ARM® v8-A Foundation Model

Version: 2.0

User Guide

ARM®

ARM DUI 0677C (ID111413)

ARMv8-A 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.
13 November 2013	C	Non-Confidential	Memory maps added. Updated for Foundation v2 Model.

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-A Foundation Model User Guide

Preface

This preface introduces the *ARMv8-A Foundation Model v2*. It contains the following sections:

About this book

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

Intended audience

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

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for general information on the ARMv8-A 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-A Foundation Model.

Chapter 3 *Programmer's Reference*

Read this for technical information on the ARMv8-A 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).
- *Fixed Virtual Platform Reference v1.4* (ARM DUI 0575).
- *Fast Models User Guide v8.2* (ARM DUI 0370).
- *ARM® PrimeCell® Real Time Clock (PL031) Technical Reference Manual* (ARM DDI 0024).
- *ARM® Dual Timer Module (SP804) Technical Reference Manual* (ARM DDI 0271).
- *ARM® Watchdog Module (SP805) Technical Reference Manual* (ARM DDI 0270).
- *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* (ARM DDI 0487).

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 0677C.
- 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-A Foundation Model is an enabling platform for the ARMv8-A 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 3-11.

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-A 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.

It contains the following sections:

1.1 ARMv8 64-bit architecture overview

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

The ARMv8-A architecture extends the existing 32-bit architecture by introducing two execution states:

- A 32-bit AArch32 state.
- A 64-bit AArch64 state.

A number of changes have been made to the AArch32 functionality, but it remains compatible with the ARMv7-A architecture. Code conforming to AArch32 can run on ARMv7-A devices. The A32 (ARM) and T32 (Thumb) instruction sets have new instructions relative to the ARMv7-A *Instruction Set Architecture* (ISA).

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

AArch64 has four Exception Levels, 0-3, that replace the eight processor modes found in ARMv7-A.

The least privileged, EL0, is equivalent to user-mode.

EL1 is equivalent to kernel-mode.

EL2 is used for hypervisors.

The highest privilege level, EL3, is used for the TrustZone security monitor.

Like ARMv7-A, AArch32 includes 13 general registers, R0-12, the Program Counter, R15, and two 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). The new general purpose registers are all 64-bits wide to handle larger addresses, so 32-bit accesses use the lower halves of registers and either ignore or zero out the upper halves. The AArch32 registers map onto the lower halves of the AArch64 registers, and this 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 registers are set to zero.
- Condition flags, where set by the instruction, are computed from the lower 32 bits.

For more information about the ARMv8 Instruction Set, see the *ARMv8 Instruction Set Overview.*

For more information about the ARMv8-A architecture, see the *ARM Architecture Reference Manual*.

1.2 Software requirements

This section describes the host software required to run the ARMv8-A 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 higher, and libstdc++ 6.0.0, or higher, 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-A processor cluster model containing 1-4 cores 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, and optional GICv3 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, you require a host with at least 8GB of RAM.

  To simulate a system with 8GB of RAM, you require a host with at least 12GB of RAM.
- Four PL011 UARTs connected to xterms.
- Platform peripherals including a Real-time clock, Watchdog timer, Real-time timer, and Power controller.
- Secure peripherals including a Trusted watchdog, Random number generator, Non-volatile counters, and Root-key storage.
- 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 size prefetch buffer so requires correct barriers to be used in target code to operate correctly.

The model 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.

Figure 1-1 on page 1-5 shows a block diagram of the ARMv8-A Foundation Model.

The Foundation Model has been revised to support the ARM T*rusted Base System Architecture* (TBSA) and *Server Base System Architecture* (SBSA). A number of peripheral devices have been added, with corresponding changes to the memory map. It has also been updated to align more closely with peripherals present in the Versatile Express baseboard. Where appropriate, the original model is now referred to as *Foundation v1* and the new model is *Foundation v2*.

Software written to target the previous version of the model will work unmodified on the model using the default `--no-gicv3` configuration option. Only software that uses the early blocks of RAM is likely to require some adjustments.

