



Fast Models Tools

Version 11.26

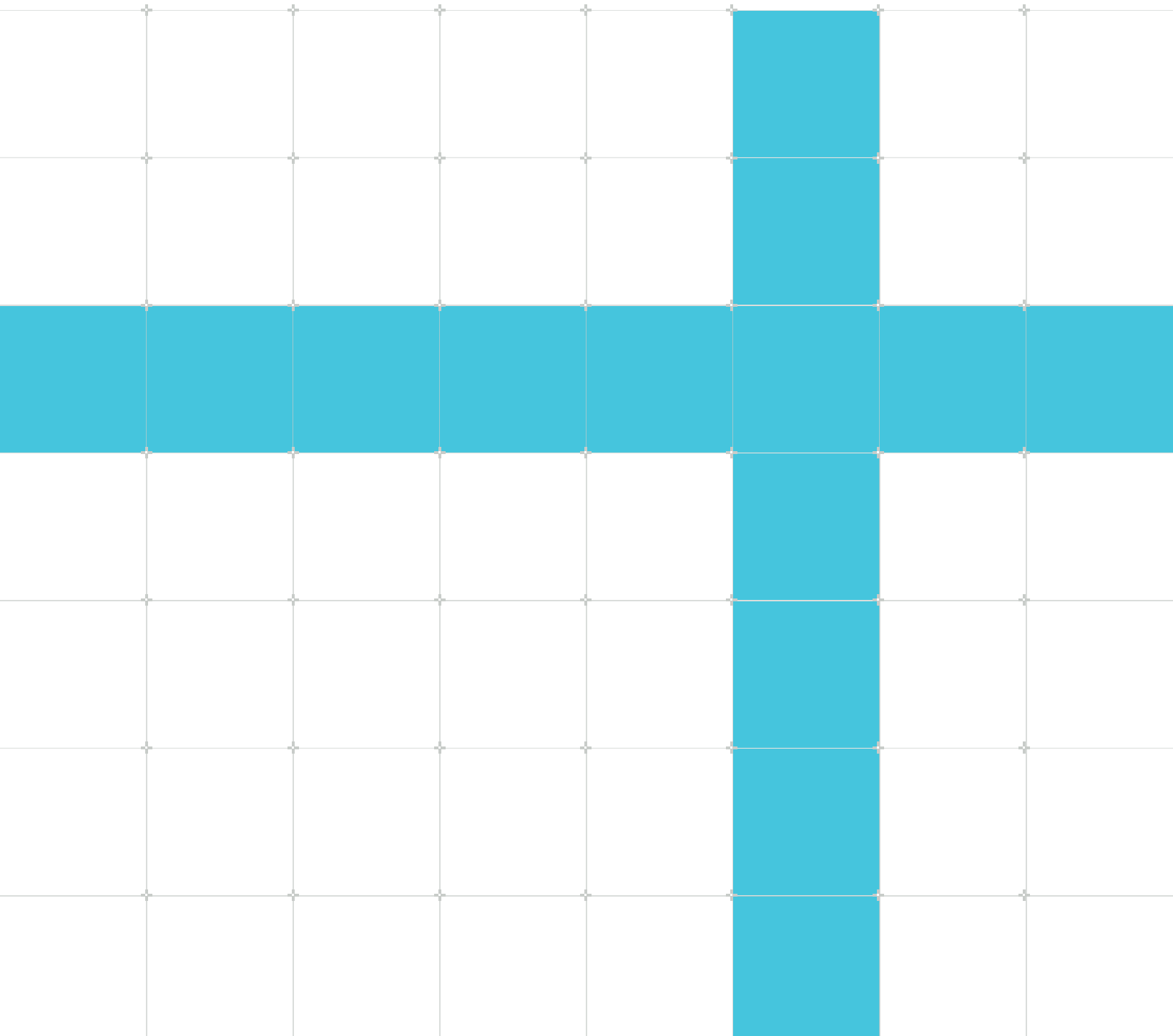
User Guide

Non-Confidential

Copyright © 2024 Arm Limited (or its affiliates).
All rights reserved.

Issue 00

109415_1126_00_en



Fast Models Tools User Guide

This document is Non-Confidential.

Copyright © 2024 Arm Limited (or its affiliates). All rights reserved.

This document is protected by copyright and other intellectual property rights.

Arm only permits use of this document if you have reviewed and accepted [Arm's Proprietary Notice](#) found at the end of this document.

This document (109415_1126_00_en) was issued on 2024-06-19. There might be a later issue at <http://developer.arm.com/documentation/109415>

The product version is 11.26.

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 the Fast Models tools to create, build, and debug custom Fast Models systems.

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. Fast Models tools.....	6
1.1 Fast Models design flow.....	6
1.2 Project files.....	8
1.3 Repository files.....	10
1.4 File processing order.....	11
1.5 Hierarchical systems.....	12
2. System Generator.....	15
2.1 SimGen command-line options.....	15
2.2 SimGen project (sgproj) file format.....	18
3. System Canvas.....	24
3.1 System Canvas tutorial.....	24
3.1.1 Starting System Canvas.....	24
3.1.2 Creating a new project.....	25
3.1.3 Adding the Arm® processor.....	28
3.1.4 Naming components.....	29
3.1.5 Resizing components.....	29
3.1.6 Hiding ports.....	29
3.1.7 Moving ports.....	30
3.1.8 Adding components.....	30
3.1.9 Using port arrays.....	31
3.1.10 Connecting components.....	32
3.1.11 View project properties and settings.....	32
3.1.12 Changing the address mapping.....	35
3.1.13 Building the system.....	37
3.1.14 Debugging with Model Debugger.....	38
3.1.15 Building a SystemC ISIM target.....	41
3.2 System Canvas GUI.....	42
3.2.1 Menu bar.....	43
3.2.2 Toolbar.....	50
3.2.3 Workspace window.....	52
3.2.4 Component window.....	55

3.2.5 Output window.....	56
3.2.6 System Canvas dialogs.....	57
4. Model Debugger.....	87
4.1 Key features.....	87
4.2 Retargetable debugger.....	88
4.3 Cluster debugging.....	88
4.4 Using Model Debugger.....	88
4.4.1 Launching Model Debugger from the command line.....	89
4.4.2 Launching Model Debugger from System Canvas.....	92
4.4.3 Configuring model parameters.....	92
4.4.4 Selecting target components.....	95
4.4.5 Loading an application.....	96
4.4.6 Connecting Model Debugger to a running ISIM.....	96
4.4.7 Starting a simulation within Model Debugger.....	97
4.4.8 Connect Model Debugger to a model running on another host.....	98
4.4.9 Using the cache view registers.....	99
4.4.10 Setting and removing breakpoints.....	100
4.4.11 Model Debugger sessions.....	104
4.5 Model Debugger GUI.....	104
4.5.1 Application windows.....	104
4.5.2 Debug views for source code and disassembly.....	120
4.5.3 Debug views for registers and memory.....	129
4.5.4 Debug views for pipelines.....	136
4.5.5 Watch window and Expression Evaluator.....	141
4.5.6 Breakpoint dialog boxes.....	145
4.5.7 Preferences dialog box.....	148
4.5.8 Keyboard shortcuts.....	150
5. Model Shell.....	152
5.1 ISIM targets.....	152
5.2 Model Shell command-line syntax.....	153
5.3 Model Shell command-line options.....	153
5.4 Configuration file syntax for specifying model parameters.....	155
5.5 SMP support.....	156
5.6 Manual Model Shell shutdown.....	157
5.7 Automatic Model Shell shutdown.....	157

5.8 License checking messages from Model Shell and ISIM systems..... 158

Proprietary Notice..... 159

Product and document information..... 161

Product status..... 161

Revision history..... 161

Conventions..... 161

Useful resources..... 164

1. Fast Models tools

Fast Models tools enable you to create custom system models from the Fast Models portfolio of component models, and debug them.

System Generator or `simgen`

A backend tool that handles system model generation. System Generator can either be invoked from the System Canvas GUI, or 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 Model Debugger, or can be exported to SystemC for integration with proprietary models.

System Canvas or `sgcanvas`

A GUI design tool for developing new model components written in LISA+ and for creating and building system models. To launch System Canvas from the command line, type `sgcanvas`. The GUI displays the model as either LISA+ source code, or graphically, in a block diagram editor.

Model Debugger

A symbolic debugger with a GUI that communicates with models using the Component Architecture Debug Interface (CADI). It enables you to launch a model or connect to a running model, and debug it.

Model Shell

A command-line tool for launching simulations that are implemented as CADI libraries. It can also run a CADI debug server to enable CADI-enabled debuggers to connect to the model.

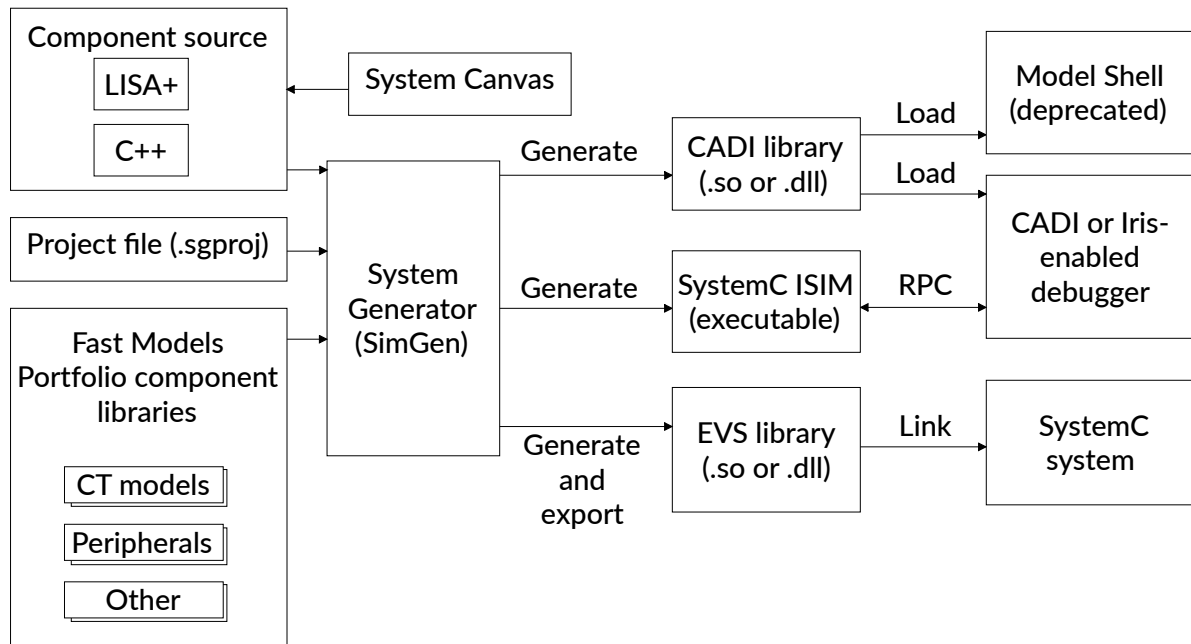


Arm deprecates Model Shell. Instead, we recommend you use Integrated SIMulators (ISIMs), which are standalone executables that do not require Model Shell.

1.1 Fast Models design flow

This section describes the process of designing a Fast Models system.

1. Create or license standard component models.
2. Use System Canvas to connect components and set parameters in the LISA+ source code.
3. Generate a new system model using System Generator either from the command line, using `simGen`, or using the System Canvas GUI.
4. Use the new model as input to a more complex system or distribute it as a standalone simulation environment.

Figure 1-1: Fast Models design flow

The inputs to System Generator are:

C++ library objects

Typically these are models of processors or standard peripherals.

LISA+ source code

The source code files define custom peripheral components. They can be existing files in the Fast Models portfolio or new LISA+ files that were created in System Canvas. The LISA+ descriptions can be located in any directory. One LISA+ file can contain one or more component descriptions.

Project file

System Generator requires a `.sgproj` project file to configure the build.

After the required components have been added and connected, System Generator uses gcc or the Visual Studio C++ compiler to produce the output object as one of the following:

- One or more CADI libraries, which you can load into Model Shell or Model Debugger.
- An ISIM executable, for instance an FVP. You could run this standalone, or you could connect a CADI-enabled debugger to it, such as Model Debugger or an Iris-enabled debugger such as Arm® Development Studio Debugger.
- An EVS, which can be used as a building block for a SystemC system. It is generated using the Fast Models SystemC Export feature.



To build ISIM executables or EVSs, you must have installed a SystemC environment, and set the `SYSTEMC_HOME` environment variable.

Related information

[SimGen project \(`sgproj`\) file format](#) on page 18

1.2 Project files

A single project file (`.sgproj`) describes to System Generator (SimGen) the build configuration to use for each host platform and the files that are required to build the model.

The build configuration includes:

- The compiler version to use.
- Whether to build release or debug binaries.
- Linker and compiler flags.
- SimGen flags.
- Build target, for example EVS library or ISIM.

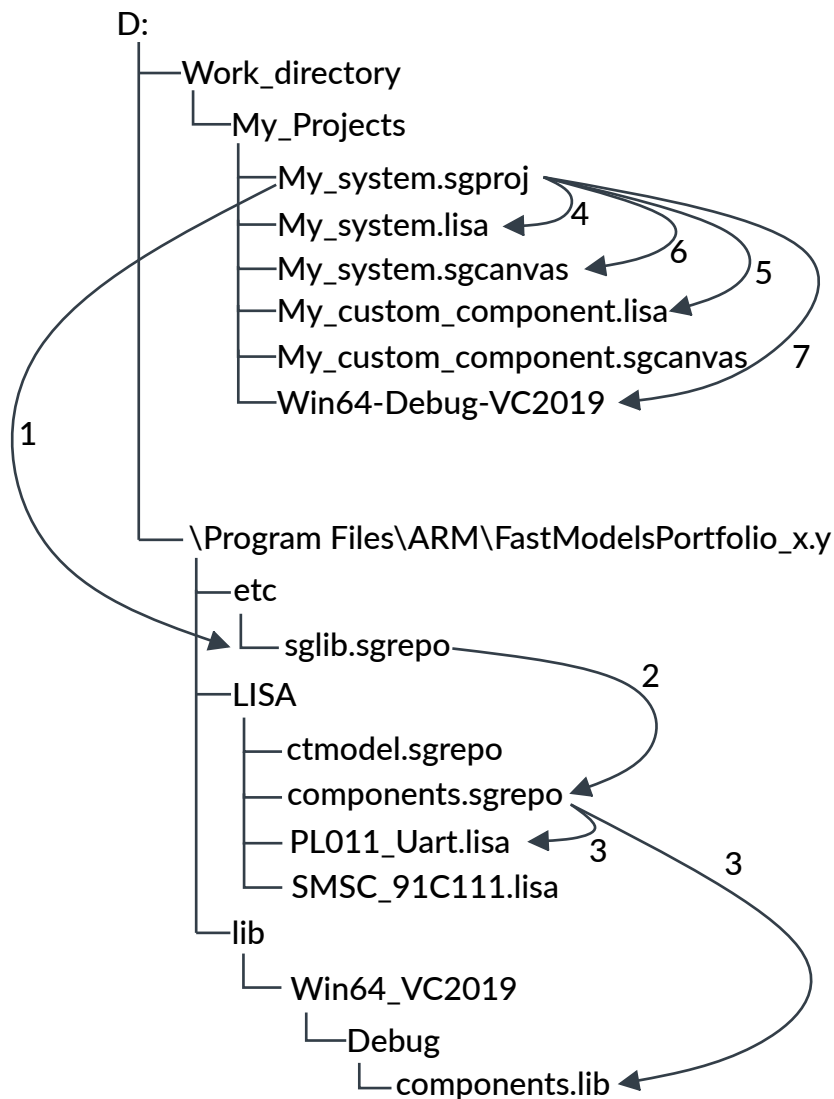
It also specifies the location of the LISA source files for the project.

There is no requirement to provide a makefile and a set of configuration files for each new project.

Each project file references all files that System Canvas or SimGen needs to build and run a simulation, including LISA, C, and C++ sources, libraries, files to deploy to the simulation directory, and nested repository files.

Repository files (`.sgrepo`) have the same format as project files.

You can add single files or a complete repository, such as the Fast Models Portfolio, to the project file.

Figure 1-2: Example organization of project directories and files on Microsoft Windows

The `My_Projects` directory contains the `My_System.sgproj` project file:

1. `My_system.sgproj` points to the standard Fast Models Portfolio repository file `sglib.sgrepo`.
2. The `sglib.sgrepo` repository file contains a list of repository locations such as `components.sgrepo`.
3. `components.sgrepo` lists the locations of the LISA files for the components and the location and type of libraries that are available for the components.
4. The project file lists `My_system.lisa` as the top-level LISA file for the system. The top-level LISA file lists the components in the system and shows how they interconnect.
5. This project uses a custom component in addition to the standard Fast Models Portfolio components. Custom components can exist anywhere in the directory structure. In this case,

only the `My_system` component uses the custom component, so the `My_custom_component.lisa` file is in the same directory.

6. System Canvas generates the `My_system.sgcanvas` and `My_custom_component.sgcanvas` files to save changes made to the display settings in the **Workspace** window. The display settings include:
 - Component location and size.
 - Label text, position and formatting.
 - Text font and size.
 - The moving of or hiding of ports.
 - Grid spacing.

The build process does not use `.sgcanvas` files. System Canvas uses them for its **Block Diagram** view.

7. `My_system.sgproj` defines `win64-Debug-vc2019` as the build directory for the selected platform. Other build options in the project file include:
 - The host platform, for instance "win64".
 - The compiler, for example "vc2019" and compiler options.
 - Additional linker options.
 - Additional options to be passed to `simGen`.
 - The type of target to build, for example an ISIM executable or a SystemC component.

Related information

[Project Settings dialog](#) on page 78

[SimGen project \(sgproj\) file format](#) on page 18

1.3 Repository files

Repository files group together references to commonly used files, eliminating the need to specify the path and library for each component in a project.

Repository files contain:

- A list of components.
- The paths to the LISA sources for the components.
- A list of library objects for the components.
- Optionally, lists of paths to other repository files. This enables a hierarchical structure.

System Canvas adds the default model repositories to a project when creating it. Changing these repository settings does not affect existing projects. The `project_name.sgproj` files contain the paths to the repositories as hard code. To change the repositories for an existing project, open the file and edit the paths.

Default repositories can also preset required configuration parameters for projects that rely on the default model library. These parameters are:

- Additional Include Directories.
- Additional Compiler Settings.
- Additional Linker Settings.

1.4 File processing order

The processing order enables a custom implementation of a Fast Models component.

