - ports

This section describes the ports.

Table 4-33 BP147_TZPC ports

Name	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

BP147_TZPC - registers

This section describes the registers.

Table 4-34 BP147_TZPC registers

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

BP147_TZPC - verification and testing

This component passes tests using a unit test suite.

4.4.7 CCI400 component

This section describes the CCI400 component.

CCI400 - about

This C++ component is a model of r1p3 of the *Cache Coherent Interconnect* (CCI) for AXI4.

ACE limitation

AXI Coherency Extensions (ACE) are extensions to AXI4 that support system-level cache-coherency between multiple clusters. The ACE cache models in the Cortex-A15 and the Cortex-A7, and the ACE support in the CCI-400 have a limitation: these functional models process only one transaction at a time. Normally, the simulation processes each transaction to completion before allowing any master to generate another transaction. However, there is a situation in which the simulation might fail. Suppose a SystemC bus slave calls `wait()` while it is processing a transaction. This call might allow another master to issue another transaction that passes through the CCI-400 or the Cortex-A15/Cortex-A7 caches. This situation could happen if a SystemC bus master running in another thread is connected to one of the ACE-lite ports on the CCI-400.

CCI400 - ports

This section describes the ports.

Table 4-35 CCI400 ports

Name	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 `pvbus_s_ace_3` and `pvbus_s_ace_4` ports.
`pvbus_m`	PVBus	Master	Master port for all downstream memory accesses.
`pvbus_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.

CCI400 - parameters

This section describes the parameters.

Table 4-36 CCI400 parameters

Name	Type	Allowed values	Default value	Description
`acchannelen`	`int`	`0x0-0x31`	`0x31`	For each upstream port, determine if it is enabled or not with respect to snoop requests.
`barrierterminate`	`int`	`0x0-0x7`	`0x7`	For each downstream port, determine if barriers are terminated at that port.

Table 4-36 CCI400 parameters (continued)

Name	Type	Allowed values	Default value	Description
broadcastcachemain	int	0x0-0x7	0x0	For each downstream port, a bit determines if broadcast cache maintenance operations are forwarded down that port.
bufferableoverride	int	0x0-0x7	0x0	For each downstream port, determine if all transactions are forced to non-bufferable.
cache_state_modelled	bool	true, false	true	Model the cache coherency operations. Enable to correctly maintain coherency between ACE masters that model cache state.
force_on_from_start	bool	true, false	false	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.
log_enabled	int	0x0, 0x1, 0x2, 0x3	0x1	Enable log messages from the CCI register file: **0** Print nothing. **1** Print access violations. **2** Also print writes. **3** Also print reads.
periphbase	int	-	0x2C000000	Value for PERIPHBASE, using only bits [39:16]. You can override it with an input on the periphbase port.
revision	string	‘r0p0’	‘r0p0’	The revision of the component, reflected in the ID register value.

CCI400 - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies.

CCI400 - debug features

This component exports a CADI debug interface.

CCI400 - verification and testing

This component passes tests by running a switching hypervisor on an example system containing an ARMCortexA7x*n*CT component and an ARMCortexA15x*n*CT CCI400 component.

CCI400 - performance

If you disable `cache_state_modelled`, this component has negligible performance impact. If you enable `cache_state_modelled`, it adds significant cost to throughput for coherent transactions.

4.4.8 CCI500 component

This section describes the CCI500 component.

CCI500 - about

This C++ component is a model of r0p0 of the CCI-500 Cache Coherent Interconnect for AXI4.

AXI Coherency Extensions (ACE) are extensions to AXI4. They support system-level cache-coherency between multiple clusters.

CCI500 - functionality

Some features differ in the model.

Address Decoder

- Only supports striping down to 4KiB.
- If the address decoder aborts the access, returns `SLVERR` rather than `DECERR`.

Performance Monitoring Unit

- PMU counters recognize only a few event sources:
  - Slave interface events:
    - **3** ReadOnce.
    - **4** ReadClean, ReadShared, ReadNotSharedDirty, ReadUnique.
    - **5** MakeUnique, CleanUnique.
    - **6** CleanInvalid, CleanShared, MakeInvalid.
    - **7** DVM transaction received from upstream.
    - **9** Read data that is satisfied by a snoop request.
  - No events are implemented for the global events or for the master events.
- The PMU does not implement the event bus (`EVNTBUS`).

Register access

The register file only supports 32-bit accesses to its registers. Later versions of the CCI500 hardware support full write strobes to the register file. This limitation means that byte and halfword accesses work on the hardware but not on this version of the component.

CCI500 - ports

This section describes the ports.

`acchannelens`*x* represents the ports `ACCHANNELENS0`-`ACCHANNELENS6` on the RTL (*Register Transfer Level*) (assuming that there are seven upstream ports).

- Each upstream ACE port *y* (`pvbus_s[`*y*`]`) has a two bit `ACCHANNELENS`*x*.

  Bit 0 == 0
  disables DVM messages from being sent to this interface.

  Bit 1 == 0
  disables snoop messages from being sent to this interface.
- Each upstream ACE-Lite port *z* (`pvbus_s[`*z*`]`) has a one bit `ACCHANNELENS`*x*.

  Bit 0 == 0
  disables DVM messages from being sent to this interface.

The model supports various configurations with one LISA file. Each channel enable behaves as though it is one bit or two bit, as appropriate. If you send an invalid value, because of the type of the port, then the CCI model halts, producing a fatal error. Parameters set the initial values until you drive them, so if they are constant then you need not drive them.

In the RTL, the CCI500 samples the signals at reset. In the model, the CCI500 samples the signals at the first transaction. Thus any controller that produces these signals has to hold them constant for long enough.

Table 4-37 CCI500 ports

Name	Protocol	Type	Description
acchannelens*x*[7]	Value	Slave	-
evntcntoverflow[8]	Signal	Master	-
reset_state_of_upstream_port[7]	Signal	Slave	-
pvbus_register_file_s	PVBus	Slave	-
pvbus_s[7]	PVBus	Slave	-
pvbus_m[6]	PVBus	Master	-
errirq	Signal	Master	-
dbgen	Signal	Slave	-
spiden	Signal	Slave	-
spniden	Signal	Slave	-
niden	Signal	Slave	-
address_decoder	CCI500_AddressDecoderProtocol	Master	-

CCI500 - parameters

This section describes the parameters.

The LISA file declares seven upstream ports. You can configure these ports with num_ace_ports and num_ace_lite_ports. The bottom num_ace_lite_ports are ACE-Lite+DVM. The next num_ace_ports are ACE. Any remaining ports are ignored; if transactions are made on them, then warnings are produced. For example, if num_ace_ports = 1 and num_ace_lite_ports = 1 then pvbus_s[1] is ACE, pvbus_s[0] is ACE-Lite+DVM and pvbus_s[6-2] are considered not to exist.

Table 4-38 CCI500 parameters

Name	Type	Allowed values	Default value	Description
cache_state_modelled	bool	true, false	true	Model the cache state.
num_ace_ports	unsigned	1-4	2	The top num_ace_ports are ACE and support full coherency. The total number of ports must not exceed seven.
num_ace_lite_ports	unsigned	1-6	5	The number of ACE-Lite+DVM ports. These ports are the lowest numbered ports. The total number of ports must not exceed seven.

Table 4-38 CCI500 parameters (continued)

Name	Type	Allowed values	Default value	Description
acchannelens*n*	uint64_t	0x0-0x3	0x3	For upstream port pvbus_s[*n*], where 0 <= *n* <= 6. Bit[0] == 0 disables DVM messages from being sent. If this port is an ACE port, then bit[1] == 0 disables snoop messages from being sent. The signal acchannelens*x*[*n*] can override this parameter. For an ACE-Lite port, bit[1] from the parameter is ignored, allowing the default value of 0x3 to create a functional system without excessive configuration.
dbgen	bool	true, false	true	Invasive debug enable. If true, enables the counting of PMU events.
spiden	bool	true, false	true	Secure invasive debug enable. If both SPIDEN and DBGEN are high, enables the counting of both Non-secure and Secure events.
spniden	bool	true, false	true	Whether Secure and Non-secure events are allowed to be counted in the performance monitor.
niden	bool	true, false	true	Whether Non-secure events are allowed to be counted in the performance monitor.
force_on_from_start	bool	true, false	false	The interconnect normally starts up with snooping disabled. This parameter allows the model to start up as enabled without programming it. This enabling is only set up at simulation reset and not at signal reset. If the upstreams can ever be held in reset, then connect the reset_state_of_upstream_port ports so that the CCI knows when to disable snoops to the upstream systems. Otherwise, the upstream system complains that it “received a snoop request while it was in reset”.

Table 4-38 CCI500 parameters (continued)

Name	Type	Allowed values	Default value	Description
reentrancy_support	string	-	"env"	Must be one of: on, off, cacheglobal, env. on: hazard checking per cache line (normal mode). off: no hazard checking (use only for single master systems). cacheglobal: hazard checking globally for cache (not per cache line, testing feature, provokes more hazards than necessary). env (or empty string): take value from FM_REENTRANCY_SUPPORT env var; if this env var is not set, use on.
number_of_phantom_entries	uint64_t	0x0000000000000001-0x00000000ffffffff	0x0000000000000020	Number of phantom entries in the cache. Certain cache operations use phantom entries to hold temporary data. The default value is safe for all systems containing up to 32 masters.
version	string	-	""	The version of the interconnect. Allowed versions: r0p0.

4.4.9 CCN502 component

This section describes the model of r0p0 of the CCN502 interconnect component.

CCN502 - ports

This section describes the ports.

CCN502 has three or five downstream ports (depending on the number of crosspoints): two or four SNF ports for the memory controller, and one Acelite port (HNI).

Table 4-39 CCN502 ports

Name	Protocol	Type	Description
pvbus_s_rnf[4]	PVBus	Slave	RNF upstream ports.
pvbus_s_rni[9]	PVBus	Slave	RNI upstream ports.
pvbus_m_hni[1]	PVBus	Master	HNI downstream port.
pvbus_m_snf[4]	PVBus	Master	SNF donwstream ports.
reset_in	Signal	Slave	Reset signal.

CCN502 - parameters

This section describes the parameters.

Table 4-40 CCN502 parameters

Name	Type	Allowed values	Default value	Description
acchannelen_rnf	uint32_t	0-15	15	Bitmap for each RNF upstream port to test if snoop requests are enabled.
acchannelen_rni	uint32_t	0x0-0x1ff	0x1ff	Bitmap for each RNI upstream port to test if DVM requests are enabled.
cache_size_in_kbytes	uint32_t	0-8192	4096	Size of the L3 cache to model.
cache_state_modelled	bool	true, false	true	Model the cache state.
force_on_from_start	bool	true, false	false	The component normally starts up with snooping disabled. Set this parameter to true to allow the model to start up as enabled without having to program it. This parameter affects simulation reset and not signal reset.
number_of_snf	uint32_t	2-4	2	Number of SNF nodes present.
periphbase	uint64_t	-	0x2C000000	Value for PERIPHBASE. Only bits [43:24] are used.
sbsx_bridge_present	bool	true, false	true	Value for SBSX bridge presence.
systemaddrmap	uint64_t	0x0-0xfffffffff	0x0	Bitmap for 20 regions in the CCN interconnect. Every two bits describe the region type for the corresponding region.
variant_name	string	0-0	0	Can be CCN502_6XP or CCN502_8XP.

4.4.10 CCN504 component

This section describes the model of r0p0 of the CCN504 interconnect component.

CCN504 - ports

This section describes the ports.

CCN504 has three downstream ports: two SNF ports for the memory controller, and one Acelite port (HNI).

Table 4-41 CCN504 ports

Name	Protocol	Type	Description
pvbus_s_rnf[4]	PVBus	Slave	RNF upstream ports.
pvbus_s_rni[18]	PVBus	Slave	RNI upstream ports.
pvbus_m_hni[1]	PVBus	Master	HNI downstream port.
pvbus_m_snf[2]	PVBus	Master	SNF donwstream ports.
reset_in	Signal	Slave	Reset signal.

CCN504 - parameters

This section describes the parameters.

Table 4-42 CCN504 parameters

Name	Type	Allowed values	Default value	Description
acchannelen_rnf	uint32_t	0-15	15	Bitmap for each RNF upstream port to test if snoop requests are enabled.
acchannelen_rni	uint32_t	0x0-0x3ffff	0x3ffff	Bitmap for each RNI upstream port to test if DVM requests are enabled.
cache_size_in_mbytes	uint32_t	-	8	Size of the L3 cache to model.
cache_state_modelled	bool	true, false	true	Model the cache state.
force_on_from_start	bool	true, false	false	The component normally starts up with snooping disabled. Set this parameter to true to allow the model to start up as enabled without having to program it. This parameter affects simulation reset and not signal reset.
number_of_snf	uint32_t	1-2	2	Number of SNF nodes present.
periphbase	uint64_t	-	0x2C000000	Value for PERIPHBASE. Only bits [43:24] are used.
sbas_bridge_present	bool	true, false	true	Value for SBAS bridge presence.
sbsx_bridge_present	bool	true, false	true	Value for SBSX bridge presence.
systemaddrmap	uint64_t	0x0-0xfffffffff	0x0	Bitmap for 20 regions in the CCN interconnect. Every two bits describe the region type for the corresponding region.

4.4.11 CCN508 component

This section describes the model of r0p0 of the CCN508 interconnect component.

CCN508 - ports

This section describes the ports.

CCN508 has six downstream ports: four SNF ports for the memory controller, and two Acelite ports (HNI).

Table 4-43 CCN508 ports

Name	Protocol	Type	Description
pvbus_s_rnf[8]	PVBus	Slave	RNF upstream ports.
pvbus_s_rni[24]	PVBus	Slave	RNI upstream ports.
pvbus_m_hni[2]	PVBus	Master	HNI downstream port.
pvbus_m_snf[4]	PVBus	Master	SNF donwstream ports.
reset_in	Signal	Slave	Reset signal.

CCN508 - parameters

This section describes the parameters.

Table 4-44 CCN504 parameters

Name	Type	Allowed values	Default value	Description
acchannelen_rnf	uint32_t	0-255	255	Bitmap for each RNF upstream port to test if snoop requests are enabled.
acchannelen_rni	uint32_t	0x0-0xffffff	0xffffff	Bitmap for each RNI upstream port to test if DVM requests are enabled.
cache_size_in_mbytes	uint32_t	-	8	Size of the L3 cache to model.
cache_state_modelled	bool	true, false	true	Model the cache state.
force_on_from_start	bool	true, false	false	The component normally starts up with snooping disabled. Set this parameter to true to allow the model to start up as enabled without having to program it. This parameter affects simulation reset and not signal reset.
number_of_snf	uint32_t	2-4	2	Number of SNF nodes present.
periphbase	uint64_t	-	0x2C000000	Value for PERIPHBASE. Only bits [43:24] are used.
sbas_bridge_present	bool	true, false	true	Value for SBAS bridge presence.
sbsx_bridge_present	bool	true, false	true	Value for SBSX bridge presence.
systemaddrmap	uint64_t	0x0-0xfffffffff	0x0	Bitmap for 20 regions in the CCN interconnect. Every two bits describe the region type for the corresponding region.

4.4.12 CCN512 component

This section describes the model of r0p0 of the CCN12 interconnect component.

CCN512 - ports

This section describes the ports.

CCN512 has six downstream ports: four SNF ports for the memory controller, and two Acelite ports (HNI).

Table 4-45 CCN512 ports

Name	Protocol	Type	Description
pvbus_s_rnf[12]	PVBus	Slave	RNF upstream ports.
pvbus_s_rni[24]	PVBus	Slave	RNI upstream ports.
pvbus_m_hni[2]	PVBus	Master	HNI downstream port.
pvbus_m_snf[4]	PVBus	Master	SNF donwstream ports.
reset_in	Signal	Slave	Reset signal.

CCN512 - parameters

This section describes the parameters.

Table 4-46 CCN512 parameters

Name	Type	Allowed values	Default value	Description
acchannelen_rnf	uint32_t	0x0-0xfff	0xfff	Bitmap for each RNF upstream port to test if snoop requests are enabled.
acchannelen_rni	uint32_t	0x0-0xffffff	0xffffff	Bitmap for each RNI upstream port to test if DVM requests are enabled.
cache_size_in_mbytes	uint32_t	-	8	Size of the L3 cache to model.
cache_state_modelled	bool	true, false	true	Model the cache state.
force_on_from_start	bool	true, false	false	The component normally starts up with snooping disabled. Set this parameter to true to allow the model to start up as enabled without having to program it. This parameter affects simulation reset and not signal reset.
number_of_snf	uint32_t	2-4	2	Number of SNF nodes present.
periphbase	uint64_t	-	0x2C000000	Value for PERIPHBASE. Only bits [43:24] are used.
sbsx_bridge_present	bool	true, false	true	Value for SBSX bridge presence.
systemaddrmap	uint64_t	0x0-0xfffffffff	0x0	Bitmap for 20 regions in the CCN interconnect. Every two bits describe the region type for the corresponding region.

4.4.13 DMC_400 component

This section describes the DMC_400 component.

DMC_400 - about

This LISA+ component is a model of the ARM CoreLink™ DMC-400 Dynamic Memory Controller.

The configuration of this model, by setting the registers, does not generally affect accesses to main memory.

DMC_400 - ports

This section describes the ports.

Table 4-47 DMC_400 ports

Name	Protocol	Type	Description
apb_interface	PVBus	Slave	Slave bus interface for register access
axi_if_in[4]	PVBus	Slave	Slave bus for connecting to bus decoder
axi_if_out[4]	PVBus	Master	Master to connect to DRAM

DMC_400 - parameters

This section describes the parameters.

Table 4-48 DMC _400 parameters

Name	Type	Allowed values	Default value	Description
diagnostics	int	0-4	0	Report diagnostics, with increasing detail as the setting increases
ECC_SUPPORT	bool	true/false	true	Error correction support (affects only reset values of some registers)
IF_CHIP0	int	-1, 0	-1	0 if port axi_if_out[0] connects chip 0 to RAM, -1 if not
IF_CHIP1	int	-1, 0	-1	0 if port axi_if_out[1] connects chip 1 to RAM, -1 if not
IF_CHIP2	int	-1, 0	-1	0 if port axi_if_out[2] connects chip 2 to RAM, -1 if not
IF_CHIP3	int	-1, 0	-1	0 if port axi_if_out[3] connects chip 3 to RAM, -1 if not
MEMORY_WIDTH	int	16, 32, 64	32	Memory width
revision	string	r0p1, r1p0, r1p1, r1p2	r0p1	The revision modeled, reflected in the ID register value

DMC_400 - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies.

This component has no timing information, so changing the values of the timing registers has no effect on behavior. The memory models do not attach to the component, and error checking does not update registers because the model does not include the possibility of errors.

DMC_400 - verification and testing

This component passes checks of the reset values for the registers against the TRM, as part of automated tests of a development system model.

4.4.14 DP500 component

This section describes the DP500 component.

DP500 - about

This component is a model of r1p0 of the DP500 Display Processor, with basic support for the display and scaling engines. Connect it to a visualization component to view LCD output.

It is a C++ component with a LISA wrapper.

It passes tests as part of a booting Android kernel and running under the control of the official DP500 drivers.

Functions

All RGB and YUV format parsing, except for the *ARM Frame Buffer Compression* (AFBC) formats.
Color adjustment in *Display Engine* (DE).
Nearest neighbor scaling.
All layers.
Alpha blending.
Memory writeback.
Inverse gamma adjustment.
Basic layer (overlay) and register security semantics.

Limitations

No support for polyphase scaling algorithm, only support for nearest neighbor.
No support for 3D or interlaced video.
No QoS support.
No support for image enhancing functionality in the *Scaling Engine* (SE).

No AFBC support.
No dithering support.
Power/test modes are modeled only as register state changes.
No chroma keying support.
No gamma adjustment support.
No support for the YUV420p2 format as an output format.

DP500 - ports

This section describes the ports.

Table 4-49 DP500 ports

Name	Protocol	Type	Description
clk_in	ClockSignal	Slave	Master clock input, typically 12.6MHz, to drive pixel clock timing.
display	LCD	Master	Connection to visualization component.
intr	Signal	Master	Interrupt signaling from display engine.
intr_se	Signal	Master	Interrupt signaling from scaling engine.
pvbus_m	PVBus	Master	DMA port for video data.
pvbus_s	PVBus	Slave	Slave port for connection for APB access.
reset_signal	Signal	Slave	Slave port for external reset line.

DP500 - parameters

This section describes the parameters.

Table 4-50 DP500 parameters

Name	Type	Allowed values	Default value	Description
force_frame_rate	uint32_t	0-60	0	0 Model uses input clock as PXLCLOCK. >0 Model refreshes screen display *n* times per simulated second.
register_debug	bool	true, false	false	Print summary of register reads and writes.

DP500 - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies.

4.4.15 DP550 component

This section describes the DP550 component.

DP550 - about

This component is a model of r0p0 of the DP550 Display Processor. Connect it to a visualization component to view LCD output.

It is a C++ component with a LISA wrapper.

It passes tests as part of a booting Android kernel and running under the control of the official DP550 drivers.

Functions

All RGB and YUV format parsing, except *ARM Frame Buffer Compression* (AFBC) formats.
Color adjustment in *Display Engine* (DE).
Nearest neighbor scaling.
All layers.
Alpha blending.
Memory writeback.
Inverse gamma adjustment.
Basic layer (overlay) and register security semantics.

Limitations

No support for polyphase scaling algorithm, only support for nearest neighbor.
No support for 3D or interlaced video.
No QoS support.
No support for image enhancing functionality in the *Scaling Engine* (SE).
No AFBC support.
No dithering support.
Power/test modes are modeled only as register state changes.
No chroma keying support.
No gamma adjustment support.
No support for the YUV420p2 format as an output format.

DP550 - ports

This section describes the ports.

Table 4-51 DP550 ports

Name	Protocol	Type	Description
`clk_in`	ClockSignal	Slave	Master clock input, typically 12.6MHz, to drive pixel clock timing.
`display`	LCD	Master	Connection to visualization component.
`intr`	Signal	Master	Interrupt signaling from display engine.
`intr_se`	Signal	Master	Interrupt signaling from scaling engine.
`pvbus_m`	PVBus	Master	DMA port for video data.
`pvbus_s`	PVBus	Slave	Slave port for connection for APB access.
`reset_signal`	Signal	Slave	Slave port for external reset line.

DP550 - parameters

This section describes the parameters.

Table 4-52 DP550 parameters

Name	Type	Allowed values	Default value	Description

`force_frame_rate`	`uint32_t`	`0-60`	`0`	`0`
				Model uses input clock as `PXLCLOCK`.
				`>0`
				Model refreshes screen display *n* times per simulated second.

`register_debug`	`bool`	`true`, `false`	`false`	Print summary of register reads and writes.

DP550 - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies.

4.4.16 ElfLoader component

This section describes the ElfLoader component.

ElfLoader - about

This component provides an alternative method of loading elf files into the system.

This is a LISA+ component.

ElfLoader - ports

This section describes the ports.

Table 4-53 ElfLoader ports

Name	Protocol	Type	Description
`pvbus_m`	PVBus	Master	Master port for all memory accesses
`start address`	Value_64	Master	Provides a value reflecting the entry point of the last ELF image to be loaded

ElfLoader - parameters

This section describes the parameters.

Table 4-54 ElfLoader parameters

Name	Type	Allowed values	Default value	Description
`elf`	String	-	[Empty string]	ELF file
`lfile`	String	-	[Empty string]	Load file for large address mapping
`ns_copy`	Boolean	`true`, `false`	`true`	Copy whole file to NS memory space

ElfLoader - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.17 FlashLoader component

This section describes the FlashLoader component.

FlashLoader - about

This 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.

This is a LISA+ component.

Related concepts

IntelStrataFlashJ3 - about on page 4-324.

FlashLoader - ports

This section describes the ports.

Table 4-55 FlashLoader ports

Name	Protocol	Type	Description
flash_device0 flash_device1 flash_device2 flash_device3	FlashLoaderPort	Master	Used to program a flash device

FlashLoader - parameters

This section describes the parameters.

Table 4-56 FlashLoader parameters

Name	Type	Allowed values	Default value	Description
fname	string	-	'(none)'	File to load into flash. The default '(none)' means do not load any file. An empty string causes a warning.
fnameWrite	string	-	'(none)'	File to save into from flash. The default '(none)' means do not save any file. An empty string causes a warning.

FlashLoader - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.18 GenericTimer component

This section describes the GenericTimer component.

CounterModule component

This section describes the CounterModule component.

CounterModule - parameters

This section describes the parameters.

Table 4-57 CounterModule parameters

Parameter	Allowed values	Default value	Description
base_frequency	-	100000000	Reset value for CNTFID0, base frequency in Hz.
diagnostics	0x0-0x4	0x0	Diagnostic verbosity.
non_arch_fixed_frequency	-	0x0	If set, ignore CNTFID0 and instead use this frequency in Hz.
non_arch_start_at_default	true, false	false	Firmware is expected to enable the timer at boot time. However, turning on this parameter is a model-specific method of enabling the CounterModule out of reset.
readonly_is_WI	true, false	false	Ignore, instead of failing, on writes to read-frame.
use_real_time	true, false	false	Update the GenericTimer counter at a real-time base frequency instead of simulator time.

Timer component

This section describes the Timer component.

Timer - parameters

This section describes the parameters.

Table 4-58 Timer parameters

Parameter	Allowed values	Default value	Description
diagnostics	0-4	0x0	Diagnostic verbosity.
frame_security	-	"	Hard-wired, configurable security for frames, N/S/X, one character per timer frame.

4.4.19 GIC_400 component

This section describes the GIC_400 component.

GIC_400 - about

This LISA+ component is a model of r0p1 of 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.

The GIC-400 has several memory-mapped interfaces at the same address. The processor that is communicating with the GIC-400 banks them. The GIC-400 must be able to distinguish from which processor a transaction originates. In the hardware, the AUSER fields on AXI supply this information to the GIC-400. In Fast Models, there is no exact equivalent to this field. However, each transaction has a `master_id` that the model can use to identify the originating processor.

ARM clusters assign the `master_id` as follows:

- Bits[31:16]: SBZ (which the GIC-400 ignores).
- Bits[5:2]: CLUSTERID.
- Bits[1:0]: cpu_id within cluster.

CLUSTERID is the 4-bit field that either a parameter on the processor sets or a value on the `clusterid` port drives. CPUID is the core 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. This distinction is essential for the VGIC to identify the cluster. The parameters in the GIC-400 component permit it to construct the map of `master_id` to interface number.

Processor interfaces that the GIC-400 supports have these parameters:

- `interfaceN.cluster_id`.
- `interfaceN.core_id`.
- `interfaceN.inout_port_number_to_use`.

N is the interface number (0-7). The `cluster_id` and `core_id` tell the GIC-400 to map that cluster or core combination to interface N.

In using `inout_port_number_to_use`, the GIC-400 has some input and output ports that pair with a particular processor interface. For example:

- The `irqcpu[]` pin wires to the `irq` port of the corresponding processor.
- The `cntpnsirq` pin from the processor wires to a `cntpnsirq[]` pin on GIC-400 to transport a *Private Peripheral Interrupt* (PPI) from the processor to the GIC-400.

The `interfaceN.inout_port_number_to_use` parameter supports clusters that can have variable numbers of cores. It tells the GIC-400 that to send to or receive a signal from the processor that is attached to interface N, it must use these 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].
- ...

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. If the control registers of the GIC-400 are set up in particular ways, they can also bypass the GIC-400. See the *ARM Generic Interrupt Controller Architecture version 2.0 Architecture Specification* for more information.

The fabric between the clusters and the GIC might remap the master_id of a transaction. 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. It ignores the rest. 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. 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.

