Fast Models

Version 11.5

Fixed Virtual Platforms (FVP) Reference Guide

arm

Fast Models

Fixed Virtual Platforms (FVP) Reference Guide

Copyright © 2014–2018 Arm Limited or its affiliates. All rights reserved.

Release Information

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents
Fast Models Fixed Virtual Platforms (FVP) Reference Guide

Preface

This preface introduces the *Fast Models Fixed Virtual Platforms (FVP) Reference Guide*.

It contains the following:

-

About this book

Arm® Fixed Virtual Platform Reference. This manual introduces the Fixed Virtual Platforms, and describes how you can use them with other tools.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces the document.

Chapter 2 Getting Started with Fixed Virtual Platforms
This chapter describes how to use FVPs.

Glossary

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

See the *Arm® Glossary* for more information.

Typographic conventions

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

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

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

<u>mono</u>space
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

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

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

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

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

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

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title *Fast Models Fixed Virtual Platforms (FVP) Reference Guide*.
- The number 100966_1105_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

——————— Note ———————

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

————————————————

Other information

- *Arm® Developer*.
- *Arm® Information Center*.
- *Arm® Technical Support Knowledge Articles*.
- *Technical Support*.
- *Arm® Glossary*.

Chapter 1
Introduction

This chapter introduces the document.

It contains the following section:

1.1 About FVPs

Fixed Virtual Platforms (FVPs) enable development of software without the requirement for real hardware.

FVPs are supplied as standalone executables for Linux and Windows. They are not customizable, although you can configure some aspects of their behavior through command-line parameters.

Arm provides different types of FVP. They can be based on the following:
*   Armv8-A or Armv8-R Base Platform. These are called Base FVPs.
*   Arm MPS2 or Arm MPS2+ platforms, for Cortex®-M series processors. These are called MPS2 FVPs.
*   Arm Versatile™ Express development boards. These are called VE FVPs.

FVPs are available with a wide range of Armv7 and Armv8 processors, and support the CADI and MTI interfaces, so can be used for debugging and for trace output.

Another type of FVP is the Foundation Platform, which is a simple FVP that includes an Armv8-A AEM processor model, that is suitable for running bare-metal applications and for booting Linux.

Arm provides validated Linux and Android deliverables for the Armv8-A AEM Base Platform FVP and for the Foundation Platform. These are available on the Arm Community website at *Arm Development Platforms*. To get started with Linux on Armv8-A FVPs, see *Armv8-A FVPs* also on Arm Community.

Related information

Base Platform
Microcontroller Prototyping System 2
Versatile Express Model

Chapter 2
Getting Started with Fixed Virtual Platforms

This chapter describes how to use FVPs.

It contains the following sections:

2.1 Loading and running an application on an FVP

There are different ways to run an application on an FVP, for example from a command prompt, or from Model Debugger, or Arm Development Studio.

To run an FVP from the command prompt, change to the directory where your model is located. At the command prompt, enter the model name followed by the model options. To see all available options, use the `--help` option. The following options are commonly used when running an application on an FVP:

`-a [`*`instance=`*`]`*`filename`*`.axf`

Specifies the application to load, and optionally, the instance to load it on. The file can be in one of the following formats, or in gzip-compressed versions of them:

- ELF.
- Motorola S-Record.
- Intel-Hex.
- Verilog-Hex, in the format:

  `@<address_in_hex> <byte_in_hex>`

If the FVP contains multiple CPU instances, you can optionally specify the instance to load the image on. The instance name can include a wildcard (*), for example:

`FVP_Base_AEMv8A-AEMv8A -a cluster0.cpu*=__image.axf`

Omitting the instance name is equivalent to specifying all cluster and CPU instances.

`--data `*`filename`*`.bin@`*`address`*

Loads binary data into memory at the address you specify.

`-C `*`instance.parameter=value`*

Sets a single model parameter. Parameters are specified as a path that separates the instance names and the parameters using dots. For example, `-C bp.flashloader0.fname=fip.bin` loads a program into flash. To list all the available parameters, with their type, default value, and description, invoke the model with the `--list-params`, or `-l` option. To set multiple parameters at the same time, use the `-f` option instead.

`-f `*`config_file`*`.txt`

Specifies the name of a plain text configuration file. Configuration files simplify managing multiple model parameters. You can set the same parameters using this option as with the `-C` option.

`-S`

Starts a CADI debug server. This option allows a CADI-enabled debugger, such as Model Debugger or Arm Development Studio Debugger, to connect to the running model. The model waits for the debugger to connect before starting.

For example:

`models_directory/FVP_Base_Cortex-A57x1 -a __image.axf -f params.txt`

You can also launch and debug bare metal and Linux applications on an FVP from Model Debugger or Arm Development Studio Debugger. These debuggers use CADI to communicate with the FVP, so you must use the `-S` option when launching the FVP.

Starting the model opens the FVP CLCD display, which shows the contents of the simulated color LCD framebuffer.