An example of a project file

```
/// project file
sgproject "MyProject.sgproj"
{
  files
  {
    path = "./MyTopComponent.lisa";
    path = "./MySubComponent1.lisa";
    path = "./repository.sgrepo";
    path = "./MySubComponent2.lisa";
  }
}
```

An example of a repository file

```
/// subrepository file
sgproject "repository.sgrepo"
{
  files
  {
    path = "../LISA/ASubComponent1.lisa";
    path = "../LISA/ASubComponent2.lisa";
  }
}
```

System Canvas processes the files in sequence, expanding sub-repositories as it encounters them:

1. ./MyTopComponent.lisa
2. ./MySubComponent1.lisa
3. ./repository.sgrepo
 - a. ../LISA/ASubComponent1.lisa
 - b. ../LISA/ASubComponent2.lisa
4. ./MySubComponent2.lisa

Changing the processing order allows customization. If `MySubComponent1.lisa` and `../LISA/ASubComponent1.lisa` both list a component with the same name, the application uses only the first definition.

The **File List** view of System Canvas shows the order of components in the project file. Use the application controls to re-order the files and repositories:

- The **Up** and **Down** context menu entries in the **File List** view of the **Component** window. The commands have keyboard shortcuts of **Alt+Arrow Up** and **Alt+Arrow Down**.

You can also drag-and-drop files inside a repository or between repositories.

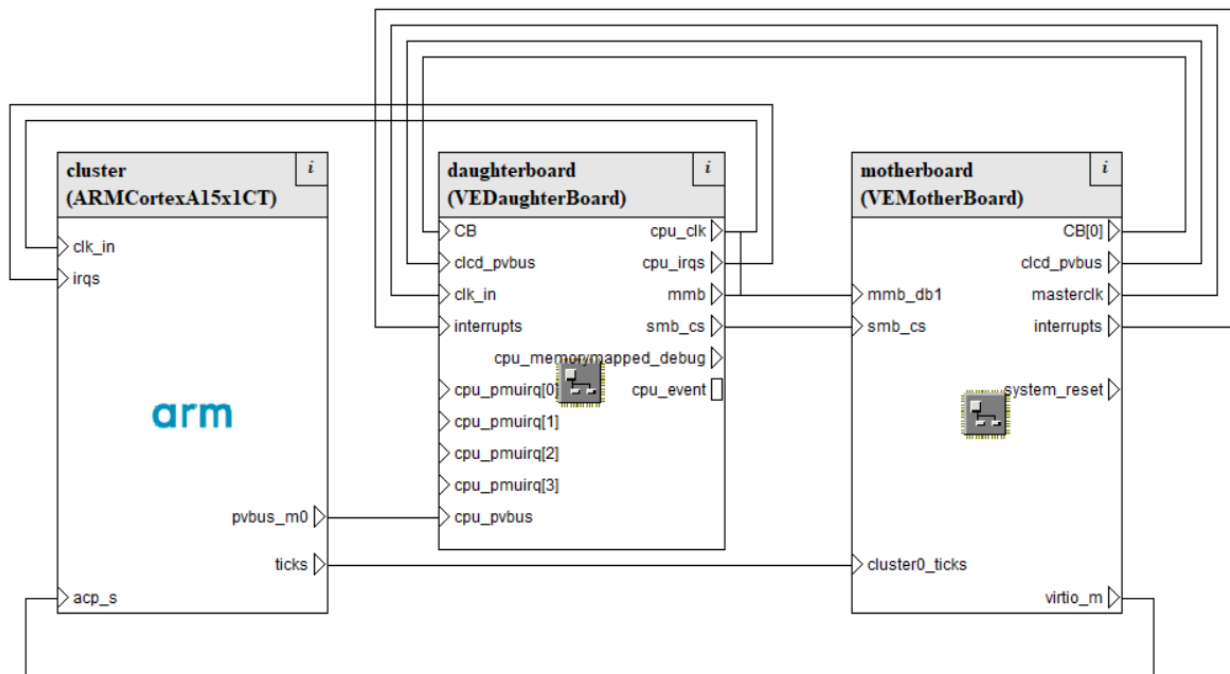
- The **Up** and **Down** buttons on the **Default Model Repository** tab in the **Properties** dialog, for repositories in new projects.

1.5 Hierarchical systems

The terms *system* and *component* are both used to describe the output from System Canvas. The main difference is whether the output is intended as a standalone system or is to be used within a larger system.

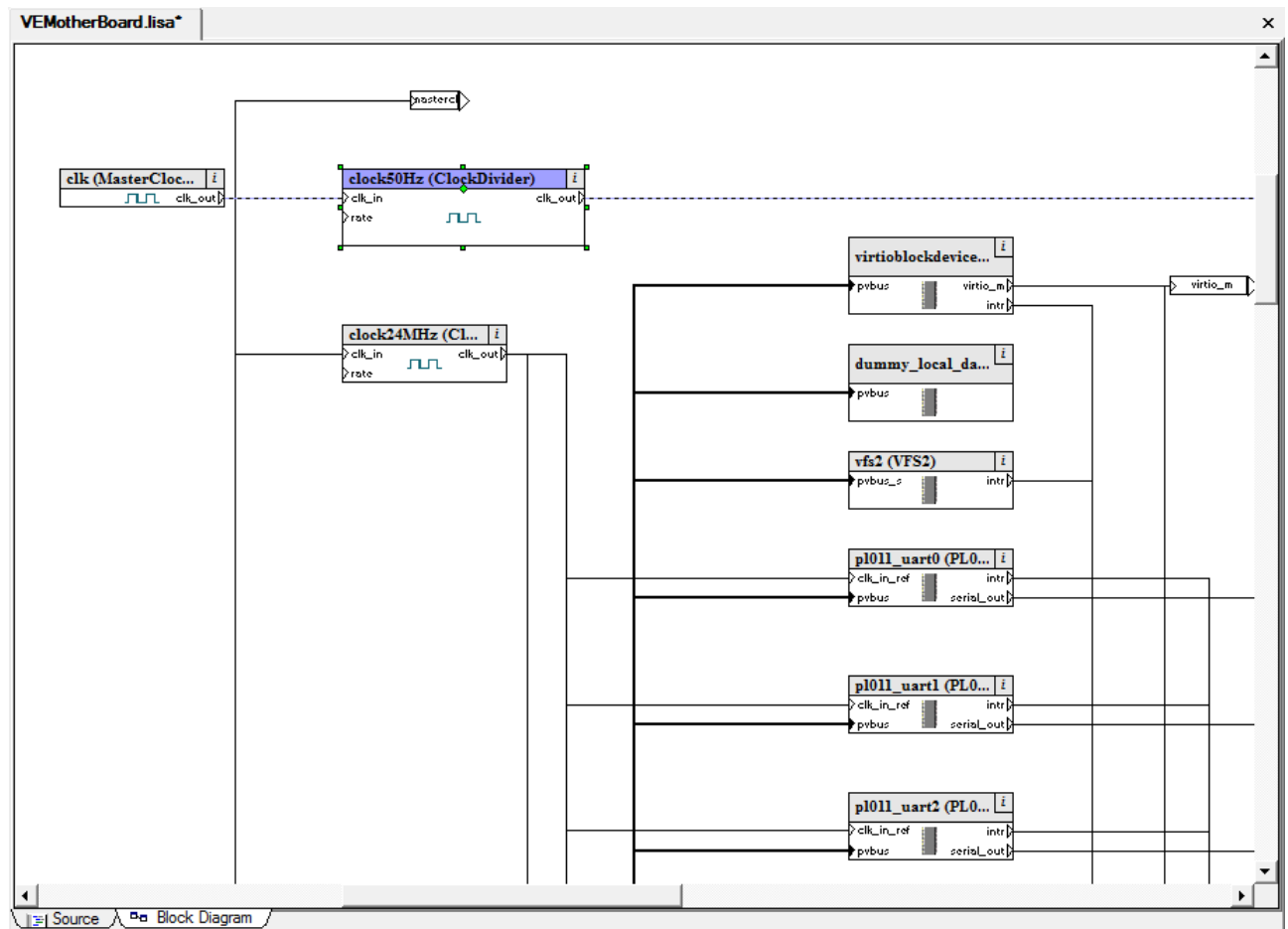
The block diagram shows the advantage of using a hierarchical system with a complex model.

Figure 1-3: Block diagram of top-level VE model



The main component in the system is a VE motherboard component. To open this item, select it and select **Open Component** from the **Object** menu. It is a complex object with many subcomponents.

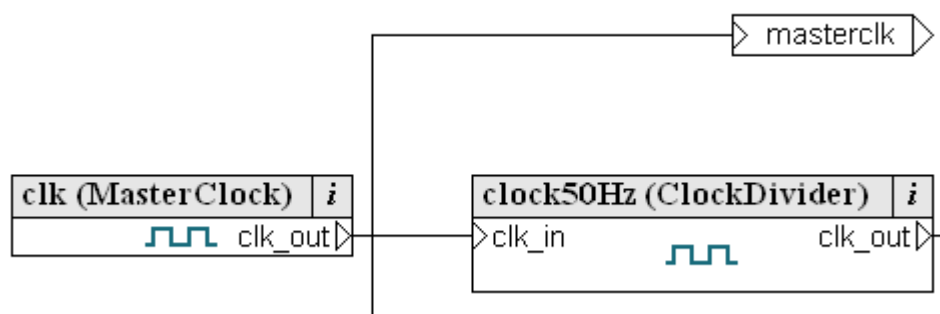
Figure 1-4: Contents of VE motherboard component



Hiding the complexity of the VE motherboard in a component simplifies the drawing and enables the VE motherboard component to be shared between different FVP models.

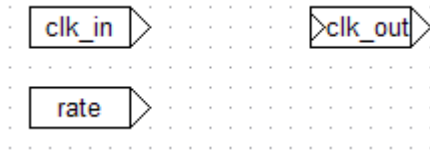
For example, the `clockDivider` component located at the top-left of [Figure 1-4: Contents of VE motherboard component](#) on page 13 has a connection to an external port called `masterclk`.

Figure 1-5: Self port detail



By double-clicking a component, in this case a clock divider, you can open it to see the LISA code, and the resulting Block Diagram window displays the external ports for that subcomponent.

Figure 1-6: Clock divider component external ports



The clock divider component contains only external ports, and it has no subcomponents. The behavior for this component is determined by the LISA code.

A component communicates with components in the higher-level system through its self ports. Self ports refer to ports in a system that are not part of a subcomponent, and are represented by a hollow rectangle with triangles to indicate data flow, and a text label in the rectangle.

Self ports can be internal or external.

Internal ports

These ports communicate with subcomponents and are not visible if the component is used in a higher-level system. Unlike hidden external ports, you cannot expose internal ports outside the subcomponent. Right-click on a port and select **Object Properties...** to identify or create internal ports. Set the port attributes to **Internal** for an internal self port.

External ports

These ports communicate with components in a higher-level system, and by default are external.

If you use the Block Diagram editor to make a connection between an external port and a subcomponent, the LISA code uses the keyword `self` to indicate the standalone port:

```
self.clk_in_master => clkdiv_ref25.clk_in;
```

2. System Generator

Platform models are built using a tool called System Generator, also called SimGen, and a Fast Models `.sgproj` project file.

You can use SimGen in the following ways:

- By building the project in System Canvas, which invokes SimGen indirectly.
- By directly invoking `simgen` on the command line.



To use SimGen, you must have installed a supported version of GCC or Visual Studio C++ compiler. On Windows, if SimGen cannot find `devenv.exe` for Visual Studio, the build fails. You can specify the path to `devenv.exe` in the System Canvas **Preferences** dialog or using the `--devenv-path` command-line option.

A SimGen command to build a Fast Models project might use the following options:

```
simgen --build --project-file <proj_file>.sgproj --configuration <config_name>
```

where `config_name` is the build configuration, for example `Linux64-Release-GCC-7.3`, `Win64-Debug-VC2019`, OR `Linux64_armv81-Release-GCC-9.3`.



A build configuration name beginning `Linux64_armv81` indicates an Arm® AArch64 Linux host.

2.1 SimGen command-line options

This table documents all System Generator (SimGen) options.

The command for invoking SimGen is:

```
simgen <options> [<additional-file-list>]
```

where:

<options>

SimGen command-line options.

<additional-file-list>

Optional, space-separated list of `.lisa` and `.sgrepo` files appended to the SimGen command line, in any order. `additional-file-list` enables you to build different variants of a platform using a single, common `.sgproj` file. List the files that are common to all variants in the `files`

section of the `.sgproj` file, and use `additional-file-list` to list the files that are specific to the platform variant.

Table 2-1: SimGen command-line options

Option	Short form	Description
<code>--allow-deprecated</code>	-	Allow the use of components marked as deprecated in the LISA+ component properties section.
<code>--allow-prerelease</code>	-	Suppress warning about pre-release model quality. If not specified, SimGen outputs a warning when it builds a model containing one or more pre-release <code>modelquality</code> components. Pre-release means the quality level is either preliminary support or alpha support. For more information about quality levels, see Quality level definitions in the Fast Models Reference Guide.
<code>--bridge-conf-file <FILENAME></code>	-	Set auto-bridging JSON configuration file <i>FILENAME</i> . For more information, see Auto-bridging in the Fast Models User Guide.
<code>--build</code>	<code>-b</code>	Build the targets.
<code>--build-directory <DIR></code>	-	Set build directory <i>DIR</i> .
<code>--clean</code>	<code>-C</code>	Clean the targets.
<code>--config <FILENAME></code>	-	Set SimGen configuration file <i>FILENAME</i> . By default, <code>simgen.conf</code> .
<code>--configuration <NAME></code>	-	The name of the build configuration, for example <code>Linux64-Release-GCC-9.3</code> .
<code>--cpp-flags-start</code>	-	Ignore all parameters between this and <code>--cpp-flags-end</code> , except <code>-D</code> and <code>-I</code> .
<code>--cpp-flags-end</code>	-	See <code>--cpp-flags-start</code> .
<code>--cxx-flags-start</code>	-	Ignore all parameters between this and <code>--cxx-flags-end</code> , except <code>-D</code> .
<code>--cxx-flags-end</code>	-	See <code>--cxx-flags-start</code> .
<code>--debug</code>	<code>-d</code>	Enable debug mode.
<code>--define <SYMBOL></code>	<code>-D</code>	Define preprocessor <i>SYMBOL</i> . You can also use <code>SYMBOL=DEF</code> .
<code>--devenv-path <ARG></code>	-	Windows only. Path to Visual Studio development environment, <code>devenv</code> .
<code>--disable-warning <NUM></code>	-	Disable warning number <i>NUM</i> . This overrides the <code>--warning-level</code> option.
<code>--dumb-term</code>	-	The terminal in which SimGen is running is dumb, so instead of fancy progress indicators, use simpler ones.
<code>--enable-warning <NUM></code>	-	Enable warning number <i>NUM</i> . This option overrides the <code>--warning-level</code> option.
<code>--gcc-path <PATH></code>	-	Linux only. Full path of the GCC C++ compiler to build the model. Ensure the compiler version matches the GCC version in the model configuration. By default, SimGen uses the g++ in the search path.
<code>--gen-sysgen-lib</code>	-	Generate system library.

Option	Short form	Description
--help	-h	Print help message with a list of command-line options then exit.
--ignore-compiler-version	-	Windows only. Do not stop on a compiler version mismatch. Try to build anyway.
--include <INC_PATH>	-I	Add include path <i>INC_PATH</i> .
--indir_tpl <DIR>	-	Set directory <i>DIR</i> where SimGen finds its template data files.
--link-against <LIBS>	-	Whether the final executable will be linked against debug or release libraries. <i>LIBS</i> can be either debug or release. This option performs some consistency checks.
--MSVC-debuginfo-type <ARG>	-	Set the type of debug information for MSVC projects. <i>ARG</i> can be one of: <ul style="list-style-type: none"> • none: No debug information. • /zi: Program database. • /Zd: Line numbers only.
--no-deploy	-	Prevent SimGen from copying deployed files from their original location to the location of the model. For example, when this option is used, SimGen does not copy <i>armctmodel.dll</i> or <i>libarmctmodel.so</i> from the model library to the location of the generated model. This option is for advanced users who are building models in a batch system, or as part of another tool where you are responsible for making sure all the required libraries are present.
--no-lineinfo	-c	Do not generate line number redirection in generated source and header files.
--num-build-cpus <NUM>	-	The number of host CPUs to use for the build.
--num-comps-file <NUM>	-	The number of components SimGen places into each C++ file. The default is one, which means SimGen places each component in a separate C++ file, along with generic code. A higher value reduces the total amount of generic code, which can reduce the compilation time when building a whole project. A lower value can reduce the compilation time if only some components have changed and need recompilation.
--outdir_arch <DIR>	-	Set output directory <i>DIR</i> for file with variable filenames.
--outdir_fixed <DIR>	-	Set output directory <i>DIR</i> for file with constant filenames.
--override-config-parameter <PARAM>=<VALUE>	-P	Override a configuration parameter defined in the <i>.sgproj</i> file. To override multiple parameters, specify this option multiple times. This option also supports appending a string to a parameter value defined in the <i>.sgproj</i> file, by using the syntax <i><PARAM>+=<STRING></i> . For example: <i>-P ADDITIONAL_COMPILER_SETTINGS+=-Wnew-warning"</i>
--print-config	-	Print the configuration parameters to file <i>.ConfigurationParameters.txt</i> .
--print-preprocessor-output	-E	Print preprocessor output, then exit.

Option	Short form	Description
<code>--print-resource-mapping</code>	-	Print flat resource mapping when generating a simulator.
<code>--project-file <FILENAME></code>	<code>-p</code>	Set SimGen project file <i>FILENAME</i> .
<code>--replace-strings</code>	-	Replace one or more strings in one or more files, then exit. Ignore binary files. Usage: <code>simgen --replace-strings FROM1 TO1 [FROM2 TO2 ...] -- FILE1 [FILE2 ...]</code>
<code>--replace-strings-bin</code>	-	Replace one or more strings in one or more files, then exit. Do not ignore binary files. Usage: <code>simgen --replace-strings-bin FROM1 TO1 [FROM2 TO2 ...] -- FILE1 [FILE2 ...]</code>
<code>--top-component <COMP></code>	-	Set the top-level component.
<code>--user-MSVC-libs-start</code>	-	Set additional libraries for MSVC projects. The list is terminated by <code>--user-MSVC-libs-end</code> .
<code>--user-MSVC-libs-end</code>	-	See <code>--user-MSVC-libs-start</code> .
<code>--user-sourcefiles-start</code>	-	Add source files listed between this option and <code>--user-sourcefiles-end</code> to the executable.
<code>--user-sourcefiles-end</code>	-	See <code>--user-sourcefiles-start</code> .
<code>--verbose <ARG></code>	<code>-v</code>	Set SimGen verbosity to either <i>on</i> , <i>sparse</i> (default), or <i>off</i> .
<code>--version</code>	<code>-V</code>	Print the version and exit.
<code>--warning-level <LEVEL></code>	<code>-w</code>	Set the warning level. Possible values are 0-3, where 0 means no warnings and 3 means all warnings. The default is 2, for all useful warnings.
<code>--warnings-as-errors</code>	-	Treat LISA parsing and compiler warnings as errors.

