RealView® Development Suite

Version 4.1

Real-Time System Models User Guide

ARM®

ARM DUI 0424F (ID052111)

RealView Development Suite
Real-Time System Models User Guide

Release Information

Change history

Description	Issue	Confidentiality	Change
August 2008	A	Non-Confidential	Release for RealView Development Suite v4.0 Professional, System Generator v4.0 SP1.
December 2008	B	Non-Confidential	Release for Fast Models 4.1. Added changes related to ARM_RTSM_PATH.
March 2009	C	Non-Confidential	Release for Fast Models 4.2. Minor changes to text. Added description for device-accurate-tlb parameter.
November 2009	D	Non-Confidential	Release for models provided by RealView Development Suite v4.0 Professional edition.
May 2010	E	Non-Confidential	Release for models provided by RealView Development Suite v4.1 Professional edition.
30 September 2010	F	Non-Confidential	Release for models provided by RealView Development Suite v4.1 SP1 Professional edition.

Proprietary Notice

XVID Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Development Suite Real-Time System Models User Guide

List of Tables
RealView Development Suite Real-Time System Models User Guide

List of Figures
RealView Development Suite Real-Time System Models User Guide

Preface

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

About this book

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

- a virtual implementation of an *Emulation Baseboard* (EB) and an attached Core Tile
- a virtual implementation of a *Microcontroller Prototyping System* (MPS)

The EB RTSM is used to model ARM application processors. The MPS RTSM is used to model the ARM Cortex™-M3 processor only.

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 the general description of software models.

Chapter 2 ***Getting Started with EB and MPS RTSMs***

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

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

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

Glossary Read this for definitions of terms and abbreviations used in this book.

Conventions

Conventions that this book can use are described in:

- *Conventions*
- *Signals* on page x.

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 Manual* (ARM DDI 0100).

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

- *RealView® Emulation Baseboard User Guide* (ARM DUI 0411)
- *Cycle Accurate Debug Interface Developer Guide* (ARM DUI 0444)
- *Fast Models User Guide* (ARM DUI 0370)
- *Fast Models Reference Manual* (ARM DUI 0423)
- *ARM® Profiler User Guide* (ARM DUI 0412)
- *RealView® Debugger Essentials Guide* (ARM DUI 0181)
- *RealView® Debugger User Guide* (ARM DUI 0153)
- *RealView® Debugger Target Configuration Guide* (ARM DUI 0182).

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

- *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 EB:

- *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 if appropriate.

Feedback on content

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

- the title
- the number, ARM DUI 0424F
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

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

1.1 Introduction to system models

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

The software models provide a *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.

1.2 Introduction to the EB RTSM

The EB and Core Tiles are hardware development platforms produced by ARM.

The EB RTSMs are system models implemented in software. They are developed using the ARM® Fast Models™ library product.

Note

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

See also:

- *About the EB and Core Tile hardware*
- *About the EB Real-Time System Models* on page 1-4.

1.2.1 About the EB and Core Tile hardware

The major components on the hardware version of the baseboard are:

- two tile sites (supports ARM Core Tiles and Logic Tiles)
- *Field Programmable Gate-Array* (FPGA) that implements a bus matrix, configuration interface, peripheral controllers, and interface logic
- 8MB configuration flash that holds FPGA images
- 256MB of 32-bit wide DDR SDRAM
- 4MB of 32-bit wide Cellular (Pseudo-static) RAM
- 64MB of 32-bit wide NOR flash
- up to 320MB (5x64MB) of static memory (flash or RAM) in an optional PISMO expansion board
- PCI expansion connector
- USB interface controller IC and connector
- Ethernet interface controller IC and connector
- connectors for VGA, color LCD display interface board, four UARTs, GPIO, keyboard, mouse, Smart Card, audio, MMC, and SSP
- electronic switches that select between the controllers located in the FPGA or on one of the tile sites
- debug and test connectors for JTAG, Integrated Logic Analyzer, and Trace port
- general purpose DIP switches and LEDs
- 2 row by 16 character LCD display
- power supply circuitry
- *Real-Time Clock* (RTC)
- time of year clock with backup battery

- programmable clock generators.

1.2.2 About the EB Real-Time System Models

The Real-Time System Models for the EB Reference System model the following components:

- Processor Core Tile options:
  - Cortex™-A9
  - Cortex-A8
  - Cortex-A5
  - Cortex-R4
  - ARM1176JZF-S™
  - ARM1136JF-S™
  - ARM926EJ-S™.
- EB model with:
  - 64MB Flash memory
  - 256MB RAM
  - Ethernet interface
  - UART interface
  - debug DIP switches and LEDs
  - *Real-Time Clock* (RTC)
  - time of year clock
  - programmable clock generators
  - *Synchronous Serial Port Interface* (SSPI)
  - DMA controller configuration registers
  - *Static Memory Controller* (SMC).

The EB RTSM also includes virtual components:

- visualization for CLCD display, keyboard and mouse
- touch screen controller
- four telnet terminals.

The Real-Time System Models for the EB Reference System are hierarchical models that consist of:

- the top-level view of the model
- the EB model
- the Core Tile model that is used by the system model.

The EB 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 architecture v5 and v6 processors, and the related write buffers, are not implemented.

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

For more detail on the differences, see *Differences between the EB and Core Tile hardware and the models* on page 3-18.

Top-level view of an EB model

A block diagram of the top-level model for an EB with an ARM1176JZF Core Tile is shown in Figure 1-1.

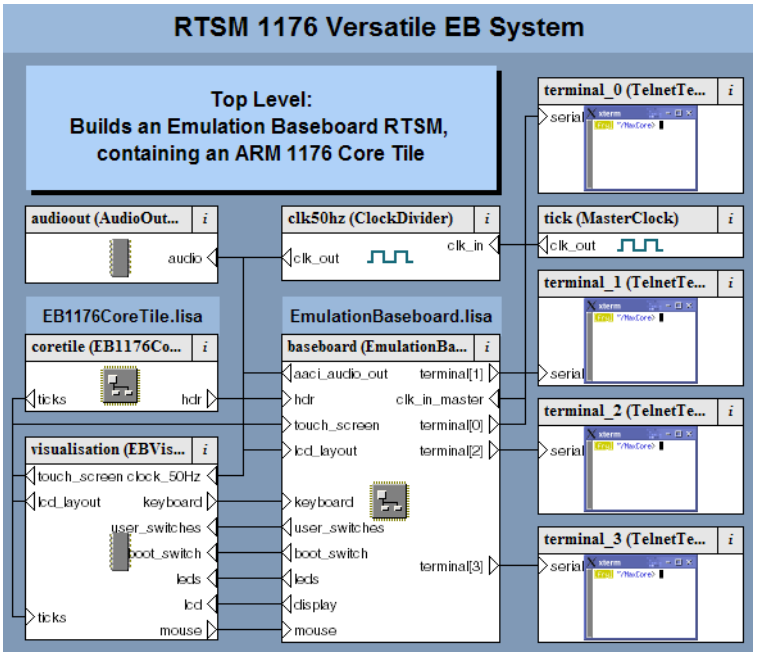

Figure 1-1 Block diagram of top-level EB model

Core Tile component

The Core Tile component provides the processor version and the associated ports that enable interconnection with other top-level components. The block diagram of the model for the ARM1176JZF Core Tile is shown in Figure 1-2.

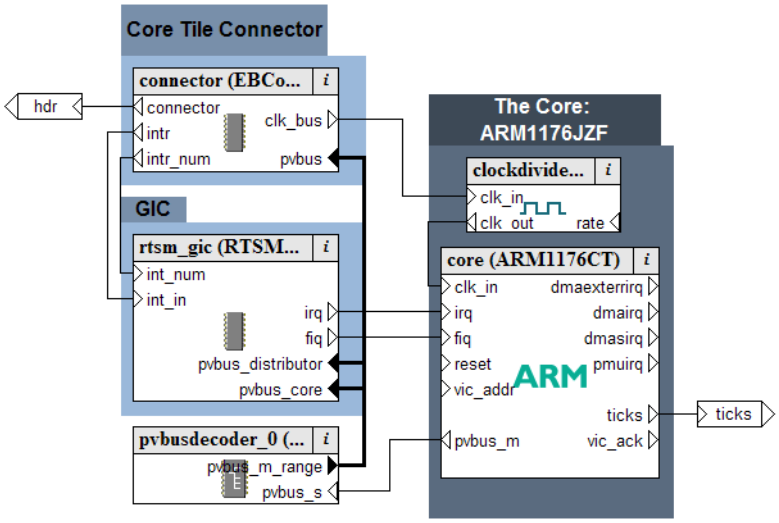

Figure 1-2 ARM1176 Block diagram of Core Tile model

EB component

The block diagram of the EB Baseboard model is shown in Figure 1-3. The figure shows the components in block form, the ports, and the interconnections between them.

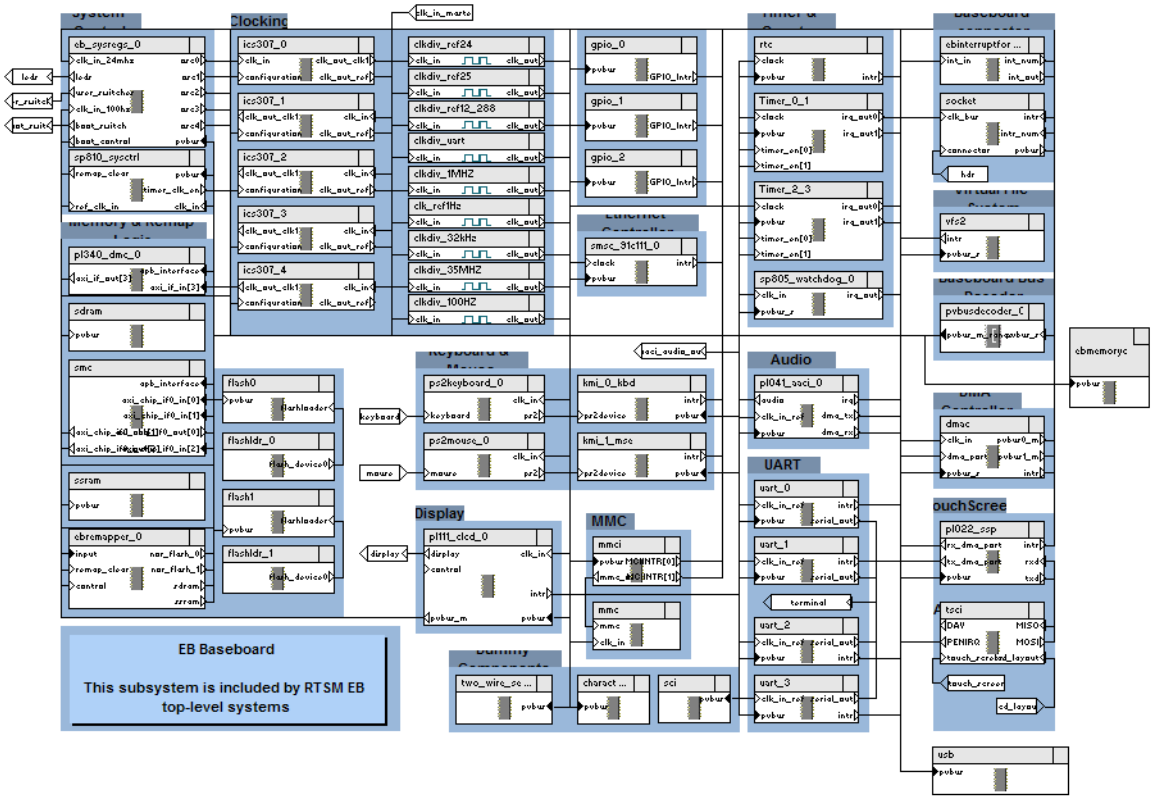

Figure 1-3 Block diagram of the EB Baseboard model

1.3 Introduction to the MPS RTSM

The Microprocessor Prototyping System 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 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 architecture v5 and v6 processors, and the related write buffers, are not implemented
- ETM is not modeled.