The GIC-400 model has these limitations:

- Reads and writes to GICD_ISACTIVERn/GICD_ICACTIVERn/GICD_ISPENDRn/ GICD_ICPENDRn might not work as expected unless there is a configured target in GICD_ICFGRm.
- Some of the interaction of GICD_CTLR.EnableGrpX and level sensitive interrupts might not work entirely correctly.
- It does not model the signals nIRQOUT/nFIQOUT.
- It models interrupts with positive logic, rather than the negative logic that the hardware uses. Hence, the signal pins omit the “n” prefix in their names.

GIC_400 - ports

This section describes the ports.

Table 4-59 GIC_400 ports

Name	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.
fiqout	Signal	Master	FIQOUT signal to the corresponding processor.
irqcpu	Signal	Master	IRQ signal to the corresponding processor.
irqout	Signal	Master	IRQOUT 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.

Table 4-59 GIC_400 ports (continued)

Name	Protocol	Type	Description
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.

GIC_400 - parameters

This section describes the parameters.

Table 4-60 GIC_400 parameters

Name	Type	Allowed values	Default value	Description
NUM_CPUS	Integer	1-8	1	Number of interfaces to support.
NUM_SPIS	Integer	0-480	224	Number of shared peripheral interrupt pins.
interface0.cluster_id	Integer	0-15	0	Cluster ID of the interface you want to appear as interface0 in the VGIC.
interface0.core_id	Integer	0-15	0	Core ID of interface0 in the cluster.
interface0.inout_port_number_to_use	Integer	0-7	0	ppiN port used for this interface.
interface1.cluster_id	Integer	0-15	0	Cluster ID of the interface you want to appear as interface1 in the VGIC.
interface1.core_id	Integer	0-15	0	The Core ID of interface1 in the cluster.
interface1.inout_port_number_to_use	Integer	0-7	1	ppiN port used for this interface.
interface2.cluster_id	Integer	0-15	0	Cluster ID of the interface you want to appear as interface2 in the VGIC.
interface2.core_id	Integer	0-15	0	Core ID of interface2 in the cluster.
interface2.inout_port_number_to_use	Integer	0-7	2	ppiN port used for this interface.
interface3.cluster_id	Integer	0-15	0	Cluster ID of the interface you want to appear as interface3 in the VGIC.
interface3.core_id	Integer	0-15	0	Core ID of interface3 in the cluster.
interface3.inout_port_number_to_use	Integer	0-7	3	ppiN port used for this interface.
interface4.cluster_id	Integer	0-15	0	Cluster ID of the interface you want to appear as interface4 in the VGIC.
interface4.core_id	Integer	0-15	0	Core ID of interface4 in the cluster.
interface4.inout_port_number_to_use	Integer	0-7	4	ppiN port used for this interface.
interface5.cluster_id	Integer	0-15	0	Cluster ID of the interface you want to appear as interface5 in the VGIC.
interface5.core_id	Integer	0-15	0	Core ID of interface5 in the cluster.
interface5.inout_port_number_to_use	Integer	0-7	5	ppiN port used for this interface.

Table 4-60 GIC_400 parameters (continued)

Name	Type	Allowed values	Default value	Description
`interface6.cluster_id`	Integer	0-15	0	Cluster ID of the interface you want to appear as `interface6` in the VGIC.
`interface6.core_id`	Integer	0-15	0	Core ID of `interface6` in the cluster.
`interface6.inout_port_number_to_use`	Integer	0-7	6	ppiN port used for this interface.
`interface7.cluster_id`	Integer	0-15	0	Cluster ID of the interface you want to appear as `interface7` in the VGIC.
`interface7.core_id`	Integer	0-15	0	Core ID of `interface7` in the cluster.
`interface7.inout_port_number_to_use`	Integer	0-7	7	ppiN port used for this interface.

GIC_400 - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies.

GIC_400 - verification and testing

This component passes tests as part of a system with network functionalities.

4.4.20 GIC500Distributor component

This section describes the GIC500Distributor component.

GIC500Distributor - about

This LISA+ component is a model of r0p0 of the GIC-500 *Generic Interrupt Controller* (GIC) for distribution of interrupts.

This component is a monolithic implementation of GICv3 architecture with support for 128 cores. You can configure the model to support a maximum of 32 clusters with eight cores per cluster. Use it with an ARMv8-A core to deliver interrupts. It supports a single Interrupt Translation Service for message-based interrupts. It supports the architectural features, but does not support the IMPLEMENTATION DEFINED features.

GIC500Distributor - ports

This section describes the ports.

Table 4-61 GIC500Distributor ports

Name	Protocol	Type	Description
`cfgsdisable`	Signal	Slave	Disable some SPI signals.
`cpu_active_`*n*`[8]`	Signal	Slave	`cpu_active` pins. 0 <= *n* <= 31.
`po_reset`	Signal	Slave	Power-on reset.
`ppi`*x*`_in_`*n*`[8]`	Signal	Slave	Private peripheral interrupts. 0 <= *n* <= 31. 16 <= *x* <= 31.
`pvbus_m`	PVBus	Master	Memory bus out: passthrough for undecoded accesses, plus allows redistributors to access main memory.
`pvbus_s`	PVBus	Slave	Memory bus in: this interface accepts memory-mapped register accesses.
`redistributor_m[256]`	GICv3Comms	Master	Input from and output to core interface.
`reset`	Signal	Slave	Reset.

Table 4-61 GIC500Distributor ports (continued)

Name	Protocol	Type	Description
spi_in[988]	Signal	Slave	Shared peripheral interrupts.
wake_request_*n*[8]	Signal	Master	Power management outputs. 0 <= *n* <= 31.

Related references

2.5.2 GICv3Comms protocol on page 2-64.

GIC500Distributor - parameters

This section describes the parameters.

Table 4-62 GIC500Distributor parameters

Name	Type	Allowed values	Default value	Description
cpu_affinities	string	-	""	Processing element affinity.
cpus_per_cluster_*n*	int	1-8	1	Number of cores in cluster *n*. 0 <= *n* <=31.
delay_ITS_accesses	bool	true, false	true	Delay accesses from the ITS until the architecture specification requires it to be complete.
delay_redistributor_accesses	bool	true, false	true	Delay memory accesses from the redistributor until GICR_SYNCR is read.
enable_protocol_checking	bool	true, false	false	Enables protocol checking at core interface.
enabled	bool	true, false	true	Enable GICv3 functionality, when false the component is inactive.
gicd_alias	uint64_t	-	0x2c001000	Additional 4KB page in memory for alias of GICD registers.
gicv2_only	bool	true, false	false	If true, pretend to be a GICv2 system.
has_two_security_states	bool	true, false	true	If true, has two security states.
itargets_razwi	bool	true, false	false	Implement GICD_ITARGETSRn registers as RAZ/WI.
its_count	int	0-1	1	Number of ITS components.
its_device_bits	int	3-20	16	Number of bits supported for ITS device IDs.
its_entry_size	int	1-1024	8	Number of bytes required to store each entry in the ITT tables.
ITS_threaded_command_queue	bool	true, false	true	Enable execution of ITS commands in a separate thread, which cosimulation sometimes requires.

Table 4-62 GIC500Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
its0_base	uint64_t	0x0-0xffffffffFFFFFFFF	0x0	Register base address for ITS0 (automatic if 0).
non_ARE_core_count	int	1-8	8	Maximum number of non-ARE cores; normally used to pass the cluster-level NUM_CORES parameter to the top-level redistributor.
num_clusters	int	1-32	1	Number of clusters.
pa_size	int	32-48	48	Number of valid bits in physical address.
processor_numbers	string	-	""	Specify processor numbers (as appear in GICR_TYPER) in the form 0.0.0.0=0,0.0.0.1=1, ...). By default, processor numbers start at 0.
reg_base	uint64_t	-	0x2c010000	Base for decoding GICv3 registers.
reg_base_per_redistributor	string	-	""	Base address for each redistributor in the form '0.0.0.0=0x2c010000, 0.0.0.1=0x2c020000'.[bt]
spi_count	int	0-988	224	Number of implemented SPIs.

4.4.21 GICv3Distributor component

This section describes version 3 of the *Generic Interrupt Connect* (GIC) component.

GICv3Distributor - about

The GICv3Distributor component is a generic architectural model of a GICv3 distributor. You can configure it to mimic a GICv3 distributor implementation.

To use this component, you need:

- An ARMv8-A core model that communicates over the stream interface (defined in the GICv3 engineering specification) to distribute interrupts.
- A SystemIP license, separate from ARMv8-A core licenses.

This component has ports with up to 256 processing elements. There is a separate port for each communications link to a core.

Use the CPU-affinities parameter to communicate to the model the affinity address of each of the ports. Match the affinity addresses to the affinity values in the MPIDR registers of the connected core. For *n* affinity addresses, associated processing elements connect to redistributor_m[0] up to redistributor_m[*n*-1].

You can use this component as a filter for memory transactions. It forwards transactions that enter it on pvbus_s but that do not match any memory address range that is configured for the distributor.

[bt] Specify all redistributors. This overrides the reg_base parameter (except that the top-level redistributor still uses reg_base).

In systems without cache state modeling and the memory interconnect, you can connect this component serially between the master and the memory system. It catches transactions for the distributor and passes transactions for other components to the `pvbus_s` port. The distributor is also a transaction master, so transactions out of `pvbus_m` must be able to access memory that is associated with GICv3 behaviors.

For more advanced systems, to separate generated and forwarded traffic, you can use the distributor as a filter using port `pvbus_filtermiss_m`. When you connect this port to a PVBus slave, generated traffic is output on port `pvbus_m` while forwarded traffic is output on `pvbus_filtermiss_m`. This arrangement allows you to:

- Flexibly configure the GIC.
- Forward the traffic to a point in the memory system where transactions can observe memory as expected in an implementation.

Finally, if the GICv3 distributor has a predictable address range, connect `pvbus_m` to a suitable point in the memory system, and connect `pvbus_filtermiss_m` to a transaction sink. In theory, these platforms have no forwarded traffic. However, this precaution creates a behavior to match implementations and protect against the creation of a cyclical path for a memory transaction.

Related references

GICv4 - functionality on page 1-18.

GICv3Distributor - ports

This section describes the ports.

Table 4-63 GICv3Distributor ports

Name	Protocol	Type	Description
`cfgsdisable`	Signal	Slave	Disable some SPI signals.
`po_reset`	Signal	Slave	Power-on reset.
`ppi_in_`*n*`[16]`	Signal	Slave	Private peripheral interrupts. 0 <= *n* <= 255.
`pvbus_filtermiss_m`	PVBus	Master	Passthrough for undecoded accesses, when connected.
`pvbus_m`	PVBus	Master	Memory bus out.[bu]
`pvbus_s`	PVBus	Slave	Memory bus in. The component accepts memory-mapped register accesses through this interface.
`redistributor_m[256]`	GICv3Comms	Master	Input from and output to core interface.
`reset`	Signal	Slave	Reset.
`spi_in[988]`	Signal	Slave	Shared peripheral interrupts.
`wake_request[256]`	Signal	Master	Power management outputs.

GICv3Distributor - parameters

This section describes the parameters. The processor has separate gic parameters, which are both gicv3 and gicv2.

[bu] If you do not connect `pvbus_filtermiss_m` then it is the passthrough for undecoded accesses, plus it allows redistributors to access main memory. If you do connect `pvbus_filtermiss_m`, then only redistributor accesses to main memory appear on this port.

Table 4-64 GICv3Distributor parameters

Name	Type	Allowed values	Default value	Description
a3_affinity_supported	bool	true, false	false	If true, support affinity level 3 values that are nonzero.
ARE_fixed_to_one	bool	true, false	false	If true, the model does not support GICv2 compatibility and GICD_CTLR.ARE_* is always one.
ARE_set_once	bool	true, false	false	If true, you can set GICD_CTLR.ARE_* but you can only reset them through reset.
collection_count	int	0-65536	1024	Number of collections that each *Interrupt Translation Services* (ITS) component supports.
cpu_affinities	string	-	""	Processing element affinity.
delay_ITS_accesses	bool	true, false	true	If true, delay accesses from the ITS until the model reads GICR_SYNCR (that is, until the architecture specification requires completion).
delay_redistributor_accesses	bool	true, false	true	If true, delay memory accesses from the redistributor until the model reads GICR_SYNCR.
DPG_bits_implemented	bool	true, false	true	If true, enable implementation of interrupt group participation bits or DPG bits in GICR_CTLR.
DS_fixed_to_zero	bool	true, false	false	If true, enable support of single security state.
enable_protocol_checking	bool	true, false	false	If true, enable protocol checking at the core interface.

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
enabled	bool	true, false	true	If true, enable this GICv3 component.
fixed_routed_spis	string	-	""	Value of IROUTER[*n*] register in the form '*n*=a.b.c.d, *n*=*'. The RM bit of IROUTER is 0 for *n* = a.b.c.d, else 1 for *n* = *. 32 <= *n* <= 1019.
gicd_alias	uint64_t	-	0x2C001000	Additional 4KB page in memory for the alias of GICD registers.
gicd_pidr	uint64_t	-	0x0	The value for the GICD_PIDR registers, if nonzero. Note: changing this value overrides fixed fields, such as device type.
gicr_pidr	uint64_t	-	0x0	The value for the GICR_PIDR registers, if nonzero. Note: changing this value overrides fixed fields, such as device type.
gicr_propbaser_read_only	bool	true, false	false	If true, the GICR_PROPBASER register is read-only.
gicr_propbaser_reset	uint64_t	-	0x0	Value of GICR_PROPBASER on reset.
gicv2_only	bool	true, false	false	If true, if using the GICv3 model, pretend to be a GICv2 system.
gits_baser*n*_entry_bytes	unsigned	1-256	8	Number of bytes for each entry for the GITS_BASER*n* register. 0 <= *n* <= 7.

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
gits_baser*n*_type	unsigned	0x0-0x4	0x0	Type field for the GITS_BASER*n* register. 0 <= *n* <= 7. 0x0, unimplemented. 0x1, devices. 0x2, virtual processors. 0x3, physical processors. 0x4, collections.
gits_pidr	uint64_t	-	0x0	The value for the GITS_PIDR registers, if nonzero. Note: changing this value overrides fixed fields, such as device type.
has_two_security_states	bool	true, false	true	If true, has two security states.
icfgr_ppi_mask	int	0x0-0xFFFFFFFF	0xAAAAAAAA	Mask for writes to ICFGR registers that configure PPIs.
icfgr_ppi_reset	int	0x0-0xFFFFFFFF	0x0	Reset value for ICFGR registers that configure PPIs.
icfgr_ppi_rsvd_bits	int	0x0-0xFFFFFFFF	0x0	If ARE bit = 0, the value of reserved bits, that is, bits 0,2,4..30 of ICFGR*n* for *n* > 0.
icfgr_sgi_mask	int	0x0-0xFFFFFFFF	0x0	Mask for writes to ICFGR registers that configure SGIs.
icfgr_sgi_reset	int	0x0-0xFFFFFFFF	0xAAAAAAAA	Reset value for ICFGR registers that configure SGIs.
icfgr_spi_mask	int	0x0-0xFFFFFFFF	0xAAAAAAAA	Mask for writes to ICFGR registers that configure SPIs.
icfgr_spi_reset	int	0x0-0xFFFFFFFF	0x0	Reset value for ICFGR registers that configure SPIs.

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
ignore_generate_sgi_when_no_are	bool	true, false	false	If true, ignore GenerateSGI packets from the core interface if both ARE_S and ARE_NS are 0.
igroup_ppi_mask	int	0x0-0xFFFF	0xFFFF	Mask for writes to PPI bits in IGROUP registers.
igroup_ppi_reset	int	0x0-0xFFFF	0x0	Reset value for SGI bits in IGROUP registers.
igroup_sgi_mask	int	0x0-0xFFFF	0xFFFF	Mask for writes to SGI bits in IGROUP registers.
igroup_sgi_reset	int	0x0-0xFFFF	0x0	Reset value for SGI bits in IGROUP registers.
IIDR	uint32_t	0x0-0xFFFFFFFF	0x0	GICD_IIDR and GICR_IIDR value.
irouter_default_mask	string	-	""	Default Mask value for the IROUTER[32..1019] register in the form a.b.c.d.
irouter_default_reset	string	-	""	Default Reset value of the IROUTER[32..1019] register in the form a.b.c.d or *.
irouter_reset_values	string	-	""	Reset value of the IROUTER[*n*] register in the form *n*=a.b.c.d or *n*=*. 32 <= *n* <= 1019.
irouter_mask_values	string	-	""	Mask value of the IROUTER[*n*] register in the form *n*=a.b.c.d. 32 <= *n* <= 1019.
itargets_razwi	bool	true, false	false	If true, the GICD_ITARGETS registers are RAZ/WI.[bv]

[bv] Legacy routing, GICv2-style, fixes interrupts to target the first processor in the system.

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
its_count	int	0x0-0x4	0x1	Number of *Interrupt Translation Services* (ITS) components.
its_device_bits	int	1-32	16	Number of bits that ITS device IDs support.
its_entry_size	int	0x1-0x400, 1-1024	0x8, 8	Number of bytes to store each entry in the ITT tables.
ITS_hardware_collection_count	int	0x0-0xFF, 0-255	0x0, 0	Number of hardware collections that the ITS holds exclusively.
its_id_bits	int	14-24	16	Number of interrupt ID bits that ITS supports.
ITS-legacy-iidr-typer-offset	bool	true, false	false	If true, put the GITS_IIDR and GITS_TYPER registers at their older offset of 0x8and 0x4, respectively.
ITS_MOVALL_update_collections	bool	true, false	false	If true, the MOVALL command updates the collection entries.
ITS-supported-collection-count		0x0-0x400	0x400	Number of collections that each ITS component supports.
ITS_threaded_command_queue	bool	true, false	true	If true, enable the execution of ITS commands in a separate cosimulation thread.
ITS_TRANSLATE64R	bool	true, false	false	If true, add an implementation specific register at 0x10008 supporting 64-bit TRANSLATER (dev[63:32], interrupt[31:0]).

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
ITS_use_physical_target_addresses	bool	true, false	true	If true, use physical hardware addresses for targets in ITS commands. Must be true for distributed implementations.
its*n*_base	uint64_t	0x0-0xffffffffFFFFFFFF	0x0	Register base address for ITS*n*. This is automatic if 0. 0 <= *n* <= 3.
has-two-security-states	bool	true, false	true	If true, support two security states. If false, the GICD_CTLR.DS, DisableSecurity bit has a fixed value of 1.
local_SEIs	bool	true, false	false	If true, support locally generated SEIs to report internal issues.
local_VSEIs	bool	true, false	false	If true, support locally generated VSEIs to report internal issues.
lockable_spi_count	int	0-31	0	Number of SPIs that the model locks down on assertion of the **CFGSDISABLE** signal. This only applies for GICv2.
LPI_cache_type	int	0x0-0x1	0x1	Cache type for LPIs. 0x0, no caching. 0x1, extensive caching.
LPI_cache_check_data	bool	true, false	false	If true, enable cached LPI data against memory checking when available for cache type.
monolithic	bool	true, false	false	If true, the distributor is monolithic rather than distributed (modifies register behavior).

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
non_ARE_core_count	int	1-8	8	Maximum number of non-ARE cores. The normal use is passing the cluster-level NUM_CORES parameter to the top-level redistributor.
pa_size	int	32-48	48	Number of valid bits in physical address.
physical_id_bits	int	16-24	16	Number of bits that represent physical interrupt ID.
ppi_implemented_mask	int	0x0-0xFFFF	0xFFFF	Mask of PPIs that the model implements. One bit for each PPI bit 0 == PPI 16, first PPI. This affects other masks.
priority_bits_implemented	uint32_t	4-8	8	Number of priority bits that the model implements.
processor_numbers	string	-	""	Specifies processor numbers, as appear in GICR_TYPER, in the form 0.0.0.0 = 0,0.0.0.1 = 1. If this is not set, processor numbers start at 0.
redistributor_threaded_command_queue	bool	true, false	true	If true, enable the execution of redistributor delayed transactions in a cosimulation thread.
reg_base	uint64_t	-	0x2C010000	Base for decoding GICv3 registers.
reg_base_per_redistributor	string	-	""	Base address for each redistributor in the form '0.0.0.0 = 0x2C010000, 0.0.0.1 = 0x2C020000'. [bw]
spi_count	int	0x0-0x3DC, 0-988	0xE0, 224	Number of SPIs that the model implements.

[bw] Specify all redistributors. This overrides the reg_base parameter for uses except that of the top-level redistributor.

Table 4-64 GICv3Distributor parameters (continued)

Name	Type	Allowed values	Default value	Description
`statusr_implemented`	`bool`	`true`, `false`	`true`	If `true`, implement the GICR_STATUSR register.
`supports_shareability`	`bool`	`true`, `false`	`true`	If `true`, support shareability attributes on outgoing memory bus, that is, model an ACElite port not an AXI4 port.
`trace_speculative_lpi_property_updates`	`bool`	`true`, `false`	`false`	If `true`, perform LPI property updates on speculative accesses. Useful for debugging LPI.
`virtual_id_bits`	`int`	`16-24`	`16`	Number of bits that represent virtual interrupt ID.
`virtual_lpi_support`	`bool`	`true`, `false`	`false`	If `true`, support GICv4 virtual LPIs and direct injection of virtual LPIs.

Related references

GICv4 - functionality on page 1-18.

4.4.22 HostBridge component

This section describes the HostBridge component.

HostBridge - about

This LISA+ component is a model of 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.

Related tasks

Configuring the networking environment for Microsoft Windows on page 1-36.
Configuring the networking environment for Linux on page 1-39.

Related information

Fast Models User Guide.

HostBridge - ports

This section describes the ports.

Table 4-65 HostBridge ports

Name	Protocol	Type	Description
eth	VirtualEthernet	Master	Send or receive Ethernet frame

Related references

2.4.18 VirtualEthernet protocol on page 2-63.

HostBridge - parameters

This section describes the parameters.

Table 4-66 HostBridge parameters

Name	Type	Allowed values	Default value	Description
interfaceName	string	Valid string characters	'ARM0'	Host interface.
userNetPorts	string	Formatted string	''	Listening ports to expose in user-mode networking.
userNetSubnet	string	Formatted string	'172.20.51.0/24'	Virtual subnet for user-mode networking.
userNetworking	bool	true, false	false	Enable user-mode networking.[bx]

HostBridge - verification and testing

This component passes tests as part of a system with network functionalities.

HostBridge - performance

ARM expects this component to have little effect on the performance of PV systems. However, heavy usage on the networking of the model, for example to run networking performance tests, might slow down the simulation.

4.4.23 ICS307 component

This section describes the ICS307 component.

ICS307 - about

This LISA+ component is a model of an ICS307 clock divider. You can use it 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.

Three values determine the divisor ratio:

- vdw.
- rdw.
- od.

To calculate the divisor ratio, use:

```
Divisor = ((rdw+2) * scale) / (2 * (vdw+8))
```

where scale is derived from this table indexed by od:

[bx] TAP/TUN mode is enabled by default, and is disabled when user mode is in use.