2.2 SimGen project (sgproj) file format

A SimGen project file, or `sgproj` file for short, is used to configure a Fast Models build. The filename has a `.sgproj` extension.

Specify the project file in one of the following ways:

- When using SimGen, use the `--project-file` option, or the short version `-p`.
- When using System Canvas, select **File > Load Project**.

Syntax

The `sgproj` file defines:

- A build configuration for each supported build target.
- The default configuration to use for Linux or Windows.
- A list of files that are used to build the platform.

In the following code block, items in angle brackets () are variables and ellipses (...) indicate elements that can be repeated. The variables are described after the code block: <>

```
# comment
sgproject "<filename>.sgproj"
{
  TOP_LEVEL_COMPONENT = "<top_level_component>";
  ACTIVE_CONFIG_LINUX = "<active_x86_64_linux_config>";
  ACTIVE_CONFIG_LINUX_ARMV8L = "<active_arm_aarch64_linux_config>";
  ACTIVE_CONFIG_WINDOWS = "<active_windows_config>";

  config "<config_name>"
  {
    <parameter_ID>="<value>";
    ...
  }
  ...
  files
  {
    path = "<file_or_directory>" [,
    platform="<platform>", compiler="<compiler>", action="<action>"];
    ...
  }
}
```

Parameters

filename

The sgproj filename.

top_level_component

The top-level component in the system. Can be overridden using the SimGen option `--top-component`.

active_x86_64_linux_config, active_arm_aarch64_linux_config, and active_windows_config

The name of the active build configuration for each supported host platform, unless overridden using the SimGen option `--configuration`.

config_name

Build configuration name. The format is `<host_OS>-[Debug|Release]-<compiler>`. For example `Linux64-Release-GCC-9.3`.

parameter_ID and value

Project parameter ID and value. These IDs also are exposed in the System Canvas **Project Settings** dialog. For a list of parameter IDs and their default values, see [3.2.6.17.3 Project parameter IDs](#) on page 81. To print all parameter IDs and values for a project, use the SimGen option `--print-config`. Parameter values set in the sgproj file can be overridden using the SimGen option `--override-config`.

file_or_directory

LISA source file, component repository file, or a directory to add to the include or library search path. Repository files have a `.sgrepo` filename extension. Directories have a trailing

/ character. File and directory names can be either absolute or relative to the project file location and paths can contain environment variables.

Any `path` statement can optionally include values for `platform`, `compiler`, and `action`. For example:

```
path = "../lib/Linux64_GCC-9.3/my_file.a", platform="Linux64", compiler="gcc-9.3", action="link|  
deploy";
```

If any of `platform`, `compiler`, or `action` is omitted, its default value is used.

platform

Optional. Host platform for which this configuration is valid. The default is any platform. Possible values are:

Linux64

Linux x86-64.

Linux64_armv81

Arm® AArch64 Linux.

Win64

64-bit Microsoft Windows, linked against release runtime library.

Win64D

64-bit Microsoft Windows, linked against debug runtime library.

compiler

Optional. On Linux, to select any compiler, omit this option. The default of `gcc` is then used. Possible values are:

vc2019

Microsoft Visual Studio 2019.

gcc

The first GCC version in the search path. To enable SimGen to automatically select the libraries that match the current GCC compiler, use this value.

gcc-7.3

GCC 7.3.

gcc-9.3

GCC 9.3.

gcc-10.3

GCC 10.3.

action

Optional. The default action depends on the file type, see [3.2.6.9 File/Path Properties dialog](#) on page 66 for details. Multiple actions, separated by `|` can be used.

If you apply a directory-specific action to a file, SimGen applies the action to the directory containing the file. In the following example, SimGen treats `MyFile.lisa` as LISA source and adds the parent directory of `MyFile.lisa` to the include and library search paths:

```
path = "MyFile.lisa", actions="lisa|incpath|libpath";
```

Possible values are:

lisa

Process the file as a LISA file. This action is not applicable to directories.

compile

Process the file as a C++ file. If acting on a directory, the compiler compiles all `*.c`, `*.cpp`, and `*.cxx` files in the directory.

ignore

Exclude the file or directory from the build and deploy process.

link

Link the file with existing files. If acting on a directory on Microsoft Windows, System Generator adds all `*.lib` and `*.obj` files in the directory to the linker input. On Linux, it adds all `*.a` and `*.o` files.

deploy

Produce a deployable file. The file is copied to the output directory of the active build configuration. If acting on a directory, SimGen copies the entire directory and its subdirectories to the destination. This action is the only action that acts recursively on subdirectories.

incpath

Add the directory to the list of additional include directories for the compiler. This value can also be set using the `INCLUDE_DIRS` project parameter. This is the default action for directories.

libpath

Add the directory to the list of directories that the linker searches for library files.

Example

This example project file builds an ISIM with configurations for Windows and Linux x86-64 and Linux Arm® AArch64 hosts:

```
sgproject "exampleSystem.sgproj"
{
    TOP_LEVEL_COMPONENT = "exampleSystem";
    ACTIVE_CONFIG_LINUX = "Linux64-Release-GCC-9.3";
    ACTIVE_CONFIG_LINUX_ARMV8L = "Linux64_armv8l-Release-GCC-9.3";
    ACTIVE_CONFIG_WINDOWS = "Win64-Release-VC2019";

    config "Linux64-Debug-GCC-9.3"
    {
        ADDITIONAL_COMPILER_SETTINGS = "-march=core2 -ggdb3 -Wall -std=c++14 -Wno-deprecated -Wno-unused-function";
        ADDITIONAL_LINKER_SETTINGS = "-Wl,--no-undefined";
        BUILD_DIR = "./Linux64-Debug-GCC-9.3";
        COMPILER = "gcc-9.3";
        CONFIG_DESCRIPTION = "Default x86_64 Linux configuration for GCC 9.3 with debug information";
    }
}
```

```

CONFIG_NAME = "Linux64-Debug-GCC-9.3";
ENABLE_DEBUG_SUPPORT = "1";
PLATFORM = "Linux64";
SIMGEN_COMMAND_LINE = "--num-comps-file 10";
TARGET_SYSTEMC_ISIM = "1";
}
config "Linux64-Release-GCC-9.3"
{
    ADDITIONAL_COMPILER_SETTINGS = "-march=core2 -O3 -Wall -std=c++14 -Wno-deprecated -Wno-
unused-function";
    ADDITIONAL_LINKER_SETTINGS = "-Wl,--no-undefined";
    BUILD_DIR = "./Linux64-Release-GCC-9.3";
    COMPILER = "gcc-9.3";
    CONFIG_DESCRIPTION = "Default x86_64 Linux configuration for GCC 9.3, optimized for
speed";
    CONFIG_NAME = "Linux64-Release-GCC-9.3";
    PLATFORM = "Linux64";
    PREPROCESSOR_DEFINES = "NDEBUG";
    SIMGEN_COMMAND_LINE = "--num-comps-file 50";
    TARGET_SYSTEMC_ISIM = "1";
}
config "Linux64_armv8l-Debug-GCC-9.3"
{
    ADDITIONAL_COMPILER_SETTINGS = "-march=armv8-a -ggdb3 -Wall -std=c++14 -Wno-deprecated -
Wno-unused-function";
    ADDITIONAL_LINKER_SETTINGS = "-Wl,--no-undefined";
    BUILD_DIR = "./Linux64_armv8l-Debug-GCC-9.3";
    COMPILER = "gcc-9.3";
    CONFIG_DESCRIPTION = "Default armv8l Linux configuration for GCC 9.3 with debug
information";
    CONFIG_NAME = "Linux64_armv8l-Debug-GCC-9.3";
    ENABLE_DEBUG_SUPPORT = "1";
    PLATFORM = "Linux64_armv8l";
    SIMGEN_COMMAND_LINE = "--num-comps-file 10";
    TARGET_SYSTEMC_ISIM = "1";
}
config "Linux64_armv8l-Release-GCC-9.3"
{
    ADDITIONAL_COMPILER_SETTINGS = "-march=armv8-a -O3 -fomit-frame-pointer -Wall -std=c++14
-Wno-deprecated -Wno-unused-function";
    ADDITIONAL_LINKER_SETTINGS = "-Wl,--no-undefined";
    BUILD_DIR = "./Linux64_armv8l-Release-GCC-9.3";
    COMPILER = "gcc-9.3";
    CONFIG_DESCRIPTION = "Default armv8l Linux configuration for GCC 9.3, optimized for
speed";
    CONFIG_NAME = "Linux64_armv8l-Release-GCC-9.3";
    PLATFORM = "Linux64_armv8l";
    PREPROCESSOR_DEFINES = "NDEBUG";
    SIMGEN_COMMAND_LINE = "--num-comps-file 50";
    TARGET_SYSTEMC_ISIM = "1";
}
config "Win64-Debug-VC2019"
{
    ADDITIONAL_COMPILER_SETTINGS = "/Od /RTCsu /Zi";
    ADDITIONAL_LINKER_SETTINGS = "/DEBUG";
    BUILD_DIR = "./Win64-Debug-VC2019";
    COMPILER = "VC2019";
    CONFIG_DESCRIPTION = "Default x86_64 Windows configuration for Visual Studio 2019 with
debug information";
    CONFIG_NAME = "Win64-Debug-VC2019";
    ENABLE_DEBUG_SUPPORT = "1";
    PLATFORM = "Win64D";
    SIMGEN_COMMAND_LINE = "--num-comps-file 10";
    TARGET_SYSTEMC_ISIM = "1";
}
config "Win64-Release-VC2019"
{
    ADDITIONAL_COMPILER_SETTINGS = "/O2";
    BUILD_DIR = "./Win64-Release-VC2019";
    COMPILER = "VC2019";

```

```
CONFIG_DESCRIPTION = "Default x86_64 Windows configuration for Visual Studio 2019,
optimized for speed";
CONFIG_NAME = "Win64-Release-VC2019";
PLATFORM = "Win64";
PREPROCESSOR_DEFINES = "NDEBUG";
SIMGEN_COMMAND_LINE = "--num-comps-file 50";
TARGET_SYSTEMC_ISIM = "1";
}
files
{
    path = "$(PVLIB_HOME)/etc/sglib.sgrepo";
    path = "../LISA/exampleSystem.lisa";
    path = "../LISA/exampleComponent.lisa";
}
}
```

Related information

[1.2 Project files](#) on page 8

[1.3 Repository files](#) on page 10

[1.4 File processing order](#) on page 11

[2.1 SimGen command-line options](#) on page 15

3. System Canvas

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

To launch System Canvas from the command line, type `sgcanvas`. It displays the platform as either LISA+ source code, or graphically, in a block diagram editor. System Canvas also allows you to build and launch platform models.

3.1 System Canvas tutorial

This tutorial describes how to perform some basic operations in System Canvas to build a standalone system model that can run an application image.

It demonstrates how to:

- Create a System Canvas project.
- Add, connect, and modify components in the project. You can use the Block Diagram view in System Canvas to do this. You do not need to edit LISA source code directly.
- Build the project.
- Debug an application on the model using Model Debugger.

3.1.1 Starting System Canvas

Start System Canvas from a Linux terminal or from the Microsoft Windows Start menu.

About this task

To start System Canvas:

- On Linux, enter `sgcanvas` in a terminal window. The command has these options:

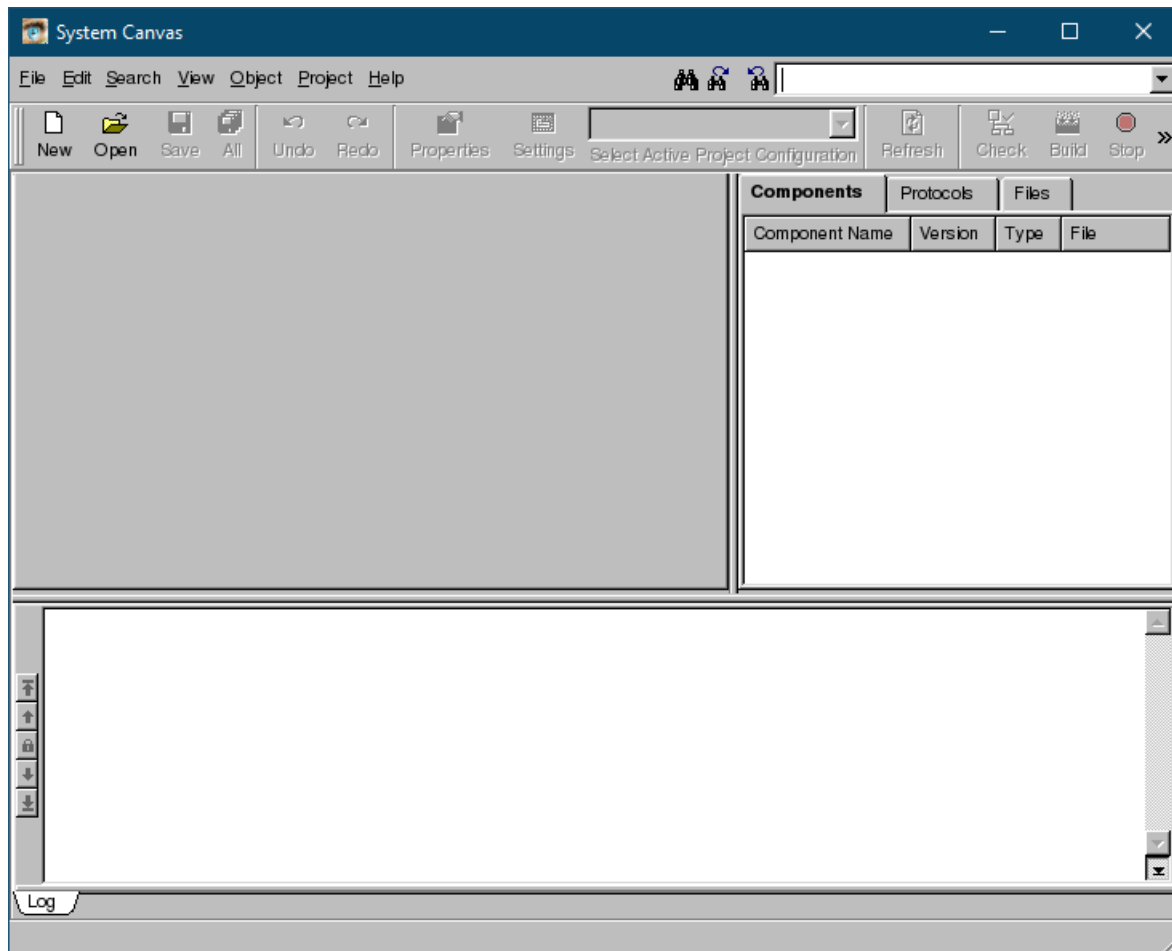
Table 3-1: System Canvas command-line options

Short form	Long form	Description
-h	--help	Print help text and exit.
-v	--version	Print version and exit.

- On Microsoft Windows, launch the **System Canvas** application from the **Start** menu.

The application contains the following subwindows:

- A blank diagram window on the left-hand side of the application window.
- A component window at the right-hand side.
- An output window across the bottom.

Figure 3-1: System Canvas at startup

Related information

[Preferences - Applications group](#) on page 74

[System Canvas GUI](#) on page 42

3.1.2 Creating a new project

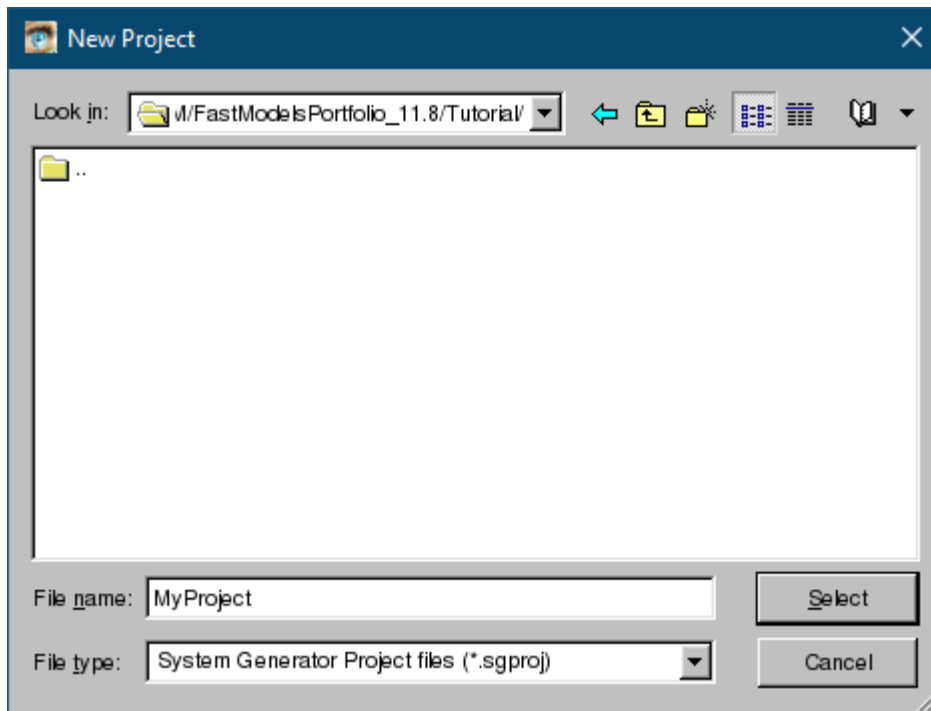
This section describes how to create a new project. The project will be used to create a new system model.