1.3.2 About the MPS RTSM

The *Microcontroller Prototyping System Real-Time System Model* (MPS RTSM) models in software some of the functionality of the Microcontroller Prototyping System hardware.

A complete model implementation of the MPS platform includes both MPS-specific components and generic ones such as buses and timers. Figure 1-4 shows a block diagram of an MPS RTSM.

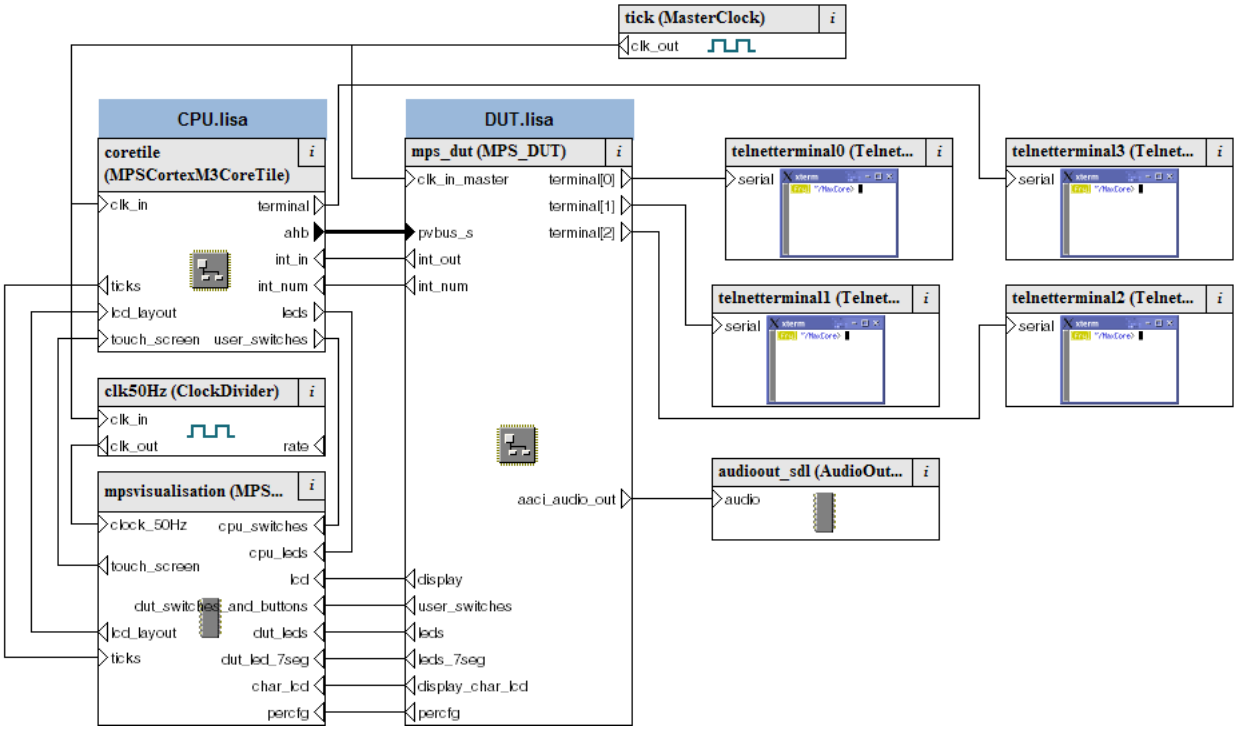

Figure 1-4 MPS RTSM block diagram

Chapter 2
Getting Started with EB and MPS RTSMs

This chapter describes the procedures for starting and configuring an EB RTSM or a Microcontroller Prototyping System RTSM, and running a software application on the model. The procedures differ depending on the ARM software tools you are using. Read the sections that apply to the software you have. This chapter contains the following sections:

2.1 Getting started with RealView Debugger

This section describes how to use RealView® Debugger to connect to an EB RTSM or an MPS RTSM. It assumes that you are using the RTSMs provided with RealView Development Suite v4.1 Professional edition.

For more information on using RealView Debugger, see the *RealView® Debugger User Guide*.

Note

The RTSMs supplied with Fast Models (version 4.0 or later) only work with RealView Debugger (version 3.1 or later).

The instructions in this section apply to RealView Debugger v4.0 and later.

Using other versions of RealView Debugger might not be successful because connection methods might change between releases.

You cannot generate an ARM Profiler analysis file for an RTSM through RealView Debugger, even if you configure the model to enable profiling.

The following pre-built RTSMs are supplied:

- `RTSM_EB_Cortex-A9_MPx2` with Cortex-A9 MP dual core
- `RTSM_EB_Cortex-A9_MPx1` with Cortex-A9 MP single core
- `RTSM_EB_Cortex-A8` with Cortex-A8
- `RTSM_EB_Cortex-A5_MPx2` with Cortex-A5 MP dual core
- `RTSM_EB_Cortex-A5_MPx1` with Cortex-A5 MP single core
- `RTSM_EB_Cortex-R4` with Cortex-R4
- `RTSM_MPS_Cortex-M3` with Cortex-M3
- `RTSM_MPS_Cortex-M4` with Cortex-M4
- `RTSM_EB_Cortex-ARM1176` with ARM1176JZF
- `RTSM_EB_Cortex-ARM1136` with ARM1136JF
- `RTSM_EB_Cortex-ARM926` with ARM926EJ.

See also:

- *Connecting to the RTSM in RealView Debugger*
- *Loading and running an application on the EB RTSM* on page 2-9.

2.1.1 Connecting to the RTSM in RealView Debugger

See the following sections for details on the ways you can connect to an RTSM:

- *Starting the RTSM as an RTSM connection* on page 2-3
- *Connecting to a running model using RealView Debugger* on page 2-3.

Note

You must not connect to more than one RTSM at any one time in RealView Debugger. If you do, you might experience unexpected behavior or crashes.

Starting the RTSM as an RTSM connection

Follow the steps below to add the RTSM to the RealView Debugger Connect to Target window under the RTSM debug interface configuration.

1. In RealView Debugger, select **Connect to Target...** from the **Target** menu to open the Connect to Target window.
2. Click the **Add** button beside the RTSM debug interface name. This opens the Model Configuration Utility window:
   - The RTSM models are automatically displayed in the list based on the path set by the `ARM_RTSM_PATH` variable.
   - If the `ARM_RTSM_PATH` variable has not been set, click the **Browse...** button to open a file browser for locating the RTSM. You can find these models in:

     `install_directory``\RVDS\Models`

     The model file names are of the form:

     — `RTSM_EB_`*`processor`*`.dll` or `RTSM_MPS_`*`processor`*`.dll` on Windows

     — `RTSM_EB_`*`processor`*`.so` or `RTSM_MPS_`*`processor`*`.so` on Linux.

     where *processor* indicates the supplied processor model such as, for example, `1176`.
3. Select the model to use in the Models pane on the left side of the Model Configuration Utility:
   - Configure the device parameters if required. See *Using a configuration GUI in RealView Debugger* on page 2-7.
   - After you have finished, or if no parameters required configuration, click **OK**.
4. You can rename the entry to a more descriptive name. In the RealView Debugger Connect to Target window, left-click on your newly-created target to display the context menu and select **Rename**. Enter the new name in the entry field.
5. In the RealView Debugger Connect to Target window, double-click on your newly-created target to connect to it. If you are grouping targets by Configuration, expand the target connection tree view to see your target instance. Connecting to a target opens a CLCD window.

Connecting to a running model using RealView Debugger

You can use RealView Debugger to connect to an already running Model Shell instance of the RTSM. You can make multiple debugger connections to a single model instance.

1. Start Model Shell, if it is not already running, as described in *Getting started with Model Shell* on page 2-6.
2. In RealView Debugger, select **Connect to Target...** from the **Target** menu to open the Connect to Target window.
3. Select **Configuration** from the Grouped By drop-down list
4. Click **Add** for the Model Process Debug Interface.

   The Model Configuration Utility dialog box is displayed containing a list of the devices for the running RTSM
5. Click **OK**.

   The Model Configuration Utility dialog box closes, and the new Debug Configuration is added to the Model Process Debug Interface.

6. In the RealView Debugger Connect to Target window, double-click on your newly-created RTSM target to connect to it.

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 disable semihosting support for the current connection, modify the value of the `@Semihosting_State` register in RealView Debugger. You can change the value from either:

- the **Semihost** tab of the Register pane
- the command-line by entering:

  ```
  RVD> setreg @Semihosting_State=0
  ```

To permanently disable semihosting for a Debug Configuration, use the Connection Properties dialog box:

1. Display the Connect to Target window.
2. Set **Grouped By** to **Configuration**.
3. Right-click on the required Debug Configuration to display the context menu.
4. Select **Properties** to display the Connection Properties dialog box.
5. Select the **Base Setting** tab and change the option.

2.2 Getting started with ARM Profiler

ARM Profiler v2.1.2 can be used to profile the cores in an EB RTSM and MPS RTSMs supplied with RealView Development Suite v4.1 Professional edition.

For multiprocessor EB RTSMs it is possible to profile one of the cores, but it is not possible to profile both cores at once.

For MPS RTSMs it is possible to profile the Cortex-M3 MPS RTSM using the ARM Profiler.

Note

RealView Debugger does not support RTSM profiling.

For more information on the ARM Profiler, see the *ARM® Profiler User Guide*.

2.3 Getting started with Model Shell

This section describes how to use the Model Shell application to start and configure an EB RTSM or an MPS RTSM. An example of loading and executing an application is documented separately. See *Loading and running an application on the EB RTSM* on page 2-9.

The RTSM can be started with its own CADI debug server. This enables the model to run independently of a debugger such as ARM RealView 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 and enter the following at the command prompt:

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

where:

model_name is the name of the model file. By default this file name is typically RTSM_EB_*processor*.dll or RTSM_MPS_*processor*.dll on Windows or RTSM_EB_*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 2-7.

instance.parameter=value

is the optional direct setting of a configuration parameter. See *Using the command line* on page 2-8.

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

Note

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

```
install_directory\RVDS\Models\..\bin\model_shell
```

For more information on all Model Shell options, see the *Model Shell for Fast Models Reference Manual*.

Starting the model opens the RTSM CLCD display. See *Using the CLCD window* on page 2-14.

