Real-Time System Model

Version 1.2

Reference

ARM®

ARM DUI 0575C (ID010512)

Real-Time System Model
Reference

Release Information

Change history

Description	Issue	Confidentiality	Change
May 2011	A	Non-Confidential	First release for Fast Models 6.1.
July 2011	B	Confidential	Update for Fast Models v6.2.
November 2011	C	Non-Confidential	Update for Fast Models v7.0.

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
Real-Time System Model Reference

Preface

This preface introduces the *Real-Time System Model Reference*. It contains the following sections:

About this book

This book describes how to configure and use the *Real-Time System Models* (RTSMs). The models let you run software applications on:

- a virtual implementation of a *Versatile Express*™ (VE) board and an attached CoreTile
- a virtual implementation of a *Microcontroller Prototyping System* (MPS).

The VE RTSM is used to model ARM® application processors. The MPS RTSM is used to model the ARM Cortex™-M3 and Cortex-M4 processors.

Intended audience

This book has been written for experienced hardware and software developers to:

- understand how the RTSM examples are constructed
- use the RTSMs as part of a development environment to aid the development of products that use ARM architecture-based processors or peripherals.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to software models.

Chapter 2 *Getting Started with Real-Time System Models*

Read this for a description of how to start using the VE and MPS RTSMs. It also contains information on the terminal and Ethernet features provided with the VE RTSMs.

Chapter 3 *Programmer's Reference for the VE RTSMs*

Read this for a description of the VE memory map and registers, in addition to information on model parameters and component configuration. It also describes differences between the VE RTSMs and their hardware equivalents.

Chapter 4 *Programmer's Reference for the MPS RTSMs*

Read this for a description of the MPS memory map and registers, in addition to information on model parameters and component configuration. It also describes differences between the MPS RTSMs and their hardware equivalents.

Glossary

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

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

Typographical conventions

Conventions that this book can use are described in:

- *Typographical* on page vi
- *Signals* on page vi.

Typographical

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`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 >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Signals

The signal conventions are:

Signal level	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means: • HIGH for active-HIGH signals • LOW for active-LOW signals.
Lower-case n	At the start or end of a signal name denotes an active-LOW signal.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *AMBA® Specification* (ARM IHI 0011)
- *ARM Architecture Reference Manuals*, `http://infocenter.arm.com/help/topic/com.arm.doc.set.architecture/index.html`
- *ARM Generic Interrupt Controller Architecture Specification* (ARM IHI 0048).

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

- *ARM DS-5 Using the Debugger* (ARM DUI 0446)
- *ARM DS-5 Debugger Command Reference* (ARM DUI 0452)
- *Component Architecture Debug Interface Developer Guide* (ARM DUI 0444)
- *Fast Models User Guide* (ARM DUI 0370)
- *Fast Models Reference Manual* (ARM DUI 0423)

- *Model Debugger for Fast Models User Guide* (ARM DUI 0314)
- *Model Shell for Fast Models Reference Manual* (ARM DUI 0457).

The following publications provide information about related ARM products:

- *CoreTile Express™ A9x4 Technical Reference Manual* (ARM DUI 0448)
- *LogicTile Express™ 3MG Technical Reference Manual* (ARM DUI 0449)
- *Versatile Express Boot Monitor Reference Manual* (ARM DUI 0465)
- *ARM Cortex-A9™ MPCore Technical Reference Manual* (ARM DDI 0407)
- *Motherboard Express µATX V2M-P1 Technical Reference Manual* (DUI 0447)
- *ARM PrimeCell UART (PL011) Technical Reference Manual* (ARM DDI 0183*)*
- *ARM PrimeCell Synchronous Serial Port Controller (PL022) Technical Reference Manual* (ARM DDI 0194)
- *ARM PrimeCell Real-Time Clock Controller (PL031) Technical Reference Manual* (ARM DDI 0224)
- *ARM PrimeCell Advanced Audio CODEC Interface (PL041) Technical Reference Manual* (ARM DDI 0173)
- *ARM PrimeCell GPIO (PL061) Technical Reference Manual* (ARM DDI 0190)
- *ARM PrimeCell DMA (PL081) Technical Reference Manual* (ARM DDI 0196)
- *ARM PrimeCell Synchronous Static Memory Controller (PL093) Technical Reference Manual* (ARM DDI 236)
- *ARM PrimeCell Color LCD Controller (PL111) Technical Reference Manual* (ARM DDI 0161)
- *ARM PrimeCell Smart Card Interface (PL131) Technical Reference Manual* (ARM DDI 0228)
- *ARM PrimeCell Multimedia Card Interface (PL180) Technical Reference Manual* (ARM DDI 0172)
- *ARM PrimeCell External Bus Interface (PL220) Technical Reference Manual* (ARM DDI 0249)
- *PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual* (ARM DDI 0246)
- *ARM Dynamic Memory Controller (PL340) Technical Reference Manual* (ARM DDI 0331)
- *PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual* (ARM DDI 0416)
- *ARM Dual-Timer Module (SP804) Technical Reference Manual* (ARM DDI 0271)
- *ARM PrimeCell Watchdog Controller (SP805) Technical Reference Manual* (ARM DDI 0270)
- *ARM PrimeCell System Controller (SP810) Technical Reference Manual* (ARM DDI 0254).

Other publications

This section lists relevant documents published by third parties. The following data sheets describe some of the integrated circuits or modules used on the VE board:

- *CODEC with Sample Rate Conversion and 3D Sound* (LM4549) National Semiconductor, Santa Clara, CA.
- *MultiMedia Card Product Manual* SanDisk, Sunnyvale, CA.
- *Serially Programmable Clock Source* (ICS307), ICS, San Jose, CA.

- *1.8 Volt Intel StrataFlash Wireless Memory* with 3.0 Volt I/O (28F256L30B90) Intel Corporation, Santa Clara, CA.
- *Three-In-One Fast Ethernet Controller* (LAN91C111) SMSC, Hauppauge, NY.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on this book

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

- the title
- the number, ARM DUI 0575C
- the relevant page number to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the Real-Time System Models. It contains the following sections:

1.1 Introduction to system models

Real-Time System Models (RTSM) enable development of software without the requirement for actual hardware.

The software models provide *Programmer's View* (PV) models of processors and devices. The functional behavior of a model is equivalent to real hardware.

Absolute timing accuracy is sacrificed to achieve fast simulated execution speed. This means that you can use the PV models for confirming software functionality, but you must not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior.

System models are supplied as a CADI shared library, and are loaded by any environment compatible with the CADI API. Such environments include:

- Model Debugger
- Model Shell.

For more information, see the *Model Debugger for Fast Models User Guide*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0314-/index.html and the *Model Shell for Fast Models Reference Manual*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0457-/index.html.

1.2 Introduction to the VE RTSM

Versatile™ Express (VE) is a hardware development platform produced by ARM. The Motherboard Express µ*Advanced Technology Extended* (ATX) V2M-P1 is the basis for a highly-integrated software and hardware development system based on the ARM *Symmetric Multiprocessor System* (SMP) architecture.

The motherboard provides the following features:

- Peripherals for multimedia or networking environments.
- All motherboard peripherals and functions are accessed through a static memory bus to simplify access from daughterboards.
- High-performance PCI-Express slots for expansion cards.
- Consistent memory maps with different processor daughterboards simplify software development and porting.
- Automatic detection and configuration of attached CoreTile Express and LogicTile Express daughterboards.
- Automatic shutdown for over-temperature or power supply failure.
- System is unable to power-on if the daughterboards cannot be configured.
- Power sequencing of system.
- Supports drag and drop file update of configuration files.
- Uses either a 12V power-supply unit or an external ATX power supply.
- Supports FPGA and processor daughterboards to provide custom peripherals, or early access to processor designs, or production test chips. Supports test chips with an IO voltage range of 0.8 to 3.3 volts.

For more information, see the *Motherboard Express µATX V2M-P1 Technical Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0447-/index.html`.

The VE RTSM is a system model implemented in software. The model contains virtual implementations of a motherboard, a single daughterboard containing a specific ARM processor, and associated interconnections.

Note

The model is based on the VE platform memory map, but is not intended to be an accurate representation of a specific VE hardware revision. The VE RTSM supports selected peripherals as described in this book. The supplied model is sufficiently complete and accurate to boot the same operating system images as for the VE hardware.

The model has been developed using the ARM Fast Models™ Portfolio product.

The RTSMs provided in this release are:

- *RTSM_VE_Cortex-A15MPx1, RTSM_VE_Cortex-A15MPx2 and RTSM_VE_Cortex-A15MPx4 coretile parameters* on page 3-10
- *RTSM_VE_Cortex-A9 coretile parameters* on page 3-13
- *RTSM_VE_Cortex-R5_MPx1 and RTSM_VE_Cortex-R5_MPx2 coretile parameters* on page 3-14

- *ARMv7A-AEM* on page 3-17.

1.2.1 About the VE Real-Time System Models

The VE RTSMs provide a functionally-accurate model for software execution. However, the model sacrifices timing accuracy to increase simulation speed. Key deviations from actual hardware are:

- timing is approximate
- buses are simplified
- caches for the processors and the related write buffers are not implemented.

Many components can be configured at instantiation time. See *VE model configuration parameters* on page 3-5.

For more detail on the differences, see *Differences between the VE and coretile hardware and the models* on page 3-25.

Top-level view of a VE model

A block diagram of a top-level VE model with a Cortex-A15 cluster is shown in Figure 1-1.

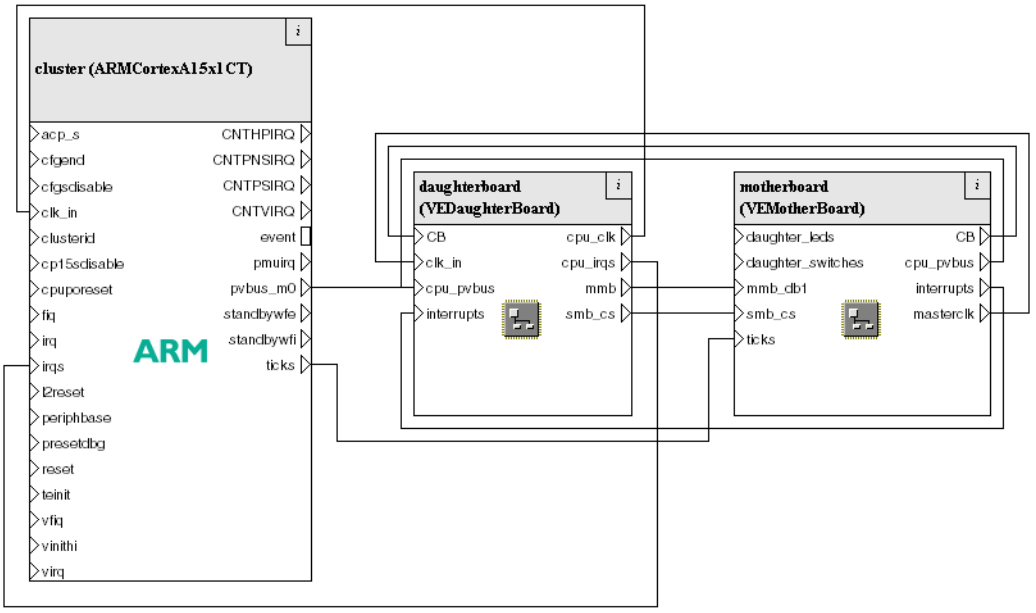

Figure 1-1 Block diagram of top-level VE model

1.3 Introduction to the MPS RTSM

The *Microcontroller Prototyping System* (MPS) is a hardware development platform produced by Gleichmann Electronics Research. The ARM Hpe® module extends the hardware to support an ARM Cortex-M3 or Cortex-M4 processor implemented in an FPGA.