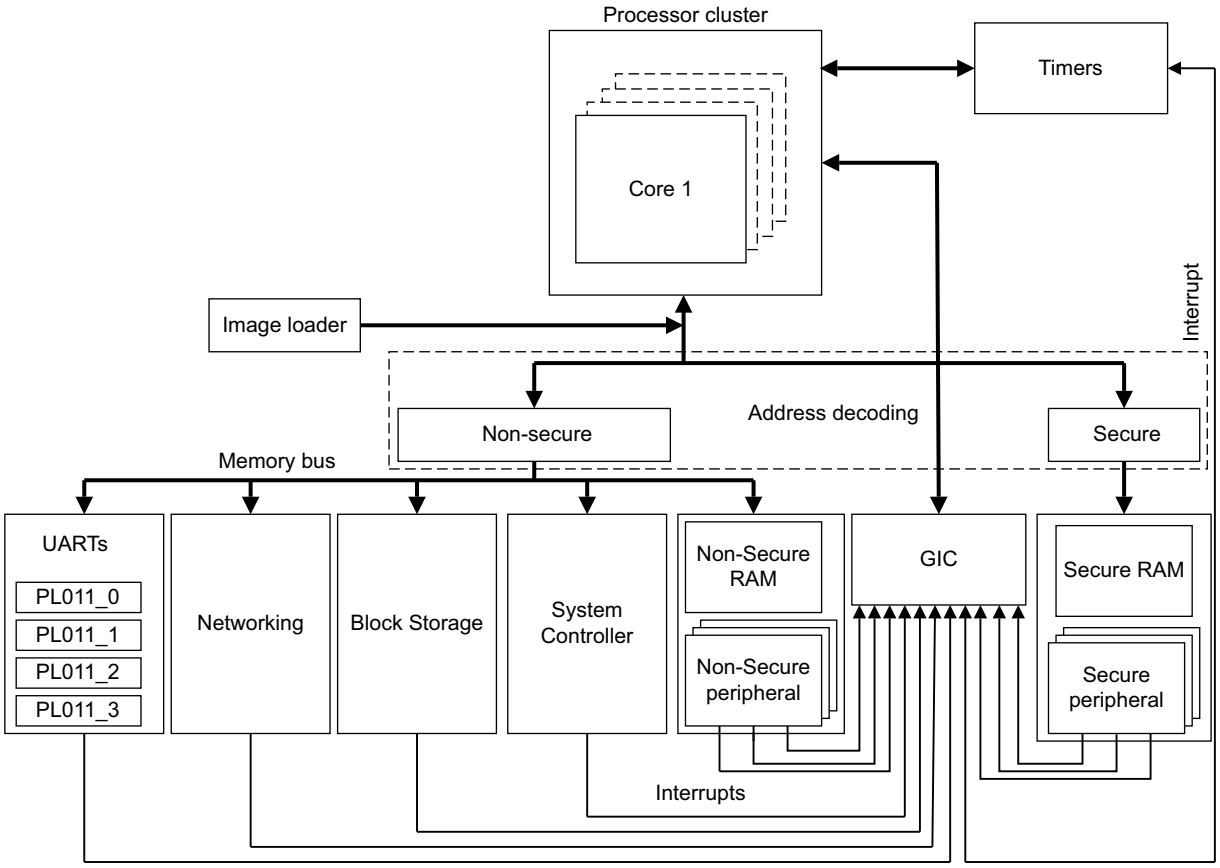

Figure 1-1 Block diagram of ARMv8-A Foundation Model

Note

The behavior of the address decoding block depends on whether the `--secure-memory` command line option is used. See *ARMv8-A Foundation Model memory map* on page 3-2.

The model provides the following types of network support:

NAT, IPv4 based 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 Bridged networking requires the setup of an ethernet bridge device to bridge between the ethernet port on the host and the network interface that the model provides. This usually requires administrator privileges. See the documentation in the Linux bridge-utils package for more information.

The ARMv8-A Foundation Model uses ARM Fast Model technology and forms 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.*

1.3.1 Limitations of the Foundation Model

The following restrictions apply to the ARMv8-A Foundation Model:

- Write buffers are not modelled.
- Interrupts are not taken at every instruction boundary.
- Caches are modelled 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-A Foundation Model. It contains the following sections:

2.1 Verifying the installation

The Foundation Model is available only as a prebuilt platform binary. The hierarchy that Figure 2-1 shows is used.