After the RTSM starts, you can use RealView Debugger to connect to it. See *Getting started with RealView Debugger* on page 2-2.

2.4 Configuring the RTSM

This section describes how to configure EB and MPS RTSMs.

Note

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

See also:

- *Using a configuration GUI in RealView Debugger*
- *Setting model configuration options from Model Shell*.

2.4.1 Using a configuration GUI in RealView Debugger

In RealView Debugger, you can configure the RTSM parameters before you connect to the model and start it:

- If you are adding the particular RTSM to the RealView Debugger Connect to Target window, the Model Configuration Utility dialog box is displayed automatically.
- Alternatively you can right click on your existing target in the Connect to Target list and select **Configure...** to open the same dialog.

To view the configuration parameters, scroll down the list of devices shown in the upper right pane. Selecting a device populates the lower right pane with the device parameters.

To change a parameter value, select a boolean from the drop down list, or enter data such as strings or addresses by clicking in the relevant field. Hovering the mouse pointer over a device or parameter shows a description or additional information.

You can change the numeric display from decimal to hexadecimal by right clicking on the parameter value in the lower right pane and selecting **Hexadecimal Display** from the resulting context menu.

2.4.2 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 2-6.

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.

A typical configuration file is listed in Example 2-1 on page 2-8:

Example 2-1 Configuration file

```
# Disable semihosting using true/false syntax
coretile.core.semihosting-enable=false
#
# Enable the boot switch using 1/0 syntax
baseboard.sp810_sysctrl.use_s8=1
#
# Set the boot switch position
baseboard.eb_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. You can also use `--parameter` as a synonym for the `-C` switch. See *Getting started with Model Shell* on page 2-6. Use the same syntax as for a configuration file, but each parameter must be preceded by the `-C` switch.

Examples

This section contains examples of configuring an MPS RTSM using Model Shell.

Example 2-2 shows how to set the boot options from Model Shell.

Example 2-2 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_Merlin.so
```

2.5 Loading and running an application on the EB RTSM

Example applications are provided for use with the RTSMs for the Emulation Baseboard.

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 EB RTSM is:

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

If you are using Fast Models, the examples are in the `%PVLIB_HOME%\images` directory.

In RVDS, the source code is available in the directory `%ARMROOT%\Examples\…\…\platform\mandelbrot`.

See also:

- *Running the brot application in ARM Profiler*
- *Running the brot application in RealView Debugger* on page 2-12.

2.5.1 Running the brot application in ARM Profiler

This section briefly describes the steps to use with ARM Profiler to load the model, run the `brot.axf` image and display the profiling results. For more detailed information on configuration options, see the *ARM® Profiler User Guide*.

Note

Virtual platforms built with versions of Fast Models prior to 5.0 SP1 are not compatible with ARM Profiler v2.1.2.

RealView Debugger does not support RTSM profiling.

1. Start the ARM Workbench IDE.
2. From the main menu, select **Open Run Dialog...** from the **Run** menu. This opens the Run dialog shown in Figure 2-1 on page 2-10.

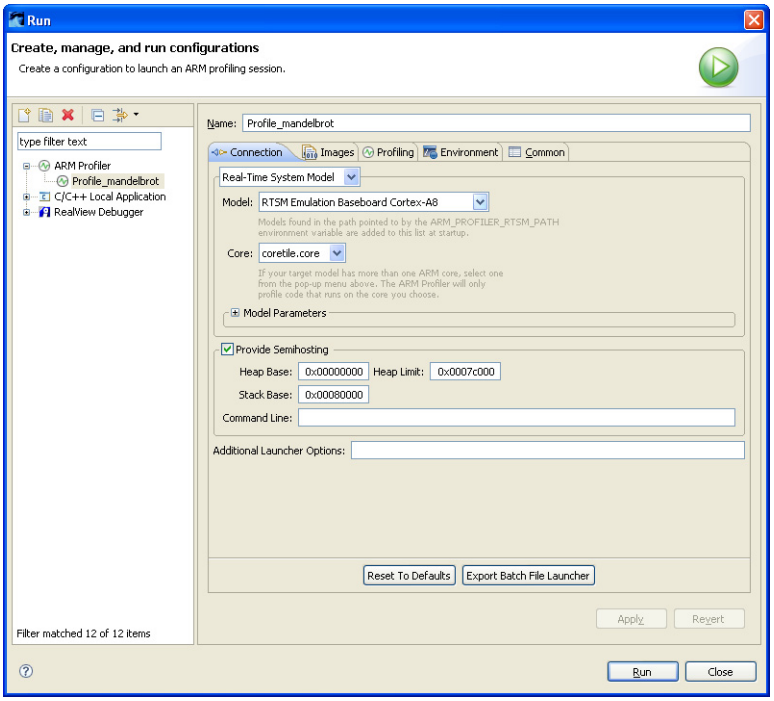

Figure 2-1 ARM Profiler Run dialog - Connection tab

3. Select the model to run in the **Model** drop down list by either:
   - choosing one of the RTSMs supplied with ARM Profiler
   - using the **Custom** option if you have built your own model.
4. Select the **Images** tab.
5. Browse to the location of the `brot.axf` image in the **Image** field.

   Note

   You might have to rebuild the Mandelbrot example image in:

   `install_directory``\RVDS\Examples\...\...\...\mandelbrot`
6. Select the **Profiling** tab as shown in Figure 2-2 on page 2-11.

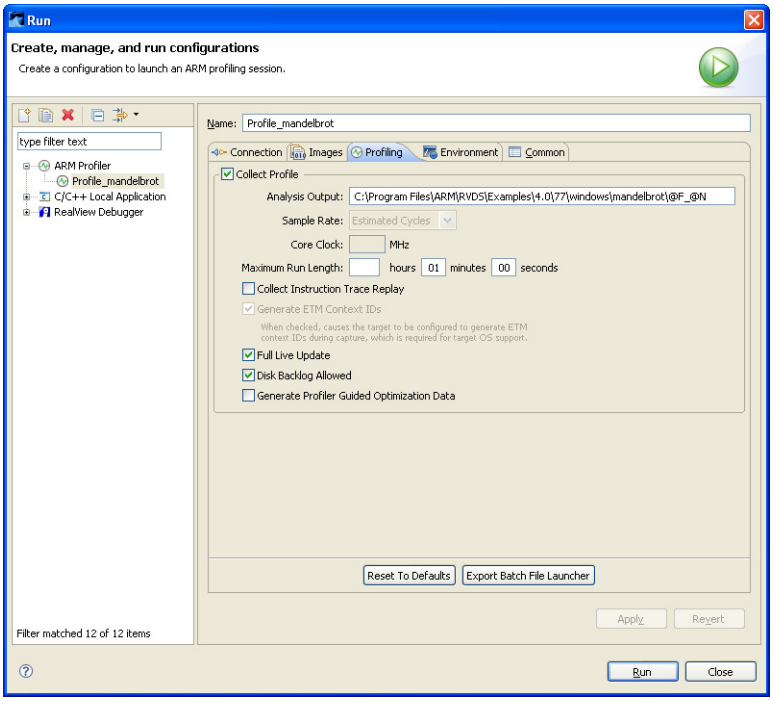

Figure 2-2 ARM Profiler Run dialog - Profiling tab

7. Set **Maximum Run Length** to one minute to specify the execution time limit.
8. Click **Run** to start the model running.
9. When execution stops, an analysis similar to that in Figure 2-3 on page 2-12 is shown.

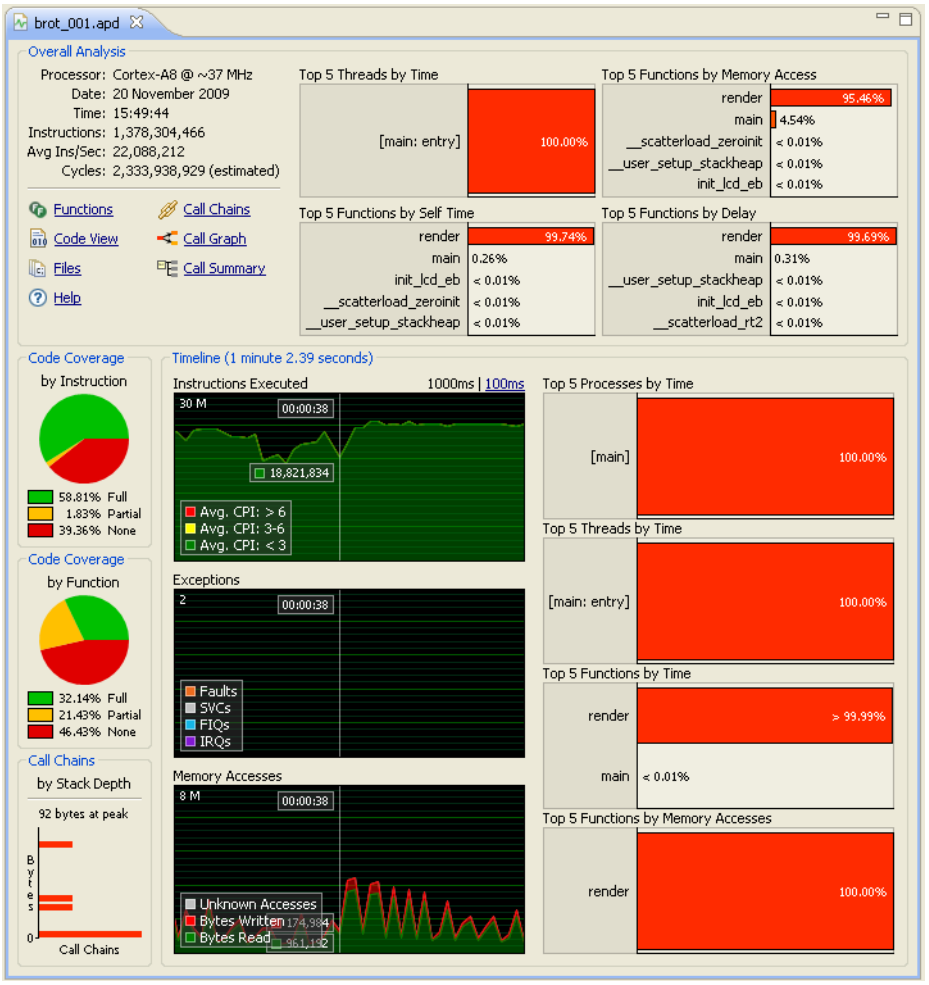

Figure 2-3 ARM Profiler analysis

2.5.2 Running the brot application in RealView Debugger

This section describes the steps to load and run the `brot.axf` image in RealView Debugger:

Note

It might be necessary to build the `brot.axf` image before loading the application. Instructions and build scripts are in `%ARMROOT%\Examples\...\...\platform\mandelbrot`.

1. Start RealView Debugger and connect to the system model as described in *Connecting to the RTSM in RealView Debugger* on page 2-2.
2. Select **Load Image** from the **Target** menu. The Select Local Files to Load dialog is displayed.
3. Browse to the location of `brot.axf`, Select it and click **Open**.
4. Select the **brot.c** tab in the RealView Debugger main window and place a breakpoint on the `render` function, as shown in Figure 2-4 on page 2-13.

