Proprietary Notice

Words and logos marked with or are registered trademarks or trademarks of ARM in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com
## Contents

**Fixed Virtual Platforms VE and MPS FVP Reference Guide**

### Chapter 1
Conventions and feedback

### Chapter 2
Introduction
- 2.1 About system models ................................................................. 2-2
- 2.2 About the VE FVP ........................................................................ 2-3
- 2.3 About the MPS FVP ..................................................................... 2-6
- 2.4 MPS hardware ............................................................................. 2-7
- 2.5 MPS FVP ..................................................................................... 2-8

### Chapter 3
Getting Started with Fixed Virtual Platforms
- 3.1 Debugging a FVP ........................................................................ 3-2
- 3.2 Starting the FVP using Model Shell ............................................. 3-3
- 3.3 Configuring VE and MPS FVPs .................................................... 3-5
- 3.4 Loading and running an application on the VE FVP ..................... 3-7
- 3.5 Using the VE CLCD window ........................................................ 3-8
- 3.6 Using the MPS Visualization window ......................................... 3-12
- 3.7 Using Ethernet with a VE FVP ..................................................... 3-15
- 3.8 Using a terminal with a system model ........................................ 3-17
- 3.9 Virtual filesystem ....................................................................... 3-19
- 3.10 Using the VFS with a prebuilt FVP ............................................ 3-21

### Chapter 4
Programmer’s Reference for the VE FVPs
- 4.1 VE model memory map .............................................................. 4-2
- 4.2 VE model parameters .................................................................. 4-5
- 4.3 Motherboard peripheral parameters ............................................ 4-6
- 4.4 Color LCD controller parameters ................................................ 4-7
Chapter 5  Programmer’s Reference for the MPS FVPs

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>MPS model memory map</td>
<td>5-2</td>
</tr>
<tr>
<td>5.2</td>
<td>MPS registers</td>
<td>5-4</td>
</tr>
<tr>
<td>5.3</td>
<td>Processor system registers</td>
<td>5-5</td>
</tr>
<tr>
<td>5.4</td>
<td>DUT system registers</td>
<td>5-6</td>
</tr>
<tr>
<td>5.5</td>
<td>Character LCD registers</td>
<td>5-7</td>
</tr>
<tr>
<td>5.6</td>
<td>Memory configuration and remap</td>
<td>5-8</td>
</tr>
<tr>
<td>5.7</td>
<td>Switches</td>
<td>5-9</td>
</tr>
<tr>
<td>5.8</td>
<td>Seven-segment display</td>
<td>5-10</td>
</tr>
<tr>
<td>5.9</td>
<td>MPS parameters</td>
<td>5-11</td>
</tr>
<tr>
<td>5.10</td>
<td>MPS visualization parameters</td>
<td>5-12</td>
</tr>
<tr>
<td>5.11</td>
<td>DUT parameters</td>
<td>5-13</td>
</tr>
<tr>
<td>5.12</td>
<td>Terminal parameters</td>
<td>5-14</td>
</tr>
<tr>
<td>5.13</td>
<td>Processor parameters</td>
<td>5-15</td>
</tr>
<tr>
<td>5.14</td>
<td>Differences between the MPS hardware and the system model</td>
<td>5-16</td>
</tr>
<tr>
<td>5.15</td>
<td>Features not present in the model</td>
<td>5-17</td>
</tr>
<tr>
<td>5.16</td>
<td>Timing considerations</td>
<td>5-18</td>
</tr>
</tbody>
</table>
Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

**Typographical conventions**

The following typographical conventions are used:

- **monospace** Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
- **monospace** Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
- **monospace italic** Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
- **monospace bold** Denotes language keywords when used outside example code.
- **italic** Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
- **bold** Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

**Feedback on this product**

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

- your name and company
• the serial number of the product
• details of the release you are using
• details of the platform you are using, such as the hardware platform, operating system type and version
• a small standalone sample of code that reproduces the problem
• a clear explanation of what you expected to happen, and what actually happened
• the commands you used, including any command-line options
• sample output illustrating the problem
• the version string of the tools, including the version number and build numbers.

Feedback on content

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

• the title
• the number, ARM DUI 0575G
• if viewing online, the topic names to which your comments apply
• if viewing a PDF version of a document, the page numbers to which your comments apply
• a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and Frequently Asked Questions (FAQs).

Other information

• ARM Information Center, http://infocenter.arm.com/help/index.jsp
• Support and Maintenance, http://www.arm.com/support/services/support-maintenance.php
Chapter 2
Introduction

The following topics introduce the Fixed Virtual Platforms (FVPs):

Concepts

• About system models on page 2-2
• About the VE FVP on page 2-3
• About the MPS FVP on page 2-6
• MPS hardware on page 2-7
• MPS FVP on page 2-8.
2.1 About system models

*Fixed Virtual Platforms* (FVPs) enable development of software without the requirement for actual hardware. The software models provide *Programmer’s View* (PV) models of processors and devices. The functional behavior of a model is equivalent to real hardware.

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

System models are supplied as a *Component Architecture Debug Interface* (CADI) shared library, and are loaded by any environment compatible with the CADI API. Such environments include:

- Model Debugger
- Model Shell.

2.1.1 See also

**Concepts**

- *About the VE FVP* on page 2-3
- *About the MPS FVP* on page 2-6
- *MPS hardware* on page 2-7
- *MPS FVP* on page 2-8.

**Reference**

2.2 About the VE FVP

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

The motherboard provides the following features:

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

The VE FVP is a system model implemented in software. The model contains:

- Virtual implementations of a motherboard.
- A single daughterboard containing a specific ARM processor.
- Associated interconnections.

Note

The model is based on the VE platform memory map, but is not intended to be an accurate representation of a specific VE hardware revision. The VE FVP supports selected peripherals. For more information about these peripherals, see the reference information at the end of this topic. The supplied model is sufficiently complete and accurate to boot the same operating system images as for the VE hardware.

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

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

- Timing is approximate.
- Buses are simplified.
- Caches for the processors and the related write buffers are not implemented.
The VE FVPs provided in this release are:

- FVP_VE_Cortex-A15x1, FVP_VE_Cortex-A15x2, FVP_VE_Cortex-A15x4
- FVP_VE_Cortex-A7x1, FVP_VE_Cortex-A7x2, FVP_VE_Cortex-A7x4
- FVP_VE_Cortex-A12x1, FVP_VE_Cortex-A12x2, FVP_VE_Cortex-A12x4
- FVP_VE_Cortex-A15x1-A7x1, FVP_VE_Cortex-A15x4-A7x4, FVP_VE_Cortex-A15x2-A7x2, FVP_VE_Cortex-A15x2-A7x3, FVP_VE_Cortex-A15x2-A7x4
- FVP_VE_AEMv8A
- FVP_VE_Cortex-A57x4
- FVP_VE_Cortex-A53x4
- Base-A57x1,x2,x4
- Base-A53x1,x2,x4
- Base-A57x1-A53x1, x4-x4

The following figure shows a block diagram of a top-level VE model with a Cortex-A15 cluster:

![Block diagram of top-level VE model](image)

**Figure 2-1 Block diagram of top-level VE model**

### 2.2.1 See also

**Concepts**

- *About system models on page 2-2.*
- *About the MPS FVP on page 2-6.*
- *MPS hardware on page 2-7.*
- *MPS FVP on page 2-8.*
- *VE model parameters on page 4-5.*
- *Differences between the VE and CoreTile hardware and the models on page 4-40.*
Reference

- Chapter 4 *Programmer's Reference for the VE FVPs*.
- *FVP_VE_Cortex-A15xn CoreTile parameters* on page 4-20.
- *ARMv8-A AEM parameters* on page 4-23.
- *Motherboard Express µATX V2M-P1 Technical Reference Manual*,
2.3 About the MPS FVP

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

The *Microcontroller Prototyping System Fixed Virtual Platforms* (MPS FVPs) are system models implemented in software. They are developed using the ARM Fast Models library product.

Note

The MPS FVPs are provided as example platform implementations, and are not intended to be accurate representations of a specific hardware revision. The MPS FVP supports selected peripherals. For more information about these peripherals, see the reference information at the end of this topic. The supplied FVPs are sufficiently complete and accurate to boot the same application images as the MPS hardware.

2.3.1 See also

Concepts

- *About system models on page 2-2*
- *About the VE FVP on page 2-3*
- *MPS hardware on page 2-7*
- *MPS FVP on page 2-8*
- *VE model parameters on page 4-5*
- *Differences between the VE and CoreTile hardware and the models on page 4-40.*

Reference

- *Chapter 5 Programmer’s Reference for the MPS FVPs.*
2.4 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 FVPs provide a functionally-accurate model for software execution. However, the model sacrifices timing accuracy to increase simulation speed. Key deviations from actual hardware are:

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

2.4.1 See also

**Concepts**

- About system models on page 2-2
- About the VE FVP on page 2-3
- About the MPS FVP on page 2-6
- MPS FVP on page 2-8
- VE model parameters on page 4-5
- Differences between the VE and CoreTile hardware and the models on page 4-40.
2.5 MPS FVP

The MPS FVP models in software some of the functionality of the MPS hardware.

A complete model implementation of the MPS platform includes both MPS-specific components and generic ones such as buses and timers. The following figure shows a block diagram of a MPS FVP:

![Figure 2-2 MPS FVP block diagram](image)

2.5.1 See also

Concepts
- About system models on page 2-2
- About the VE FVP on page 2-3
- About the MPS FVP on page 2-6
- MPS hardware on page 2-7
- VE model parameters on page 4-5
- Differences between the VE and CoreTile hardware and the models on page 4-40.
Chapter 3
Getting Started with Fixed Virtual Platforms

The following topics describe the procedures for starting and configuring FVPs, and running a software application on the model. The procedures differ, depending on the ARM software tools that you are using.

Tasks
• Starting the FVP using Model Shell on page 3-3.

Reference
• Debugging a FVP on page 3-2
• Configuring VE and MPS FVPs on page 3-5
• Loading and running an application on the VE FVP on page 3-7
• Using the VE CLCD window on page 3-8
• Using the MPS Visualization window on page 3-12
• Using Ethernet with a VE FVP on page 3-15
• Using a terminal with a system model on page 3-17.
• Virtual filesystem on page 3-19
• Using the VFS with a prebuilt FVP on page 3-21.
3.1 Debugging a FVP

To debug a FVP, you can either:

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

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

3.1.1 Semihosting support

Semi-hosting enables code running on a platform model to directly access the I/O facilities on a host computer. Examples of these facilities include console I/O and file I/O. You can find more information on semihosting in the *ARM Compiler toolchain Developing Software for ARM Processors*.

The simulator handles semihosting by intercepting HLT 0xF000, or SVC 0x123456 or 0x_AB, depending on whether the processor is in A64, A32 or T32 state. All other HLTs and SVCs are handled as normal.

If the operating system does not use HLT 0xF000, SVC 0x123456 or 0x_AB for its own purposes, it is not necessary to disable semihosting support to boot an operating system.

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

3.1.2 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Configuring VE and MPS FVPs* on page 3-5
- *Using a configuration file* on page 3-5
- *Using the command line* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
- *ARM Compiler toolchain Developing Software for ARM Processors*,
### 3.2 Starting the FVP using Model Shell

You can use the Model Shell application to start VE and MPS FVPs. You can start the FVP with its own CADI debug server, enabling the model to run independently of a debugger. However, it means that you must configure your model using arguments that are passed to the model at start time.

To start the FVP using Model Shell:

1. Change to the directory where your model file is located.

2. Enter the following at the command prompt:
   ```
   model_shell --cadi-server --model model_name [--config-file filename] [--parameter instance.parameter=value] [--application app_filename]
   ```
   
   where:
   - `model_name` is the name of the model file. By default this file name is typically `FVP_VE_processor.dll` or `FVP_MPS_processor.dll` on Microsoft Windows or `FVP_VE_processor.so` or `FVP_MPS_processor.so` on Linux.
   - `filename` is the name of your optional plain-text configuration file. Configuration files simplify managing multiple parameters.
   - `instance.parameter=value` is the optional direct setting of a configuration parameter.
   - `app_filename` is the file name of an image to load to your model at startup.