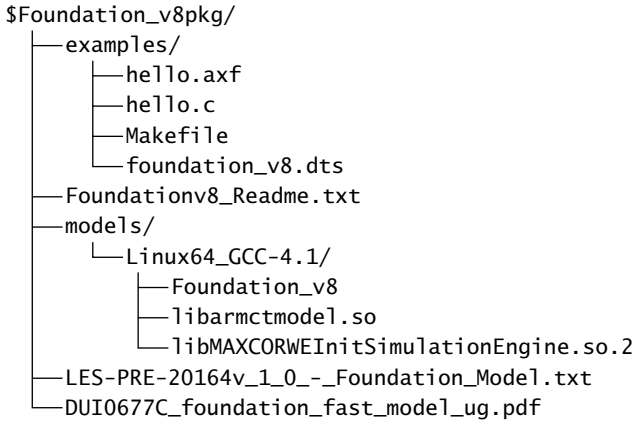

Figure 2-1 Installed files

The installation directory contents are:

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

`FoundationV8_Readme.txt`

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

`Foundation_v8`

The ARMv8-A 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.

`DUI0677C_foundation_fast_model_ug.pdf`

This document.

2.2 Running the example program

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

Run the model with the following command line:

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

Add `--quiet` to suppress everything except for the output from the example program.

It should print output similar to the following:

```
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.

2.2.1 Troubleshooting the example program

This section describes the common error messages you might encounter when running the example program.

- If you attempt to run the example program on a 32-bit Linux host, it gives an error similar to the following:

  ```
  ./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 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 get 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` are normally 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-A Foundation Model, see, http://www.linaro.org/engineering/armv8.

Chapter 3
Programmer's Reference

This chapter contains reference information. It contains the following sections:

3.1 ARMv8-A Foundation Model memory map

This section describes the memory map for the ARMv8-A Foundation Model.

Table 3-1 shows the Secure and Non-secure access permissions enabled through the use of the `--(no)-secure-memory` parameter that *Command line overview* on page 3-9 describes.

Table 3-1 Access permissions

	`--no-secure-memory`	`--secure-memory`
S	Secure and Non-secure access permitted	Secure access permitted, Non-secure access aborts
S/NS	Secure and Non-secure access permitted	Secure and Non-secure access permitted

Table 3-2 shows the global memory map for the ARMv8-A Foundation Model. This map is based on the Versatile Express RS2 memory map with extensions.

Note

- Areas of memory highlighted in the table return a warning to the console, together with RAZ/WI access behavior if accessed in the Foundation v2 Model, unless you use the `--quiet` command line option.
- Writes are ignored.
- Accesses from Foundation v1 Model cause an Abort exception.

Note

The Security (v2 only) column in Table 3-2 applies to the Foundation v2 Model only.

Table 3-2 ARMv8-A Foundation Model memory map

Start address	End address	Foundation v1 peripheral	Foundation v2 peripheral	Size	Security (v2 only)
0x00_0000_0000	0x00_03FF_FFFF	RAM	Trusted Boot ROM, secureflash	64MB	S
0x00_0400_0000	0x00_0403_FFFF		Trusted SRAM	256KB	S
0x00_0600_0000	0x00_07FF_FFFF		Trusted DRAM	32MB	S
0x00_0800_0000	0x00_0BFF_FFFF	-	NOR flash, flash0	64MB	S/NS
0x00_0C00_0000	0x00_0FFF_FFFF	-	NOR flash, flash1	64MB	S/NS
0x00_1800_0000	0x00_19FF_FFFF	-	Warning + RAZ/WI		
0x00_1A00_0000	0x00_1AFF_FFFF	Ethernet, SMSC 91C111	Ethernet, SMSC 91C111	16MB	S/NS
0x00_1C01_0000	0x00_1C01_FFFF	System Registers	System Registers	64KB	S/NS
0x00_1C02_0000	0x00_1C02_FFFF	-	System Controller, SP810	64KB	S/NS
0x00_1C04_0000	0x00_1C07_FFFF	-	Warning + RAZ/WI		
0x00_1C09_0000	0x00_1C09_FFFF	UART0, PL011	UART0, PL011	64KB	S/NS
0x00_1C0A_0000	0x00_1C0A_FFFF	UART1, PL011	UART1, PL011	64KB	S/NS

Table 3-2 ARMv8-A Foundation Model memory map (continued)