The *Microcontroller Prototyping System Real-Time System Models* (MPS RTSMs) are system models implemented in software. They are developed using the ARM Fast Models library product.

Note

The MPS RTSMs are provided as example platform implementations and are not intended to be accurate representations of a specific hardware revision. The RTSMs support selected peripherals as described in this book. The supplied RTSMs are sufficiently complete and accurate to boot the same application images as the MPS hardware.

1.3.1 About the MPS hardware

The MPS hardware contains two FPGAs that implement the system:

CPU This FPGA contains:

- one instance of the Cortex-M3 or Cortex-M4 processor with ETM
- two memory controllers for RAM and FLASH on the board
- touchscreen interface
- pushbutton and DIP switch interfaces
- I2C interface
- an RS232 interface
- a configuration register block.

DUT This FPGA contains an example system that includes:

- timers
- display drivers (CLCD, character LCD, and seven-segment LED)
- audio interface
- pushbutton and DIP switch interfaces
- two RS232 interfaces
- an Hpe module interface
- MCI/SD card interface
- a USB interface.

The MPS RTSMs provide a functionally-accurate model for software execution. However, the model sacrifices timing accuracy to increase simulation speed. Key deviations from actual hardware are:

- timing is approximate
- buses are simplified
- caches for the processors and the related write buffers are not implemented
- ETM is not modeled.

1.3.2 About the MPS RTSM

The MPS RTSM models in software some of the functionality of the MPS hardware. For more detail on the differences, see *Differences between the VE and coretile hardware and the models* on page 3-25.

A complete model implementation of the MPS platform includes both MPS-specific components and generic ones such as buses and timers. Figure 1-2 shows a block diagram of a MPS RTSM.

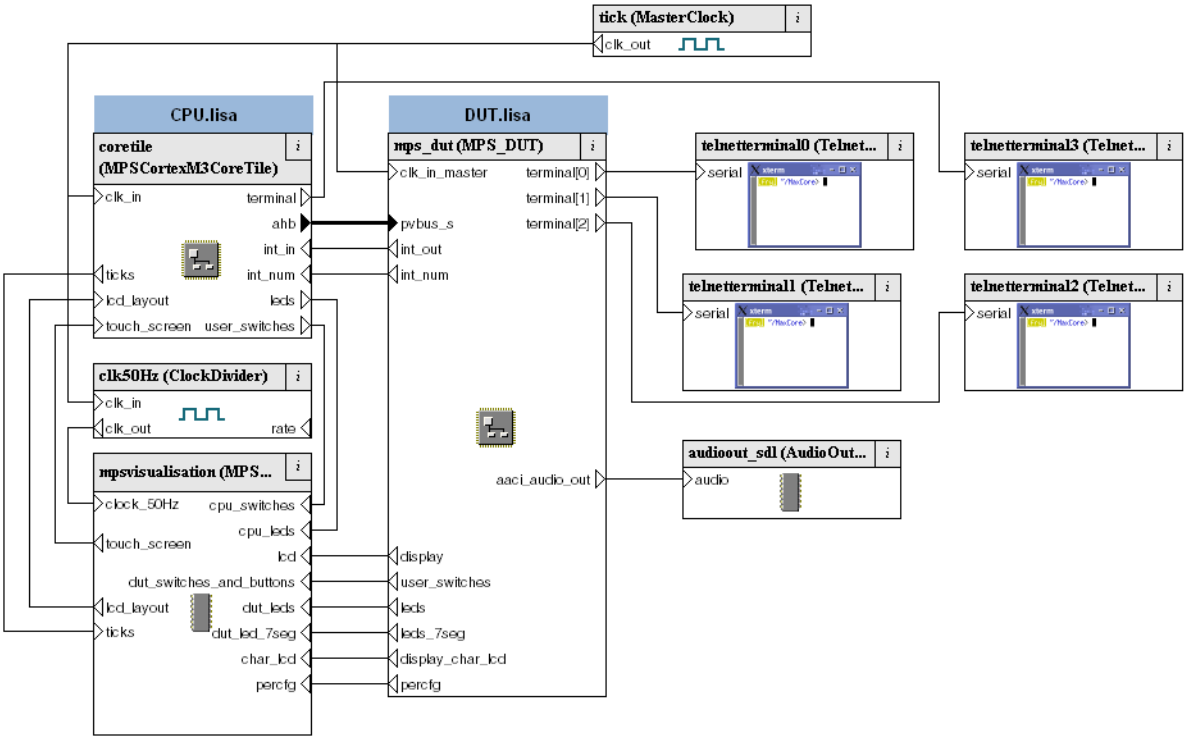

Figure 1-2 MPS RTSM block diagram

Chapter 2
Getting Started with Real-Time System Models

This chapter describes the procedures for starting and configuring RTSMs, and running a software application on the model. The procedures differ, depending on the ARM software tools that you are using. This chapter contains the following sections:

2.1 Getting started with a debugger

To debug a RTSM, you can either:

- start the RTSM from within a debugger
- connect a debugger to a model that is already running.

You can use your own debugger if it has a CADI interface to connect to a RTSM. For information about using your debugger in this way, see your debugger documentation.

See also *Loading and running an application on the VE RTSM* on page 2-7.

2.1.1 Semihosting support

The simulator handles semihosting by intercepting SVC `0x123456` or `0xAB`, depending on whether the processor is in ARM or Thumb state. All other SVCs are handled by causing the simulated core to jump to the SVC vector.

If the operating system does not use SVC `0x123456` or `0xAB` for its own purposes, it is not necessary to disable semihosting support to boot an operating system.

To temporarily or permanently disable semihosting support for a current debug connection, see the documentation that accompanies your debugger.

2.2 Getting started with Model Shell

This section describes how to use the Model Shell application to start VE and MPS RTSMs. To configure VE and MPS RTSMs, see *Configuring the RTSM* on page 2-5. An example of loading and executing an application is documented separately. See *Loading and running an application on the VE RTSM* on page 2-7.

The RTSM can be started with its own CADI debug server. This enables the model to run independently of a debugger. However, it does mean that you must configure your model using arguments that are passed to the model at start time.

To start the RTSM using Model Shell, change to the directory where your model file is located and enter the following at the command prompt:

```
model_shell --cadi-server --model model_name [--config-file filename] [--parameter
instance.parameter=value] [--application app_filename]
```

where:

model_name is the name of the model file. By default this file name is typically RTSM_VE_*processor*.dll or RTSM_MPS_*processor*.dll on Microsoft Windows or RTSM_VE_*processor*.so or RTSM_MPS_*processor*.so on Linux.

filename is the name of your optional plain-text configuration file. Configuration files simplify managing multiple parameters. See *Using a configuration file* on page 2-5.

instance.parameter=value
is the optional direct setting of a configuration parameter. See *Using the command line* on page 2-6.

app_filename is the file name of an image to load to your model at startup.

Example 2-1 shows the format for using Model Shell to load and run an image from an ELF file:

Example 2-1 Load and run an image from an ELF file

```
# Load and run from an ELF image file
model_shell \
    --parameter "motherboard.vis.rate_limit-enable=0" \
    --application test_image.axf \
    RTSM_VE_Cortex-A15x1.so
```

Note

On Microsoft Windows, it might be necessary to add to your PATH the directory in which the Model Shell executable is found. This location is typically:

```
install_directory\..\bin\model_shell
```

You can use * to load the same image into all the sub-cores in one CPU cluster, for example:

```
model_shell $MODEL -a "cluster0.*=image.axf"
```

Note

You must quote the argument as shown if you are using csh, or if you have spaces in the filename, otherwise the shell tries to expand the * instead of passing it to the application.

For more information on all Model Shell options, see the *Model Shell for Fast Models Reference Manual*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0457-/index.html.

Starting the model opens the RTSM CLCD display. See *Using the CLCD window* on page 2-8. For an example of starting and configuring a RTSM using Model Shell, see *Using Model Shell to boot a model from a flash image* on page 2-6. After the RTSM starts, you can use your debugger if it has a CADI interface to connect to the RTSM.

2.3 Configuring the RTSM

This section describes how to configure VE and MPS RTSMs. See:

- *Using a configuration GUI in your debugger*
- *Setting model configuration options from Model Shell*.

Note

Valid user settings for the VE RTSM parameters and their effects are described in *VE model configuration parameters* on page 3-5. Valid user settings for the MPS RTSM are described in *MPS configuration parameters* on page 4-6.

2.3.1 Using a configuration GUI in your debugger

In your debugger, you might be able to configure RTSM parameters before you connect to the model and start it. See the documentation that accompanies your debugger.

Note

To connect to a RTSM, your debugger must have a CADI interface.

2.3.2 Setting model configuration options from Model Shell

The initial state of the RTSM can be controlled by configuration settings provided on the command line or in the CADI properties for the model.

Using a configuration file

To configure a model that you start from the command line with Model Shell, include a reference to an optional plain text configuration file as described in *Getting started with Model Shell* on page 2-3.

Comment lines in the configuration file must begin with a `#` character.

Each non-comment line of the configuration file contains:

- the name of the component instance
- the parameter to be modified and its value.

  Boolean values can be set using either `true`/`false` or `1`/`0`. Strings must be enclosed in double quotes if they contain whitespace.

Example 2-2 shows a typical configuration file:

Example 2-2 Configuration file

```
# Disable semihosting using true/false syntax
cluster.semihosting-enable=false
#
# Enable the boot switch using 1/0 syntax
motherboard.sp810_sysctrl.use_s8=1
#
# Set the boot switch position
motherboard.ve_sysregs_0.boot_switch_value=1
```

Using the command line

You can use the -C switch to define model parameters when you invoke the model. You can also use --parameter as a synonym for the -C switch. See *Getting started with Model Shell* on page 2-3. Use the same syntax as for a configuration file, but each parameter must be preceded by the -C switch.

Example 2-3 shows how to configure a MPS RTSM using Model Shell.

Example 2-3 Using Model Shell to boot a model from a flash image

```
# Boot from a flash image
model_shell \
    --parameter "coretile.core.semihosting-cmd_line="\
    --parameter "coretile.fname=flash.bin" \
    --parameter "coretile.mps_sysregs.user_switches_value=4" \
    --parameter "coretile.mps_sysregs.memcfg_value=0" \
    --parameter "mpsvisualisation.disable-visualisation=false" \
    --parameter "mpsvisualisation.rate_limit-enable=0" \
    RTSM_MPS_Cortex-M3.so
```

2.4 Loading and running an application on the VE RTSM

Example applications are provided for use with the RTSMs for the VE.

Note

These applications are provided for demonstration purposes only and are not supported by ARM. The number of examples or implementation details might change with different versions of the system model.

A useful example application that runs on all versions of the VE RTSM is:

`brot_ve.axf` This demo application provides a simple demonstration of rendering an image to the CLCD display. Source code is supplied.

In Fast Models, the examples are in the `%PVLIB_HOME%\images` directory.

If you are using non-Fast Models software, the source code might be in the directory `%ARMROOT%\Examples\…\…\platform\mandelbrot`.

2.5 Using the CLCD window

When a RTSM starts, the RTSM CLCD window is opened.

This window represents the contents of the simulated color LCD frame buffer. It automatically resizes to match the horizontal and vertical resolution set in the CLCD peripheral registers.

For more information on the CLCD model components and other peripherals, see the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

This section describes the CLCD window for VE RTSMs and MPS RTSMs:

- *Using the VE CLCD window*
- *Using the MPS Visualization window* on page 2-10

2.5.1 Using the VE CLCD window

Figure 2-1 shows the VE RTSM CLCD in its default state, immediately after being started.

Figure 2-1 CLCD window at startup

The top section of the CLCD window displays the following status information:

USERSW Eight white boxes show the state of the VE User DIP switches:

These represent switch S6 on the VE hardware, USERSW[8:1], which is mapped to bits [7:0] of the SYS_SW register at address `0x10000004`.