Table 4-67 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
```

ICS307 - ports

This section describes the ports.

Table 4-68 ICS307 ports

Name	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

Related references

2.4.5 ICS307Configuration protocol on page 2-56.

ICS307 - parameters

This section describes the parameters.

Table 4-69 ICS307 parameters

Name	Type	Allowed values	Default value	Description
`vdw`	Integer	0-255	4	Used to calculate clock divider ratio
`rdr`	Integer	0-255	6	Used to calculate clock divider ratio
`od`	Integer	0-7	3	Used to calculate clock divider ratio

ICS307 - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

ICS307 - performance

ARM expects this component to have little effect on the performance of PV systems. However, modifying the ICS307 timing parameters is relatively slow, so ARM recommends you do so rarely.

4.4.24 IntelStrataFlashJ3 component

This section describes the IntelStrataFlashJ3 component.

IntelStrataFlashJ3 - about

This component is an efficient implementation of a NOR Flash memory type device, an Intel StrataFlash Memory (J3).

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.

This component implementation is approximately that of the Intel part in the VE development board. The component is effectively organized as a bank of two 16-bit Intel Flash components forming a 32-bit component that can be read or programmed in parallel. The component supports all hardware behavior except for:

- Protection register.
- Enhanced configuration register.
- Unique device identifier.
- One time programmable cells.
- 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.

This is a LISA+ component.

Related concepts

FlashLoader - about on page 4-305.

Related information

Intel Download Center, Intel StrataFlash Memory (J3) datasheet.

IntelStrataFlashJ3 - ports

This section describes the ports.

Table 4-70 IntelStrataFlashJ3 component ports

Name	Protocol	Type	Description
`flashloader`	FlashLoaderPort	Slave	Permits a FlashLoader component to initialize the flash contents from a binary file
`mbs_control`	PVBusSlaveControl	Master	ARM test control
`mem_port`	PVDevice	Slave	ARM test control
`pvbus`	PVBus	Slave	Slave port for connection to PV bus master/decoder

Related references

2.4.3 FlashLoaderPort protocol on page 2-55.

IntelStrataFlashJ3 - parameters

This section describes the parameters.

Table 4-71 IntelStrataFlashJ3 component parameters

Name	Type	Allowed values	Default value	Description
`diagnostics`	`uint32_t`	0-4	0	Diagnostic level.
`model_blocklock`	`bool`	`true`, `false`	`false`	Per block locking.
`size`	`uint64_t`	≥256KB, ≤4GB	256KB	Component size in bytes, in multiples of 256KB (18 bits).
`trapwrite`	`bool`	`true`, `false`	`false`	Generate abort on write. Use when modeling ROM with flash memory.
`unphysical_writes`	`bool`	`true`, `false`	`true`	Writes to flash are overwrite not AND. Use when debugging drivers.

IntelStrataFlashJ3 - registers

In normal operation, the component has no user visible registers, but you can read from it as if it is memory.

Programming it or changing the configuration requires a sequence of special write operations: see general flash programming documentation. Note that the model interprets all writes as requests to the programming state machine, and that there are many state-machine states that do not support subsequent reads and return `0xdeaddead` for them. Therefore, when simulating a ROM, use the `trapwrite=true` option. The component supports Common Flash Interface query operations, which allow drivers to determine the properties of the flash memory.

IntelStrataFlashJ3 - debug features

Read/write to this component using normal debugWrites.

Use the `diagnostics` parameter to select the level of diagnostic output:

Level 0
None.

Level 1
Report probable driver error operations:
- Unaligned operations that fault.
- Accesses that the state machine does not expect.
- Transitions of the state machine to unknown states.
- Writes to locked blocks and illegal lock commands.

Level 2
Report unimplemented and so ignored operations, and log lock commands.

Level 3
Warn if a flash write attempts to set bits (the write works if `unphysical_writes=true`).

Level 4
Log every read and write.

IntelStrataFlashJ3 - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.25 MemoryMappedCounterModule component

This section describes the MemoryMappedCounterModule component.

MemoryMappedCounterModule - about

This LISA+ component is a model of a memory mapped counter module.

It is essential for multiple clusters of cores with Generic Timers, and also for single core systems where the Generic Timer runs at a different rate to the input clock to the core.

MemoryMappedCounterModule - ports

This section describes the ports.

Note

The component has two bus slave ports because the architecture specification permits you to map each set of registers at different, noncontiguous base addresses.

Table 4-72 MemoryMappedCounterModule ports

Name	Protocol	Type	Description
`clk_in`	Clock signal	Slave	This clock input determines the frequency of the Physical Count provided to the clusters connected to the `cntvalueb` port.
`cntvalueb`	CounterInterface	Master	This master port implements a private protocol between the cluster and the MemoryMappedCounterModule. Connect this port to the `cntvalueb` port on each cluster in the system and to the MemoryMappedCounterModule component.
`pvbus_control_s`	PVBus	Slave	This slave port provides memory-mapped read/write access to the control registers of the module.
`pvbus_read_s`	PVBus	Slave	This slave port provides memory-mapped read access to a different set of registers. It is not implemented.

MemoryMappedCounterModule - parameters

This section describes the parameters.

Table 4-73 MemoryMappedCounterModule parameters

Name	Type	Allowed values	Default value	Description
`non_arch_start_at_default`	`bool`	`true`, `false`	`false`	A model-specific way of enabling the counter module out of reset.
`base_frequency`	`uint32_t`	-	`100000000`	Reset value for the CNTFID0 register; base frequency in Hz.
`use_real_time`	`bool`	`true`, `false`	`false`	Update the Generic Timer counter at a real-time base frequency instead of simulator time.
`non_arch_fixed_frequency`	`uint32_t`	-	`0`	If set, ignore CNTFID0 and instead use this frequency in Hz.
`cntcidr0123_C`	`uint32_t`	-	`0`	Values to return for control-frame CIDR registers.
`cntpidr0123_C`	`uint32_t`	-	`0`	Values to return for control-frame PIDR registers 0-3.
`cntpidr4567_C`	`uint32_t`	-	`0`	Values to return for control-frame PIDR registers 4-7.
`cntcidr0123_R`	`uint32_t`	-	`0`	Values to return for read-frame CIDR registers.
`cntpidr0123_R`	`uint32_t`	-	`0`	Values to return for read-frame PIDR registers 0-3.

Table 4-73 MemoryMappedCounterModule parameters (continued)

Name	Type	Allowed values	Default value	Description
cntpidr4567_R	uint32_t	-	0	Values to return for read-frame PIDR registers 4-7.
readonly_is_WI	bool	true, false	false	Ignore (rather than failing) on writes to read-frame.
diagnostics	uint32_t	0-4	0	Diagnostics.

MemoryMappedCounterModule - verification and testing

This component passes tests as part of a system with network functionalities.

4.4.26 MemoryMappedGenericWatchdog component

This section describes the MemoryMappedGenericWatchdog component.

MemoryMappedGenericWatchdog - about

This component is a model of the memory mapped counter module. It is a high-level watchdog, one which generates two interrupts rather than an interrupt then a reset.

MemoryMappedGenericWatchdog - ports

This section describes the ports.

Table 4-74 MemoryMappedGenericWatchdog - ports

Name	Protocol	Type	Description
cntvalueb	CounterInterface	Slave	-
ctl_pvbus_s	PVBus	Slave	-
ref_pvbus_s	PVBus	Slave	-
WS0	Signal	Master	-
WS1	Signal	Master	-

MemoryMappedGenericWatchdog - parameters

This section describes the parameters.

Table 4-75 MemoryMappedGenericWatchdog parameters

Name	Type	Allowed values	Default value	Description
diagnostics	uint32_t	0x0-0x4	0x0	Diagnostics
NONSECURE	bool	true, false	false	Non-Secure
product_id	uint8_t	-	0x0	Product identifier

4.4.27 MessageBox component

This section describes the MessageBox component.

MessageBox - about

This component permits passing of blocks of data, or messages, between the driver and the parent VFS2 device, with a suitable driver.

It operates as a subcomponent to the VFS2 component. It is not a hardware model, and is designed to operate efficiently within a Fast Models platform model.

This implementation is generic but primarily provides 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++.

This is a LISA+ component.

Related references

4.4.65 VFS2 component on page 4-395.

MessageBox - ports

This section describes the ports.

Table 4-76 MessageBox ports

Name	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 - parameters

This section describes the parameters.

Table 4-77 MessageBox parameters

Name	Type	Allowed values	Default value	Description
`id`	Integer	-	`0x01400400`	MessageBox ID

MessageBox - registers

This section describes the registers.

Table 4-78 MessageBox registers

Name	Offset	Access	Description
MESSAGEBOX_ID	`0x00`	Read only	Returns a user-configurable ID value
MESSAGEBOX_DATA	`0x04`	Read/write	MessageBox DATA register
MESSAGEBOX_CONTROL	`0x08`	Read/write	MessageBox CONTROL register
MESSAGEBOX_STATUS	`0x0C`	Read only	MessageBox STATUS signal bits
MESSAGEBOX_START	`0x10`	Read/write	MessageBox START register

Table 4-78 MessageBox registers (continued)

Name	Offset	Access	Description
MESSAGEBOX_END	0x14	Read/write	MessageBox END register
MESSAGEBOX_IRQMASK	0x18	Read/write	Controls which of the status bits can cause the interrupt signal to be asserted

MessageBox - 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.

MessageBox - CONTROL register

The CONTROL register issues commands to control message passing.

The register can have 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.

MessageBox - 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.

MessageBox - 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.

MessageBox - 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.

MessageBox - buffer

The data buffer is mapped starting at the device offset of `0x1000` and is 60KB in size, occupying the rest of the 64KB of device space. Do not use the first 4KB.

MessageBox - interrupts

The interrupt line is asserted through `sg::Signal::Set` whenever `STATUS & IRQMASK` is nonzero.

MessageBox - verification and testing

This component passes tests through use with Linux operating systems.

MessageBox - performance

ARM expects this component to have little effect on the performance of PV systems. The component depends on the performance of the host filesystem.

4.4.28 MMC component

This section describes the MMC component.

MMC - about

This LISA+ component is a model of a *MultiMediaCard* (MMC) device.

It can simulate an SD card compatible with the *MultiMedia Card Association* (MMCA, `www.jedec.org`) specification version 3.31, or an eMMC card compatible with version 4.4. You can extend functionality using the supplied source code.

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 can contain a partition table and filesystems such as FAT or EXT2. The image file is a direct bit copy of the contents of an SD card. If the image file that p_mmc_file refers to does not exist, the component behaves as if the card is absent. If the image file is read-only, then the component behaves as if the card is read-only. Note that operating-system boots often attempt to write to the boot filesystem. They might not work properly if the boot filesystem is on a read-only card.

The MMC component does not model card insertion or removal. It models having already been inserted at system instantiation time.

You can configure the MMC component to behave as an eMMC module by setting the card_type parameter to eMMC. In this mode, it handles the MMC_SLEEP_AWAKE, MMC_SWITCH, MMC_SEND_EXT_CSD, and MMC_SET_BLOCKLEN commands, which are otherwise illegal.

The component supports these commands:

- 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.
- MMC_SLEEP_AWAKE.
- MMC_SWITCH.
- MMC_SEND_EXT_CSD.
- MMC_SET_BLOCKLEN.

In SD mode, the block length is 512 bytes. SimGen reports attempts to change it as errors.

The component supports these erase commands (Class 5), but they have no effect on the disk backing storage:

- MMC_ERASE_GROUP_START.
- MMC_ERASE_GROUP_END.
- MMC_ERASE.

The component does not support these commands:

- MMC_BUSTEST_R.
- MMC_BUSTEST_W.

The component does not support stream read and write commands (Classes 1 and 3):

- MMC_READ_DAT_UNTIL_STOP.
- MMC_WRITE_DAT_UNTIL_STOP.
- MMC_PROGRAM_CID.
- MMC_PROGRAM_CSD.

The component does not support block oriented write protection commands (Class 6):

- MMC_SET_WRITE_PROT.
- MMC_CLR_WRITE_PROT.
- MMC_SEND_WRITE_PROT.

The component does not support lock card commands (Class 7) or application-specific commands (Class 8):

- MMC_LOCK_UNLOCK.
- MMC_APP_CMD.
- MMC_GEN_CMD.

The component does not support I/O mode commands (Class 9):

- MMC_FAST_IO.
- MMC_GO_IRQ_STATE.

The component does not support reserved commands. Using a reserved command sets the MMC ST_ER_B_ILLEGAL_COMMAND bit in the status register of the card. Read this with the MMC_SEND_STATUS command.

MMC - ports

This section describes the ports.

Table 4-79 MMC ports

Name	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.

Related references

2.4.9 MMC_Protocol protocol on page 2-57.

MMC - parameters

This section describes the parameters.

The parameters permit configuration of a number of attributes reflected in the CID and CSD registers. You can customize the component further by modifying the MMC model source code directly.

Table 4-80 MMC parameters

Name	Type	Allowed values	Default value	Description
`card_type`	`string`	'SD', 'eMMC'	'SD'	The card is eMMC (MMCA v4.4) or SD (v3.31).
`force_sector_addressing`	`bool`	`true`, `false`	`false`	Use sector addressing not byte addressing even if `p_mmc_file` is a file <2GB in size.
`p_diagnostics`	`int`	0-4	0	Level of diagnostics printed to `stdout` on MMC accesses: 0 none, 4 many.
`p_fast_access`	`bool`	`true`, `false`	`true`	Complete MMC-card block reads and writes as fast as possible, or simulate a delay.
`p_mmc_file`	`string`	Valid filename	'mmc.dat'	File for the MMC component backing store.
`p_prodName`	`string`	Six character string	'ARMmmc'	Card ID product name.
`p_prodRev`	`int`	-	`0x1`	Card ID product revision.
`p_manid`	`int`	-	`0x2`[by]	Card ID manufacturer ID.
`p_OEMid`	`int`	-	`0x0000`	Card ID OEM ID.
`p_sernum`	`int`	-	`0xCA4D0001`	Card serial number.

Related references

MMC - registers on page 4-332.

MMC - registers

This section describes the registers.

The registers are not memory mapped. Instead, you access them 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.

For a full definition of MMC registers, see the MMCA System Summary documentation. Device-specific register information can also be obtained from MMC vendors.

Table 4-81 MMC registers

Name	CADI register number	Description
OCR_REG	`0x000`	Operating conditions register
CID_REG0	`0x004`	Card ID bits 127:96
CID_REG1	`0x005`	Card ID bits 95:64

by This is equivalent to ‘Sandisk’.

Table 4-81 MMC registers (continued)

Name	CADI register number	Description
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

MMC - debug features

This component provides a CADI interface for viewing internal registers.

Use the `p_diagnostics` parameter to select the level of diagnostic output, to help to debug device driver and controller-to-card protocol issues:

Level 0
None.

Level 1
Warnings about attempting to change read-only settings.

Level 2
Trace of command calls.

Level 3
Information about every step in the MMC_Protocol interaction.

Level 4
Hex dump of every block sent or received.

See the source code for details.

Related references

MMC - registers on page 4-332.

MMC - verification and testing

This component passes tests in conjunction with the ARM MMC reference model, and in the VE example with Boot Monitor and Linux drivers.

4.4.29 MMU_400 component

This section describes the MMU_400 component.

MMU_400 - about

This LISA+ component is a model of r0p1 of the ARM CoreLink MMU-400 System Memory Management Unit.

Related references

4.5.3 PVBus Transaction Master ID on page 4-401.
4.5.11 Labeller and LabellerForDMA330 components on page 4-407.

MMU_400 - transaction labels

Incoming transactions need labels, made up of a `StreamId` and an `SSD_Index` or `SSD` (*Security State Determinator*).

The `SSD_Index`/`SSD`, `StreamId` pair maps a transaction to a security world (Secure or Non-secure) and a translation context that holds the information necessary to perform the translation. Transactions with the same label appear the same.

You use `SSD_Index` to indirectly find the security world of the transaction using a mapping table (SMMU_SSDRn) in the SMMU, while `SSD` directly encodes the security state of the transaction.

Note

The security state of the transaction is independent of the `Normal/Secure` attribute of the transaction.

In the case of the MMU-400, the secure transactions never go to a translation context and instead can either be faulted, or bypass the address translation and have a limited attribute-only transformation applied.

In a hardware *System on a Chip* (SoC), the `StreamId` and the `SSD_Index` are usually formed by side-band signals on the bus or by AXI routing IDs, for example. Models do not have direct parallels for these signals. Instead, you must label transactions coming from a device.

The MMU_400 component uses the top 16 bits of the 32 bit `master_id` as a label. It has parameters such as:

```
label0_read_stream_id label0_read_ssd label0_write_stream_id label0_write_ssd ...
```

These map a label *`N`* (range 0..31) in the top 16 bits of the `master_id` to a particular `StreamId` and `SSD_Index`/`SSD`.

Note

The read and write transactions can have different `StreamId`s and `SSD_Index`/`SSD`s.

If MMU_400 uses `SSD_Index`es, then the parameter `label{`*`N`*`}_{`*`read/write`*`}_ssd` is the index. Otherwise it is 0 or 1, for Secure and Non-secure, respectively.

Transactions have a `master_id` field, set by a master. Many masters simply use zero, and these often do not allow you to change it. A simple Labeller component exists to put the labels in the `master_id`. Place it under each upstream device or bus-segment that the SoC wishes to distinguish. The LISA code shows you how it works, so you can create alternative labeling components.

The MMU_400 component is a wrapper round the component MMU_400_BASE. If the SoC identifies transactions in an alternative way, then you can write a new wrapper for MMU_400_BASE and implement your own identification scheme. The LISA code shows you how it works. In particular, you must replace the identify behavior with one specific to the SoC.

MMU_400 - ports

This section describes the ports.

Table 4-82 MMU_400 ports

Name	Protocol	Type	Description
reset_in	Signal	Slave	Signal to reset the MMU.
pvbus_s	PVBus	Slave	Upstream port of the MMU. Addresses on the port are in VA/IPA.
apb3_control_ns	PVBus	Slave	APBv3 control port for Non-secure access to the register file. If this port is used do not use the APBv4 port.
apb3_control_s	PVBus	Slave	APBv3 control port for Secure access to the register file. If this port is used do not use the APBv4 port.
apb4_control	PVBus	Slave	APBv4 control port for access to the register file. If this port is used do not use the APBv3 ports.
cfg_cttw_in	Signal	Slave	Enables coherent page table walks.
pvbus_m	PVBus	Master	Downstream port of the MMU, where translated transactions emerge.
pvbs_ptw_m	PVBus	Master	Downstream port for page table walks if configured using the ptw_has_separate_port parameter.
cfg_flt_irpt_ns	Signal	Master	Non-secure configuration access fault interrupt. Corresponds to SMMU architectural signal SMMU_NSgCfgIrpt.
cfg_flt_irpt_s	Signal	Master	Secure configuration access fault interrupt. Corresponds to SMMU architectural signal SMMU_gCfgIrpt.
cxt_irpt_ns	Signal	Master	Non-secure context bank fault.
glbl_flt_irpt_ns	Signal	Master	Global Non-secure fault interrupt. Corresponds to SMMU architectural signal SMMU_NSgIrpt.
glbl_flt_irpt_s	Signal	Master	Global Secure fault interrupt. Corresponds to SMMU architectural signal SMMU_gIrpt.
comb_irpt_ns	Signal	Master	Non-secure combined interrupt.
comb_irpt_s	Signal	Master	Secure combined interrupt.

MMU_400 - parameters

This section describes the parameters.

Table 4-83 MMU_400 parameters

Name	Type	Allowed values	Default value	Description
always_secure_ssd_indices	string	-	""	Non-programmable SSD_Indexes that are always Secure, for example 0, 6, 35-64.
cfg_cttw	bool	true, false	true	Perform coherent page table walks.
label*N*_read_ssd	int	0-65535	0	Label *N*: read SSD or SSD_Index.
label*N*_read_stream_id	int	0-65535	0	Label *N*: read StreamID.
label*N*_write_ssd	int	0-65535	0	Label *N*: write SSD or SSD_Index.
label*N*_write_stream_id	int	0-65535	0	Label *N*: write StreamID.
number_of_contexts	int	1-8	8	Number of context banks.
number_of_smrs	int	2-32	32	Number of stream match registers.

Table 4-83 MMU_400 parameters (continued)

Name	Type	Allowed values	Default value	Description
percent_tlbstatus_commits	int	0-100	10	Percentage of time that a poll of TLBSTATUS commits the TLBI commands.
programmable_non_secure_by_default_ssd_indices	string	-	""	Programmable SSD_Indexes that are Non-secure by default, for example 0, 6, 35-84.
programmable_secure_by_default_ssd_indices	string	-	""	Programmable SSD_Indexes that are Secure by default, for example 0, 6, 35-84.
ptw_has_separate_port	bool	true, false	true	Page table walks use the pvbus_ptw_m port.
pvbus_m_is_ace_lite	bool	true, false	true	Downstream port pvbus_m is ACE-Lite.
stream_id_width	int	0-15	6	StreamID bit width.
tlb_depth	int	0-2048	64	TLB depth. Zero means infinite.
use_label_mapping	bool	true, false	true	Configure derivation of StreamIDs and SSD_Indexes from transaction attributes. true Treat the top 16 bits of transaction MasterID attributes as a label number that is mapped to a StreamID and SSD_Index using the labelN parameters. The MMU_400 does not use the bottom 16 bits. false Treat the top 16 bits of the MasterID attributes as StreamID and the bottom 16 bits encode either the SSD_Index or the SSD state directly according to the use_ssd_determination_table parameter.
use_ssd_determination_table	bool	true, false	true	true Encode SSD data as SSD_Index. false Treat SSD data as SSD state directly, where zero is Secure and nonzero is Non-secure.

MMU_400 - registers

This section describes the registers.

This component models all architectural registers as the *Technical Reference Manual* (TRM) specifies, except for performance registers. It does not model any of the performance registers.

MMU-400 does not have an SMMU_STLBGSTATUS register because the Secure side is a nominal pass-through. MMU-400 only has stage 2 support and you cannot use stage 2 on the Secure side.

The SMMU_NSACR is an alias of the Non-secure SMMU_ACR. This component models SMMU_ACR as RAZ/WI.

The *ACR registers have IMP DEF contents. This component models only the PAGESIZE bit of the SACR, as non-RAZ/WI. It models no other IMP DEF registers.

MMU_400 - debug features

This component exports a CADI debug interface.

MMU_400 - verification and testing

This component is functionally complete and passes unit tests.

MMU_400 - performance

This component has little effect on system performance.

Just as for the hardware, TLB misses affect performance. To reduce this, increase the TLB size with the `tlb_depth` parameter.

4.4.30 MMU_500 component

This section describes the MMU_500 component.

MMU_500 - about

This LISA+ component is a model of a basic MMU-500. Set the version with the `version` parameter.

You cannot arbitrarily configure how you derive StreamIDs and SSD_Indexes from the transaction attributes.

This component has two label modes: you select which one with the parameter `use_label_mapping`.

MMU_500 - ports

This section describes the ports.

Table 4-84 MMU_500 ports

Name	Protocol	Type	Description
`cfg_cttw_in`	Signal	Slave	Enables coherent page table walks.
`cxt_irpt[128]`	Signal	Master	Context interrupt.
`comb_irpt_ns`	Signal	Master	Non-secure combined interrupt.
`comb_irpt_s`	Signal	Master	Secure combined interrupt.
`glbl_flt_irpt_ns`	Signal	Master	Global Non-secure fault interrupt.
`glbl_flt_irpt_s`	Signal	Master	Global Secure fault interrupt.
`pvbus_control_s`	PVBus	Slave	Provides memory-mapped read write access to the control registers of the module.
`pvbus_m[32]`	PVBus	Master	For all memory accesses. One for each *Translation Buffer Unit* (TBU).
`pvbus_ptw_m`	PVBus	Master	If `ptw_has_separate_port` is `true`, use for page table walks.
`pvbus_s[32]`	PVBus	Slave	For transactions from PVBus master/decoder. One for each TBU.
`reset_in`	Signal	Slave	Reset signal.

MMU_500 - parameters

This section describes the parameters.

Table 4-85 MMU_500 parameters

Name	Type	Allowed values	Default value	Description
always_secure_ssd_indices	string	-	""	Non-programmable *Security State Determination* (SSD) indexes that are always Secure, for example 0, 6, 35-64.
cfg_cttw	bool	true, false	true	Perform coherent page table walks.
dump_unpredictablity_in_user_flags	bool	true, false	false	Override the user flags to encode unpredictable information (validation only).
number_of_contexts	unsigned	1-128	8	Number of context banks.
number_of_smrs	unsigned	2-128	32	Number of stream match registers.
percent_tlbstatus_commits	uint32_t	1-100	10	Percentage of times that a poll of TLBSTATUS will commit the TLBI commands.
programmable_non_secure_by_default_ssd_indices	string	-	""	Programmable SSD indexes that are by default Non-secure, for example 0, 6, 35-84.
programmable_secure_by_default_ssd_indices	string	-	""	Programmable SSD indexes that are by default Secure, for example 0, 6, 35-84.
ptw_has_separate_port	bool	true, false	true	Page table walks use pvbus_ptw_m.
pvbus_ptw_m_is_ace_lite	bool	true, false	true	pvbus_ptw_m port is ACE-Lite.
supports_nested_translations	bool	true, false	true	Supports nested translations (stage 1 + stage 2).
tlb_depth	unsigned	0-2048	2048	TLB depth. 0 means 'infinite'.
use_label_mapping	bool	true, false	true	Use label mapping.[bz]

[bz] Use `true` if your upstream devices have labels in the top 16 bits of the transaction MasterID. Note that the model does not have a concept of AXI-ID, but a transaction can have a MasterID set on it. Label your upstream components 0-*N* so that the parameters of this component can map those integers to StreamID and SSD_Index. Use `false` if the StreamID is in the top 16 bits of the MasterID and the bottom 16 bits encode either the SSD_Index or the SSD state directly, depending on `use_ssd_determination_table`. Typically in hardware, a device emits different AXI-IDs, determined by what it is doing. In the model, MasterIDs are usually not diverse and a device might only emit one MasterID.

Table 4-85 MMU_500 parameters (continued)

Name	Type	Allowed values	Default value	Description
`use_ssd_determination_table`	`bool`	`true`, `false`	`true`	Use the SSD determination table.[ca]
`version`	`string`	-	""	Version of the RTL that the model represents. Valid values are `LACr1` and `EAC`. The default is empty, for which the simulation fails on startup.

MMU_500 identify parameters

These parameters map a 'label' of a transaction to a StreamID and either an SSD_Index or a *Security State Determination* (SSD).

Note

The `identify` function uses these parameters. You can implement your own `identify` function.

Table 4-86 MMU_500 identify parameters

Name	Type	Allowed values	Default value	Description
`label`*N*`_read_ssd`	`unsigned`	`0-65535`	`0`	Label*N*: read SDD or SSD_Index. 0 <= *N* <= 31.
`label`*N*`_read_stream_id`	`unsigned`	`0-65535`	`0`	Label*N*: read StreamID. 0 <= *N* <= 31.
`label`*N*`_write_ssd`	`unsigned`	`0-65535`	`0`	Label*N*: write SDD or SSD_Index. 0 <= *N* <= 31.
`label`*N*`_write_stream_id`	`unsigned`	`0-65535`	`0`	Label*N*: write StreamID. 0 <= *N* <= 31.
`use_label_mapping`	`bool`	`true`, `false`	`true`	Use label mapping.

MMU_500 - registers

This section describes the registers.

This component models all architectural registers as the *Technical Reference Manual* (TRM) specifies, except for performance registers. It does not model any of the performance registers.

Unlike the MMU-400, MMU-500 does have an SMMU_STLBGSTATUS register because it has stage 1 and stage 2 support.

The SMMU_NSACR is an alias of the Non-secure SMMU_ACR. This component models SMMU_ACR as RAZ/WI.

The *ACR registers have IMP DEF contents. This component models only the PAGESIZE bit of the SACR, as non-RAZ/WI. It models no other IMP DEF registers.

4.4.31 OrGate component

This section describes the OrGate component.

OrGate - about

This 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.

[ca] If `true`, the bottom 16 bits of the MasterID encode the SSD_Index. They must be < 2^`ssd_index_width`. If `false`, they encode the SSD state directly (zero is Secure and nonzero is Non-secure).

This is a LISA+ component.

OrGate - ports

This section describes the ports.

Table 4-87 OrGate ports

Name	Protocol	Type	Description
`input[0]`	Signal	Slave	First input signal
`input[1]`	Signal	Slave	Second input signal
`output`	Signal	Master	Combined output signal

OrGate - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.32 PL011_Uart component

This section describes the PL011_Uart component.

PL011_Uart - about

This LISA+ component is a model of r1p4 of a PL011_UART PrimeCell component.

This component does not implement the DMA functionality of the PL011 PrimeCell.

Related concepts

TelnetTerminal - about on page 4-390.

PL011_Uart - ports

This section describes the ports.

Table 4-88 PL011_Uart ports

Name	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

Related references

2.4.16 SerialData protocol on page 2-61.

PL011_Uart - parameters

This section describes the parameters.

Table 4-89 PL011_Uart parameters

Name	Type	Allowed values	Default value	Description
baud_rate	int	-	38400	Baud rate.
clock_rate	int	-	14745600, 0xE10000	Clock rate for PL011.
in_file	string	-	""	Input file.
in_file_escape_sequence	string	-	##	Input file escape sequence.
out_file	string	-	""	Output file.
revision	string	-	'r1p4'	Revision to simulate.
shutdown_on_eot	bool	true, false	false	Shut down simulation on receipt of an EOT (ASCII 4) character. This is useful for regression tests when semihosting is not available.
shutdown_tag	string	-	""	Shut down simulation on receipt of this string.
uart_enable	bool	true, false	false	Enable UART when the system starts.
unbuffered_output	bool	true, false	false	Unbuffered output.
untimed_fifos[cb]	bool	true, false	true	Ignore the clock rate and allow immediate serial data transfer between the Tx/Rx FIFOs of the UART and the SerialData port.

PL011_Uart - registers

This section describes the registers.

Table 4-90 PL011_Uart registers

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[cc]
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

[cb] 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.

[cc] Not implemented.

Table 4-90 PL011_Uart registers (continued)

Name	Offset	Access	Description
UARTMIS	0x40	Read only	Masked interrupt register
UARTICR	0x44	Write only	Interrupt clear register
UARTDMACR	0x48	Read/write	DMA control register[cc]

PL011_Uart - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PL011_Uart - performance

ARM expects this component to have little effect on the performance of PV systems. However, at very high baud rates such as in excess of 1MHz, simulation performance might be reduced.

4.4.33 PL022_SSP component

This section describes the PL022_SSP component.

PL022_SSP - about

This LISA+ component is a model of an 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.

Note

This component is a preliminary release. It is provided as-is with the VE reference platform model, and is not a fully supported peripheral.

PL022_SSP - ports

This section describes the ports.

Table 4-91 PL022_SSP ports

Name	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
clkout	ClockSignal	Master	Clock output
intr	Signal	Master	Interrupt signaling
rorintr	Signal	Master	Receive overrun interrupt
rtintr	Signal	Master	Receive timeout interrupt[cd]
rx_dma_port	PL080_DMAC_DmaPortProtocol	Master	PrimeCell SSP receive DMA port
rxintr	Signal	Master	Receive FIFO service request port

[cd] Not supported.

Table 4-91 PL022_SSP ports (continued)

Name	Protocol	Type	Description
tx_dma_port	PL080_DMAC_DmaPortProtocol	Master	PrimeCell SSP transmit DMA port
txd	ValueState	Master	PrimeCell SSP transmit data
txintr	Signal	Master	Transmit FIFO service request

Related references

2.4.11 PL080_DMAC_DmaPortProtocol protocol on page 2-58.

PL022_SSP - registers

This section describes the registers.

Table 4-92 PL022_SSP registers

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]
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]

PL022_SSP - verification and testing

The functions of this component have been tested individually by 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.

4.4.34 PL030_RTC component

This section describes the PL030_RTC component.

PL030_RTC - about

This LISA+ component is a model of a PL030_RTC PrimeCell.

PL030_RTC - ports

This section describes the ports.

Table 4-93 PL030_RTC ports

Name	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

PL030_RTC - registers

This section describes the ports.

Table 4-94 PL030_RTC registers

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`	Rad/write	Counter register

PL030_RTC - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PL030_RTC - performance

This 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.

4.4.35 PL031_RTC component

This section describes the PL031_RTC component.

PL031_RTC - about

This LISA+ component is a model of a PL031 Real Time Clock.

It can provide a basic alarm function or long time base counter.

PL031_RTC - ports

This section describes the ports.

Table 4-95 PL031_RTC ports

Name	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

PL031_RTC - registers

This section describes the registers.

Table 4-96 PL031_RTC registers

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[ce]
RTCPeriphID0	0xFE0	Read only	Peripheral ID register[ce]
RTCPeriphID1	0xFE4	Read only	Peripheral ID register[ce]
RTCPeriphID2	0xFE8	Read only	Peripheral ID register[ce]
RTCPeriphID3	0xFEC	Read only	Peripheral ID register[ce]
RTCPCellID0	0xFF0	Read only	PrimeCell ID register[ce]
RTCPCellID1	0xFF4	Read only	PrimeCell ID register[ce]
RTCPCellID2	0xFF8	Read only	PrimeCell ID register[ce]
RTCPCellID3	0xFFC	Read only	PrimeCell ID register[ce]

PL031_RTC - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems. This component passes tests as part of the SMLT component.

PL031_RTC - performance

This 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.

4.4.36 PL041_AACI component

This section describes the PL041_AACI component.

[ce] This register has no CADI interface.

PL041_AACI - about

This LISA+ component is a model of a 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.

This component is not a complete implementation of the AACI because the following functionality is not 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.

Related references

4.4.2 AudioOut_File component on page 4-287.
4.4.3 AudioOut_SDL component on page 4-288.

PL041_AACI - ports

This section describes the ports.

Table 4-97 PL041_AACI ports

Name	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

Related references

2.4.1 AudioControl protocol on page 2-55.

PL041_AACI - registers

This section describes the registers.

Table 4-98 PL041_AACI registers

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

Table 4-98 PL041_AACI registers (continued)

Name	Offset	Access	Description
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
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

PL041_AACI - verification and testing

This component passes tests using the ALSA driver for this component under Linux.

PL041_AACI - performance

This 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.

4.4.37 PL050_KMI component

This section describes the PL050_KMI component.

PL050_KMI - about

This LISA+ component is a model of a PL050 Keyboard/Mouse Interface PrimeCell. It communicates with models of PS/2-like devices such as a PS2Keyboard or PS2Mouse.

PL050_KMI - ports

This section describes the ports.

Table 4-99 PL050_KMI ports

Name	Protocol	Type	Description
`pvbus`	PVBus	Slave	Slave port for connection to PV bus master/decoder
`intr`	Signal	Master	Master port signaling 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

Related references

2.4.12 PS2Data protocol on page 2-59.

PL050_KMI - registers

This section describes the registers.

Table 4-100 PL050_KMI registers

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

PL050_KMI - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PL050_KMI - performance

This 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 depends on that of the visualization.

4.4.38 PL061_GPIO component

This section describes the *General Purpose Input/Output* (GPIO) component.

PL061_GPIO - about

This LISA+ component is a model of an 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.

PL061_GPIO - ports

This section describes the ports.

Table 4-101 PL061_GPIO ports

Name	Protocol	Type	Description
`pvbus`	PVBus	Slave	Slave port for connection to PV bus master/decoder
`GPIO_In`	Value	Slave	Input lines[cf]
`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[cf]
`GPIO_Out`	Value	Master	Output lines[cf]

PL061_GPIO - registers

This section describes the registers.

Table 4-102 PL061_GPIO registers

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.[cg]
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.

[cf] Only the lower end eight bits[7:0] are used.
[cg] 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.

Table 4-102 PL061_GPIO registers (continued)

Name	Offset	Access	Description
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.
GPIOPCellID1	0xff4	Read	PrimeCell ID register.
GPIOPCellID2	0xff8	Read	PrimeCell ID register.
GPIOPCellID3	0xffc	Read	PrimeCell ID register.

PL061_GPIO - verification and testing

The functions of this component have been tested individually using a tailored test suite.

4.4.39 PL080_DMAC component

This section describes the PL080_DMAC component.

PL080_DMAC - about

This LISA+ component is a 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.

PL080_DMAC - ports

This section describes the ports.

Table 4-103 PL080_DMAC ports

Name	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	Slave	Peripheral handshake ports

Related references

2.4.11 PL080_DMAC_DmaPortProtocol protocol on page 2-58.

PL080_DMAC - parameters

This section describes the parameters.

Table 4-104 PL080_DMAC parameters

Name	Type	Allowed values	Default value	Description
`fifo_size`	Integer	0-1024	16	Controls the size of channel FIFOs in bytes
`max_transfer`	Integer	1-1024	256	Limits the number of transfers that can be made atomically
`generate_clear`	Boolean	`true`, `false`	`false`	Controls whether completion of a burst/single transfer generates a clear response to peripherals
`activate_delay`	Integer	0-256	0	Sets the minimum number of cycles after a request or channel enable

PL080_DMAC - registers

This section describes the registers.

Table 4-105 PL080_DMAC registers

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
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

Table 4-105 PL080_DMAC registers (continued)

Name	Offset	Access	Description
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
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

Table 4-105 PL080_DMAC registers (continued)

Name	Offset	Access	Description
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

PL080_DMAC - verification and testing

The functions of this component have been tested individually using a tailored test suite.

PL080_DMAC - performance

This 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.

4.4.40 PL110_CLCD component

This section describes the PL110_CLCD component.

PL110_CLCD - about

This LISA+ component is a model of the PL110 *Color LCD* (CLCD) controller PrimeCell.

You can connect the model through a framebuffer port to, for instance, a visualization component, so that LCD output can be viewed.

The implementation provides a register model of the LCD controller, and both timing and bus utilization models favor efficiency of implementation and model speed rather than accuracy.

PL110_CLCD - ports

This section describes the ports.

Table 4-106 PL110_CLCD ports

Name	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
control	Value	Slave	Auxiliary control register 1
pvbus_m	PVBus	Master	DMA port for video data

Related references

4.6 Visualisation Library on page 4-411.

PL110_CLCD - parameters

This section describes the parameters.

Table 4-107 PL110_CLCD parameters

Name	Type	Allowed values	Default value	Description
pixel_double_limit	Integer	-	300	Sets a threshold in horizontal pixels, below which pixels sent to the framebuffer are doubled in size horizontally and vertically

PL110_CLCD - registers

This section describes the registers.

Table 4-108 PL110_CLCD registers

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
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

PL110_CLCD - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PL110_CLCD - performance

This 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.

4.4.41 PL111_CLCD component

This section describes the PL111_CLCD component.

PL111_CLCD - about

This LISA+ component is a model of the PL110 CLCD, and also implements the hardware cursor of the PL111_CLCD (*ARM PrimeCell Color LCD Controller* (PL111)), which is the major change compared with PL110.

PL111_CLCD - ports

This section describes the ports.

Table 4-109 PL111_CLCD ports

Name	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
`control`	Value	Slave	Auxiliary control register 1
`pvbus_m`	PVBus	Master	DMA port for video data

Related references

4.6 Visualisation Library on page 4-411.

PL111_CLCD - parameters

This section describes the parameters.

Table 4-110 PL111_CLCD parameters

Name	Type	Allowed values	Default value	Description
`pixel_double_limit`	`int`	-	300, `0x12C`	Threshold in horizontal pixels, below which pixels sent to the framebuffer are doubled in size horizontally and vertically.

PL111_CLCD - registers

This section describes the registers.

Table 4-111 PL111_CLCD registers

Name	Offset	Access	Description
LCDTiming0	`0x0`	Read/write	Horizontal timing
LCDTiming1	`0x4`	Read/write	Vertical timing
LCDTiming2	`0x8`	Read/write	Clock and polarity control

Table 4-111 PL111_CLCD registers (continued)

Name	Offset	Access	Description
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
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

PL111_CLCD - verification and testing

This component passes tests as part of the PL111 test system using PL11x test suites.

4.4.42 PL180_MCI component

This section describes the PL180_MCI component.

PL180_MCI - about

This LISA+ component is a model of the PL180 *Multimedia Card Interface* (MCI).

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:

- 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.
- 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.

Related references

4.4.28 MMC component on page 4-330.

PL180_MCI - ports

This section describes the ports.

Table 4-112 PL180_MCI ports

Name	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 *MultiMediaCard* (MMC) master port

Related references

2.4.9 MMC_Protocol protocol on page 2-57.

PL180_MCI - registers

This section describes the registers.

Table 4-113 PL180_MCI registers

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 register
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

Table 4-113 PL180_MCI registers (continued)

Name	Offset	Access	Description
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
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]

PL180_MCI - debug features

At compile time, you can enable command tracing 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.

PL180_MCI - verification and testing

This component passes tests in conjunction with the ARM MMC reference model, and in the VE example with Boot Monitor and Linux drivers.

4.4.43 PL192_VIC component

This section describes the PL192_VIC component.

PL192_VIC - about

This LISA+ component is a model of an ARM PrimeCell Vectored Interrupt Controller (PL192). It aggregates interrupts and generates interrupt signals for 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.

PL192_VIC - ports

This section describes the ports.

Table 4-114 PL192_VIC ports

Name	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

PL192_VIC - registers

This section describes the registers.

Table 4-115 PL192_VIC registers

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

PL192_VIC - verification and testing

This component has been run against the RTL validation suite and has been successfully used in validation platforms.

4.4.44 PL310_L2CC component

This section describes the PL310_L2CC component.

PL310_L2CC - about

This LISA+ component is a model of an ARM PrimeCell Level 2 Cache Controller (PL310).

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 that is closely coupled or internal to the processor.

This 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 `cache-state_modelled` parameter controls the mode of operation.

ARM supports the use of the PL310 when connected to the ARM Cortex-A5 or Cortex-A9 processor.

Related references

PL310_L2CC - parameters on page 4-360.
1.4.1 Caches in PV models on page 1-16.

PL310_L2CC - ports

This section describes the ports.

Table 4-116 PL310_L2CC ports

Name	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

PL310_L2CC - parameters

This section describes the parameters.

Table 4-117 PL310_L2CC parameters

Name	Type	Allowed values	Default value	Description
`ASSOCIATIVITY`	Integer	0 (8-way), 1 (16-way)	0	Associativity for auxiliary control register
`CACHEID`	Integer	0-63	0	Cache controller cache ID

Table 4-117 PL310_L2CC parameters (continued)

Name	Type	Allowed values	Default value	Description
cache-state_modelled[ch]	Boolean	true, false	false	Specifies whether real cache state is modeled (vs. register model)
CFGBIGEND	Integer	0, 1	0	Big-endian mode for accessing configuration registers out of reset
LOCKDOWN_BY_LINE[ci]	Integer	0, 1	0	Lockdown by line
LOCKDOWN_BY_MASTER[cj]	Integer	0, 1	0	Lockdown by master
REGFILEBASE	Integer	0-0xfffff000	0x1f002000	Base address for accessing configuration registers
WAYSIZE	Integer	0-7	1	Size of ways for auxiliary control register

Related references

PL310_L2CC - performance on page 4-363.

PL310_L2CC - registers

This section describes the registers.

Table 4-118 PL310_L2CC registers

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
TagLatencyCtrl[ck]	0x108	Read/write	r1 tag RAM latency control
DataLatencyCtrl[ck]	0x10C	Write only	r1 data RAM latency control
EventCounterCtrl[ck]	0x200	Read/write	r2 event counter control
EventCounter1CfgTopic[ck]	0x204	Write only	r2 event counter 1 configuration
EventCounter0Cfg[ck]	0x208	Read/write	r2 event counter0 configuration
EventCounter1[ck]	0x20C	Read/write	r2 event counter 1 value
EventCounter0[ck]	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[ck]	0x730	Read/write	r7 cache sync
InvalidateByPA	0x770	Read/write	r7 invalidate line by PA
InvalidateByWay	0x77C	Read/write	r7 invalidate by way

[ch] Enabling this parameter might affect performance.
[ci] Value is reflected in CacheType register bit 25, but the feature is not switched off when the parameter is 0.
[cj] Value is reflected in CacheType register bit 26, but the feature is not switched off when the parameter is 0.
[ck] Operation of this register is not functionally modeled.

Table 4-118 PL310_L2CC registers (continued)

Name	Offset	Access	Description
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
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[ck]	0xC00	Read/write	r12 address filtering start
AFilterEnd[ck]	0xC04	Read/write	r12 address filtering end
DebugControl[ck]	0xF40	Read/write	r15 debug control register

PL310_L2CC - debug features

This component exports the PL310 registers by CADI.

PL310_L2CC - verification and testing

This component 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 been used in validation platforms.

PL310_L2CC - performance

The performance of this component depends on the configuration of the associated L1 caches and the mode it is in.

- Register mode: no significant affect.
- 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: there is a significant impact on system performance.

PL310_L2CC - functionality

This component implements the programmer visible functionality of the PL310, and excludes some non-programmer visible features.

PL310_L2CC - hardware features present

This component reproduces many features of the hardware.

- 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.

PL310_L2CC - hardware features absent

This component does not model some features of the hardware. 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.

PL310_L2CC - hardware features different

This component handles some features differently 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.

One feature is additional.

- Data RAM and Tag RAM are embedded to the model.

4.4.45 PL330_DMAC component

This section describes the PL330_DMAC component.

PL330_DMAC - about

This LISA+ component is a model of the ARM PrimeCell Dynamic Memory Access Controller (PL330).

The model uses 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.

PL330_DMAC - transaction labels

Each transaction carries the identity of the requesting thread.

This controller has up to eight channel threads and a manager thread. Each has an ID. In the hardware, the ID is AxID[3:0], with `0x0 - (number of channels – 1)` identifying channels and `(number of channels)` identifying the manager: for example, `0x0-0x7` and `0x8`, respectively. The manager originates only instruction fetches, and the manager ID is also used for instruction fetches issued by the channels.

In the model, the identity of the requesting thread is encoded into each transaction using the low-order 16 bits of the Master ID field:

- Channel data: `0-7`.
- Channel instruction fetch: `0xffff`.
- Manager instruction fetch: `0xffff`.

If a downstream component needs to know the IDs of bus masters that use either the low-order 16 bits or the label, use the label. The LabellerForDMA330 component shifts the low-order 16 bits into the label, while providing a degree of control over the label encoding. The example below maintains separate IDs for each data channel while using the correct hardware ID to identify instruction fetch for a DMA-330 with 8 channels:

```
pl330_dma : PL330_DMAC( "p_max_channels" = 8 );
dma_labeller : LabellerForDMA330(
    "dma330_discriminate_data_channels" = true,
    "dma330_s_instruction_label" = 8,
    "dma330_ns_instruction_label" = 8 );
pl330_dma.pvbus_m => dma_labeller.pvbus_s;
dma_labeller.pvbus_m => output_bus.pvbus_s;
```

Related references

4.5.3 PVBus Transaction Master ID on page 4-401.

4.5.11 Labeller and LabellerForDMA330 components on page 4-407.

PL330_DMAC - ports

This section describes the ports.

Table 4-119 PL330_DMAC ports

Name	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

PL330_DMAC - parameters

This section describes the parameters.

Table 4-120 PL330_DMAC parameters

Name	Type	Allowed values	Default value	Description
`p_max_irqs`	Integer	0-32	32	Number of interrupts
`p_max_channels`	Integer	≤8	8	Virtual channels
`p_controller_nsecure`	Boolean	`true`, `false`	`false`	Controller non-secure at reset
`p_controller_boots`	Boolean	`true`, `false`	`true`	DMA boots from reset
`p_reset_pc`	Integer	Any valid address	`0x60000000`	DMA PC at reset

PL330_DMAC - registers

This section describes the registers.

Table 4-121 PL330_DMAC registers

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
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 4-121 PL330_DMAC registers (continued)

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
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

Table 4-121 PL330_DMAC registers (continued)

Name	Offset	Access	Description
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	Read only	Debug command register
DBGINST0	0xD08	Read only	Debug instruction-0 register
DBGINST1	0xD0C	Read only	Debug instruction-1 register
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

PL330_DMAC - verification and testing

The functions of this component have been tested individually using a tailored test suite.

4.4.46 PL340_DMC component

This section describes the PL340_DMC component.

PL340_DMC - about

This LISA+ component is a model of the ARM PrimeCell Dynamic Memory Controller (PL340).

It provides an interface for up to four DRAM chips. The implementation also provides an apb interface to configure the controller behavior.

PL340_DMC - ports

This section describes the ports.

Table 4-122 PL340_DMC ports

Name	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

PL340_DMC - parameters

This section describes the parameters.

Table 4-123 PL340_DMC parameters

Name	Type	Allowed values	Default value	Description
IF_CHIP_0 to IF_CHIP_3	int	-1, 0	-1	**-1** No memory connected to the interface. **0** DRAM connected.
MEMORY_WIDTH	int	16, 32, 64	32	Indicates the width, in bits, of connected memory

PL340_DMC - registers

This section describes the registers.

You can access the registers through the APB interface.

Table 4-124 PL340_DMC registers

Name	Offset	Access	Description
memc_status	0x000	Read only	Memory controller status register.
memc_cmd	0x004	Write only	Modify the state machine of the controller.
direct_cmd	0x008	Write only	Set the memory controller configurations.
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	Set the QOS.
id_01_cfg	0x104	Read/write	Set the QOS.
id_02_cfg	0x108	Read/write	Set the QOS.
id_03_cfg	0x10C	Read/write	Set the QOS.
id_04_cfg	0x110	Read/write	Set the QOS.
id_05_cfg	0x114	Read/write	Set the QOS.

Table 4-124 PL340_DMC registers (continued)

Name	Offset	Access	Description
id_06_cfg	0x118	Read/write	Set the QOS.
id_07_cfg	0x11C	Read/write	Set the QOS.
id_08_cfg	0x120	Read/write	Set the QOS.
id_09_cfg	0x124	Read/write	Set the QOS.
id_10_cfg	0x128	Read/write	Set the QOS.
id_11_cfg	0x12C	Read/write	Set the QOS.
id_12_cfg	0x130	Read/write	Set the QOS.
id_13_cfg	0x134	Read/write	Set the QOS.
id_14_cfg	0x138	Read/write	Set the QOS.
id_15_cfg	0x13C	Read/write	Set 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[cl]
periph_id_1	0xFE4	Read only	Peripheral ID register 1[cl]
periph_id_2	0xFE8	Read only	Peripheral ID register 2[cl]
periph_id_3	0xFEC	Read only	Peripheral ID register 3[cl]
pcell_id_0	0xFF0	Read only	PrimeCell ID register 0[cl]
pcell_id_1	0xFF4	Read only	PrimeCell ID register 1[cl]
pcell_id_2	0xFF8	Read only	PrimeCell ID register 2[cl]
pcell_id_3	0xFFC	Read only	PrimeCell ID register 3[cl]

PL340_DMC - verification and testing

The PL340_DMC functions of the component have been tested individually using a tailored test suite.

4.4.47 PL350_SMC component

This section describes the PL350_SMC component.

PL350_SMC - about

This LISA+ component is a model of the ARM PrimeCell Static Memory Controller (PL350).

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.

[cl] This register has no CADI interface.

This component provides a PVBus slave to control the device behavior. A remap port is also provided to assist in remapping particular memory regions.

PL350_SMC - ports

This section describes the ports.

Table 4-125 PL350_SMC ports

Name	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
`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

PL350_SMC - parameters

This section describes the parameters.

Table 4-126 PL350_SMC parameters

Name	Type	Allowed values	Default value	Description
`IF0_MEM_TYPE_PARAMETER` `IF1_MEM_TYPE_PARAMETER`	Integer	0, 1	0	Memory type for interfaces 0 and 1: **0** SRAM. **1** NAND.
`REMAP`	Integer	-1, 0-7	-1	If an interface is remapped: **-1** Remap not enabled. **0 to 7** Device that gets address `0x0`.

`IF0_CHIP_0` to `IF0_CHIP_3`
`IF1_CHIP_0` to `IF1_CHIP_3`
	Boolean	`true`, `false`	`false`	Memory connected to chip slots for interfaces 0 and 1:
				`true`
				Nothing connected.
				`false`
				Memory connected.

Table 4-126 PL350_SMC parameters (continued)

Name	Type	Allowed values	Default value	Description
IF0_CHIP0_BASE to IF0_CHIP3_BASE IF1_CHIP0_BASE to IF1_CHIP3_BASE	Integer	Address where chips connected	0	Chip *y* base address for interfaces 0 and 1.
IF0_CHIP0_SIZE to IF0_CHIP3_SIZE IF1_CHIP0_SIZE to IF1_CHIP3_SIZE	Integer	Device size	0	Chip *y* size for interfaces 0 and 1.

PL350_SMC - registers

This section describes the registers.

You can access the configuration registers through the APB interface.

Table 4-127 PL350_SMC registers

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	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

Table 4-127 PL350_SMC registers (continued)

Name	Offset	Access	Description
opmode1_3	0x1E4	Read only	Opmode configuration
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[cm]
periph_id_1	0xFE4	Read only	Peripheral ID register 1[cm]
periph_id_2	0xFE8	Read only	Peripheral ID register 2[cm]
periph_id_3	0xFEC	Read only	Peripheral ID register 3[cm]
pcell_id_0	0xFF0	Read only	PrimeCell ID register 0[cm]
pcell_id_1	0xFF4	Read only	PrimeCell ID register 1[cm]
pcell_id_2	0xFF8	Read only	PrimeCell ID register 2[cm]
pcell_id_3	0xFFC	Read only	PrimeCell ID register 3[cm]

PL350_SMC - verification and testing

The functions of this component have been tested individually using a tailored test suite.

PL350_SMC - performance

This component is optimized to have negligible impact on transaction performance, except when memory remap settings are changed when there might be a significant effect.

4.4.48 PL350_SMC_NAND_FLASH component

This section describes the PL350_SMC_NAND_FLASH component.

PL350_SMC_NAND_FLASH - about

This LISA+ component is a model of the flash that you must connect to the PL350_SMC component, which is a model of the ARM PrimeCell Static Memory Controller (PL350).

Program the component as you would the hardware.

PL350_SMC_NAND_FLASH - ports

This section describes the ports.

Table 4-128 PL350_SMC_NAND_FLASH ports

Name	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

PL350_SMC_NAND_FLASH - parameters

This section describes the registers.

[cm] This register has no CADI interface.

Table 4-129 PL350_SMC_NAND_FLASH parameters

Name	Type	Allowed values	Default value	Description
DEVICE_NAME	String	-	"Samsung K9F1G08U0M"	Name of the device
DEVICE_1	Integer	-	default(0xEC)	The device ID
DEVICE_2			default(0xEC)	
DEVICE_3			default(0xDA)	
DEVICE_4			default(0x80)	
			default(0x15)	
NAND_FLASH_SIZE	Integer	-	0x1080000	Size of the flash device in bytes
NAND_PAGE_SIZE	Integer	0x2112, 0x528	0x2112	Page size
NAND_SPARE_SIZE_PER_PAGE	Integer	64, 16	64	Extra bits
NAND_VALID_SIZE_PER_PAGE	Integer	-	2048	Valid page size
NAND_PAGE_COUNT_PER_BLOCK	Integer	-	64	Number of pages in each block
NAND_BLOCK_COUNT	Integer	-	2048	Number of blocks in the flash device

PL350_SMC_NAND_FLASH - verification and testing

This component passes tests as part of an integrated platform.

4.4.49 PL370_HDLCD component

This section describes the PL370_HDLCD component.

PL370_HDLCD - about

This LISA+ component is a model of the HDLCD controller supporting *High Definition* (HD) resolutions.

PL370_HDLCD - ports

This section describes the ports.

Table 4-130 PL370_HDLCD ports

Name	Protocol	Type	Description
clk_in	ClockSignal	Slave	Master clock input, typically 24MHz, to drive pixel clock timing.
display	LCD	Master	Connection to visualization component
intr	Signal	Master	Interrupt signaling line for flyback events
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
pvbus_m	PVBus	Master	DMA port for collecting video data from memory

PL370_HDLCD - parameters

This section describes the parameters.

Table 4-131 PL370_HDLCD parameters

Name	Type	Allowed values	Default value	Description
diagnostics	int	0x0-0x4	0x0	Diagnostics level

PL370_HDLCD - registers

This section describes the registers.

Table 4-132 PL370_HDLCD registers

Name	Offset	Access	Description
VERSION	0x0000	Read only	Version Register
INT_RAWSTAT	0x0010	Read/write	Interrupt Raw Status Register
INT_CLEAR	0x0014	Write only	Interrupt Clear Register
INT_MASK	0x0018	Read/write	Interrupt Mask Register
INT_STATUS	0x001C	Read only	Interrupt Status Register
FB_BASE	0x0100	Read/write	Frame Buffer Base Address Register
FB_LINE_LENGTH	0x0104	Read/write	Frame Buffer Line Length Register
FB_LINE_COUNT	0x0108	Read/write	Frame Buffer Line Count Register
FB_LINE_PITCH	0x010C	Read/write	Frame Buffer Line Pitch Register
BUS_OPTIONS	0x0110	Read/write	Bus Options Register
V_SYNC	0x0200	Read/write	Vertical Synch Width Register
V_BACK_PORCH	0x0204	Read/write	Vertical Back Porch Width Register
V_DATA	0x0208	Read/write	Vertical Data Width Register
V_FRONT_PORCH	0x020C	Read/write	Vertical Front Porch Width Register
H_SYNC	0x0210	Read/write	Horizontal Synch Width Register
H_BACK_PORCH	0x0214	Read/write	Horizontal Back Porch Width Register
H_DATA	0x0218	Read/write	Horizontal Data Width Register
H_FRONT_PORCH	0x021C	Read/write	Horizontal Front Porch Width Register
POLARITIES	0x0220	Read/write	Polarities Register
COMMAND	0x0230	Read/write	Command Register
PIXEL_FORMAT	0x0240	Read/write	Pixel Format Register
RED_SELECT	0x0244	Read/write	Color Select Registers
GREEN_SELECT	0x0248	Read/write	Color Select Registers
BLUE_SELECT	0x024C	Read/write	Color Select Registers

PL370_HDLCD - verification and testing

This component passes tests using Linux and bare metal code on a development model.

PL370_HDLCD - performance

Too fast a pixel clock can slow the rest of the simulation.

4.4.50 PL390_GIC component

This section describes the PL390, which is a *Generic Interrupt Controller* (GIC).

PL390_GIC - about

This LISA and C++ component is a model of r0p0 of the ARM PrimeCell Generic Interrupt Controller (PL390), which implements the Generic Interrupt Controller Architecture Specification.

The GIC provides support for three interrupt types:

- *Software Generated Interrupt* (SGI).
- *Private Peripheral Interrupt* (PPI).
- *Shared Peripheral Interrupt* (SPI).

You can set:

- Security state for an interrupt.
- Priority state for an interrupt.
- Enabling or disabling state for an interrupt.
- Processors that receive an interrupt.

PL390_GIC - ports

This section describes the ports.

Table 4-133 PL390_GIC ports

Name	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
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

Table 4-133 PL390_GIC ports (continued)

Name	Protocol	Type	Description
reset_in	Signal	Slave	Reset signal
spi[*988*]	Signal	Slave	Shared peripheral interrupt inputs

PL390_GIC - parameters

This section describes the parameters.

Table 4-134 PL390_GIC parameters

Name	Type	Allowed values	Default value	Description
ARCHITECTURE_VERSION	Integer	0-1	1	Set architecture version in periph_id register
AXI_IF	Boolean	true, false	true	
C_ID_WIDTH	Integer	0-32	32	Width of the processor interface master id
D_ID_WIDTH	Integer	0-32	32	Width of the distributor interface master id
ENABLE_LEGACY_FIQ	Boolean	true, false	true	Provide legacy fiq interrupt inputs
ENABLE_LEGACY_FIQ	Boolean	true, false	true	Provide legacy irq interrupt inputs
ENABLE_PPI_EDGE	Boolean	true, false	false	PPI edge sensitive
ENABLE_TRUSTZONE	Boolean	true, false	true	Support trust zone
INIT_ENABLE_C0 to INIT_ENABLE_C7	Integer	-	0xFFFFFFFF	INIT value of ENABLE_C<n>
INIT_ENABLE_D0 to INIT_ENABLE_D7	Integer	-	0xFFFFFFFF	INIT value of ENABLE_D<n>
INIT_MATCH_C0 to INIT_MATCH_C7	Integer	-	0xFFFFFFFF	INIT value of MATCH_C<n>
INIT_MATCH_D0 to INIT_MATCH_D7	Integer	-	0xFFFFFFFF	INIT value of MATCH_D<n>
NUM_CPU	Integer	1-8	8	Number of processor interfaces
NUM_LSPI	Integer	0-31	31	Number of lockable SPIs
NUM_PPI	Integer	0-16	16	Number of private peripheral interrupts
NUM_PRIORITY_LEVELS	Integer	16, 32, 64, 128, 256	256	Number of priority levels
NUM_SGI	Integer	0-16	16	Number of software generated interrupts
NUM_SPI	Integer	0-988	988	Number of shared peripheral interrupts

PL390_GIC - registers

This section describes the registers.

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 identify the originator of a transaction on pvbus_distributor. <n> corresponds to the number of a processor interface.

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 4-135 PL 390_GIC registers: distributor interface

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
...			
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
...			

Table 4-135 PL 390_GIC registers: distributor interface (continued)

Name	Offset	Access	Description
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
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
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

PL390_GIC - debug features

This component provides some registers for functional verification and integration testing.

PL390_GIC - verification and testing

This component has been run against the RTL validation suite and has been successfully used in validation platforms.

4.4.51 PS2Keyboard component

This section describes the PS2Keyboard component.

PS2Keyboard - about

This component 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.

This is a LISA+ component.

PS2Keyboard - ports

This section describes the ports.

Table 4-136 PS2Keyboard ports

Name	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

Related references

2.4.6 KeyboardStatus protocol on page 2-56.

PS2Keyboard - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PS2Keyboard - performance

ARM expects this component to have little effect on the performance of PV systems. However if it is connected to the Visualisation component, then the performance of the component depends on that of the visualization.

4.4.52 PS2Mouse component

This section describes the PS2Mouse component.

PS2Mouse - about

This component implements the PS/2 register interface of a PS/2 style mouse.

The mouse movement and button press data must come from another component such as the Visualisation component.

This is a LISA+ component.

PS2Mouse - ports

This section describes the ports.

Table 4-137 PS2Mouse ports

Name	Protocol	Type	Description
`mouse`	MouseStatus	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

Related references

4.6 Visualisation Library on page 4-411.
2.4.10 MouseStatus protocol on page 2-58.

PS2Mouse - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PS2Mouse - performance

ARM expects this component to have little effect on the performance of PV systems. However if it is connected to the Visualisation component, then the performance of the component depends on that of the visualization.

4.4.53 RAMDevice component

This section describes the RAMDevice component.

RAMDevice - about

This LISA+ component is an efficient implementation of a generic memory device.

As a generic device, it does not have a hardware revision code.

RAMDevice - ports

This section describes the ports.

Table 4-138 RAMDevice ports

Name	Protocol	Type	Description
`pvbus`	PVBus	Slave	Bus slave interface

RAMDevice - parameters

This section describes the parameters.

Table 4-139 RAMDevice parameters

Name	Type	Allowed values	Default value	Description
`size`	`uint64_t`	`0x1000` to 2^{64} `- 1`, but must also be a multiple of `0x1000` (4KB)	`0x100000000` (4GB)	Size of the memory in bytes
`global_monitor_ignores_non_ex_store`	`bool`	`true`, `false`	`false`	Global monitor ignores non-exclusive stores

RAMDevice - debug features

This component implements a CADI MEMORY view. You can connect a CADI client to the target and view the physical memory contents.

RAMDevice - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.54 RemapDecoder component

This section describes the RemapDecoder component.

RemapDecoder - about

The RemapDecoder component provides the low memory flash/RAM remap behavior of the example systems.

This is a LISA+ component.

RemapDecoder - ports

This section describes the ports.

Table 4-140 RemapDecoder ports

Name	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.

RemapDecoder - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.4.55 SerialCrossover component

This section describes the SerialCrossover component.

SerialCrossover - about

This component provides the functionality of a serial crossover cable. It implements two SerialData slave ports and can connect two SerialData master ports, such as from PL011_Uart components.

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.

This is a C++ component.

SerialCrossover - ports

This section describes the ports.

Table 4-141 SerialCrossover ports

Name	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

SerialCrossover - verification and testing

This component passes tests as part of the Dual Processor example system by using the test suites and by booting operating systems.

4.4.56 SMSC_91C111 component

This section describes the SMSC_91C111 component.

SMSC_91C111 - about

This C++ component is a model of the SMSC 91C111 Ethernet controller.

It 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.

Related tasks

Configuring the networking environment for Microsoft Windows on page 1-36.
Configuring the networking environment for Linux on page 1-39.

SMSC_91C111 - ports

This section describes the ports.

Table 4-142 SMSC_91C111 ports

Name	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

Related references

2.4.18 VirtualEthernet protocol on page 2-63.

SMSC_91C111 - parameters

This section describes the parameters.

Table 4-143 SMSC_91C111 parameters

Name	Type	Allowed values	Default value	Description
enabled	bool	true, false	false	Enables user-mode networking, for sending Ethernet frames between components.
mac_address	string	-	'00:02:f7:ef:00:02'	MAC address to use on host.
promiscuous	bool	true, false	true	Puts host Ethernet controller into promiscuous mode, for instance when sharing the Ethernet controller with the host OS.

SMSC_91C111 - mac_address parameter

This parameter has two options.

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 allocate the 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.

Related references

SMSC_91C111 - mac_address parameter on page 4-384.

SMSC_91C111 - registers

This component uses a banked register model of primarily 16-bit registers. There are also indirectly accessible registers for the PHY unit.

SMSC_91C111 - bank 0 registers

This section describes the bank 0 registers.

Table 4-144 SMSC_91C111 bank 0 registers

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

SMSC_91C111 - bank 1 registers

This section describes the bank 1 registers.

Table 4-145 SMSC_91C111 bank 1 registers

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

Table 4-145 SMSC_91C111 bank 1 registers (continued)

Name	Offset	Access	Description
CONTROL	0xC	Read/write	Control
BANK	0xE	Read/write	Bank select

SMSC_91C111 - bank 2 registers

This section describes the bank 2 registers.

Table 4-146 SMSC_91C111 bank 2 registers

Name	Offset	Access	Description
MMU_COMMAND	0x0	Read/write	MMU commands
PNR	0x2	Read/write	Packet number
ALLOCATED	0x3	Read/write	Allocated packet number
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[cn]
INTERRUPT	0xC	Read/write	Interrupt status
INTERRUPT_MASK	0xD	Read/write	Interrupt mask
BANK	0xE	Read/write	Bank select

SMSC_91C111 - bank 3 registers

This section describes the bank 3 registers.

Table 4-147 SMSC_91C111 bank 3 registers

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

SMSC_91C111 - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

[cn] The data register can be accessed as 8, 16 or 32 bits and adjusts the pointer accordingly.

4.4.57 SP804_Timer component

This section describes the SP804_Timer component.

SP804_Timer - about

This LISA+ component is a model of the ARM Dual-Timer Module (SP804).

SP804_Timer - ports

This section describes the ports.

Table 4-148 SP804_Timer ports

Name	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

SP804_Timer - registers

This section describes the registers.

Table 4-149 SP804_Timer registers

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
Timer2RIS	`0x030`	Read only	Raw interrupt status register
Timer2MIS	`0x034`	Read only	Masked interrupt status register
Timer2BGLoad	`0x038`	Read/write	Background load register

SP804_Timer - verification and testing

This component passes tests as part of the SMLT component.

4.4.58 SP805_Watchdog component

This section describes the SP805_Watchdog component.

SP805_Watchdog - about

This LISA+ component is a model of the ARM Watchdog Module (SP805).

SP805_Watchdog - ports

This section describes the ports.

Table 4-150 SP805_Watchdog ports

Name	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

SP805_Watchdog - parameters

This section describes the parameters.

Table 4-151 SP805_Watchdog parameters

Name	Type	Allowed values	Default value	Description
`simhalt`	`bool`	`true`, `false`	`false`	If `true`, halt simulation instead of signaling reset.

SP805_Watchdog - registers

This section describes the registers.

Table 4-152 SP805_Watchdog registers

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
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

SP805_Watchdog - verification and testing

This component passes tests as part of an integrated platform.

4.4.59 SP810_SysCtrl component

This section describes the SP810_SysCtrl component.

SP810_SysCtrl - about

This LISA+ component is a model of the PrimeXsys System Controller (SP810).

SP810_SysCtrl - ports

This section describes the ports.

Table 4-153 SP810_SysCtrl ports

Name	Protocol	Type	Description
clk_in	ClockSignal	Slave	Clock input
hclkdivsel[co]	ValueState	Master	Define the processor clock/bus clock ratio.
npor[co]	Signal	Slave	Power on reset
pll_en[co]	Signal	Master	PLL enable output
pvbus	PVBus	Slave	Slave port for connection to PV bus master/decoder
ref_clk_in	ClockSignal	Slave	Clock source used by the Timer and Watchdog modules.
sleep_mode[co]	Signal	Master	Control clocks for SLEEP mode.
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
remap_stat[co]	StateSignal	Slave	Remap status input
sys_mode[co]	ValueState	Slave	Present system mode
sys_stat[co]	ValueState	Slave	System status input
wd_clk_en[co]	Signal	Master	Watchdog module clock enable output
wd_en[co]	Signal	Slave	Watchdog module enable input

SP810_SysCtrl - parameters

This section describes the parameters.

Table 4-154 SP810_SysCtrl parameters

Name	Type	Allowed values	Default value	Description
sysid	int	-	0x00000000	System identification register.
use_s8	bool	true, false	false	Enable switch S8.

SP810_SysCtrl - registers

This section describes the registers.

[co] Not fully implemented. Using this port has unpredictable results.

Table 4-155 SP810_SysCtrl registers

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
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

4.4.60 TelnetTerminal component

This section describes the TelnetTerminal component.

TelnetTerminal - about

This component permits UART data to transfer 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.

This is a C++ component.

Related concepts

PL011_Uart - about on page 4-340.
1.11 Use of the TelnetTerminal on page 1-32.

TelnetTerminal - ports

This section describes the ports.

Table 4-156 TelnetTerminal ports

Name	Protocol	Type	Description
`serial`	SerialData	Slave	Slave port for connecting to a SerialData master.

TelnetTerminal - parameters

This section describes the parameters.

Table 4-157 TelnetTerminal parameters

Name	Type	Allowed values	Default value	Description
`mode`	`string`	`telnet`[cp], `raw`[cq]	`telnet`	Terminal operation mode.
`quiet`	`bool`	`true`, `false`	`false`	Avoid output on `stdout` or `stderr`.
`start_port`	`int`	-	5000	Telnet TCP port number, of the port for the terminal when the system starts. If this port is not free, the port value is incremented by 1 until a free port is found.
`start_telnet`	`bool`	`true`, `false`	`true`	Enable terminal when the system starts.

TelnetTerminal - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

TelnetTerminal - library dependencies

The performance of this component depends on the host terminal.

[cp] In telnet mode, this component supports a subset of the telnet protocol defined in RFC 854.
[cq] In raw mode, this component does not interpret or modify the byte stream contents. This permits a debugger connection, for example, to connect a gdb client to a gdbserver running on the target operating system.

4.4.61 TZC_400 component

This section describes the TZC_400 component.

TZC_400 - about

This LISA+ component is a model of r0p1 of the ARM TZC-400 CoreLink TZC-400 TrustZone® Address Space Controller.

The TZC-400 determines, under software control, whether a particular bus master is permitted to issue Non-secure accesses to a particular physical address.

The component has:

- Eight address regions in addition to the base region, region 0.
- A programmable control block for security-access permissions configuration through the *Advanced Peripheral Bus* (APB).
- Up to four address filters that share common set region set-up registers.
- Software configurable permission check failure reporting and interrupt signaling.
- Filtering with a *Non-Secure Access ID* (NSAID).
- A gate keeper, to allow or block accesses to the filter unit.
- Configurable reset values of region configuration registers and other key configuration registers.

Unlike the hardware, it does not have:

- Asynchronous clocks. The model does not need clocks for data transfer, or clock signals.
- *QoS Virtual Network* (QVN) support. Specifically, it does not implement the vnet bits[27:24] in `FAIL_ID_<x>` registers.
- Fast Path and Fast Path ID. In the model, transactions occur at similar speeds.
- 256 outstanding accesses globally for each read or write Normal Paths and configurable 8, 16, or 32 outstanding accesses on Fast Path read access. The model does not support QVN, and this concept is meaningless for a PV level model.
- Configurable address bus width, data bus width, transaction ID tag, and USER bus width. A single bus implementation, PVBus, covers these AXI bus hardware implementation details.

Related references

4.5.3 PVBus Transaction Master ID on page 4-401.

4.5.11 Labeller and LabellerForDMA330 components on page 4-407.

TZC_400 - ports

This section describes the ports.

Table 4-158 TZC_400 ports

Name	Protocol	Type	Description
`master tzcint`	Signal	Master	TrustZone interrupt signal, controlled by ACTION register
`slave tzc_reset`	Signal	Slave	Reset signal from external master
`apbslave_s`	Signal	Slave	Bus access for control register
`filter_pvbus_s[4]`	PVBus	Slave	Incoming bus traffic to filter units
`filter_pvbus_m[4]`	PVBus	Master	Outgoing bus traffic from filter units

Related references

2.4.17 TZFilterControl protocol on page 2-62.

TZC_400 - parameters

This section describes the parameters.

Table 4-159 TZC_400 parameters

Name	Type	Allowed values	Default value	Description
diagnostics	int	0x0-0x4	0x0	Level of diagnostics messages in the model.
id_mapping[cr]	int	-	-[cs]	Maps from the low-order 16 bits of the Master ID into an NSAID value that the system designer specifies.[ct] Deprecated: ARM recommends master_id_from_label.
master_id_from_label[cr]	int	-	false	Take Master IDs directly from the label field in PVBus transactions, and from them draw NSAIDs directly without mapping. When true, this parameter overrides the id_mapping parameter.
rst_action	int	0x0-0xFFFFFFFF	0x0	Reset value of ACTION register.
rst_build_config[cr]	int	0x0-0xFFFFFFFF	-[cu]	Reset value of BUILD_CONFIG register.
rst_gate_keeper	int	0x0-0xFFFFFFFF	0x0	Reset value of GATE_KEEPER register.
rst_region_attributes_<*x*>[cr]	int	0x0-0xFFFFFFFF	-	Reset value of the REGION_<*x*>_ATTRIBUTES register, region <*x*> Secure state attributes.
rst_region_base_low_<*x*>	int	0x0-0xFFFFFFFF	-	Reset value of the REGION_<*x*>_BASE_ADDRESS_LOW register, region <*x*> base memory address, low 32 bits. <*x*> is the filter unit number. There is no register for region 0 because the value is fixed.
rst_region_base_high_<*x*>	int	0x0-0xFFFFFFFF	-	Reset value of the REGION_<*x*>_BASE_ADDRESS_HIGH register, region <*x*> base memory address, high 32 bits. There is no register for region 0 because the value is fixed.
rst_region_id_access_<*x*>	int	0x0-0xFFFFFFFF	-	Reset value of the REGION_<*x*>_ID_ACCESS register, region <*x*> NSAID permissions.
rst_region_top_low_<*x*>	int	0x0-0xFFFFFFFF	-	Reset value of the REGION_<*x*>_TOP_ADDRESS_LOW register, region <*x*> top memory address, low 32 bits. There is no register for region 0 because the value is fixed.
rst_region_top_high_<*x*>	int	0x0-0xFFFFFFFF	-	Reset value of the REGION_<*x*>_TOP_ADDRESS_HIGH register, region <*x*> top memory address, high 32 bits.

TZC_400 - registers

This component provides the registers that the *Technical Reference Manual* (TRM) specifies.

However, it does not implement:

- The vnet bits[27:24] in FAIL_ID_<*x*> registers.
- Any background logic for the speculation control register. This does not affect model behavior.

[cr] Configure master_id_from_label or id_mapping, rst_build_config, and rst_region_attributes_0 before running the model to set the desired behaviors. Otherwise, the system resets all region configuration registers, rst_action, and rst_gate_keeper to 0, and resets rst_build_config and rst_region_attributes_0 to sensible default values. Configure either id_mapping or master_id_from_label at model init, or a warning message appears.

[cs] No default.

[ct] The syntax of id_mapping is:<masterid_0>:<nsaid_0>,<masterid_1>:<nsaid_1>,<masterid_*n*>:<nsaid_*n*>.

Separate the mapping pairs by ,. The masterid is the ID of the bus master, such as the parameter CLUSTER_ID on Cortex-A15/7, cluster_id port of Cortex-A15/7, or master_id parameter for Cortex-M3.

[cu] The reset values vary with the system. See the system design documentation or system integration documentation. (0x3003F08 for AEMv8-A.)

TZC_400 - subcomponents

This component contains TZFilterUnits and a TZDummyDevice.

TZFilterUnits
The TZC-400 has four TZFilterUnits. The BUILD_CONFIG register sets the configuration. The `rst_build_config` parameter controls the register.

TZDummyDevice
An internal dummy device that mimics RAZ/WI for TZFilterUnits. The system uses it when there is a permission violation and a bus returns Transaction OK.

TZC_400 - verification and testing

This component passes use tests with Linux operating systems.

4.4.62 TZIC component

This section describes the TZIC component.

TZIC - about

This LISA+ component is a model of r0p0 of the ARM AMBA 3 TrustZone Interrupt Controller (SP890).

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.

TZIC - ports

This section describes the ports.

Table 4-160 TZIC ports

Name	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

TZIC - registers

This section describes the registers.

Table 4-161 TZIC registers

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.

Table 4-161 TZIC registers (continued)

Name	Offset	Access	Description
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.

TZIC - verification and testing

This component passes tests separately using its own test suite.

This component passes tests inside the SMLT component. The FIQ passes tests under the secure environment.

4.4.63 v8EmbeddedCrossTrigger_Interface component

This component shows what a LISA+ component needs to have to connect to a *Cross Trigger Matrix* (CTM).

A processor with a *Cross Trigger Interface* (CTI) connects to a CTM.

This is a LISA+ component.

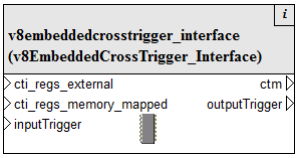

Figure 4-1 v8EmbeddedCrossTrigger_Interface in System Canvas

4.4.64 v8EmbeddedCrossTrigger_Matrix component

This section describes the v8EmbeddedCrossTrigger_Matrix component.

v8EmbeddedCrossTrigger_Matrix - about

This C++ component is a model of a platform level *Cross Trigger Matrix* (CTM) for connection to the *Cross Trigger Interface* (CTI) ports provided on ARMv8 processors within Fast Models.

The combination of the CTI and the CTM provides an architectural model of the CoreSight embedded triggering system.

A single instance of the v8EmbeddedCrossTrigger_Matrix component supports up to four clusters, each containing four cores. For example:

```
cluster0.cti[0] => v8ect.cti[0];
cluster0.cti[1] => v8ect.cti[1];
cluster0.cti[2] => v8ect.cti[2];
cluster0.cti[3] => v8ect.cti[3];
...
cluster4.cti[3] => v8ect.cti[12];
cluster4.cti[3] => v8ect.cti[13];
cluster4.cti[3] => v8ect.cti[14];
cluster4.cti[3] => v8ect.cti[15];
```

v8EmbeddedCrossTrigger_Matrix - ports

This section describes the ports.

Table 4-162 v8EmbeddedCrossTrigger_Matrix ports

Name	Protocol	Type	Description
`cti[0-15]`	v8EmbeddedCrossTrigger_controlprotocol	Slave	Port for connection of CTI components in ARMv8 processor components.

Related references

2.5.4 v8EmbeddedCrossTrigger_controlprotocol protocol on page 2-65.

v8EmbeddedCrossTrigger_Matrix - parameters

This section describes the parameters.

Table 4-163 v8EmbeddedCrossTrigger_Matrix parameters

Name	Type	Allowed values	Default value	Description
`has_CTIAUTHSTATUS`	`bool`	`true`, `false`	`true`	Enables the CTIAUTHSTATUS register.
`number-of-channels`	`uint32_t`	`0x3-0x20`, `3-32`	`0x4`, `4`	Number of channels in the CTM.

v8EmbeddedCrossTrigger_Matrix - registers

The CTI gives access to the registers. There is no direct access.

v8EmbeddedCrossTrigger_Matrix - verification and testing

This component passes tests as part of the ARMv8 Architecture Verification Kit.

4.4.65 VFS2 component

This section describes the VFS2 component.

VFS2 - about

This component implements the *Virtual File System* (VFS).

The VFS is a virtual device that provides access to parts of the underlying host filesystem through a target OS-specific driver and a memory-mapped device called the MessageBox. The VFS component is virtual, so you might have to write your own driver. Fast Models contains example drivers. These drivers are for bare metal, written in C++, and for Linux, written in C. You can use much of the driver code for porting to other operating systems.

This component contains:

VFS LISA component
: Coordinates device activity.

MessageBox component
: Handles bus activity and interrupts.

`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 this 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.

This is a LISA+ component.

Related references

4.4.27 MessageBox component on page 4-327.

VFS2 - operations

This section describes the operations.

Access to the host filesystem through the VFS is analogous to access to a shared network drive, and you can expect it to behave in the same way.

These are the VFS operations:

`getattr`
: Retrieve metadata for the file, directory, or symbolic link.

`mkdir`
: Create a new directory.

`remove`
: Remove a file, directory, or symbolic link.

`rename`
: Rename a file, directory, or symbolic link.

`rmdir`
: Remove an empty directory.

`setattr`
: Set metadata for the file, directory, or symbolic link.

  Note

  The VFS does not implement `setattr`.

The VFS does not support symbolic links. The model cannot create hard links, but hard links that the host operating system created function correctly.

These are the VFS mount points:

`closemounts`
: Free the iterator handle returned from `openmounts`.

`openmounts`
: Retrieve an iterator handle for the list of available mounts.

`readmounts`
: Read one entry from the mount iterator ID.

These are the VFS directory iterators:

`closedir`
: Free a directory iterator handle retrieved by `opendir`.

`opendir`
: Retrieve an iterator handle for the directory specified.

`readdir`
: Read the next entry from the directory iterator.

These are the VFS file operations:

`closefile`
: Free a handle opened with `openfile`.

`filesync`
: Force the host OS to flush all file data to persistent storage.

`getfilesize`
: Return the size of a file, in bytes.

`openfile`
: Return a handle to the file specified.

`readfile`
: Read a block of data from a file.

`setfilesize`
: Set the size of a file in bytes, either by truncating, or extending the file with zeroes.

`writefile`
: Write a block of data to a file.

Note

Datestamps are in milliseconds since the VFS epoch of January 01 1970 00:00 UTC and are host datestamps. The host datestamp might be in the future relative to the OS datestamp.

VFS2 - ports

This section describes the ports.

Table 4-164 VFS2 ports

Name	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.

Related references

MessageBox - registers on page 4-328.

VFS2 - parameters

This section describes the parameters.

Table 4-165 VFS2 parameters

Name	Type	Allowed values	Default value	Description
`mount`	`string`	-	"	Path to host folder to make accessible inside the model.

VFS2 - verification and testing

This component passes tests through use with Linux operating systems.

VFS2 - performance

ARM expects this component to have little effect on the performance of PV systems. The component depends on the performance of the host filesystem.

4.4.66 VirtioBlockDevice component

This section describes the VirtioBlockDevice component.

VirtioBlockDevice - parameters

This section describes the parameters.

Table 4-166 VirtioBlockDevice parameters

Parameter	Type	Allowed values	Default value	Description
image_path	string	-	"	Image file path.
quiet	bool	true, false	false	Do not print warnings for malformed commands or descriptors.
read_only	bool	true, false	false	Permit reads from the device, but not writes to it.
secure_accesses	bool	true, false	false	Force the device to generate transactions with NS=0.

4.4.67 VirtioP9Device component

This section describes the VirtioP9Device component.

VirtioP9Device component - about

This C++ component is a model of r0p0 of a virtio P9 server. It implements a subset of the Plan 9 File Protocol over a virtio transport.

This component accesses a directory on the host filesystem within Linux (and other operating systems that implement the protocol) running on a model.

The component implements a subset of the Linux 9P2000.L protocol. It has limitations:

- You can mount only one host directory per instance of the component.
- It supports a subset of 9P2000.L message types: Tversion, Tlopen, Tlcreate, Tgetattr, Tsetattr, Treaddir, Tmkdir, Tattach, Twalk, Tread, Twrite, Tclunk, Tremove, Trename. On Linux hosts, it also supports: Treadlink, Tsymlink.
- On Windows hosts, it ignores Unix permissions when writing files.
- On Windows hosts, it performs a simple mapping from Windows to Unix permissions when reading.
- On Windows hosts, symbolic links appear as regular files.
- On Windows hosts, it does not perform operations (writing, deleting, renaming) on a file that another process has open.

VirtioP9Device component - setting up

This section describes how to set up the component in a virtual platform.

To use this component:

Prerequisites

Use a version of Linux that supports v9fs over virtio and virtio-mmio devices.

Procedure

1. Add the VirtioP9Device component to the platform.
2. Update the device tree to include the VirtioP9Device component, or specify it on the kernel command line.

   Do this step in the same way as for the VirtioBlockDevice component.
3. Set the parameter `<component>.root_path` to the host path to the directory to mount in the model.
4. Mount the directory inside the model with this command:

   ```
   mount -t 9p -o trans=virtio,version=9p2000.L FM /mnt/host
   ```

   where `/mnt/host` is the path to where to mount the directory in the model.

   Unmount the directory, if necessary, with:

   ```
   umount /mnt/host
   ```

   Fast Models can now access the Plan 9 filesystem of a host using the VirtioP9Device component.

VirtioP9Device component - ports

This section describes the ports.

Table 4-167 VirtioP9Device ports

Name	Protocol	Type	Description
intr	Signal	Master	Virtio device sets interrupt to signal completion.
pvbus	PVBus	Slave	Virtio MMIO control/config/status registers.
virtio_m	PVBus	Master	Virtio device performs DMA accesses via master.

VirtioP9Device component - parameters

This section describes the parameters.

Table 4-168 VirtioP9Device parameters

Name	Type	Allowed values	Default value	Description
mount_tag	string	-	FM	Mount tag.
quiet	bool	true, false	false	Do not print warnings on malformed commands or descriptors.
root_path	string	-	-	Root directory path.
secure_accesses	bool	true, false	false	Make device generate transactions with NS=0.

4.4.68 VirtualEthernetCrossover component

This section describes the VirtualEthernetCrossover component.

VirtualEthernetCrossover - about

This component provides the functionality of an Ethernet crossover cable.

It implements two VirtualEthernet slave ports and enables you to connect two VirtualEthernet master ports. It forwards data received on one port to the other port without delay.

This is a LISA+ component.

VirtualEthernetCrossover - ports

This section describes the ports.

Table 4-169 VirtualEthernetCrossover ports

Name	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

Related references

2.4.18 VirtualEthernet protocol on page 2-63.

VirtualEthernetCrossover - verification and testing

This component passes tests as part of the VE example system by using the networking test suites and by booting operating systems.

4.5 PVBus components

This section introduces the PVBus components that Fast Models provides.

This section contains the following subsections:

4.5.1 About 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.

A number of components, which abstract the internal details, access the PVBus API. You must use these components correctly to achieve high simulation speeds.

4.5.2 About 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 connect to one slave, but any number of other masters can connect 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 connect 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.

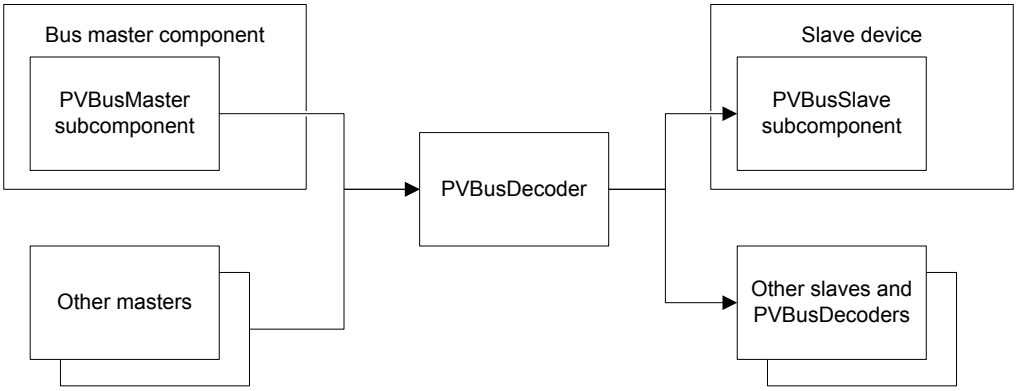

Figure 4-2 Sample bus topology

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. Transactions are atomic unless slave devices block transactions, for example an SMMU with stall mode enabled. A slave device that can block transactions and all its upstream bus components must be re-entrant safe for bus transactions.

4.5.3 PVBus Transaction Master ID

Fast Models PVBus transactions have a 32-bit Master ID.

The Master ID is the ID of the bus master, for example of the DMA-330, Cortex-A15. You can make the Master ID of processing elements within a Cortex-A15 model unique by using the parameter `CLUSTER_ID` or the `cluster_id` port.

4.5.4 PVBus examples

This section describes the locations of examples of PVBus components.

For paths on Linux, substitute `$PVLIB_HOME` for the Microsoft Windows environment variable `%PVLIB_HOME%`.

Table 4-170 PVBus examples

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

4.5.5 PVBusDecoder component

This section describes the PVBusDecoder component.

PVBusDecoder - about

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.

This is a C++ component.

PVBusDecoder - ports

This section describes the ports.

Table 4-171 PVBusDecoder ports

Name	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.[cv]

PVBusDecoder - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

4.5.6 PVBusMapper component

This section describes the PVBusMapper component.

PVBusMapper - about

This component is a general purpose modeling component.

It performs like PVBusModifier, but it also has multiple downstream ports, allows routing of transactions to any one of these ports, and allows arbitrary remapping of transaction addresses and attributes.

As a generic device, it does not have a hardware revision code.

PVBusMapper - ports

This section describes the ports.

Table 4-172 PVBusMapper ports

Name	Protocol	Type	Description
control	PVBusMapperControl	Master	Control port for configuring the routing and mapping behavior.
pvbus_m[64]	PVBus	Master	Bus master ports.
pvbus_s	PVBus	Slave	Bus slave port.
reset	Signal	Slave	Reset signal.

PVBusMapper - parameters

This section describes the parameters.

[cv] 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. The default address range for a slave is [0-(sizeOfMasterRange – 1)].

Table 4-173 PVBusMapper parameters

Name	Type	Allowed values	Default value	Description
handling_of_dvm_messages_from_downstream	string	forward, handle, terminate	forward	What to do with *Distributed Virtual Memory* (DVM) messages from downstream. The options are to 'forward' them upstream unaltered, to 'terminate' them, or to 'handle' them locally with handleDownstreamDVMMessage(). This parameter is not a runtime parameter.
handling_of_dvm_messages_from_upstream	string	forward, handle, terminate	forward	What to do with *Distributed Virtual Memory* (DVM) messages from upstream. The options are to 'forward' them downstream unaltered, to 'terminate' them, or to 'handle' them locally with handleUpstreamDVMMessage(). This parameter is not a runtime parameter.
handling_of_upstream_snoop_requests	string	forward, handle, terminate	forward	What to do with snoop requests from downstream. The options are to 'forward', 'terminate' or 'handle'. The snoop request addresses are not translated: if your device alters the address translation then you almost certainly want to 'terminate'. This parameter is not a runtime parameter.

4.5.7 PVBusMaster component

This section describes the PVBusMaster component.

PVBusMaster - about

This 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.

This is a C++ component.

PVBusMaster - ports

This section describes the ports.

Table 4-174 PVBusMaster ports

Name	Protocol	Type	Description
pvbus_m	PVBus	Master	Sends out generated transactions.
control	PVTransactionMaster	Slave	Enables the owning component to instantiate pv::TransactionGenerator objects.

Related references

2.4.15 PVTransactionMaster protocol on page 2-61.

PVBusMaster - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PVBusMaster - performance

ARM expects this component to have little effect on the performance of PV systems. However, the way in which you implement the component can influence performance.

Related concepts

TransactionGenerator efficiency considerations on page 4-408.

4.5.8 PVBusRange component

This section describes the PVBusRange component.

PVBusRange - about

Use PVBusRange to divide and remap the memory range into two output ports, a and b.

Use this component 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.

This is a C++ component.

PVBusRange - ports

This section describes the ports.

Table 4-175 PVBusRange ports

Name	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]

Related references

2.6.5 Value_64 protocol on page 2-66.

PVBusRange - verification and testing

This component passes directed component tests.

PVBusRange - performance

This component is optimized to have negligible impact on transaction performance except if memory remap settings change, when there might be a significant effect.

4.5.9 PVBusSlave component

This section describes the PVBusSlave component.

PVBusSlave - about

This 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.

This is a C++ component.

PVBusSlave - ports

This section describes the ports.

Table 4-176 PVBus ports

Name	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`	PVBusSlaveControl	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.

Related references

2.4.14 PVDevice protocol on page 2-61.
2.4.13 PVBusSlaveControl protocol on page 2-59.

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.

PVBusSlave - parameters

This section describes the parameters.

Table 4-177 PVBusSlave parameters

Name	Type	Allowed values	Default value	Description
max_access_width	uin32_t	Power of 2: 1, 2, 4, 8, 16	8	Maximum width of an access in bytes
size	uint64_t	0 to 2^{64} - 1, where 0 represents 2^{64} bytes, but must also be a multiple of 0x1000 (4KB)	0 (2^{64})	Addressable size of the device in bytes

PVBusSlave - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

PVBusSlave - performance

This 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 component handle the request internally. ARM recommends using the internal support for memory-like regions where possible.

4.5.10 TZSwitch component

This section describes the TZSwitch component.

TZSwitch - about

The TZSwitch component permits you to control transaction routing based on the TrustZone security state of the transaction.

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.

This is a C++ component.

TZSwitch - ports

This section describes the ports.

Table 4-178 TZSwitch ports

Name	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

TZSwitch - parameters

This section describes the parameters.

Table 4-179 TZSwitch parameters

Name	Type	Allowed values	Default value	Description
`secure`	Integer	0, 1, 2, 3	1	Default routing for secure transactions
`normal`	Integer	0, 1, 2, 3	2	Default routing for normal transactions

The secure and normal parameter values control the initial state of this component.

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.

TZSwitch - verification and testing

This component passes tests as part of the VE example system by using VE test suites and by booting operating systems.

TZSwitch - performance

This component is optimized to have negligible impact on transaction performance. Changing memory remap settings might significantly affect performance.

4.5.11 Labeller and LabellerForDMA330 components

This section describes the Labeller and LabellerForDMA330 utility components.

Labeller and LabellerForDMA330 - about

The Labeller and LabellerForDMA330 utility components allow the system designer to embed values into the Label field for transactions generated by a Bus Master. They go between PVBus Master and Slave ports.

The following example codes for a labeller to add an ID for an HDLCD controller that is upstream of a TZC_400. The system designer specifies a unique set of IDs for use as *Non-Secure Access IDs* (NSAIDs) in the TZC_400. The labeller can insert these IDs directly into the transaction.