The following example shows the format for using Model Shell to load and run an image from an ELF file:

```
Example 3-1 Load and run an image from an ELF file
```

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

**Note**

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

`install_directory\..\bin\model_shell`

You can use `*` to load the same image into all the processors in one multiprocessor cluster, for example:

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

**Note**

You must quote the argument as shown if you are using csh, or if you have spaces in the filename, otherwise the shell tries to expand the `*` instead of passing it to the application.
Starting the model opens the FVP CLCD display. After the FVP starts, you can use your debugger if it has a CADI interface to connect to the FVP.

### 3.2.1 See also

**Reference**

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Using a configuration file* on page 3-5
- *Using the command line* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
3.3 Configuring VE and MPS FVPs

You can configure VE and MPS FVPs either by using a configuration GUI in your debugger or by setting model configuration options from Model Shell.

3.3.1 Using a configuration GUI in your debugger

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

*Note*

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

3.3.2 Setting model configuration options from Model Shell

You can control the initial state of the FVP 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 when you are starting the FVP.

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`. You must enclose strings in double quotation marks if they contain whitespace.

The following example shows a typical configuration file:

**Example 3-2 Configuration file**

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

**Using the command line**

You can use the `-C` switch to define model parameters when you invoke the model. You can also use `--parameter` as a synonym for the `-C` switch. Use the same syntax as for a configuration file, but each parameter must be preceded by the `-C` switch.

The following example shows how to configure a MPS FVP using Model Shell:
Example 3-3 Using Model Shell to boot a model from a flash image

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

3.3.3 See also

Tasks
- Starting the FVP using Model Shell on page 3-3.

Reference
- Debugging a FVP on page 3-2
- Loading and running an application on the VE FVP on page 3-7
- Using the VE CLCD window on page 3-8
- Using the MPS Visualization window on page 3-12
- Using Ethernet with a VE FVP on page 3-15
- Using a terminal with a system model on page 3-17
- Virtual filesystem on page 3-19.
- VE model parameters on page 4-5
- MPS parameters on page 5-11.
### 3.4 Loading and running an application on the VE FVP

Example applications are provided for use with the FVPs for the VE system board.

**Note**

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

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

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

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

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

#### 3.4.1 See also

**Tasks**
- Starting the FVP using Model Shell on page 3-3.

**Reference**
- Debugging a FVP on page 3-2
- Configuring VE and MPS FVPs on page 3-5
- Using the VE CLCD window on page 3-8
- Using the MPS Visualization window on page 3-12
- Using Ethernet with a VE FVP on page 3-15
- Using a terminal with a system model on page 3-17
- Virtual filesystem on page 3-19.
3.5 Using the VE CLCD window

When an FVP starts, the FVP CLCD window opens, representing the contents of the simulated color LCD frame buffer. It automatically resizes to match the horizontal and vertical resolution set in the CLCD peripheral registers.

The following figure shows the VE FVP CLCD in its default state, immediately after being started.

![Figure 3-1 CLCD window at startup](image)

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

**USERSW**  
Eight white boxes show the state of the VE User DIP switches.  
These represent switch S6 on the VE hardware, USERSW[8:1], which is mapped to bits [7:0] of the SYS_SW register at address 0x10000004.  
The switches are in the off position by default. Click in the area above or below a white box to change its state.

**BOOTSW**  
Eight white boxes show the state of the VE Boot DIP switches.  
These represent switch S8 on the VE hardware, BOOTSEL[8:1], which is mapped to bits [15:8] of the SYS_SW register at address 0x100000004.  
The switches are in the off position by default.

*Note*  
ARM recommends that you configure the Boot DIP switches using the `boot_switch` model parameter instead of using the CLCD interface. Changing Boot DIP switch positions while the model is running can result in unpredictable behavior.

**S6LED**  
Eight colored boxes indicate the state of the VE User LEDs.  
These represent LEDs D[21:14] on the VE hardware, which are mapped to bits [7:0] of the SYS_LED register at address 0x10000008. The boxes correspond to the red/yellow/green LEDs on the VE hardware.

**Total Instr**  
A counter showing the total number of instructions executed.  
Because the FVP models provide a programmer’s view of the system, the CLCD displays total instructions rather than total processor cycles. Timing might differ substantially from the hardware because:

- the bus fabric is simplified
- memory latencies are minimized
- cycle approximate processor 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.

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. The figure below shows the CLCD with Rate Limit disabled.

--- Note ---
You can control whether Rate Limit is enabled by using the `rate_limit-enable` parameter, one of the visualization parameters for the MPS Visualization component, when instantiating the model.

When you click on the **Total Instr** or **Total Time** items in the CLCD, the display changes to show **Instr/sec** (instructions per second) and **Perf Index** (performance index). This is shown in the following figure:

![Figure 3-2 CLCD window with Rate Limit ON](image)

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

- **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.

The VE FVP CLCD displays the processor run state for each processor with a colored icon. The icons are to the left of the **Total Instr** (or **Inst/sec**) item, as shown for example in Figure 3-3:

![Figure 3-3 Processor run state icons for a quad processor model](image)
The description of each of the possible icons is shown in Table 3-1:

<table>
<thead>
<tr>
<th>Icon</th>
<th>State label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="question_mark" alt="Icon" /></td>
<td>UNKNOWN</td>
<td>Run status unknown, that is, simulation has not started.</td>
</tr>
<tr>
<td><img src="play_arrow" alt="Icon" /></td>
<td>RUNNING</td>
<td>Processor running, is not idle, and is executing instructions.</td>
</tr>
<tr>
<td><img src="pause" alt="Icon" /></td>
<td>HALTED</td>
<td>External halt signal asserted.</td>
</tr>
<tr>
<td><img src="arrow_upward" alt="Icon" /></td>
<td>STANDBY_WFE</td>
<td>Last instruction executed was WFE and standby mode has been entered.</td>
</tr>
<tr>
<td><img src="arrow_downward" alt="Icon" /></td>
<td>STANDBY_WFI</td>
<td>Last instruction executed was WFI and standby mode has been entered.</td>
</tr>
<tr>
<td><img src="exit_to_square" alt="Icon" /></td>
<td>IN_RESET</td>
<td>External reset signal asserted.</td>
</tr>
<tr>
<td><img src="power_settings" alt="Icon" /></td>
<td>DORMANT</td>
<td>Partial processor power down.</td>
</tr>
<tr>
<td><img src="power_settings" alt="Icon" /></td>
<td>SHUTDOWN</td>
<td>Complete processor power down.</td>
</tr>
</tbody>
</table>

--- Note ---
The icons do not appear until you start the simulation.

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. This is one of the visualization parameters for the MPS Visualization component.

### 3.5.1 See also

#### Tasks
- *Starting the FVP using Model Shell on page 3-3.*

#### Reference
- *Debugging a FVP on page 3-2*
• Configuring VE and MPS FVPs on page 3-5
• Using the MPS Visualization window on page 3-12
• Using Ethernet with a VE FVP on page 3-15
• Using a terminal with a system model on page 3-17
• Virtual filesystem on page 3-19
• Timing considerations on page 4-47
• Visualization parameters on page 4-19
• Fast Models Reference Manual,
3.6 Using the MPS Visualization window

When an MPS FVP starts, the FVP CLCD window opens, which displays a simulated color LCD frame buffer. It automatically resizes to match the horizontal and vertical resolution set in the CLCD peripheral registers.

![Visualization window at startup](image)

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 processor LEDs.

**DUT**
- Eight colored circles indicate the state of the DUT LEDs.

**Fan**
- Two colored circles indicate the state of the fan LEDs.

**Power**
- Four colored circles indicate the state of the power LEDs.

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

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

**DIP CPU**
- Eight white boxes show the state of the processor switches.

**DIP DUT**
- Four white boxes show the state of the DUT switches.

--- **Note** ---
ARM recommends that you configure the Boot DIP switches using the `boot_switch` model parameter instead of using the CLCD interface.
Changing Boot DIP switch positions while the model is running can result in unpredictable behavior.

**Total Instr**
- A counter showing the total number of instructions executed.
The system models provide a programmer’s view of the system, so the total instructions are displayed rather than total processor cycles. Timing might differ substantially from the hardware because:

- the bus fabric is simplified
- memory latencies are minimized
- cycle approximate processor 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. The figure below shows the CLCD with Rate Limit enabled.

--- Note ---
You can control whether Rate Limit is enabled by using the rate_limit-enable parameter, one of the visualization parameters for the MPS Visualization component, when instantiating the model.

**CLCD display**
The area at the bottom of the window displays the contents of the CLCD buffer.
Figure 3-5 Visualization window with CLCD buffer displayed

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

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. This is one of the visualization parameters for the MPS Visualization component.

3.6.1 See also

Tasks
- Starting the FVP using Model Shell on page 3-3.

Reference
- Debugging a FVP on page 3-2
- Configuring VE and MPS FVPs on page 3-5
- Using the VE CLCD window on page 3-8
- Using Ethernet with a VE FVP on page 3-15
- Using a terminal with a system model on page 3-17
- Virtual filesystem on page 3-19
- MPS visualization parameters on page 5-12
3.7 Using Ethernet with a VE FVP

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

3.7.1 Host requirements

Before you can use the Ethernet capability of the VE FVP, you must first set up your host computer. You can find more information in the Fast Models User Guide.

3.7.2 Target requirements

The VE FVPs include a software implementation of the SMSC91C111 Ethernet controller. Your target OS must therefore include a driver for this specific device, and you must configure the kernel to use the SMSC chip. Linux supports the SMSC91C111.

There are three SMSC91C111 component parameters, and when you configure these prior to starting the VE FVP, you:

• Specify the TAP device name.
• Set the MAC address.
• Define whether promiscuous mode is enabled.

enabled

This is the default state. When the device is disabled, the kernel cannot detect the device. For more information, see the SMSC_91C111 component section in the Fast Models Reference Manual. The following figure shows a block diagram of the model networking structure:

![Figure 3-6 Model networking structure block diagram](image)

You must configure a HostBridge component to perform read and write operations on the TAP device. The HostBridge component is a virtual programmer’s view model, which acts as a networking gateway to exchange Ethernet packets with the TAP device on the host, and to forward packets to NIC models.

mac_address

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

**promiscuous**

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

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

### 3.7.3 Configuring Ethernet

For information on configuring a connection to the Ethernet interface on the FVP from Microsoft Windows or Linux, see the *Fast Models User Guide*.

### 3.7.4 See also

**Tasks**

- *Starting the FVP using Model Shell* on page 3-3.

**Reference**

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
3.8 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 Microsoft Windows 7 client, you must first install Telnet. The Telnet application is not installed on Microsoft Windows 7 by default.

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

1. Select Start → Control Panel → Programs and Features. This opens a window that enables you to uninstall or change programs.

2. Select Turn Windows features on or off on the left side of the bar. This opens the Microsoft Windows Features dialog. Select the Telnet Client check box.

3. Click OK. The installation of Telnet might take several minutes to complete.

The following figure shows a block diagram of one possible relationship between the target and host through the Terminal component. The TelnetTerminal block is what you configure when you define Terminal component parameters. The Virtual Machine is your VE FVP or MPS FVP.

![Terminal block diagram](image)

On the target side, the console process invoked by your target OS relies on a suitable driver being present. Such drivers are normally part of the OS kernel. The driver passes serial data through a UART. The data is forwarded to the TelnetTerminal component, which exposes a TCP/IP port to the world outside of the FVP. This port can be connected to by, for example, a Telnet process on the host.
By default, the VE FVP or MPS FVP starts four telnet Terminals when the model is initialized. You can change the startup behavior for each of the four Terminals by modifying the corresponding component parameters.

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 FVP 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 either telnet mode or raw mode.