The switches are in the off position by default. Click in the area above or below a white box to change its state.

BOOTSW Eight white boxes showing the state of the VE Boot DIP switches.

These represent switch S8 on the VE hardware, BOOTSEL[8:1], which is mapped to bits [15:8] of the SYS_SW register at address `0x100000004`.

The switches are in the off position by default.

Note

ARM recommends you configure the Boot DIP switches using the `boot_switch` model parameter rather than by using the CLCD interface.

Changing Boot DIP switch positions while the model is running can result in unpredictable behavior.

S6LED Eight colored boxes indicate the state of the VE User LEDs.

These represent LEDs D[21:14] on the VE hardware, which are mapped to bits [7:0] of the SYS_LED register at address `0x10000008`. The boxes correspond to the red/yellow/green LEDs on the VE hardware.

Total Instr A counter showing the total number of instructions executed.

Because the RTSM models provide a programmer's view of the system, the CLCD displays total instructions rather than total core cycles. Timing might differ substantially from the hardware because:

- the bus fabric is simplified
- memory latencies are minimized
- cycle approximate core and peripheral models are used.

In general bus transaction timing is consistent with the hardware, but timing of operations within the model is not accurate.

Total Time A counter showing the total elapsed time, in seconds.

This is wall clock time, not simulated time.

Rate Limit A feature that disables or enables fast simulation.

Because the system model is highly optimized, your code might run faster than it would on real hardware. This might cause timing issues.

Rate Limit is enabled by default. Simulation time is restricted so that it more closely matches real time. See *Timing considerations* on page 3-27.

Click on the square button to disable or enable Rate Limit. The text changes from ON to OFF and the colored box becomes darker when Rate Limit is disabled. Figure 2-2 shows the CLCD with Rate Limit disabled.

Note

You can control whether Rate Limit is enabled by using the `rate_limit-enable` parameter when instantiating the model. See *Visualization parameters* on page 3-10.

If you click on the **Total Instr** or **Total Time** items in the CLCD, the display changes to show **Inst/sec** (instructions per second) and **Perf Index** (performance index) as shown in Figure 2-2. You can click on the items again to toggle between the original and alternative displays.

Figure 2-2 CLCD window with Rate Limit off

Instr/sec Shows the number of instructions executed per second of wall clock time.

Perf Index The ratio of real time to simulation time. The larger the ratio, the faster the simulation runs. If you enable the Rate Limit feature, the Perf Index approaches unity.

You can reset the simulation counters by resetting the model.

If the CLCD window has focus:

- any keyboard input is translated to PS/2 keyboard data.
- Any mouse activity over the window is translated into PS/2 relative mouse motion data. This is then streamed to the KMI peripheral model FIFOs.

Note

The simulator only sends relative mouse motion events to the model. As a result, the host mouse pointer does not necessarily align with the target OS mouse pointer.

You can hide the host mouse pointer by pressing the **Left Ctrl**+**Left Alt** keys. Press the keys again to redisplay the host mouse pointer. Only the **Left Ctrl** key is operational. The **Right Ctrl** key on the right of the keyboard does not have the same effect.

If you prefer to use a different key, use the `trap_key` configuration option. See the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html` for information about CADI parameter documentation.

2.5.2 Using the MPS Visualization window

When a MPS RTSM starts, the Real-Time System Model CLCD window is opened.

This window represents the contents of the simulated color LCD frame buffer. It automatically resizes to match the horizontal and vertical resolution set in the CLCD peripheral registers.

Figure 2-3 shows the MPS RTSM CLCD in its default state, immediately after being started.

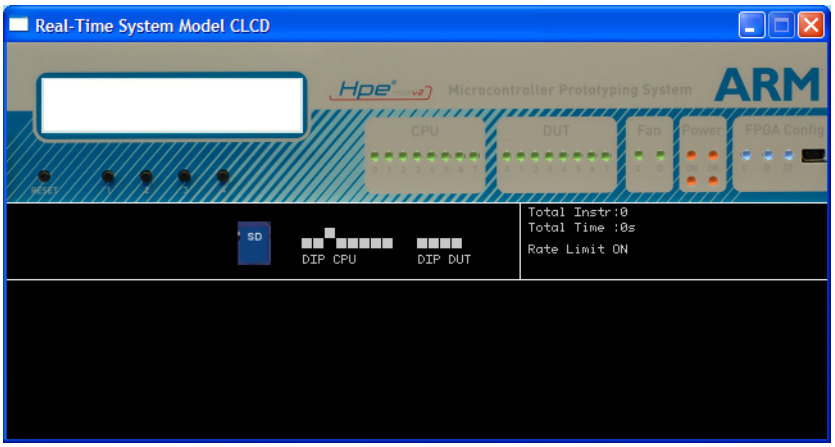

Figure 2-3 Visualization window at startup

The top section of the CLCD window displays the following status information:

Character LCD
: The large box shows the state of the character LCD.

CPU
: Eight colored circles indicate the state of the CPU LEDs.

DUT
: Eight colored circles indicate the state of the DUT LEDs.

Fan
: Two colored circles indicate the state of the fan LEDs.

Power
: Four colored circles indicate the state of the power LEDs.

FPGA Config
: Three colored circles indicate the state of the FPGA configuration LEDs.

SD
: The box with the letters SD indicates the state of the SD memory. Click the box to enable or disable the device.

DIP CPU
: Eight white boxes show the state of the CPU switches.

DIP DUT
: Four white boxes show the state of the DUT switches.

Note

ARM recommends you configure the Boot DIP switches using the `boot_switch` model parameter rather than by using the CLCD interface.

Changing Boot DIP switch positions while the model is running can result in unpredictable behavior.

Total Instr A counter showing the total number of instructions executed.

The system models provide a programmer's view of the system, so the total instructions are displayed rather than total core cycles. Timing might differ substantially from the hardware because:

- the bus fabric is simplified
- memory latencies are minimized
- cycle approximate core and peripheral models are used.

In general bus transaction timing is consistent with the hardware, but timing of operations within the model is not accurate.

Total Time A counter showing the total elapsed time, in seconds.

This is wall clock time, not simulated time.

Rate Limit A feature that disables or enables fast simulation.

Because the system model is highly optimized, your code might run faster than it would on real hardware. This might cause timing issues.

If Rate Limit is enabled, the default, simulation time is restricted so that it more closely matches real time.

Click on the square button to disable or enable Rate Limit. The text changes from ON to OFF and the colored box becomes darker when Rate Limit is disabled. Figure 2-4 on page 2-12 shows the CLCD with Rate Limit enabled.

Note

You can control whether Rate Limit is enabled by using the `rate_limit-enable` parameter when instantiating the model. See *MPS visualization configuration parameters* on page 4-6.

CLCD display

The large area at the bottom of the window displays the contents of the CLCD buffer. See Figure 2-4 on page 2-12.

If the CLCD component is not used in the simulation, the display area is black.

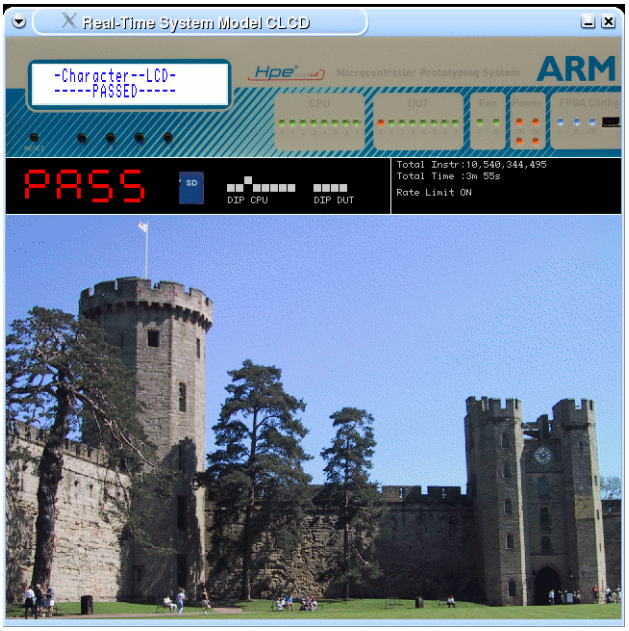

Figure 2-4 Visualization window with CLCD buffer displayed

You can hide the host mouse pointer by pressing the **Left Ctrl**+**Left Alt** keys. Press the keys again to redisplay the host mouse pointer. Only the **Left Ctrl** key is operational. The **Right Ctrl** key on the right of the keyboard does not have the same effect.

If you prefer to use a different key, use the `trap_key` configuration option. See the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html` for information about CADI parameter documentation.

2.6 Using Ethernet with a VE RTSM

The VE RTSMs provide you with a virtual Ethernet component. This is a model of the SMSC91C111 Ethernet controller, and uses a TAP device to communicate with the network. By default, the Ethernet component is disabled.

The following sections describe aspects of the VE RTSM Ethernet component:

- *Host requirements*
- *Target requirements*
- *Configuring Ethernet* on page 2-14.

2.6.1 Host requirements

Before you can use the Ethernet capability of the VE RTSM, you must first set up your host computer. For more information, see the *Fast Models User Guide*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html.

2.6.2 Target requirements

The VE RTSMs include a software implementation of the SMSC91C111 Ethernet controller. Your target OS must therefore include a driver for this specific device, and the kernel must be configured to use the SMSC chip. Operating systems that support the SMSC91C111 include WinCE, Symbian and Linux.

There are three SMSC91C111 component parameters:

- *enabled*
- *mac_address* on page 2-14
- *promiscuous* on page 2-14.

When you configure these parameters prior to starting the VE RTSM, you specify the TAP device name, set the MAC address, and define whether promiscuous mode is enabled.

enabled

This is the default state. When the device is disabled, the kernel cannot detect the device. For more information, see the SMSC_91C111 component section in the *Fast Models Reference Manual*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html. Figure 2-5 on page 2-14 shows a block diagram of the model networking structure:

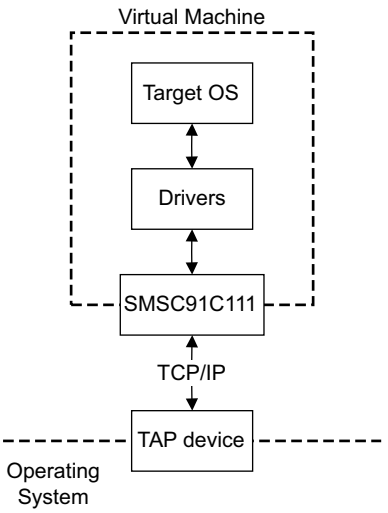

Figure 2-5 Model networking structure block diagram

A HostBridge component must be configured to perform read and write operations on the TAP device. The HostBridge component is a virtual programmer's view model. It acts as a networking gateway to exchange Ethernet packets with the TAP device on the host, and to forward packets to NIC models.

mac_address

There are two options for the mac_address parameter.

If a MAC address is not specified, when the simulator is run it takes the default MAC address, which is randomly-generated. This provides some degree of MAC address uniqueness when running models on multiple hosts on a local network.

promiscuous

The Ethernet component starts in promiscuous mode by default. This means that it receives all network traffic, even that not specifically addressed to the device. You must use this mode if you are using a single network device for multiple MAC addresses. Use this mode if, for example, you are sharing the same network card between your host OS and the VE RTSM Ethernet component.

By default, the Ethernet device on the VE RTSM has a randomly-generated MAC address and starts in promiscuous mode.

2.6.3 Configuring Ethernet

To configure a connection to the Ethernet interface on the RTSM from Microsoft Windows or Linux, see the *Fast Models User Guide*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html.

2.7 Using a terminal with a system model

The Terminal component is a virtual component that enables UART data to be transferred between a TCP/IP socket on the host and a serial port on the target.