Before you begin

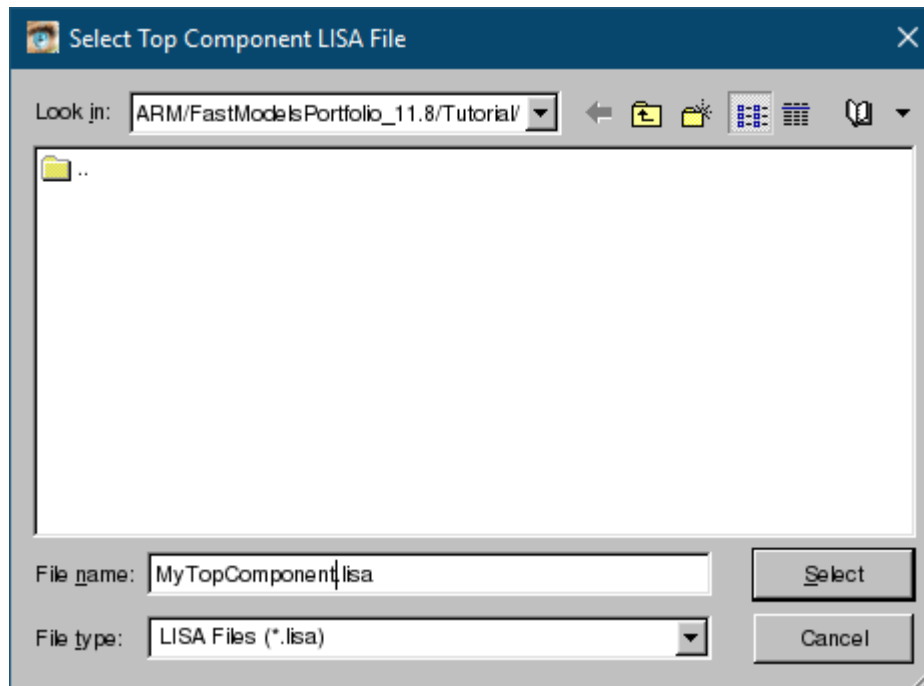
Make sure you have write permission for the directory in which you will create the project.

Procedure

1. Select **New Project** from the **File** menu. Alternatively, click the **New** button on the toolbar. The **New Project** dialog appears.

Figure 3-2: New Project dialog

2. Navigate to the directory to use for your project. Enter `MyProject` in the filename box and click the **Select** button.
A dialog appears for you to enter the name and location of the LISA+ file that represents your new system.

Figure 3-3: Select Top Component LISA File dialog

3. Enter `MyTopComponent.lisa` in the filename box and click the **Select** button.
The component name for the top component is, by default, set to the name of the LISA+ file.

The Workspace area contains a blank block diagram with scroll bars. The Component window, to the right of the Workspace area, lists the components in the default repositories.

Results

These steps create a project file, `MyProject.sgproj` and a LISA+ source file, `MyTopComponent.lisa`. The project file contains:

- System components.
- Connections between system components.
- References to the component repositories.
- Settings for model generation and compilation.

Do not edit the project file. System Canvas modifies it if you change project settings.

The block diagram view of your system is a graphical representation of the LISA+ source. To display the contents of `MyTopComponent.lisa`, click the **Source** tab. This file is automatically updated if you add or rename components in the block diagram.

You can view the LISA+ source for many of the supplied components. To do so, double-click on a component in the Block Diagram. Alternatively, right click on a component in the Components window and select **Open Component**.

3.1.3 Adding the Arm® processor

This section describes how to add an Arm® processor component to the system model.

Procedure

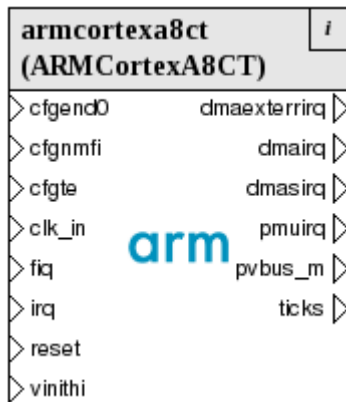
1. Click the **Block Diagram** tab in the Workspace window, unless the block diagram window is already visible.
A blank window with grid points appears.
2. Select the **Components** tab in the Components window to display the Fast Models Repository components.
3. Move the mouse pointer over the `ARMCortexA8CT` processor component in the Component window and press and hold the left mouse button.
4. Drag the component to the middle of the Workspace window.



If you move the component within the Workspace window, the component automatically snaps to the grid points.

5. Release the left mouse button when the component is in the required location.
The system receives the component.

Figure 3-4: ARMCortexA8CT processor component in the Block Diagram window



6. Save the file by selecting **File > Save File** or using **Ctrl+S**.
The asterisk (*) at the end of the system name, in the title bar, shows unsaved changes.

Results

These steps create a System Canvas file, `MyTopComponent.sgcanvas`, in the same location as the project and LISA+ files. It contains the block diagram layout information for your system. Do not edit this file.

3.1.4 Naming components

This section describes how to change the name of a component, for example the processor.

About this task



Component names cannot have spaces in them, and must be valid C identifiers.

Procedure

1. Select the component and click the **Properties** button on the toolbar to display the **Component Instance Properties** dialog.
You can also display the dialog by either:
 - Right-clicking on the component and select **Object Properties** from the context menu.
 - Selecting the component and then selecting **Object Properties** from the **Object** menu.
2. Click the **General** tab on the **Component Instance Properties** dialog.
3. Enter `Arm` in the **Instance name** field.
4. Click **OK** to accept the change. The instance name of the component, that is the name displayed in the processor component title, is now `Arm`.

3.1.5 Resizing components

This section describes how to resize components.

Procedure

1. Select the processor component and move the mouse pointer over one of the green resize control boxes on the edges of the component.
2. Hold the left mouse button down and drag the pointer to resize the component.
3. Release the mouse button to end the resize operation.
To vertically resize the component title bar to avoid truncating text, click the component and drag the lower handle of the shaded title bar.

3.1.6 Hiding ports

This section describes how to hide ports, for instance because they are not connected to anything.

About this task

If there are only a few ports to hide, use the port context menu. Right click on the port and select **Hide Port**. To hide multiple ports:

Procedure

1. Select the component and then select **Object Properties** from the **Object** menu.

2. Click the **Ports** tab on the dialog.
3. Click **Select All** to select all of the ports.
4. Click **Hide selected ports**.
5. Select the boxes next to `clk_in` and `pvbus_m`.
6. Click **OK** to accept the change, so that all ports except `clk_in` and `pvbus_m` are hidden in the Block Diagram view.

Results

Figure 3-5: Processor component after changes



Related information

[Using port arrays](#) on page 31

3.1.7 Moving ports

This section describes how to move ports, for example to improve readability.

Procedure

1. Place the mouse pointer over the port. The mouse pointer changes shape to a hand with a pointing finger. This is the move-port mouse pointer.
2. Press and hold the left mouse button down over the port, and drag the port to the new location.
This can be anywhere along the inner border of the component that is not on top of an existing port. If you select an invalid position, the port returns to its original location.
3. When the port is in position, release the mouse button.
Arrange any other ports as needed. The `clk_in` port must be on the left side.

3.1.8 Adding components

This section describes how to add components to a project.

Procedure

1. Drag and drop the following components onto the Block Diagram window:
 - `ClockDivider`.
 - `MasterClock`.
 - `PL340_DMC`.
 - `PVBusDecoder`.
 - `RAMDevice`.

The `PL340_DMC` component is included to demonstrate some features of System Canvas and is not part of the final example system.

2. Select the new components individually and use the **General** tab of the **Component Instance Properties** dialog to rename them to:
 - Divider.
 - Clock.
 - PL340.
 - BusDecoder.
 - Memory.

3.1.9 Using port arrays

This section describes how to expand, collapse, and hide port arrays.

Procedure

1. Right click on one of the `axi_if_in` ports in the `PL340` component to open a context menu. Select **Collapse Port** to reduce the port array to a single visible item in the component.
2. Select the `PL340` component and then select **Object Properties** from the **Object** menu.
3. Select the **Ports** tab in the **Component Instance Properties** dialog. The `axi_if_in` port is a port array as indicated by the + beside the port name. Click the + to expand the port tree view.
4. Deselect the checkboxes beside `axi_if_in[2]` and `axi_if_in[3]` to hide the chosen array ports so that expanding the port array still does not display them. Click **OK** to close the dialog. You can also hide a port by using the port context menu and selecting **Hide Port**.
5. To expand the `axi_if_in` port in the `PL340` component, you can:
 - Right click on the port and select **Expand Port** from the port context menu.
 - a. Display the **Component Instance Properties** dialog.
 - b. Select the **Ports** tab.
 - c. Click the + next to the port array to expand the port tree view.
 - d. Select the **Show as Expanded** radio button.

Only the `axi_if_in[0]` and `axi_if_in[1]` ports are shown.

6. To redisplay the `axi_if_in[2]` and `axi_if_in[3]` ports, you can:
 - Use the port context menu and select **Show All Ports**.
 - Reverse the deselection step, selecting the checkboxes next to the hidden ports, in the **Component Instance Properties** dialog.

Ports with more than eight items are shown collapsed by default.

Next steps

The rest of this tutorial does not require the `PL340` component, so you can delete it.

Figure 3-6: Example system with added components

3.1.10 Connecting components

This section describes how to connect components.

Procedure

1. Select connection mode, by doing either of the following:

- Click the **Connect** button.
- Select **Connect Ports Mode** from the **Edit** menu.

2. Move the mouse pointer around in the Block Diagram window:

Option	Description
Not over an object	The pointer changes to the invalid pointer, a circle with a diagonal line through it.
Over an object	The pointer changes to the start connection pointer and the closest valid port is highlighted.

3. Move the cursor so that it is over the `clock` component and close to the `clk_out` port.
4. Highlight the `clk_out` port, then press and hold the left mouse button down.
5. Move the cursor over the `clk_in` port of the `Divider` component.
6. Release the mouse button to connect the two ports.
The application remains in connect mode after the connection is made.
7. Make the remaining connections.

Figure 3-7: Connected components

Connections between the addressable bus ports have bold lines.

3.1.11 View project properties and settings

Before building the model, verify the toolchain configuration and top component using the **Project Settings** dialog.

3.1.11.1 Viewing the project settings

Use the **Project Settings** dialog to view and edit the project configuration. Although no changes are required for this tutorial, this section demonstrates the steps to use if changes were necessary.

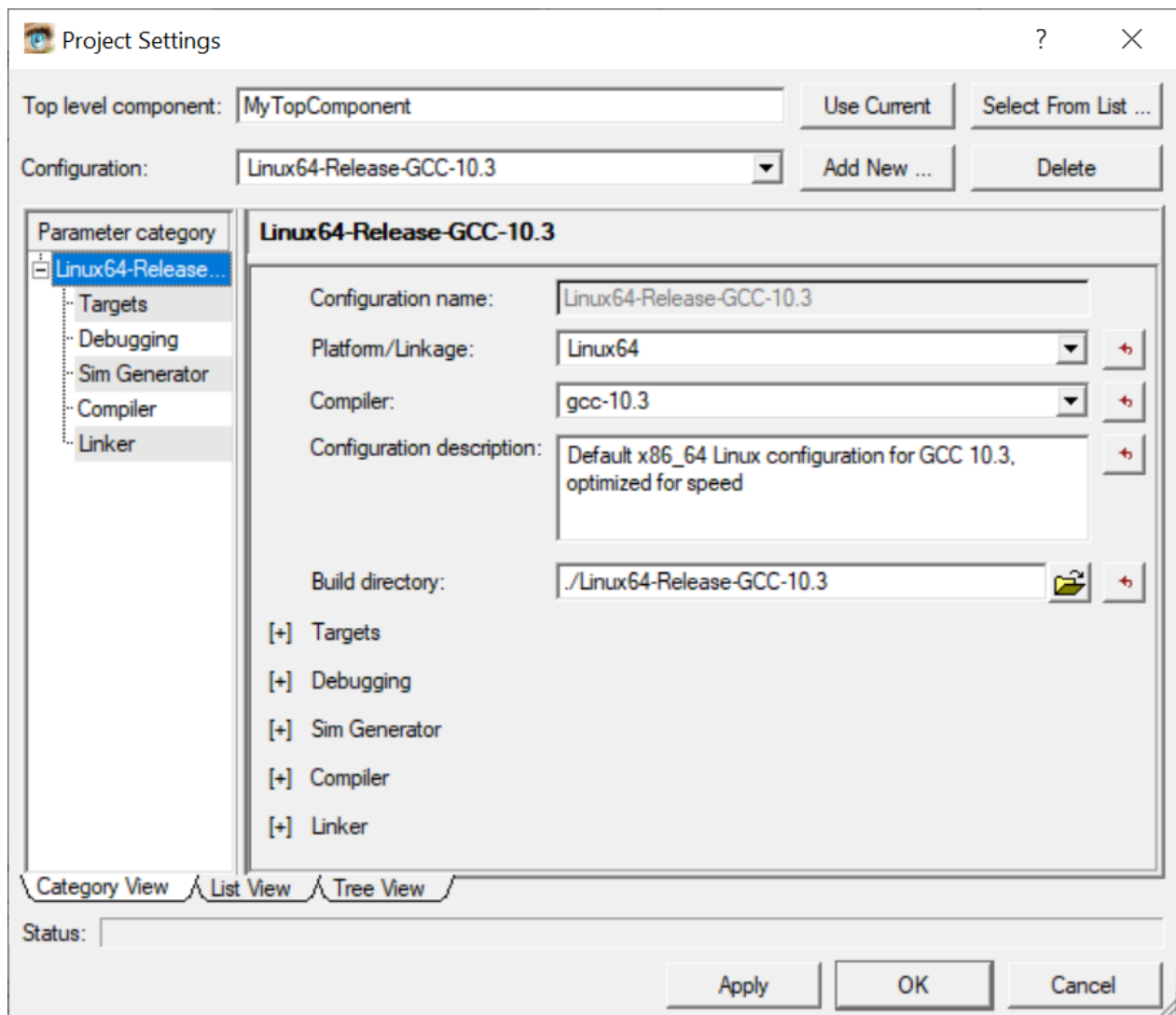
Procedure

Open the **Project Settings** dialog to inspect the project settings for the system, by doing either of the following:

- Click the **Settings** button.
- Select **Project Settings** from the **Project** menu.

The **Project Settings** dialog appears:

Figure 3-8: Project settings for the example



The **Category View**, **List View**, and **Tree View** tabs present different views of the project parameters.

3.1.11.2 Specifying the Active Project Configuration

Use the **Select Active Project Configuration** drop-down menu on the main toolbar to display the configuration options that control how the target model is generated.

About this task

You can choose to build:

- Models with debug support.
- Release models that are optimized for speed.

Display and edit the full list of project settings by selecting **Project Settings** from the **Project** menu. Inspect and modify a configuration for your operating system by selecting it from the **Configuration** drop-down list and clicking the different list elements to view the settings.



- The configuration options available, including compilers and platforms, depend on the operating system.
 - Projects that were created with earlier versions of System Generator might not have the compiler version specified in the **Project Settings** dialog, but are updateable.
-

3.1.11.3 Selecting the top component

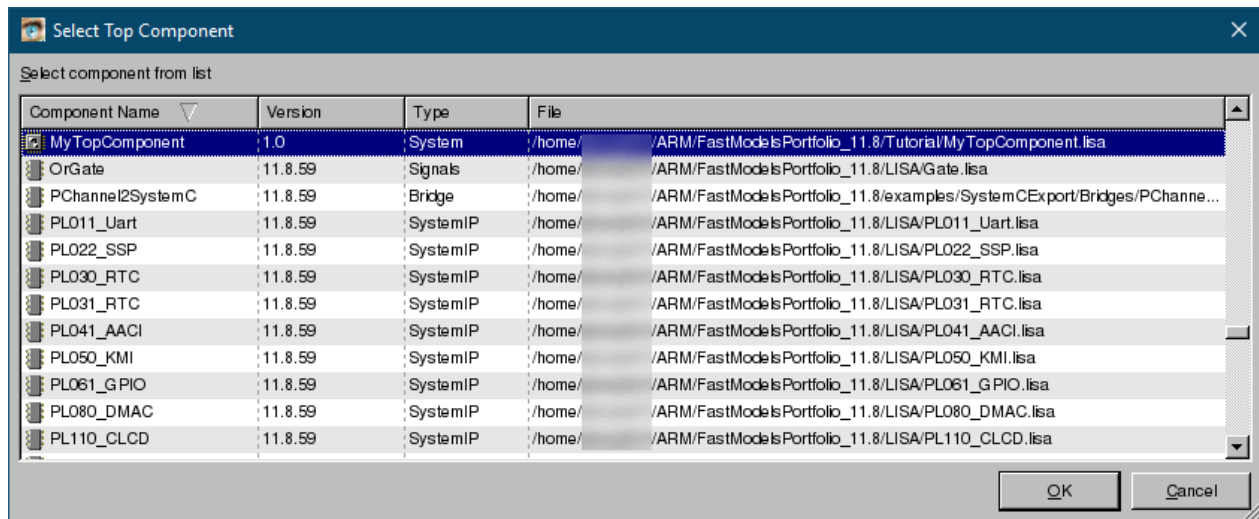
The top component defines the root component of the system. Any component can be set as the top component. This flexibility enables building models from subsystems.

About this task

In the **Project Settings** dialog, click the **Select From List...** button. The **Select Top Component** dialog opens and lists all the components in the system.



If the value in the **Type** column is `system`, the component has subcomponents.

Figure 3-9: Select Top Component dialog showing available components

3.1.12 Changing the address mapping

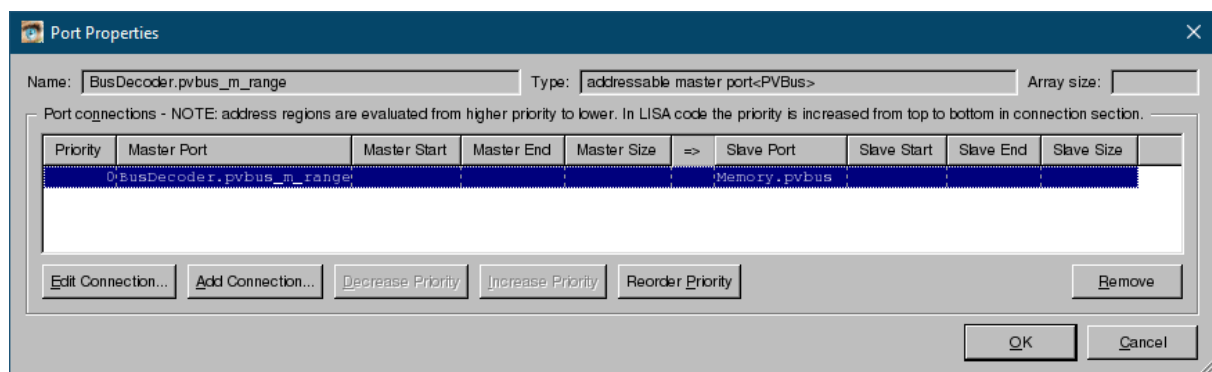
Addressable bus mappings, connections that have bold lines, have editable address maps.