### 3.8.1 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.

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

### 3.8.3 See also

**Tasks**

- *Starting the FVP using Model Shell on page 3-3.*

**Reference**

- *Debugging a FVP on page 3-2*
- *Configuring VE and MPS FVPs on page 3-5*
- *Loading and running an application on the VE FVP on page 3-7*
- *Using the VE CLCD window on page 3-8*
- *Using Ethernet with a VE FVP on page 3-15*
- *Virtual filesystem on page 3-19*
- *Using the VFS with a prebuilt FVP on page 3-21*
- *Terminal parameters on page 4-17.*
3.9 Virtual filesystem

The Virtual File System (VFS) enables your target to access parts of a host file system. 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 file system is analogous to access to a shared network drive, and can be expected to behave in the same way.

If you want to build your own system that includes the VFS, see the reference information at the end of this topic. See also the WritingADriver.txt file in %PVLIB_HOME%\VFS2\docs\.

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

The VFS supports the following file system operations:
- getattr retrieves metadata for the file, directory or symbolic link
- mkdir creates a new directory
- remove removes a file, directory or symbolic link
- rename renames a file, directory or symbolic link
- rmdir removes an empty directory
- setattr sets metadata for the file, directory or symbolic link.

Note
setattr is not currently implemented.

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

The VFS supports the following mount points:
- closemounts frees the iterator handle returned from openmounts
- openmounts retrieves an iterator handle for the list of available mounts
- readmounts reads one entry from the mount iterator ID.

The VFS supports the following directory iterators:
- closedir frees a directory iterator handle retrieved by opendir
- opendir retrieves an iterator handle for the directory specified
- readdir reads the next entry from the directory iterator.

Note
Datestamps returned are in milliseconds elapsed since the VFS epoch of January 01 1970 00:00 UTC and are host datestamps. The host datestamp might be in the future relative to the simulated OS datestamp.

The VFS supports the following file operations:
- closefile frees a handle opened with openfile
- filesystem 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.

### 3.9.1 See also

**Tasks**
- Starting the FVP using Model Shell on page 3-3.

**Reference**
- Debugging a FVP on page 3-2
- Configuring VE and MPS FVPs on page 3-5
- Loading and running an application on the VE FVP on page 3-7
- Using the VE CLCD window on page 3-8
- Using Ethernet with a VE FVP on page 3-15
- Using a terminal with a system model on page 3-17
- Using the VFS with a prebuilt FVP on page 3-21
- Fast Models Reference Manual,
3.10 Using the VFS with a prebuilt FVP

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

To use the VFS functionality of the VE FVP, use the `motherboard.vfs2.mount` configuration parameter when you start the model. The value of the parameter is the path to the host filesystem directory that is to be made accessible within the model.

3.10.1 Mount names

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

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

You can then access the host filesystem from the target OS through a supported filesystem operation. See the `ReadMe.txt` file in the `%PVLIB_HOME%\VFS2\\linux\` directory.

3.10.2 Path names

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

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

3.10.3 See also

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *VFS2 parameters* on page 4-18
Chapter 4
Programmer’s Reference for the VE FVPs

The following topics describe the memory map and the configuration registers for the peripheral and system component models.

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

Tasks
• Starting the FVP using Model Shell on page 3-3.

Reference
• VE model memory map on page 4-2
• VE model parameters on page 4-5
• Motherboard peripheral parameters on page 4-6
• Motherboard virtual component parameters on page 4-13
• FVP_VE_Cortex-A15xn CoreTile parameters on page 4-20
• ARMv8-A AEM parameters on page 4-23
• Differences between the VE and CoreTile hardware and the models on page 4-40
• Memory map on page 4-41
• Memory aliasing on page 4-42
• Features not present in the model on page 4-43
• Features partially implemented in the model on page 4-44
• Restrictions on the processor models on page 4-45
• Timing considerations on page 4-47.
4.1 VE model memory map

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

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Modeled</th>
<th>Address range</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR FLASH0 (CS0)</td>
<td>Yes</td>
<td>0x00_00000000–0x00_03FFFFFF</td>
<td>64MB</td>
</tr>
<tr>
<td>Reserved</td>
<td>-</td>
<td>0x00_04000000–0x00_07FFFFFF</td>
<td>64MB</td>
</tr>
<tr>
<td>NOR FLASH0 alias (CS0)</td>
<td>Yes</td>
<td>0x00_08000000–0x00_0BFFFFFF</td>
<td>64MB</td>
</tr>
<tr>
<td>NOR FLASH1 (CS4)</td>
<td>Yes</td>
<td>0x00_0C000000–0x00_0FFFFFFF</td>
<td>64MB</td>
</tr>
<tr>
<td>Unused (CS5)</td>
<td>-</td>
<td>0x00_10000000–0x00_13FFFFFF</td>
<td>-</td>
</tr>
<tr>
<td>PSRAM (CS1) - unused</td>
<td>No</td>
<td>0x00_14000000–0x00_17FFFFFF</td>
<td>-</td>
</tr>
<tr>
<td>Peripherals (CS2). See Table 4-3 on page 4-3.</td>
<td>Yes</td>
<td>0x00_18000000–0x00_1BFFFFFF</td>
<td>64MB</td>
</tr>
<tr>
<td>Peripherals (CS3). See Table 4-4 on page 4-3.</td>
<td>Yes</td>
<td>0x00_1C000000–0x00_1FFFFFFF</td>
<td>64MB</td>
</tr>
<tr>
<td>CoreSight and peripherals</td>
<td>No</td>
<td>0x00_20000000–0x00_20000000</td>
<td>-</td>
</tr>
<tr>
<td>Graphics space</td>
<td>No</td>
<td>0x00_20000000–0x00_20000000</td>
<td>-</td>
</tr>
<tr>
<td>System SRAM</td>
<td>Yes</td>
<td>0x00_20000000–0x00_20000000</td>
<td>64KB</td>
</tr>
<tr>
<td>Ext AXI</td>
<td>No</td>
<td>0x00_20000000–0x00_20000000</td>
<td>-</td>
</tr>
<tr>
<td>4GB DRAM (in 32-bit address space)b</td>
<td>Yes</td>
<td>0x00_80000000–0x00_80000000</td>
<td>2GB</td>
</tr>
<tr>
<td>Unused</td>
<td>-</td>
<td>0x00_80000000–0x00_80000000</td>
<td>-</td>
</tr>
<tr>
<td>4GB DRAM (in 36-bit address space)b</td>
<td>Yes</td>
<td>0x08_00000000–0x08_00000000</td>
<td>4GB</td>
</tr>
<tr>
<td>Unused</td>
<td>-</td>
<td>0x08_00000000–0x08_00000000</td>
<td>-</td>
</tr>
<tr>
<td>4GB DRAM (in 40-bit address space)b</td>
<td>Yes</td>
<td>0x80_00000000–0x80_00000000</td>
<td>4GB</td>
</tr>
</tbody>
</table>

a. The private peripheral region address 0x2c000000 is mapped in this region. The parameter PERIPHBASE can be used to map the peripherals to a different address.
b. The model contains only 4GB of DRAM. The DRAM memory address space is aliased across the three different regions and where the mapped address space is greater than 4GB.
The model has a secure_memory option. When you enable this option, the memory map is changed for a number of peripherals as shown in the following table:

**Table 4-2 CS2 peripheral memory map for secure_memory option**

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Address range</th>
<th>Functionality with secure_memory enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR FLASH0 (CS0)</td>
<td>0x00_00000000–0x00_0001FFFF</td>
<td>Secure RO, aborts on non-secure accesses.</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00_04000000–0x00_0401FFFF</td>
<td>Secure SRAM, aborts on non-secure accesses.</td>
</tr>
<tr>
<td>NOR FLASH0 alias (CS0)</td>
<td>0x00_08000000–0x00_7DFFFFFF</td>
<td>Normal memory map, aborts on secure accesses.</td>
</tr>
<tr>
<td>Ext AXI</td>
<td>0x00_7e000000–0x00_7FFFFFFF</td>
<td>Secure DRAM, aborts on non-secure accesses.</td>
</tr>
<tr>
<td>4GB DRAM (in 32-bit address space)</td>
<td>0x00_80000000–0xFF_FFFFFFFF</td>
<td>Normal memory map, aborts on secure accesses.</td>
</tr>
</tbody>
</table>

The following table shows details of the memory map for peripherals in the CS2 region:

**Table 4-3 CS2 peripheral memory map**

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Modeled</th>
<th>Address range</th>
<th>Size</th>
<th>GIC Inta</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRAM - aliased</td>
<td>Yes</td>
<td>0x00_18000000–0x00_19FFFFFF</td>
<td>32MB</td>
<td>-</td>
</tr>
<tr>
<td>Ethernet (SMSC 91C111)</td>
<td>Yes</td>
<td>0x00_1A000000–0x00_1AFFFFFF</td>
<td>16MB</td>
<td>47</td>
</tr>
<tr>
<td>USB - unused</td>
<td>No</td>
<td>0x00_1B000000–0x00_1BFFFFF</td>
<td>16MB</td>
<td>-</td>
</tr>
</tbody>
</table>

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

The following table shows details of the memory map for peripherals in the CS3 region:

**Table 4-4 CS3 peripheral memory map**

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Modeled</th>
<th>Address range</th>
<th>Size</th>
<th>GIC Inta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local DAP ROM</td>
<td>No</td>
<td>0x00_1C000000–0x00_1C00FFFF</td>
<td>64KB</td>
<td>-</td>
</tr>
<tr>
<td>VE System Registers</td>
<td>Yes</td>
<td>0x00_1C010000–0x00_1C01FFFF</td>
<td>64KB</td>
<td>-</td>
</tr>
<tr>
<td>System Controller (SP810)</td>
<td>Yes</td>
<td>0x00_1C020000–0x00_1C02FFFF</td>
<td>64KB</td>
<td>-</td>
</tr>
<tr>
<td>TwoWire serial interface (PCIe)</td>
<td>No</td>
<td>0x00_1C030000–0x00_1C03FFFF</td>
<td>64KB</td>
<td>-</td>
</tr>
<tr>
<td>AACI (PL041)</td>
<td>Yes</td>
<td>0x00_1C040000–0x00_1C04FFFF</td>
<td>64KB</td>
<td>43</td>
</tr>
<tr>
<td>MCI (PL180)</td>
<td>Yes</td>
<td>0x00_1C050000–0x00_1C05FFFF</td>
<td>64KB</td>
<td>41, 42</td>
</tr>
<tr>
<td>KMI - keyboard (PL050)</td>
<td>Yes</td>
<td>0x00_1C060000–0x00_1C06FFFF</td>
<td>64KB</td>
<td>44</td>
</tr>
<tr>
<td>KMI - mouse (PL050)</td>
<td>Yes</td>
<td>0x00_1C070000–0x00_1C07FFFF</td>
<td>64KB</td>
<td>45</td>
</tr>
</tbody>
</table>
The VE FVP implementation of memory does not require programming the memory controller with the correct values. This means you must ensure that the memory controller is set up properly if you run an application on actual hardware. If this is not done, applications that run on a FVP might fail on actual hardware.

### 4.1.1 See also

**Reference**

- *VE model parameters on page 4-5*
- *Differences between the VE and CoreTile hardware and the models on page 4-40.*
4.2 VE model parameters

The Fixed Virtual Platforms for the VE reference system have configuration parameters that you can define at run time:

- *Motherboard peripheral parameters* on page 4-6
- *Motherboard virtual component parameters* on page 4-13
- *FVP_VE_Cortex-A15xn CoreTile parameters* on page 4-20
- *ARMv8-A AEM parameters* on page 4-23.

Note
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.

4.2.1 See also

Reference
- *VE model memory map* on page 4-2
- *Differences between the VE and CoreTile hardware and the models* on page 4-40.
4.3 Motherboard peripheral parameters

You can configure the following peripheral parameters on the motherboard:

- Color LCD controller parameters on page 4-7
- Ethernet parameters on page 4-8
- System controller parameters on page 4-9
- VE system register block parameters on page 4-10
- UART parameters on page 4-11
- Watchdog parameters on page 4-12.

4.3.1 See also

Reference

- FLASH loader parameters on page 4-14
- Host bridge parameters on page 4-15
- Multimedia card parameters on page 4-16
- Terminal parameters on page 4-17
- Visualization parameters on page 4-19.
4.4  Color LCD controller parameters

The table below lists the Color LCD Controller instantiation-time parameters that you can change after the model starts.