Note

To use the Terminal component with a Microsoft Windows 7 client, you must first install Telnet. The Telnet application is not installed on Microsoft Windows 7 by default.

Download the application by following the instructions on the Microsoft web site. Search for "Windows 7 Telnet" to find the Telnet FAQ page. To install Telnet:

1. Select **Start → Control Panel → Programs and Features**. This opens a window that enables you to uninstall or change programs.
2. Select **Turn Windows features on or off** on the left side of the bar. This opens the Microsoft Windows Features dialog. Select the **Telnet Client** check box.
3. Click **OK**. The installation of Telnet might take several minutes to complete.

A block diagram of one possible relationship between the target and host through the Terminal component is shown in Figure 2-6. The TelnetTerminal block is what you configure when you define Terminal component parameters. The Virtual Machine is your VE RTSM or MPS RTSM.

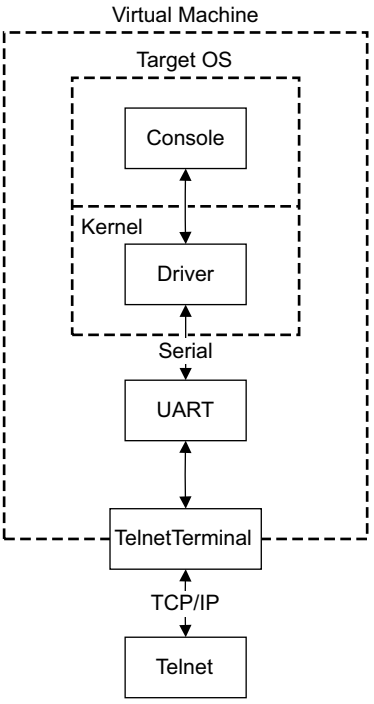

Figure 2-6 Terminal block diagram

On the target side, the console process invoked by your target OS relies on a suitable driver being present. Such drivers are normally part of the OS kernel. The driver passes serial data through a UART. The data is forwarded to the TelnetTerminal component, which exposes a TCP/IP port to the world outside of the RTSM. This port can be connected to by, for example, a Telnet process on the host.

By default, the VE RTSM or MPS RTSM starts four telnet Terminals when the model is initialized. You can change the startup behavior for each of the four Terminals by modifying the corresponding component parameters. See *Terminal parameters* on page 3-9.

If the Terminal connection is broken, for example by closing a client telnet session, the port is re-opened on the host. This might have a different port number if the original one is no longer available. Before the first data access, you can connect a client of your choice to the network socket. If there is no existing connection when the first data access is made, and the `start_telnet` parameter is `true`, a host telnet session is started automatically.

The port number of a particular Terminal instance can be defined when the RTSM starts. The actual value of the port used by each Terminal is declared when it starts or restarts, and might not be the value you specified if the port is already in use. If you are using Model Shell, the port numbers are displayed in the host window in which you started the model.

You can start the Terminal component in one of two modes:

- *Telnet mode*
- *Raw mode*.

2.7.1 Telnet mode

In telnet mode, the Terminal component supports a subset of the RFC 854 protocol. This means that the Terminal participates in negotiations between the host and client concerning what is and is not supported, but flow control is not implemented.

2.7.2 Raw mode

Raw mode enables the byte stream to pass unmodified between the host and the target. This means that the Terminal component does not participate in initial capability negotiations between the host and client. It acts as a TCP/IP port. You can use this feature to directly connect to your target through the Terminal component.

2.8 Virtual filesystem

The *Virtual FileSystem* (VFS) enables your target to access parts of a host filesystem. This access is achieved through a target OS-specific driver and a memory-mapped device called the MessageBox. When using the VFS, access to the host filesystem is analogous to access to a shared network drive, and can be expected to behave in the same way.

This section contains the following sections:

- *VFS operations*
- *Using the VFS with a pre-built RTSM* on page 2-18.

This section does not cover the process for adding the VFS component to your model system, but instead on how the end user interacts with the VFS. If you need to build your own system including the VFS, see the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`. See also the `WritingADriver.txt` file in `%PVLIB_HOME%\VFS\docs\`.

Note

VFS support is only provided by VE RTSM models. MPS RTSM models do not support VFS functionality.

2.8.1 VFS operations

The VFS supports the following filesystem operations:

getattr retrieves metadata for the file, directory or symbolic link

mkdir creates a new directory

remove removes a file, directory or symbolic link

rename renames a file, directory or symbolic link

rmdir removes an empty directory

setattr sets metadata for the file, directory or symbolic link.

Note

`setattr` is not currently implemented.

Symbolic links are not currently supported. Hard links cannot be created by the model but hard links created by the host operating system function correctly.

The VFS supports the following mount points:

closemounts
frees the iterator handle returned from `openmounts`

openmounts
retrieves an iterator handle for the list of available mounts

readmounts reads one entry from the mount iterator ID.

The VFS supports the following directory iterators:

closedir frees a directory iterator handle retrieved by `opendir`

opendir retrieves an iterator handle for the directory specified

readdir reads the next entry from the directory iterator.

Note

Datestamps returned are in milliseconds elapsed 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 simulated OS datestamp.

The VFS supports the following file operations:

closefile frees a handle opened with `openfile`

filesync forces the host OS to flush all file data to persistent storage

getfilesize returns the current size of a file, in bytes

openfile returns a handle to the file specified

readfile reads a block of data from a file

setfilesize sets the current size of a file in bytes, either by truncating, or extending the file with zeroes

writefile writes a block of data to a file.

2.8.2 Using the VFS with a pre-built RTSM

The supplied VE RTSMs include the necessary VFS components. This permits you to run a Linux image, for example, on the VE RTSM and access the filesystem running on your computer.

To use the VFS functionality of the VE RTSM, use the `motherboard.vfs2.mount` configuration parameter when you start the model. The value of the parameter is the path to the host filesystem directory that is to be made accessible within the model. See *VFS2 parameters* on page 3-9.

Mount names

When the target OS is running, create a mount point, such as `/mnt/host`. For example, on a Linux target, use the `mount` command as follows:

```
mount -t vmfs A /mnt/host
```

You can then access the host filesystem from the target OS through a supported filesystem operation. See *VFS operations* on page 2-17. See also the `ReadMe.txt` file in the `%PVLIB_HOME%\VFS2\linux\` directory.

Path names

All path names must be fully qualified paths of the form:

```
mountpoint:/path/to/object
```

Chapter 3
Programmer's Reference for the VE RTSMs

This chapter describes the memory map and the configuration registers for the peripheral and system component models. It contains the following sections:

Note

For detailed information on the programming interface for ARM PrimeCell peripherals and controllers, see the appropriate technical reference manual.

3.1 VE model memory map

Table 3-1 shows the global memory map for the platform model. This map is based on the Versatile Express RS1 memory map with the RS2 extensions.

Table 3-1 Memory map

Peripheral	Modeled	Address range	Size
NOR FLASH0 (CS0)	Yes	0x00_00000000-0x00_03FFFFFF	64MB
Reserved	-	0x00_04000000-0x00_07FFFFFF	64MB
NOR FLASH0 alias (CS0)	Yes	0x00_08000000-0x00_0BFFFFFF	64MB
NOR FLASH1 (CS4)	Yes	0x00_0C000000-0x00_0FFFFFFF	64MB
Unused (CS5)	-	0x00_10000000-0x00_13FFFFFF	-
PSRAM (CS1) - unused	No	0x00_14000000-0x00_17FFFFFF	-
Peripherals (CS2) see Table 3-2.	Yes	0x00_18000000-0x00_1BFFFFFF	64MB
Peripherals (CS3) see Table 3-3 on page 3-3.	Yes	0x00_1C000000-0x00_1FFFFFFF	64MB
CoreSight and peripherals	No	0x00_20000000-0x00_2CFFFFFF	-
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)[a]	Yes	0x00_80000000-0x00_FFFFFFFF	2GB
Unused	-	0x01_00000000-0x07_FFFFFFFF	-
4GB DRAM (in 36-bit address space)[a]	Yes	0x08_00000000-0x08_FFFFFFFF	4GB
Unused	-	0x09_00000000-0x7F_FFFFFFFF	-
4GB DRAM (in 40-bit address space)[a]	Yes	0x80_00000000-0xFF_FFFFFFFF	4GB

a. 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 3-2 shows details of the memory map for peripherals in the CS2 region.

Table 3-2 CS2 peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[a]
VRAM - aliased	Yes	0x00_18000000-0x00_19FFFFFF	32MB	-
Ethernet (SMSC 91C111)	Yes	0x00_1A000000-0x00_1AFFFFFF	16MB	47
USB - unused	No	0x00_1B000000-0x00_1BFFFFFF	16MB	-

a. The Interrupt signal column lists the values to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

Table 3-3 shows details of the memory map for peripherals in the CS3 region.

Table 3-3 CS3 peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[a]
Local DAP ROM	No	0x00_1C000000-0x00_1C00FFFF	64KB	-
VE System Registers	Yes	0x00_1C010000-0x00_1C01FFFF	64KB	-
System Controller (SP810)	Yes	0x00_1C020000-0x00_1C02FFFF	64KB	-
TwoWire serial interface (PCIe)	No	0x00_1C030000-0x00_1C03FFFF	64KB	-
AACI (PL041)	Yes	0x00_1C040000-0x00_1C04FFFF	64KB	43
MCI (PL180)	Yes	0x00_1C050000-0x00_1C05FFFF	64KB	41, 42
KMI - keyboard (PL050)	Yes	0x00_1C060000-0x00_1C06FFFF	64KB	44
KMI - mouse (PL050)	Yes	0x00_1C070000-0x00_1C07FFFF	64KB	45
Reserved	-	0x00_1C080000-0x00_1C08FFFF	64KB	-
UART0 (PL011)	Yes	0x00_1C090000-0x00_1C09FFFF	64KB	37
UART1 (PL011)	Yes	0x00_1C0A0000-0x00_1C0AFFFF	64KB	38
UART2 (PL011)	Yes	0x00_1C0B0000-0x00_1C0BFFFF	64KB	39
UART3 (PL011)	Yes	0x00_1C0C0000-0x00_1C0CFFFF	64KB	40
VFS2	Yes	0x00_1C0D0000-0x00_1C0DFFFF	64KB	73
Reserved	-	0x00_1C0E0000-0x00_1C0EFFFF	64KB	-
Watchdog (SP805)	Yes	0x00_1C0F0000-0x00_1C0FFFFF	64KB	32
Reserved	-	0x00_1C100000-0x00_1C10FFFF	64KB	-
Timer-0 (SP804)	Yes	0x00_1C110000-0x00_1C11FFFF	64KB	34
Timer-1 (SP804)	Yes	0x00_1C120000-0x00_1C12FFFF	64KB	35
Reserved	-	0x00_1C130000-0x00_1C15FFFF	192KB	-
TwoWire serial interface (DVI) - unused	No	0x00_1C160000-0x00_1C16FFFF	64KB	-
Real-time Clock (PL031)	Yes	0x00_1C170000-0x00_1C17FFFF	64KB	36
Reserved	-	0x00_1C180000-0x00_1C19FFFF	128KB	-
CF Card - unused	No	0x00_1C1A0000-0x00_1C1AFFFF	64KB	
Reserved	-	0x00_1C1B0000-0x00_1C1EFFFF	256KB	-
Color LCD Controller (PL111)	Yes	0x00_1C1F0000-0x00_1C1FFFFF	64KB	46
Reserved	-	0x00_1C200000-0x00_1FFFFFFF	62KB	-

a. The Interrupt signal column lists the values to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

Note

The VE RTSM implementation of memory does not require programming the memory controller with the correct values.

This means you must ensure that the memory controller is set up properly if you run an application on actual hardware. If this is not done, applications that run on a RTSM might fail on actual hardware.