```
pl370_hdlcd : PL370_HDLCD();
hdlcd_labeller : Labeller( "label" = 2 );
pl370_hdlcd.pvbus_m => hdlcd_labeller.pvbus_s;
hdlcd_labeller.pvbus_m => output_bus.pvbus_s;
```

These are LISA+ components.

Labeller and LabellerForDMA330 - ports

This section describes the ports.

Table 4-180 Labeller and LabellerForDMA330 ports

Name	Protocol	Type	Description
`pvbus_s`	PVBus	Slave	Handles incoming transactions from PVBus masters.
`pvbus_m`	PVBus	Master	Handles outgoing PVBus transactions. Converted transactions go through this port.

Related references

4.5.3 PVBus Transaction Master ID on page 4-401.

4.5.12 PVBus C++ transaction and Tx_Result classes

This section describes the C++ transaction and `Tx_Result` classes.

Class pv::TransactionGenerator

This 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.

```
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.
    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);
};
```

TransactionGenerator efficiency considerations

TransactionGenerators are most efficient for multiple accesses to one 4KB page.

Each TransactionGenerator caches connection information internally. This improves efficiency for multiple accesses to a single 4KB page. If a component requires repeated access data from different pages, for example when streaming from one location to another, ARM recommends you create a TransactionGenerator for each location.

You can dynamically create and destroy TransactionGenerators, 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.

Enum pv::AccessWidth

This enum 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

This class is a base class for read and write transactions that are visible in the PVBusSlave subcomponent. It contains functionality common to both types of transaction.

This 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

This class 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

This class 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();
};
```

4.6 Visualisation Library

This section describes the components and protocols included in the Visualisation Library for Fast Models.

This section contains the following subsections:

4.6.1 Visualisation Library - about

The Visualisation Library does not model hardware directly but instead provides components, protocols, and a library. These permit a GUI display that lets you interact with the external I/O from the model platform.

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.

You can add the prebuilt GUITick component to your custom component. The GUITick component provides a LISA visualization component with a periodic signal that keeps 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.

The Visualisation Library supports one signaling protocol, the LCD protocol.

Related references

2.4.7 LCD protocol on page 2-56.

4.6.2 LISA Visualisation models on page 4-411.

4.6.4 Visualisation Library C++ classes on page 4-412.

4.6.2 LISA Visualisation models

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, copy the component LISA files from the relevant platform model directory, because they are not in the generic Model Library.

Related references

Chapter 7 Versatile Express Model on page 7-448.

Related information

Fast Models User Guide.

4.6.3 GUIPoll component

This section describes the GUIPoll component.

GUIPoll - about

This component provides a periodic signal for driving a GUI refresh in a visualization component

ARM intends it for use 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.

This is a LISA+ component.

GUIPoll - ports

This section describes the ports.

Table 4-181 GUIPoll ports

Name	Protocol	Type	Description
`gui_callback`	GUIPollCallback	Master	Sends callback requests to the visualization component.

Related references

2.4.4 GUIPollCallback protocol on page 2-56.

GUIPoll - parameters

This section describes the parameters.

Table 4-182 GUIPoll parameters

Name	Type	Allowed values	Default value	Description
`delay_ms`	Integer	-	20	Determines the period, in milliseconds of real time, between `gui_callback()` calls.

GUIPoll - verification and testing

This component passes tests as part of the PhoneSkin example platform, confirming that the system continues to refresh the GUI after the simulation stops.

4.6.4 Visualisation Library C++ classes

This section describes the Visualisation Library C++ classes and structures.

C++ classes inclusion

To use these Visualisation Library classes, begin your LISA component with the correct `#include` statement.

```
includes
{
#include “components/Visualisation.h”
}
```

Visualisation class

This class provides access to the API for creating a custom LISA visualization component.

Visualisation class - about

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 implemented using the *Simple DirectMedia Layer* (SDL) cross-platform rendering library.

Global functions

The global function is `createVisualisation()`.

`Visualisation *createVisualisation()`
: This generates an instance of the Visualisation Library. You can only call this function once, because SDL only supports opening a single display window.

Class Visualisation

This section describes the methods of the Visualisation class.

`~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 must 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

This class is the common base class for VisPushButtonRegion and VisRenderRegion, representing a region of the visualization display.

`~VisRegion()`
: Permits clients to delete a VisPushButtonRegion when it is no longer required.

`void setId(void *id)`
: Permits a client-defined identifier to be associated with the region.

`void *getId()`
: 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)`
: Sets the location of this region relative to the visualization window.

Class VisPushButtonRegion : public VisRegion

This 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 these methods:

`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

This class defines a region of the visualization window that can render client-drawn graphics, including a representation of the contents of an LCD.

In addition to the public method defined in VisRegion, the class defines these methods:

`VisRasterLayout const *lock()`
: Locks the region for client rendering. 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()`
: 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)`
: Causes the specified rectangle to be drawn to the GUI.

`int writeText(const char *text, int x, int y)`
: 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

This 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 these 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

This 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)`.

Chapter 5
Base Platform

This chapter describes the Base Platform system model.

It contains the following sections:

5.1 Base - about

The Base Platform system model allows early development, distribution, and demonstration of software deliverables.

The standard peripheral set enables software development and porting. The platform is an evolution of the VE *Fixed Virtual Platforms* (FVPs), based on the ARM *Versatile™ Express* (VE) hardware development platform.

The Base Platform system model provides:

- Two configurable clusters of up to four core models that implement:
  - AArch64 at all exception levels.
  - Configurable AArch32 support at all exception levels.
  - Configurable support for little and big endian at all exception levels.
  - Generic timers.
  - Self-hosted debug.
  - CADI debug.
  - GICv3 memory-mapped processor interfaces and distributor.
- Peripherals for multimedia or networking environments.
- Four PL011 UARTs.
- A CoreLink CCI-400 Cache Coherent Interconnect.
- Architectural GICv3 model.
- High Definition LCD Display Controller, 1920×1080 resolution at 60fps, with single I2S and four stereo channels.
- 64MB NOR flash and board peripherals.
- CoreLink TZC-400 TrustZone Address Space Controller.

ARM supplies these Base FVPs:

- FVP_Base AEMv8A-AEMv8A.
- FVP_Base AEMv8A-AEMv8A-CCN504.
- FVP_Base AEMv8A-AEMv8A-MMU500+DMA330.
- FVP_Base AEMv8A-AEMv8A-MMU500+DMA330-CCI500.
- FVP_Base Cortex-A57x1, 2, 4-A53x1, 4.
- FVP_Base Cortex-A72x1, 2, 4-A53x1, 4.
- FVP_Base Cortex-A72x4-A53x4-CCI500.

To run FVP_Base_AEMv8A-AEMv8A-MMU500+DMA330 or FVP_Base_AEMv8A-AEMv8A-MMU500+DMA330-CCI500, reconfigure the DMAC or SMMU ranges. The default ranges for these components overlap.

5.2 Base - memory

This section describes the memory of the Base Platform.

This section contains the following subsections:

5.2.1 Base - secure memory

Enable the access permissions with the `bp.secure_memory` parameter.

Table 5-1 Secure and Non-secure access permissions

Security	`bp.secure_memory = false`	`bp.secure_memory = true`
S	Secure and Non-secure access permitted.	Secure access is permitted, Non-secure access aborts.
S/NS	Secure and Non-secure access permitted.	Secure and Non-secure access are permitted.
P	Secure and Non-secure access permitted.	Access conditions are programmable by the TZC-400.

Note

The default state of the TZC-400 is to abort all accesses, even from Secure state.

Table 5-2 NSAIDs and filters that masters present to the TZC-400

Component	NSAID[cw]	Filter
Cluster 0	9	0
Cluster 1	9	0
VirtIO	8	0
HDLCD0	2	2
CLCD	1	2

5.2.2 Base - memory map

The basis of this map is the Versatile Express RS2 memory map with extensions.

Table 5-3 Base Platform memory map

Peripheral	Start address	Size	End address	Security
Trusted Boot ROM, secure flash, IntelStrataFlashJ3	`0x00_0000_0000`	64MB	`0x00_03FF_FFFF`	S
Trusted SRAM	`0x00_0400_0000`	256KB	`0x00_0403_FFFF`	S
Trusted DRAM	`0x00_0600_0000`	32MB	`0x00_07FF_FFFF`	S
NOR flash, flash0, IntelStrataFlashJ3	`0x00_0800_0000`	64MB	`0x00_0BFF_FFFF`	S/NS
NOR flash, flash1, IntelStrataFlashJ3	`0x00_0C00_0000`	64MB	`0x00_0FFF_FFFF`	S/NS
PSRAM[cx]	`0x00_1400_0000`	64MB	`0x00_17FF_FFFF`	S/NS

[cw] *Non-Secure Access IDentity*.
[cx] The device is implemented as RAM and is 8MB in size.

Table 5-3 Base Platform memory map (continued)

Peripheral	Start address	Size	End address	Security
VRAM	0x00_1800_0000	32MB	0x00_19FF_FFFF	S/NS
Ethernet, SMSC 91C111	0x00_1A00_0000	16MB	0x00_1AFF_FFFF	S/NS
USB, unimplemented	0x00_1B00_0000	16MB	0x00_1BFF_FFFF	S/NS
VE System Registers	0x00_1C01_0000	64KB	0x00_1C01_FFFF	S/NS
System Controller, SP810	0x00_1C02_0000	64KB	0x00_1C02_FFFF	S/NS
AACI, PL041	0x00_1C04_0000	64KB	0x00_1C04_FFFF	S/NS
MCI, PL180	0x00_1C05_0000	64KB	0x00_1C05_FFFF	S/NS
KMI - Keyboard, PL050	0x00_1C06_0000	64KB	0x00_1C06_FFFF	S/NS
KMI - Mouse, PL050	0x00_1C07_0000	64KB	0x00_1C07_FFFF	S/NS
UART0, PL011	0x00_1C09_0000	64KB	0x00_1C09_FFFF	S/NS
UART1, PL011	0x00_1C0A_0000	64KB	0x00_1C0A_FFFF	S/NS
UART2, PL011	0x00_1C0B_0000	64KB	0x00_1C0B_FFFF	S/NS
UART3, PL011	0x00_1C0C_0000	64KB	0x00_1C0C_FFFF	S/NS
VFS2	0x00_1C0D_0000	64KB	0x00_1C0D_FFFF	S/NS
Watchdog, SP805	0x00_1C0F_0000	64KB	0x00_1C0F_FFFF	S/NS
Base Platform Power Controller	0x00_1C10_0000	64KB	0x00_1C10_FFFF	S/NS
Dual-Timer 0, SP804	0x00_1C11_0000	64KB	0x00_1C11_FFFF	S/NS
Dual-Timer 1, SP804	0x00_1C12_0000	64KB	0x00_1C12_FFFF	S/NS
Virtio block device	0x00_1C13_0000	64KB	0x00_1C13_FFFF	S/NS
Real-time Clock, PL031	0x00_1C17_0000	64KB	0x00_1C17_FFFF	S/NS
CF Card, unimplemented	0x00_1C1A_0000	64KB	0x00_1C1A_FFFF	S/NS
Color LCD Controller, PL111	0x00_1C1F_0000	64KB	0x00_1C1F_FFFF	S/NS
Non-trusted ROM, nontrustedrom	0x00_1F00_0000	4KB	0x00_1F00_0FFF	S/NS
CoreSight and peripherals	0x00_2000_0000	128MB	0x00_27FF_FFFF	S/NS
REFCLK CNTControl, Generic Timer	0x00_2A43_0000	64KB	0x00_2A43_FFFF	S
EL2 Generic Watchdog Control	0x00_2A44_0000	64KB	0x00_2A44_FFFF	S/NS
EL2 Generic Watchdog Refresh	0x00_2A45_0000	64KB	0x00_2A45_FFFF	S/NS
Trusted Watchdog, SP805	0x00_2A49_0000	64KB	0x00_2A49_FFFF	S
TrustZone Address Space Controller, TZC-400	0x00_2A4A_0000	64KB	0x00_2A4A_FFFF	S
REFCLK CNTRead, Generic Timer	0x00_2A80_0000	64KB	0x00_2A80_FFFF	S/NS
AP_REFCLK CNTCTL, Generic Timer	0x00_2A81_0000	64KB	0x00_2A81_FFFF	S
AP_REFCLK CNTBase0, Generic Timer	0x00_2A82_0000	64KB	0x00_2A82_FFFF	S
AP_REFCLK CNTBase1, Generic Timer	0x00_2A83_0000	64KB	0x00_2A83_FFFF	S/NS
DMC-400 CFG, unimplemented	0x00_2B0A_0000	64KB	0x00_2B0A_FFFF	S/NS
GIC Physical CPU interface, GICC[cy]	0x00_2C00_0000	8KB	0x00_2C00_1FFF	S/NS

Table 5-3 Base Platform memory map (continued)

Peripheral	Start address	Size	End address	Security
GIC Virtual Interface Control, GICH[cy]	0x00_2C01_0000	4KB	0x00_2C01_0FFF	S/NS
GIC Virtual CPU Interface, GICV[cy]	0x00_2C02_F000	8KB	0x00_2C03_0FFF	S/NS
CCI-400	0x00_2C09_0000	64KB	0x00_2C09_FFFF	S/NS
Non-trusted SRAM	0x00_2E00_0000	64KB	0x00_2E00_FFFF	S/NS
GICv3 Distributor GICD[cy]	0x00_2F00_0000	64KB	0x00_2F00_FFFF	S/NS
GICv3 Distributor ITS[cy]	0x00_2F02_0000	128KB	0x00_2F03_FFFF	S/NS
GICv3 Distributor GICR[cy]	0x00_2F10_0000	1MB	0x00_2F1F_FFFF	S/NS
Trusted Random Number Generator	0x00_7FE6_0000	4KB	0x00_7FE6_0FFF	S
Trusted Non-volatile counters	0x00_7FE7_0000	4KB	0x00_7FE7_0FFF	S
Trusted Root-Key Storage	0x00_7FE8_0000	4KB	0x00_7FE8_0FFF	S
DDR3 PHY, unimplemented	0x00_7FEF_0000	64KB	0x00_7FEF_FFFF	S/NS
HD LCD Controller, PL370	0x00_7FF6_0000	64KB	0x00_7FF6_FFFF	S/NS
DRAM, 0GB-2GB	0x00_8000_0000	2GB	0x00_FFFF_FFFF	P
DRAM, 2GB-32GB	0x08_8000_0000	30GB	0x0F_FFFF_FFFF	P
DRAM, 32GB-512GB	0x88_0000_0000	480GB	0xFF_FFFF_FFFF	P

Related references

3.2.2 ARMAEMv8AMPCT component on page 3-75.
3.2.4 ARMCortexA53xnCT component on page 3-187.
3.2.3 ARMCortexA57xnCT component on page 3-88.
4.4.7 CCI400 component on page 4-291.
4.3.2 ClockDivider component on page 4-282.
5.6.3 Base - DebugAccessPort component on page 5-431.
4.4.16 ElfLoader component on page 4-305.
4.4.17 FlashLoader component on page 4-305.
4.4.18 GenericTimer component on page 4-306.
4.4.19 GIC_400 component on page 4-307.
4.4.21 GICv3Distributor component on page 4-312.
4.4.22 HostBridge component on page 4-321.
4.4.24 IntelStrataFlashJ3 component on page 4-324.
4.5.11 Labeller and LabellerForDMA330 components on page 4-407.
4.4.28 MMC component on page 4-330.
4.4.32 PL011_Uart component on page 4-340.
4.4.35 PL031_RTC component on page 4-344.
4.4.36 PL041_AACI component on page 4-345.
4.4.37 PL050_KMI component on page 4-348.
4.4.41 PL111_CLCD component on page 4-355.
4.4.42 PL180_MCI component on page 4-356.
5.6.2 Base - Base_PowerController component on page 5-427.
4.5.5 PVBusDecoder component on page 4-401.

[cy] You can configure the address of this region using parameters to the model. See *GICv3Distributor - parameters* on page 4-313.

4.4.56 SMSC_91C111 component on page 4-383.
4.4.57 SP804_Timer component on page 4-386.
4.4.58 SP805_Watchdog component on page 4-387.
4.4.59 SP810_SysCtrl component on page 4-387.
4.4.61 TZC_400 component on page 4-391.
4.4.64 v8EmbeddedCrossTrigger_Matrix component on page 4-394.
4.4.65 VFS2 component on page 4-395.
4.4.66 VirtioBlockDevice component on page 4-397.

5.2.3 Base - DRAM

The multiple DRAM regions do not alias each other and form a contiguous 512GB area. The total amount of DRAM on the Base Platform system model is configurable. This ability affects where usable DRAM appears.

If the Base Platform system model has `bp.dram_size=4`, the default, then 2GB of DRAM is accessible at `0x00_8000_0000` to `0x00_FFFF_FFFF`, and the remaining 2GB is accessible at `0x08_8000_0000` to `0x08_FFFF_FFFF`.

If, instead, the Base Platform system model has `bp.dram_size=8`, then 2GB of DRAM is accessible at `0x00_8000_0000` to `0x00_FFFF_FFFF` and the remaining 6GB is accessible at `0x08_8000_0000` to `0x09_FFFF_FFFF`.

The default contents of RAM not otherwise written by the simulation is a repeating sequence of the following 64-bit value: `0xCFDFDFDFDFDFDFCF`.

5.3 Base - interrupt assignments

The platform assigns the *Shared Peripheral Interrupts* (SPIs) and *Private Peripheral Interrupts* (PPIs) on the GIC.

Note

SPI and PPI numbers are mapped onto GIC interrupt IDs as the *ARM Generic Interrupt Controller Specification* describes.

Table 5-4 SPI GIC assignments

IRQ ID	SPI offset	Device
32	0	Watchdog, SP805
34	2	Dual-Timer 0, SP804
35	3	Dual-Timer 1, SP804
36	4	Real-time Clock, PL031
37	5	UART0, PL011
38	6	UART1, PL011
39	7	UART2, PL011
40	8	UART3, PL011
41	9	MCI, PL180, MCIINTR0
42	10	MCI, PL180, MCIINTR1
43	11	AACI, PL041
44	12	KMI - Keyboard, PL050
45	13	KMI - Mouse, PL050
46	14	Color LCD Controller, PL111
47	15	Ethernet, SMSC 91C111
56	24	Trusted Watchdog, SP085
57	25	AP_REFCLK, Generic Timer, CNTPSIRQ
58	26	AP_REFCLK, Generic Timer, CNTPSIRQ1
59	27	EL2 Generic Watchdog WS0
60	28	EL2 Generic Watchdog WS1
73	41	VFS2
74	42	Virtio block device
80	48	TZC-400 interrupt
92	60	cluster0.cpu0 PMUIRQ
93	61	cluster0.cpu1 PMUIRQ
94	62	cluster0.cpu2 PMUIRQ
95	63	cluster0.cpu3 PMUIRQ

Table 5-4 SPI GIC assignments (continued)

IRQ ID	SPI offset	Device
96	64	cluster1.cpu0 PMUIRQ
97	65	cluster1.cpu1 PMUIRQ
98	66	cluster1.cpu2 PMUIRQ
99	67	cluster1.cpu3 PMUIRQ
117	85	HD LCD Controller, PL370
139	107	Trusted Random Number Generator

Table 5-5 PPI GIC assignments

IRQ ID	PPI offset	Device
22	6	DCC, comms channel, interrupt
23	7	PMU, performance counter, overflow
24	8	CTI, Cross Trigger Interface, interrupt
25	9	Virtual CPU interface maintenance interrupt
26	10	Hypervisor timer interrupt
27	11	Virtual timer interrupt
29	13	Secure physical timer interrupt
30	14	Non-secure physical timer interrupt

5.4 Base - clocks

This section describes the clock frequencies of the Base Platform peripherals.

Table 5-6 Peripheral clock frequencies in the Base Platform

Device	Clock
Clusters	100MHz
REFCLK CNTControl, Generic Timer	100MHz
AP_REFCLK CNTCTL, Generic Timer	100MHz
Dual-Timer 0-1, SP804	35MHz
VE system registers	24MHz
UART 0-3, PL011	24MHz
KMI 0-1, PL050	24MHz
MCI, PL180	24MHz
AACI, PL041	24MHz
Ethernet, SMSC 91C111	24MHz
Watchdog, SP805	24MHz
Color LCD Controller, PL111	23.75MHz
HD LCD Controller, PL370	10MHz
Trusted Watchdog, SP805	32.768kHz
Real-time Clock, PL031	1Hz

5.5 Base - parameters

This section describes the parameters.