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

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

### Table 4-5 Color LCD controller parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel_double_limit</td>
<td>The threshold in horizontal pixels below which pixels sent to the frame-buffer are doubled in size in both dimensions.</td>
<td>Integer</td>
<td>-</td>
<td>0x12C</td>
</tr>
</tbody>
</table>

4.4.1  See also

Reference

- Motherboard peripheral parameters on page 4-6
- Ethernet parameters on page 4-8
- System controller parameters on page 4-9
- VE system register block parameters on page 4-10
- UART parameters on page 4-11
- Watchdog parameters on page 4-12.
4.5 Ethernet parameters

The table below lists the Ethernet instantiation-time parameters that you can change when the model is started.

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Host interface connection enabled</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>mac_address</td>
<td>Host/model MAC address</td>
<td>String</td>
<td>See mac_address parameter</td>
<td>00:02:f7:ef:31:11</td>
</tr>
<tr>
<td>promiscuous</td>
<td>Put host into promiscuous mode, for example when sharing the Ethernet controller with the host OS.</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
</tbody>
</table>

4.5.1 mac_address parameter

There are two options for the mac_address parameter:

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

--- Note ---

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

4.5.2 See also

Reference

- Using Ethernet with a VE FVP on page 3-15
- Motherboard peripheral parameters on page 4-6
- Color LCD controller parameters on page 4-7
- System controller parameters on page 4-9
- VE system register block parameters on page 4-10
- UART parameters on page 4-11
- Watchdog parameters on page 4-12.
4.6 System controller parameters

The table below lists the system controller instantiation-time parameters that you can change when the model is started.

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

```
motherboard.sp810_sysctrl1.parameter=value
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysid</td>
<td>Value for system identification register</td>
<td>Integer</td>
<td>0, 1, 2&lt;sup&gt;a&lt;/sup&gt;</td>
<td>0x00000000</td>
</tr>
<tr>
<td>use_s8</td>
<td>Select whether switch S8 is enabled</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

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

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

4.6.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-6
- *Color LCD controller parameters* on page 4-7
- *Ethernet parameters* on page 4-8
- *VE system register block parameters* on page 4-10
- *UART parameters* on page 4-11
- *Watchdog parameters* on page 4-12.
4.7 VE system register block parameters

The table below lists the VE system register instantiation-time parameters that you can change when the model is started.

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_switches_value</td>
<td>User switch</td>
<td>Integer</td>
<td>-</td>
<td>0x00</td>
</tr>
<tr>
<td>tilePresent</td>
<td>CoreTile fitted status</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
</tbody>
</table>

4.7.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-6
- *Color LCD controller parameters* on page 4-7
- *Ethernet parameters* on page 4-8
- *System controller parameters* on page 4-9
- *UART parameters* on page 4-11
- *Watchdog parameters* on page 4-12.
4.8 UART parameters

The table below lists the UART instantiation-time parameters that you can change when the model is started.

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

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

where \( x \) is the UART identifier 0, 1, 2, or 3.

Table 4-9 UART parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>baud_rate</td>
<td>Baud rate</td>
<td>Integer</td>
<td>-</td>
<td>0x9600</td>
</tr>
<tr>
<td>clock_rate</td>
<td>Clock rate for PL011</td>
<td>Integer</td>
<td>-</td>
<td>0xE10000</td>
</tr>
<tr>
<td>in_file</td>
<td>Input file</td>
<td>String</td>
<td>[empty string]</td>
<td></td>
</tr>
<tr>
<td>out_file</td>
<td>Output file (use “.” to send all output to stdout)</td>
<td>String</td>
<td>[empty string]</td>
<td></td>
</tr>
<tr>
<td>in_file_escape_sequence</td>
<td>Input file escape sequence</td>
<td>String</td>
<td>##</td>
<td></td>
</tr>
<tr>
<td>shutdown_on_eot</td>
<td>Shutdown simulation when an EOT (ASCII 4) char is transmitted</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>buffered_output</td>
<td>Unbuffered output</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>untimed_fifos</td>
<td>Ignore the clock rate and transmit/receive serial data immediately</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>uart_enable</td>
<td>Enable the UART when the system starts</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

4.8.1 See also

Reference

- Motherboard peripheral parameters on page 4-6
- Color LCD controller parameters on page 4-7
- Ethernet parameters on page 4-8
- System controller parameters on page 4-9
- VE system register block parameters on page 4-10
- Watchdog parameters on page 4-12.
4.9 Watchdog parameters

The table below lists the watchdog instantiation-time parameters that you can change when the model is started.

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simhalt</td>
<td>Halt on reset</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

4.9.1 See also

Reference

- Motherboard peripheral parameters on page 4-6
- Color LCD controller parameters on page 4-7
- Ethernet parameters on page 4-8
- System controller parameters on page 4-9
- VE system register block parameters on page 4-10
- UART parameters on page 4-11.
4.10 Motherboard virtual component parameters

The following topics describe the virtual component parameters that you can change on the motherboard:

- FLASH loader parameters on page 4-14
- Host bridge parameters on page 4-15
- Multimedia card parameters on page 4-16
- Terminal parameters on page 4-17
- VFS2 parameters on page 4-18
- Visualization parameters on page 4-19.

4.10.1 See also

Reference

- VE model parameters on page 4-5
- Motherboard peripheral parameters on page 4-6
- FVP_VE_Cortex-A15xn CoreTile parameters on page 4-20
- ARMv8-A AEM parameters on page 4-23
4.11 FLASH loader parameters

The table below lists the FLASH loader instantiation-time parameters that you can change when the model is started.

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

```
motherboard.flashloaderx.parameter=value
```

where \( x \) is the FLASH identifier 0 or 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>Path to the host file used to initialize FLASH contents when the model starts. The file can be gzip compressed.</td>
<td>String</td>
<td>Valid filename</td>
<td>[empty string]</td>
</tr>
<tr>
<td>fnameWrite</td>
<td>Path to the host file used to save FLASH contents when the model exits.</td>
<td>String</td>
<td>Valid filename</td>
<td>[empty string]</td>
</tr>
</tbody>
</table>

4.11.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-13
- *Host bridge parameters* on page 4-15
- *Multimedia card parameters* on page 4-16
- *Terminal parameters* on page 4-17
- *VFS2 parameters* on page 4-18
- *Visualization parameters* on page 4-19.
4.12 Host bridge parameters

The table below lists the host bridge instantiation-time parameters that you can change when the model is started.

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

motherboard.hostbridge.parameter=value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>interfaceName</td>
<td>Host interface identifier</td>
<td>String</td>
<td>Valid string</td>
<td>ARM0</td>
</tr>
</tbody>
</table>

4.12.1 See also

Reference
- Motherboard virtual component parameters on page 4-13
- FLASH loader parameters on page 4-14
- Multimedia card parameters on page 4-16
- Terminal parameters on page 4-17
- VFS2 parameters on page 4-18
- Visualization parameters on page 4-19
4.13 Multimedia card parameters

The table below lists the MultiMedia Card (MMC) instantiation-time parameters that you can change when the model is started.

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

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

### Table 4-13 Multimedia card parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_mmc_file</td>
<td>File used for the MMC component backing store</td>
<td>String</td>
<td>Valid string</td>
<td>mmc.dat</td>
</tr>
<tr>
<td>p_prodName</td>
<td>Card ID product name</td>
<td>String</td>
<td>Six-character string</td>
<td>ARMmmc</td>
</tr>
<tr>
<td>p_prodRev</td>
<td>Card ID product revision</td>
<td>Integer</td>
<td>-</td>
<td>0x1</td>
</tr>
<tr>
<td>p_manid</td>
<td>Card ID manufacturer ID</td>
<td>Integer</td>
<td>-</td>
<td>0x2</td>
</tr>
<tr>
<td>p_OEMid</td>
<td>Card ID OEM ID</td>
<td>Integer</td>
<td>-</td>
<td>0xCA4D0001</td>
</tr>
<tr>
<td>p_sernum</td>
<td>Card serial number</td>
<td>Integer</td>
<td>-</td>
<td>0xCA4D0001</td>
</tr>
</tbody>
</table>

4.13.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-13
- *FLASH loader parameters* on page 4-14
- *Host bridge parameters* on page 4-15
- *Terminal parameters* on page 4-17
- *VFS2 parameters* on page 4-18
- *Visualization parameters* on page 4-19.
4.14 Terminal parameters

The table below lists the terminal instantiation-time parameters that you can change when the model is started.

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mode</code></td>
<td>Terminal initialization mode</td>
<td>String</td>
<td>telnet, raw</td>
<td>telnet</td>
</tr>
<tr>
<td><code>start_telnet</code></td>
<td>Enable terminal when the system starts</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td><code>start_port</code></td>
<td>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.</td>
<td>Integer</td>
<td>Valid port number</td>
<td>5000</td>
</tr>
</tbody>
</table>

4.14.1 See also

Reference

- Using a terminal with a system model on page 3-17
- Motherboard virtual component parameters on page 4-13
- FLASH loader parameters on page 4-14
- Host bridge parameters on page 4-15
- Multimedia card parameters on page 4-16
- VFS2 parameters on page 4-18
- Visualization parameters on page 4-19.
4.15 VFS2 parameters

The table below lists the VFS2 instantiation-time parameters that you can change when the model is started.

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

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

### Table 4-15 VFS2 parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mount</td>
<td>Path to host folder to make accessible inside the model.</td>
<td>String</td>
<td>Valid path</td>
<td>[empty string]</td>
</tr>
</tbody>
</table>

4.15.1 See also

**Reference**

- Motherboard virtual component parameters on page 4-13
- FLASH loader parameters on page 4-14
- Host bridge parameters on page 4-15
- Multimedia card parameters on page 4-16
- Terminal parameters on page 4-17
- Visualization parameters on page 4-19.
4.16 Visualization parameters

The table below lists the visualization instantiation-time parameters that you can change when the model is started.

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>trap_key</td>
<td>Trap key that works with <strong>Left Ctrl</strong> to toggle mouse display</td>
<td>Integer</td>
<td>-</td>
<td>0x68</td>
</tr>
<tr>
<td>rate_limit-enable</td>
<td>Rate limit simulation</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>disable_visualisation</td>
<td>Disable the VEVisualisation component on model startup</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

4.16.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-13
- *FLASH loader parameters* on page 4-14
- *Host bridge parameters* on page 4-15
- *Multimedia card parameters* on page 4-16
- *Terminal parameters* on page 4-17
- *VFS2 parameters* on page 4-18
4.17 **FVP_VE_Cortex-A15xn CoreTile parameters**

The table below lists the Cortex-A15 multiprocessor CoreTile parameters that you can change when you start any of the following models:

- **FVP_VE_Cortex-A15x1.**
- **FVP_VE_Cortex-A15x2.**
- **FVP_VE_Cortex-A15x4.**

All listed parameters are instantiation-time parameters. This CoreTile FVP is based on revision 2, patch 0 (r2p0) of the Cortex-A15 multiprocessor.

The syntax to use in a configuration file is:

```
cluster.parameter=value
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Allowed value</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFGSDISABLE</td>
<td>Disable some accesses to DIC registers</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>CLUSTER_ID</td>
<td>Multiprocessor cluster ID value</td>
<td>Integer</td>
<td>0-15</td>
<td>0</td>
</tr>
<tr>
<td>IMINLN</td>
<td>Instruction cache minimum line size: false=32 bytes, true=64 bytes</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>PERIPHBASE</td>
<td>Base address of peripheral memory space</td>
<td>Integer</td>
<td>-</td>
<td>0x13080000</td>
</tr>
<tr>
<td>dic-spi_count</td>
<td>Number of shared peripheral interrupts implemented</td>
<td>Integer</td>
<td>0-224, in increments of 32</td>
<td>64</td>
</tr>
<tr>
<td>internal_vgic</td>
<td>Configures whether the model of the multiprocessor contains a <strong>Virtual Generic Interrupt Controller</strong> (VGIC)</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>l1_dcache-state_modelled</td>
<td>Set whether L1 D-cache has stateful implementation</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>l1_icache-state_modelled</td>
<td>Set whether L1 I-cache has stateful implementation</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>l2_cache-size</td>
<td>Set L2 cache size in bytes</td>
<td>Integer</td>
<td>0x080000, 0x100000, 0x200000, 0x400000</td>
<td>0x400000</td>
</tr>
<tr>
<td>l2_cache-state_modelled</td>
<td>Set whether L2 cache has stateful implementation</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>l2-data-slice</td>
<td>L2 data RAM slice</td>
<td>Integer</td>
<td>0, 1 or 2</td>
<td>0</td>
</tr>
<tr>
<td>l2-tag-slice</td>
<td>L2 tag RAM slice</td>
<td>Integer</td>
<td>0 or 1</td>
<td>0</td>
</tr>
</tbody>
</table>

---

The FVP_VE_Cortex-A15MPx1 has the PERIPHBASE parameter set to 0x1F000000, which is the base address of peripheral memory space on VE hardware.
The table below describes the parameters for each Cortex-A15 processor. These parameters are set individually for each Cortex-A15 processor you have in your system. Each processor has its own timer and watchdog.

