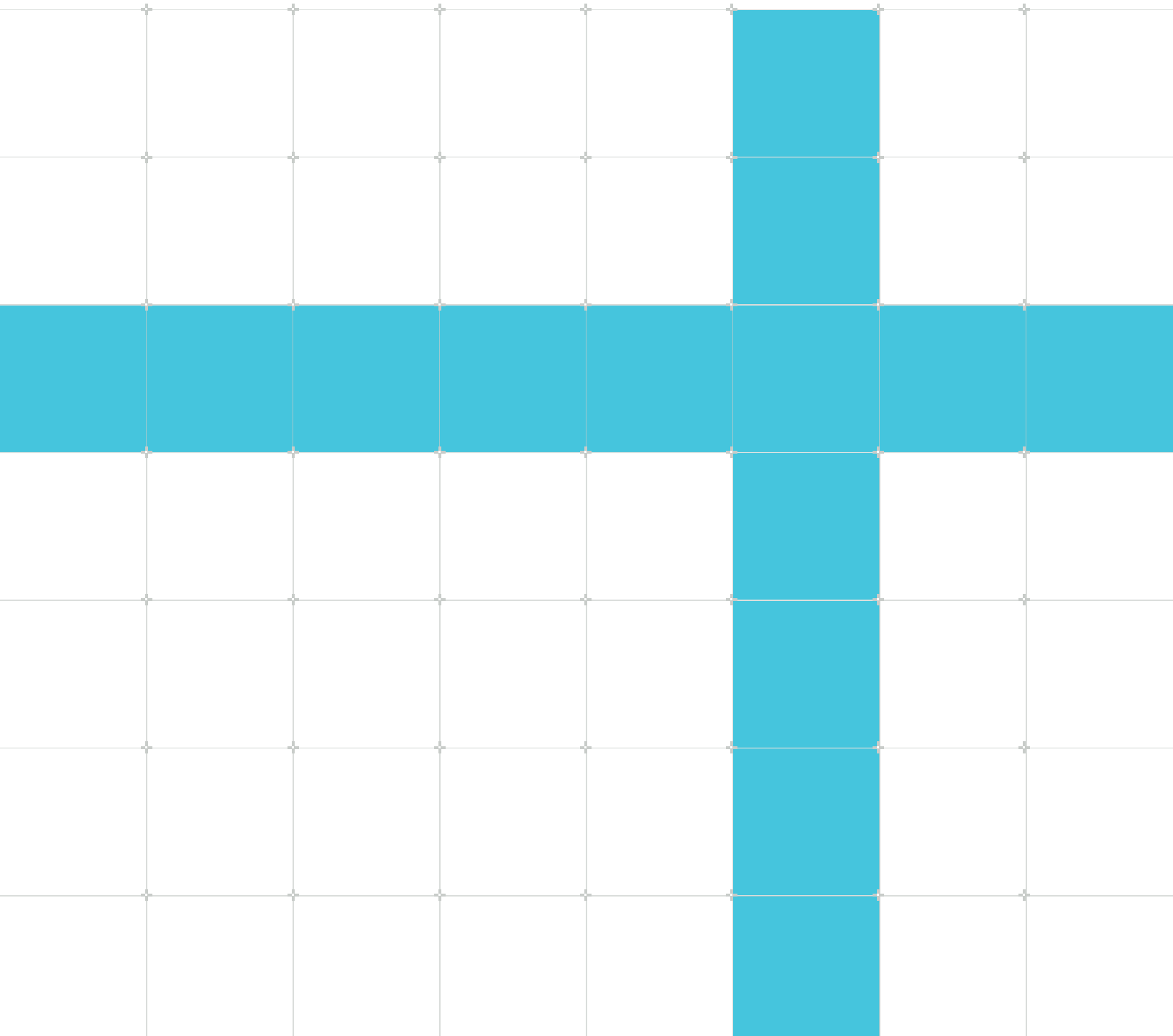
Fast Models

Version 11.31

User Guide

Non-Confidential

Issue 00
100965_1131_00_en



Fast Models User Guide

This document (100965_1131_00_en) was issued on 2026-03-04. There might be a later issue at https://developer.arm.com/documentation/100965

The product version is 11.31.

See also: Proprietary Notice | Product and document information | Useful resources

Start reading

If you prefer, you can skip to the start of the content.

Intended audience

This document is written for software developers using Fast Models to build and run custom platform models.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Contents

1. Introduction to Fast Models

This chapter provides a general introduction to Fast Models.

1.1 What is Fast Models?

The Fast Models product comprises a library of Programmer's View (PV) models and tools that enable partners to build, execute, and debug virtual platforms. Virtual platforms enable the development and validation of software without the need for target silicon. The same virtual platform can be used to represent the processor or processor subsystem in SoC validation.

Fast Models are delivered in two ways:

- As a portfolio of models of Arm® IP and tools to let you build a custom model of your exact system.
- As standalone models of complete Arm® platforms that run out-of-the-box to let you test your code on a generic system quickly. These pre-built platform models are built by Arm using Fast Models tools and components and are called Fixed Virtual Platforms, or FVPs.

The benefits of using Fast Models include:

Develop code without hardware

Fast Models provides early access to Arm® IP, well ahead of silicon being available. Virtual platforms are suitable for OS bring-up and for driver, firmware, and application development. They provide an early development platform for new Arm® technology and accelerate time-to-market.

High performance

Fast Models uses Code Translation (CT) processor models, which translate Arm® instructions into the instruction set of the host dynamically, and cache translated blocks of code. This and other optimization techniques, for instance temporal decoupling and Direct Memory Interface (DMI), produce fast simulation speeds for generated platforms, between 20-200 MIPS on a typical workstation, enabling an OS to boot in tens of seconds.

Customize to model your exact system

Fast Models provides a portfolio of models that are flexible and can easily be customized using parameters to test different configurations. You can model your own IP and integrate it with existing model components.

You can also export Fast Models components and subsystems to SystemC for use in a SystemC environment. Such an exported component is called an Exported Virtual Subsystem (EVS). EVSs are compliant with SystemC TLM 2.0 specifications to provide compatibility with Accellera SystemC and a range of commercial simulation solutions.

Run standalone or debug using development tools

Generated platform models support the Iris debug API. This allows them to be used standalone or with development tools such as Arm® Development Studio or Iris Monitor, as well as providing an API for third party tool developers.

Test architecture compliance

Fast Models provides Architecture Envelope Models (AEMs) for Arm®v8-A, Arm®v9-A, Arm®v8-R, and Arm®v8-M. These are specialist architectural models that are used by Arm and by Arm® architecture licensees to validate that implementations are compliant with the architecture definition.

Trace and debug interfaces

Fast Models provides Model Trace Interface (MTI) and the Iris API for trace and debug. These APIs enable you to write plug-ins to trace and debug software running on models. Fast Models also provides some pre-built MTI plug-ins, for example Tarmac Trace, that you can use to output trace information.