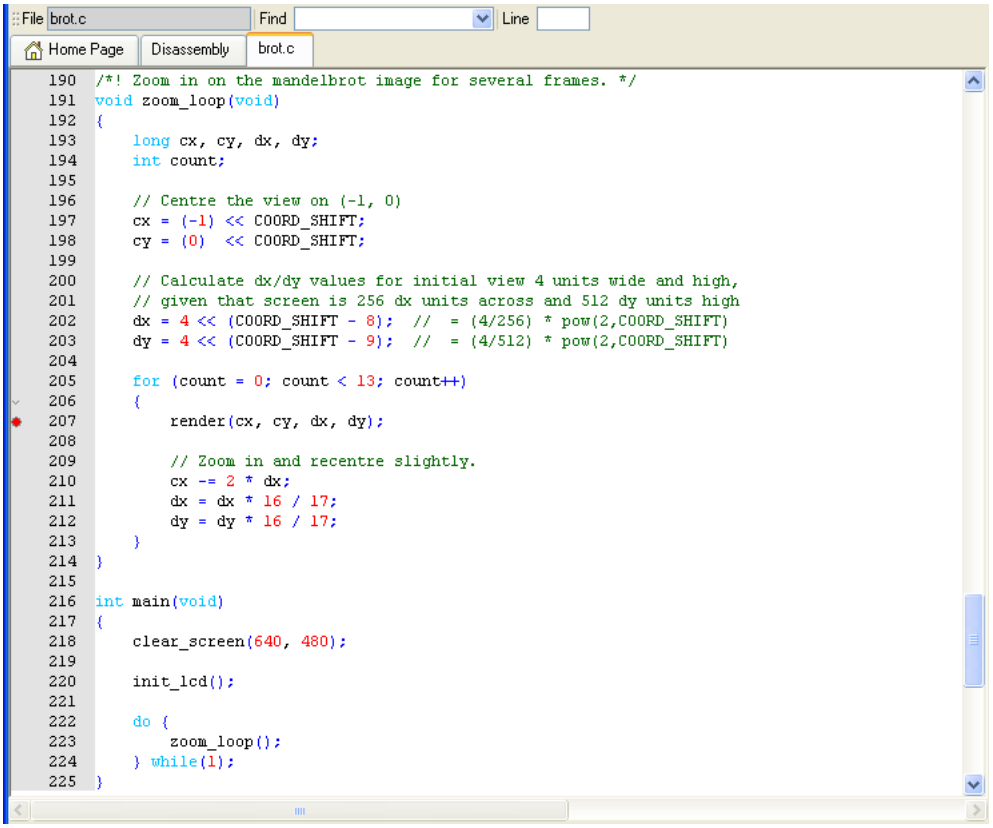

Figure 2-4 Breakpoint in brot.c

5. Press **F5**, or select **Run** from the **Debug** menu, until the CLCD window is displayed as shown in Figure 2-5 on page 2-14.

6. Repeatedly press **F5** and observe the changes in the CLCD window.

2.6 Using the CLCD window

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

This window represents the contents of the simulated color LCD framebuffer. 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*.

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

- *Using the EB CLCD window*
- *Using the MPS Visualization window* on page 2-16

2.6.1 Using the EB CLCD window

Figure 2-5 shows the EB RTSM CLCD in its default state, immediately after being started.

Figure 2-5 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 EB User DIP switches:

These represent switch S6 on the EB 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. See *Switch S6* on page 3-6.

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

These represent switch S8 on the EB 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. See *Switch S8* on page 3-7.

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 EB User LEDs.

These represent LEDs D[21:14] on the EB 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 EB 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 3-23.

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 2-6 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 2-6. You can click on the items again to toggle between the original and alternative displays.

Figure 2-6 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. Refer to the CADI parameter documentation for details in the *Fast Models Reference Manual*.

2.6.2 Using the MPS Visualization window

When an MPS RTSM starts, the Real-Time System Model CLCD window is opened:

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

Figure 2-7 shows the MPS RTSM CLCD in its default state, immediately after being started.

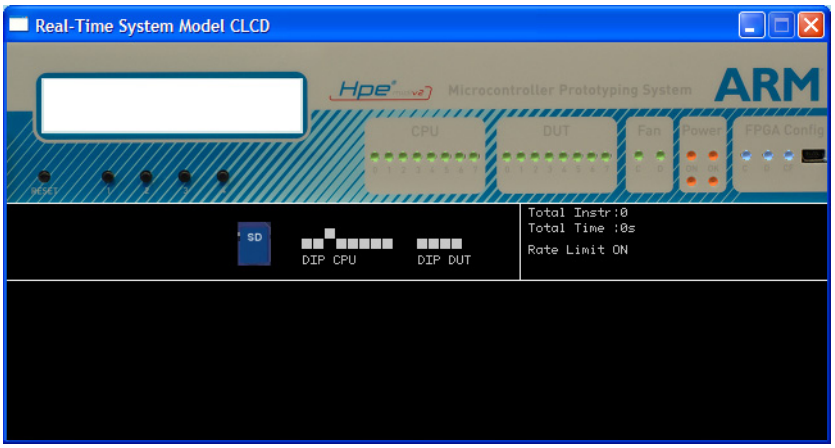

Figure 2-7 Visualization window at startup

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

Four 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.

Because the system model models provide a programmer's view of the system, 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 2-8 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 2-8.

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

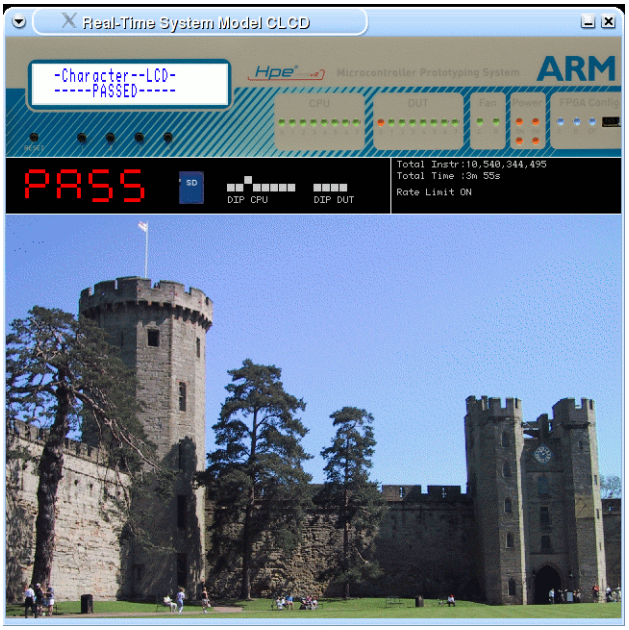

Figure 2-8 Visualization window with CLCD buffer displayed

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. Refer to the CADI parameter documentation for details in the *Fast Models Reference Manual*.

2.7 Using Ethernet with an EB RTSM

The EB RTSMs provide you with a virtual Ethernet component. This is a model of the SMSC91C111 Ethernet controller. You can configure the component to use a host Ethernet port, or connect to an Ethernet port on another host over TCP/IP. By default, the component starts as an unconnected Ethernet port.

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

- *Host requirements*
- *Target requirements* on page 2-20
- *Configuring Ethernet* on page 2-23.

2.7.1 Host requirements

Before you can use the Ethernet capability of the EB RTSM, you must first set up your host computer. Unless your applications have direct access to the network devices on the host, you must install the `nicserver` proxy application. It might also be necessary to install packet capture software.

nicserver

The `nicserver` application has two main functions:

- to provide a connection to the host packet capture facilities
- to provide a remote connection target.

Not all hosts allow non-administrator or non-root processes to access network devices. To work around this, you are given the `nicserver` application that acts as a proxy. This workaround is useful on Linux, because typically only root has access to network devices. On Windows, you must have at least standard user permissions to use the `nicserver` application, as a restricted user is unable to access the necessary devices.

The `nicserver` application has uses other than as a proxy, however. For example, you can use `nicserver`, running on a different computer, as a target for your RTSM Ethernet accesses.

Note

The `nicserver` application is included with RVDS, in the directory:

`install_directory``\RVDS\Models\...\...\bin`

The syntax for `nicserver` is:

`nicserver [-a` *`adapter`*`] [-dedicated] [--help] [-l] [-n` *`IP_address`*`] [-p` *`port`*`] [-shared] [-version]`

The command line arguments are:

`-a` *`adapter`* The host Ethernet adapter on which to send and receive Ethernet packets. Use a device name provided by the `nicserver -l` command. This name can be either part of the device text description, such as "`wireless`" or the manufacturer's name, or a complete device ID, such as "`eth0`" or "`\Device\NPF_{1FBF9456-7A62-43AB-B683-83F4142FB7E6}`". If the name is not unique, the first match as shown in the `nicserver -l` list is used.

`-dedicated` Run in non-promiscuous mode. Use this option if you are using the pipe transport to bind the `nicserver` instance to the specific RTSM that uses it.

`--help` Print out a summary of `nicserver` commands then quit.

`-l` List the available network adapters on the host then quit. Sample output from `nicserver -l` on Windows might look like this:

```
C:\> nicserver -l
Available network adapters:
\Device\NPF_{1FBF9456-7A62-43AB-B683-83F4142FB7E6}
(NetworkCardInc Wireless Pro Network Connection)
\Device\NPF_{924EB4D2-6588-438C-7115-DACFD1754EA2}
(NetworkCardInc 100Gbit Ethernet Network Connection)
```

`-n` *`IP_address`*

Specify the IP address for `nicserver` to bind to. Together with the port, the IP address forms the TCP/IP socket for `nicserver`.

`-p` *`port`* Specify the port for `nicserver` to bind to. Together with the IP address, the port forms the TCP/IP socket for `nicserver`.

`-shared` Run in promiscuous mode. This is the default.

`-version` Print the `nicserver` version then quit.

You can start the `nicserver` application at a command prompt with, for example:

```
nicserver -p 7010 -n 192.168.0.42 -a NetworkCardInc
```

This command starts `nicserver` in promiscuous mode with a TCP/IP socket of 192.168.0.42:7010 and uses the first available NetworkCardInc network controller on the host. If the command has succeeded, a message is output to confirm that `nicserver` is listening on a given IP address and port. To terminate `nicserver`, issue the normal SIGINT for your terminal, such as **Ctrl + C**. If the `nicserver` command fails, you are shown an error message, or you are returned straight to the command prompt. In either case, check your settings and try again.

On Linux, you might require root privileges to use `nicserver`. If so, you must request that an administrator uses `setuid` to enable `nicserver` to be run with root permissions.

On Windows, you must have the WinPcap driver installed to use `nicserver`. See *Packet capture*.

Packet capture

On Microsoft Windows the Ethernet component depends on the WinPcap driver being installed. This enables the unique identification of Ethernet devices if more than one is present. You can find the WinPcap distribution at `http://www.winpcap.org/`. Installing the binary distribution for Microsoft Windows installs the correct driver. A user with administrator privileges must install the software, and the end user must have at least Power User access privileges. Use WinPcap version 3.1 beta 4 or later.

On supported Linux systems, the Pcap packet capture library is present by default. No additional software is required on Linux for `nicserver` to work.