The syntax to use in a configuration file is:

```
cluster.cpu[n].parameter=value
```

where `n` is the processor number, from 0 to 3 inclusive.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Allowed value</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFGEND</td>
<td>Initialize to BE8 endianness.</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>CP15DISABLE</td>
<td>Initialize to disable access to some CP15 registers.</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>DBGROMADDR</td>
<td>This value is used to initialize the CP15 DBGDRAR register. Bits[39:12] of</td>
<td>Integer</td>
<td>0x12000003</td>
<td>0x12000003</td>
</tr>
<tr>
<td></td>
<td>this register specify the ROM table physical address.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBGROMADDRV</td>
<td>If true, this sets bits[1:0] of the CP15 DBGDRAR to indicate that the address</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>is valid.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBCSELFADDR</td>
<td>This value is used to initialize the CP15 DBGDSAR register. Bits[39:17] of</td>
<td>Integer</td>
<td>0x00010003</td>
<td>0x00010003</td>
</tr>
<tr>
<td></td>
<td>this register specify the ROM table physical address.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBCSELFADDRV</td>
<td>If true, this sets bits[1:0] of the CP15 DBGDSAR to indicate that the address</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>is valid.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEINIT</td>
<td>T32 exception enable. The default has exceptions including reset handled in</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>A32 state.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VINITHI</td>
<td>Initialize with high vectors enabled.</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>ase-presenta</td>
<td>Set whether processor model has been built with NEON™ support.</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>min_sync_level</td>
<td>Controls the minimum syncLevel by the CADI parameter interface.</td>
<td>Integer</td>
<td>0-3</td>
<td>0</td>
</tr>
<tr>
<td>semihosting-cmd_line</td>
<td>Command line available to semihosting SVC calls.</td>
<td>String</td>
<td>no limit except memory</td>
<td>[empty string]</td>
</tr>
<tr>
<td>semihosting-cwd</td>
<td>Virtual address of CWD.</td>
<td>String</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>semihosting-enable</td>
<td>Enable semihosting SVC traps.</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>semihosting-ARM_SVC</td>
<td>A32 SVC number for semihosting.</td>
<td>Integer</td>
<td>0x00000000 -</td>
<td>0x123456</td>
</tr>
<tr>
<td></td>
<td>0xFFFFFFFF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>semihosting-Thumb_SVC</td>
<td>T32 SVC number for semihosting.</td>
<td>Integer</td>
<td>0x00 - 0xFF</td>
<td>0xA8</td>
</tr>
</tbody>
</table>
4.17.1 See also

Reference

- VE model parameters on page 4-5
- Motherboard peripheral parameters on page 4-6
- Motherboard virtual component parameters on page 4-13
- ARMv8-A AEM parameters on page 4-23.
4.18 ARMv8-A AEM parameters

The ARMv8-A Architecture Envelope Model (AEM) parameters adjust the behavior of external platform components on the Versatile™ Express (VE) system board. The following topics describe these parameters:

- ARMv8-A AEM general multiprocessor parameters on page 4-24
- ARMv8-A AEM general processor parameters on page 4-28
- ARMv8-A AEM general cache parameters on page 4-30
- ARMv8-A AEM memory parameters on page 4-33
- ARMv8-A AEM debug architecture parameters on page 4-34
- ARMv8-A AEM message parameters on page 4-35
- Semihosting parameters on page 4-37
- Boundary features and architectural checkers on page 4-38
- IMPLEMENTATION DEFINED features on page 4-39.

4.18.1 See also

Reference

- VE model parameters on page 4-5
- Motherboard peripheral parameters on page 4-6
- Motherboard virtual component parameters on page 4-13
- ARMv8 Instruction Set Overview,
4.19 ARMv8-A AEM general multiprocessor parameters

You can configure the overall behavior of the models with the general multiprocessor parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_CORES</td>
<td>Number of processors implemented.</td>
<td>int</td>
<td>0x1-0x4</td>
<td>0x1</td>
</tr>
<tr>
<td>is_uniprocessor</td>
<td>true for a uniprocessor implementation. When true, NUM_CORES must be 0x1.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>PA_SIZE</td>
<td>Physical address size, in bits.</td>
<td>int</td>
<td>0x0-0x30</td>
<td>0x28</td>
</tr>
<tr>
<td>has_16bit_asids</td>
<td>Enable 16-bit Address Space IDentifiers (ASIDs).</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>auxilliary_feature_register0</td>
<td>Value for Auxiliary Feature Register 0 (ID_AFR0).</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>MIDR</td>
<td>Value for Main ID Register (MIDR).</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x410FD0F0</td>
</tr>
<tr>
<td>clear_reg_top_eret</td>
<td>Clear top 32 bits of general purpose registers on exception return.</td>
<td>int</td>
<td>0x0-0x2</td>
<td>0x1</td>
</tr>
<tr>
<td>mixed_endian</td>
<td>Enable processor to change endianness at runtime.</td>
<td>int</td>
<td>0x0-0x2</td>
<td>0x1</td>
</tr>
<tr>
<td>take_ccfail_undef</td>
<td>In AArch32, take Undefined Instruction exception even if the instruction fails its condition-codes check.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>has_thumb2ee</td>
<td>Enable T32EE support.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>t32ee_bx_to_arm</td>
<td>Behavior when T32EE attempts to Branch and Exchange (BX) to A32.</td>
<td>int</td>
<td>0x0-0x2</td>
<td>0x0</td>
</tr>
<tr>
<td>has_el2</td>
<td>Enable EL2.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>has_el3</td>
<td>Enable EL3.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>max_32bit_el</td>
<td>Maximum exception level supporting AArch32 modes.</td>
<td>int</td>
<td>-0x1-0x3</td>
<td>0x3</td>
</tr>
<tr>
<td>el0_el1_only_non_secure</td>
<td>Controls security state of EL0 and EL1 if neither EL2 nor EL3 are implemented. true means non-secure.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>has_writebuffer</td>
<td>Implement write access buffering before L1 cache.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>has_delayed_sysreg</td>
<td>Delay the functional effect of system register writes until ISB or implicit barrier.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
</tbody>
</table>
4.19.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM abort parameters on page 4-26
- ARMv8-A AEM GIC parameters on page 4-27
- ARMv8-A AEM general processor parameters on page 4-28
- ARMv8-A AEM general cache parameters on page 4-30
- ARMv8-A AEM memory parameters on page 4-33
- ARMv8-A AEM debug architecture parameters on page 4-34
- ARMv8-A AEM message parameters on page 4-35
- ARMv8-A AEM simulator parameters on page 4-36
- Boundary features and architectural checkers on page 4-38
- IMPLEMENTATION DEFINED features on page 4-39.
4.20 ARMv8-A AEM abort parameters

You can configure the abort behavior of the models with the abort parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort_execution_from_device_memory</td>
<td>Abort on execution from device memory.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>ext_abort_normal_cacheable_read_is_sync</td>
<td>Synchronous reporting of normal cacheable-read external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_normal_noncacheable_read_is_sync</td>
<td>Synchronous reporting of normal noncacheable-read external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_device_read_is_sync</td>
<td>Synchronous reporting of device read external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_so_read_is_sync</td>
<td>Synchronous reporting of strongly ordered read external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_normal_cacheable_write_is_sync</td>
<td>Synchronous reporting of normal cacheable write external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>ext_abort_normal_noncacheable_write_is_sync</td>
<td>Synchronous reporting of normal noncacheable write external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>ext_abort_device_write_is_sync</td>
<td>Synchronous reporting of device write external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>ext_abort_so_write_is_sync</td>
<td>Synchronous reporting of strongly ordered write external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_ttw_cacheable_read_is_sync</td>
<td>Synchronous reporting of TTW cacheable read external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_ttw_noncacheable_read_is_sync</td>
<td>Synchronous reporting of TTW noncacheable read external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_prefetch_is_sync</td>
<td>Synchronous reporting of instruction fetch external aborts.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>ext_abort_fill_data</td>
<td>Returned data, if external aborts are asynchronous.</td>
<td>int</td>
<td>-</td>
<td>0xFDFDFDF</td>
</tr>
<tr>
<td>unpredictable-exclusive_abort_memtype</td>
<td>MMU abort if exclusive access is not supported. 0 = none, exclusives allowed in all memory, 1 = exclusives abort in Device memory, 2 = exclusives abort in any memory type other than WB inner cacheable.</td>
<td>int</td>
<td>0x0-0x2</td>
<td>0x0</td>
</tr>
</tbody>
</table>

4.20.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general multiprocessor parameters on page 4-24.
4.21 ARMv8-A AEM GIC parameters

You can configure the Generic Interrupt Controller (GIC) behavior of the models with the GIC parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dic-spi_count</td>
<td>Number of Shared Peripheral Interrupts (SPIs) supported.</td>
<td>int</td>
<td>0x0-0xE0</td>
<td>0x40</td>
</tr>
<tr>
<td>non_secure_vgic_alias_when_ns_only</td>
<td>If no EL3 and no Secure state, the VGIC has a Secure alias. If this parameter is nonzero, the model forms a Non-secure alias from its value for the VGIC, aligned to 32KiB.</td>
<td>int</td>
<td>0x0-0xFFFFFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>internal_vgic</td>
<td>Enable VGIC peripheral.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>gicv3_cpu_interface</td>
<td>Enable GICv3 processor interface in each processor model.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>gicv3.STATUSR-implemented</td>
<td>If GICv3 processor interface enabled, enable STATUS registers.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>gicv3.IIDR_base</td>
<td>Base value for calculating GICC_IIDR value.</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x43B</td>
</tr>
<tr>
<td>gicv3.BPR-min</td>
<td>Minimum value for GICC_BPR.</td>
<td>int</td>
<td>0x0-0x3</td>
<td>0x2</td>
</tr>
</tbody>
</table>

- Enable unless a shared VGIC is present.
- Disable unless a GICv3 distributor is present.
- Non-secure copy will be this value + 1.