Build once, run anywhere

Since the same binary runs on the model, the target development hardware, and the final product, you only need to build it once, using the Arm® toolchain.

Host platform compatibility

Fast Models supports x86-64 host platforms running Linux or Microsoft Windows, and Arm® AArch64 hosts running Linux. For details, see 2.1 Requirements for Fast Models on page 13.

Related information

LISA+ Language for Fast Models Reference Guide
Iris Monitor

1.2 What does Fast Models consist of?

The Fast Models package contains the tools and model components that are needed to model a system.

It includes the following:

- Tools that enable you to create custom system models from the Fast Models component models, and debug them:

  System Generator or `simgen`

  Handles system model generation. It is invoked by using the `simgen` command-line utility. System models that are created using System Generator can be used with other Arm® development tools, for example Arm® Development Studio or Iris Monitor, or can be exported to SystemC for integration with proprietary models.

  Iris Monitor

  A browser-based debugger for Fast Models that lets you debug and view the state of the Fast Models components in a system.

- A library of component models of Arm® IP and protocols provided as LISA+ components. You can use them to create a system using the Fast Models tools. Ports and protocols are used for communication between components. Some models are of Arm® IP, for example:
  - Arm® Cortex® and Neoverse™ processors, and architectural models, called AEMs.
  - Arm® media IP such as GPUs, video processors, and display processors.
  - Peripherals, for instance Arm® CoreLink™ interconnect, interrupt controllers, and memory management units.

  Some models are abstract components that do not model Arm® IP, but are required by the software modeling environment. For example:
  - `PVBus` components to model bus communication between components.
  - Emulated I/O components to allow communication between the simulation and the host, such as a terminal, a visualization window, and an ethernet bridge.
- Platform model examples that show how to integrate the model components. They are supplied as project files, so must be built using System Generator (SimGen). Examples are provided for both standalone simulation and for SystemC export, and include:
  - Base Platform systems for Arm®v8-A, Arm®v8-R, and Arm®v9-A.
  - Systems based on Arm® Versatile™ Express development boards for Arm®v7-R processors.
  - Systems based on MPS2 development boards for Arm®v6-M, Arm®v7-M, and Arm®v8-M processors.
- Accellera SystemC and TLM header files and libraries, which are required to build FVPs and the platform model examples.
- Model Trace Interface (MTI) plug-ins. MTI is the interface used by Fast Models to emit trace events during execution of a program, for example branches, exceptions, and cache hits and misses. Fast Models provides some pre-built MTI plug-ins that you can use to capture trace data, without having to write your own. For example:
  - `TarmacTrace` can trace all processor activity or a subset of it, for instance only branch instructions or memory accesses.
  - `GenericTrace` allows you to trace any of the MTI trace sources that the models can produce.

  Some trace plug-ins are provided in source form as programming examples.
- ELF images that you can run on models for evaluation purposes. They include Dhrystone, which is required by some platform model examples.
- Networking setup scripts to bridge network traffic from the simulation to the host machine's network.

Related information

System Generator
Iris Monitor

1.3 Other Fast Models products

The following Fast Models products are available separately from the main Fast Models package:

Fixed Virtual Platforms (FVPs)

FVPs are models of Arm® platforms, including processors, memory, and peripherals.

Arm provides different types of FVP, based on the following platforms:

- Base Platform, for A-profile architectures.
- BaseR Platform for Arm®v8-R.
- Arm® Versatile™ Express development boards.
- Arm® MPS2 or Arm® MPS2+ platforms, for Cortex®-M series processors.

FVPs support the MTI and Iris interfaces, so can be used for debugging and for trace output. Some FVPs are supplied as pre-built executables in Arm Development Studio, see Arm Development Studio Getting Started Guide.

Arm provides validated Linux and Android deliverables for the AEM Base Platform FVP and for the Foundation Platform. For more information, see Getting Started with Linux or Android on Arm on Arm Developer and Armv-A Base RevC AEM FVP Platform Software User Guide in GitLab.

Foundation Platform

A simple FVP that includes an AEM that supports both Arm®v8-A and Armv9-A architectures, that is suitable for running bare-metal applications and for booting Linux. It is available for Linux hosts only and can be downloaded free of charge from Fixed Virtual Platforms on Arm Developer. Registration and login are required.

System Guidance platforms

These FVPs include documentation to guide SoC design and a reference software stack that is validated on the FVP. They are also known as Reference Design FVPs. For more information, see Reference Design on Arm Developer.

Third party IP

A package that contains source for all GPL or LGPL-licensed third party IP for Fast Models.

1.4 Fast Models glossary

This glossary defines some Arm-specific technical terms and acronyms that are used in the Fast Models documentation.

AMBA-PV

A set of classes and interfaces that model AMBA® buses. They are implemented as an extension to the TLM v2.0 standard.

See AMBA-PV extensions.

Architecture Envelope Model (AEM)

A fully-configurable, generic model of an Arm® architecture. It aims to expose software bugs by modeling the range of behavior that the architecture allows. Fast Models provides AEMs for Arm®v8-A, Arm®v8-R, Arm®v8-M, and Arm®v9-A. For example, AEMvACT models both Arm®v8-A and Arm®v9-A.

Auto-bridging

A Fast Models feature that SimGen uses to automatically convert between LISA+ protocols and their SystemC equivalents. It helps to automate the generation of SystemC wrappers for LISA+ subsystem models.


Note

Auto-bridging is deprecated and will be removed in a future release.

Base Platform

A range of example Fast Models platforms that can boot a full OS, including Linux and Android images that can be downloaded from Linaro. Base Platforms support the Arm®v8 and Arm®v9 architectures, replacing VE platforms, which support Arm®v7.

See Base Platform.

Component Architecture Debug Interface (CADI)

A legacy C++ debug interface that enables run control and inspection of models. It has been replaced by the Iris debug API.

See Introduction to the Component Architecture Debug Interface.

Code Translation (CT)

A technique that processor models use to enable fast execution of code. CT models translate code dynamically and cache translated code sequences to achieve fast simulation speeds.

Exported Virtual Subsystem (EVS)

A Fast Models component or subsystem that is built by SimGen as a SystemC shared library from a LISA+ model description. The library can be incorporated into a SystemC platform.

Fast Models

High performance software models of components of Arm® SoCs, for example processors, system IP, and bus infrastructure. Fast Models components can be connected together and configured to build a platform model, for example an FVP.

Fixed Virtual Platform (FVP)

A pre-built platform model composed of Fast Models components and built using Fast Models tools. FVPs enable applications and operating systems to be written and debugged without the need for real hardware.

See Fast Models FVPs in Arm Development Studio Reference Guide.

Foundation Model

See Foundation Platform.

Foundation Platform

A freely available, easy-to-use FVP for application developers that supports the Arm®v8-A and Arm®v9-A architectures. It can be downloaded from Fixed Virtual Platforms on Arm Developer, registration and login are required.

IMP DEF

Used in register descriptions in the *Fast Models Reference Guide* to indicate behavior that the architecture does not define. Short for Implementation Defined.