2.7.2 Target requirements

The EB RTSMs include a software implementation of the SMSC91C111 device. 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 Ethernet component parameters. When you configure these parameters prior to starting the EB RTSM, you can specify which host interface to use, set the MAC address, and define whether promiscuous mode is enabled. These parameters are:

- *Ethernet interface* on page 2-21

- *mac_address* on page 2-23
- *promiscuous* on page 2-23.

Configuration file syntax for the Ethernet component is given elsewhere in this document. See *Ethernet parameters* on page 3-8.

Ethernet interface

The interface parameter setting in the EB RTSM Ethernet component sets what host interface to use. When enabled, the Ethernet component can use one of two transports, *host* or *pipe*:

Host transport

A block diagram showing the host transport is shown in Figure 2-9.

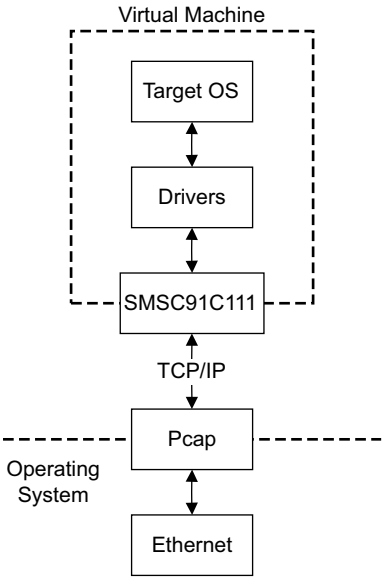

Figure 2-9 Host transport block diagram

In the Figure, the Virtual Machine is your EB RTSM, and the SMSC91C111 block represents the configurable Ethernet component. When using the host transport, the Ethernet component communicates directly using TCP/IP with the packet capture component, Pcap. This means that your model must have sufficient permissions to access Pcap directly. This is normally not an issue on Windows if you have administrator rights. On Linux, you might find that you cannot use the host transport unless you have root privileges. This is not recommended for standard user applications. An alternative for Linux is to instead use the pipe transport.

After data reaches the Ethernet component block, it can be handled in the same way as any other Ethernet data. For example, you can connect to a `nicserver` session running on another host.

Pipe A block diagram, showing the situation if you use the pipe transport, is shown in Figure 2-10 on page 2-22.

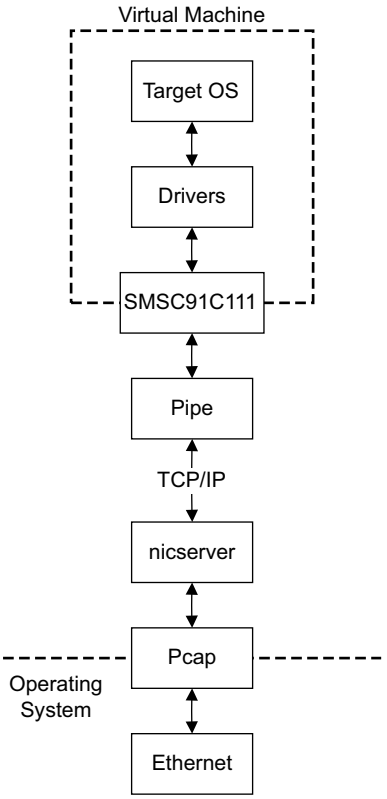

Figure 2-10 Pipe transport block diagram

The Virtual Machine represents your EB RTSM, with the SMSC91C111 block as the Ethernet component interface to the world outside of the model. The pipe block provides a TCP/IP address and port with which the `nicserver` application communicates. This enables you to disregard the specifics of the network hardware on your host system, and instead rely on `nicserver` to route communications appropriately.

The `nicserver` application here acts as a proxy between you and the real network hardware, and removes the requirement for direct permission to use network resources. This might be particularly useful on Linux platforms, where `nicserver` can be given root permissions but still be run by a user. Because the connection is made over TCP/IP, the pipe block can communicate with a `nicserver` application running on a completely different computer from that on which the EB RTSM is running.

The EB RTSM Ethernet interface parameter must be configured with one of the three following values:

disabled Implement the Ethernet component as though no cable were connected. This means you can send packets to the component but they are not sent elsewhere. This is the default state.

host:*controller*

The *controller* is the first host Ethernet controller that matches the string specified. On Linux, you can normally use the name of a specific adapter, such as `eth0`. On Windows, matters are more complex because you might have two network adaptors with similar names.

If *controller* is set to "`NetworkCardInc`", the first match in the `nicserver` list is used if there is more than one device with that string in the name.

If you require a different controller, you must specify as much of the name as necessary for a unique match. Alternatively you can use the device ID supplied by `nicserver`.

Use the **host:*controller*** setting if you require the EB RTSM Ethernet component to connect directly to an Ethernet adaptor on the host computer.

pipe:*address*:*port*

The numeric *address* and *port* values are those of the `nicserver` application, if used on the host. See *nicserver* on page 2-19. Use this setting if you require the EB RTSM Ethernet component to connect to a virtual network interface card running on the host computer.

mac_address

You must define whether the MAC address of the Ethernet component is to be either a fixed or a random value.

Specifying the MAC address in a format analogous to the default value of 00:01:02:03:04:05 defines a static MAC address that does not change from one model invocation to the next.

Specifying the MAC address as **auto** generates a random local MAC address, with bit 1 set and bit 0 clear. A different IP address is allocated each time the simulator is reset. The chances of the address clashing with an existing MAC address are small but you are discouraged from using this method if IP addresses are being allocated automatically by a DHCP server.

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 EB RTSM Ethernet component.

As an example, consider that you are using the nicserver application and have set it up to listen on port 7010 and have an IP address of `192.168.0.42`. The Ethernet device on the EB RTSM has a static MAC address of `0012AB9CE830` and starts in promiscuous mode. The corresponding syntax to use in a configuration file is:

```
baseboard.smsc_91c111_0.interface=pipe:192.168.0.42:7010
baseboard.smsc_91c111_0.mac_address=00:12:AB:9C:E8:30
baseboard.smsc_91c111_0.promiscuous=T
```

If you are using a GUI configuration interface, you must modify the parameters corresponding to those given in the configuration file syntax example.

2.7.3 Configuring Ethernet

This section describes how to configure a connection to the Ethernet interface from Windows or Linux.

On Windows the Ethernet component requires installation of the WinPcap driver. This enables the unique identification of Ethernet devices if more than one is present.

On supported Linux systems, the Pcap packet capture library is present by default. No additional software is required on Linux for the `nicserver` to work.

Windows setup

To configure a connection to the Ethernet interface:

1. Install the WinPcap software.

   The WinPcap distribution is at `http://www.winpcap.org/`. Installing the binary distribution for Microsoft Windows installs the correct driver. A user with administrator privileges must install the software, and the end user must have at least Power User access privileges. Use WinPcap version 3.1 beta 4 or later.

2. Open a terminal window and enter the following command:

   ```
   nicserver -l
   ```

3. The nicserver application lists the network drivers. For example:

   ```
   Available network adapters:
   \Device\NPF_{D2857DCE-C557-4F74-8C7F-F2A29C6113CC}
   (Intel(R) 82567LM Gigabit Network Connection)
   ```

4. Enter the following command to configure a proxy port for one of the available devices. For example, enter:

```
nicserver -n 127.0.0.1 -p 50000 -a NPF_{D2857DCE-C557-4F74-8C7F-F2A29C6113CC}
```

   to bind port 50000 at address 127.0.0.1 to the network card.

5. The nicserver confirms the setting:

   ```
   listening on 127.0.0.1 port 50000
   ```

6. In the SMSC_91C111 model parameters, set the `interface` option to `pipe:127.0.0.1:50000`.

7. The model now sends all network traffic to the IP address and port of the nicserver and it is proxied.

Linux setup

On supported Linux systems, the Pcap packet capture library is present by default. No additional software is required on Linux for `nicserver`.

To install a proxy server:

1. Open a terminal window and enter the following command:

   ```
   nicserver -l
   ```

2. The nicserver application lists the network drivers. For example:

   ```
   Available network adapters:
   any
     (Pseudo-device that captures on all interfaces)
   eth0
   lo
   ```

3. Enter the following command to configure a proxy port for one of the available devices. For example, enter:

   ```
   nicserver -n 127.0.0.1 -p 50000 -a eth0
   ```

   to bind port 50000 at address 127.0.0.1 to the network card.

4. The nicserver confirms the setting:

   ```
   listening on 127.0.0.1 port 50000
   ```

5. In the SMSC_91C111 model parameters, set the `interface` option to `pipe:127.0.0.1:50000`.
6. The model now sends all network traffic to the IP address and port of the nicserver and it is proxied.

2.8 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 Windows Vista client, you must first install Telnet. The Telnet application is not installed on Windows Vista by default.

Download the application by following the instructions on the Microsoft website. Search for "Windows Vista 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 uninstall or change programs.
2. Select **Turn Windows features on or off** on the left side of the bar. This opens the Windows Features dialog. Select the **Telnet Client** check box.
3. Click **OK**. The installation of Telnet might take several minutes to complete.

A block diagram of one possible relationship between the target and host through the Terminal component is shown in Figure 2-11. The TelnetTerminal block is what you configure when you define Terminal component parameters. The Virtual Machine is your EB RTSM or MPS RTSM.

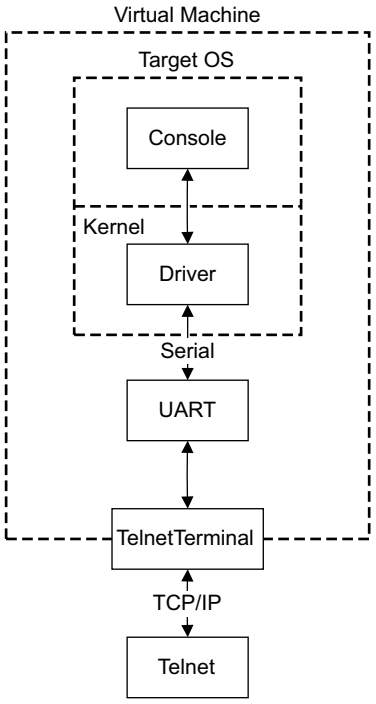

Figure 2-11 Terminal block diagram

On the target side, the console process invoked by your target OS relies upon 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, that 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 EB RTSM or MPS RTSM start 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*.

2.8.1 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.

2.8.2 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 simply acts as a TCP/IP port. You can use this feature to directly connect to your target through the Terminal component.

2.9 Virtual filesystem

The *Virtual FileSystem* (VFS) allows 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 2-29.

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*. See also the `WritingADriver.txt` file in `%PVLIB_HOME%\VFS\docs\`.

Note

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

2.9.1 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 link functions are implemented to support symbolic links in Linux, but they are not currently implemented. File permissions are defined but not implemented.

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.

2.9.2 Using the VFS with a pre-built RTSM

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

To use the VFS functionality of the EB RTSM, use the `baseboard.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 *EB RTSM baseboard parameters* on page 3-6.

Mount names

Once 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 2-28. 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 EB RTSMs

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

- *EB model memory map* on page 3-2
- *EB model configuration parameters* on page 3-5
- *Differences between the EB and Core Tile hardware and the models* on page 3-18.

