ARM® v8 Foundation Model

Version: 1.0

User Guide

ARM®

ARM DUI 0677B (ID050413)

ARMv8 Foundation Model
User Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
10 October 2012	A	Non-Confidential	First release
1 May 2013	B	Non-Confidential	Minor updates. Directory structure changed.

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARMv8 Foundation Model User Guide

Preface

This preface introduces the *ARMv8 Foundation Model*. It contains the following sections:

About this book

This book describes how to set up and use the ARMv8 Foundation Model.

Intended audience

This book is written for hardware and software developers to aid in the development of ARM-based products based on the ARMv8 architecture, using the ARMv8 Foundation Model as part of the development process.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for general information on the ARMv8 Foundation Model.

Chapter 2 *Getting started*

Read this for information on validation and testing, and how to set up and boot Linux on the ARMv8 Foundation Model.

Chapter 3 *Programmer's Reference*

Read this for technical information on the ARMv8 Foundation Model.

Glossary

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

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

Typographical conventions

The following table describes the typographical conventions:

Typographical conventions

Style	Purpose
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *PrimeCell® UART (PL011) Technical Reference Manual* (ARM DDI 0183).
- *ARM Generic Interrupt Controller Architecture Specification* (ARM IHI 0048).
- *ARMv8 Instruction Set Overview* (PRD03-GENC-010197).
- *Fast Virtual Platform Reference v1.4* (ARM DUI 0575F)
- *Fast Models User Guide v8.1* (ARM DUI 0370P)

Other publications

This section lists relevant documents published by third parties:

- Virtio specification at, https://github.com/rustyrussell/virtio-spec
- Linux kernel documentation available at, http://kernel.org/doc/Documentation/ or in your `./linux/Documentation` directory.
- The Linaro ARMv8 Linux distribution and toolchain are available from, http://www.linaro.org/engineering/armv8

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact support-esl@arm.com 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.
- The number, ARM DUI 0677B.
- The page numbers to which your comments apply.
- 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.

Customer support

ARM does not offer direct customer support for the Foundation Model. For technical support use the ARM support forums, http://forums.arm.com.

Chapter 1
Introduction

The ARMv8 Foundation Model is an enabling platform for the ARMv8 Architecture. It is a simple platform model capable of running bare-metal semi-hosted applications and booting a full operating system, with processor cluster, RAM and some basic I/O devices such as *Universal Asynchronous Receiver/Transmitters* (UARTs), block storage and network support. It also contains a simple web interface to indicate the status of the model. See *Web interface* on page 3-4.

It is supplied as a platform model with configuration of the simulation from the command line and control using peripherals in the model.

The processors in this model are not based on any existing processor design, but conform to the ARMv8 architectural specifications. This means that you can use the model for confirming software functionality, but you must not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior.

1.1 ARMv8 64-bit architecture overview

The ARMv8 architecture is both an extension and a successor to the ARMv7 architecture. The first release of the ARMv8 architecture covers the A-profile only.

The ARMv8 architecture extends the existing 32-bit architecture by introducing two execution states, a 32-bit AArch32 and a 64-bit AArch64. A number of changes have been made to the AArch32 functionality, but it remains compatible with the ARMv7 architecture. The A32 (ARM) and T32 (Thumb) instruction sets have new instructions relative to the ARMv7 ISA.

The AArch64 state introduces a new fixed-length 32-bit instruction set (A64) while maintaining support for the same architectural capabilities as ARMv7, such as TrustZone and Vitualization. Some of the new enhancements in A64 are applicable to A32, so software written for AArch32 might not be compatible with ARMv7.

AArch64 has four Exception Levels (0-3), that replace the eight processor modes found in ARMv7. The least privileged, EL0, is equivalent to user-mode and EL1 to kernel-mode with EL2 for hypervisors, while the highest privilege level, EL3, is used for the TrustZone security monitor.