Integrated Simulator (ISIM)

A simulation executable generated by SimGen from a LISA+ model description. SimGen links the executable against a SystemC shared library.

See Building a SystemC ISIM target.

Iris

An interface for debugging and tracing model behavior. Iris is the replacement for CADI.

See Iris User Guide

Iris Monitor

A debugger for Fast Models which enables you to inspect the details and behavior of a Fast Models system.

See Iris Monitor.

Language for Instruction Set Architectures (LISA, LISA+)

LISA is a language that describes instruction set architectures. LISA+ is an extended form of LISA that supports peripheral modeling. LISA+ is used for creating and connecting model components. The Fast Models documentation does not always distinguish between the two terms, and sometimes uses LISA to mean both.

See LISA+ Language for Fast Models Reference Guide.

Microcontroller Prototyping System (MPS2)

Arm® Versatile™ Express V2M-MPS2 and V2M-MPS2+ are motherboards that enable software prototyping and development for Cortex®-M processors. The MPS2 FVP models a subset of the functionality of this hardware.

See MPS2.

Model Debugger

Legacy Fast Models debugger that enables you to execute, connect to, and debug any CADI-compliant model. You can run Model Debugger using a GUI or from the command line. It has been replaced by Iris Monitor.

Model Shell

Legacy command-line utility for configuring and running CADI-compliant models. It has been replaced by ISIM executables.

Model Trace Interface (MTI)

A trace interface that is used by Fast Models to expose real-time information from the model.

See Model Trace Interface Reference Manual.

Platform Model

A model of a development platform, for example an FVP.

Programmers' View (PV) Model

A high performance, functionally accurate model of a hardware platform. It can be used for booting an operating system and executing software, but not to provide hardware-accurate timing information.

PVBus

An abstract, programmers view model of the communication between components. Bus masters generate transactions over the PVBus and bus slaves fulfill them.

SimGen

An alternative name for System Generator.

Synchronous CADI (SCADI)

Legacy interface that provides a subset of CADI functions to synchronously read and write registers and memory. You can only call SCADI functions from the model thread itself, rather than from a debugger thread. SCADI is typically used from within MTI or by peripheral components to access the model state and to perform run control.

syncLevel

Each processor model has a syncLevel with four possible values. It determines when a synchronous watchpoint or an external peripheral breakpoint can stop the model, and the accuracy of the model state when it is stopped.

See syncLevel definitions.

System Canvas

Legacy GUI design tool for visualizing, configuring, and connecting the components of a platform model.

SystemC Virtual Platform (SVP)

A Fast Models platform that consists of components and subsystems that are individually exported to SystemC as a collection of multiple EVSs.

System Generator

A utility that generates a platform model by processing LISA files. You can run System Generator from the command line by invoking `simgen`. It is also referred to as SimGen.

See System Generator.

System Model

An alternative term for Platform Model.

Tarmac trace

A format for tracing the execution on code on an Arm® core. Fast Models includes the TarmacTrace plug-in which can consume and display tarmac trace.

See TarmacTrace.

Timing Annotation

A Fast Models feature that adds delays to transactions in the platform, enabling timing configuration for various operations, for instance branch prediction. It also supports configuring Cycles Per Instruction (CPI).

See Timing annotation.

Versatile™ Express (VE)

A family of Arm® hardware development boards targeting the Arm®v7 architecture. The term is abbreviated to VE when used in names of FVPs, for example, FVP_VE_Cortex-R4. For Arm®v8 and Arm®v9 support, VE platform models have been replaced by Base Platform models.

Related information

Arm Glossary

1.5 Security assumptions for Fast Models

The threat model for Fast Models is very permissive. We make the following assumptions about how you use Fast Models with respect to security.

- We expect you not to run Fast Models with elevated privileges, for example as root on Linux or administrator on Windows.
- If you build your own virtual platform using the Fast Models portfolio and distribute it within your own environment or to third parties, the integrity of the platform is your responsibility. For example it is up to you to ensure that the platform is not inadvertently modified.
- Fast Models does not provide a sandbox environment. We expect code running on the model to be able to trivially interact with the host environment for ease of use or during development, for example through semihosting. It is your responsibility to verify the integrity and validity of any code you run on the model.
- Similarly, plug-ins that you create are not isolated in any way. It is your responsibility to verify the integrity and validity of plug-ins that you use with a model, or any other code that you integrate with the model.

2. Installing Fast Models

This chapter describes the system requirements for Fast Models and how to install and uninstall Fast Models.

2.1 Requirements for Fast Models

This section describes the host hardware and software requirements for using Fast Models.

Host machine

Architecture

x86-64 and Arm® AArch64 host platforms are supported.

Minimum specification

At least 2GB RAM, preferably 4GB.

2GHz Intel Core2Duo, or similar, that supports the MMX, SSE, SSE2, SSE3, and SSSE3 instruction sets.

Recommended specification

At least double the RAM of the platform you intend to simulate. For example, a simulated platform containing 8GB of DRAM should be run on a 16GB host machine.

Fast Models and associated FVPs benefit most from high single-threaded performance. For example, a high frequency (4-5GHz) Intel Core i9 or i7 or AMD Ryzen 9 or 7 host CPU gives a significant improvement, between 30-60%, over Intel Xeon cores (2-3 GHz).

Python

Some platform models contain Arm Virtual Hardware Target (VHT) components. To operate these components correctly, Fast Models must be able to find a Python 3 library from Python 3.9 or later on the host system.

If platform software does not use the VHT components, no Python installation is required. The VHT component models ignore any accesses.

Windows

Ensure that the directory containing `python3.dll` is included in the `PATH` environment variable.

Linux

Ensure that the directory containing `libpython3.X.so` is included in the `LD_LIBRARY_PATH` environment variable, where `X` is the minor version of the Python installation.

Linux

Operating system

Red Hat Enterprise Linux 8 (for 64-bit architectures), Ubuntu 20.04 or 22.04 Long Term Support (LTS).

Shell

A shell compatible with `sh`, such as `bash` or `tcsh`.

Compiler

GCC 9.3.0 (x86-64 and Arm® AArch64 hosts), GCC 10.3.0 (x86-64 and Arm® AArch64 hosts), GCC 12.3.0 (x86-64 and Arm® AArch64 hosts).
The following table shows the supported GCC versions on a Linux x86 host:

Table 2-1: Supported GCC versions on Linux (x86 host)

OS	GCC versions supported
RHEL 8	GCC 9.3.0, GCC 10.3.0, GCC 12.3.0
Ubuntu 20.04 LTS	GCC 9.3.0
Ubuntu 22.04 LTS	GCC 10.3.0

The following table shows the supported GCC versions on a Linux Arm® AArch64 host:

Table 2-2: Supported GCC versions on Linux (Arm® AArch64 host)

OS	GCC versions supported
RHEL 8	GCC 9.3.0, GCC 10.3.0, GCC 12.3.0
Ubuntu 20.04 LTS	GCC 9.3.0
Ubuntu 22.04 LTS	GCC 10.3.0


Note

For full compatibility, it is highly recommended that all code that links against the Fast Models is compiled with C++17 support enabled. There are no known issues when linking non-C++17 code with the Fast Models. However, the compiler does not guarantee that the ABI is the same for both types of code. Compiling models with C++17 support disabled might fail.