About this task

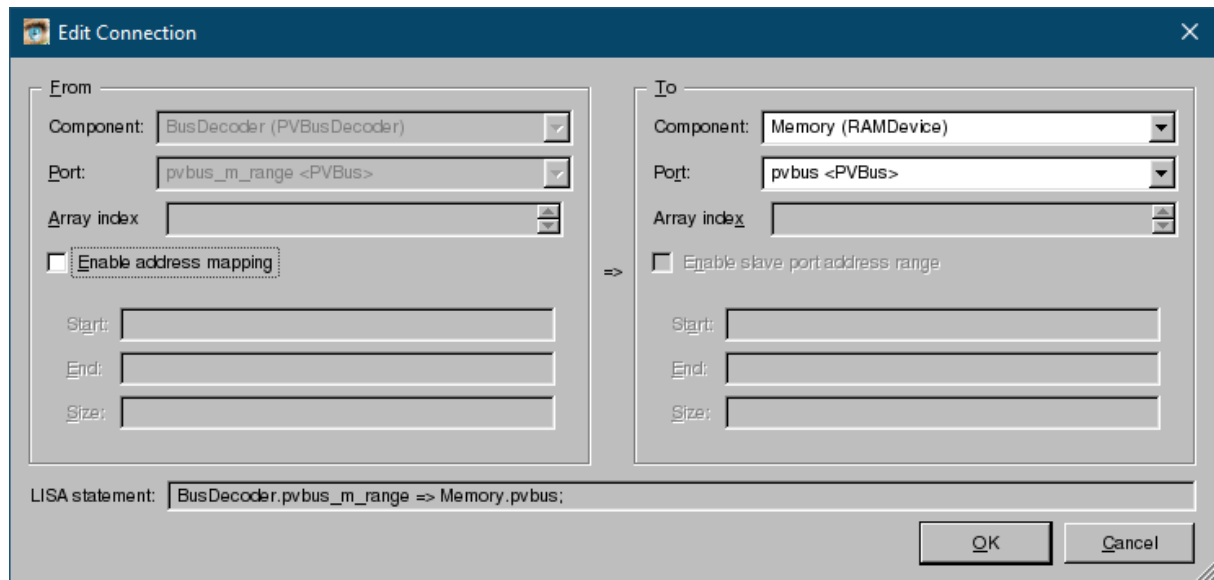
Follow this procedure to change the address mapping.

Procedure

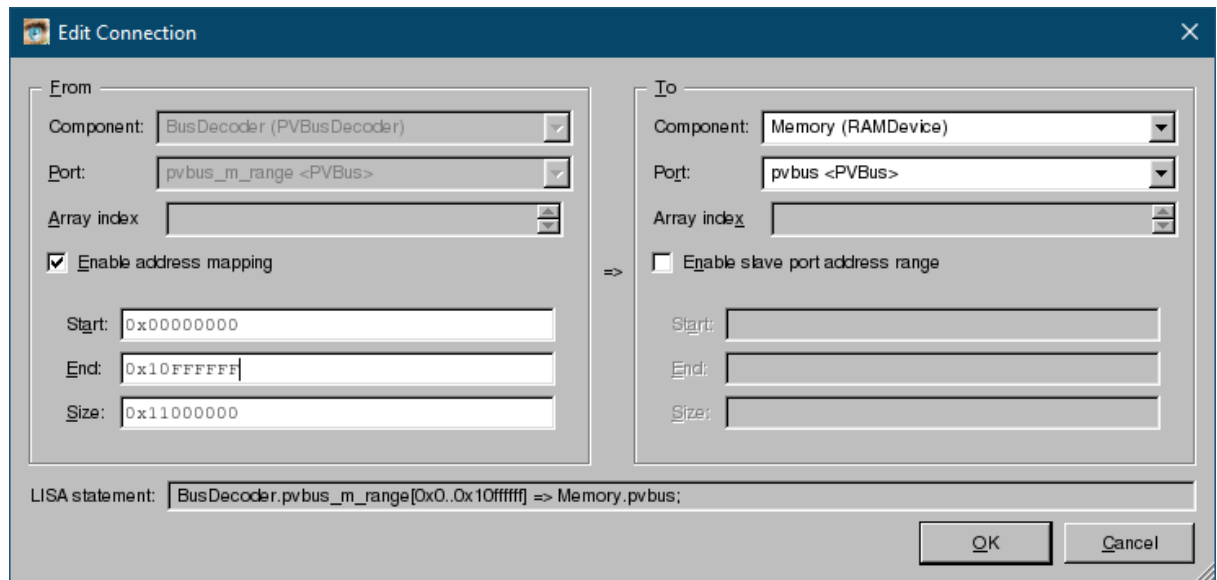
1. Double-click the `pbus_m_range` port of the BusDecoder component to open the **Port Properties** dialog.

Figure 3-10: Viewing the address mapping from the Port Properties dialog

2. Open the **Edit Connection** dialog by doing either of the following:
 - Select the `Memory.pbus` Slave Port line, and click **Edit Connection...**
 - Double click on the entry.

Figure 3-11: Edit Connection dialog

3. Select the **Enable address mapping** checkbox to activate the address text fields. The address mapping for the master port is shown on the left side of the **Edit Connection** dialog. **Start**, **End**, and **Size** are all editable. If one value changes, the other values are automatically updated if necessary. The equivalent LISA statement is displayed at the bottom of the **Edit Connection** dialog.
4. Enter a **Start** address of 0x00000000 and an **End** address of 0x10FFFFFFF in the active left-hand side of the **Edit Connection** dialog. The **Size** of 0x11000000 is automatically calculated. This step maps the master port to the selected address range. If mapping the master port to a different address range on the slave port is required, select **Enable slave port address range**. Checking it makes the parameters for the slave port editable. The default values are the same as for the master port when the slave address range is enabled. Disabling the slave address range is equivalent to specifying the address range 0...size-1, and not the master address range. In this case, a slave port address range is not required, so deselect the **Enable slave port address range** checkbox.

Figure 3-12: Edit address map for master port

5. Click **OK** to close the **Edit Address Mapping** dialog for the `Memory.pvbus` slave port.
6. Click **OK** to close the **Port Properties** dialog.

3.1.13 Building the system

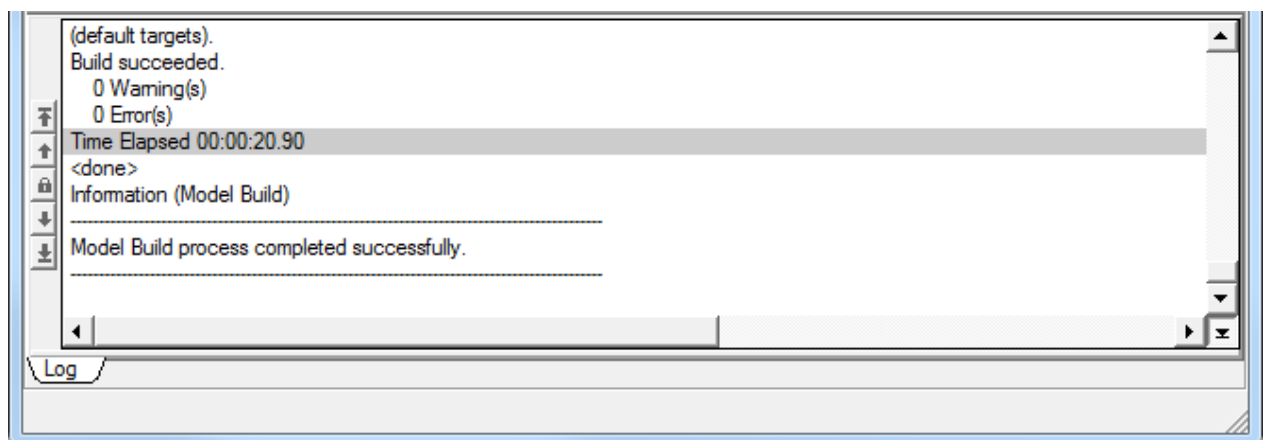
This section describes how to build the model as an `.so` or `.dll` library.

Procedure

Click the **Build** icon on the System Canvas toolbar to build the model.

System Canvas might perform a system check, depending on your preference setting. If warnings or errors occur, a window might open. Click **Proceed** to start the build.

The progress of the build is displayed in the log window.

Figure 3-13: Build process output

Depending on the speed of your computer and the type of build selected, this process might take several minutes.

You can reduce compilation time by setting the `simGen` options `--num-comps-file` and `--num-build-cpus` in the **Project Settings** dialog.

Related information

[Building a SystemC ISIM target](#) on page 41

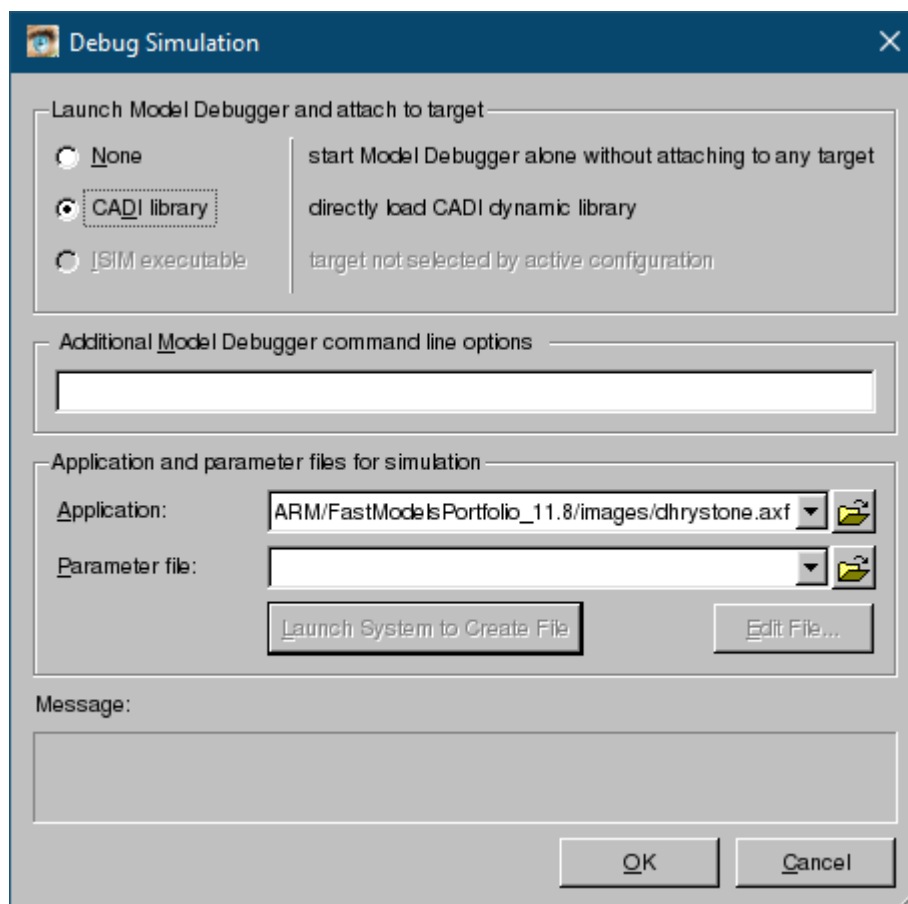
3.1.14 Debugging with Model Debugger

This section describes how to use Model Debugger to debug the model.

Procedure

1. Click the **Debug** button on the System Canvas toolbar to open the **Debug Simulation** dialog:

Figure 3-14: Debug Simulation dialog



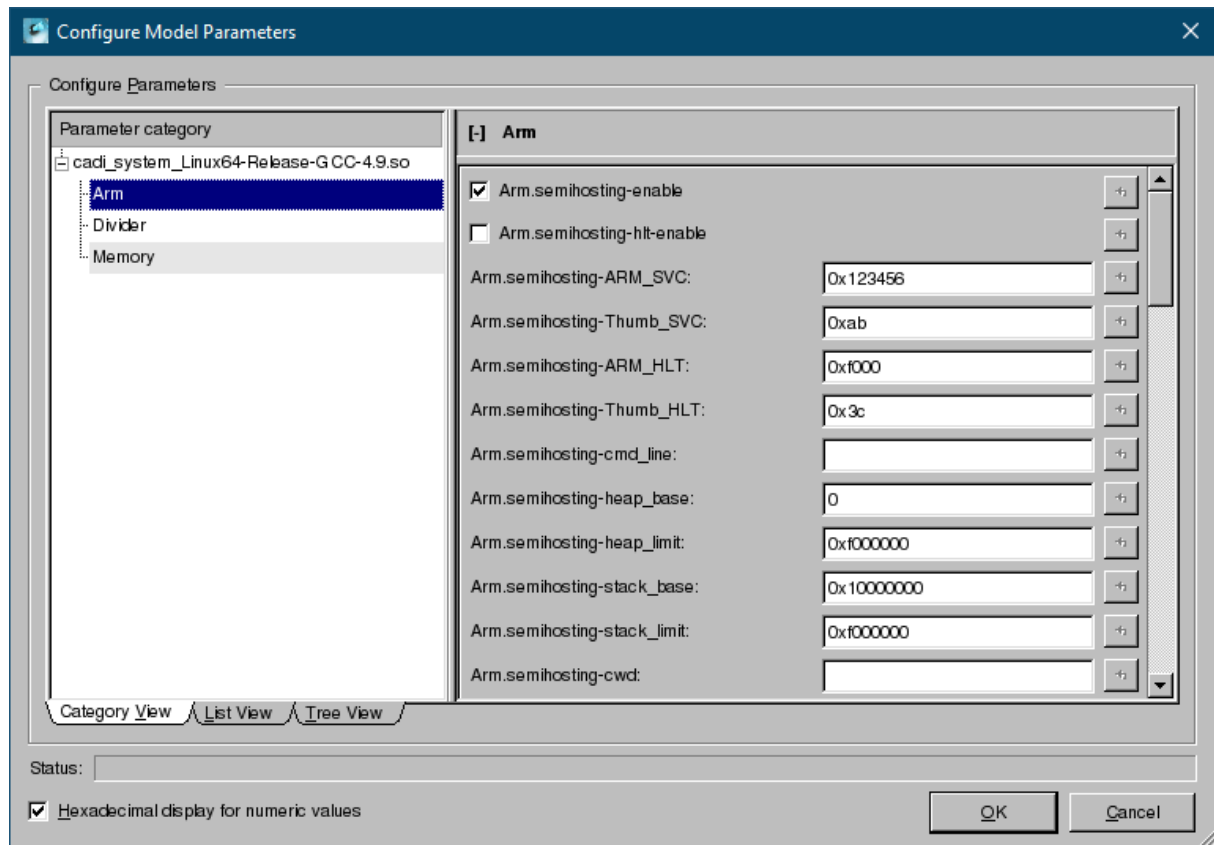
2. Select the **CADI library** radio button to attach Model Debugger to your CADI target. The radio buttons that are available depend on the target settings.
3. Specify the location of the application that you want to run in the **Application** field.

This example uses `dhrystone.axf`, which is part of the Third-Party IP add-on package for the Fast Models Portfolio.

4. Click **OK** to start Model Debugger.

An instance of Model Debugger starts. The debugger loads the model library from the `build` directory of the active configuration. Model Debugger displays the **Configure Model Parameters** dialog containing the instantiation parameters for the top-level components in the model:

Figure 3-15: Configure Model Parameters dialog

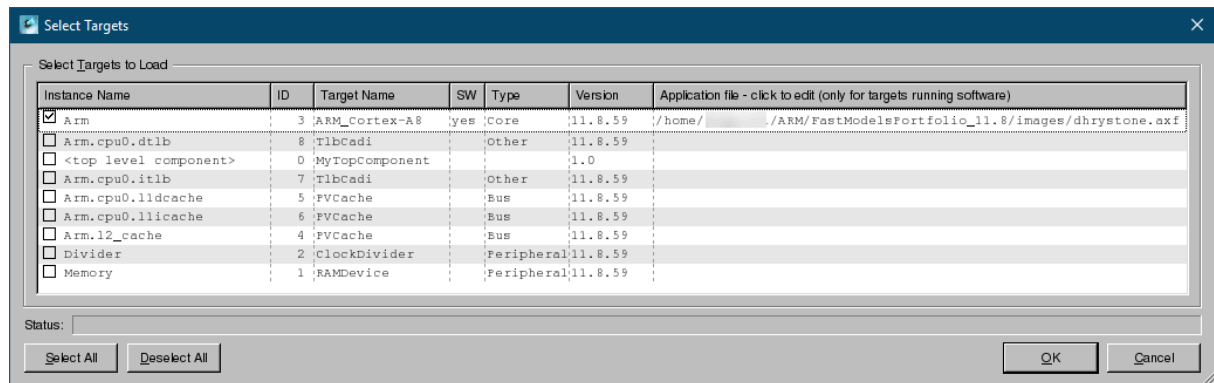


To display parameter sets:

- Select a Parameter category in the left-hand side of the dialog.
- Click a + next to a component name in the right-hand side.

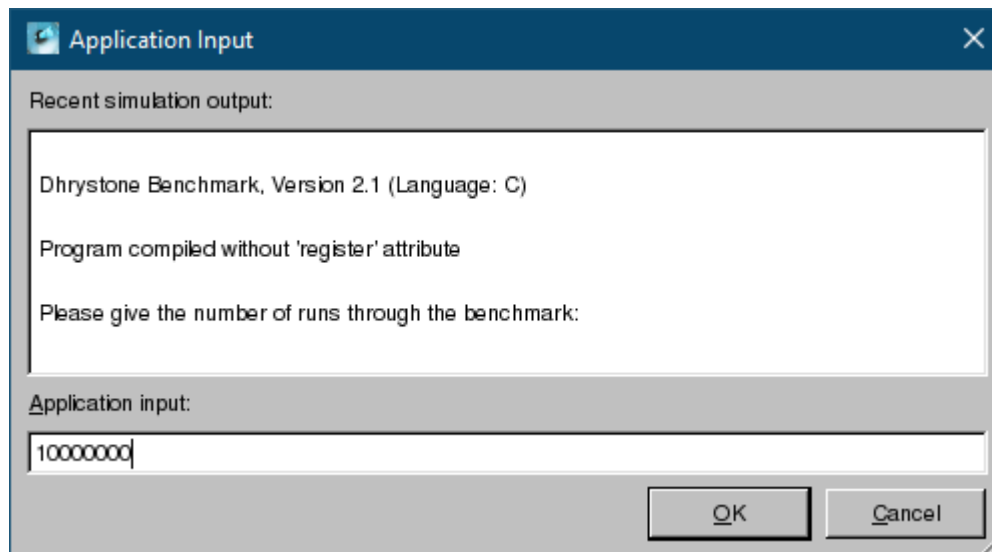
For different views of the system parameters, select the **List View** or **Tree View** tabs.