Like ARMv7, AArch32 includes 13 general registers (R0-12), the Program Counter (R15) and 2 banked registers that contain the Stack Pointer (R13) and Link Register (R14). The User and System modes share these 16 registers and a Program Status Register (PSR). Naturally, the new general purpose registers are all 64-bits wide to handle larger addresses, so 32-bit accesses use the lower half of registers and either ignore or zero out the upper half. The AArch32 registers map onto the lower half of the AArch64 registers, that permits AArch32 exceptions to be taken in AArch64 at a higher exception level.

The two forms of instruction operate on either 32-bit or 64-bit values within the 64-bit general-purpose register file. Where a 32-bit instruction form is selected, the following holds true:

• The upper 32 bits of the source registers are ignored.

• The upper 32 bits of the destination register are set to zero.

• Condition flags, where set by the instruction, are computed from the lower 32 bits.

You can find more information on the ARMv8 Instruction Set in the *ARMv8 Instruction Set Overview.*

1.2 Software requirements

This section lists the host software required to run the ARMv8 Foundation Model:

Operating Systems

- Red Hat Enterprise Linux version 5.x for 64-bit Intel architectures.
- Red Hat Enterprise Linux version 6.x for 64-bit Intel architectures.
- Ubuntu 10.04 or later for 64-bit Intel architectures.

Note

At present there is no support for running the model on other operating systems. However, the model should run on any recent x86 64-bit Linux OS provided glibc v2.3.2 (or greater) and libstdc++ 6.0.0 (or greater), are present.

UART Output

For the *Universal Asynchronous Receiver/Transmitter* (UART) output to be visible, both `xterm` and `telnet` must be installed on the host, and be specified in your PATH.

1.3 Model overview

The platform provides:

- An ARMv8 processor cluster model with one to four processors that implements:
  - AArch64 at all exception levels
  - AArch32 support at EL0 and EL1.
  - Little and big endian at all exception levels.
  - Generic timers.
  - Self-hosted debug.
  - GIC v2 memory mapped processor interfaces and distributor.
- 8GB of RAM.

  Note

  The model simulates up to 8GB of RAM. To simulate a system with 4GB of RAM requires a host with at least 8GB of RAM, and to simulate 8GB RAM requires a host with at least 12GB RAM.
- An optional 128MB of secure RAM.
- Four PL011 UARTs connected to xterms.
- A network device model connected to host network resources.
- A block storage device implemented as a file on the host.
- A small system register block with LEDs and switches visible using a web server.

Caches are modelled as stateless and there are no write buffers. This gives the effect of perfect memory coherence on the data side. The instruction side has a variable side prefetch buffer so requires correct cache maintenance to operate correctly.

The models runs as fast as possible unless all the processors in the cluster are *Wait for Interrupt* (`WFI`) or *Wait for Exception* (`WFE`), in which case the model idles until an interrupt or external event occurs.

A block diagram of the ARMv8 Foundation Model is shown in Figure 1-1 on page 1-5.