4.21.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general multiprocessor parameters on page 4-24
4.22 ARMv8-A AEM general processor parameters

Each processor in the multiprocessor has its own parameters. The models use the parameters for processors in sequence, from cpu0 onwards. In cases where fewer processors than the maximum number are instantiated, any parameters for uninstantiated processors are ignored.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu[n].CONFIG64</td>
<td>Enable AArch64.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].POWERCTLI</td>
<td>Default power control state.</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>cpu[n].SMPnAMP</td>
<td>This processor is in the inner shared domain, and uses its cache coherency protocol.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].CFGEND</td>
<td>Use big-endian order.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>cpu[n].CP15DISABLE</td>
<td>Disable access to some CP15 registers.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>cpu[n].ase-present</td>
<td>Enable NEON™.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].VINITHI</td>
<td>Enable high vectors. Base address 0xFFFF0000.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>cpu[n].RVBAR</td>
<td>Reset Vector Base Address when resetting into AArch64.</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>cpu[n].vfp-present</td>
<td>Enable floating-point arithmetic.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].vfp-enable_at_reset</td>
<td>Enable coprocessor access and VFP at reset.a</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>cpu[n].vfp-traps</td>
<td>Enable hardware trapping of VFP exceptions for VFPv4U.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].force-fpsid</td>
<td>Override the FPSID value.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>cpu[n].force-fpsid-value</td>
<td>Value for the overridden FPSID.</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>cpu[n].TEINIT</td>
<td>Controls the initial state of SCTLR.TE in AArch32. When set, causes AArch32 exceptions (including reset) to be taken in T32 mode.</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>cpu[n].etm-present</td>
<td>Enable Embedded Trace Macrocell (ETM).</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].min_sync_level</td>
<td>Minimum CADI syncLevel. 0 = off, 1 = syncState, 2 = postInsnIO, 3 = postInsnAll.</td>
<td>int</td>
<td>0x0-0x3</td>
<td>0x0</td>
</tr>
</tbody>
</table>

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

4.22.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general multiprocessor parameters on page 4-24
- ARMv8-A AEM cryptography parameters on page 4-29
- ARMv8-A AEM general cache parameters on page 4-30
- ARMv8-A AEM memory parameters on page 4-33
- ARMv8-A AEM debug architecture parameters on page 4-34
- Semihosting parameters on page 4-37.
4.23 ARMv8-A AEM cryptography parameters

You can configure the cryptographic behavior of the processors in the models with the cryptography parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu[n].crypto_aes</td>
<td>AES hash level. 0 = AES-128, 1 = AES-192, 2 = AES-256.</td>
<td>int</td>
<td>0x0-0x2</td>
<td>0x2</td>
</tr>
<tr>
<td>cpu[n].crypto_sha1</td>
<td>Enable SHA1.</td>
<td>int</td>
<td>0x0-0x1</td>
<td>0x1</td>
</tr>
<tr>
<td>cpu[n].crypto_sha256</td>
<td>Enable SHA256.</td>
<td>int</td>
<td>0x0-0x1</td>
<td>0x1</td>
</tr>
</tbody>
</table>

4.23.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general processor parameters on page 4-28.
4.24 ARMv8-A AEM general cache parameters

You can configure the caches of the multiprocessor with the general cache parameters.

### Table 4-24 General cache parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache_maintenance_hits_watchpoints</td>
<td>Enable AArch32 cache maintenance by DCIMAC to trigger watchpoints.(^a)</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>dcache-state_modelled</td>
<td>Stateful implementation, with line allocation in D-caches at all levels.(^b)</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>icache-state_modelled</td>
<td>Stateful implementation, with line allocation in I-caches at all levels.(^b)</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>memory.l2_cache.is_inner_cacheable</td>
<td>L2 cache is inner cacheable, not outer cacheable.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>memory.l2_cache.is_inner_shareable</td>
<td>L2 cache is inner shareable, not outer shareable.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cache-log2linelen</td>
<td>Log(_2)(cache-line length, in bytes)</td>
<td>int</td>
<td>0x4-0x8</td>
<td>0x6</td>
</tr>
<tr>
<td>cpu[, DCZID-log2-block-size]</td>
<td>Log(_2)(block size) cleared by DC ZVA instruction(^c)</td>
<td>int</td>
<td>0x0-0x9</td>
<td>0x8</td>
</tr>
<tr>
<td>dcache-size</td>
<td>L1 D-cache size, in bytes</td>
<td>int</td>
<td>0x4000-0x100000</td>
<td>0x8000</td>
</tr>
<tr>
<td>dcache-ways</td>
<td>Number of L1 D-cache ways(^d)</td>
<td>int</td>
<td>0x1-0x40</td>
<td>0x2</td>
</tr>
<tr>
<td>icache-size</td>
<td>L1 I-cache size, in bytes</td>
<td>int</td>
<td>0x4000-0x100000</td>
<td>0x8000</td>
</tr>
<tr>
<td>icache-ways</td>
<td>Number of L1 I-cache ways(^d)</td>
<td>int</td>
<td>0x1-0x40</td>
<td>0x2</td>
</tr>
<tr>
<td>l2cache-size</td>
<td>L2 cache size, in bytes</td>
<td>int</td>
<td>0x0-0x1000000</td>
<td>0x80000</td>
</tr>
<tr>
<td>l2cache-ways</td>
<td>Number of L2 cache ways(^d)</td>
<td>int</td>
<td>0x1-0x40</td>
<td>0x10</td>
</tr>
<tr>
<td></td>
<td>a. UNPREDICTABLE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b. Unified caches allocate lines only if these parameters are enabled at both I-side and D-side.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c. As read from DCZID_EL0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d. Sets are implicit from size.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.24.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general multiprocessor parameters on page 4-24
- ARMv8-A AEM general processor parameters on page 4-28
- ARMv8-A AEM L2 cache-controller parameters on page 4-31
- ARMv8-A AEM TLB parameters on page 4-32
- ARMv8-A AEM memory parameters on page 4-33
4.25 ARMv8-A AEM L2 cache-controller parameters

You can configure the Level 2 (L2) cache-controller of the multiprocessor with the L2 cache-controller parameters.

Table 4-25 L2 cache-controller parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>l2cc.cache-state_modelled</td>
<td>Model a functional cache state</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>l2cc.ASSOCIATIVITY</td>
<td>Associativity for Auxiliary Control Register</td>
<td>int</td>
<td>0x0-0x1</td>
<td>0x0</td>
</tr>
<tr>
<td>l2cc.CACHEID</td>
<td>Cache controller cache ID</td>
<td>int</td>
<td>0x0-0x3F</td>
<td>0x0</td>
</tr>
<tr>
<td>l2cc.WAYSIZE</td>
<td>Size of ways for Auxiliary Control Register</td>
<td>int</td>
<td>0x0-0x7</td>
<td>0x1</td>
</tr>
<tr>
<td>l2cc.CFGBIGEND</td>
<td>Access configuration registers as big-endian on reset</td>
<td>int</td>
<td>0x0-0x1</td>
<td>0x0</td>
</tr>
<tr>
<td>l2cc.LOCKDOWN_BY_MASTER</td>
<td>Lockdown by mastera</td>
<td>int</td>
<td>0x0-0x1</td>
<td>0x0</td>
</tr>
<tr>
<td>l2cc.LOCKDOWN_BY_LINE</td>
<td>Lockdown by lineb</td>
<td>int</td>
<td>0x0-0x1</td>
<td>0x0</td>
</tr>
</tbody>
</table>

a. The value is reflected in CacheType register Bit 26, but the feature is not switched off when the parameter is 0.
b. The value is reflected in CacheType register Bit 25, but the feature is not switched off when the parameter is 0.

4.25.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general cache parameters on page 4-30.
4.26 ARMv8-A AEM TLB parameters

You can configure the Translation Lookaside Buffer (TLB) configuration of the multiprocessor with the TLB parameters.

Table 4-26 TLB parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>stage12_tlb_size</td>
<td>Number of stage 1 and stage 2 TLB entries</td>
<td>int</td>
<td>0x1-0xFFFFFFFF</td>
<td>0x80</td>
</tr>
<tr>
<td>stage1_tlb_size</td>
<td>Number of stage 1 TLB entries</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>stage2_tlb_size</td>
<td>Number of stage 2 TLB entries</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>stage1_walkcache_size</td>
<td>Number of stage 1 TLB walk cache entries</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>stage2_walkcache_size</td>
<td>Number of stage 2 TLB walk cache entries</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>instruction_tlb_size</td>
<td>Number of stage 1 and stage 2 ITLB entries(^a)</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>enable_tlb_contig_check</td>
<td>Check consistency of TLB entries in regions with the Contiguous Bit set</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>has_tlb_conflict_abort</td>
<td>Inconsistent TLB content generates aborts</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
<tr>
<td>use_tlb_contig_hint</td>
<td>Pagetable entries with the Contiguous Bit set generate large TLB entries</td>
<td>bool</td>
<td>false-true</td>
<td>false</td>
</tr>
</tbody>
</table>

\(^a\) 0 for unified ITLB + DTLB.

4.26.1 See also

Reference
- *ARMv8-A AEM parameters on page 4-23*
- *ARMv8-A AEM general cache parameters on page 4-30.*
4.27 ARMv8-A AEM memory parameters

You can configure the memory of the multiprocessor with the memory parameters.

Table 4-27 Memory parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>elfloader.elf</td>
<td>ELF file name</td>
<td>string</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>elfloader.lfile</td>
<td>Load file for large address mapping</td>
<td>string</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>elfloader.ns_copy</td>
<td>Copy whole file to NS memory space</td>
<td>bool</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

4.27.1 See also

Reference

- *ARMv8-A AEM parameters on page 4-23*
- *ARMv8-A AEM general multiprocessor parameters on page 4-24*
- *ARMv8-A AEM general processor parameters on page 4-28*
- *ARMv8-A AEM general cache parameters on page 4-30.*
4.28 ARMv8-A AEM debug architecture parameters

You can configure the debug architecture with the debug architecture parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBGPIIDR</td>
<td>If zero, build a value for the DeBuG Peripheral Identification Register (DBGPIIDR). If nonzero, override DBGPIIDR with this value.</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>cpu[n].number-of-breakpoints</td>
<td>Number of breakpoints.</td>
<td>int</td>
<td>0x2-0x10</td>
<td>0x10</td>
</tr>
<tr>
<td>cpu[n].number-of-watchpoints</td>
<td>Number of watchpoints.</td>
<td>int</td>
<td>0x2-0x10</td>
<td>0x10</td>
</tr>
<tr>
<td>cpu[n].number-of-context-breakpoints</td>
<td>Number of context-aware breakpoints.</td>
<td>int</td>
<td>0x0-0x10</td>
<td>0x10</td>
</tr>
<tr>
<td>cpu[n].unpredictable_WPMASKANDBAS</td>
<td>Constrained unpredictable handling of watchpoints when mask and BAS fields specified. 0 = IGNOREMASK, 1 = IGNOREBAS, 2 = REPEATBAS8, 3 = REPEATBAS.</td>
<td>int</td>
<td>0x0-0x3</td>
<td>0x1</td>
</tr>
<tr>
<td>cpu[n].unpredictable_non-contigous_BAS</td>
<td>Treat noncontiguous BAS field in watchpoint control register as all ones.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].cti-number_of_triggers</td>
<td>Number of CTI event triggers.</td>
<td>int</td>
<td>0x0-0x8</td>
<td>0x8</td>
</tr>
<tr>
<td>cpu[n].cti-intack_mask</td>
<td>Set bits mean the corresponding triggers need software acknowledgment through CTIINTACK.a</td>
<td>int</td>
<td>0x0-0xFF</td>
<td>0x1</td>
</tr>
<tr>
<td>v8ect.has_CTIAUTHSTATUS</td>
<td>Enable CTIAUTHSTATUS register.</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>v8ect.number-of-channels</td>
<td>Number of channels in Cross Trigger Matrix.</td>
<td>int</td>
<td>0x3-0x20</td>
<td>0x4</td>
</tr>
<tr>
<td>watchpoint-log2secondary_restriction</td>
<td>Log2(secondary restriction of FAR/EDWAR) on watchpoint hit for load/store operations.</td>
<td>int</td>
<td>0x0-0x3F</td>
<td>0x0</td>
</tr>
</tbody>
</table>

a. One bit per trigger.

4.28.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general multiprocessor parameters on page 4-24
- ARMv8-A AEM general processor parameters on page 4-28
- ARMv8-A AEM general cache parameters on page 4-30
- ARMv8-A AEM memory parameters on page 4-33
- ARMv8-A AEM message parameters on page 4-35
4.29   ARMv8-A AEM message parameters

You can configure the warning and error messages with the message parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE.ArchMsg.suppress_repeated</td>
<td>Suppress repeated messages from similar call sites</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>TRACE.ArchMsg.suppress_sources</td>
<td>Space-separated list of components or events to not print</td>
<td>string</td>
<td>-</td>
</tr>
<tr>
<td>TRACE.ArchMsg.trace-file</td>
<td>ArchMsg output file</td>
<td>string</td>
<td>-</td>
</tr>
</tbody>
</table>

4.29.1   See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM debug architecture parameters on page 4-34
- Boundary features and architectural checkers on page 4-38.
4.30 ARMv8-A AEM simulator parameters

You can configure the simulator with these parameters.