5. Click **OK** to close the dialog.

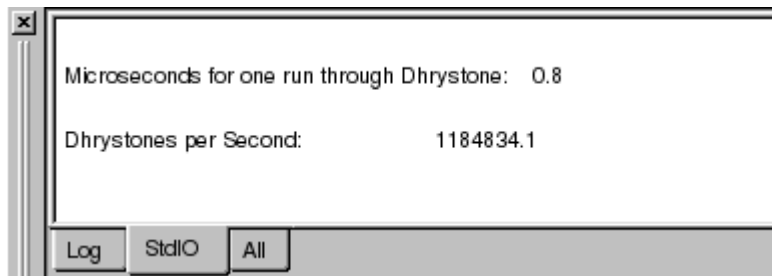
Figure 3-16: Select Targets dialog

The **Select Targets** dialog displays the components to use in Model Debugger. The Arm® processor component is the default.

6. Click **OK** to close the dialog.
7. Click **Run** to start the simulation.
The **Application Input** window appears:

Figure 3-17: Model Debugger Application Input window

8. Enter the required number of runs through the benchmark in the **Application input** field and click **OK**.
After a short pause, the benchmark results are shown in the **StdIO** window.

Figure 3-18: Model Debugger StdIO window

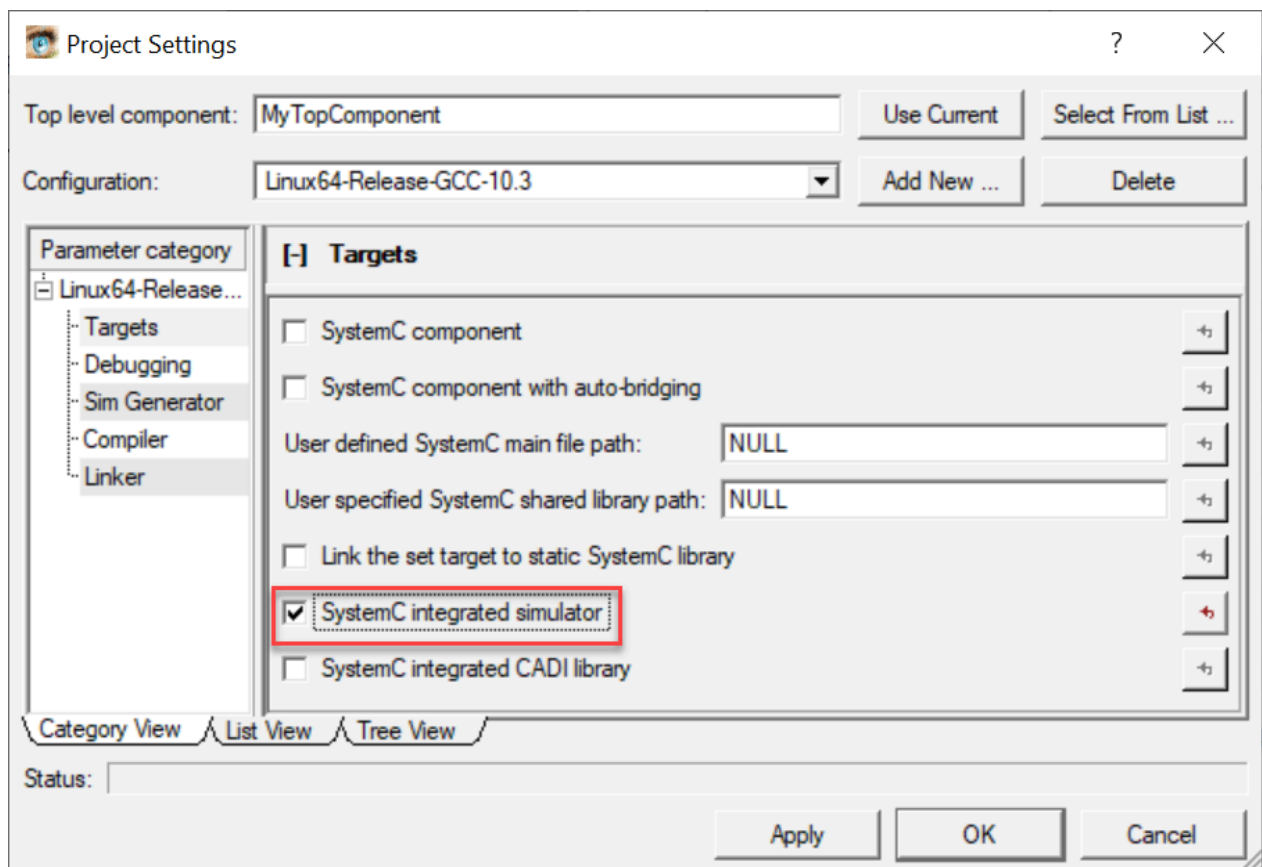
Related information

[Model Debugger](#) on page 87

3.1.15 Building a SystemC ISIM target

To build the platform as a standalone executable SystemC Integrated Simulator (ISIM), tick the **SystemC integrated simulator** checkbox in the **Targets** category in the **Project Settings** dialog.

About this task

Figure 3-19: Building a SystemC integrated simulator target

This option is selected by default for new projects.

System Canvas generates a SystemC ISIM target by statically linking the model with the SystemC framework.

The output executable is called `isim_system`, and is generated in the build directory.

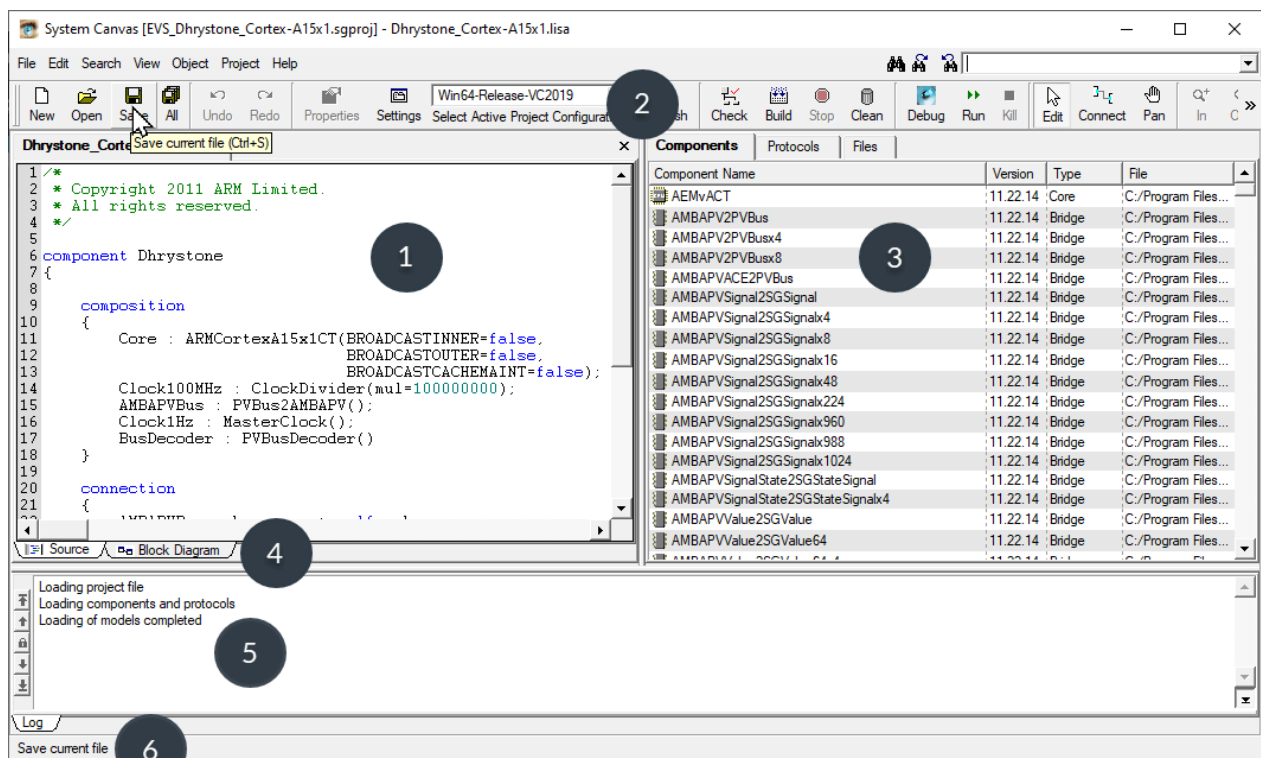
Related information

[Building Fast Models](#)

3.2 System Canvas GUI

This section describes System Canvas, the GUI to the Fast Models tools, which shows the components in a system, component ports, external ports (if the system itself is a component), and connections between ports.

Figure 3-20: Layout of System Canvas



1. **Workspace** displays either the model source code or a block diagram view of the model. You can edit the model through these views.
2. The main toolbar contains buttons for frequently-used options.
3. **Component list** lists all of the available components, protocols, and files in the current project.
4. **Workspace** tabs let you switch between the source code view and block diagram view.

5. **Output window** displays the output from the build or script command.
6. **Status bar** displays information about menu items, commands, and buttons.

3.2.1 Menu bar

The main bar provides access to System Canvas functions and commands.

3.2.1.1 File menu

The **File** menu lists file and project operations.

New Project

Create a new model project.

Load Project

Open an existing project.

Close Project

Close a project. If there are pending changes, the **Save changes** dialog appears.

Save Project

Save the changes made to a project.

Save Project As

Save a project to a new location and name.

New File

Create a new file. The **New File** dialog appears. Select the type from the **File type** drop-down list.

Open File

This displays the **Open File** dialog. Filter the types to display by selecting the type from the **File type** drop-down list. Non-LISA files open as text in the source editor.

Close File

Close a LISA file. A dialog prompts to save any changes.

Save File

Save the changes made to the current LISA file.

Save File As

Save a LISA file to a new location and name.

Save All

Save the changes made to the project and the LISA files.

Print

Print the contents of the **Block Diagram** window.

Preferences

Modify the user preferences.

Recently Opened Files

Display the 16 most recently opened LISA files. Click on a list entry to open the file.

To remove a file from the list, move the mouse cursor over the filename and press the **Delete** key or right click and select **Remove from list** from the context menu.

Recently Opened Projects

Display the 16 most recently opened projects. Click on a list entry to open the project.

To remove a project from the list, move the mouse cursor over the project name and press the **Delete** key or right click and select **Remove from list** from the context menu.

Exit

Close System Canvas. A dialog prompts to save any changes. Disable it by selecting **Do not show this message again**. Re-enable it in the preferences.

Related information

[New project dialogs](#) on page 70

[Preferences - Suppressed messages group](#) on page 78

3.2.1.2 Edit menu

The **Edit** menu lists content operations.

Undo

Undo up to 42 of the latest changes to a file in the **Source** view or to the layout in the **Block Diagram** view. These actions are undoable:

- Add an object such as a component, label, or connection.
- Paste or duplicate.
- Cut or delete.
- Edit object properties.
- Move.
- Resize.

Undo and **Redo** operations can affect **Block Diagram** view zoom and scroll actions.



Undo and Redo typically work normally. For example:

1. Change the system in the **Block Diagram** view by adding a RAMDevice component with name `RAM`.
2. Switch to **Source** view. The text `RAM : RAMDevice();` is present in the composition section.
3. Change the code by removing the line `RAM : RAMDevice();`.

4. Change the code by adding, for example, the line `PVS : PVBusSlave();`.
 5. Click on the **Block Diagram** tab. The change to the source code is reflected by the `RAM` component being replaced by the `PVS` component.
 6. Select **Undo** from the **Edit** menu. The **Block Diagram** view shows that `RAM` is present but `PVS` is not.
 7. Select **Redo** from the **Edit** menu. The **Block Diagram** view shows that `PVS` is present but `RAM` is not.
-

Redo

Redo the last undone change. This cancels the result of selecting **Undo**. Selecting **Redo** multiple times cancels multiple **Undo** actions.

Cut

Cut the marked element into the copy buffer.

Copy

Copy the marked element into the copy buffer.

Paste

Paste the content of the copy buffer at the current cursor position.

Duplicate

Duplicate the marked content.

Delete

Delete the marked element.

Select All

Select all elements.

Edit Mode

Change the **Workspace** to **Edit** mode. The cursor can select components.

Connect Ports Mode

Select **Connection** mode. The cursor can connect components.

Pan Mode

Select **Movement** mode. The cursor can move the entire system in the **Workspace** window.

3.2.1.3 Search menu

The **Search** menu lists find, replace and go to functions.

Find

Search for a string in the active window (with a thick black frame).

Find Next

Repeat the last search.

Find Previous

Repeat the last search, backwards in the document.

Replace

In the **Source** view, search for and replace strings in a text document.

Go To Line

In the **Source** view, specify a line number in the currently open LISA file to go to.



Use the search icons at the top right of the application window to search for text. Entering text in the search box starts an incremental search in the active window.

Related information

[Find and Replace dialogs](#) on page 68

3.2.1.4 View menu

The **View** menu lists the **Workspace** window display options.

Show Grid

Using the grid simplifies component alignment.

Zoom In

Show more detail.

Zoom Out

Show more of the system.

Zoom 100%

Change the magnification to the default.

Zoom Fit

Fit the entire system into the canvas area.

Zoom Fit Selection

Fit the selected portion into the canvas area.

3.2.1.5 Object menu

The **Object** menu lists system and system component operations.

Open Component

Open the source for the selected component.

Add Component

Display all of the components available for adding to the block diagram.

Add Label

The mouse cursor becomes a default label. To add the label, move it to the required location in the **Block Diagram** window and click the left mouse button. The **Label Properties** dialog appears.

Add Port

Display the **External Port** dialog. Specify the type of port to add.

Mirror Self Port

Switch the direction that the external port image points in. It does not reverse the signal direction, so a master port remains a master port. If an unconnected port is not selected, this option is disabled.

Expand Port

For a port array, display all of the individual port elements. Expanded is the default for port arrays with eight or fewer ports. Collapsed is the default for port arrays with more than eight elements.



Ports with many elements might expand so that elements appear on top of one another. Either: click and drag them apart, or collapse the port, increase the component size, then expand the port again.

Collapse Port

For a port array, hide the individual port elements and only display the top-level port name.

Hide Port

Disable the selected port and make it invisible.

Hide All Unconnected Ports

Hide all ports that are not connected to a component.

Show/Hide Ports of Protocol Types...

Hide all ports that use a specified protocol. The **Show/Hide Connection Types** dialog appears. Select the protocols to filter.

Show All Ports

Show all ports. Some might overlap if there is not enough space.

Autoroute Connection

Redraw the selected connection.

Autoroute All Connections

Redraw all of the connections.

Documentation

Open the documentation for the selected component.

Object Properties

Display the **Component Instance Properties** dialog to view and edit the properties for the selected component.

3.2.1.6 Project menu

The **Project** menu lists build, check, configure, run, and set options.

Check System

Check for errors or missing information. This feature does not check everything, but does give useful feedback.

Generate System

Generate the C++ source code, but do not compile it. After generation, click **Build System** and **Debug** to run the model.

Build System

Generate and compile the generated C++ source code, producing a library or a runnable model.

Stop Build

Cancel the active build process.

Clean

Delete all generated files.

Launch Model Debugger

Execute the simulation under the control of Model Debugger.

Run

Run...

Open the **Run** dialog to specify the run command.

Run in Model Shell

Execute the simulation under the control of Model Shell with command-line options taken from project settings and user preferences.

Run ISIM system

Execute the simulation as an ISIM executable with Model Shell command-line options taken from project settings and user preferences.

Clear History

Clear all recent run command entries.

Recent Command Entries (up to 10)

Call recent command entries.

Kill Running Command

Stop the running synchronous command.

Launch Host Debugger

Microsoft Windows

Launch Microsoft Visual Studio. Build the system there, and start a debug session.



You can take the command-line arguments for ISIM systems or Model Shell from Microsoft Visual Studio by selecting **Project > Properties > Configuration Properties > Debugging**.

Linux

Launch the executable or script set in the application preferences. The target must be an ISIM executable. Arm recommends this method for debugging at source-level.

Add Files

Add files to the system.

Add Current File

Add the currently open file to the system.

Refresh Component List

Update the **Component List** window to show all available components.

Setup Default Repository

Display the **Default Model Repository** section of the **Preferences** window, and select the default repositories for the next new project.



This option does not affect the currently open project.

Set Top Level Component

Displays the **Select Top Component** dialog that lists all available components in the system.

The top component defines the root component of the system. It can be any component. This option enables building models from subsystems.



If the value in the **Type** column is `system`, the component has subcomponents.

Active Configuration

Select the system build configuration from the project file list.

Project Settings

Display the **Project Settings** dialog.

Related information

[Preferences - Applications group](#) on page 74

3.2.1.7 Help menu

The **Help** menu lists documentation, software and system information links.

Fast Models User Guide

Display the Fast Models User Guide.

Fast Models Tools User Guide

Display this document.

LISA+ Language Reference Guide

Display the LISA+ Language for Fast Models Reference Guide.

AMBA-PV User Guide

Display the AMBA-PV Extensions to TLM 2.0 User Guide.

CADI User Guide

Display the Component Architecture Debug Interface v2.0 User Guide.

Release Notes

Display the Fast Models release notes.

Documents in \$PVLIB_HOME/Docs

List the PDF files in the directory `$PVLIB_HOME/Docs`.

End User License Agreement (EULA)

Display the license agreement.

About

Display the version and license information.

System Information

Display information about the tools and loaded models.

3.2.2 Toolbar

The toolbar sets out frequently used menu functions.

New

Create a new project or LISA file.

Open

Open an existing project or file.

Save

Save current changes to the file.

All

Save project and all open files.

Undo

Undo the last change in the **Source** or **Block Diagram** view.

Redo

Undo the last undo.

Properties

Display the **Properties** dialog for the selected object:

Nothing

The **Component Model Properties** dialog, with the properties for the top-level component.

Component

The **Component Instance Properties** dialog.

Connection

The **Connection Properties** dialog.

Port

The **Port Properties** dialog.

Self port

The **Self Port Properties** dialog.

Label

The **Label Properties** dialog.



The **Properties** button only displays properties for items in the block diagram.

Settings

Display the project settings.

Select Active Project Configuration

Select the build target for the project.

Refresh

Refresh the component and protocol lists.

Check

Perform a basic model error and consistency check.

Build

Generate a virtual system model using the project settings.

Stop

Stop the current generation process.

Clean

Delete all generated files.

Debug

Start Model Debugger to debug the generated simulator.

Run

Execute the most recent run command. The down arrow next to the button opens the **Run** dialog.

Kill

Stop Model Shell and end the simulation.

Edit

Edit mode: the cursor selects and moves components.

Connect

Connection mode: the cursor connects components.

Pan

Movement mode: the cursor moves the entire system in the **Workspace** window.

Zoom

Use the **In**, **Out**, **100%**, and **Fit** buttons to change the system view zoom factor in the **Workspace** window.