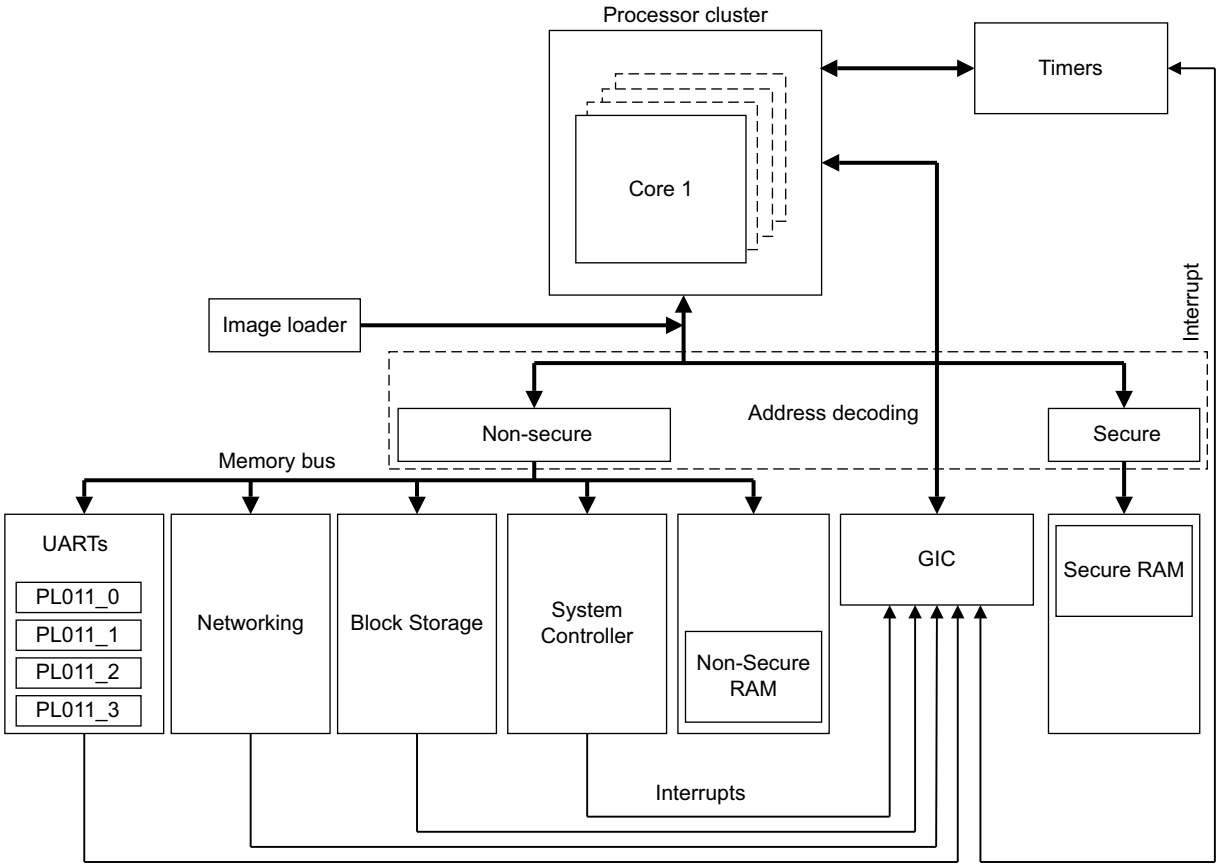

Figure 1-1 Block diagram of ARMv8 Foundation Model

Note

The behavior of the address decoding block depends on whether the `--secure-memory` command line option is used. See *Memory maps* on page 3-7.

The model provides two types of network support:

- NAT, IPv4 based networking provides limited IP connectivity by using user level IP services. This requires no extra privileges to set up or use but has inherent limitations. System level services, or services conflicting with those on the host can be provided using port remapping.
- Bridged networking requires the setup of an ethernet bridge device to bridge between the ethernet port on the host and the network interface provided by the model. This usually requires administrator privileges. See the documentation in the Linux bridge-utils package for more information.

The ARMv8 Foundation Model uses ARM Fast Model technology and forms a part of a comprehensive suite of modelling solutions for ARM processors. These modelling solutions are available:

- In the portfolio of models delivered through the ARM Fast Models product. For more information see the *Fast Models User Guide.*
- As a *Fixed Virtual Platform* (FVP).

1.3.1 Limitations of the Foundation Model

The following restrictions apply to the ARMv8 Foundation Model:

- Write buffers are not modeled.
- Interrupts are not taken at every instruction boundary.
- Caches are modeled as stateless.
- There is no *Component Architecture Debug Interface* (CADI), CADI Server, Trace or other plug-in support.
- There is no support for Thumb2EE.
- There is no support for the ARMv8 cryptography extensions.