Start address	End address	Foundation v1 peripheral	Foundation v2 peripheral	Size	Security (v2 only)
0x00_1C0B_0000	0x00_1C0B_FFFF	UART2, PL011	UART2, PL011	64KB	S/NS
0x00_1C0C_0000	0x00_1C0C_FFFF	UART3, PL011	UART3, PL011	64KB	S/NS
0x00_1C0D_0000	0x00_1C0D_FFFF	-	Warning + RAZ/WI	-	-
0x00_1C0F_0000	0x00_1C0F_FFFF	-	Watchdog, SP805	64KB	S/NS
0x00_1C10_0000	0x00_1C10_FFFF	-	Base Platform Power Controller	64KB	S/NS
0x00_1C11_0000	0x00_1C11_FFFF	-	Dual-Timer 0, SP804	64KB	S/NS
0x00_1C12_0000	0x00_1C12_FFFF	-	Dual-Timer 1, SP804	64KB	S/NS
0x00_1C13_0000	0x00_1C13_FFFF	Virtio block device	Virtio block device	64KB	S/NS
0x00_1C14_0000	0x00_1C16_FFFF	-	Warning + RAZ/W	-	-
0x00_1C17_0000	0x00_1C17_FFFF	-	Realtime Clock, PL031	64KB	S/NS
0x00_1C1A_0000	0x00_1FFF_FFFF	-	Warning + RAZ/W	-	-
0x00_1F00_0000	0x00_1F00_0FFF	-	Non-trusted ROM	4KB	S/NS
0x00_2A43_0000	0x00_2A43_FFFF	-	REFCLK CNTControl, Generic Timer	64KB	S
0x00_2A44_0000	0x00_2A44_FFFF	-	EL2 Generic Watchdog Control	64KB	S/NS
0x00_2A45_0000	0x00_2A45_FFFF	-	EL2 Generic Watchdog Refresh	64KB	S/NS
0x00_2A49_0000	0x00_2A49_FFFF	-	Trusted Watchdog, SP805	64KB	S
0x00_2A4A_0000	0x00_2A4A_FFFF	-	Warning + RAZ/W	-	-
0x00_2A80_0000	0x00_2A80_FFFF	-	REFCLK CNTRead, Generic Timer	64KB	S/NS
0x00_2A81_0000	0x00_2A81_FFFF	-	AP_REFCLK CNTCTL, Generic Timer	64KB	S
0x00_2A82_0000	0x00_2A82_FFFF	-	AP_REFCLK CNTBase0, Generic Timer	64KB	S
0x00_2A83_0000	0x00_2A83_FFFF	-	AP_REFCLK CNTBase1, Generic Timer	64KB	S/NS
0x00_2C00_0000	0x00_2C00_1FFF	-	GIC Physical CPU interface, GICC[b]	8KB	S/NS
0x00_2C00_1000	0x00_2C00_1FFF	GIC Distributor	GIC Distributor[a]	4KB	-
0x00_2C00_2000	0x00_2C00_2FFF	GIC Processor Interface	GIC Processor Interface[a]	4KB	-
0x00_2C00_4000	0x00_2C00_4FFF	GIC Processor Hyp Interface	GIC Processor Hyp Interface[a]	4KB	-
0x00_2C00_5000	0x00_2C00_5FFF	GIC Hyp Interface	GIC Hyp Interface[a]	4KB	-
0x00_2C00_6000	0x00_2C00_7FFF	GIC Virtual CPU Interface	GIC Virtual CPU Interface[a]	8KB	-
0x00_2C01_0000	0x00_2C01_0FFF	-	GIC Virtual Interface Control, GICH[b]	4KB	S/NS

Table 3-2 ARMv8-A Foundation Model memory map (continued)

Start address	End address	Foundation v1 peripheral	Foundation v2 peripheral	Size	Security (v2 only)
0x00_2C02_F000	0x00_2C03_0FFF	-	GIC Virtual CPU Interface, GICV[b]	8KB	S/NS
0x00_2C09_0000	0x00_2C09_FFFF	-	Warning + RAZ/W		
0x00_2E00_0000	0x00_2E00_FFFF	-	Non-trusted SRAM	64KB	S/NS
0x00_2F00_0000	0x00_2F00_FFFF	-	GICv3 Distributor GICD[b]	64KB	S/NS
0x00_2F02_0000	0x00_2F03_FFFF	-	GICv3 Distributor ITS[b]	128KB	S/NS
0x00_2F10_0000	0x00_2F1F_FFFF	-	GICv3 Distributor GICR[b]	1MB	S/NS
0x00_7FE6_0000	0x00_7FE6_0FFF	-	Trusted Random Number Generator	4KB	S
0x00_7FE7_0000	0x00_7FE7_0FFF	-	Trusted Non-volatile counters	4KB	S
0x00_7FE8_0000	0x00_7FE8_0FFF	-	Trusted Root-Key Storage	4KB	S
0x00_8000_0000	0x00_FFFF_FFFF	DRAM (0GB - 2GB)	DRAM (0GB - 2GB)	2GB	S/NS
0x08_8000_0000	0x09_FFFF_FFFF	DRAM (2GB - 8GB)	DRAM (2GB - 8GB)	6GB	S/NS