Note

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

3.1 EB model memory map

Table 3-1 lists the locations and interrupts for memory, peripherals, and controllers used in the EB Real-Time System Models. See the *Emulation Baseboard User Guide* for more detail on the controllers and peripherals.

Table 3-1 Memory map and interrupts for standard peripherals

Peripheral	Modeled	Address range	Bus	Size	GIC Int[a]	DCCI Int[b]
Dynamic memory	Yes	0x00000000–0x0FFFFFFF	AHB	256MB	-	-
System registers (see *Status and system control registers* on page 3-22)	Yes	0x10000000–0x10000FFF	APB	4KB	-	-
SP810 System Controller	Yes	0x10001000–0x10001FFF	APB	4KB	-	-
Two-Wire Serial Bus Interface	No	0x10002000–0x10002FFF	APB	4KB	-	-
Reserved	-	0x10003000–0x10003FFF	APB	4KB	-	-
PL041 *Advanced Audio CODEC Interface* (AACI)	Partial[c]	0x10004000–0x10004FFF	APB	4KB	51	51
PL180 *MultiMedia Card Interface* (MCI)	Partial[d]	0x10005000–0x10005FFF	APB	4KB	49, 50	49, 50
Keyboard/Mouse Interface 0	Yes	0x10006000–0x10006FFF	APB	4KB	52	7
Keyboard/Mouse Interface 1	Yes	0x10007000–0x10007FFF	APB	4KB	53	8
Character LCD Interface	No	0x10008000–0x10008FFF	APB	4KB	55	55
UART 0 Interface	Yes	0x10009000–0x10009FFF	APB	4KB	44	4
UART 1 Interface	Yes	0x1000A000–0x1000AFFF	APB	4KB	45	5
UART 2 Interface	Yes	0x1000B000–0x1000BFFF	APB	4KB	46	46
UART 3 Interface	Yes	0x1000C000–0x1000CFFF	APB	4KB	47	47
Synchronous Serial Port Interface	Yes	0x1000D000–0x1000DFFF	APB	4KB	43	43
Smart Card Interface	No	0x1000E000–0x1000EFFF	APB	4KB	62	62
Reserved	-	0x1000F000–0x1000FFFF	APB	4KB	-	-
SP805 Watchdog Interface	Yes	0x10010000–0x10010FFF	APB	4KB	32	32
SP804 Timer modules 0 and 1 interface (Timer 1 starts at 0x10011020)	Yes	0x10011000–0x10011FFF	APB	4KB	36	1
SP804 Timer modules 2 and 3 interface (Timer 3 starts at 0x10020020)	Yes	0x10012000–0x10012FFF	APB	4KB	37	2
PL061 GPIO Interface 0	Yes	0x10013000–0x10013FFF	APB	4KB	38	38
PL061 GPIO Interface 1	Yes	0x10014000–0x10014FFF	APB	4KB	39	39
PL061 GPIO Interface 2 (miscellaneous onboard I/O)	Yes	0x10015000–0x10015FFF	APB	4KB	40	40
Reserved	-	0x10016000–0x10016FFF	APB	4KB	-	-
PL030 Real-Time Clock Interface	Yes	0x10017000–0x10017FFF	APB	4KB	-	-

Table 3-1 Memory map and interrupts for standard peripherals (continued)

Peripheral	Modeled	Address range	Bus	Size	GIC Int[a]	DCCI Int[b]
Dynamic Memory Controller configuration	Partial[e]	0x10018000–0x10018FFF	APB	4KB	-	-
PCI controller configuration registers	No	0x10019000–0x10019FFF	AHB	4KB	-	-
Reserved	-	0x1001A000–0x1001FFFF	APB	24KB	-	-
PL111 Color LCD Controller	Yes	0x10020000–0x1002FFFF	AHB	64KB	55	55
DMA Controller configuration registers	Yes	0x10030000–0x1003FFFF	AHB	64KB	-	-
Generic Interrupt Controller 1 CPU interface	Yes[b]	0x10040000–0x10040FFF	AHB	4KB	-	-
Generic Interrupt Controller 1 Distributor interface	Yes	0x10041000–0x10041FFF	AHB	4KB	-	-
Generic Interrupt Controller 2 CPU interface (nFIQ for tile 1)	No[f]	0x10050000–0x10050FFF	AHB	4KB	-	-
Generic Interrupt Controller 2 Distributor interface	Yes	0x10051000–0x10051FFF	AHB	4KB	-	-
Generic Interrupt Controller 3 CPU interface (nIRQ for tile 2)	No[f]	0x10060000–0x10060FFF	AHB	4KB	-	-
Generic Interrupt Controller 3 Distributor interface	No	0x10061000–0x10061FFF	AHB	4KB	-	-
Generic Interrupt Controller 4 CPU interface (nFIQ for tile 2)	No[f]	0x10070000–0x10070FFF	AHB	4KB	-	-
Generic Interrupt Controller 4 Distributor interface	No	0x10071000–0x10071FFF	AHB	4KB	-	-
PL350 Static Memory Controller configuration[g]	Yes	0x10080000–0x1008FFFF	AHB	64KB	-	-
Reserved	-	0x10090000–0x100EFFFF	AHB	448MB	-	-
Debug Access Port (DAP) ROM table. Some debuggers read information on the target processor and the debug chain from the DAP table.	No	0x100F0000-0x100FFFFF	AHB	64KB	-	-
Reserved	-	0x10100000-0x1FFFFFFF	-	255MB	-	-
Reserved	-	0x20000000-0x3FFFFFFF	-	512MB	-	-
NOR Flash	Yes[h]	0x40000000-0x43FFFFFF	AXI	64MB	-	-
Disk on Chip	No	0x44000000-0x47FFFFFF	AXI	64MB	41	41
SRAM	Yes	0x48000000-0x4BFFFFFF	AXI	64MB	-	-
Configuration flash	No	0x4C000000-0x4DFFFFFF	AXI	32MB	-	-
Ethernet	Yes[i]	0x4E000000-0x4EFFFFFF	AXI	16MB	60	60
USB	No	0x4F000000-0x4FFFFFFF	AXI	16MB	-	-

Table 3-1 Memory map and interrupts for standard peripherals (continued)

Peripheral	Modeled	Address range	Bus	Size	GIC Int[a]	DCCI Int[b]
PISMO expansion memory	No	0x50000000-0x5FFFFFFF	AXI	256MB	58	58
PCI interface bus windows	No	0x60000000-0x6FFFFFFF	AXI	256MB	-	-
Dynamic memory (mirror)	Yes	0x70000000-0x7FFFFFFF	AXI	256MB	-	-
Memory tile (Second Core Tile)	Yes	0x70000000-0x7FFFFFFF	AXI	0 to 1GB	-	-

a. The Interrupt signal column lists the value to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.
b. The Cortex-A9 and Cortex-A5 models implement an internal GIC. This is mapped at 0x1F000000 - 0x1F001FFF. For the EB model that uses the Cortex-A9 MPCore or Cortex-A5 MPCore, a DIC is used as the interrupt controller instead of the GIC. The numbers in this column are the interrupt numbers used by the DCCI system.
c. See *Sound* on page 3-18.
d. The implementation of the PL180 is currently limited, so not all features are present.
e. See *Differences between the EB and Core Tile hardware and the models* on page 3-18.
f. The EB RTSM GICs are not the same as those implemented on the EB hardware as the register map is different. See *Generic Interrupt Controller* on page 3-22.
g. Although the EB hardware uses the PL093 static memory controller, the model implements PL350. These are functionally equivalent.
h. This peripheral is implemented in the IntelStrataFlashJ3 component in the EB RTSM.
i. This peripheral is implemented in the SC91C111 component in the EB RTSM.

Note

The EB 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 an RTSM might fail on actual hardware.

3.2 EB model configuration parameters

The EB Real-Time System Models have parameters that can be defined at instantiation or 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. Unless otherwise described in the model release notes, these additional parameters are not intended to be changed by the end user.

For RealView Debugger, the parameters are listed under the component names in the configuration dialog. See *Using a configuration GUI in RealView Debugger* on page 2-7.

If you are using text configuration, details of the parameter syntax to use are given with the parameter lists in this section.

For more information on the EB hardware, see *Emulation Baseboard User Guide (Lead Free)*.

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

- *EB RTSM baseboard parameters* on page 3-6
- *Ethernet parameters* on page 3-8
- *UART parameters* on page 3-9
- *Terminal parameters* on page 3-9
- *Visualization parameters* on page 3-10
- *RTSM_EB_Cortex-A9_MPx1 and RTSM_EB_Cortex-A9_MPx2 core tile parameters* on page 3-11
- *RTSM_EB_Cortex-A8 core tile parameters* on page 3-12
- *RTSM_EB_Cortex-A5_MPx1 and RTSM_EB_Cortex-A5_MPx2 core tile parameters* on page 3-13
- *RTSM_EB_Cortex-R4 core tile parameters* on page 3-14
- *RTSM_EB_ARM1176 core tile parameters* on page 3-15
- *RTSM_EB_ARM1136 core tile parameters* on page 3-16
- *RTSM_EB_ARM926 core tile parameters* on page 3-17.

3.2.1 EB RTSM baseboard parameters

Table 3-2 lists the baseboard instantiation time parameters that you can change when you start the model.

The syntax to use in a configuration file is:

`baseboard.`*`component_name.parameter=value`*

Table 3-2 EB Baseboard Model instantiation parameters

Component name	Parameter	Description	Type	Values	Default
eb_sysregs_0	`user_switches_value`	switch S6 setting	integer	see *Switch S6*	0
eb_sysregs_0	`boot_switch_value`	switch S8 setting	integer	see *Switch S8* on page 3-7	0
flashldr_0	`fname`	path to flash image file	string	valid path	""
	`fnameWrite`	name of flash image file	string	valid filename	""
flashldr_1	`fname`	path to flash image file	string	valid filename	""
	`fnameWrite`	name of flash image file	string	valid filename	""
mmc	`p_mmc_file`	multimedia card filename	string	valid filename	`mmc.dat`
pl111_clcd_0	`pixel_double_limit`	sets threshold in horizontal pixels below which pixels sent to framebuffer are doubled in size in both dimensions	integer	-	`0x12c`
sdram_size	`sdram_size`	sets the size of the SDRAM on the second installed Core Tile	integer	0 to 0x40000000	0
sp805_watchdog_0	`simhalt`	enables or disables the ARM Watchdog Module (SP805)	boolean	`true` or `false`	false
sp810_sysctrl	`use_s8`	indicates whether to read `boot_switches_value`	boolean	`true` or `false`	false
vfs2	`mount`	name of mount directory	string	valid filename	""

Switch S6

Switch S6 is equivalent to the Boot Monitor configuration switch on the EB hardware. Default settings are listed in Table 3-3 on page 3-7.

If you have the standard ARM Boot Monitor flash image loaded, the setting of switch S6-1 changes what happens on model reset. Otherwise, the function of switch S6 is implementation dependent.

To write the switch position directly to the S6 parameter in the model, you must convert the switch settings to an integer value from the equivalent binary, where 1 is on and 0 is off.