The following combinations of GCC and GNU binutils were used to build Fast Models libraries:

Table 2-3: GCC and binutils versions

GCC version	GNU binutils version
9.3.0	2.32
10.3.0	2.38
12.3.0	2.38

PDF Reader

Adobe does not support Adobe Reader on Linux. Arm recommends system provided equivalents, such as Evince, instead.

Microsoft Windows

Operating system

Microsoft Windows 10 64-bit.

Compiler

Microsoft Visual Studio 2019 version 16.11 or later.
The following Visual Studio components are required:

- Visual C++ ATL for x86 and x64.
- Windows SDK version 10.0.16299.0 or later.

PDF Reader

Adobe Reader 8 or higher.


Note

- On Windows, Fast Models libraries are built with one of the following MSVC compiler options:
  - `/MD` for release builds
  - `/MDd` for debug builds

  Any objects or libraries that link against the Fast Models libraries must also be built with the same `/MD` or `/MDd` option.
- Fast Models does not support Express or Community editions of Visual Studio.

Licensing

Fast Models use either User-Based Licensing or FlexNet Licensing:

- Arm user-based licensing is only available to customers with a user-based licensing license. Documentation for user-based licensing is available at https://lm.arm.com. For assistance with user-based licensing issues, visit https://developer.arm.com/support and open a support case.
- For FlexNet Publisher node-locked or floating licensing, the latest version of the FlexNet software is available for download from License Server Management Software.


Note

- Set up a single `armlmd` license server. Spreading Fast Models license features over servers can cause feature denials.
- To run `armlmd` and `lmgrd` on Linux, install these libraries:

  Red Hat

  `lsb`, `lsb-linux`

  Ubuntu

  `lsb`

- If you use Microsoft Windows Remote Desktop (RDP) to access a simulation, your license type might restrict you:
  - Floating licenses require a license server, and have no RDP restrictions. Arm issues them on purchase.
  - Node locked licenses apply to specific workstations. Existing node locked licenses and evaluation licenses do not support running the product over RDP connections. Visit https://developer.arm.com/support for more information.

2.2 Install Fast Models

This topic provides installation instructions for Fast Models and optional add-on packages on Linux and Windows. It describes unpacking the package, setting up the environment, and verifying the setup.

Before you begin

Before starting the installation, ensure you have the following:

- A supported Linux or Windows host system, as described in 2.1 Requirements for Fast Models on page 13.
- Tools for extracting archives:

  Linux

  `tar` command-line utility or archive manager

  Windows

  File Explorer or a tool such as 7-Zip

Procedure

1. Log in to your account on Product Download Hub. If you have problems logging in, see the Product Download Hub getting started guide.
2. Search for **Fast Models**. If you have the necessary entitlement, select **Fast Models (FM000A)**.
3. Select the product revision to download from the drop-down list under **Select Revision to Download**. Click the **Download Now** button for your host architecture.
4. When the download has completed, unpack the Fast Models archive.
   In the following commands, replace the values in angle brackets with real values:

   Linux

   Use a terminal to extract the `.tar.gz` archive using the following command:

   ```
   bash
   tar -xzf FastModels_Portfolio_<version>_<build>_<os>_<arch>.tar.gz -C
    <install_path>
   ```


Note

If `-C <install_path>` is omitted, the archive is extracted into the current directory.

Windows

Extract the `.zip` archive using File Explorer or 7-Zip:

File Explorer:

a. Right-click the `.zip` file.

b. Choose **Extract All...**

c. Select the destination folder, for example `C:\Program Files\Arm\` and extract the archive.

7-Zip (command line):

If 7-Zip is installed and added to your system `PATH`, run the following command in a command prompt:

```
7z x FastModels_Portfolio_<version>_<build>_<os>_<arch>.zip -o"C:
\Program Files\Arm"
```


Note

The `-o` switch specifies the destination folder.

5. Configure the environment variables needed to use Fast Models.

Linux:

Source the environment setup script included with the package:

```
source <install_path>/FastModels_<version>_<build>/scripts/setup.sh
```

Windows:

Run the batch script included with the package:

```
call <install_path>\FastModels_<version>_<build>\scripts\setup.bat
```


Note

To apply these settings automatically at login, source the script in the profile for your shell.

You should see output similar to the following:

```
*****************************************
Environment configured successfully!
*****************************************
FASTMODELS_HOME set to /home/myname/ARM/FastModels_11.31_4
```

```
PVLIB_HOME set to /home/myname/ARM/FastModels_11.31_4
MAXCORE_HOME set to /home/myname/ARM/FastModels_11.31_4
SYSTEMC_HOME set to /home/myname/ARM/FastModels_11.31_4/SystemC
IRIS_HOME set to /home/myname/ARM/FastModels_11.31_4/Iris
PATH updated to include /home/myname/ARM/FastModels_11.31_4/bin
PYTHONPATH prefixed with /home/myname/ARM/FastModels_11.31_4/Iris/Python
```

Where:

FASTMODELS_HOME

Points to the root directory of the Fast Models installation

PVLIB_HOME

Points to the root directory of the Fast Models installation. This environment variable will be removed in a future release.

MAXCORE_HOME

Points to the root directory of the Fast Models installation. This environment variable will be removed in a future release.

SYSTEMC_HOME

Points to the Accellera SystemC library installation directory. The Fast Models package includes the SystemC and TLM header files and libraries that you need to build a Fast Models platform model.

IRIS_HOME

Points to the `$FASTMODELS_HOME/Iris/` directory, which contains Iris headers, examples, and the `iris.debug` Python module.

PYTHONPATH

This variable is updated to include `$IRIS_HOME/Python/`.

6. Optionally, install add-on packages.
   Fast Models add-on packages are also distributed as archives on Product Download Hub. Their installation process is similar to that of the main Fast Models package, but they do not require sourcing a setup script after extraction.

   Install add-ons into the same directory as the main Fast Models package. For example on Linux, use the following command, replacing the values in angle brackets with real values:

   ```
   tar -xzf FastModels_<package>_<name>_<version>_<build>_<arch>.tar.gz -C ~/Arm
   ```

   
   Note

   Ensure the destination folder, specified in this example by the `-C` option, matches the root of the main Fast Models package installation.

7. Verify the setup.
   After running the setup script, confirm that the environment variables have been correctly configured. For example, use the following command to print the SimGen tool version from the Fast Models installation:

   ```
   simgen --version
   ```

Next steps

For advanced configuration, modeling, and simulation guidance, see the Fast Models documentation included in the `Docs` directory or available from Arm Developer.

2.3 Uninstall Fast Models

On Windows and Linux, uninstall Fast Models by deleting the installation directories.