Table 5-7 Base Platform parameters

Parameter	Type	Allowed values	Default value	Description
`bp.dram_size`	`int`	`0x2-0x8`	`0x4`	Size of main memory in gigabytes: 2, 4, or 8.
`bp.proc_id`*n*[cz]	`uint32_t`	`-`	-[da]	Processor ID for VE_SysRegs SYS_PROCID*n*.
`bp.secure_memory`	`bool`	`true`, `false`	`true`	The security state of the processor limits access to peripherals and RAM.
`bp.variant`[cz]	`uint32_t`	`0x0-0xF`	-[da]	Board variant for VE_SysRegs SYS_ID.
`cache_state_modelled`	`bool`	`true`, `false`	`true`	Enable d-cache and i-cache state for all components.

Table 5-8 Base Platform debug parameters

Parameter	Type	Allowed values	Default value	Description
`dbgen`	`bool`	`true`, `false`	`true`	Debug authentication signal, **dbgen**.
`niden`	`bool`	`true`, `false`	`true`	Debug authentication signal, **niden**.
`spiden`	`bool`	`true`, `false`	`true`	Debug authentication signal, **spiden**.
`spniden`	`bool`	`true`, `false`	`true`	Debug authentication signal, **spniden**.

Related references

VE_SysRegs - parameters on page 5-432.

[cz] Some platforms do not expose this parameter.
[da] Platform specific.

5.6 Base - components

This section describes the components.

This section contains the following subsections:

5.6.1 Base - components - about

These component models implement some of the functionality of the *Versatile Express* (VE) hardware.

A complete model implementation of a Base Platform system model includes both Base Platform-specific components and generic components such as buses and timers.

5.6.2 Base - Base_PowerController component

This section describes the Base_PowerController component.

Base_PowerController - control interface

The Base_PowerController provides a basic register interface for software to control the power-up and power-down of cores in the cluster.

Identify cores in the system to the Base_PowerController by writing 24 bits in MPIDR format, providing the following levels of affinity:

Bits [23:16]
Affinity level 2.

Bits [15:8]
Affinity level 1.

Bits [7:0]
Affinity level 0.

Examples of affinity usage are `not_applicable`/`cluster`/`processor` and `cluster`/`processor`/`thread`.

Base_PowerController - parameters

This section describes the parameters.

Table 5-9 Base_PowerController parameters

Parameter	Allowed values	Default value	Description
`startup`	-	'0.0.0.*'	A comma-separated list of cores to power up at startup or system reset. Specify core affinities with a dotted-quad ('0.0.0.0' refers to cluster0.cpu0 and '0.0.1.1' refers to cluster1.cpu1). Use wildcards to indicate all cores at an affinity level ('0.0.0.*' indicates all cores in cluster 0 and is equivalent to '0.0.0.0,0.0.0.1,0.0.0.2,…,0.0.0.255').

Base_PowerController - registers

This section describes the registers.

Register summary

This section describes the power control registers in order of offset from the base memory address.

Table 5-10 Base_PowerController register summary

Offset	Name	Type	Reset	Width	Description
0x00	PPOFFR	RW	0x---	32	Power Control Processor Off Register
0x04	PPONR	RW	0x---	32	Power Control Processor On Register
0x08	PCOFFR	RW	0x---	32	Power Control Cluster Off Register
0x0C	PWKUPR	RW	0x---	32	Power Control Wakeup Register
0x10	PSYSR	RW	0x---	32	Power Control SYS Status Register

PPOFFR

The *Power Control Processor Off Register* (PPOFFR) characteristics are: purpose, usage constraints, configurations, and attributes.

Purpose

Processor SUSPEND command when PWKUPR and the GIC are programmed appropriately to provide wakeup events from IRQ and FIQ events to that processor.

Usage constraints

Processor must make power-off requests only for itself.

Configurations

Available in all configurations.

Attributes

See the register summary table.

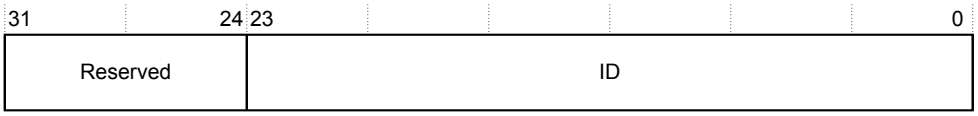

Figure 5-1 Power Control Processor Off Register bit assignments

Table 5-11 Power Control Processor Off Register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	ID	MPIDR format affinity value of the processor to be switched off. Programming error if MPIDR != self.

PPONR

The *Power Control Processor On Register* (PPONR) characteristics are: purpose, usage constraints, configurations, and attributes.

Purpose

Brings up a processor from low-power mode.

Usage constraints

Processor must make power-on requests only for other powered-off processors in the system.

Configurations

Available in all configurations.

Attributes

See the register summary table.

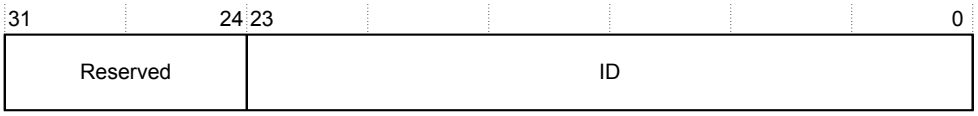

Figure 5-2 Power Control Processor On Register bit assignments

Table 5-12 Power Control Processor On Register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	ID	MPIDR format affinity value of the processor to be switched on. Programming error if MPIDR == self.

PCOFFR

The *Power Control Cluster Off Register* (PCOFFR) characteristics are: purpose, usage constraints, configurations, and attributes.

Purpose
Turns the cluster off.

Usage constraints
Cluster must make power-off requests only for itself.

Configurations
Available in all configurations.

Attributes
See the register summary table.

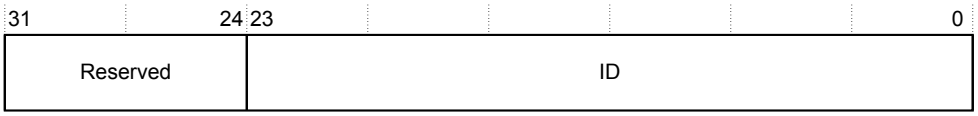

Figure 5-3 Power Control Cluster Off Register bit assignments

Table 5-13 Power Control Cluster Off Register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	ID	MPIDR format affinity value of powered-on processor in the cluster to be switched off. Programming error if MPIDR != self.

PWKUPR

The *Power Control Wakeup Register* (PWKUPR) characteristics are: purpose, usage constraints, configurations, and attributes.

Purpose
Configures whether wakeup requests from the GIC are enabled for this cluster.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes
See the register summary table.

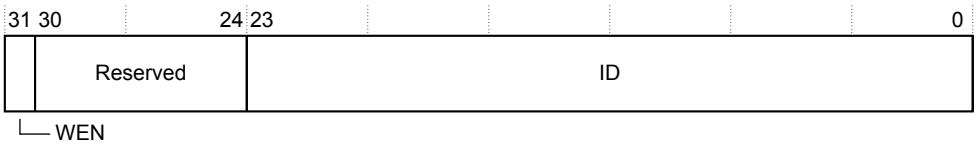

Figure 5-4 Power Control Wakeup Register bit assignments

Table 5-14 Power Control Wakeup Register bit assignments

Bits	Name	Function
[31]	WEN	If set, enables wakeup interrupts (return from SUSPEND) for this cluster.
[30:24]	-	Reserved.
[23:0]	ID	MPIDR format affinity value of processor whose Wakeup Enable bit is to be configured.

PSYSR

The *Power Control SYS Status Register* (PSYSR) characteristics are: purpose, usage constraints, configurations, and attributes.

Purpose

Provides information on the powered status of a given core. Software writes bits [23:0] for the required core and reads the value along with the associated status in bits [31:24].

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

See the register summary table.

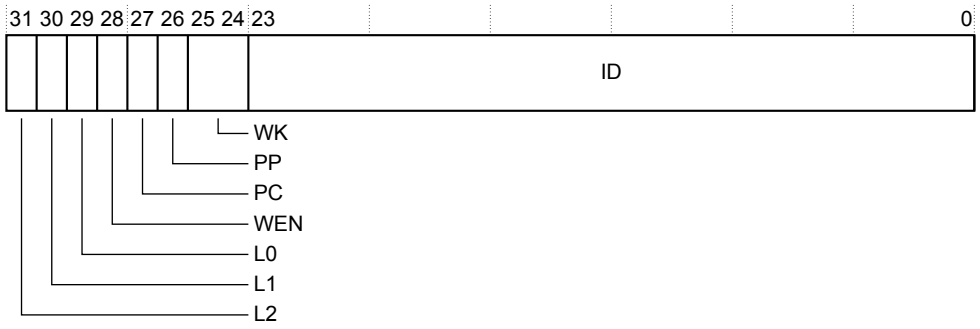

Figure 5-5 Power Control SYS Status Register bit assignments

Table 5-15 Power Control SYS Status Register bit assignments

Bits	Name	Function
[31]	L2	Read-only. A value of 1 indicates that affinity level 2 is active/on. If affinity level 2 is not implemented this bit is RAZ.
[30]	L1	Read-only. A value of 1 indicates that affinity level 1 is active/on. If affinity level 1 is not implemented this bit is RAZ.
[29]	L0	Read-only. A value of 1 indicates that affinity level 0 is active/on.
[28]	WEN	Read-only. A value of 1 indicates wakeup interrupts, return from SUSPEND, enabled for this processor. This is an alias of PWKUPR.WEN for this core.
[27]	PC	Read-only. A value of 1 indicates pending cluster off, the cluster enters low-power mode the next time it raises signal STANDBYWFIL2.

Table 5-15 Power Control SYS Status Register bit assignments (continued)

Bits	Name	Function
[26]	PP	Read-only. A value of 1 indicates pending processor off, the processor enters low-power mode the next time it raises signal STANDBYWFI.
[25:24]	WK	Read-only. Indicates the reason for LEVEL0 power on: `0b00` Cold power-on. `0b01` System reset pin. `0b10` Wake by PPONR. `0b11` Wake by GIC WakeRequest signal.
[23:0]	ID	MPIDR format affinity value.

5.6.3 Base - DebugAccessPort component

This section describes the DebugAccessPort component, a model of the *Debug Access Port* (DAP) for external debug connections.

DebugAccessPort - ports

This section describes the ports.

Table 5-16 DebugAccessPort ports

Name	Protocol	Type	Description
`ap_pvbus_m[2]`	PVBus	Master	Debug-access port to bus master channels 0 and 1.
`clock`	ClockSignal	Slave	Clock input.
`paddrdbg31`	Signal	Master	Output signal that indicates which master the access came from, AP0 or AP1. Configurable.

DebugAccessPort - parameters

This section describes the parameters.

Table 5-17 Base Platform DebugAccessPort parameters

Name	Type	Allowed values	Default value	Description
`ap0_rom_base_address`	`uint64_t`	`0x0-0xffffffffffffffff`	`0x0`	ROM base address for AP0.
`ap1_rom_base_address`	`uint64_t`	`0x0-0xffffffffffffffff`	`0x0`	ROM base address for AP1.
`ap0_has_debug_rom`	`bool`	`true`, `false`	`false`	AP0 has a Debug ROM.
`ap1_has_debug_rom`	`bool`	`true`, `false`	`false`	AP1 has a Debug ROM.
`ap0_set_paddrdbg31`	`bool`	`true`, `false`	`false`	Set `paddrdbg31` signal during accesses on AP0.
`ap1_set_paddrdbg31`	`bool`	`true`, `false`	`false`	Set `paddrdbg31` signal during accesses on AP1.

5.6.4 Base - simulator visualization component

This section describes the simulator visualization component.

Simulator visualization - parameters

This section describes the parameters.

Table 5-18 Simulator visualization parameters

Parameter	Allowed values	Default value	Description
`cluster0_name`	-	"Cluster0"	Cluster0 name.
`cluster1_name`	-	"Cluster1"	Cluster1 name.
`cpu_name`	-	""	Window title displays core name.
`disable_visualisation`	`true`, `false`	`false`	Enables or disables visualization.
`rate_limit-enable`	`true`, `false`	`true`	Rate limit simulation.
`recorder.checkInstructionCount`	`true`, `false`	`true`	Checks instruction count in recording file against actual instruction count during playback.
`recorder.playbackFileName`	-	""	Playback filename. An empty string disables playback.
`recorder.recordingFileName`	-	""	Recording filename. An empty string disables recording.
`recorder.recordingTimeBase`	-	`0x5F5E100`	Timebase in 1/s relative to the master clock. For example, 100 000 000 means 10 nanoseconds resolution simulated time for a 1Hz master clock. Used for recording. Higher values give a higher time resolution. Playback time base is always taken from the playback file.
`recorder.verbose`	-	`0x0`	Enables verbose messages. `0x0`, no messages. `0x1`, normal. `0x2`, more detail.
`trap_key`	-	`0x4a`	Trap key that works with left **`Ctrl`** to toggle mouse display.
`window_title`	-	"Fast Models - CLCD %cpu%"	`cpu_name` replaces window title, `%cpu%`.

5.6.5 Base - VE_SysRegs component

This section describes the VE system registers component.

VE_SysRegs - parameters

This section describes the parameters.

Table 5-19 Base Platform VE_SysRegs parameters

Name	Type	Allowed values	Default value	Description
exit_on_shutdown	bool	true, false	false	true: if software uses the SYS_CFGCTRL function SYS_CFG_SHUTDOWN, then the simulator shuts down and exits.[db]
mmbSiteDefault	uint8_t	0x0-0x2	0x1	Default MMB source. 0x0, MB. 0x1, DB1. 0x2, DB2.
tilePresent	bool	true, false	true	Tile fitted.
user_switches_value	uint32_t	-	0x0	User switches.

Related references

5.5 Base - parameters on page 5-426.

VE_SysRegs - registers

This section describes the configuration registers.

Table 5-20 Base Platform VE_SysRegs registers

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.
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_PROCID2	0x8C	Read/write	Processor ID.
SYS_PROCID3	0x90	Read/write	Processor ID.
SYS_CFGDATA	0xA0	Read/write	Data to read/write 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 controller.

[db] For more information on SYS_CFGCTRL, see the *Motherboard Express µATX V2M-P1 Technical Reference Manual*.

5.7 Base - differences between the AEMv8-A FVP and core FVPs

This section describes implementation features of the core models that the AEMv8-A model does not implement, or implements with significant differences.

- The default value of `cache_state_modelled` is 0.
- Components `cluster0` and `cluster1` are implementation cores, not AEMs. All parameters in these components are the parameters of the core implementation, not the parameters of the AEM. The values of `bp.proc_id0` and `bp.proc_id1` have fixed values consistent with the cores and are not configurable.
- The core defines the memory map of register banks within the GIC region, and the map is therefore not configurable. The parameter `bp.variant` communicates the nature of the memory map to target software. The parameter has a fixed value consistent with the memory map of the core, and is not configurable. Because the core implementations contain a specific version of the GIC, the parameter `gicv3.gicv2-only` is not available. The following registers in the GIC distributor are given fixed values to match the implementation, and are not configurable: `gic_distributor.reg-base`, `gic_distributor.reg-base-per-distributor`, `gic_distributor.GICD-alias`, `gic_distributor.ITS0-base`.
- The GICv2 CPU IDs are contiguous in implementation platform models, because the number of cores in each cluster is fixed. They can be noncontiguous in the AEMv8-A Base Platform FVP because there is space for four cores in the cluster. You can configure fewer than four cores.

5.8 Base - VE compatibility

ARM expects software that ran on the previous VE model to be compatible with this system model, but you might need to apply some configuration options.

This section contains the following subsections:

5.8.1 Base - VE compatibility - GICv2

This system model uses GICv3 by default. You can configure it to support GICv2 or GICv2m.

To configure the model as GICv2m, set the following:

```
-C gicv3.gicv2-only=1                          \
-C cluster0.gic.GICD-offset=0x10000            \
-C cluster0.gic.GICC-offset=0x2F000            \
-C cluster0.gic.GICH-offset=0x4F000            \
-C cluster0.gic.GICH-other-CPU-offset=0x50000  \
-C cluster0.gic.GICV-offset=0x6F000            \
-C cluster0.gic.PERIPH-size=0x80000            \
-C cluster1.gic.GICD-offset=0x10000            \
-C cluster1.gic.GICC-offset=0x2F000            \
-C cluster1.gic.GICH-offset=0x4F000            \
-C cluster1.gic.GICH-other-CPU-offset=0x50000  \
-C cluster1.gic.GICV-offset=0x6F000            \
-C cluster1.gic.PERIPH-size=0x80000            \
-C gic_distributor.GICD-alias=0x2c010000
```

To configure the model as GICv2, set the following:

```
-C gicv3.gicv2-only=1                         \
-C cluster0.gic.GICD-offset=0x1000            \
-C cluster0.gic.GICC-offset=0x2000            \
-C cluster0.gic.GICH-offset=0x4000            \
-C cluster0.gic.GICH-other-CPU-offset=0x5000  \
-C cluster0.gic.GICV-offset=0x6000            \
-C cluster0.gic.PERIPH-size=0x8000            \
-C cluster1.gic.GICD-offset=0x1000            \
-C cluster1.gic.GICC-offset=0x2000            \
-C cluster1.gic.GICH-offset=0x4000            \
-C cluster1.gic.GICH-other-CPU-offset=0x5000  \
-C cluster1.gic.GICV-offset=0x6000            \
-C cluster1.gic.PERIPH-size=0x8000            \
-C gic_distributor.GICD-alias=0x2c010000
```

To configure MSI frames for GICv2m, parameters are available to set the base address and configuration of each of 16 possible frames. Eight frames are Secure and eight frames are Non-secure:

```
-C gic_distributor.MSI_S-frame0-base=ADDRESS  \
-C gic_distributor.MSI_S-frame0-min-SPI=NUM   \
-C gic_distributor.MSI_S-frame0-max-SPI=NUM
```

In this example, you can replace `MSI_S` with MSI_NS, for NS frames, and you can replace `frame0` with `frame1` to `frame7` for each of the possible 16 frames. If the base address is not specified for a given frame, or the SPI numbers are out of range, the corresponding frame is not instantiated.

5.8.2 Base - VE compatibility - GICv3

If a Base Platform includes an implementation of the GICv3 system registers, it is enabled by default.

The GIC distributor and CPU (core) interface have parameters that allow configuration of the model to match different implementation options. Use `--list-params` to get a full list. Configuration options for the GIC model must be available under:

- `cluster[0-`*n*`].gic.*`
- `cluster[0-`*n*`].gicv3.*`
- `gic_distributor.*`

5.8.3 Base - VE compatibility - system global counter

The Generic Timer registers of the cores do not operate by default.

The model provides a memory-mapped interface to the system global counter, and enables the free-running timer from reset. However, the architectural requirement is that such a counter is not enabled at reset. As a result, the Generic Timer registers of the cores do not operate unless either:

- Software enables the counter peripheral by writing the FCREQ[0] and EN bits in CNTCR at `0x2a43000`. ARM recommends this approach.
- The `-C bp.refcounter.non_arch_start_at_default=1` parameter is set. This approach provides compatibility with older software.

5.8.4 Base - VE compatibility - disable security

Base Platform FVPs have an enhanced security map for peripherals. By default, it restricts access to some peripherals.

Software must program the TZC-400 to make any accesses to DRAM, because the reset configuration blocks all accesses.

For backward compatibility with software that cannot program the TZC-400, this parameter setting permits all accesses regardless of security state:

```
-C bp.secure_memory=false
```

Related references

5.2.2 Base - memory map on page 5-419.

5.9 Base - unsupported VE features

This system model does not support software that relies on some features of the VE model.

This section contains the following subsections:

5.9.1 Base - unsupported VE features - memory aliasing at 0x08_00000000

The VE model permits an alias of the 2GB region of DRAM between addresses `0x80000000` and `0xFFFFFFFF` with addresses `0x08_00000000` to `0x08_7FFFFFF`. The Base Platform does not have this alias and the region `0x08_00000000` to `0x08_7FFFFFF` is Reserved.

5.9.2 Base - unsupported VE features - boot ROM alias at 0x00_0800_0000

In the VE model, the region at `0x00_0800_0000` was an alias of the trusted boot ROM at `0x00_0000_0000`. It is now an independent region of NOR flash.

5.9.3 Base - unsupported VE features - change of older parameters

Most parameter names have been simplified between the VE model and the Base Platform system model.

Components that were previously in *motherboard* or *daughterboard* groups are now in a *bp* group. The model does not recognize the previous parameter names.

In a change to the previous default, the Base Platform models the core cache state by default. You can disable this using a single parameter for all cores in the simulation, using the `cache_state_modelled` parameter.

```
-C cache_state_modelled=0
```

Note

Cortex Base Platforms do not model the cache state by default.

Chapter 6
Microcontroller Prototyping System 2

This chapter describes the model of the hardware platform.

It contains the following sections:

6.1 MPS2 - about

The *Microcontroller Prototyping System 2* (MPS2) *Fixed Virtual Platform* (FVP) model implements a subset of the functionality of the MPS2 hardware.

MPS2 model platforms include MPS2 components and generic ones, such as buses and timers.

MPS2 platforms are sufficiently accurate to boot the Keil® RTX RTOS and run the Blinky application.

To list the model parameters and their types, allowed values, default values, and descriptions, run the model, passing in the `--list-params` argument.

On Windows, the MPS2 components are in the `%PVLIB_HOME%\examples\FVP_MPS2\LISA` directory.

On Linux, the MPS2 components are in the `$PVLIB_HOME/examples/FVP_MPS2/LISA` directory.

Related information

AN400 - ARM Cortex-M7 SMM on V2M-MPS2.

6.2 MPS2 - memory maps

This section describes the MPS2 memory maps.

This section contains the following subsections:

6.2.1 MPS2 - memory map for models without the ARMv8-M additions

This section describes the MPS2 memory map for older cores, without the ARMv8-M additions.

For standard ARM peripherals, see the TRM for that device.

Note

- A bus error is generated for accesses to memory areas not shown in this table.
- Any memory device that does not occupy the total region is aliased within that region.

Table 6-1 Overview of MPS2 memory map

Description	Modeled	Address range
Ethernet[dc]	Partial	`0xA0000000-0xA000FFFF`
PSRAM (16MB)	Yes	`0x60000000-0x60FFFFFF`
VGA Image (512x128) (AHB)	Yes	`0x41100000-0x4110FFFF`
VGA Console (AHB)	Yes	`0x41000000-0x4100FFFF`
Block RAM (boot time)[dd]	Yes	`0x40200000-0x402FFFFF`
Reserved	N/A	`0x40030000-0x401FFFFF`
SCC register	Yes	`0x4002F000-0x4002FFFF`
Reserved	N/A	`0x40029000-0x4002EFFF`
FPGA System Control & I/O, APB	Yes	`0x40028000-0x40028FFF`
Reserved	N/A	`0x40025000-0x40027FFF`
Audio I2S, APB	Partial	`0x40024000-0x40024FFF`
SBCon (Audio Configuration), APB	Yes	`0x40023000-0x40023FFF`
SBCon (Touch for LCD module), APB	Partial	`0x40022000-0x40022FFF`
PL022 (SPI for LCD module), APB	Partial	`0x40021000-0x40021FFF`
PL022 (SPI), APB	Yes	`0x40020000-0x40020FFF`
CMSDK system controller	Yes	`0x4001F000-0x4001FFFF`
Reserved for extra GPIO & other AHB peripherals	N/A	`0x40012000-0x4001EFFF`
CMSDK AHB GPIO #1	Yes	`0x40011000-0x40011FFF`
CMSDK AHB GPIO #0	Yes	`0x40010000-0x40010FFF`
CMSDK APB subsystem	Yes	`0x40000000-0x4000FFFF`
Reserved	N/A	`0x20800000-0x20FFFFFF`

dc Through ahb_to_extmem16. Offset `0x0-0x0FE` for CSRs, `0x100-0x1FE` for FIFO.
dd Reserved 64KB, 16K implemented. This memory is wrapped through the region.

Table 6-1 Overview of MPS2 memory map (continued)

Description	Modeled	Address range
ZBTSRAM 2 & 3 (2x32-bit)[de]	Yes	`0x20000000-0x207FFFFF`
Reserved	N/A	`0x01010000-0x1FFFFFFF`
Reserved	N/A	`0x00800000-0x00FFFFFF`
ZBTSRAM 1 (64-bit)[df]	Yes	`0x00400000-0x007FFFFF`
ZBTSRAM 1 (64-bit)	Yes	`0x00004000-0x003FFFFF`
Mappable memory[dg]	Yes	`0x00000000-0x00003FFF`

6.2.2 MPS2 - memory map for models with the ARMv8-M additions

This section describes the MPS2 memory map for newer cores, with the ARMv8-M additions.

For standard ARM peripherals, see the TRM for that device.

Note

- A bus error is generated for accesses to memory areas not shown in this table.
- Any memory device that does not occupy the total region is aliased within that region.

Table 6-2 Overview of MPS2 memory map

Description	IDAU	Modeled	Address range
ZBTSRAM 1 (4MB) in *Non-secure* (NS) world. Reserved 8MB, only 4MB implemented. VTOR initialization value to be configurable in LAC). Second half (4MB) aliased to first half (4MB).	NS	Yes	`0x00000000-0x007FFFFF`
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	`0x00800000-0x0FFFFFFF`
ZBTSRAM 1 (4MB) in *Secure* (S) world. Reserved 8MB, only 4MB implemented. Second half (4MB) aliased to first half (4MB).	S	Yes	`0x10000000-0x107FFFFF`
Not used. Default expansion port: bus error.	S	N/A	`0x10800000-0x1FFFFFFF`
ZBTSRAM 2&3 (4MB) in NS world. Reserved 8MB, only 4MB implemented. For IoT subsystems, different cores have different memory sizes. Second half (4MB) aliased to first half (4MB).	NS	Yes	`0x20000000-0x207FFFFF`
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	`0x20800000-0x20FFFFFF`
PSRAM (16MB)	NS	Yes	`0x21000000-0x21FFFFFF`
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	`0x22000000-0x23FFFFFF`
MTB SRAM. Reserved 64KB, only 16KB implemented. Aliased to `0x0` for booting in RTL simulation.	NS	Yes	`0x24000000-0x2400FFFF`
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	`0x24010000-0x2FFFFFFF`
ZBTSRAM 2&3 (4MB) in S world. Reserved 8MB, only 4MB implemented. Second half (4MB) aliased to first half (4MB).	S	Yes	`0x30000000-0x307FFFFF`
Not used. Default expansion port: bus error.	S	N/A	`0x30800000-0x30FFFFFF`

[de] Reserved 8MB, 4MB available. The two SRAM blocks are interleaved.
[df] Wrapped. Only 4MB ZBTSRAM fitted.
[dg] When `zbt_boot_ctrl` = 0, ZBTSRAM 1 is mapped to this region. Otherwise, `Remap_ctrl` = 0 maps Block RAM and `Remap_ctrl` = 1 maps ZBTSRAM 1. The V2M-MPS2 board microcontroller controls the zbt_boot_ctrl signal. The zbt_boot_ctrl signal overrides the boot option to enable use of the ZBT RAM.