Table 4-30 ARMv8-A AEM simulator parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler_mode</td>
<td>Control instruction interleaving. 0x0 = default long quantum, 0x1 = low latency mode, short quantum and signal checking, 0x2 = lock-breaking mode, long quantum with additional context switches near load-exclusive instructions.</td>
<td>int</td>
<td>0x0-0x2</td>
</tr>
<tr>
<td>cpu[n].max_code_cache</td>
<td>Maximum cache size for code translations, in bytes.</td>
<td>int</td>
<td>-</td>
</tr>
</tbody>
</table>

4.30.1 See also

Reference

- ARMv8-A AEM parameters on page 4-23
- ARMv8-A AEM general multiprocessor parameters on page 4-24
- ARMv8-A AEM general cache parameters on page 4-30.
4.31  Semihosting parameters

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

The semihosting parameters are repeated in groups for each processor in the multiprocessor, from cpu0 onwards.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu[n].semihosting-ARM_SVC</td>
<td>A32 SVC number for semihosted calls</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0x123456</td>
</tr>
<tr>
<td>cpu[n].semihosting-Thumb_SVC</td>
<td>T32 SVC number for semihosted calls</td>
<td>int</td>
<td>0x0-0xFFFFFFFF</td>
<td>0xAB</td>
</tr>
<tr>
<td>cpu[n].semihosting-cmd_line</td>
<td>Program name and arguments, argc, argv, for target programs using the semihosted C library</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cpu[n].semihosting-cwd</td>
<td>Virtual address of CWD</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cpu[n].semihosting-enable</td>
<td>Enable semihosting of SVC instructions</td>
<td>bool</td>
<td>false-true</td>
<td>true</td>
</tr>
<tr>
<td>cpu[n].semihosting-heap_base</td>
<td>Virtual address of heap base</td>
<td>int</td>
<td>-</td>
<td>0x00000000</td>
</tr>
<tr>
<td>cpu[n].semihosting-heap_limit</td>
<td>Virtual address of top of heap</td>
<td>int</td>
<td>-</td>
<td>0x0F000000</td>
</tr>
<tr>
<td>cpu[n].semihosting-stack_base</td>
<td>Virtual address of base of descending stack</td>
<td>int</td>
<td>-</td>
<td>0x10000000</td>
</tr>
<tr>
<td>cpu[n].semihosting-stack_limit</td>
<td>Virtual address of stack limit</td>
<td>int</td>
<td>-</td>
<td>0x0F000000</td>
</tr>
</tbody>
</table>

4.31.1  See also

Reference
-  ARMv8-A AEM parameters on page 4-23
-  ARMv8-A AEM general multiprocessor parameters on page 4-24
-  ARMv8-A AEM general processor parameters on page 4-28
-  ARMv8-A AEM general cache parameters on page 4-30
-  ARMv8-A AEM memory parameters on page 4-33.
4.32  Boundary features and architectural checkers

Boundary features and architectural checkers are model capabilities that help your development and testing process by exposing latent problems in the target code. Certain boundary features or architectural checkers, however, might have an adverse effect on the overall running speed of target code.

4.32.1  See also

Reference
- *ARMv8-A AEM parameters* on page 4-23
- *Semihosting parameters* on page 4-37
- *IMPLEMENTATION DEFINED features* on page 4-39
4.33 IMPLEMENTATION DEFINED features

Some aspects of the behavior of the processor are IMPLEMENTATION DEFINED in the ARM architecture, meaning that they can legally vary between different processor implementations. Any code that is intended to be run portably across the multiple ARM implementations must take care when using any of these facilities, because they might or might not be present.

4.33.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-23
- *Semihosting parameters* on page 4-37
- *Boundary features and architectural checkers* on page 4-38
4.34 Differences between the VE and CoreTile hardware and the models

The following topics describe features of the VE hardware that are not implemented in the models, or that have significant differences in implementation:

- Memory map on page 4-41
- Memory aliasing on page 4-42
- Features not present in the model on page 4-43
- Features partially implemented in the model on page 4-44
- Restrictions on the processor models on page 4-45
- Timing considerations on page 4-47.

4.34.1 See also

Reference

- VE model memory map on page 4-2
- VE model parameters on page 4-5.
4.35 Memory map

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

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

4.35.1 See also

Reference

• Memory aliasing on page 4-42
• Features not present in the model on page 4-43
• Features partially implemented in the model on page 4-44
• Restrictions on the processor models on page 4-45
• Timing considerations on page 4-47.
4.36 Memory aliasing

The model implements address space aliasing of the DRAM. This means that the same physical memory locations are visible at different addresses. The lower 2GB of the DRAM is accessible at 0x00_80000000. The full 8GB of DRAM is accessible at 0x08_00000000 and again at 0x80_00000000.

4.36.1 See also

Reference

- Memory map on page 4-41
- Features not present in the model on page 4-43
- Features partially implemented in the model on page 4-44
- Restrictions on the processor models on page 4-45
- Timing considerations on page 4-47.
4.37 Features not present in the model

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

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

4.37.1 See also

Reference

- VE model memory map on page 4-2
- Memory map on page 4-41
- Memory aliasing on page 4-42
- Features partially implemented in the model on page 4-44
- Restrictions on the processor models on page 4-45
- Timing considerations on page 4-47.
4.38 Features partially implemented in the model

The Sound feature present on the hardware version of the VE motherboard is only partially implemented in the Fixed Virtual Platforms. Partial implementation means that some of the components are present, but the functionality has not been fully modeled. If you use such features, they might not work as you expect. Check the model release notes for the latest information.

For the Sound feature, the VE FVPs implement the PL041 AACI PrimeCell and the audio CODEC as in the VE hardware, but with a limited number of sample rates.

4.38.1 See also

Reference

- Memory map on page 4-41
- Memory aliasing on page 4-42
- Features not present in the model on page 4-43
- Restrictions on the processor models on page 4-45
- Timing considerations on page 4-47.
4.39 Restrictions on the processor models

Detailed information concerning what features are not fully implemented in the processor models included with the VE FVPs can be found in separate documentation. See the reference information at the end of this topic.

The following general restrictions apply to the Fixed Virtual Platform implementations of ARM processors:

- The simulator does not model cycle timing. In aggregate, all instructions execute in one processor master clock cycle, with the exception of Wait For Interrupt.
- Write buffers are not modeled, except in AEMs.
- Most aspects of TLB behavior are implemented in the models. In ARMv7 models, and later ones, the TLB memory attribute settings are used when stateful cache is enabled.
- No device-accurate MicroTLB is implemented.
- A single memory access port is implemented. The port combines accesses for instruction, data, DMA and peripherals. Configuration of the peripheral port memory map register is ignored.
- All memory accesses are atomic and are performed in programmer’s view order. All transactions on the PVBus are a maximum of 64 bits wide. Unaligned accesses are always performed as byte transfers.
- Some instruction sequences are executed atomically, ahead of the component master clock, so that system time does advance during their execution. This can sometimes have an effect in sequential access of device registers where devices are expecting time to move on between each access.
- Interrupts are not taken at every instruction boundary.
- Integration and test registers are not implemented.
- Not all CP14 debug registers are implemented on all processors.
- Breakpoint types supported directly by the model are:
  - single address unconditional instruction breakpoints
  - single address unconditional data breakpoints
  - unconditional instruction address range breakpoints
- Processor exception breakpoints are supported by pseudoregisters in the debugger. Setting an exception register to a nonzero value stops execution on entry to the associated exception vector.
- Performance counters are not implemented on all models.

The following additional restrictions apply to the Fixed Virtual Platform implementation of a Cortex-A9 MPCore multiprocessor:

- The Cortex-A9MPCore multiprocessor contains some memory-mapped peripherals. These are modeled by the FVP.
- Two 4GB address spaces are seen by the model multiprocessor, 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.

4.39.1 See also

Reference
- Memory map on page 4-41
- Memory aliasing on page 4-42
- Features not present in the model on page 4-43
- Features partially implemented in the model on page 4-44
- Timing considerations on page 4-47
- Fast Models Reference Manual,
4.40 Timing considerations

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

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

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

4.40.1 See also

Reference

- Memory map on page 4-41
- Memory aliasing on page 4-42
- Features not present in the model on page 4-43
- Features partially implemented in the model on page 4-44
- Restrictions on the processor models on page 4-45.
Chapter 5
Programmer’s Reference for the MPS FVPs

The following topics describe the memory map and the configuration registers for the peripheral and system component models:

Reference
•  *MPS model memory map* on page 5-2
•  *MPS parameters* on page 5-11
•  *Differences between the MPS hardware and the system model* on page 5-16.
## 5.1 MPS model memory map

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Modeled</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>4MB Remap region for SRAM0 overlay of Flash</td>
<td>Yes</td>
<td>0x00000000–0x003FFFFF</td>
</tr>
<tr>
<td>Nonremapped Flash memory</td>
<td>Yes</td>
<td>0x00400000–0x03FFFFF</td>
</tr>
<tr>
<td>SRAM for code and data storage (remap RAM)</td>
<td>Yes</td>
<td>0x10000000–0x103FFFFF</td>
</tr>
<tr>
<td>SRAM for code and data storage</td>
<td>Yes</td>
<td>0x10400000–0x107FFFFF</td>
</tr>
<tr>
<td>FLASH aliased for programming</td>
<td>Yes</td>
<td>0x18000000–0x1BFFFFF</td>
</tr>
<tr>
<td>Processor system registers</td>
<td>Yes</td>
<td>0x1F000000–0x1F002FFF</td>
</tr>
<tr>
<td>Reserved for SMC configuration registers</td>
<td>N/A</td>
<td>0x1F001000–0x1F003FFF</td>
</tr>
<tr>
<td>I2C for DVI</td>
<td>Yes</td>
<td>0x1F003000–0x1F004FFF</td>
</tr>
<tr>
<td>PL022 SPI for Touch Screen</td>
<td>Yes</td>
<td>0x1F004000–0x1F004FFF</td>
</tr>
<tr>
<td>PL011 UART</td>
<td>Yes</td>
<td>0x1F005000–0x1F005FFF</td>
</tr>
<tr>
<td>Reserved</td>
<td>N/A</td>
<td>0x1F006000–0x1FFFFF</td>
</tr>
<tr>
<td>SP805 Watchdog</td>
<td>Yes</td>
<td>0x40000000–0x40001FFF</td>
</tr>
<tr>
<td>PL031 RTC</td>
<td>Yes</td>
<td>0x40001000–0x40001FFF</td>
</tr>
<tr>
<td>SP804 Timer (0)</td>
<td>Yes</td>
<td>0x40002000–0x40002FFF</td>
</tr>
<tr>
<td>SP804 Timer (1)</td>
<td>Yes</td>
<td>0x40003000–0x40003FFF</td>
</tr>
<tr>
<td>DUT system registers</td>
<td>Yes</td>
<td>0x40004000–0x40004FFF</td>
</tr>
<tr>
<td>PL181 SD/MMC controller</td>
<td>Yes</td>
<td>0x40005000–0x40005FFF</td>
</tr>
<tr>
<td>Reserved</td>
<td>N/A</td>
<td>0x40006000–0x40006FFF</td>
</tr>
<tr>
<td>PL011 UART (1)</td>
<td>Yes</td>
<td>0x40007000–0x40007FFF</td>
</tr>
<tr>
<td>PL011 UART (2)</td>
<td>Yes</td>
<td>0x40008000–0x40008FFF</td>
</tr>
<tr>
<td>PL011 UART (3)</td>
<td>Yes</td>
<td>0x40009000–0x40009FFF</td>
</tr>
<tr>
<td>PL041 AC97 controller</td>
<td>Yes</td>
<td>0x4000A000–0x4000AFFF</td>
</tr>
<tr>
<td>DS702 I2C (ADC DAC)</td>
<td>Partiala</td>
<td>0x4000B000–0x4000BBFF</td>
</tr>
<tr>
<td>DUT Character LCD</td>
<td>Yes</td>
<td>0x4000C000–0x4000CFFF</td>
</tr>
<tr>
<td>Reserved</td>
<td>N/A</td>
<td>0x4000D000–0x4000EFFF</td>
</tr>
<tr>
<td>Reserved</td>
<td>N/A</td>
<td>0x4FFA0000–0x4FFAFFF</td>
</tr>
<tr>
<td>Flexray</td>
<td>Partiala</td>
<td>0x4FFB0000–0x4FFBFFFFF</td>
</tr>
<tr>
<td>CAN</td>
<td>Partiala</td>
<td>0x4FFC0000–0x4FFCFFFF</td>
</tr>
<tr>
<td>LIN</td>
<td>Partiala</td>
<td>0x4FFD0000–0x4FFDFFFF</td>
</tr>
</tbody>
</table>
Table 5-1 Overview of MPS memory map (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Modeled</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>Partial</td>
<td>0x4FFE0000–0xFFFFFEFF</td>
</tr>
<tr>
<td>Video</td>
<td>Yes</td>
<td>0x4FFFF000–0xFFFFFEFF</td>
</tr>
<tr>
<td>External AHB interface to DUT FPGA</td>
<td>Yes</td>
<td>0x50000000–0xFFFFFEFF</td>
</tr>
<tr>
<td>DMC</td>
<td>Yes</td>
<td>0x60000000–0x7FFFFFFF</td>
</tr>
<tr>
<td>SMC</td>
<td>Yes</td>
<td>0xA0000000–0xAFFFFFFF</td>
</tr>
<tr>
<td>Private Peripheral Bus</td>
<td>Yes</td>
<td>0xE0000000–0xE0FFFFFF</td>
</tr>
<tr>
<td>System bus interface to DUT FPGA</td>
<td>Yes</td>
<td>0xE0100000–0xFFFFFFFF</td>
</tr>
</tbody>
</table>