2.4 Dependencies for Red Hat Enterprise Linux

Fast Models requires some packages that are part of Red Hat Enterprise Linux, which you might need to install.

Some packages might depend on other packages. If you install with the Add/Remove software GUI tool or the `yum` command line tool, these dependencies resolve automatically. If you install packages directly using the `rpm` command, you must resolve these dependencies manually.

To display the package containing a library file on your installation, enter:

`rpm -qf` *`library_file`*

For example, to list the package containing `/lib/tls/libc.so.6`, enter the following on the command line:

```
rpm -qf /lib/tls/libc.so.6
```

The following output indicates that the library is in version `2.3.2-95.37` of the `glibc` package:

```
glibc-2.3.2-95.37
```

Table 2-4: Dependencies for Red Hat Enterprise Linux

Package	Required for
`alsa-lib`	Fast Models virtual platforms
`gcc-toolset-10`	Fast Models tools
`gcc-toolset-10-libatomic-devel.x86_64`	Fast Models tools
`glibc`	Fast Models tools and virtual platforms
`glibc-devel`	Fast Models tools
`libgcc`	Fast Models tools and virtual platforms
`libstdc++`	Fast Models tools and virtual platforms
`libstdc++-devel`	Fast Models tools
`libXext`	Fast Models tools and virtual platforms
`libX11`	Fast Models tools and virtual platforms

Package	Required for
libXau	Fast Models tools and virtual platforms
libxcb	Fast Models tools and virtual platforms
libSM	Fast Models tools and virtual platforms
libICE	Fast Models tools and virtual platforms
libuuid	Fast Models tools and virtual platforms
libXcursor	Fast Models tools and virtual platforms
libXfixes	Fast Models tools and virtual platforms
libXrender	Fast Models tools and virtual platforms
libXft	Fast Models tools and virtual platforms
libXrandr	Fast Models tools and virtual platforms
libXt	Fast Models tools and virtual platforms
make	Fast Models tools
telnet	Fast Models virtual platforms
xterm	Fast Models virtual platforms

3. Building Fast Models

This chapter describes how to use Fast Models tools to build model components and platforms.

It describes the following types of model:

- Integrated SIMulators (ISIMs)
- Exported Virtual Subsystem (EVS) components and platforms

The example command lines shown use a Linux host and GCC 9.3. Other hosts and GCC versions are also supported, see 2.1 Requirements for Fast Models on page 13 for details.

Fast Models includes source code for a range of example platforms, under `$PVLIB_HOME/examples/`. For instructions on building and running them, see the Fast Models Reference Guide:

- For an ISIM, see Build and run an FVP example.
- For an EVS platform, see Build and run an EVS platform example.

3.1 Build targets

Fast Models are built using a tool called System Generator, also called SimGen. To configure the build, SimGen uses a project file, with a `.sgproj` extension.

SimGen supports different build targets, which you specify in the `.sgproj` file. This chapter describes the following build targets:

Integrated SIMulator (ISIM)

A simulation executable generated by SimGen from a LISA+ model description. SimGen links the executable against a SystemC shared library.

The build target name for an ISIM is `TARGET_SYSTEMC_ISIM`.

Exported Virtual Subsystem (EVS)

An EVS can either be a component or a platform:

- An EVS component is a simulation shared library and associated C++ generated by SimGen from a LISA+ model description. You can incorporate the shared library into your own SystemC platform. SimGen links the shared library against a shared SystemC library but it can optionally be linked against a static SystemC library, by setting `USE_STATIC_SYSTEMC_LIB=1` in the `.sgproj` file.


Note

`USE_STATIC_SYSTEMC_LIB` is deprecated and will be removed in a future release. If you set it, the simulation executable must not instantiate more than one EVS. Otherwise the global variables in the static libraries are duplicated, which can cause errors or unpredictable behavior.

- An EVS platform is an example SystemC platform which provides reference code to demonstrate building a simulation executable by instantiating an EVS component in a hand-coded SystemC main (`sc_main()`).

The build target name for an EVS is `TARGET_SYSTEMC`.

For more information about SimGen, see Fast Models Tools User Guide.

3.2 ISIM build target

Build an ISIM by invoking SimGen on a `TARGET_SYSTEMC_ISIM` build target.

Select this build target by using the following statement in the `.sgproj` file's active configuration:

```
TARGET_SYSTEMC_ISIM = "1";
```

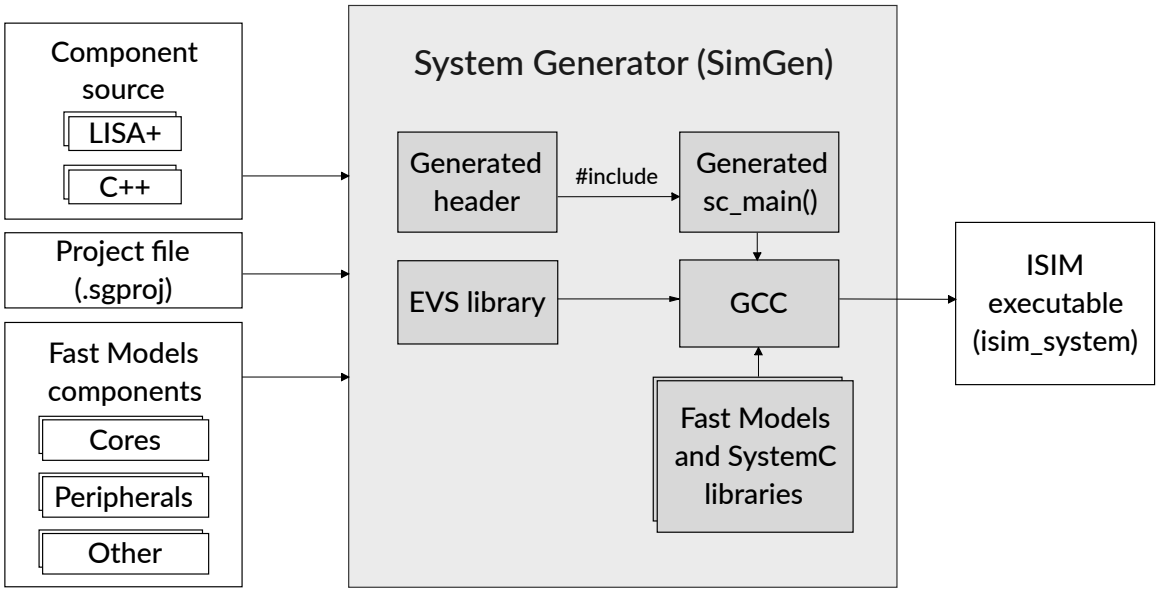
There is no default build target, so you must specify one or `simgen` returns an error.

For more information about `.sgproj` files, see:

- SimGen project (sgproj) file format and Project files in the Fast Models Tools User Guide
- The `.sgproj` files for the ISIM platform examples under `$PVLIB_HOME/examples/LISA/`.