Chapter 2
Getting started

This chapter describes validation and testing on the ARMv8 Foundation Model. It has the following sections:

2.1 Verifying the installation

The Foundation Model is available only as a prebuilt platform binary. The hierarchy shown in Figure 2-1 is used:

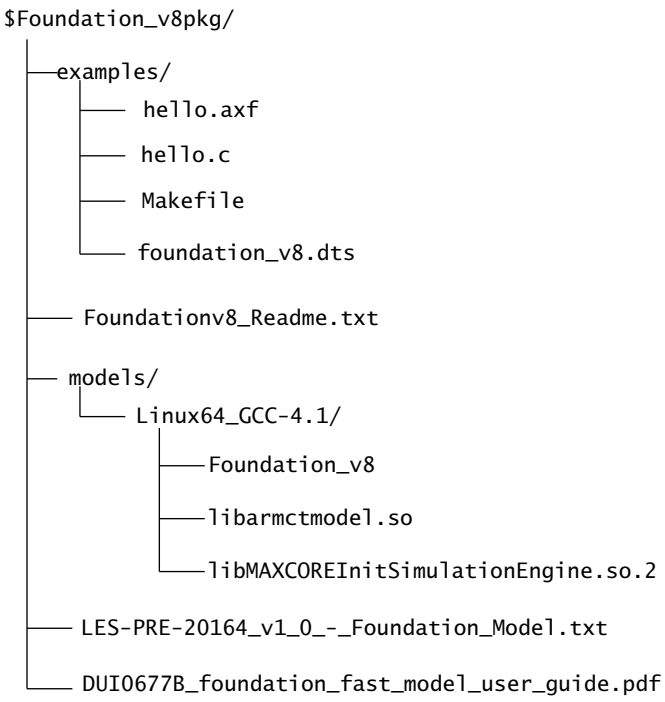

Figure 2-1 Installed files

The installation directory contents are:

examples Includes a C version and `.axf` file of the example program described in *Running the example program* on page 2-3. Also includes the Makefile and the example source code for the device tree `Foundation_v8.dts`.

FoundationV8_Readme

Read me file. A short summary of this user guide.

Foundation_v8

The ARMv8 Foundation Model executable file.

libarmctmodel.so

Code translation library.

libMAXCOREInitSimulationEngine.so

Helper library required by the model.

LES-PRE-20164_V1_-_Foundation_Model.txt

End User License Agreement text.

DUI0677B_foundation_fast_model_user_guide.pdf

This document.

2.2 Running the example program

Use the example program supplied to confirm that the ARMv8 Foundation Model is working correctly.

Run the model with the command line:

```
./Foundation_v8 --image hello.axf
```

It should print output similar to:

```
terminal_0: Listening for serial connection on port 5000
terminal_1: Listening for serial connection on port 5001
terminal_2: Listening for serial connection on port 5002
terminal_3: Listening for serial connection on port 5003

Simulation is started

Hello, 64-bit world!
Simulation is terminating. Reason: Simulation stopped
```

The example demonstrates that the model initializes correctly, loads and executes the example program, and that the semi-hosting calls to print output and stop the model are working.

Add `--quiet` to suppress all but the output from the example program.

2.2.1 Troubleshooting the example program

If you try to run the example program on a 32-bit Linux host, it will give an error similar to:

```
./Foundation_v8: /lib64/ld-linux-x86-64.so.2: bad ELF interpreter: No such file or
directory
```

If `libstdc++` is not installed on your system, you will get the following error on startup:

```
./Foundation_v8: error while loading shared libraries: libstdc++.so.6: cannot open
shared object file
```

If your system `glibc` is too old, or your `libstdc++` is too old, you will see the following messages:

```
./Foundation_v8: /usr/lib64/libstdc++.so.6: version `GLIBCXX_3.4' not found (required
by Foundation_v8)

./Foundation_v8: /lib64/libc.so.6: version `GLIBC_2.3.2' not found (required by
Foundation_v8)

./Foundation_v8: /lib64/libc.so.6: version `GLIBC_2.2.5' not found (required by
Foundation_v8)
```

`libstdc++` and `glibc` will normally be part of your core OS installation.