Table 3-3 Default positions for EB System Model switch S6

Switch	Default Position	Function in default position
S6-1	OFF	Displays prompt allowing Boot Monitor command entry after system start.
S6-2	OFF	See Table 3-4.
S6-3	OFF	See Table 3-4.
S6-4 to S6-8	OFF	Reserved for application use.

If S6-1 is in the ON position, the Boot Monitor executes the boot script that was loaded into flash. If there is no script, the Boot Monitor prompt is displayed.

The settings of S6-2 and S6-3 affect STDIO source and destination on model reset as defined in Table 3-4.

Table 3-4 STDIO redirection

S6-2	S6-3	Output	Input	Description
OFF	OFF	UART0	UART0	STDIO autodetects whether to use semihosting I/O or a UART. If a debugger is connected, STDIO is redirected to the debugger output window, otherwise STDIO goes to UART0.
OFF	ON	UART0	UART0	STDIO is redirected to UART0, regardless of semihosting settings.
ON	OFF	CLCD	Keyboard	STDIO is redirected to the CLCD and keyboard, regardless of semihosting settings.
ON	ON	CLCD	UART0	STDIO output is redirected to the LCD and input is redirected to the keyboard, regardless of semihosting settings.

For more information on Boot Monitor configuration and commands, see the *Emulation Baseboard User Guide (Lead Free)*.

Switch S8

Switch S8 is disabled by default. To enable it, you must change the state of the parameter `baseboard.sp810_sysctrl.use_s8` to `true` before you start the model. See *EB RTSM baseboard parameters* on page 3-6.

If you have a Boot Monitor flash image loaded, switch S8 enables you to remap boot memory.

On reset, the EB hardware starts to execute code at `0x0`, which is typically volatile DRAM. You can put the contents of non-volatile RAM at this location by setting the S8 switch in the EB RTSM CLCD as shown in Table 3-5. The settings take effect on model reset.

Table 3-5 EB System Model switch S8 settings

Switch S8[4:1]	Memory Range	Description
0000	`0x40000000–0x4FEFFFFF`	NOR flash remapped to `0x0`
0001	`0x44000000–0x47FFFFFF`	NOR flash remapped to `0x0`
0010	`0x48000000–0x4BFFFFFF`	SRAM remapped to `0x0`

Note

Attempting to change switch S8 settings after the model has started, for example by using the CLCD DIP switches, might lead to unpredictable behavior.

3.2.2 Ethernet parameters

Table 3-6 lists the Ethernet instantiation-time parameters that you can change when you start the model. For detailed information on how to set up and use the Ethernet component, see *Using Ethernet with an EB RTSM* on page 2-19.

The syntax to use in a configuration file is:

`baseboard.smsc_91c111_0.`*`parameter`*`=`*`value`*

where *`parameter`* is the Ethernet parameter you are defining.

Table 3-6 Ethernet instantiation parameters

Parameter	Description	Type	Values	Default
`interface`	sets the host interface type to use	string	see *interface*	"`disabled`"
`mac_address`	the MAC address to use on the host	string	see *mac_address* on page 3-9	"`00:01:02:03:04:05`"
`promiscuous`	puts host Ethernet controller into promiscuous mode	boolean	`true` or `false`	`true`

interface

The interface parameter has a transport and one or more optional parameters separated by colons. It can take the form:

`disabled` Implement the interface as though no cable were connected.

`host:`*`controller`*

controller is the first host controller that matches the text substring you specify, such as **eth0**.

`pipe:`*`address`*`:`*`port`*

address and *port* are the IP address and port on which a `nicserver` application is listening.

mac_address

There are two options for setting the `mac_address` parameter:

- For a fixed address, set the value of the `mac_address` parameter to a valid IP address for your network. Fixed addresses do not change when the EB RTSM is reset.
- For a random value, set the mac_address parameter to **auto**.

Note

The address might change each time the EB RTSM is reset. Because of the risk of address duplication, this is not recommended if your network normally allocates addresses automatically using a DHCP server.

3.2.3 UART parameters

Table 3-7 lists the PL011 UART instantiation-time parameters that you can change when you start the model. The syntax to use in a configuration file is:

`baseboard.uart_`*`x`*`.`*`parameter`*`=`*`value`*

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

Table 3-7 UART instantiation parameters

Component name	Parameter	Description	Type	Values	Default
uart_[0-3]	`untimed_fifo`	ignore the clock rate and receive serial data immediately	boolean	`0` or `1`	`0`
uart_[0-3]	`unbuffered_output`	do not buffer the output	boolean	`0` or `1`	`0`

3.2.4 Terminal parameters

When the EB 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 2-26.

Table 3-8 on page 3-10 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 3-8 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.

3.2.5 Visualization parameters

Table 3-9 lists the Visualization instantiation-time parameters that you can change when you start the model. Detailed information on the Visualisation component is given in a separate document. See *Fast Models Reference Manual*. The syntax to use in a configuration file is:

`visualisation.`*`parameter`*`=`*`value`*.

Note

The component name spelling is British, so use *visualisation* rather than *visualization*.

Table 3-9 Visualisation instantiation parameters

Parameter	Description	Type	Values	Default
disable_visualisation	disable the EBVisualisation component on model startup	boolean	true or false	false
rate_limit-enable[a]	restrict simulation speed so that simulation time more closely matches real time rather than running as fast as possible	boolean	true or false	true
trap_key	trap key that works with **Left Ctrl** key to toggle mouse pointer display	integer	valid ATKeyCode key value[b]	74[c]

a. You can click the Rate Limit button in the CLCD instead of setting the parameter at instantiation time. See *Using the CLCD window* on page 2-14.
b. If you have Fast Models installed, see the header file, %PVLIB_HOME%\components\KeyCode.h, for a list of ATKeyCode values. On Linux, see the file $PVLIB_HOME/components/KeyCode.h.
c. This is equivalent to the **Left Alt** key.

3.2.6 RTSM_EB_Cortex-A9_MPx1 and RTSM_EB_Cortex-A9_MPx2 core tile parameters

This section describes the Cortex-A9 MPCore parameters that you can change when you start the RTSM_EB_Cortex-A9_MPx1 or RTSM_EB_Cortex-A9_MPx2 models. This core tile RTSM is based on r1p0 of the Cortex-A9 MPCore processor.

Table 3-10 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:

`coretile.core.cpu`*`n`*`.`*`parameter`*`=`*`value`*

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

Table 3-10 RTSM_EB_Cortex-A9_MPx*n* core tile 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`
`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. Currently ignored.

3.2.7 RTSM_EB_Cortex-A8 core tile parameters

Table 3-11 lists the Cortex-A8 core tile RTSM parameters that you can change when you start the model. All listed parameters are instantiation-time parameters. This core tile RTSM is based on r2p1 of the Cortex-A8 processor.

The syntax to use in a configuration file is:

`coretile.core.`*`parameter`*`=`*`value`*

Table 3-11 RTSM_EB_Cortex-A8 core tile parameters

Parameter	Description	Type	Values	Default
`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	24 bit integer	`0x123456`
`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`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	integer	`0x00000000 - 0xFFFFFFFF`	`0x0F0000000`

a. Currently ignored.

The Cortex-A8 core tile RTSM also includes a GIC but this cannot be configured at instantiation time.

3.2.8 RTSM_EB_Cortex-A5_MPx1 and RTSM_EB_Cortex-A5_MPx2 core tile parameters

This section describes the Cortex-A5 core tile parameters that you can change when you start the RTSM_EB_Cortex-A5_MPx1 or RTSM_EB_Cortex-A5_MPx2 models. This core tile RTSM is based on r0p0 of the Cortex-A5 processor.

The syntax to use in a configuration file is:

`coretile.core.cpu`*`n`*`.`*`parameter`*`=`*`value`*

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

Table 3-12 RTSM_EB_Cortex-A5_MPx*n* core tile parameters for the individual cores

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`[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`
`semihosting-stack_limit`	Virtual address of stack limit.	integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`

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

The RTSM_EB_Cortex-A5_MPx*n* models contain a specific *Distributed Interrupt Controller* (DIC), rather than the *Generic Interrupt Controller* (GIC) used in several other EB RTSMs. The syntax to use in a configuration file is:

`coretile.eb_intmapper.num_interrupts=`*`value`*

where *value* is the number of interrupts, from 0 to 224 inclusive. The default number is 64.

3.2.9 RTSM_EB_Cortex-R4 core tile parameters

Table 3-13 lists the Cortex-R4 core tile parameters that you can change when you start the RTSM model. All listed parameters are instantiation-time parameters. This core tile model is based on r1p2 of the Cortex-R4 processor.

The syntax to use in a configuration file is:

`coretile.core.`*`parameter`*`=`*`value`*

Table 3-13 RTSM_EB_Cortex-R4 core tile parameters

Parameter	Description	Type	Values	Default
`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`	`true` or `false`
`semihosting-enable`	Enable semihosting SVC traps.	boolean	`true` or `false`	true
`semihosting-ARM_SVC`	ARM SVC number for semihosting.	integer	24 bit integer	`0x123456`
`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`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	integer	`0x00000000` - `0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	integer	`0x00000000` - `0xFFFFFFFF`	`0x0F0000000`

a. Currently ignored.

The Cortex-R4 core tile RTSM also includes a GIC but this cannot be configured at instantiation time.

3.2.10 RTSM_EB_ARM1176 core tile parameters

Table 3-14 lists the ARM1176JZ-S core tile RTSM parameters that you can change when you start the model. All listed parameters are instantiation-time parameters except for `num_interrupts`, which is a run time parameter. This Core Tile RTSM is based on r0p4 of the ARM1176JZ-S processor.

The syntax to use in a configuration file is:

`coretile.core.`*`parameter`*`=`*`value`*

Table 3-14 RTSM_EB_ARM1176 core tile parameters

Parameter	Description	Type	Values	Default
`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	24 bit integer	`0x123456`
`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`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	integer	`0x00000000` - `0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	integer	`0x00000000` - `0xFFFFFFFF`	`0x0F0000000`

a. Currently ignored.

The ARM1176 core tile RTSM also includes a GIC but this cannot be configured at instantiation time.

3.2.11 RTSM_EB_ARM1136 core tile parameters

Table 3-15 lists the ARM1136JF-S core tile RTSM parameters that you can change when you start the model. All listed parameters are instantiation-time parameters except for num_interrupts, which is a run time parameter. This core tile RTSM is based on r1p1 of the ARM1136JF-S processor.

The syntax to use in a configuration file is:

`coretile.core.`*`parameter`*`=`*`value`*

Table 3-15 RTSM_EB_ARM1136 core tile parameters

Parameter	Description	Type	Values	Default
`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	24 bit integer	`0x123456`
`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`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	integer	`0x00000000` - `0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	integer	`0x00000000` - `0xFFFFFFFF`	`0x0F0000000`

a. Currently ignored.

The ARM1136 core tile RTSM also includes a GIC but this cannot be configured at instantiation time.

3.2.12 RTSM_EB_ARM926 core tile parameters

Table 3-16 lists the ARM926EJ-S core tile RTSM parameters that you can change when you start the model. All listed parameters are instantiation-time parameters except for num_interrupts, which is a run time parameter. This core tile RTSM is based on r0p5 of the ARM926EJ-S processor.