The following diagram shows the process for building an ISIM. The shaded area represents the work that SimGen does for you:

Figure 3-1: Build process for an ISIM



SimGen takes as input the LISA+ or C++ source code for the platform and its components, and the `.sgproj` file.

It generates:

- An EVS library.
- The EVS header file, for example `./Linux64-Release-GCC-9.3/gen/scx_evs_<top_level_component>.h`, which defines the SystemC wrapper.
- A SystemC source file, `./Linux64-Release-GCC-9.3/gen/scx_main_system.cpp` which defines a default `sc_main()` function. This function is the entry point for the simulation. It initializes the simulation, constructs the SystemC wrapper, parses the command-line options, and starts the simulation.

SimGen then links the EVS library with the Fast Models and SystemC libraries, and outputs the executable, named `isim_system`.

Note

For information on how to build an ISIM platform example, see Build and run an FVP example in the Fast Models Reference Guide.

Related information

System Generator

3.3 EVS build target

SimGen builds an Exported Virtual Subsystem (EVS) as a single, shared library.

Select EVS as the build target by using the following statement in the `.sgproj` file's active configuration:

```
TARGET_SYSTEMC = "1";
```

There is no default build target, so you must specify one or SimGen returns an error.

For more information about `.sgproj` files, see:

- SimGen project (sgproj) file format and Project files in the Fast Models Tools User Guide
- The `.sgproj` files for the EVS examples under `$PVLIB_HOME/examples/SystemCExport/EVS_Components/`.

When building the EVS, it must link against the SystemC library. Select the library to link against in one of the following ways:

- Link against the dynamic SystemC shared library whose location is given by the `SYSTEMC_HOME` environment variable. This is the default.
- Link against a static SystemC shared library by setting the `.sgproj` configuration parameter `USE_STATIC_SYSTEMC_LIB`. By default, SimGen links against the library located in `SYSTEMC_HOME`.

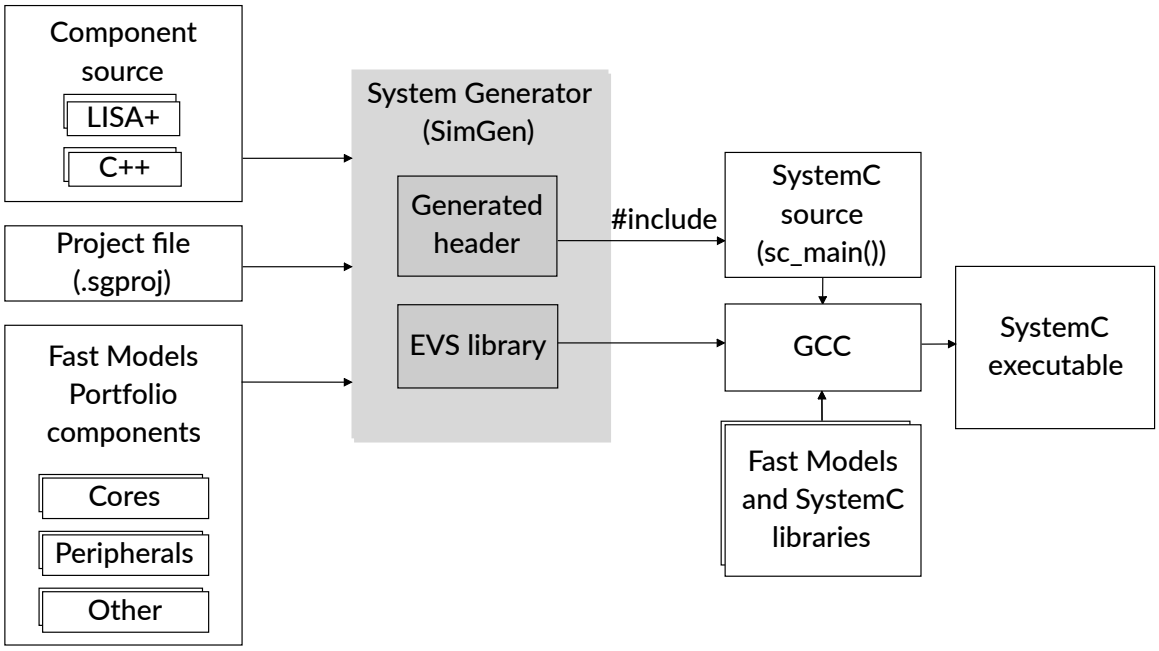
> **Note**
> `USE_STATIC_SYSTEMC_LIB` is deprecated and will be removed in a future release. If you set it, the simulation executable must not instantiate more than one EVS. Otherwise the global variables in the static libraries are duplicated, which can cause errors or unpredictable behavior.

- Link against either a static or dynamic SystemC library in a different location to `SYSTEMC_HOME`. To do this, set the `USER_SYSTEMC_DYNLIB` `.sgproj` configuration parameter to the full path of the library.

For more information, see 3.6 Linking against the SystemC library on page 27.

The following diagram shows the build process for an EVS. The shaded area represents SimGen and its output. The rest of the diagram is the responsibility of the user. Unlike an ISIM, an EVS platform must provide its own `sc_main()`:

Figure 3-2: Build process for an EVS



The Fast Models EVS platform examples are located in `$PVLIB_HOME/examples/SystemCExport/EVS_Platforms/`.

Their purpose is to demonstrate how to instantiate an EVS component, including:

- The generated header file to include, `./Linux64-Release-GCC-9.3/gen/scx_evs_<top_level_component>.h`
- The EVS shared library name to link against
- The component port names for the SystemC exported components

They are built using a Makefile whose target configuration has the pattern:

```
<release>_<compiler_version>_<arch_bits>
```

For instance, to build an example platform in release mode using a model library built with GCC 9.3, as a 64-bit binary, use this command:

```
make rel_gcc93_64
```

Or on Windows:

```
nmake rel_vs142_64
```

Note

For information on how to build an EVS platform example, see Build and run an EVS platform example in the Fast Models Reference Guide.

Related information

System Generator

3.4 SVP build target

An SVP (SystemC Virtual Platform) is a platform model that consists of LISA+ components or subsystems that are individually exported to SystemC as multiple EVSs, using the Multiple Instantiation (MI) feature.

The build process for an SVP is the same as for an EVS platform, except you must build and link multiple EVS libraries.

SVPs can provide more flexibility than EVS platforms because components in an SVP can be replaced without the need to modify any LISA+ code.

The Fast Models SVP examples are located in `$PVLIB_HOME/examples/SystemCExport/SVP_Platforms/`. For information on how to build an SVP example, see Build and run an SVP example in the Fast Models Reference Guide.

3.5 Building an EVS on Windows

When building an EVS on Windows, there are a few extra considerations to be aware of.

- On Windows, SimGen generates a solution containing several project files. For example, the following command outputs `Dhrystone.sln`, where `Dhrystone` is the name of the top-level component, and the following project files, which must be built in the order in which they are listed here:

```
"%MAXCORE_HOME%"\bin\simgen -b -p EVS_Dhrystone_Cortex-A710x1.sgproj --configuration Win64-Release-VC2019 …
```

Note

When invoking SimGen on Windows, you might need to use the `--devenv-path` option to specify the path to `devenv`.