a. The Foundation v2 Model uses the GICv2 memory map by default, or if you use the `--no-gicv3` configuration parameter that *Command line overview* on page 3-9 describes.

b. The Foundation v2 Model uses the GICv3 memory map if you use the `--gicv3` configuration parameter that *Command line overview* on page 3-9 describes.

3.2 Interrupt maps

This section describes the SPIs and PPIs on the GIC that the platform assigns.

Note

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

Table 3-3 lists the SPI assignments.

Table 3-3 Shared peripheral interrupt assignments

IRQ ID	SPI offset	Device
32	0	Watchdog, SP805
34	2	Dual-Timer 0, SP804
35	3	Dual-Timer 1, SP804
36	4	Realtime Clock, PL031
37	5	UART0, PL011
38	6	UART1, PL011
39	7	UART2, PL011
40	8	UART3, PL011
41	9	MCI, PL180, MCIINTR0
47	15	Ethernet, SMSC 91C111
56	24	Trusted Watchdog, SP085
57	25	AP_REFCLK, Generic Timer, CNTPSIRQ
58	26	AP_REFCLK, Generic Timer, CNTPSIRQ1
59	27	EL2 Generic Watchdog WS0
60	28	EL2 Generic Watchdog WS1
74	42	Virtio block device
92	60	cpu0 PMUIRQ
93	61	cpu1 PMUIRQ
94	62	cpu2 PMUIRQ
95	63	cpu3 PMUIRQ

Table 3-4 shows the PPI assignments:

Table 3-4 Private Peripheral Interrupt map

PPI	Device
9	Virtual maintenance interrupt
10	Hypervisor timer event
11	Virtual timer event
13	Secure physical timer event
14	Non-secure physical timer event

3.3 System register block

The system register block that Table 3-5 shows provides a minimal set of registers.

Table 3-5 System register block

Offset	Type	Bits	Register
0x0000	R/O	[31:0]	System ID 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 System ID Register is divided into fields:

- ID[31:28] Revision.
  - 0x0 Foundation Model v2.
- ID[27:16] HBI board number.
  - 0x010 ARMv8-A Foundation Model, default.
  - 0x020 ARM Base Model FVP.
- ID[15:12] Build variant. The value depends on the following command line options:
  - 0x0 Variant A is the Foundation Model with the GICv2 legacy map, when the `--no-givc3` command line option is used. This is the default.
  - 0x1 Variant B is the Foundation Model with the GICv3 64kB memory map, when the `--gicv3` command line option is used.
- ID[11:8] Platform type:
  - 0x0 Board.
  - 0x1 Model, default.
  - 0x2 Emulator.
  - 0x3 Simulator.
  - 0x4 FPGA.
- ID[7:0] FPGA build.
  - — Not used.

The System ID register is not implemented in the Foundation v1 Model. All unimplemented registers in the Foundation v1 system register block return the value `0xDEADDEAD` on reads. You can use this to distinguish Foundation Model v1 from both Foundation Model v2+, and FVP VE Base platforms.

The user-programmable Switches store eight bits of state that can be read or written by software on the model. You can configure the startup value, `val`, using `--switches=val`.

You can view and set the switches at run time from the web interface, as *Web interface* on page 3-11 describes.

The LEDs store eight bits of state that software can read or write on the model and can 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

Writes to the system configuration register might take a number of instructions to complete, therefore 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.

3.4 Command line overview

Command line arguments provide all model configuration. Run the model with `--help` to obtain a summary of the available commands.

The syntax to use on the command line is:

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

Table 3-6 shows the options.