3.2 VE model configuration parameters

The Real-Time System Models for the VE reference system have parameters that can be defined at run time.

Parameters that can be modified only at model build time, or that are not normally modified by the user in the equivalent hardware system, are not discussed.

See the following sections for details of the model parameter sets:

- *Motherboard peripheral parameters*
- *Motherboard virtual component parameters* on page 3-8
- *RTSM_VE_Cortex-A15MPx1, RTSM_VE_Cortex-A15MPx2 and RTSM_VE_Cortex-A15MPx4 coretile parameters* on page 3-10
- *RTSM_VE_Cortex-A9 coretile parameters* on page 3-13
- *RTSM_VE_Cortex-R5_MPx1 and RTSM_VE_Cortex-R5_MPx2 coretile parameters* on page 3-14
- *ARMv7A-AEM* on page 3-17.

3.2.1 Motherboard peripheral parameters

This section describes the peripheral parameters that you can change on the motherboard. It contains:

- *Color LCD controller parameters*
- *Ethernet parameters*
- *System controller parameters* on page 3-6
- *VE system register block parameters* on page 3-7
- *UART parameters* on page 3-7
- *Watchdog parameters* on page 3-7.

Color LCD controller parameters

Table 3-4 lists the Color LCD Controller instantiation-time parameters that you can change when the model is started.

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

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

Table 3-4 Color LCD controller configuration parameters

Parameter	Description	Type	Values	Default
`pixel_double_limit`	The threshold in horizontal pixels below which pixels sent to the frame-buffer are doubled in size in both dimensions.	integer	-	`0x12C`

Ethernet parameters

Table 3-5 on page 3-6 lists the Ethernet instantiation-time parameters that you can change when the model is started.

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

`motherboard.smsc_91c111.parameter=value`

Table 3-5 Ethernet configuration parameters

Parameter	Description	Type	Values	Default
`enabled`	Host interface connection enabled	boolean	`true` or `false`	`false`
`mac_address`	Host/model MAC address	string	See *mac_address parameter*	`00:02:f7:ef:31:11`
`promiscuous`	Put host into promiscuous mode, for example when sharing the Ethernet controller with the host OS.	boolean	`true` or `false`	`true`

mac_address parameter

There are two options for the mac_address parameter:

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

Note

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

For more information on how to set up and use the Ethernet component, see *Using Ethernet with a VE RTSM* on page 2-13

System controller parameters

Table 3-6 lists the system controller instantiation-time parameters that you can change when the model is started.

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

`motherboard.sp810_sysctrl.parameter=value`

Table 3-6 System controller configuration parameters

Parameter	Description	Type	Values	Default
`sysid`	Value for system identification register	integer	0, 1, 2[a]	`0x00000000`
`use_s8`	Select whether switch S8 is enabled	boolean	`true` or `false`	`false`

a. The sysid parameter takes values 0, 1, or 2. These correspond to SYS_ID register read values of:
- `sysid` parameter value = 0 => SYS_ID register value = `0x0225f500`, corresponding to REV_A
- `sysid` parameter value = 1 => SYS_ID register value = `0x12257500`, corresponding to REV_B
- `sysid` parameter value = 2 => SYS_ID register value = `0x22252500`, corresponding to REV_C.

Any other value for parameter `sysid` results in a SYS_ID register value of `0x0`.

VE system register block parameters

Table 3-7 lists the VE system register instantiation-time parameters that you can change when the model is started.

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

`motherboard.ve+sysregs.`*`parameter=value`*

Table 3-7 System register configuration parameters

Parameter	Description	Type	Values	Default
`user_switches_value`	User switch	integer	-	`0x00`
`tilePresent`	CoreTile fitted status	boolean	`true` or `false`	`true`

UART parameters

Table 3-8 lists the UART instantiation-time parameters that you can change when the model is started.

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

`motherboard.pl011_uartx.`*`parameter=value`*

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

Table 3-8 UART configuration parameters

Parameter	Description	Type	Values	Default
`baud_rate`	Baud rate	integer	-	0x9600
`clock_rate`	Clock rate for PL011	integer	-	0xE10000
`in_file`	Input file	string		[empty string]
`out_file`	Output file (use "-" to send all output to stdout).	string		[empty string]
`in_file_escape_sequence`	Input file escape sequence	string		##
`shutdown_on_eot`	Shutdown simulation when an EOT (ASCII 4) char is transmitted.	boolean	`true` or `false`	`false`
`unbuferred_output`	Unbuffered output	boolean	`true` or `false`	`false`
`untimed_fifos`	Ignore the clock rate and transmit/receive serial data immediately.	boolean	`true` or `false`	`false`
`uart_enable`	Enable the UART when the system starts.	boolean	`true` or `false`	`false`

Watchdog parameters

Table 3-9 on page 3-8 lists the watchdog instantiation-time parameters that you can change when the model is started.

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

`motherboard.sp805_wdog.parameter=value`

Table 3-9 Watchdog configuration parameters

Parameter	Description	Type	Values	Default
`simhalt`	Halt on reset	boolean	`true` or `false`	`false`

3.2.2 Motherboard virtual component parameters

This section describes the virtual component parameters that you can change on the motherboard. It contains:

- *FLASH loader parameters*
- *Host bridge parameters*
- *MultiMedia Card parameters* on page 3-9
- *Terminal parameters* on page 3-9
- *VFS2 parameters* on page 3-9
- *Visualization parameters* on page 3-10.

FLASH loader parameters

Table 3-10 lists the FLASH loader instantiation-time parameters that you can change when the model is started.

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

`motherboard.flashloaderx.parameter=value`

where *x* is the FLASH identifier 0 or 1.

Table 3-10 FLASH loader configuration parameters

Parameter	Description	Type	Values	Default
`fname`	Path to the host file used to initialize FLASH contents when the model starts. The file can be gzip compressed.	string	Valid filename	[empty string]
`fnameWrite`	Path to the host file used to save FLASH contents when the model exits.	string	Valid filename	[empty string]

Host bridge parameters

Table 3-11 lists the host bridge instantiation-time parameters that you can change when the model is started.

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

`motherboard.hostbridge.parameter=value`

Table 3-11 Host bridge configuration parameters

Parameter	Description	Type	Values	Default
`interfaceName`	Host interface identifier	string	Valid string	`ARM0`

For information on networking, see the sections on setting-up a network connection in the *Fast Models User Guide*, `../com.arm.doc.dui0370-/index.html`.

MultiMedia Card parameters

Table 3-12 lists the multimedia card instantiation-time parameters that you can change when the model is started.

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

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

Table 3-12 Multimedia card configuration parameters

Parameter	Description	Type	Values	Default
`p_mmc_file`	File used for the MMC component backing store	string	Valid string	`mmc.dat`
`p_prodName`	Card ID product name	string	Six-character string	`ARMmmc`
`p_prodRev`	Card ID product revision	integer	-	`0x1`
`p_manid`	Card ID manufacturer ID	integer	-	`0x2`
`p_OEMid`	Card ID OEM ID	integer	-	`0xCA4D0001`
`p_sernum`	Card serial number	integer	-	`0xCA4D0001`

Terminal parameters

Table 3-13 lists the terminal instantiation-time parameters that you can change when the model is started.

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

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

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

Table 3-13 Terminal configuration parameters

Parameter	Description	Type	Values	Default
`mode`	Terminal initialization mode	string	telnet, raw	telnet
`start_telnet`	Enable terminal when the system starts	boolean	`true` or `false`	`true`
`start_port`	Port used for the terminal when the system starts. If the specified port is not free, the port value is incremented by 1 until a free port is found.	integer	Valid port number	5000

For more information on using the Terminal component, see *Using a terminal with a system model* on page 2-15

VFS2 parameters

Table 3-14 on page 3-10 lists the VFS2 instantiation-time parameters that you can change when the model is started.

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

`motherboard.vfs2.parameter=value`

Table 3-14 VFS2 configuration parameters

Parameter	Description	Type	Values	Default
`mount`	Path to host folder to make accessible inside the model.	string	Valid path	[empty string]

Visualization parameters

Table 3-15 lists the visualization instantiation-time parameters that you can change when the model is started.

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

`motherboard.vis.parameter=value`

Table 3-15 Visualization configuration parameters

Parameter	Description	Type	Values	Default
`trap_key`	Trap key that works with **Left Ctrl** to toggle mouse display.	integer	-	`0x6B`
`rate_limit-enable`	Rate limit simulation	boolean	`true` or `false`	`true`
`disable_visualization`	Disable the VEVisualisation component on model startup	boolean	`true` or `false`	`false`

For more information on the visualization component, see the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

3.2.3 RTSM_VE_Cortex-A15MPx1, RTSM_VE_Cortex-A15MPx2 and RTSM_VE_Cortex-A15MPx4 coretile parameters

Table 3-16 lists the Cortex-A15 multiprocessor coretile parameters that you can change when you start the RTSM_VE_Cortex-A15MPx1, RTSM_VE_Cortex-A15MPx2 or RTSM_VE_Cortex-A15MPx4 models. All listed parameters are instantiation-time parameters. This coretile RTSM is based on r2p0 of the Cortex-A15 processor.

The syntax to use in a configuration file is:

`cluster.parameter=value`

Table 3-16 RTSM_VE_Cortex-A15MPx*n* coretile parameters

Parameter	Description	Type	Allowed Value	Default Value
`CFGSDISABLE`	Disable some accesses to DIC registers.	boolean	`true` or `false`	`false`
`CLUSTER_ID`	CPU cluster ID value.	integer	0-15	0
`IMINLN`	Instruction cache minimum line size: false=32 bytes, true=64 bytes	boolean	`true` or `false`	`true`
`PERIPHBASE`	Base address of peripheral memory space.	integer	-	`0x13080000`[a]

Table 3-16 RTSM_VE_Cortex-A15MPx*n* coretile parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
dic-spi_count	Number of shared peripheral interrupts implemented.	integer	0-224, in increments of 32	64
internal_vgic	Configures whether the model of the core contains a *Virtual Generic Interrupt Controller* (VGIC).	boolean	true or false	true
l1_dcache-state_modelled	Set whether L1 D-cache has stateful implementation	boolean	true or false	false
l1_icache-state_modelled	Set whether L1 I-cache has stateful implementation	boolean	true or false	false
l2_cache-size	Set L2 cache size in bytes	integer	0x080000, 0x100000, 0x200000, 0x400000.	0x400000
l2_cache-state_modelled	Set whether L2 cache has stateful implementation	boolean	true or false	false
l2-data-slice	L2 data RAM slice.	integer	0, 1 or 2	0
l2-tag-slice	L2 tag RAM slice.	integer	0 or 1	0

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

The RTSM_VE_Cortex-A15MPx1 has the PERIPHBASE parameter set to 0x1F000000, which is the base address of peripheral memory space on VE hardware.

Table 3-17 provides a description of the configuration parameters for each Cortex-A15MP core. These parameters are set individually for each Cortex-A15 core you have in your system. Each core has its own timer and watchdog.

The syntax to use in a configuration file is:

`cluster.cpu`*`n`*`.`*`parameter`*`=`*`value`*

where *n* is the CPU number, from 0 to 3 inclusive.