Related information

Arm DS-5 Debugger User Guide
Model Debugger for Fast Models User Guide

2.2 Configuring the model

When you start the model from the command line, you can configure it using either:

- One or more `-C` command-line arguments.
- A configuration file and the `-f` command-line argument.

Each `-C` command-line argument or line in the configuration file must contain:

- The name of the component instance.
- The parameter to modify.
- Its value.

Use the following format:

instance.parameter=value

The `instance` can be a hierarchical path, with each level separated by a dot "." character.

———— Note ————

- Comment lines in the configuration file begin with a `#` character.
- You can set Boolean values using either `true` or `false`, or `1` or `0`.

You can generate a configuration file with all parameters set to default values by redirecting the output from the `--list-params` option into a new file, for example:

```
FVP_Base_AEMv8A.exe --list-params > params.txt
```

Example 2-1 Sample lines in a configuration file

```
# Disable semihosting using true/false syntax
coretile.cluster0.cpu0.semihosting-enable=false
#
# Enable VFP at reset using 1/0 syntax
coretile.cluster0.cpu0.vfp-enable_at_reset=1
#
# Set the baud rate for UART 0
motherboard.pl011_uart0.baud_rate=0x4800
```

2.3   FVP debug

This section describes how to debug an FVP.

FVP debug options

To debug an FVP, you can either:

- Run the FVP from within a CADI-enabled debugger.
- Start the FVP with the `-S` command-line argument and then connect a CADI-enabled debugger to it.

For information about using your debugger in these ways, see your debugger documentation.

Semihosting support

Semihosting 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.

The simulator handles semihosting by intercepting `HLT 0xF000`, `SVC 0x123456`, or `SVC 0xAB`, depending on whether the processor is in A64, A32 or T32 state. It handles all other `HLT`s and `SVC`s as normal.

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

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

Related information

Arm Compiler Software Development Guide

2.4 Using the CLCD window

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

Figure 2-1 CLCD window in its default state at startup

The top section of the CLCD window displays the status information.

USERSW

Eight white boxes show the state of the 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 `0x1C010004`.

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

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 `0x1C010004`.

The switches are in the off position by default.

———— **Note** ————

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 the red/yellow/green LEDs on the VE hardware, which are mapped to bits [7:0] of the SYS_LED register at address `0x1C010008`.

Daughter

Eight white boxes show the state of the daughterboard DIP switches and eight colored boxes show the state of the daughterboard LEDs.

Total Instr

A counter showing the total number of instructions executed.

Because the FVP models provide a *Programmer's View* (PV) of the system, the CLCD displays total instructions rather than total processor cycles. Timing might differ substantially from the hardware because:

- 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 the timing of operations within the model is not accurate.

Total Time

A counter showing the total elapsed time, in seconds.

This time 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 effect might cause timing issues.

Rate Limit is enabled by default. Simulation time is restricted so that it more closely matches real time.

To disable or enable Rate Limit, click the square button. You can configure this option when instantiating the model with the `rate_limit-enable` visualization component parameter.

When you click the **Total Instr** item in the CLCD, the display toggles to show the following:

Instr/sec

The number of instructions that execute 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 FVP CLCD displays the core run state for each core on each cluster using a colored icon. The icons are to the left of the **Total Instr** (or **Inst/sec**) item.
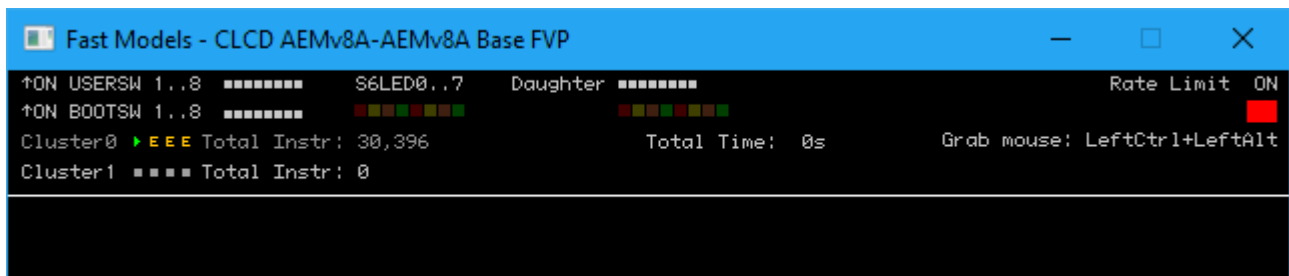

Figure 2-2 Core run state icons for a dual-cluster, quad-core model

Table 2-1 Core run state icon descriptions

Icon	State label	Description
	UNKNOWN	Run status unknown, that is, simulation has not started.
	RUNNING	Core running, is not idle, and is executing instructions.
	HALTED	External halt signal asserted.
	STANDBY_WFE	Last instruction executed was WFE and standby mode has been entered.
	STANDBY_WFI	Last instruction executed was WFI and standby mode has been entered.
	IN_RESET	External reset signal asserted.
	DORMANT	Partial core power down.
	SHUTDOWN	Complete core power down.