Related information

[Viewing the project settings](#) on page 32

[Edit menu](#) on page 44

[Component Instance Properties dialog](#) on page 60

[Component Model Properties dialog for the system](#) on page 62

[Connection Properties dialog](#) on page 65

[Label Properties dialog](#) on page 69

[New File dialog \(File menu\)](#) on page 70

[Open File dialog](#) on page 71

[Port Properties dialog](#) on page 72

[Self Port dialog](#) on page 86

3.2.3 Workspace window

This section describes the **Workspace** window, which displays editable representations of the system.

Related information

[Open File dialog](#) on page 71

[Preferences dialog](#) on page 73

3.2.3.1 Source view

The **Source** view displays the LISA source code of components. It can also display other text files.

The source text editor features:

- Similar operation to common Microsoft Windows text editors.
- Standard copy and paste operations on selected text, including with an external text editor.
- Undo/redo operations. Text changes can be undone by using **Ctrl-Z** or **Edit > Undo**. Repeat text changes with **Ctrl-Y** or **Edit > Redo**.
- Syntax highlighting for LISA, C++, HTML, Makefiles, project (*.sgproj) and repository (*.sgrepo) files.
- Auto-indenting and brace matching. Indenting uses four spaces not single tab characters.
- Auto-completion for LISA source. If you type a delimiter such as “.” or “:”, a list box with appropriate components, ports, or behaviors appears. Icons indicate master and slave ports.
- Call hint functionality. If you type a delimiter such as “(”, a tooltip appears with either a component constructor or behavior prototype, depending on the context. Enable call hints by enabling tooltips in the **Appearance** pane of the **Preferences** dialog.



Every time System Canvas parses a LISA file, it updates lexical information for auto-completion and call hint functionality. This occurs, for example, when switching between the views.

3.2.3.2 Source view context menu

The **Source** view context menu lists text operations.

Undo

Undo the last change.

Redo

Undo the last undo.

Cut

Cut the selected text.

Copy

Copy the selected text.

Paste

Paste text from the global clipboard.

Delete

Delete the selected text.

Select All

Selects all of the text in the window.

3.2.3.3 Block Diagram view

The **Block Diagram** view displays a graphical representation of the system. It provides a rapid way to create and configure components or systems consisting of multiple components.

This view supports copy and paste operations on selected components, connections, labels, and self ports:

- Use the cursor to draw a bounding rectangle around the box.
- Press and hold shift while clicking on the components to copy.

Copied components will have different names. To copy connections, select both ends of the connection.



Changes made in one view immediately affect the other view.

Open files have a named workspace tab at the top of the **Workspace** window. An asterisk after the name indicates unsaved changes. A question mark means that the file is not part of the project.

Click the right mouse button in the workspace to open the context menu for the view.

Displaying the block diagram fails if:

- The file is not a LISA file.
- The syntax of the LISA file is incorrect.
- The LISA file contains more than one component.
- The LISA file contains a protocol.

3.2.3.4 Block Diagram view context menu

The **Block Diagram** view context menu lists object operations.

Open Component

Open a new workspace tab for the selected component.

Delete

Delete the object under the mouse pointer.

Add Port...

Add a port to the component.

Mirror Self Port

Mirror the port image.

Expand Port

For a port array, display all of the individual port elements.

Collapse Port

For a port array, hide the individual port elements.

Hide Port

Disable the selected port and make it invisible.

Hide All Unconnected Ports

Hide all ports that are not connected to a component.

Show/Hide Ports of Protocol Types...

Hide all ports that use a specified protocol.

Show All Ports

Show all ports of the component.

Autoroute connection

Redraw the selected connection.

Documentation

Open the documentation for the selected component.

Object Properties

Open the object properties dialog.

3.2.4 Component window

This section describes the **Component** window, which lists the available components and their protocols and libraries.

3.2.4.1 Component window views

The **Component** window has view tabs.

Components

The components, and their version numbers, types, and file locations. Drag and drop to place in the block diagram. Double click to open in the workspace.

Protocols

The protocols of these components, and their file locations. Double click to open in the workspace.

Files

The project files, in a fully expanded file tree with the project file as the root. Double click to open in the workspace. The project file can contain LISA files and component repositories. A repository can itself contain a repository.



Note

The order of file processing is from the top to the bottom. To move objects:

- Select and use **Up** and **Down** in the context menu, or use **Alt + Arrow Up** or **Alt + Arrow Down**.
- Drag and drop.

3.2.4.2 Component window context menu

The **Component** window context menu lists file operations and a documentation link.

Open

Open the associated file.

Add...

Add a repository, component or protocol file, or a library.

Add New...

Add a new file.

Add Directory...

Add an include path to be used by the compiler (**Files** tab only). To simplify navigation, the add dialog also shows the filename.

Remove

Remove an item.

Up

Move a file up the file list (**Files** tab only).

Down

Move a file down the file list (**Files** tab only).

Reload

Reload a component or protocol.

Refresh Component List

Refresh the entire component list.

Documentation

Open the documentation for the component.

Properties

Show the properties of the item.

3.2.5 Output window

The **Output** window displays the build or script command output.

The left side of the window has controls:

First

Go to the first message.

Previous

Go to the previous message.

Stop

Do not scroll automatically.

Next

Go to the next message.

Last

Go to the last message.

The right side of the window has controls:

Scroll bar

Move up and down in the output.

Stick

Force the window to show the latest output, at the bottom.

3.2.6 System Canvas dialogs

This section describes the dialog boxes of System Canvas.

3.2.6.1 Add Existing Files and Add New File dialogs (Component window)

This section describes these dialogs that add components, protocols, libraries, repositories, or source code to a project.

Related information

[Add Files dialog \(Project menu\)](#) on page 59

[New File dialog \(File menu\)](#) on page 70

3.2.6.1.1 Displaying the Add Existing Files and Add New File dialogs (Component window)

This section describes how to display dialogs that add components, protocols, libraries, repositories, or source code to a project.

Procedure

Display a dialog by right-clicking in the **Component** window and selecting from the context menu:

- **Add.**
- **Add New.**

3.2.6.1.2 Using the Add Existing Files and Add New File dialogs (Component window)

This section describes how to add a file using the **Component** window context menu.

Procedure

1. Select the **Components**, **Protocols**, or **Files** tab in the **Component** window.
To add a file at the top level of the file list, select the top entry. To add a file to an existing repository in the file list, select the repository.

2. Right-click in the **Component** window and select **Add** or **Add New** from the context menu.
- | Option | Description |
|----------------|--|
| Add | In the Add Existing Files dialog, go to the file and select it. |
| Add New | In the Add New File dialog, go to the directory to contain the file and enter the name. |

Save time with the **Recently selected files** drop-down list. To remove a file, mouse over it and press **Delete**, or right-click and select **Remove from list** from the context menu.

3. Click **Open** to add the file and close the dialog.

Next steps

Library files, those with `.lib` or `.a` extensions, need build actions and a platform.

Related information

[File/Path Properties dialog](#) on page 66

3.2.6.1.3 Using environment variables in filepaths

Environment variables in filepaths enable switching to new repository versions without modifying the project.

About this task

For example, using `$(PVLIB_HOME)/etc/sglib.sgrepo` as the reference to the components of the Fast Models Portfolio enables migration to future versions of the library by modifying environment variable `PVLIB_HOME`.



On Microsoft Windows, Unix syntax is valid for environment variables and paths, for example `$PVLIB_HOME/etc/my.sgrepo`.

Edit a filepath through the **File Properties** dialog:

Procedure

1. Select the file and click select **Properties** from the context menu.

2. Edit the **File** entry to modify the filepath.

Related information

[File/Path Properties dialog](#) on page 66

3.2.6.1.4 Assigning platforms and compilers for libraries

This section describes how to set the operating system that a library is for, and the compiler that built it.

Procedure

Use the **File Properties** dialog to specify the operating system and compilers by checking the appropriate boxes in the **Supported platforms** pane.

Microsoft Visual Studio distinguishes between debug and release versions.

Related information

[File/Path Properties dialog](#) on page 66

[Project Settings dialog](#) on page 78

3.2.6.2 Add Files dialog (Project menu)

Add files to a project with this dialog.

Select **Add File** from the **Project** menu to add a new file to the project.

The behavior of this dialog is identical to that of the **Add Existing Files** dialog.

To create a new file from code in the **Source** view, select **Add Current File** from the **Projects** menu to add the file to the project. No dialog appears.



Save time with the **Recently selected files** drop-down list. To remove a file, mouse over it and press **Delete**, or right-click and select **Remove from list** from the context menu.

Related information

[Add Existing Files and Add New File dialogs \(Component window\)](#) on page 57

[File/Path Properties dialog](#) on page 66

3.2.6.3 Add Connection dialog

This dialog adds a connection to a component port.

To open the dialog:

1. Select a component port.
2. Display the **Port Properties** dialog by selecting **Object Properties** from the context menu or from the **Object** menu.
3. Click the **Add Connection** button.

The enabled fields for the dialog depend on whether a slave or master was displayed in the Port Properties dialog.



This dialog also appears if you use the cursor in connect mode to connect two ports in the block diagram and one or more of the ports is a port array.

Related information

[Edit Connection dialog](#) on page 65

3.2.6.4 Component Instance Properties dialog

This dialog displays the properties of a component.

To open the dialog, select a component in the block diagram, and click on the **Properties** button in the toolbar or select **Object Properties** from the **Object** menu.

General

The component name, instance name, filename and path, and repository.

The **Instance name** field is editable.



To view the properties of the top-level component, double-click in an area of the workspace that does not contain a component.

Properties

All properties for the component. If the properties are not editable, the tab says **Properties (read only)**.

If the property is a Boolean variable, a checkbox appears next to it.

Parameters

All editable parameters for this component. Enter a new value in the **Value** edit box.

The following controls are present:

Parameter name

The parameters for this component.

Value

Select a parameter and then click the text box in the **Value** column to set the default value for the parameter.

Integer parameters in decimal format can contain binary multiplication suffixes. These left-shift the bits in parameter value by the corresponding power of two.

Table 3-2: Suffixes for parameter values

Suffix	Name	Multiplier
K	Kilo	2^{10}
M	Mega	2^{20}
G	Giga	2^{30}
T	Tera	2^{40}
P	Peta	2^{50}

Ports

All the ports in the component.

For port arrays, display all of the individual ports or only the port array name by selecting **Show as Expanded** or **Collapsed**.

The properties of individual ports are editable:

1. Select a port from the list.
2. Click **Edit** and change the properties of the port.
3. Click **OK** to save the changes.



If you click **OK**, the changes apply immediately.

Enable/disable individual ports with the checkboxes:

- Click **Show selected ports** to display the checked ports.
- Click **Hide selected ports** to hide the checked ports.



Hiding the top level of a port array hides all of the individual ports but they retain their check mark setting.

Methods

All the behaviors (component functions) that the component implements.

Related information

[Component Model Properties dialog for the system](#) on page 62

[Component Properties dialog for a library component](#) on page 64

[Label Properties dialog](#) on page 69

[Port Properties dialog](#) on page 72

[Protocol Properties dialog](#) on page 84

[Self Port dialog](#) on page 86

3.2.6.5 Component Model Properties dialog for the system

This dialog displays the properties for the system.

To open the dialog, select a blank area in the block diagram, right-click and select **Object Properties** from the context menu to display the properties for the system or select **Object Properties** from the **Object** menu.

General

The system name, filename and path, and repository.

The **Component name** field is editable.

Properties

If the property is a Boolean variable, a checkbox appears next to it.

Changes in these dialogs alter the LISA code in the model.

Double-click in the **Value** column to change the property.

Table 3-3: Component properties

Property	ID	Default	Description
Component name	component_name	""	A string containing the name for the component.
Component category	component_type	""	A string describing the type of component. This can be "Processor", "Bus", "Memory", "System", or any free-form category text.
Component description	description	""	A textual component description.
Component documentation	documentation_file	""	A filepath or an HTTP link to documentation. Supported file formats are PDF, TXT, and HTML.
Executes software	executes_software	0	The component executes software and can load application files. 1 for processor-like components, 0 for other components.
Hidden	hidden	0	1 for components hidden from the Component window. Otherwise, hidden components behave exactly as normal components, and they do appear in the Workspace window.

Property	ID	Default	Description
Has CADI interface	has_cadi	1	1 for components with a CADI interface, permitting connection to the target with a CADI-compliant debugger. 0 for components with no CADI interface.
Icon pixmap file	icon_file	""	The XPM file that contains the system icon.
License feature	license_feature	""	The license feature string required to run this system model.
Load file extension	loadfile_extension	""	The application filename extension for this target. Example: ".elf" or ".hex".
Small icon pixmap file	small_icon_file	""	The XPM file that contains the 12x12 pixel system icon.
Component version	version	"1.0"	The version of the component.

Parameters

Parameter name

The parameters for this component.

Value

Select a parameter and then click the text box in the **Value** column to set the default value. Integer parameters in decimal format can contain binary multiplication suffixes. These left-shift the bits in parameter value by the corresponding power of two.

Table 3-4: Suffixes for parameter values

Suffix	Name	Multiplier
K	Kilo	2^{10}
M	Mega	2^{20}
G	Giga	2^{30}
T	Tera	2^{40}
P	Peta	2^{50}

Parameter ID in LISA code

The LISA ID for the component parameters.

Add

Click to add a new parameter.

Edit

Select a parameter and then click to change the name.

Delete

Select a parameter and then click to delete it.

Ports

All external ports.

If a port contains an array of ports, the **Size** column displays the number of ports in the array.

Enable/disable individual ports with the checkboxes:

- Click **Show selected ports** to display the checked ports.
- Click **Hide selected ports** to hide the checked ports.

Methods

The available LISA prototypes. The list is for reference only. It is not editable.

3.2.6.6 Component Properties dialog for a library component

This dialog displays the properties of a library component.

To open the dialog, select a component from the **Components** list, and right-click and select **Properties** from the context menu or select **Object Properties** from the **Object** menu.

General

Component name

The name of the component.

Type

The component category, for example `core` or `peripheral`.

Version

The revision number for the component.

File

The file that defines the component.

Repository

The repository that contains the component.

Description

Information about the component.

Properties (read only)

All the usable properties of the component.



A valid `license_feature` string allows this component to work in a model.

Parameters (read only)

All the parameters for the component.

Ports (read only)

All the ports in the component.



No port arrays are expandable here.

Methods

The LISA prototypes of the methods, that is, behaviors, of the component. The list is for reference only. It is not editable.

3.2.6.7 Connection Properties dialog

This dialog displays port connection properties.

To open the dialog, double click on a connection between components in the workspace.

Name

The name of the port.

Type

The type of port and the protocol.

To change the address mapping, click **Master Port Properties** or **Slave Port Properties**.

Related information

[Port Properties dialog](#) on page 72

3.2.6.8 Edit Connection dialog

This dialog controls port connection properties.

To open the dialog and change the connected port or the address mapping, select a connection from the **Port Properties** dialog and click **Edit Connection....**

Component

For a slave port, the source component is editable. For a master port, the destination component is editable.

Port

For a slave port, the master port is editable. For a master port, the slave port is editable.

Array index

For port arrays, an index value for the element to use.

Enable address mapping

Set the port address range with the **Start** and **End** boxes.

Start

The start address for the port.

End

The end address for the port.

Size

The size of the address region. Given the **Start** and **End** values, System Canvas calculates this value.

OK/Cancel

Click **OK** to modify the connection. Click **Cancel** to close the dialog without changing the connection.

LISA statement

The code equivalent to the address range.

3.2.6.9 File/Path Properties dialog

This dialog displays properties for the file and controls build and compile options.



- On Microsoft Windows, the / and \ directory separators both appear as /. This simplification does not affect operation.
 - Avoid using Japanese or Korean characters in filepaths. They can cause failure to find libraries.
-

Select a component from the **Component** window **Files** tab, right click on it to open the context menu, then click **Properties** to display the dialog.

General**File or path**

The name of the file.



The **File Properties** dialog is modeless. You can select a different file without closing the dialog. A warning message prompts to save any changes.

Absolute path

The full path to the file.

Repository

The repository file that contains this component entry.

Type

A brief description of the component type.

Info

The status of the file. For example, `file does not exist`.

Supported platforms

Select the platforms that the component supports:

- Linux64.
- Win64 (release runtime library).
- Win64D (debug runtime library).

Compiler

Select the compiler for this component from the drop-down list:

- No preference.
- Specific Microsoft Visual C++ compiler.
- gcc version found in \$PATH at compile time.
- Specific gcc version.

Build actions

Default actions depending on file extension

.lisa

A LISA source file that SimGen parses.

.c .cpp .cxx

A C or C++ source file that the compiler compiles.

.a .o

A Linux object file that SimGen links to.

.lib .obj

A Microsoft Windows object file that SimGen links to.

.sgproj

A project file that SimGen parses.

.sgrepo

A component repository file that SimGen parses.

directory_path/

An include directory for the search path that the compiler uses. The trailing slash identifies it as an include path. For example, to add the directory that contains the *.sgproj file, specify ./ (dot slash), not only the dot.

All other files

Copy a *deploy* file to the build directory.



Simulation Generator (SimGen) is one of the Fast Models tools.

Ignore

Exclude the selected file from *build and deploy*. This feature can be useful for examples, notes, or temporarily disabled files.

Customize actions

Ignore the file extension. Specify the actions with the check boxes:

LISA - input file passed to Simulator Generator as LISA

System Canvas passes the file to SimGen as a LISA file. Do not use this option for non-LISA files.

Compile - compile as C/C++ source code

To compile a file as C/C++ code during the build process, add it to this list of files.

Link - input file for linker

Link the file with the object code during the build process.

Deploy - copy to build directory

Copy the file into the build directory. This option can, for example, add dynamic link libraries for running the generated system model.

Include path - add the file's path to additional include directories

Add the path of the parent directory that holds the file to the list of include directories for the compiler.

Library path - add the file's path to additional library directories

Add the path of the parent directory that holds the file to the list of library directories for the compiler.

Related information

[Project parameter IDs](#) on page 81

3.2.6.10 Find and Replace dialogs

This dialog enables searching for and replacement of text in an editor window.

The **Find** dialog and the **Find and Replace** dialog are essentially the same dialog in two modes, find only, and find and replace. Switch modes by clicking the **Find mode** or **Find and replace mode** buttons. By default, matches are case sensitive but matches can appear as part of longer words. Change the default behavior by setting or clearing the relevant checkboxes in the dialog.

Open the **Find** dialog by clicking **Search > Find...** in the main menu. Type the text to find in the box and click the **Find Next** or **Find Previous** buttons to search upwards or downwards from the

current cursor position. You can re-use previous search terms by clicking on the drop-down arrow on the right of the text entry box.