Table 3-17 RTSM_VE_Cortex-A15MPx*n* coretile parameters - individual cores

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

Table 3-17 RTSM_VE_Cortex-A15MPx*n* coretile parameters - individual cores (continued)

Parameter	Description	Type	Allowed Value	Default Value
DBGROMADDR	This value is used to initialize the CP15 **DBGDRAR** register. Bits[39:12] of this register specify the ROM table physical address.	integer	0x12000003	0x12000003
DBGROMADDRV	If true, this sets bits[1:0] of the CP15 **DBGDRAR** to indicate that the address is valid.	boolean	true or false	true
DBGSELFADDR	This value is used to initialize the CP15 **DBGDSAR** register. Bits[39:17] of this register specify the ROM table physical address.	integer	0x00010003	0x00010003
DBGSELFADDRV	If true, this sets bits[1:0] of the CP15 **DBGDSAR** to indicate that the address is valid.	boolean	true or false	true
POWERCTLI	Default power control state for CPU.	integer	0-3	0
SMPnAMP	Set whether the processor is part of a coherent domain.	boolean	true or false	false
TEINIT	Thumb exception enable. The default has exceptions including reset handled in ARM state.	boolean	true or false	false
VINITHI	Initialize with high vectors enabled.	boolean	true or false	false
ase-present[a]	Set whether processor model has been built with NEON™ support.	boolean	true or false	true
min_sync_level	Controls the minimum syncLevel by the CADI parameter interface.	integer	0-3	0
semihosting-cmd_line	Command line available to semihosting SVC calls.	string	no limit except memory	[empty string]
semihosting-debug[b]	Enable debug output of semihosting SVC calls.	boolean	true or false	false
semihosting-enable	Enable semihosting SVC traps.	boolean	true or false	true
semihosting-ARM_SVC	ARM SVC number for semihosting.	integer	0x000000 - 0xFFFFFF	0x123456
semihosting-Thumb_SVC	Thumb SVC number for semihosting.	integer	0x00 - 0xFF	0xAB
semihosting-heap_base	Virtual address of heap base.	integer	0x00000000 - 0xFFFFFFFF	0x0
semihosting-heap_limit	Virtual address of top of heap.	integer	0x00000000 - 0xFFFFFFFF	0x0F000000
semihosting-stack_base	Virtual address of base of descending stack.	integer	0x00000000 - 0xFFFFFFFF	0x10000000

Table 3-17 RTSM_VE_Cortex-A15MPx*n* coretile parameters - individual cores (continued)

Parameter	Description	Type	Allowed Value	Default Value
semihosting-stack_limit	Virtual address of stack limit.	integer	0x00000000 - 0xFFFFFFFF	0x0F000000
vfp-enable_at_reset[c]	Enable coprocessor access and VFP at reset.	boolean	true or false	false
vfp-present[a]	Set whether processor model has been built with VFP support.	boolean	true or false	true

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

vfp present and ase present

NEON and VFPv3-D32 supported.

vfp present and ase not present

VFPv3-D16 supported.

vfp not present and ase present

Illegal. Forces vfp-present to true so model has NEON and VFPv3-D32 support.

vfp not present and ase not present

Model has neither NEON nor VFPv3-D32 support.

b. Currently ignored.

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

3.2.4 RTSM_VE_Cortex-A9 coretile parameters

This section describes the Cortex-A9 MPCore parameters that you can change when you start the RTSM_VE_Cortex-A9 model. This coretile RTSM is based on r3p0 of the Cortex-A9 MPCore processor.

Table 3-18 provides a description of the configuration parameters for each Cortex-A9MP core. These parameters are set individually for each Cortex-A9 core you have in your system. Each core has its own timer and watchdog.

The syntax to use in a configuration file is:

`cluster.cpu`*`n`*`.`*`parameter`*`=`*`value`*

where *n* is the CPU number, from 0 to 3 inclusive.

Table 3-18 RTSM_VE_Cortex-A9_MPx*n* coretile parameters for the individual cores

Parameter	Description	Type	Allowed Value	Default Value
semihosting-cmd_line	Command line available to semihosting SVC calls.	string	no limit except memory	[empty string]
semihosting-debug[a]	Enable debug output of semihosting SVC calls.	boolean	true or false	false
semihosting-enable	Enable semihosting SVC traps.	boolean	true or false	true
semihosting-ARM_SVC	ARM SVC number for semihosting.	integer	0x000000 - 0xFFFFFF	0x123456

Table 3-18 RTSM_VE_Cortex-A9_MPx*n* coretile parameters for the individual cores

Parameter	Description	Type	Allowed Value	Default Value
semihosting-Thumb_SVC	Thumb SVC number for semihosting.	integer	0x00 - 0xFF	0xAB
semihosting-heap_base	Virtual address of heap base.	integer	0x00000000 - 0xFFFFFFFF	0x0
semihosting-heap_limit	Virtual address of top of heap.	integer	0x00000000 - 0xFFFFFFFF	0x0F000000
semihosting-stack_base	Virtual address of base of descending stack.	integer	0x00000000 - 0xFFFFFFFF	0x10000000
semihosting-stack_limit	Virtual address of stack limit.	integer	0x00000000 - 0xFFFFFFFF	0x0F000000

a. This is ignored.

3.2.5 RTSM_VE_Cortex-R5_MPx1 and RTSM_VE_Cortex-R5_MPx2 coretile parameters

Table 3-19 provides a description of the configuration parameters for the RTSM_VE_Cortex-R5_MPx1 or RTSM_VE_Cortex-R5_MPx2 models. These parameters are set once, irrespective of the number of Cortex-R5 processors in your system. If you have multiple Cortex-R5 processors, then each processor has its own parameters.

Table 3-19 RTSM_VE_CortexR5_MPx*n* coretile parameters

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

Table 3-19 RTSM_VE_CortexR5_MPx*n* coretile parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
MICRO_SCU	Controls whether the effects of the MicroSCU are modeled. For more information, see the *Cortex-R5 and Cortex-R5F Technical Reference Manual*.	Boolean	true/false	true
NUM_BREAKPOINTS	Controls with how many breakpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.	Integer	2-8	3
NUM_WATCHPOINTS	Controls with how many watchpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.	Integer	1-8	2
dcache-state_modelled	Set whether D-cache has stateful implementation.	Boolean	true/false	false
icache-state_modelled	Set whether I-cache has stateful implementation.	Boolean	true/false	false

Table 3-20 provides a description of the configuration parameters for each RTSM_VE_Cortex-R5_MPx1 component processor. These parameters are set individually for each processor you have in your system.

Table 3-20 RTSM_VE_CortexR5_MPx*n* coretile parameters - individual cores

Parameter	Description	Type	Allowed Value	Default Value
CFGATCMSZ	Sets the size of the ATCM.	Integer	0x00000000 - 0xE	0xE
CFGBTCMSZ	Sets the size of the BTCM.	Integer	0x00000000 - 0xE	0xE
CFGEND	Initialize to BE8 endianness.	Boolean	true/false	false
CFGIE	Set the reset value of the instruction endian bit.	Boolean	true/false	false
CFGNMFI	Enable nonmaskable FIQ interrupts on startup.	Boolean	true/false	false
DP_FLOAT	Sets whether double-precision instructions are available. For more information, see the *ARM Architecture Reference Manuals*.	Boolean	true/false. If True, then double precision VFP is supported. If False, then the VFP is single precision only.	true

Table 3-20 RTSM_VE_CortexR5_MPx*n* coretile parameters - individual cores (continued)

Parameter	Description	Type	Allowed Value	Default Value
NUM_MPU_REGION	Sets the number of MPU regions.	Integer	0x00, 0xC, 0x10. 0 = no MPU.	0xC
TEINIT	Thumb exception enable. The default has exceptions including reset handled in ARM state.	Boolean	true/false	false
VINITHI	Initialize with high vectors enabled.	Boolean	true/false	false
atcm_base[a]	Model-specific. Sets the base address of the ATCM. See the *Cortex-R5 and Cortex-R5F Technical Reference Manual* for details of the processor configuration signals.	Integer	0x00000000 - 0xFFFFFFFF	0x40000000
btcm_base[a]	Model-specific. Sets the base address of the BTCM. See the *Cortex-R5 and Cortex-R5F Technical Reference Manual* for details of the processor configuration signals.	Integer	0x00000000 - 0xFFFFFFFF	0x00000000
dcache-size	Set D-cache size in bytes.	Integer	0x4000 - 0x10000	0x10000
icache-size	Set I-cache size in bytes.	Integer	0x4000 - 0x10000	0x10000
semihosting-ARM_SVC	ARM SVC number for semihosting.	Integer	0x000000 - 0xFFFFFF	0x123456
semihosting-cmd_line	Command line available to semihosting SVC calls.	String	No limit except memory	[Empty string]
semihosting-debug[b]	Enable debug output of semihosting SVC calls.	Boolean	true/false	false
semihosting-enable	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to False.	Boolean	true/false	true
semihosting-heap_base	Virtual address of heap base.	Integer	0x00000000 - 0xFFFFFFFF	0x0
semihosting-heap_limit	Virtual address of top of heap.	Integer	0x00000000 - 0xFFFFFFFF	0x0F000000
semihosting-stack_base	Virtual address of base of descending stack.	Integer	0x00000000 - 0xFFFFFFFF	0x10000000

Table 3-20 RTSM_VE_CortexR5_MPx*n* coretile parameters - individual cores (continued)

Parameter	Description	Type	Allowed Value	Default Value
semihosting-stack_limit	Virtual address of stack limit.	Integer	0x00000000 - 0xFFFFFFFF	0x0F000000
semihosting-Thumb_SVC	Thumb SVC number for semihosting.	Integer	0x00 - 0xFF	0xAB
vfp-enable_at_reset[a]	Enable coprocessor access and VFP at reset.	Boolean	true/false	false
vfp-present	Set whether model has VFP support.	Boolean	true/false	true

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

b. Currently ignored.

3.2.6 ARMv7A-AEM

This section describes the parameters that you can change when you want to adjust the behavior of external platform components on the VE system board. These are:

- *Multicore configuration*
- *General processor configuration* on page 3-18
- *Memory configuration* on page 3-19
- *Cache geometry configuration* on page 3-20
- *Debug architecture configuration* on page 3-22
- *Core configuration* on page 3-22
- *Semihosting configuration* on page 3-23
- *Message configuration* on page 3-24.

Multicore configuration

You can configure this model as a multicore processor, so there are separate groups of configuration parameters for each core in the system. In cases where fewer cores than the maximum number possible are instantiated, the parameters from cpu0 are always used first. See Table 3-21.

Table 3-21 Multiprocessing parameters

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

General processor configuration

This section describes processor configuration parameters. See Table 3-22.

Table 3-22 Processor configuration parameters

Parameter	Description	Default
auxilliary_feature_register0	Value for AFR0 ID register	0
cpuID	Value for main CPU ID register	0x411fc081
dic-spi_count	Number of shared peripheral interrupts implemented.	64
dtcm0_base	DTCM base address at reset	0
dtcm0_enable	Enable DTCM at reset	false
dtcm0_size	DTCM size in KB	32
FILTEREN	Enable filtering of accesses between master bus ports. This is usually not used inside a VE system and should be left false.	false
FILTEREND	End of region filtered to pvbus_m1. Values must be aligned to a 1MB boundary.	0
FILTERSTART	Start of region filtered to pvbus_m1. Values must be aligned to a 1MB boundary.	0
implements_ple_like_a8	Add support for the PLE from a Cortex-A8 processor	false
IS_VALIDATION[a]	Reserved. Enables A9-validation-like trickbox-coprocessor, which is only usable in validation platform model.	false
itcm0_base	ITCM base address at reset	0x40000000
itcm0_enable	Enable ITCM at reset	false
itcm0_size	ITCM size in KB	32
PERIPHBASE[b]	Base address of MP "private" peripherals (WatchdogTimers, GIC) (bits 31:13 used).	0x13080000
siliconID	Value for Auxilliary ID register	0x41000000
CFGSDISABLE	Disables access to some registers in the internal interrupt controller peripheral.	false
implements_virtualization	Implement the Virtualization extension in this processor. When set, this also enables LPAE.	false
implements_lpae	Implement the Large Physical Address extension in this processor.	false
use_Cortex-A15_peripherals	Changes the layout of the internal peripheral memory map to mimic that of the Cortex-A15 processor.	false
delayed_CP15_operations	Delay the functional effect of CP15 operations.	false
take_ccfail_undef	Take undefined exceptions even if the instruction failed its condition codes check.	false
low_latency_mode	Run only a single instruction between checks for IRQ and other events. This ensures that when the platform raises an interrupt, the exception vector is taken immediately, but it involves a considerable penalty in performance.	false

a. IS_VALIDATION is not exposed in the VE platform model, and fixed as `false`.
b. PERIPHBASE is not exposed in the VE platform model, and fixed as `0x2C000000`.