2.3 Using Linux

For information on configuring and building the arm64 port of Linux to run on the ARMv8 Foundation model, see, http://www.linaro.org/engineering/armv8.

Chapter 3
Programmer's Reference

This chapter contains reference information in the following sections:

3.1 Command line overview

All model configuration is provided by command line arguments. Run the model with `--help` to get a summary of the available commands.

The syntax to use on the command line is:

```
./Foundation_v8 [OPTIONS...]
```

The options are shown in Table 3-1:

Table 3-1 Command line options

`--help`	Display this help message and quit.
`--version`	Display the version and build numbers and quit.
`--quiet`	Suppress any non-simulated output on stdout or stderr.
`--cores=N`	Specify the number of cores, where `N` is 1 to 4 (default: 1). See *Multicore configuration* on page 3-6.
`--bigendian`	Start processors in big endian mode (default: little endian).
`--(no-)secure-memory`	Enable or disable separate Secure and Non-secure address spaces (default: disabled).
`--block-device=file`	Image file to use as persistent block storage.
`--read-only`	Mount block device image in read-only mode.
`--image=file`	ELF image to load.
`--data=file@address`	Raw file to load at an address in secure memory.
`--nsdata=file@address`	Raw file to load at an address in non-secure memory.
`--(no-)semihost`	Enable or disable semi-hosting support (default: enabled). See *Semi-hosting* on page 3-10.
`--semihost-cmd=cmd`	A string used as the semi-hosting command line. See *Semi-hosting* on page 3-10.
`--uart-start-port=P`	Attempt to listen on a free TCP port in the range P to P+100 for each UART (default: 5000)
`--network=(none|nat|bridged)`	Configure mode of network access (default: none).
`--network-nat-subnet=S`	Subnet used for NAT networking (default: 172.20.51.0/24).
`--network-nat-ports=M`	Optional comma separated list of NAT port mappings in the form: host_port=model_port, for example, 8022=22.
`--network-mac-address`	MAC address to use for networking (default: 00:02:f7:ef:f6:74).
`--network-bridge=dev`	Bridged network device name (default: ARM0).
`--switches=val`	Initial setting of switches in the system register block (default: 0).
`--(no-)visualization`	Starts a small web server to visualize model state (default: disabled). See *Web interface* on page 3-4.
`--use-real-time`	Sets the generic timer registers to report a view of real time as it is seen on the host platform, irrespective of how slow or fast the simulation is running.

If more than one `--image`, `--data` or `--nsdata` option is given, the images and data are loaded in the order that they appear on the command line and the simulation starts from the entry point of the last ELF specified.

More than one `--image`, `--data` or `--nsdata` option can be given.

3.2 Web interface

The syntax to use on the command line is:

```
./Foundation_v8 --(no-)visualization
```

Running the model with the `--visualization` option and without the `--quiet` option, shows the additional output:

```
terminal_0: Listening for serial connection on port 5000
terminal_1: Listening for serial connection on port 5001
terminal_2: Listening for serial connection on port 5002
terminal_3: Listening for serial connection on port 5003

Visualization web server started on port 2001
```

The `terminal_n` lines relate to the UARTs. See *UARTs* on page 3-5.

Use your web browser to go to the address http://127.0.0.1:2001.

The browser displays a visualization window as in Figure 3-1.