1. `scx.vcxproj`. This project builds the static library `scx.lib` containing default implementations of the SystemC report handler, simulation controller, and scheduler.

2. `Dhrystone_scx_wrapper_Win64-Release-VC2019.vcxproj`. This project builds the dynamic library `Dhrystone-Win64-Release-VC2019.dll`, which is required when you launch the platform.
3. For an ISIM, an extra project is created, called `scx_isim_<top-component>_<config>.vcxproj`. This file is the SystemC project that creates the executable, `isim_system_<config>.exe`.

- On Windows, any SystemC code that instantiates an exported Fast Model must include `%PVLIB_HOME%\include\fmruntime\sg\IncludeMeFirst.h` before any other include files. If not, the compiler throws an error.

  This file is used to check the underlying Windows API version. It is automatically included in EVSs generated by SimGen, but SystemC models that use Fast Models libraries can be built without using SimGen.
- On Windows, Fast Models libraries are built with one of the following MSVC compiler options:
  - `/MD` for release builds.
  - `/MDd` for debug builds.

  Any objects or libraries that link against the Fast Models libraries must also be built with the same `/MD` or `/MDd` option.
- For a list of additional libraries that are needed when building the EVS, see `%PVLIB_HOME%\examples\SystemCExport\Common\nMakefile.common`.
- Use the `/vmg` compiler option to correctly compile source code for use with SystemC on Windows.

3.6 Linking against the SystemC library

To build an ISIM or EVS, you must have installed SystemC 2.3.4, including the integrated TLM 2.0.6, and set the `SYSTEMC_HOME` environment variable to its location.

Installing the main Fast Models package installs Accellera SystemC 2.3.4. To set the `SYSTEMC_HOME` environment variable:

- On Linux, source the environment setup script included with the package
- On Windows, run the batch script included with the package

By default, SimGen uses the Accellera SystemC dynamic library in `SYSTEMC_HOME`. To use a library stored in a different location, for example if `SYSTEMC_HOME` is not available, provide the path and filename of the library to SimGen in either of the following ways:

- Use the `.sgproj` configuration parameter `USER_SYSTEMC_DYNLIB`. For example, on Linux:

```
config "Linux64-Release-GCC-9.3"
{
    …
    USER_SYSTEMC_DYNLIB = "/path/to/libname_of_systemc.so";
}
```

On Windows:

```
config "Win64-Release-VC2019"
{
    …
    USER_SYSTEMC_DYNLIB = "C:\\path\\to\\import_library_name_of_systemc.lib";
}
```

- Use the SimGen command-line option `--override-config-parameter`. For example, on Linux:

```
--override-config-parameter USER_SYSTEMC_DYNLIB="/path/to/libname_of_systemc.so"
```

On Windows:

```
--override-config-parameter USER_SYSTEMC_DYNLIB="C:\\path\\to\
\import_library_name_of_systemc.lib"
```

- To link against a static SystemC library, set the `.sgproj` configuration parameter `USE_STATIC_SYSTEMC_LIB` to 1. The static library can either be located in the default location, `SYSTEMC_HOME`, or in a different location, defined by the parameter `USER_SYSTEMC_DYNLIB`.


Note

`USE_STATIC_SYSTEMC_LIB` is deprecated and will be removed in a future release. If you set it, the simulation executable must not instantiate more than one EVS. Otherwise the global variables in the static libraries are duplicated, which can cause errors or unpredictable behavior.

Related information

SimGen command-line options

3.7 Libraries required to run the platform

When you run a platform model, some shared libraries must be present in the same directory as the model, or a diagnostic message is given and the model might fail to run.

If you build a model using SimGen, it can copy the required shared libraries into the location of the generated model. If you then copy the model elsewhere, then you must also copy these shared libraries to the new location.

The list of libraries and executables evolves over time and can vary depending on the IP included in the platform, but might include the following:

- `armlm-ipc` or `armlm-ipc.exe`
- `arm_singleton_registry.so` on Linux or `arm_singleton_registry.dll` on Windows


Note

This is the singleton registry library, which enables multiple simultaneous simulations on the same host platform. If it is not located in the same directory as the executable, set the `FASTSIM_SINGLETON_REGISTRY` environment variable to the full path of the library. If the library is not found, a warning is reported. In

this case, a single simulation can still run, but multiple simultaneous simulations might lead to a crash.

- `libarmctmodel.so` or `armctmodel.dll`
- `libarmlm.so` or `armlm.dll`
- `libMAXCOREInitSimulationEngine.3.so` or `libMAXCOREInitSimulationEngine.3.dll`
- `libscxframework.so` or `scxframework.dll`
- `libSDL2-2.0.so.0.10.0` or `SDL2.dll`
- `newt.so` or `newt.dll`

Note

The libraries `libarmlm.so` or `armlm.dll` and `armlm-ipc` or `armlm-ipc.exe` are required by User Based Licensing (UBL). For more information, see the Knowledge Base Article How do I ensure my Fast Model works with User Based Licensing (UBL)?

4. Optimizing runtime performance of Fast Models

Fast Models platforms have many configuration options and parameters. The default ones are reasonable for most workloads on most platforms but there are a few you can change to make the model run as fast as possible.

4.1 Use a suitable host machine

The choice of processor and the amount of RAM available on the machine on which you run the Fast Model can significantly affect its performance.

The Fast Models simulation code runs primarily on a single thread. Arm FVPs and platforms built against the reference SystemC scheduler do not directly support multithreading. As a result, the single-threaded performance of the host machine matters much more than the number of processors it has. The choice of processor can influence model runtime by 1.5x, or more, so use the fastest possible single-threaded processor. As chip manufacturers make improvements, later generations of chips tend to be faster than previous ones.

The amount of memory that a Fast Model uses is unbounded. It allocates enough virtual memory to simulate the platform. If you add the `--stat` option to your FVP command line, on exit, as well as printing the performance statistics for the run, it prints the maximum amount of virtual memory that the platform used. Performance is greatly affected if this figure is greater than the physical memory available to the Fast Model process. We recommend host RAM to be at least 2.5x the amount of RAM the hosted workload uses.

4.2 Configure the model using options and parameters

Some command-line options and parameters should always be set to improve performance. Others should be configured depending on the workload.

These options are recommended:

- If you are targeting performance, always turn cache state modeling off. Running the model with cache state modeling on can slow down the model by more than an order of magnitude. See also Caches in PV models in the *Fast Models Reference Guide*.
- Platforms that support FastRAM always run faster with FastRAM enabled. For more information, see FastRAM in the Fast Models Reference Guide.
- Fast Models implements a large suite of trace sources. MTI trace plug-ins, for example TarmacTrace or GenericTrace, can subscribe to these trace sources, but this has an overhead as some trace fields, for example instruction disassembly, can be expensive to produce. For maximum speed, run the model with no trace plug-ins loaded.