Memory configuration

This section describes memory configuration parameters. See Table 3-23.

Table 3-23 Memory configuration parameters

Parameter	Description	Default
`vmsa.implements_fcse`	Support `fcse` in this processor	`false`
`vmsa.infinite_write_buffer`	Enable infinite write-buffer.	`false`
`vmsa.write_buffer_delay`	Elapsed time between natural buffer drains.	1000
`vmsa.delayed_read_buffer`	Enable deferred read values in conjunction with `use_IR`.	`false`
`vmsa.cache_incoherence_check`	Enable the check for cache incoherence.	`false`
`vmsa.memory_marking_check`	Enable the check for inconsistent memory marking in the TLB.	`false`
`vmsa.instruction_tlb_lockable_entries`	Number of lockable entries in instruction TLB	32
`vmsa.instruction_tlb_size`	Total number of entries in instruction TLB	32
`vmsa.main_tlb_lockable_entries`	Number of lockable entries in data or unified TLB	32
`vmsa.main_tlb_size`	Total number of entries in data or unified TLB	32
`vmsa.separate_tlbs`	Separate ITLB and DTLB. If the TLB is unified, its size is defined by parameter `vmsa.main_tlb_size`.	`true`
`vmsa.tlb_prefetch`	Enables aggressive pre-fetching into the TLB.	`false`
`vmsa.implements_outer_shareable`	Distinguish between inner shareable and outer shareable memory access types. Outer shareable is implemented as Non Cacheable.	`true`
`vmsa.access_flags_hardware_management`	Enable support for the hardware management of the Access Flag in the pagetables.	`true`
`dcache-state_modelled`	Allow line allocation in d-side caches at all levels	`true`
`icache-state_modelled`	Allow line allocation in i-side caches at all levels. Unified caches allocate lines only if these parameters are enabled at both i-side and d-side.	`true`

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

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

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

Cache geometry configuration

You can configure the processor with up to four levels of cache. The cache layout is not required to be symmetrical for each CPU in the processor, so the parameters listed in Table 3-24 are repeated in groups `cpu0-cpu3` corresponding to the view for each core of the memory hierarchy.

Table 3-24 General cache configuration parameters

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

Each cache block in the system is configured using the parameters listed in Table 3-25, which are repeated for groups `cpu0-cpu3`, and within each group in caches `l1icache`, `l1dcache-l4icache`, `l4dcache`.

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

Table 3-25 Cache block configuration parameters

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

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

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

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

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

See Figure 3-1 and Figure 3-2 for examples of CPU-cache architecture configurations.

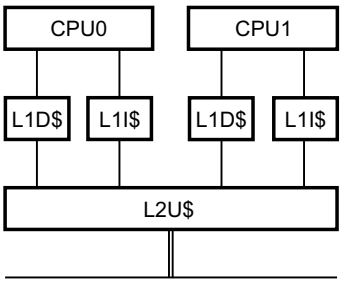

Figure 3-1 CPU-cache architecture configuration - Example 1

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

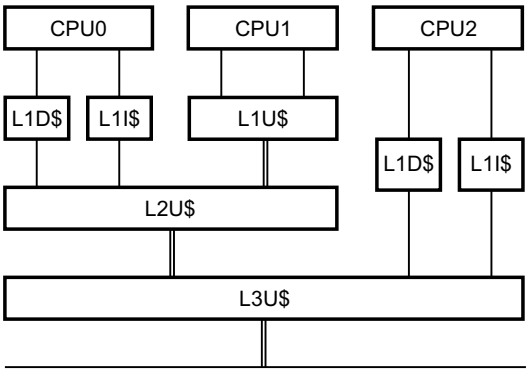

Figure 3-2 CPU-cache architecture configuration - Example 2

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

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

Note

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

Debug architecture configuration

The ARMv7 Debug architecture contains a number of optional features. The parameters listed in Table 3-26 control which of these features are implemented by the model.

Table 3-26 Debug architecture configuration parameters

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

Core configuration

These parameters are repeated in groups cpu0-cpu3 for each core in the processor. See Table 3-27.

Table 3-27 Core configuration parameters

Parameter	Description	Default
cpu[n].CFGEND0	Starts the core in big endian BE8 mode	false
cpu[n].CFGNMFI	Sets the NMFI bit in the *System Control Register* (SCTLR) that prevents the FIQ interrupt from being masked in APSR.	false
cpu[n].CFGTE	Starts the core in Thumb2 mode	false
cpu[n].CP15SDISABLE	Disables access to some CP15 registers	false
cpu[n].VINITHI	Starts the core with high vectors enabled, the vector base address is 0xFFFF0000	false
cpu[n].implements_neon	Support NEON in this CPU	true

Table 3-27 Core configuration parameters (continued)

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

Semihosting configuration

Semihosting is a method of target software running on the model to communicate with the host environment. This model permits the target C library to access I/O facilities of the host computer, filesystem, keyboard input, clock, and so on.

These parameters are repeated in groups cpu0-cpu3 for each core in the processor. See Table 3-28.

Table 3-28 Core configuration parameters

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

Message configuration

The parameters listed in Table 3-29 control how warning and error messages from the architectural checkers are generated.

Table 3-29 Message severity levels

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

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

Except for fatal errors, the severity level of each message can be reconfigured in parameters `messages.severity_level_[*]`, enabling you to concentrate only on those warnings that are appropriate to your task. See Table 3-30.

Table 3-30 Message configuration parameters

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

Boundary features and architectural checkers

Boundary features and architectural checkers are model capabilities that help your development and testing process by exposing latent problems in the target code. Certain boundary features or architectural checkers, however, might have an adverse effect on the overall running speed of target code. For more information on these aspects, see the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

`IMPLEMENTATION DEFINED` features

Some aspects of the behavior of the processor are `IMPLEMENTATION DEFINED` in the ARM architecture, meaning that they can legally vary between different CPU implementations. Any code that is intended to be run portably across the multiple ARM implementations must take care when using any of these facilities, since they might or might not be present. For more information, see the *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

3.3 Differences between the VE and coretile hardware and the models

This section describes features of the VE hardware that are not implemented in the models, or that have significant differences in implementation:

- *Memory map*
- *Memory aliasing*
- *Features not present in the model*
- *Features partially implemented in the model*
- *Restrictions on the processor models* on page 3-26
- *Timing considerations* on page 3-27.

3.3.1 Memory map

The model is based on the memory map of the hardware VE platform, but is not intended to be an accurate representation of specific VE hardware revision. The memory map in the supplied model is sufficiently complete and accurate to boot the same operating system images as for the VE hardware.

In the memory map, memory regions that are not explicitly occupied by a peripheral or by memory are unmapped. This includes regions otherwise occupied by a peripheral that is not implemented, and those areas that are documented as reserved. Accessing these regions from the host processor results in the model presenting a warning.

3.3.2 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`.

3.3.3 Features not present in the model

The following features present on the hardware version of the VE motherboard are not implemented in the system models:

- two-wire serial bus interfaces
- USB interfaces
- PCI Express interfaces
- compact flash
- *Digital Visual Interface* (DVI)
- debug and test interfaces
- *Dynamic Memory Controller* (DMC)
- *Static Memory Controller* (SMC).

Note

For more information on memory-mapped peripherals, see *VE model memory map* on page 3-2.

3.3.4 Features partially implemented in the model

The following feature present on the hardware version of the VE motherboard is only partially implemented in the Real-Time System Models:

- *Sound* on page 3-26.

Partial implementation means that some of the components are present but the functionality has not been fully modeled. If you use these features, they might not work as you expect. Check the model release notes for the latest information.

Sound

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

3.3.5 Restrictions on the processor models

Detailed information concerning what features are not fully implemented in the processor models included with the VE RTSMs can be found in separate documentation. See the *Fast Models Reference Manual*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html. The following general restrictions apply to the Real-Time System Model implementations of ARM processors:

- The simulator does not model cycle timing. In aggregate, all instructions execute in one core master clock cycle, with the exception of Wait For Interrupt.
- Write buffers are not modeled.
- Most aspects of TLB behavior are implemented in the models. In ARMv7 models, the TLB memory attribute settings are used when stateful cache is enabled.
- No MicroTLB is implemented.
- A single memory access port is implemented. The port combines accesses for instruction, data, DMA and peripherals. Configuration of the peripheral port memory map register is ignored.
- All memory accesses are atomic and are performed in programmer's view order. All transactions on the PVBus are a maximum of 32 bits wide. Unaligned accesses are always performed as byte transfers.
- Some instruction sequences are executed atomically, ahead of the component master clock, so that system time does advance during their execution. This can sometimes have an effect in sequential access of device registers where devices are expecting time to move on between each access.
- Interrupts are not taken at every instruction boundary.
- The semihosting-debug configuration parameter is ignored.
- Integration and test registers are not implemented.
- Not all CP14 debug registers are implemented.
- To debug a RTSM you must use an external debugger.
- Breakpoint types supported directly by the model are:
  - single address unconditional instruction breakpoints
  - single address unconditional data breakpoints
  - unconditional instruction address range breakpoints
- Processor exception breakpoints are supported by pseudo-registers in the debugger. Setting an exception register to a non-zero value stops execution on entry to the associated exception vector.

- Performance counters are not implemented.

RTSM_VE_Cortex-A9 coretile

The following additional restrictions apply to the Real-Time System Model implementation of a Cortex-A9 MPCore processor:

- The Cortex-A9MPCore processor contains some memory-mapped peripherals. These are modeled by the RTSM.
- Two 4GB address spaces are seen by the model core, one as seen from secure mode and one as seen from normal mode. The address spaces contain zero-wait state memory and peripherals, but a lot of the space is unmapped.
- 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 SCU is only partially modeled:
  - The SCU enable bit is ignored. The SCU is always enabled.
  - The SCU ignores the invalidate all register.
  - Coherency operations are represented by a memory write followed by a read to refill from memory, rather than using cache-to-cache transfers.
  - There is no address filtering within the SCU. The enable bit for this feature is ignored.

3.3.6 Timing considerations

The Real-Time System Models provide an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

When code interacts with real world devices like timers and keyboards, data arrives in the modeled device in real-world, or wall-clock, time, but simulation time can be running much faster than the wall clock. This means that a single keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

The VE RTSMs provide the Rate Limit feature to match simulation time to match wall-clock time. Enabling Rate Limit, either by using the Rate Limit button in the CLCD display, or the `rate_limit-enable` model instantiation parameter, forces the model to run at wall-clock time. This avoids issues with two clocks running at significantly different rates. For interactive applications, ARM recommends enabling Rate Limit.

Chapter 4
Programmer's Reference for the MPS RTSMs

This chapter describes the memory map and the configuration registers for the peripheral and system component models. It contains the following sections:

4.1 MPS model memory map

This section describes the MPS memory map. For standard ARM peripherals, see the TRM for that device.