The syntax to use in a configuration file is:

`coretile.core.`*`parameter`*`=`*`value`*

Table 3-16 RTSM_EB_ARM926 core tile parameters

Parameter	Description	Type	Values	Default
`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	24 bit integer	`0x123456`
`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`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	integer	`0x00000000` - `0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	integer	`0x00000000` - `0xFFFFFFFF`	`0x0F0000000`

a. Currently ignored.

3.3 Differences between the EB and Core Tile hardware and the models

The following sections describe features of the Emulation Baseboard and Core Tile hardware that are not implemented in the models or have significant differences in implementation:

- *Features not present in the baseboard model*
- *Restrictions on the processor models* on page 3-19
- *Remapping and DRAM aliasing* on page 3-22
- *Dynamic memory characteristics* on page 3-22
- *Status and system control registers* on page 3-22
- *Generic Interrupt Controller* on page 3-22
- *GPIO2* on page 3-23
- *Timing considerations* on page 3-23.

3.3.1 Features not present in the baseboard model

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

- two wire serial bus interface
- character LCD interface
- smart card interface
- PCI controller configuration registers
- debug access port
- disk on chip
- configuration flash
- USB
- PISMO expansion memory
- PCI interface bus windows
- UART Modem handshake signals
- VGA support.

Note

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

The following features present on the hardware version of the Emulation Baseboard are only partially implemented in the Real-Time System Models:

- *Sound*
- *Dynamic memory controller* on page 3-19.

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

Sound

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

Dynamic memory controller

The dynamic memory controller, though modeled in the EB RTSMs, does not provide direct memory access to all peripherals. Only the audio and synchronous serial port interface components can be accessed through the DMC.

3.3.2 Restrictions on the processor models

Detailed information concerning what features are not fully implemented in the processor models included with the EB RTSMs can be found in separate documentation. See the *Fast Models Reference Manual*. 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 and Wait For Event.
- Level 1 and level 2 caches are not part of the models for ARM Architecture v5 and Architecture v6 processors. Although cache control registers are included, in most cases they only enable you to check register access permissions. Cache flush operations are supported, but they have no effect. As a consequence, code that might fail on real hardware due to cache aliasing problems might run without problems on the EB RTSM.
- VFP and NEON instruction set execution on the model is not currently high performance.
- Write buffers are not modeled.
- Most aspects of TLB behavior are implemented in the models. However TLB memory attribute settings are ignored in Architecture v5 and v6 models, as they relate to cache and write buffer behavior. In Architecture v7 models, the TLB memory attribute settings are used when stateful cache is enabled.
- No MicroTLB is implemented.
- The ARMv6 architecture deprecates MMU sub-page permissions. These deprecated features are not supported by the simulator.
- 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 not 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.
- Only one CP14 debug coprocessor register is included, CP14 DSCR. The register reads 0 and ignores writes. Access to other CP14 registers causes an undefined instruction exception. To debug an 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.
- The *Performance Management Unit* (PMU) is not implemented except for the instruction counter.

RTSM_EB_Cortex-A9_MPx1 and RTSM_EB_Cortex-A9_MPx2 core tiles

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

- The Cortex-A9MPCore 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.

RTSM_EB_Cortex-A8 core tile

The following additional restrictions apply to the Real-Time System Model implementation of the Cortex-A8 processor:

- 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 PLE model is purely register-based and has no implemented behavior.
- VFP and NEON instruction set execution on the model is not currently high performance.
- Unaligned accesses with the MMU disabled do not cause data aborts.

RTSM_EB_Cortex-A5 and Cortex-A5_MPx2 core tiles

The following additional restrictions apply to the Real-Time System Model implementation of Cortex-A5 and Cortex-A5 MPCore processors:

- The model implements L1 cache as architecturally defined.
- 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.
- VFP and NEON instruction set execution on the model is not currently high performance.

RTSM_EB_Cortex-R4 core tile

The following additional restrictions apply to the Real-Time System Model implementation of an Cortex-R4 processor:

- The model uses a programmer's view-accurate implementation of cache.
- A single flat 4GB address region is seen by the model core. Some of this space is mapped to zero wait state memory or peripherals.
- VFP instruction set execution, and instructions in protection regions smaller than 1KB, are not currently high performance.
- ECC and parity schemes are not supported (though the registers might be present)

RTSM_EB_ARM1176 core tile

The following additional restrictions apply to the Real-Time System Model implementation of an ARM1176JZF processor:

- 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.
- VFP instruction set execution on the model is not currently high performance.
- A simplified VIC is implemented. The interaction between the CPU and VIC is untimed after the interrupt is acknowledged.

RTSM_EB_ARM1136 core tile

The following additional restrictions apply to the Real-Time System Model implementation of an ARM1136JF-S processor:

- A single flat 4GB address region is seen by the model core. Some of this space is mapped to zero wait state memory or peripherals.
- VFP instruction set execution on the model is not currently high performance.
- A simplified VIC is implemented. The interaction between the CPU and VIC is untimed once the interrupt is acknowledged.
- System coprocessor registers pertaining to the TLB or MicroTLB read 0 and ignore writes.

RTSM_EB_ARM926 core tile

The following additional restrictions apply to the Real-Time System Model implementation of an ARM926JF-S processor:

- A single flat 4GB address region is seen by the model core. Some of this space is mapped to zero wait state memory or peripherals.

3.3.3 Remapping and DRAM aliasing

The EB hardware provides considerable memory remap functionality. During this boot remapping, the bottom 64MB of the physical address map can be:

- NOR flash
- Static expansion memory.

As well as providing remap functionality, the hardware aliases all 256MB of system DRAM at `0x70000000`.

Remapping does not typically apply to the system models. However, NOR flash is modelled and can be remapped. See *Switch S8* on page 3-7.

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

3.3.4 Dynamic memory characteristics

The Emulation Baseboard hardware contains a PL340 DMC. This presents a configuration interface at address `0x10030000` in the memory map.

The system models configure a generic area of DRAM and do not model the PL340. This simplification helps speed the simulation.

3.3.5 Status and system control registers

For the hardware version of the Emulation Baseboard, the status and system control registers enable the processor to determine its environment and to control some on-board operations.

Note

Most of the EB RTSM functionality is determined by its configuration on startup. See *Configuring the RTSM* on page 2-7.

All EB system registers have been implemented in the system model, except for SYS_TEST_OSC[4:0], the oscillator test registers. Registers that are not implemented function as memory and the values written to them do not alter the behavior of the model.

3.3.6 Generic Interrupt Controller

The *Generic Interrupt Controller* (GIC) provided with the EB RTSMs differs substantially from that in the current Emulation Board firmware. The programmer's model of the newer device is largely backwards compatible. The model GIC is an implementation of the PL390 PrimeCell, for which comprehensive documentation is provided elsewhere. See *PrimeCell® Generic Interrupt Controller (PL390) Technical Reference Manual*.

3.3.7 GPIO2

On the EB hardware, GPIO2 is dedicated to USB, a push button, and MCI status signals. USB and MCI are not implemented in the EB RTSMs, and no push button is modelled. The GPIO is therefore simply provided as another generic IO device.

3.3.8 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.

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

The EB 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:

- *MPS model memory map* on page 4-2
- *MPS configuration parameters* on page 4-6.
- *Differences between the MPS hardware and the system model* on page 4-9.

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 allow the interrupt vector table to be modified. It is also used to allow 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 allow the Flash memory to be reprogrammed at this offset. It also allows for 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/false	true
`disable_visualisation`	enable/disable visualization	boolean	true/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/false	false
`mps_dut.uart[0|1|2].untimed_fifos`	Operate UART FIFO in fast (no timing) mode	boolean	true/false	false
`mps_dut.uart[0|1|2].unbuffered_output`	Unbuffered output	boolean		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 2-26.

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/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-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/false	false
coretile.uart3.unbuffered_output	Unbuffered output	boolean	true/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 Microcontroller Prototyping System hardware that are not implemented in the models or have significant differences in implementation:

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

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 Microcontroller Prototyping System 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 Microcontroller Prototyping System 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 may 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.

Glossary

This glossary lists abbreviations used in this document or the *Emulation Baseboard User Guide*.

AACI *Advanced Audio CODEC Interface*.

AHB™ *Advanced High-performance Bus*. The ARM open standard for on-chip buses.

AMBA® *Advanced Microcontroller Bus Architecture*.

APB *Advanced Peripheral Bus*. The ARM open standard for peripheral buses. This design is optimized for low power and minimal interface complexity.

AXI™ *Advanced eXtensible Interface*. The ARM open standard for high-performance and high-frequency buses.

CADI *Cycle Accurate Debug Interface*. A C++ API supporting interface capabilities for re-targetable, multi-core debug integration.

CLCD *Color Liquid-Crystal Display*.

CODEC *COder DECoder* for converting between analog and digital audio signals.

CXSM *Cycle approXimate System Model*.

DOC *Disk-On-Chip*. A non-volatile flash memory device with an interface that simplifies file accesses. Also called NAND flash referring to the logic gates used internally. The memory can only be accessed sequentially in blocks.

DMC *Dynamic Memory Controller*.

DMA *Direct Memory Access*.

DRAM *Dynamic Random Access Memory*.

DSR	*Data Set Ready*, a UART flow-control signal.
DTR	*Data Terminal Ready*, a UART flow-control signal.
EB	*RealView Emulation Baseboard*. A hardware development platform that supports various Core Tiles and FPGA tiles.
GIC	*Generic Interrupt Controller*.
GPIO	*General Purpose Input/Output*.
Integrator®/CP	Integrator *Compact Platform*.
I/O	*Input/Output*.
KMI	*Keyboard/Mouse Interface*.
LCD	*Liquid Crystal Display*.
LED	*Light Emitting Diode*.
MAC	*Media Access Control*. A layer in the Ethernet specification.
MCI	*MultiMedia Card Interface*.
MMC	*MultiMedia Card*.
NAND flash	Non-volatile memory. *NAND* refers to the type of logic gate used internally. See *DOC*.
NOR flash	Non-volatile memory. *NOR* refers to the type of logic gate used internally. Any memory address can be accessed randomly.
PCI	*Peripheral Component Interconnect*. A computer bus for attaching peripherals.
PHY	*PHYsical* layer. The layer in the Ethernet specification that describes the physical interface.
PISMO	*Platform Independent Storage Module*. Memory specification for plug in memory modules.
PLL	*Phase-Locked Loop*, a type of programmable oscillator.
RAM	*Random Access Memory*.
RTC	*Real-Time Clock*.
RTSM	*Real-Time System Model*.
SRAM	*Static Random Access Memory*.
SCI	*Smart Card Interface*.
SD	*Secure Digital* memory card specification.
SMC	*Static Memory Controller*.
SSP	*Synchronous Serial Port*.
TCM	*Tightly Coupled Memory*. Memory present inside the test chip that typically runs at or near the processor speed.
UART	*Universal Asynchronous Receiver/Transmitter*.
USB	*Universal Serial Bus*.
VGA	*Video Graphics Array*.