Table 6-2 Overview of MPS2 memory map (continued)

Description	IDAU	Modeled	Address range
Not used. No memory gating unit on PSRAM (16MB) path because it is shared with Ethernet control. Default expansion port: bus error.	S	N/A	0x31000000-0x31FFFFFF
Not used. Default expansion port: bus error.	S	N/A	0x30800000-0x3FFFFFFF
CMSDK APB subsystem:	NS	-	0x40000000-0x4000FFFF
Timer 0.	NS	Yes	0x40000000-0x40000FFF
Timer 1.	NS	Yes	0x40001000-0x40001FFF
Dual Timer.	NS	Yes	0x40002000-0x40002FFF
Not used.	NS	N/A	0x40003000-0x40003FFF
UART #0.	NS	Yes	0x40004000-0x40004FFF
UART #1.	NS	Yes	0x40005000-0x40005FFF
UART #2.	NS	Yes	0x40006000-0x40006FFF
Not used.	NS	N/A	0x40007000-0x40007FFF
Watchdog.	NS	Yes	0x40008000-0x40008FFF
Not used.	NS	N/A	0x40009000-0x4000F000
GPIO #0.	NS	Yes	0x40010000-0x40010FFF
GPIO #1.	NS	Yes	0x40011000-0x40011FFF
GPIO #2.	NS	Yes	0x40012000-0x40012FFF
GPIO #3.	NS	Yes	0x40013000-0x40013FFF
Default slave inside AHB peripheral subsystem. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	Yes	0x40014000-0x40017FFF
DMA Controller #0.	NS	Yes	0x40018000-0x40018FFF
DMA Controller #1.	NS	Yes	0x40019000-0x40019FFF
Default slave inside AHB peripheral subsystem. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	Yes	0x4001A000-0x40017FFF
CMSDK system controller. PMU control and remap registers unused. Only reset option (lockup reset) and rest info available.	NS	Yes	0x4001F000-0x4001FFFF
FPGA APB Subsystem (unused APB space: RAZ/WI):	NS	-	0x40020000-0x4002FFFF
PL022 (SPI).	NS	Yes	0x40020000-0x40020FFF
PL022 (SPI for LCD).	NS	Partial	0x40021000-0x40021FFF
SBCon I2C (Touch for LCD).	NS	Partial	0x40022000-0x40022FFF
SBCon I2C (Audio configuration).	NS	Yes	0x40023000-0x40023FFF
Audio I2S.	NS	Partial	0x40024000-0x40024FFF
Not used.	NS	N/A	0x40025000-0x40027FFF

Table 6-2 Overview of MPS2 memory map (continued)

Description	IDAU	Modeled	Address range
FPGA system control & I/O (LEDs, buttons...).	NS	Yes	0x40028000-0x40028FFF
Not used.	NS	N/A	0x40029000-0x4002EFFF
SCC registers.	NS	Yes	0x4002F000-0x401FFFFF
Ethernet.	NS	Partial	0x40200000-0x402FFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0x40300000-0x40FFFFFF
VGA console.	NS	Yes	0x41000000-0x4100FFFF
VGA image.	NS	Yes	0x41100000-0x4113FFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0x41140000-0x4FFFFFFF
CMSDK APB subsystem:	S	-	0x50000000-0x5000FFFF
Timer 0.	S	Yes	0x50000000-0x50000FFF
Timer 1.	S	Yes	0x50001000-0x50001FFF
Dual Timer.	S	Yes	0x50002000-0x50002FFF
Not used.	S	N/A	0x50003000-0x50003FFF
UART #0.	S	Yes	0x50004000-0x50004FFF
UART #1.	S	Yes	0x50005000-0x50005FFF
UART #2.	S	Yes	0x50006000-0x50006FFF
Not used.	S	N/A	0x50007000-0x50007FFF
Watchdog.	S	Yes	0x50008000-0x50008FFF
Not used.	S	N/A	0x50009000-0x5000F000
GPIO #0	S	Yes	0x50010000-0x50010FFF
GPIO #1	S	Yes	0x50011000-0x50011FFF
GPIO #2	S	Yes	0x50012000-0x50012FFF
GPIO #3	S	Yes	0x50013000-0x50013FFF
Default slave. Default expansion port: bus error.	S	Yes	0x50014000-0x50017FFF
DMA Controller #0.	S	Yes	0x50018000-0x50018FFF
DMA Controller #1.	S	Yes	0x50019000-0x50019FFF
Default slave. Default expansion port: bus error.	S	Yes	0x5001A000-0x5001EFFF
CMSDK system controller. PMU control and remap registers unused. Only reset option (lockup reset) and rest info available.	S	Yes	0x5001F000-0x5001FFFF
FPGA APB subsystem:	S	-	0x50020000-0x5002FFFF
PL022 (SPI).	S	Yes	0x50020000-0x50020FFF
PL022 (SPI for LCD).	S	Partial	0x50021000-0x50021FFF

Table 6-2 Overview of MPS2 memory map (continued)

Description	IDAU	Modeled	Address range
SBCon I2C (touch for LCD).	S	Partial	0x50022000-0x50022FFF
SBCon I2C (audio configuration).	S	Yes	0x50023000-0x50023FFF
Audio I2S.	S	Partial	0x50024000-0x50024FFF
Not used.	S	N/A	0x50025000-0x50027FFF
FPGA system control & I/O (LEDs, buttons...).	S	Yes	0x50028000-0x50028FFF
Not used.	S	N/A	0x50029000-0x5002EFFF
SCC registers.	S	Yes	0x5002F000-0x501FFFFF
Ethernet.	S	Partial	0x50200000-0x502FFFFF
Not used. Default expansion port: bus error.	S	N/A	0x50300000-0x50FFFFFF
VGA console.	S	Yes	0x51000000-0x5100FFFF
VGA image.	S	Yes	0x51100000-0x5113FFFF
Not used. Default expansion port: bus error.	S	N/A	0x51140000-0x57FFFFFS
Secure APB subsystem:	S	-	0x58000000-0x5800FFFF
TRNG / PRNG.	S	Yes	0x58000000-0x58000FFF
Unique ID or Secure storage.	S	Yes	0x58001000-0x58006FFF
Secure Control Registers.	S	Yes	0x58007000-0x58007FFF
Flash memory gating unit configuration (mapped to AHB port for CODE region in the bus matrix, not APB).	S	Yes	0x58008000-0x58009FFF
SRAM memory gating unit configuration (mapped to AHB port for SRAM region in the bus matrix, not APB).	S	Yes	0x5800A000-0x5800DFFF
Reserved.	S	N/A	0x5800E000-0x5800EFFF
Reserved.	S	N/A	0x5800F000-0x5800FFFF
Not used. Default expansion port: bus error.	S	N/A	0x50010000-0x5FFFFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0x60000000-0x6FFFFFFF
Not used. Default expansion port: bus error.	S	N/A	0x70000000-0x7FFFFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0x800000000-0x8FFFFFFF
Not used. Default expansion port: bus error.	S	N/A	0x90000000-0x9FFFFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0xA0000000-0xAFFFFFFF
Not used. Default expansion port: bus error.	S	N/A	0xB0000000-0xBFFFFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0xC0000000-0xCFFFFFFF
Not used. Default expansion port: bus error.	S	N/A	0xD0000000-0xDFFFFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0xE0000000-0xEFFFFFFF

Table 6-2 Overview of MPS2 memory map (continued)

Description	IDAU	Modeled	Address range
System ROM table. Exempted from checking.	Exempt	Yes	0xF0000000-0xF0000FFF
Not used. Default expansion port: bus error.	Exempt	N/A	0xF0001000-0xF00FFFFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0xF0100000-0xF01FFFFF
MTB SFR address space.	NS	Yes	0xF0200000-0xF0200FFF
Reserved. This region is nonexecutable.	NS	N/A	0xF0210000-0xF0213FFF
Not used. Default expansion port (MPS2 AHB subsystem): bus error or RAZ/WI.	NS	N/A	0xF0214000-0xFFFFFFFF

6.3 MPS2 - interrupt assignments

This section describes the interrupt assignments.

Table 6-3 Interrupt assignments

Number	Interrupt
NMI	Watchdog.
0	UART 0 receive interrupt.
1	UART 0 transmit interrupt.
2	UART 1 receive interrupt.
3	UART 1 transmit interrupt.
4	UART 2 receive interrupt.
5	UART 2 transmit interrupt.
6	GPIO 0, 2 combined interrupt.
7	GPIO 1, 3 combined interrupt.
8	Timer 0.
9	Timer 1.
10	Dual Timer.
11	SPI #1 (LCD). The LCD had shared SPI #0 and SPI #1.
12	UART overflow (0, 1, 2).
13	Ethernet.
14	Audio I2S.
15	Touch screen.
16-31	GPIO 0 individual interrupts.
32-47	GPIO 1 individual interrupts. ARMv8-M additions.
48	SPI #0. ARMv8-M addition.
49	Reserved.
50	TRNG (Secure). ARMv8-M addition.
51	Unique ID and Secure storage (Secure). ARMv8-M addition.
52	DMA controller #0.
53	DMA controller #1.
54	SecureErrorIRQ. ARMv8-M addition.[dh]

[dh] Detection of Non-secure access to Secure address spaces (including other bus masters). Generated by Memory Gating unit, Peripheral Gating units, bus gasket for legacy bus masters.

6.4 MPS2 - differences between models and hardware

This section describes the features of the hardware that the models do not implement, or implement with significant differences.

MPS2 implements most devices. Some peripherals have minimal implementations:

- The Ethernet module in the model is a LAN91C111. The hardware documents specify a LAN9220.
- The Audio module is RAZ/WI.
- The STMPE811 touchscreen module only reports touch positions.
- The model of the Ampire LCD module supports a subset of the graphics modes.

You can display images and text on an emulated VGA output, images on the LCD, and text on the UART.

ARMv8-M

The model implements an *Implementation Defined Attribution Unit* (IDAU) inside each core. The top-level component coordinates the IDAU implementations by passing down parameters. In contrast, the MPS2 specification [ARMv8-M MPS2 System Specification (ARM-ECM-0468897), v0.8] has a common, system level IDAU, which all cores and various devices use.

You cannot reprogram the IDAUs of the model. The model only reads the Secure Controller Register Block register, NSC_CFG. Set it using the top-level parameters, NSC_CFG_0 and NSC_CFG_1 (corresponding to bits 0 and 1 of NSC_CFG, respectively).

The model does not have the random number generator or unique ID/secure storage of the MPS2 specification.

The model does not support MTB, ETM, and TPIU. MTB RAM is absent.

In the Memory Gating Unit, the model provides a configurable block size. For performance reasons, the minimum block size in the model is 4096 bytes. Hardware and later models might allow smaller block sizes. Software must use the BLK_CFG register to determine block size.

Timing

FVPs enable software applications to run 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 such as timers and keyboards, data arrives in the modeled device in real world, or wall clock, time. However, simulation time can run faster than the wall clock. So, a single key press might be interpreted as several repeated key presses, or a single mouse click might be interpreted as a double click.

To avoid this mismatch, the FVPs provide the Rate Limit feature. Enabling Rate Limit forces the model to run at wall clock time. For interactive applications, ARM recommends enabling Rate Limit. Use the Rate Limit button in the CLCD display or the `rate_limit-enable` model instantiation parameter.

Chapter 7
Versatile Express Model

This chapter describes the components of Fast Models that are specific to the *Versatile Express* (VE) *Fixed Virtual Platform* (FVP) model of the hardware platform.

It contains the following sections:

7.1 About the Versatile Express baseboard components

These components model some of the functionality of the *Versatile Express* (VE) hardware.

A complete model implementation of the VE platform includes both VE-specific components and generic ones such as buses and timers. See the hardware documentation for functionality information.

The VE components are in the `%PVLIB_HOME%\examples\FVP_VE\LISA` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_VE/LISA` directory.

Related information

Getting Started with Fixed Virtual Platforms, Fixed Virtual Platforms FVP Reference Guide.
Programmer's Reference for the VE FVPs, Fixed Virtual Platforms FVP Reference Guide.

7.2 VE memory map for Cortex-A series

The Versatile Express RS1 memory map with the RS2 extensions is the base of the global memory map for the Cortex-A series platform model.

Note

The VE FVP implementation of memory does not require the memory controller to have the correct values. If you run applications on actual hardware, ensure that the memory controller is set up properly so that they run properly.

Table 7-1 Cortex-A series platform model memory map

Name	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 *below* on page 7-451.	Yes	0x00_18000000-0x00_1BFFFFFF	64MB
Peripherals (CS3). See *below* on page 7-451.	Yes	0x00_1C000000-0x00_1FFFFFFF	64MB
CoreSight and peripherals	No	0x00_20000000-0x00_2CFFFFFF[di]	-
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)[dj]	Yes	0x00_80000000-0x00_FFFFFFFF	2GB
Unused	-	0x01_00000000-0x07_FFFFFFFF	-
4GB DRAM (in 36-bit address space)[dj]	Yes	0x08_00000000-0x08_FFFFFFFF	4GB
Unused	-	0x09_00000000-0x7F_FFFFFFFF	-
4GB DRAM (in 40-bit address space)[dj]	Yes	0x80_00000000-0xFF_FFFFFFFF	4GB

The model has a `secure_memory` option. When you enable this option, the memory map changes for a number of peripherals.

Table 7-2 CS2 region 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.

[di] The private peripheral region address `0x2c000000` is mapped in this region. You can use the parameter PERIPHBASE to map the peripherals to a different address.
[dj] 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.

Table 7-2 CS2 region peripheral memory map for secure_memory option (continued)

Peripheral	Address range	Functionality with secure_memory enabled
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 map, aborts on secure accesses.

Table 7-3 CS2 region peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[dk]
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	-

Table 7-4 CS3 region peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[dk]
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
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

[dk] Use these interrupt signal values to program your interrupt controller. They are the SPI number plus 32. Add 32 to the interrupt numbers from the peripherals to form the interrupt number that the GIC sees. GIC interrupts 0-31 are for internal use.

Table 7-4 CS3 region peripheral memory map (continued)

Peripheral	Modeled	Address range	Size	GIC Int[dk]
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	-

7.3 VE memory map for Cortex-R series

The Versatile Express RS1 memory map with the RS2 extensions is the base of the global memory map for the Cortex-R series platform model.

Table 7-5 Memory map

Memory	Modeled	Address range
DRAM	Yes	0x00000000-0x03FFFFFF
FLASH0	Yes	0x40000000-0x43FFFFFF
FLASH1	Yes	0x44000000-0x47FFFFFF
PSRAM	Yes	0x48000000-0x4BFFFFFF
Reserved (CS4)	No	0x4C000000-0x4FFFFFFF
Peripherals. See *below* on page 7-453.	Yes	0xA0000000-0x01FFFFFF
RAM	No	0x40000000-0xBBFFFFFF
NOR_FLASH	Yes	0xBE000000-0xBFFFFFFF

Table 7-6 Peripheral memory map

Peripheral	Modeled	Address range
PL111 CLCD	Yes	0xA0000000-0xA0010000
PL390 GIC CPU Interface	Yes	0xAE000000-0xAE000FFF
PL390 GIC Distributor	Yes	0xAE001000-0xAE001FFF
PL041 AACI	Yes	0xB0040000-0xB004FFFF
PL180 MCI	Yes	0xB0050000-0xB00fFFFF
PL050 KMI0	Yes	0xB0060000-0xB006FFFF
PL050 KMI1	Yes	0xB0070000-0xB007FFFF
PL011 UART0	Yes	0xB0090000-0xB009FFFF
PL011 UART1	Yes	0xB00A0000-0xB00AFFFF
PL011 UART2	Yes	0xB00B0000-0xB00BFFFF
PL011 UART3	Yes	0xB00C0000-0xB00CFFFF
VFS2	Yes	0xB00D0000-0xB00DFFFF
SP805 WATCHDOG	Yes	0xB00F0000-0xB00FFFFF
TIMER_0_1	Yes	0xB0110000-0xB011FFFF
TIMER_2_3	Yes	0xB0120000-0xB012FFFF
PL031 Real Time Clock	Yes	0xB0170000-0xB017FFFF
Compact Flash	No	0xB01A0000-0xB01AFFFF
PL011 UART4	Yes	0xB01B0000-0xB01BFFFF
RAM	No	0xB01F0000-0xB01FFFFF
USB	No	0xBD000000-0xBDFFFFFF

7.4 VE parameters

This section describes the VE FVP instantiation parameters.

This section contains the following subsections:

7.4.1 VE instantiation parameters

This section describes the instantiation parameters for VE models.

Table 7-7 VE instantiation parameters

Component	Parameter	Type	Allowed values	Default value	Description
`ve_sysregs`	`user_switches_value`	Integer	See the related reference topic.	`0`	Switch S6 setting.
`flashloader0`	`fname`	String	Valid filename	""	Path to flash image file.
`flashloader1`	`fname`	String	Valid filename	""	Path to flash image file.
`mmc`	`p_mmc_file`	String	Valid filename	`mmc.dat`	Multimedia card filename.
`pl111_clcd_0`	`pixel_double_limit`	Integer	-	`0x12c`	Sets threshold in horizontal pixels below which pixels sent to framebuffer are doubled in size in both dimensions.
`sp810_sysctrl`	`use_s8`	Boolean	`true`, `false`	`false`	Indicates whether to read `boot_switches_value`.
`vfs2`	`mount`	String	Directory path on host	""	Mount point for the host file system.

Related references

7.4.3 VE switch S6 on page 7-454.

7.4.2 VE secure memory parameters

This section describes the VE FVP secure memory parameters that you can change when you start the model.

Table 7-8 VE secure memory parameters

Name	Type	Allowed values	Default value	Description

`secure_memory`	Boolean	`true`, `false`	`false`	`false`
				The platform behaves as before.
				`true`
				The address space is segregated according to the security mode of the core. Some memory blocks near the bottom of the address space are available to Secure transactions only. The rest of the address space is available to Non-secure transactions only.

7.4.3 VE switch S6

This section describes the behavior and default positions of the VE system model switch.

Switch S6 is equivalent to the Boot Monitor configuration switch on the VE hardware.

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 7-9 Default positions of VE system model switch

Switch	Default position	Function in default position
S6-1	OFF	Displays prompt permitting Boot Monitor command entry after system start.
S6-2	OFF	See STDIO redirection, below.
S6-3	OFF	See STDIO redirection, below.
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.

Table 7-10 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 UART0, regardless of semihosting settings.

7.5 VEVisualisation component

This section describes the VEVisualisation component.

This section contains the following subsections:

7.5.1 VEVisualisation - about

This component can generate events from host mouse or keyboard use when the visualization window is in focus. For example, you can toggle the switch elements from the visualization window.

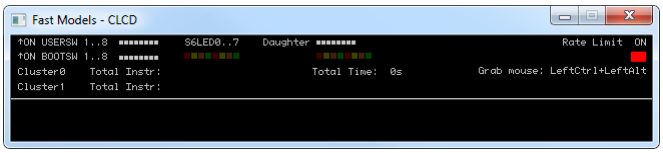

Figure 7-1 Startup VE Fixed Virtual Platform CLCD visualization window

When a suitable application or system image loads, and configures the PL110_CLCD controller registers, the window expands to show the contents of the frame buffer.

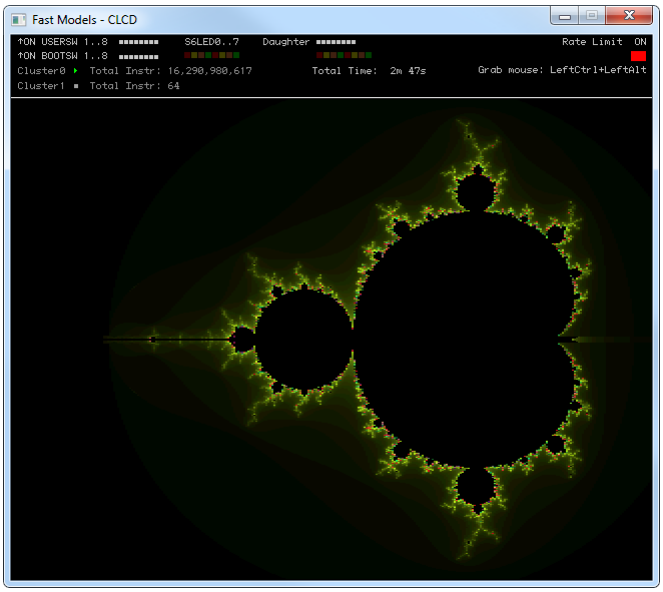

Figure 7-2 VE FVP CLCD with brot.axf image

The VEVisualisation component in System Canvas can be found in the `%PVLIB_HOME%\examples\FVP_VE\LISA\` directory. On Linux, use the `$PVLIB_HOME/examples/FVP_VE/LISA` directory.

This is a LISA+ component.

Note

Using this component can reduce simulation performance. The simulation speed is affected by the setting of the `rate_limit-enable` parameter.

Related references

7.5.3 VEVisualisation - parameters on page 7-457.

Related information

Fixed Virtual Platforms FVP Reference Guide.

7.5.2 VEVisualisation - ports

This section describes the VEVisualisation component ports.

Table 7-11 VEVisualisation ports

Name	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.
`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.

Related references

2.4.6 KeyboardStatus protocol on page 2-56.
2.4.7 LCD protocol on page 2-56.
2.4.10 MouseStatus protocol on page 2-58.

7.5.3 VEVisualisation - parameters

This section describes the configuration parameters.

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. However, it reduces the overall simulation speed. If your priority is high simulation speed, set `rate_limit-enable` to `false`.

Table 7-12 VEVisualisation parameters

Name	Type	Allowed values	Default value	Description
`cluster0_name`	`string`	-	Cluster0	Label for cluster 0 performance values.
`cluster1_name`	`string`	-	Cluster1	Label for cluster 1 performance values.
`cpu_name`	`string`	-		Processor name displayed in window title.
`daughter_led_count`	`int`	`0-32`	`0`	Set to nonzero to display up to 32 LEDs. See `daughter_leds` port.
`daughter_user_switch_count`	`int`	`0-32`	`0`	Set this parameter to display up to 32 switches. See `daughter_user_switches` port.
`disable_visualisation`	`bool`	`true`, `false`	`false`	Disable the VEVisualisation component on model startup.
`rate_limit-enable`	`bool`	`true`, `false`	`true`	Restrict simulation speed so that simulation time more closely matches real time rather than running as fast as possible.
`recorder.checkInstructionCount`	`bool`	`true`, `false`	`true`	Check instruction count in recording file against actual instruction count during playback.
`recorder.playbackFileName`	`string`	-	`""`	Playback filename (empty string disables playback).
`recorder.recordingFileName`	`string`	-	`""`	Recording filename (empty string disables recording).
`recorder.recordingTimeBase`	`int`	-	`0x5F5E100`	Timebase in 1/s (relative to the master clock (where 100000000 means 10 nanoseconds resolution simulated time for a 1Hz master clock)) for recording (higher values give higher time resolution, playback timebase is always taken from the playback file).
`recorder.verbose`	`int`	-	`0x0`	Enable verbose messages (1=normal, 2=even more).
`trap_key`	`int`	Valid ATKeyCode key value[dl]	`74`, `0x4A`[dm]	Trap key that works with left **Ctrl** key to toggle mouse display.
`window_title`	`string`	-	`"Fast Models - CLCD %cpu%"`	Window title (`cpu_name` replaces `%cpu%`).

7.5.4 VEVisualisation - verification and testing

This component passes tests by use as an I/O device for booting Linux and other operating systems.

7.5.5 VEVisualisation - performance

ARM expects the elements in the status bar to have little effect on the performance of PV systems. However, applications that often redraw the contents of the frame buffer might incur overhead through GUI interactions on the host OS.

[dl] See the header file, `%PVLIB_HOME%\components\KeyCode.h`, for a list of ATKeyCode values. On Linux, use `$PVLIB_HOME/components/KeyCode.h`.
[dm] This is equivalent to the left **Alt** key, so pressing Left Alt and Left Ctrl simultaneously toggles the mouse display.

7.5.6 VEVisualisation - library dependencies

This component relies on the *Simple DirectMedia Layer* (SDL) libraries, specifically `libSDL-1.2.so.0.11.4`.

This library is bundled with the Model Library and is also available as an rpm for Red Hat Enterprise Linux.

Related information

Simple DirectMedia Layer Cross-platform Development Library.

7.6 VE_SysRegs component

This section describes the VE system registers component.

This section contains the following subsections:

7.6.1 VE_SysRegs - about

This LISA+ component is a model of the VE model status and system control registers.

7.6.2 VE_SysRegs - ports

This section describes the ports.

Table 7-13 VE_SysRegs ports

Name	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	Multimedia 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	Multimedia bus interface input.
`mmc_card_present`	StateSignal	Slave	Indicates the presence of a *MultiMedia Card* (MMC) image.
`pvbus`	PVBus	Slave	Slave port for connection to PV bus master/decoder.
`system_reset`	Signal	Master	Signal to the platform a complete system reset. Writes to the System Configuration registers can trigger the reset signal.
`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.

7.6.3 VE_SysRegs - parameters

This section describes the parameters.

Table 7-14 VE_SysRegs parameters

Name	Type	Allowed values	Default value	Description
`exit_on_shutdown`	`bool`	`true`, `false`	`false`	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.[dn]
`sys_proc_id0`	`int`	`0x0c000000`	`0x0c000000`	Processor ID register at CoreTile Express Site 1.

[dn] For more information on the `SYS_CFGCTRL` function values, see the *Motherboard Express µATX V2M-P1 Technical Reference Manual*.

Table 7-14 VE_SysRegs parameters (continued)

Name	Type	Allowed values	Default value	Description
sys_proc_id1	int	0xff000000	0xff000000	Processor ID at CoreTile Express Site 2.
tilePresent	bool	true, false	true	Tile fitted.
user_switches_value	int	0	0	User switches.

7.6.4 VE_SysRegs - registers

This section describes the configuration registers.

Table 7-15 VE_SysRegs registers

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.
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.

7.6.5 VE_SysRegs - verification and testing

This component passes tests as part of the Versatile Express model.

7.7 Other VE components

This section describes the purpose of selected additional components included with the VE model platform model.

This section contains the following subsections:

7.7.1 VEConnector and VESocket components

These components forward interrupts and CLK and PVBus transactions between the VEBaseboard and the CoreTile.

They emulate the connections in a CoreTile to the VE model.

7.7.2 VEInterruptForwarder component

This component assists VESocket.

It receives all interrupts and tells the socket which interrupt happened.

7.7.3 VERemapper component

This component is a control component involved with remapping logic on the VE model.

It receives the `remap_clear` signal from the SP810_SysCtrl component and `boot_switch` value from the VE_SysReg component to determine which devices to remap and unremap.

Related references

7.6 VE_SysRegs component on page 7-460.

4.4.59 SP810_SysCtrl component on page 4-387.

7.8 Differences between the VE hardware and the system model

This section describes features of the hardware that the models do not implement, or implement with significant differences.

This section contains the following subsections:

7.8.1 Memory map

The model represents the memory map of the hardware VE platform, but is not an accurate representation of a specific 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.

7.8.2 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`.

7.8.3 VE hardware features absent

These Fixed Virtual Platforms do not implement some features of the hardware.

- 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).

Related references

7.2 VE memory map for Cortex-A series on page 7-450.

7.8.4 VE hardware features different

These Fixed Virtual Platforms only partially implement some features of the hardware.

The partially implemented features 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.

7.8.5 Restrictions on the processor models

Some 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, except for 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* (PV) order. Unaligned accesses are always performed as byte transfers.
- Some instruction sequences are executed atomically so that system time does not advance during their execution. This difference in behavior can affect 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 that the model supports directly are:
  - Single address unconditional instruction breakpoints.
  - Single address unconditional data breakpoints.
  - Unconditional instruction address range breakpoints.
- Pseudoregisters in the debugger support processor exception breakpoints. 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.

7.8.6 Timing considerations for the VE FVPs

The Rate Limit feature matches simulation time to wall-clock time.

The *Fixed Virtual Platforms* (FVPs) 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 run much faster than the wall clock. This means that a single key press might register as several repeated key presses, or a single mouse click incorrectly becomes a double click.

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.