Table 3-6 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. The default is 1. See *Multicore configuration* on page 3-13.
`--bigendian`	Start processors in big endian mode.The default is little endian.
`--(no-)secure-memory`	Enable or disable separate Secure and Non-secure address spaces The default is disabled.
`--(no-)gicv3`	Enable GICv3 or legacy compatible GICv2 as interrupt controller The default is `--no-gicv3`, that is, GICv2 mode.
`--block-device=file`	Image file to use as persistent block storage.
`--read-only`	Mount block device image in read-only mode.
`--image=file`	*Executable and Linking Format* (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. The default is enabled. See *Semi-hosting* on page 3-14.
`--semihost-cmd=cmd`	A string used as the semi-hosting command line. See *Semi-hosting* on page 3-14.
`--uart-start-port=P`	Attempt to listen on a free TCP port in the range P to P+100 for each UART. The default is 5000.
`--network=(none|nat|bridged)`	Configure mode of network access. The default is none.
`--network-nat-subnet=S`	Subnet used for NAT networking.The default is 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. The default is 00:02:f7:ef:f6:74.
`--network-bridge=dev`	Bridged network device name. The default is ARM0.

Table 3-6 Command line options (continued)

`--switches=val`	Initial setting of switches in the system register block (default: 0).
`--(no-)visualization`	Starts a small web server to visualize model state. The default is disabled. See *Web interface* on page 3-11.
`--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 provided, 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. You can specify more than one `--image`, `--data` or `--nsdata` option.

3.5 Web interface

The syntax to use on the command line is as follows:

```
./Foundation_v8 --visualization
```

or

```
./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 3-12.

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

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

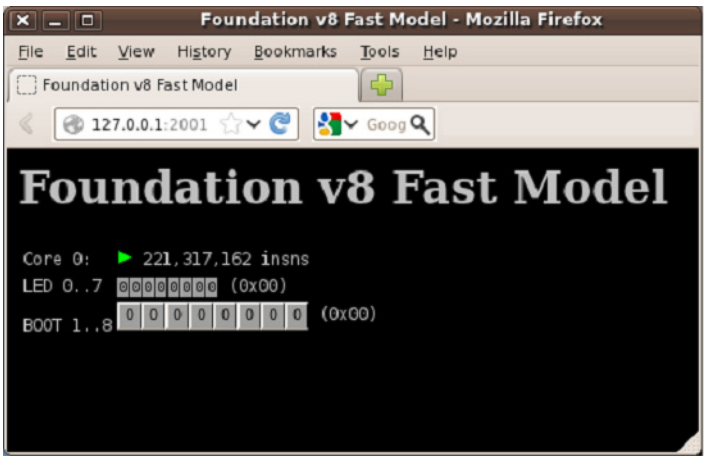

Figure 3-1 Visualization window

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.

3.6 UARTs

When the Foundation Model starts, it initializes four UARTs. For each UART, it searches 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 you can change it 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.

3.6.1 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.

3.7 Multicore configuration

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

You can configure the model using `--cores=N` to have up to four processor cores. Each core 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 Figure 3-2 shows.

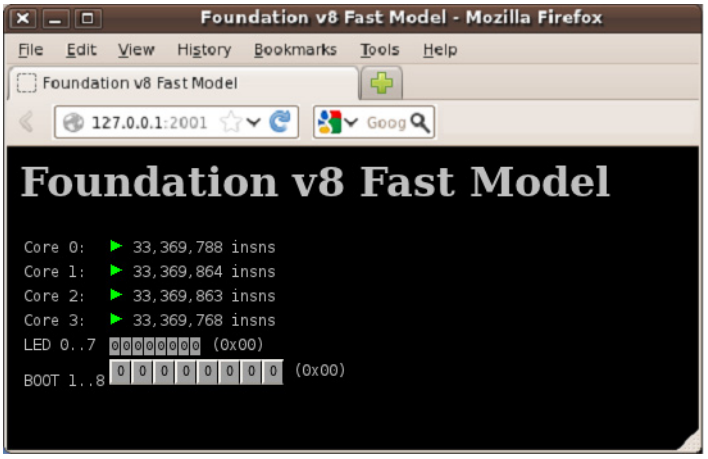

Figure 3-2 Multicore option with number of cores = 4

3.8 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. For more information on semi-hosting, see the *ARM Compiler Toolchain Developing Software for ARM Processors*.

The simulator handles semi-hosting by either:

- Intercepting SVC `0x123456` or `0xAB` in AArch32 depending on whether the processor is in the ARM or Thumb state.
- Intercepting HLT `0xF000` in AArch64.

3.8.1 Semi-hosting configuration

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

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

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

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