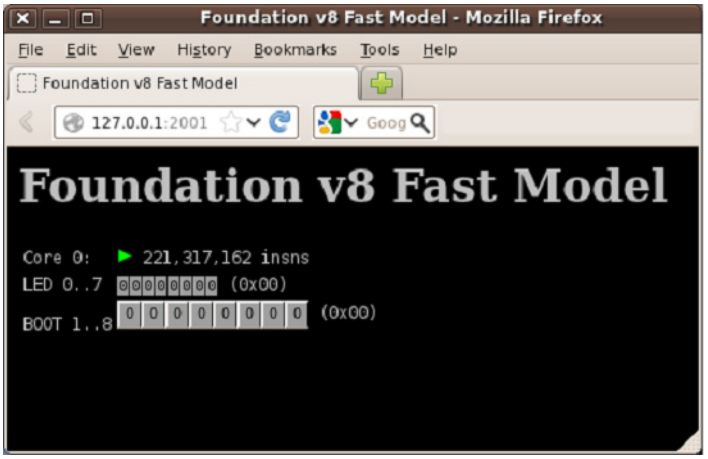

Figure 3-1 Visualization window

The visualization window provides a dynamic view of the state of various parts of the model and the ability to change the state of platform switches.

3.3 UARTs

When the Foundation Model starts, it initializes four UARTs. For each UART it serches for a free TCP port to use for telnet access to the UART. It does this by sequentially scanning a range of 100 ports and using the first free port. The start port defaults to 5000 and can be changed using the `--uart-start-port` command line parameter.

Connecting a terminal or program to the given port displays and receives output from the associated UART and permits input to the UART.

If no terminal or program is connected to the port when data is output from the UART, a terminal is started automatically.

3.3.1 UART output

For the UART output to be visible, both `xterm` and `telnet` must be installed on the host, and be specified in your PATH.

3.4 Multicore configuration

By default the model starts up with a single processor that starts executing from the entry point in the last provided ELF image, or 0 if no ELF images are provided.

The model can be configured using `--cores=N` to have up to four processors. Each processor starts executing the same set of images, starting at the same address. The `--visualization` command line option used with the multicore option results in a visualization window as shown in Figure 3-2.

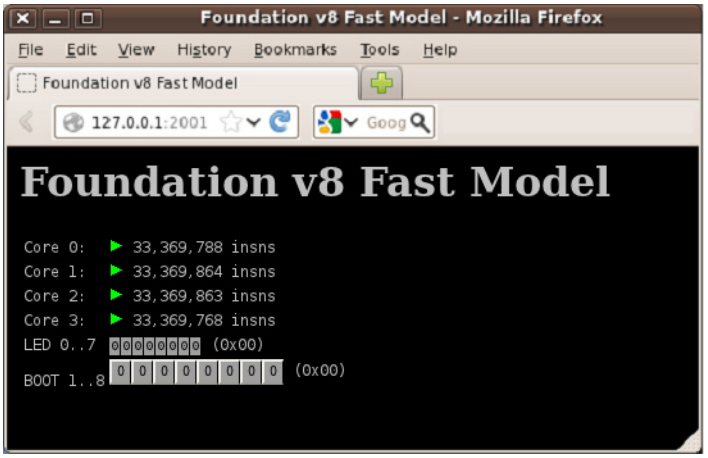

Figure 3-2 Multicore option with number of cores = 4

3.5 Memory and interrupt maps

This section describes the memory and interrupt maps for the ARMv8 Foundation Model.

3.5.1 Memory maps

The platform provides 36 bits of physical address space that can be configured either as a single address space, or separate secure and non-secure address spaces.

When the `--secure-memory` command line option is used, Secure memory accesses use the memory map in Table 3-3 and Non-secure memory accesses use the memory map in Table 3-4.

When the `--secure-memory` command line option is used, `--data` loads into secure memory, and `--nsdata` loads into non-secure memory.

When the `--no-secure-memory` command line option is used, both Secure and Non-secure accesses use the memory map in Table 3-4. There is no separate secure RAM.

Note

When the `--no-secure-memory` command line option is used, there is no RAM at address `0x0`.