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.

5.1.1 See also

Reference

- MPS registers on page 5-4
- MPS parameters on page 5-11
- Differences between the MPS hardware and the system model on page 5-16.
5.2 MPS registers

The following topics describe the MPS memory-mapped registers:

- Processor system registers on page 5-5
- DUT system registers on page 5-6
- Character LCD registers on page 5-7
- Memory configuration and remap on page 5-8
- Switches on page 5-9
- Seven-segment display on page 5-10.
5.3 Processor system registers

The following table provides a description of the processor system registers.

### Table 5-2 MPS processor system registers

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

5.3.1 See also

Reference

- MPS registers on page 5-4
- DUT system registers on page 5-6
- Character LCD registers on page 5-7
- Memory configuration and remap on page 5-8
- Switches on page 5-9
- Seven-segment display on page 5-10.
5.4 DUT system registers

The following table provides a description of the DUT system registers.

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

5.4.1 See also

Reference

- MPS registers on page 5-4
- Processor system registers on page 5-5
- Character LCD registers on page 5-7
- Memory configuration and remap on page 5-8
- Switches on page 5-9
- Seven-segment display on page 5-10.
5.5 Character LCD registers

The following table provides a description of the character LCD registers.

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

5.5.1 See also

Reference

- MPS registers on page 5-4
- Processor system registers on page 5-5
- DUT system registers on page 5-6
- Memory configuration and remap on page 5-8
- Switches on page 5-9
- Seven-segment display on page 5-10.
5.6 Memory configuration and remap

The following table provides a description of the memory configuration register.

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

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

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

5.6.1 See also

Reference
- MPS registers on page 5-4
- Processor system registers on page 5-5
- DUT system registers on page 5-6
- Character LCD registers on page 5-7
- Switches on page 5-9
- Seven-segment display on page 5-10.
5.7 Switches

The following table lists the bits for the user switch inputs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Bits</th>
<th>Access</th>
<th>Reset</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>31:8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>USER_BUT[3:0]</td>
<td>7:4</td>
<td>RO</td>
<td>-</td>
<td>Always returns value of user buttons</td>
</tr>
<tr>
<td>USER_SW[3:0]</td>
<td>3:0</td>
<td>RO</td>
<td>-</td>
<td>Always returns value of user switches</td>
</tr>
</tbody>
</table>

5.7.1 See also

Reference
- MPS registers on page 5-4
- Processor system registers on page 5-5
- DUT system registers on page 5-6
- Character LCD registers on page 5-7
- Memory configuration and remap on page 5-8
- Seven-segment display on page 5-10.
5.8 Seven-segment display

The following table lists the bits that control the seven-segment display.

Table 5-7 Seven-segment register

<table>
<thead>
<tr>
<th>Name</th>
<th>Bits</th>
<th>Access</th>
<th>Reset</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3</td>
<td>31:24</td>
<td>RW</td>
<td>0x00</td>
<td>Segments for display 3</td>
</tr>
<tr>
<td>DISP2</td>
<td>23:16</td>
<td>RW</td>
<td>0x00</td>
<td>Segments for display 2</td>
</tr>
<tr>
<td>DISP1</td>
<td>15:8</td>
<td>RW</td>
<td>0x00</td>
<td>Segments for display 1</td>
</tr>
<tr>
<td>DISP0</td>
<td>7:0</td>
<td>RW</td>
<td>0x00</td>
<td>Segments for display 0</td>
</tr>
</tbody>
</table>

5.8.1 See also

Reference

- [MPS registers](#) on page 5-4
- [Processor system registers](#) on page 5-5
- [DUT system registers](#) on page 5-6
- [Character LCD registers](#) on page 5-7
- [Memory configuration and remap](#) on page 5-8
- [Switches](#) on page 5-9.
5.9 MPS parameters

The following topics describe the system parameters that can be configured at runtime:

- MPS visualization parameters on page 5-12
- DUT parameters on page 5-13
- Terminal parameters on page 5-14
- Processor parameters on page 5-15.

5.9.1 See also

Reference

- MPS model memory map on page 5-2
- Differences between the MPS hardware and the system model on page 5-16.
5.10 MPS visualization parameters

The following table provides a description of the visualization parameters for the MPSVisualisation component.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Type</th>
<th>Allowed value</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>trap_key</td>
<td>trap key that works with Left Ctrl key to toggle mouse pointer display</td>
<td>Integer</td>
<td>valid ATKeyCode key value(^a)</td>
<td>74(^b)</td>
</tr>
<tr>
<td>rate_limit_enable</td>
<td>rate limit simulation</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>disable_visualisation</td>
<td>enable/disable visualization</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

\(^{a}\) 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.

5.10.1 See also

Reference

- DUT parameters on page 5-13
- Terminal parameters on page 5-14
- Processor parameters on page 5-15.
5.11 DUT parameters

The following table provides a description of the parameters for the DUT.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Type</th>
<th>Allowed value</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mps_dut.dut_sysregs.user_switches_value</td>
<td>User switches</td>
<td>Integer</td>
<td>0x0–0xFF</td>
<td>0</td>
</tr>
<tr>
<td>mps_dut.mmc.p_mmc_file</td>
<td>MMC contents file name</td>
<td>String</td>
<td>mmc.dat</td>
<td></td>
</tr>
<tr>
<td>mps_dut.sp805.sinhalt</td>
<td>Halt on reset</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>mps_dut.uart[0</td>
<td>1</td>
<td>2].untimed_fifos</td>
<td>Operate UART FIFO in fast (no timing) mode</td>
<td>Boolean</td>
</tr>
<tr>
<td>mps_dut.uart[0</td>
<td>1</td>
<td>2].unbuffered_output</td>
<td>Unbuffered output</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

5.11.1 See also

Reference

- MPS visualization parameters on page 5-12
- Terminal parameters on page 5-14
- Processor parameters on page 5-15.
5.12 Terminal parameters

When the MPS FVP 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 the reference information at the end of this topic.

The table below 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>
<thead>
<tr>
<th>Component name</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminal_[0-3]</td>
<td>mode</td>
<td>Terminal operation mode.</td>
<td>String</td>
<td>telnet, raw</td>
<td>telnet</td>
</tr>
<tr>
<td>terminal_[0-3]</td>
<td>start_telnet</td>
<td>Enable terminal when the system starts.</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>terminal_[0-3]</td>
<td>start_port</td>
<td>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.</td>
<td>Integer</td>
<td>Valid port number</td>
<td>5000</td>
</tr>
</tbody>
</table>

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.

5.12.1 See also

Reference

- Using a terminal with a system model on page 3-17
- MPS visualization parameters on page 5-12
- DUT parameters on page 5-13
- Processor parameters on page 5-15.
5.13 Processor parameters

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Allowed Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>semihosting-cmd_line⁴</td>
<td>Command line available to semihosting SVC calls.</td>
<td>String</td>
<td>no limit except memory</td>
<td>[empty string]</td>
</tr>
<tr>
<td>semihosting-cwd</td>
<td>Virtual address of CWD.</td>
<td>String</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>semihosting-enable</td>
<td>Enable semihosting SVC traps.</td>
<td>Boolean</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td><strong>Caution</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Applications that do not use semihosting must set this parameter to false.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>semihosting-Thumb_SVC</td>
<td>T32 SVC number for semihosting.</td>
<td>Integer</td>
<td>8 bit integer</td>
<td>0xA8</td>
</tr>
<tr>
<td>semihosting-heap_base</td>
<td>Virtual address of heap base.</td>
<td>Integer</td>
<td>0x00000000 - 0xFFFFFFFF</td>
<td>0x0</td>
</tr>
<tr>
<td>semihosting-heap_limit</td>
<td>Virtual address of top of heap.</td>
<td>Integer</td>
<td>0x00000000 - 0xFFFFFFFF</td>
<td>0x10700000</td>
</tr>
<tr>
<td>semihosting-stack_base</td>
<td>Virtual address of base of descending stack.</td>
<td>Integer</td>
<td>0x00000000 - 0xFFFFFFFF</td>
<td>0x10700000</td>
</tr>
<tr>
<td>semihosting-stack_limit</td>
<td>Virtual address of stack limit.</td>
<td>Integer</td>
<td>0x00000000 - 0xFFFFFFFF</td>
<td>0x10800000</td>
</tr>
<tr>
<td>coretile.fname</td>
<td>Flash loader filename.</td>
<td>String</td>
<td>-</td>
<td>[empty string]</td>
</tr>
<tr>
<td>coretile.flashloader.fnameWrite</td>
<td>Filename to write to if flash image is modified.</td>
<td>String</td>
<td>-</td>
<td>[empty string]</td>
</tr>
<tr>
<td>coretile.uart3.untimed_fifos</td>
<td>Ignore the clock rate and transmit or receive serial data immediately.</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>coretile.uart3.unbuffered_output</td>
<td>Unbuffered output.</td>
<td>Boolean</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

⁴ The value of argv[0] points to the first command line argument, not to the name of an image.

5.13.1 See also

Reference

- MPS visualization parameters on page 5-12
- DUT parameters on page 5-13
- Terminal parameters on page 5-14.
5.14 Differences between the MPS hardware and the system model

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

- Features not present in the model on page 5-17
- Timing considerations on page 5-18.

5.14.1 See also

Reference

- MPS model memory map on page 5-2
- MPS parameters on page 5-11.
5.15 Features not present in the model

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

The following features present on the hardware version of the MPS are not implemented in the system models:
- I2C interface
- CAN interface
- LIN
- FlexRay.

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

Note

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

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

5.15.1 See also

Reference
- Differences between the MPS hardware and the system model on page 5-16
- Timing considerations on page 5-18.
5.16 Timing considerations

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

If your code interacts with real world devices 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 FVP provides the Rate Limit feature. Enabling Rate Limit, either using the Rate Limit button in the CLCD display, or the rate_limit-enable model instantiation parameter, forces the model to run at wall clock time. This avoids issues with two clocks running at significantly different rates. For interactive applications, ARM recommends enabling Rate Limit.

5.16.1 See also

Reference

- Differences between the MPS hardware and the system model on page 5-16
- Features not present in the model on page 5-17.