Table 4-1 Overview of MPS memory map

Description	Modelled	Address range
4MB Remap region for SRAM0 overlay of Flash	Yes	0x00000000–0x003FFFFF
Non remapped Flash memory	Yes	0x00400000–0x03FFFFFF
SRAM for code and data storage (remap RAM)	Yes	0x10000000–0x103FFFFF
SRAM for code and data storage	Yes	0x10400000–0x107FFFFF
FLASH aliased for programming	Yes	0x18000000–0x1BFFFFFF
CPU System Registers	Yes	0x1F000000–0x1F000FFF
Reserved for SMC configuration registers	N/A	0x1F001000–0x1F002FFF
I2C for DVI	Yes	0x1F003000–0x1F003FFF
PL022 SPI for Touch Screen	Yes	0x1F004000–0x1F004FFF
PL011 UART	Yes	0x1F005000–0x1F005FFF
Reserved	N/A	0x1F006000–0x1FFFFFFF
SP805 Watch Dog	Yes	0x40000000–0x4000FFFF
PL031 RTC	Yes	0x40001000–0x40001FFF
SP804 Timer (0)	Yes	0x40002000–0x40002FFF
SP804 Timer (1)	Yes	0x40003000–0x40003FFF
DUT Sys Regs	Yes	0x40004000–0x40004FFF
PL181 SD/MMC controller	Yes	0x40005000–0x40005FFF
Reserved	N/A	0x40006000–0x40006FFF
PL011 UART (1)	Yes	0x40007000–0x40007FFF
PL011 UART (2)	Yes	0x40008000–0x40008FFF
PL011 UART (3)	Yes	0x40009000–0x40009FFF
PL041 AC97 controller	Yes	0x4000A000–0x4000AFFF
DS702 I2C (ADCDAC)	Partial[a]	0x4000B000–0x4000BFFF
DUT Character LCD	Yes	0x4000C000–0x4000CFFF
Reserved	N/A	0x4000D000–0x4000EFFF
Reserved	N/A	0x4FFA0000–0x4FFAFFFF
Flexray	Partial[a]	0x4FFB0000–0x4FFBFFFF
CAN	Partial[a]	0x4FFC0000–0x4FFCFFFF

Table 4-1 Overview of MPS memory map (continued)

Description	Modelled	Address range
LIN	Partial[a]	0x4FFD0000–0x4FFDFFFF
Ethernet	Partial[a]	0x4FFE0000–0x4FFEFFFF
Video	Yes	0x4FFF0000–0x4FFFFFFF
External AHB interface to DUT FPGA	Yes	0x50000000–0x5FFFFFFF
DMC	Yes	0x60000000–0x9FFFFFFF
SMC	Yes	0xA0000000–0xAFFFFFFF
Private Peripheral Bus	Yes	0xE0000000–0xE00FFFFF
System bus interface to DUT FPGA	Yes	0xE0100000–0xFFFFFFFF

a. This model is represented by a register bank and has no functionality beyond this.

Note

A Bus Error is generated for accesses to memory areas not shown in this table.

Any memory device that does not occupy the total region is aliased within that region.

4.1.1 MPS registers

This section describes the MPS memory-mapped registers.

CPU system registers

Table 4-2 provides a description of the CPU system registers.

Table 4-2 MPS CPU system registers

Register name	Address	Access	Description
SYS_ID	0x1f000000	read/write	Board and FPGA identifier
SYS_MEMCFG	0x1f000004	read/write	Memory remap and alias
SYS_SW	0x1f000008	read/write	Indicates user switch settings
SYS_LED	0x1f00000C	read/write	Sets LED outputs
SYS_TS	0x1f000010	read/write	Touchscreen register

DUT system registers

Table 4-3 provides a description of the DUT system registers.

Table 4-3 MPS DUT system registers

Register name	Address	Access	Description
SYS_ID	0x40004000	read/write	Board and FPGA identifier
SYS_PERCFG	0x40004004	read/write	Peripheral control signals
SYS_SW	0x40004008	read/write	Indicates user switch settings
SYS_LED	0x4000400C	read/write	Sets LED outputs
SYS_7SEG	0x40004010	read/write	Sets seven-segment LED outputs
SYS_CNT25MHZ	0x40004014	read/write	Free running counter incrementing at 25MHz.
SYS_CNT100HZ	0x40004018	read/write	Free running counter incrementing at 100Hz

Character LCD registers

Table 4-4 provides a description of the character LCD registers.

Table 4-4 MPS LCD registers

Register name	Address	Access	Description
CHAR_COM	0x4000C000	write	Command register. The command set is compatible with the commands of the Hitachi HD44780U controller.
CHAR_DAT	0x4000C004	write	Write data register.
CHAR_RD	0x4000C008	read	Read data register.
CHAR_RAW	0x4000C00C	read/write	Raw interrupt.
CHAR_MASK	0x4000C010	read/write	Interrupt mask.
CHAR_STAT	0x4000C014	read/write	Masked interrupt.

Memory configuration and remap

Table 4-5 provides a description of the memory configuration register.

Table 4-5 Memory configuration

Name	Bits	Access	Power On Reset	Description
Reserved	31:3	-	-	-
SWDPEN	2	RW	0b	Single Wire Debug Port Enable. 1 is SWD 0 JTAG
ALIAS	1	RW	1b	Alias FLASH. 1 is Aliased on 0 Aliased off
REMAP	0	RW	0b	Remap SSRAM. 1 is Remap on 0 Remap off

The ability to remap the static RAM into the bottom of memory (overlaying the Flash) is required for booting and code execution to enable the interrupt vector table to be modified. It is also used to enable boot code execution from SRAM for code development, rather than programming the FLASH each time.

The aliasing of the Flash memory into SRAM space is required to permit the Flash memory to be reprogrammed at this offset. It also enables full flash memory access when remapping is enabled. If remapping of flash is disabled only the Flash memory above 4MB is accessible.

Switches

Table 4-6 lists the bits for the user switch inputs.

Table 4-6 User switches

Name	Bits	Access	Reset	Note
Reserved	31:8	-	-	-
USER_BUT[3:0]	7:4	RO	-	Always returns value of user buttons
USER_SW[3:0]	3:0	RO	-	Always returns value of user switches

Seven-segment display

Table 4-7 lists the bits that control the seven-segment display.

Table 4-7 Seven-segment register

Name	Bits	Access	Reset	Note
DISP3	31:24	RW	0x00	Segments for display 3
DISP2	23:16	RW	0x00	Segments for display 2
DISP1	15:8	RW	0x00	Segments for display 1
DISP0	7:0	RW	0x00	Segments for display 0

4.2 MPS configuration parameters

This section describes the system parameters that can be configured at runtime.

4.2.1 MPS visualization configuration parameters

Table 4-8 provides a description of the visualization parameters for the MPSVisualisation component.

Table 4-8 Visualization parameters

Parameter name	Description	Type	Allowed value	Default value
`trap_key`	trap key that works with **Left Ctrl** key to toggle mouse pointer display	integer	valid ATKeyCode key value[a]	74[b]
`rate_limit_enable`	rate limit simulation	boolean	`true` or `false`	`true`
`disable_visualisation`	enable/disable visualization	boolean	`true` or `false`	`false`

a. If you have Fast Models installed, see the header file, `%PVLIB_HOME%\components\KeyCode.h`, for a list of ATKeyCode values. On Linux use `$PVLIB_HOME/components/KeyCode.h`.

b. This is equivalent to the **Left Alt** key.

4.2.2 DUT configuration parameters

Table 4-9 provides a description of the configuration parameters for the DUT.

Table 4-9 DUT configuration parameters

Parameter name	Description	Type	Allowed value	Default value
`mps_dut.dut_sysregs.user_switches_value`	User switches	integer	`0x0`–`0xFF`	0
`mps_dut.mmc.p_mmc_file`	MMC contents file name	string		mmc.dat
`mps_dut.sp805.simhalt`	Halt on reset	boolean	`true` or `false`	`false`
`mps_dut.uart[0|1|2].untimed_fifos`	Operate UART FIFO in fast (no timing) mode	boolean	`true` or `false`	`false`
`mps_dut.uart[0|1|2].unbuffered_output`	Unbuffered output	boolean	`true` or `false`	`false`

4.2.3 Terminal parameters

When the MPS RTSM starts, a TCP/IP port for each enabled Terminal is opened. This is port 5000 by default, but increments by 1 until a free user port is found. For more information on using the Terminal component, see *Using a terminal with a system model* on page 2-15.

Table 4-10 on page 4-7 lists the terminal instantiation-time parameters that you can change when you start the model. The syntax to use in a configuration file is:

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

where *x* is the terminal ID 0, 1, 2 or 3 and *parameter* is the parameter name.

Note

The telnet Terminal does not obey control flow signals. This means that the timing characteristics of Terminal are not the same as a standard serial port.

Table 4-10 Terminal instantiation parameters

Component name	Parameter	Description	Type	Values	Default
terminal_[0-3]	mode	Terminal operation mode.	string	telnet[a], raw[b]	telnet
terminal_[0-3]	start_telnet	Enable terminal when the system starts.	boolean	true or false	true
terminal_[0-3]	start_port	Port used for the terminal when the system starts. If the specified port is not free, the port value is incremented by 1 until a free port is found.	integer	valid port number	5000

a. In telnet mode, the Terminal component supports a subset of the telnet protocol defined in RFC 854.

b. In raw mode, the Terminal component does not interpret or modify the byte stream contents.

4.2.4 Core configuration parameters

This section describes the configuration parameters for the ARM Cortex-M3 and Cortex-M4 processor models.

Table 4-11 Configuration parameters

Parameter	Description	Type	Allowed Value	Default Value
semihosting-cmd_line[a]	Command line available to semihosting SVC calls.	string	no limit except memory	[empty string]
semihosting-debug	Enable debug output of semihosting SVC calls.	boolean	true or false	false
semihosting-enable	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to false.	boolean	true or false	true
semihosting-Thumb_SVC	Thumb SVC number for semihosting.	integer	8 bit integer	0xAB
semihosting-heap_base	Virtual address of heap base.	integer	0x00000000 - 0xFFFFFFFF	0x0
semihosting-heap_limit	Virtual address of top of heap.	integer	0x00000000 - 0xFFFFFFFF	0x10700000
semihosting-stack_base	Virtual address of base of descending stack.	integer	0x00000000 - 0xFFFFFFFF	0x10700000

Table 4-11 Configuration parameters (continued)

Parameter	Description	Type	Allowed Value	Default Value
semihosting-stack_limit	Virtual address of stack limit.	integer	0x00000000 - 0xFFFFFFFF	0x10800000
coretile.fname	Flash loader filename.	string	-	[empty string]
coretile.flashloader.fnameWrite	Filename to write to if flash image is modified.	string	-	[empty string]
coretile.uart3.untimed_fifos	Ignore the clock rate and transmit or receive serial data immediately.	boolean	true or false	false
coretile.uart3.unbuffered_output	Unbuffered output.	boolean	true or false	false

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

4.3 Differences between the MPS hardware and the system model

The following sections describe features of the MPS hardware that are not implemented in the models or have significant differences in implementation:

- *Features not present in the model*
- *Timing considerations*.

4.3.1 Features not present in the model

The Ethernet component is currently not implemented in either the model or the hardware.

The following features present on the hardware version of the MPS are not implemented in the system models:

- I2C interface
- CAN interface
- LIN
- FlexRay.

Sound

The MPS model implements the PL041 AACI PrimeCell and the audio CODEC as in the MPS hardware, but with a limited number of sample rates. AACI Audio Input is not supported.

Note

The sound component present on the hardware version of the MPS is only partially implemented in the model.

Partial implementation means that some of the components are present but the functionality has not been fully modeled. If you use these features, it might result in the model not behaving as expected. Check the model release notes for the latest information.

4.3.2 Timing considerations

The Real-Time System Models provide you with an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

If your code interacts with real world devices such as timers and keyboards, data arrives in the modeled device in real world, or wall clock, time, but simulation time can be running much faster than the wall clock. This means that a single keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

To correct for this, the MPS RTSM provides the Rate Limit feature. Enabling Rate Limit, either using the Rate Limit button in the CLCD display, or the `rate_limit-enable` model instantiation parameter, forces the model to run at wall clock time. This avoids issues with two clocks running at significantly different rates. For interactive applications, ARM recommends enabling Rate Limit.