Table 3-2 Private Peripheral Block

Address	Size	Device
`0x02C001000`	8KB	GIC distributor
`0x02C002000`	4KB	GIC processor interface
`0x02C004000`	4KB	GIC processor hypervisor interface
`0x02C005000`	4KB	GIC hypervisor interface
`0x02C006000`	8KB	GIC virtual processor interface

Table 3-3 Secure memory map

Address	Size	Device
`0x000000000`	128MB	RAM
`0x02C000000`	32KB	Private Peripheral Block

Table 3-4 Non-secure memory map

Address	Size	Device
`0x018000000`	64MB	Peripheral space, all read as zero or write ignored except
`0x01A000000`	16MB	Network (SMSC)
`0x01C000000`	64MB	Peripheral space, all read as zero or write ignored except
`0x01C010000`	64KB	System registers
`0x01C090000`	64KB	PL011_0
`0x01C0A0000`	64KB	PL011_1
`0x01C0B0000`	64KB	PL011_2

Table 3-4 Non-secure memory map (continued)

Address	Size	Device
0x01C0C0000	64KB	PL011_3
0x01c130000	64KB	Block storage (virtio mmio)
0x02C000000	32KB	Private Peripheral Block
0x080000000	2GB	RAM
0x880000000	6GB	RAM

Note

Accesses to unassigned peripheral spaces will print warnings to the console unless the `--quiet` command line option is used.

Accesses to all other unassigned addresses cause synchronous aborts in the memory system.

3.5.2 Interrupt maps

The platforms assigns the following SPIs and PPIs on the GIC.

Note

Shared Peripheral Interrupt (SPI) and *Private Peripheral Interrupt* (PPI) numbers are mapped onto GIC interrupt IDs as described in the *ARM Geneic Interrupt Controller Architecture Specification.*

Table 3-5 lists the SPIs:

Table 3-5 Shared Peripheral Interrupt map

SPI	Device
5	PL011_0
6	PL011_1
7	PL011_2
8	PL011_3
10	Virtio block device
15	Network

Table 3-6 lists the PPIs:

Table 3-6 Private Peripheral Interrupt map

PPI	Device
9	Virtual maintenance interrupt
10	Hypervisor timer event
11	Virtual timer event

Table 3-6 Private Peripheral Interrupt map (continued)

PPI	Device
12	nFIQ
13	Secure physical timer event
14	Non-secure physical timer event
15	nIRQ

3.5.3 System register block

The system register block provides a minimal set of registers:

Table 3-7 System register block

Offset	Type	Bits	Register
`0x0004`	R/W	[7:0]	User Programmable Switches
`0x0008`	R/W	[7:0]	LEDs
`0x00A0`	R/W	[31:0]	System configuration data
`0x00A4`	R/W	[31:0]	System configuration control
`0x00A8`	R/W	[31:0]	System configuration status

The User Programmable Switches store eight bits of state which may be read or written by software on the model. The startup value (`val`) may be configured using `--switches=val`. The switches can be viewed and set at run time from the web interface, as described in *Web interface* on page 3-4.

The LEDs store eight bits of state which can be read or written by software on the model and may be viewed at runtime from the web interface

The system configuration control register provides a single function. Writing the value `0xC0800000` stops the simulation and returns control to the command line.

Note

This might take a number of instructions to complete so a write to this register must be followed by a DSB and infinite loop.

The system configuration data and status registers always return 0 on reads, and writes are ignored.

3.6 Semi-hosting

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 out more information on semi-hosting in the *ARM Compiler Toolchain Developing Software for ARM Processors*.

The simulator handles semi-hosting by intercepting SVC `0x123456` or `0xAB` in AArch32, depending on whether the processor is in the ARM or Thumb state, and HLT `0xF000` in AArch64.

3.6.1 Semi-hosting configuration

The syntax to use on the command line to enable or disable semi-hosting is:

```
./Foundation_v8 --(no-)semihost
```

The syntax to use on the command line to set the semi-hosting command line string is:

```
./Foundation_v8 --semihost-cmd=<command string>
```