- Some architectural features can be expensive to simulate and might not be required for development purposes. In such cases, a parameter might be present that treats the architectural feature as a NOP, which can improve performance. For example, to disable pointer authentication, use the `treat_PAC_as_NOP` parameter. To discover whether a feature can be treated as a NOP in this way, print a list of the available model parameters using the `-l` command-line option.
- Disable Memory Tagging Extension (MTE) if your workload allows it, or set `force_mte_tag_access_razwi_and_ignore_tag_checks` to true.
- To reduce overhead caused by the scheduler in a Fast Models simulation, use experimentation to tune the quantum and the minimum synchronization latency to your workload. In general, the longer the quantum, the faster the model runs, although this reduces latency which can result in lower performance. A PE can only see changes in the platform when it yields to the scheduler. So the quantum must be short enough that the PE does not spend time on work that will be superceded by other work happening in the platform. To configure the quantum and the minimum synchronization latency, use the options `--quantum` (`-Q` for short) and `--min-sync-latency` (`-M` for short).
- To significantly improve the performance of the Ethos models, set the `ethosu.extra_args="--fast"` parameter.

Related information

FVP command line options

4.3 Make the platform faster

If you can customize your platform, or are implementing your own components, use these techniques to make the platform run faster.

- For speed, it is essential that your platform uses DMI (Direct Memory Interface). Fast Models aggressively attempts to use DMI for all load and store operations. A DMI memory operation is often two orders of magnitude faster than a memory transaction that walks the bus to the memory device. If possible, all memory-like components should provide DMI to the Fast Model. Since DMI is so important, invalidation of DMI should be done carefully.
- The Fast Models portfolio contains an MMC component which is based on an old MMC standard and takes a lot of wall clock time to load a large file. If possible, use the Virtio model in the Fast Models portfolio for storage instead.

Related information

MMC component
VirtioP9Device

4.4 Make the workload faster

If possible, ensure your workload avoids busy loops.

Fast Models uses a cooperative scheduler so when a PE is running a busy loop, nothing else can run. The following techniques can avoid this problem:

- Busy loops waiting on time are faster if the GenericTimers and WFE/WFI are used instead. The Fast Model can advance time faster than real time if all PEs are in a WFI/WFE state, skipping over time when no instructions need to run.
- Busy loops waiting on DRAM memory to change should use the Exclusive loads and stores mechanism, so the core can go into WFE, yielding to the scheduler, and wait for the exclusive monitor to wake them.
- You should check busy loops that are part of peripheral initialization to make sure that the peripheral is modeled by the Fast Models platform. If Linux/Android has stalled for many minutes, it could be waiting on peripherals present in the OS device tree and memory map, but which are not modeled.

Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in the Arm documents.

Product status

All products and Services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
1131-00	4 March 2026	Non-Confidential	Update for v11.31.
1130-00	19 November 2025	Non-Confidential	Update for v11.30.
1129-00	16 May 2025	Non-Confidential	Update for v11.29.
1128-00	19 February 2025	Non-Confidential	Update for v11.28.
1127-00	16 September 2024	Non-Confidential	Update for v11.27.
1126-00	19 June 2024	Non-Confidential	Update for v11.26.

Issue	Date	Confidentiality	Change
1125-00	13 March 2024	Non-Confidential	Update for v11.25.
1124-00	6 December 2023	Non-Confidential	Update for v11.24.
1123-00	13 September 2023	Non-Confidential	Update for v11.23.
1122-00	14 June 2023	Non-Confidential	Update for v11.22.
1121-00	22 March 2023	Non-Confidential	Update for v11.21.
1120-00	7 December 2022	Non-Confidential	Update for v11.20.
1119-00	14 September 2022	Non-Confidential	Update for v11.19.
1118-00	15 June 2022	Non-Confidential	Update for v11.18.
1117-00	16 February 2022	Non-Confidential	Update for v11.17.
1116-00	6 October 2021	Non-Confidential	Update for v11.16.
1115-00	29 June 2021	Non-Confidential	Update for v11.15.
1114-00	17 March 2021	Non-Confidential	Update for v11.14.
1113-00	9 December 2020	Non-Confidential	Update for v11.13.
1112-00	22 September 2020	Non-Confidential	Update for v11.12.
1111-00	9 June 2020	Non-Confidential	Update for v11.11.
1110-00	12 March 2020	Non-Confidential	Update for v11.10.
1109-00	28 November 2019	Non-Confidential	Update for v11.9.

Issue	Date	Confidentiality	Change
1108-01	3 October 2019	Non-Confidential	Update for v11.8.1.
1108-00	5 September 2019	Non-Confidential	Update for v11.8.
1107-00	17 May 2019	Non-Confidential	Update for v11.7.
1106-00	26 February 2019	Non-Confidential	Update for v11.6.
1105-00	23 November 2018	Non-Confidential	Update for v11.5.
1104-01	17 August 2018	Non-Confidential	Update for v11.4.2.
1104-00	22 June 2018	Non-Confidential	Update for v11.4.
1103-00	23 February 2018	Non-Confidential	Update for v11.3.
1102-00	17 November 2017	Non-Confidential	Update for v11.2.
1101-00	31 August 2017	Non-Confidential	Update for v11.1.
1100-00	31 May 2017	Non-Confidential	Update for v11.0. Document numbering scheme has changed.

For information about the functional changes to Fast Models, see the Fast Models Release Notes.

Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

We recommend the following. If you do not follow these recommendations your system might not work.

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.

This information is important and needs your attention.

This information might help you perform a task in an easier, better, or faster way.

This information reminds you of something important relating to the current content.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the following tables provides direct access to the online version of the document.

Table 1: Arm publications

Document name	Document ID	Licensee only
Armv-A Base RevC AEM FVP Platform Software User Guide	-	No
Arm® Architecture Models	-	No
Download FlexNet Publisher	-	No
Fast Models Reference Guide	100964	No
Fixed Virtual Platforms	-	No
Fast Models FVPs Reference Guide	110379	No
Fast Models Tools User Guide	109415	No
Getting Started with Linux or Android on Arm	110597	No
How do I ensure my Fast Model works with User Based Licensing (UBL)?	ka005524	No
Iris User Guide	101196	No
IrisSupportLib Reference Guide	101319	No
LISA+ Language for Fast Models Reference Guide	101092	No
Product Download Hub	-	No
Product Download Hub getting started guide	107572	No
User-based Licensing User Guide	102516	No

Table 2: Arm publications

Document name	Document ID	Licensee only
Arm® Architecture Reference Manual for A-profile architecture	DDI 0487	Non-Confidential

Table 3: Other publications

Document ID	Organization	Document name
-	Accellera Systems Initiative	*Accellera Systems Initiative (ASI)*
-	Android Open Source Project	*Android Partitions*
IEEE 1666-2005	IEEE Standards association	*IEEE Standard SystemC(R) Language Reference Manual*