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. The data 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 does not have the same effect.

If you prefer to use a different key, configure it with the `trap_key` visualization component parameter.

Related information

VEVisualisation component

2.5 Ethernet with VE FVPs

This section describes how to use Ethernet with VE FVPs.

Using Ethernet with VE FVPs

The VE FVPs have a virtual Ethernet component. This component is a model of the SMSC 91C111 Ethernet controller, and uses a TAP device to communicate with the network. By default, the Ethernet component is disabled.

Host requirements

Before you can use the Ethernet capability of VE FVPs, set up your host computer.

Target requirements

This section describes the target requirements.

Target requirements - about

The VE FVPs include a software implementation of the SMSC 91C111 Ethernet controller. Your target OS must therefore include a driver for this specific device. To use the SMSC chip, configure the kernel. Linux supports the SMSC 91C111.

The configurable SMSC 91C111 component parameters are:
- `enabled`.
- `mac_address`.
- `promiscuous`.

enabled

When the device is disabled, the kernel cannot detect the device.
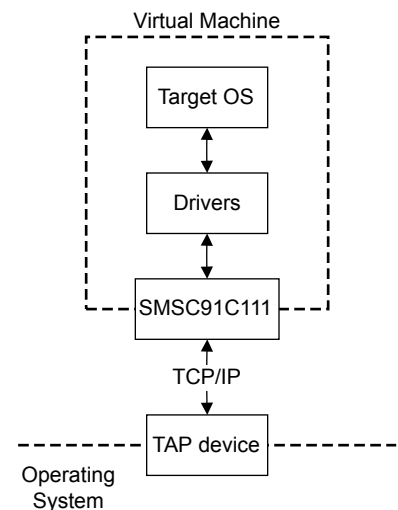
Figure 2-3 Model networking structure block diagram

To perform read and write operations on the TAP device, configure a HostBridge component. The HostBridge component is a virtual *Programmer's View* (PV) model. It acts as a networking gateway to exchange Ethernet packets with the TAP device on the host, and to forward packets to NIC models.

mac_address

There are two options for the `mac_address` parameter.

If a MAC address is not specified, when the simulator is run it takes the default MAC address, which is randomly generated. This random generation 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. In this mode, it receives all network traffic, even any not addressed to the device. Use this mode if you are using a single network device for multiple MAC addresses. Use this mode if, for example, you share the 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.

2.6   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** to open 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 FVP.
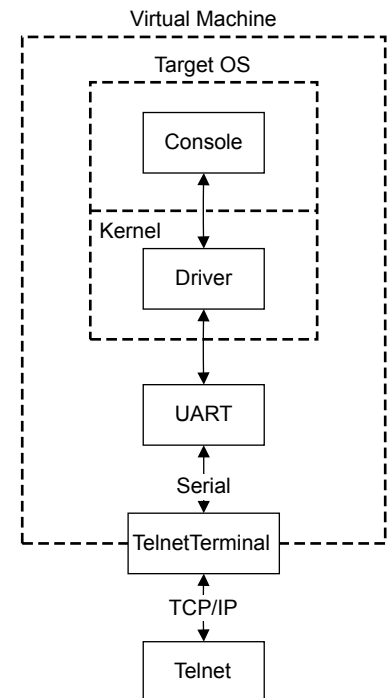

Figure 2-4 Terminal block diagram

On the target side, the console process that is 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 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 that is used by each Terminal is declared when it starts or restarts, and might not be the value that 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.

Telnet mode

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

Raw mode

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

2.7 Virtio P9 device component

The VirtioP9Device component is included in Base, BaseR, and A-profile VE platforms. It implements a subset of the Plan 9 file protocol over a virtio transport. It enables accessing a directory on the host's filesystem within Linux, or another operating system that implements the protocol, running on a platform model.

Setting up VirtioP9Device

Take the following steps to use this component:

- Use a version of Linux that supports v9fs over virtio and virtio-mmio devices.
- Update the device tree to include the VirtioP9Device component, or specify it on the kernel command-line, as shown below. The address range for both VE and Base platforms is `0x1C140000-0x1C14FFFF`.

  The interrupt number is 43, or IRQ 75, for both VE and Base platforms.
- Set the following parameter to the directory on the host that you want to mount in the model:

  VE:
  ```
  motherboard.virtiop9device.root_path
  ```

  Base:
  ```
  bp.virtiop9device.root_path
  ```
- On Linux, mount the host directory by using the following command in the model:

  ```
  $ mount -t 9p -o trans=virtio,version=9p2000.L FM <mount point>
  ```

Example kernel command-line argument

```
virtio_mmio.device=0x10000@0x1c140000:75
```

Example entry for DTS files

Add this entry next to the corresponding `virtio_block` entry:

```
virtio_p9@0140000 {
        compatible = "virtio,mmio";
        reg = <0x0 0x1c140000 0x0 0x1000>;
        interrupts = <0x0 0x2b 0x4>;
};
```