Open the **Find and Replace** dialog by clicking **Search > Replace** in the main menu. Replace the current match with new text by clicking the **Replace** button, or all matches by clicking the **Replace All** button. You can re-use previous find or replacement terms by clicking on the drop-down arrow on the right of the text entry boxes.

Find and Replace mode is only available if the current active window is a source editor. In that mode, additional replace controls appear. The dialog is modeless, so you can change views without closing it.

3.2.6.11 Label Properties dialog

This dialog controls the text and display properties for a label.

Double-click on a label to display the dialog. Select **Add Label** from the **Object** menu to add a label to the component.

Label

Specify the text to display on the label.

Font

The text font. Click **Select Font...** to change it.

Select Text Color...

Click to select a color for the text.

Select Background Color...

Click to select the background color for the label.

Check Transparent Background

Check to make objects behind the label visible, and to ignore the background color setting.

Horizontal

Set the horizontal justification for the label text.

Vertical

Set the vertical justification for the label text.

Rotation

Set the orientation for the label.

Frame Thickness

Set the thickness of the label border.

Shadow Thickness

Set the thickness of the label drop shadow.

Display on Top

Check to display the label on top of any components below it.

Use these settings as default

Check to use the current settings as the default settings for any new labels.

3.2.6.12 New File dialog (File menu)

This dialog creates new projects and LISA source files.

To display the dialog, select **New File** from the **File** menu or click the **New** button.

Look in

Specify the directory for the new file.

File name

Enter the name for the new file.

File type

- If a project is not open, this box displays `.sgproj` by default to create a project.
- If a project is open, this box displays `.lisa` by default to create a LISA source file.

Add to

Active for non-`.sgproj` files. Check to enable the adding of the created file to the open project.

Select

Click to accept the name and path.

If the new file is of type `.sgproj`, System Canvas prompts for the top level LISA file.



Save time with the **Recently selected files** drop-down list. To remove a file, mouse over it and press **Delete**, or right-click and select **Remove from list** from the context menu.

Related information

[Select Top Component LISA File dialog](#) on page 71

3.2.6.13 New project dialogs

This section describes the dialogs that create new projects.

3.2.6.13.1 New Project dialog

This dialog creates new projects.

To display the dialog, select **New Project** from the **File** menu.

Look in

Specify the directory for the new project file.

File name

Enter the name for the new project.

If you select an existing file, the new project replaces the existing project.

File type

The default type for Fast Models projects is `.sgproj`.

Select

Click to accept the name and path.

For existing projects, System Canvas queries the replacement of the existing project with a new project of the same name.

After you click **Select**, the **Select Top Component LISA File** dialog appears.



The project file includes the path to the model repositories from the **Default Model Repositories** pane of the **Preferences** dialog.

Related information

[Preferences - Default Model Repository group](#) on page 76

3.2.6.13.2 Select Top Component LISA File dialog

This dialog controls the name of the top-level LISA file for a project.

After clicking **Select** in the **New Project** dialog, this dialog appears. By default, the filename for the top-level LISA file is the same as the project name. You can, however, specify a different name in this dialog.

3.2.6.14 Open File dialog

This dialog opens project files, LISA source files, and text documents.

To display the dialog:

- Select **Open File** from the **File** menu.
- Select a file in the **Component** window and select **Open** from the context menu.

Look in

Specify the directory.

File name

Enter the name of the file.

File type

Select the type of file.

Open

Click to open the file.

Open project file as text in source editor

Active for non-`.lisa` and for `.sgproj` files. Check to enable the opening of the file as plain text in the **Source** window.



- Use this option, for example, for a `.sgproj` file to manually edit the list of repositories. Such changes take effect after you close and reopen the file.
 - If you select a `.sgproj` file without checking this box, the project loads.
-



Save time with the **Recently selected files** drop-down list. To remove a file, mouse over it and press **Delete**, or right-click and select **Remove from list** from the context menu.

3.2.6.15 Port Properties dialog

This dialog controls port properties.

To display the **Port Properties** dialog, select a port or a connection.

- Select a component port in the **Block Diagram** view and:
 - Double-click on the port.
 - Click the **Properties** button.
 - Select **Object Properties** from the **Object** menu.
 - Right-click and select **Object Properties** from the context menu.
- Select a connection in the **Block Diagram** view and double-click to display the **Connection Properties** dialog. To display the **Port Properties** dialog:
 - Click the **Master Port Properties** button to display the properties for the master port.
 - Click the **Slave Port Properties** button to display the properties for the slave port.

Name

The name of the port.

Type

The type of port and the protocol.

Array size

For port arrays, the number of elements.

Show connections for port array index

For port arrays, enter an index value in the integer box to display only that element.

For individual ports of port arrays, this box displays the index for the selected port.

Port connections

- Sort the connections: click on the column headings.
- Change the connected port or address mapping: select a connection and click **Edit Connection**.
- Add a connection: select a connection and click **Add Connection**.
- Delete a connection: select it and click **Remove**.
- Change the priority of a single connection: select it and click **Increase Priority** or **Decrease Priority**.

Related information

[Connection Properties dialog](#) on page 65

3.2.6.16 Preferences dialog

This section describes the **Preferences** dialog (**File > Preferences**), which configures the working environment of System Canvas.

3.2.6.16.1 Preferences - Appearance group

This group sets the appearance of System Canvas.

Show Tool Tips

Display all tool tips.

Display tool bar text labels

Display the status bar labels.

Word wrap in source windows

Wrap long lines to display them within the source window.

Show splash screen on startup

Show the splash screen on startup.

Reload recent layout on startup

Reload the layout settings from the last modified project.

Recent files and directories

Set the number of directories and files shown in System Canvas file dialogs and menus, up to 32 directories and 16 files.

3.2.6.16.2 Preferences - Applications group

This group sets the application paths.



- On Microsoft Windows, environment variables appear as `$MAXxxxx_HOME`. You can use this format instead of `%MAXxxxx_HOME%`.
- The different path specifications enable the use of different versions of Model Debugger and provide more flexibility for installing Model Debugger separately from System Canvas.

Simulator Generator Executable

SimGen

Set the path to the `simgen.exe` file.

Command arguments

Set additional command-line options.

Model Debugger Executable

Model Debugger

Set the path to the Model Debugger executable.

Command arguments

Set additional command-line options.

Model Shell Executable

Model Shell

Set the path to the Model Shell executable.

Command arguments

Set additional command-line options.

Run Model Shell asynchronously with output to console in separate window

Check to enable starting a separate Model Shell instance with its own output window.



To start the simulation, select the **Run in Model Shell** entry on the **Projects** menu.

Path to Microsoft Visual Studio application 'devenv.com'

Select the path to the Microsoft Visual Studio `devenv.com` file. This application is the development environment and builds the model.

Reset to Defaults

Click to reset the application paths.

Apply

Click to save the changes.

Run Model Shell asynchronously

On Linux, check to use the command line:

```
xterm -e <Model Shell Executable> optional_command_arguments_list -m model.so
```

Host Debugger Command Line

On Linux, set the command-line options. The default text is:

```
xterm -e gdb --args %ISIM%
```

where %ISIM% is a placeholder for the isim_system executable file.



On Linux, select the GCC compiler to build the model by using the SimGen command-line option `--gcc-path`.

Related information

[SimGen command-line options](#) on page 15

[Model Debugger](#) on page 87

[Model Shell](#) on page 152

[Project Settings dialog](#) on page 78

3.2.6.16.3 Preferences - External Tools group

This group sets the tools that display the documentation.

use operating system file associations

Check to inactivate the external tool edit fields and buttons. Clear to activate them.



This checkbox is not available on Linux.

3.2.6.16.4 Preferences - Fonts group

This group sets the application fonts.

Application

The application font.

Base fixed font

The **Source** view font.

Block Diagram Component Name

The component title block font.

Fonts depend on \$DISPLAY variable

Check to use the font set in the \$DISPLAY variable.

Reset to base size

Reset all font sizes to the selected value.

Reset to defaults

Click to reset the fonts to the factory settings.

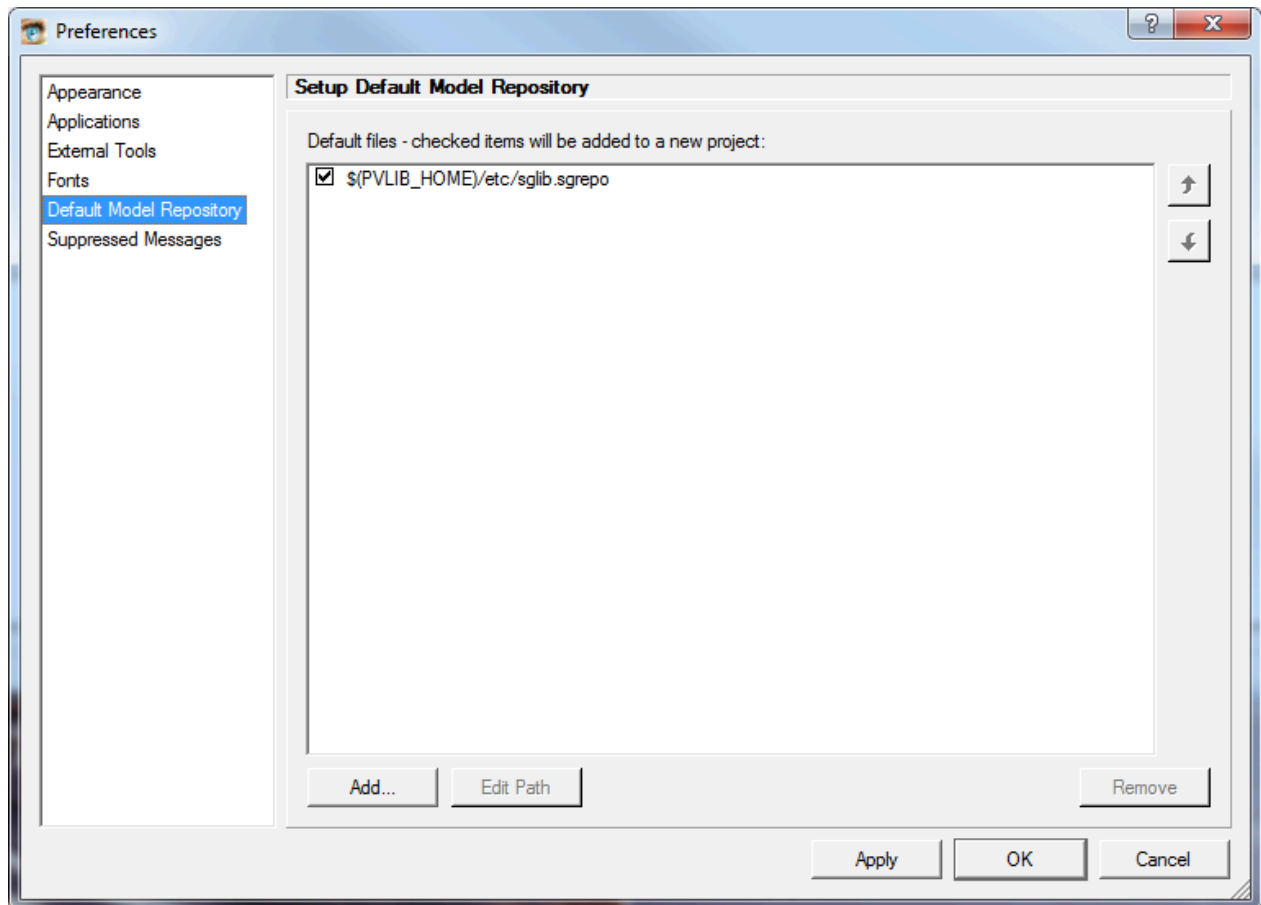


If non-Latin characters are used in LISA code, the base fixed font must support them. The default font might not support non-Latin characters.

3.2.6.16.5 Preferences - Default Model Repository group

This group sets the default model repositories for new projects.

Figure 3-21: Preferences dialog, Setup Default Model Repository



To incorporate components into a system, System Canvas requires information about them, such as their ports, protocols, and library dependencies. For convenience, model repositories, such as `sglib.sgrepo`, group multiple components together and specify the location of the LISA files and the libraries that are needed to build them.

Default repositories are added by default to new projects. To add a repository to an existing project, use the Component window context menu.



Note

To enable the immediate use of models in new projects, System Canvas has a default entry `$(PVLIB_HOME)/etc/sglib.sgrepo`. This entry is not deletable, but clearing the checkbox deactivates it.

Add

Click **Add** to open a file selection dialog and add a new `.sgrepo` repository file to the list.

Select a directory to add all of the repositories in that directory to the list of repositories.

Edit Path

Select a repository and click **Edit** to edit the path to it.

The path to the default repository `$(PVLIB_HOME)/etc/sglib.sgrepo` is not editable.

Remove

Select a repository and click **Remove** to exclude the selected repository from new projects. This does not affect the repository itself.

The default repository `$(PVLIB_HOME)/etc/sglib.sgrepo` is not deletable.

File checkboxes

Check to automatically include the repository in new projects. Clear to prevent automatic inclusion, but to keep the path to the repository available.

Up/Down

Use the **Up** and **Down** buttons to change the order of repositories. File processing follows the repository order.

Related information

[Repository files](#) on page 10

3.2.6.16.6 Preferences - Suppressed messages group

This group lists the suppressed messages and controls their re-enabling.

Enable selected messages

Click to enable selected suppressed messages.

3.2.6.17 Project Settings dialog

This section describes the dialog (**Project > Project Settings**, or **Settings** toolbar button) that sets the project settings and customizes the generation process.

3.2.6.17.1 Project top-level settings

This part of the dialog sets the project build options.

Top level component

- Enter a name into the **Top Level Component** edit box.
- Click **Use Current** to set the component in the workspace as the top component.
- Click **Select From List** to open a dialog and select any component in the system.

Configuration

- Select an entry from the drop-down list to use an existing configuration.

- Click **Add New** to create a new configuration. A dialog prompts for the name and a description. Use **Copy values from** to select a configuration to copy the settings values from. This can be an existing configuration or a default set of configuration settings.
- Click **Delete** to delete the selected configuration from the list.

The values default to those of the active configuration.

Selecting a configuration in this dialog does not set the configuration in the **Select Active Project Configuration** drop-down box on the main window. System Canvas stores the configuration set in this dialog in the project file, to use if you specify it for a build. You can use this control to specify all of the configurations for a project, to simplify switching active configurations.



Note

If you build systems on Microsoft Windows workstations, other Microsoft Windows workstations need the matching support libraries to run the systems:

Debug builds

Microsoft Visual Studio.

Release builds

Microsoft Visual Studio redistributable package.

3.2.6.17.2 Parameter category panel

This section describes the **Parameter category** panel, which lists parameters for the selected build, under different views.

3.2.6.17.2.1 Parameters - Category View

This view lists categories and the parameters for the selected category.

Top-level configuration details

Select the top-most category item to configure the project settings.

Table 3-5: Configuration parameters in the Category View

Control name	Parameter
Configuration name	CONFIG_NAME
Platform/Linkage	PLATFORM
Compiler	COMPILER
Configuration description	CONFIG_DESCRIPTION
Build directory	BUILD_DIR

Targets

Select the **Targets** item to configure the build target parameters.

Table 3-6: Target parameters in the Category View

Control name	Parameter
SystemC component	TARGET_SYSTEMC
SystemC component with auto-bridging	TARGET_SYSTEMC_AUTO
Link the set target to static SystemC library	USE_STATIC_SYSTEMC_LIB
SystemC integrated simulator	TARGET_SYSTEMC_ISIM
SystemC integrated CADI library	TARGET_SYSTEMC_MAXVIEW

Debugging

Select the **Debugging** item in the panel to configure the debug parameters.

Table 3-7: Debugging parameters in the Category View

Control name	Parameter
Enable model debugging	ENABLE_DEBUG_SUPPORT
Source reference	GENERATE_LINEINFO
Verbosity	VERBOSITY
Model Debugger	MODEL_DEBUGGER_COMMAND_LINE
Model Shell and ISIM	MODEL_SHELL_COMMAND_LINE
SystemC executable	SYSTEMC_EXE
SystemC arguments	SYSTEMC_COMMAND_LINE

Sim Generator

Select the **Sim Generator** item in the panel to configure the Simulation Generator parameters.

Table 3-8: Simulation Generator parameters in the Category View

Control name	Parameter
Simgen options	SIMGEN_COMMAND_LINE
Warnings as errors	SIMGEN_WARNINGS_AS_ERRORS
Using namespace std	ENABLE_NAMESPACE_STD
Make options	MAKE_OPTIONS

Compiler

Select the **Compiler** item in the panel to configure the compiler parameters.

Table 3-9: Compiler parameters in the Category View

Control name	Parameter
Pre-Compile Actions	PRE_COMPILE_EVENT
Include Directories	INCLUDE_DIRS
Preprocessor Defines	PREPROCESSOR_DEFINES
Compiler Settings	ADDITIONAL_COMPILER_SETTINGS
SCX Library Settings	ADDITIONAL_SCX_LIB_SETTINGS

Control name	Parameter
Enable pre-compiling	ENABLE_PRECOMPILE_HEADER

Linker

Select the **Linker** item in the panel to configure the linker parameters.

Table 3-10: Linker parameters in the Category View tab

Control name	Parameter
Pre-Link Actions	PRE_LINK_EVENT
Linker Settings	ADDITIONAL_LINKER_SETTINGS
Post-Build Actions	POST_BUILD_EVENT
Post-Clean Actions	POST_CLEAN_EVENT
Disable suppression of symbols	DISABLE_SYMBOL_SUPPRESSION

3.2.6.17.2.2 Parameters - List View

This view lists the parameters and their values. Reorder them by clicking on a column heading.

3.2.6.17.2.3 Parameters - Tree View

This view displays parameters in a tree structure, with expandable categories.

3.2.6.17.3 Project parameter IDs

The parameters that configure a project. These parameters can either be set in the System Canvas **Project Settings** dialog or in the sgproj file passed to SimGen.

Table 3-11: List of parameters shown in List View

Parameter ID	Parameter name	Default	Description
ADDITIONAL_COMPILER_SETTINGS	Compiler settings	-	Compiler settings. If your C++ source code uses C++14 syntax, specify <code>-std=c++14</code> in this parameter. For Microsoft Windows, consult the Visual Studio documentation.
ADDITIONAL_LINKER_SETTINGS	Linker settings	-	Linker settings. For Microsoft Windows, consult the Visual Studio documentation.
ADDITIONAL_SCX_LIB_SETTINGS	SCX library settings	-	Compiler flags specified here are added to the scx library compiler flags.
BUILD_DIR	Build directory	-	Build directory. The path can be absolute or relative to the location of the project file.

Parameter ID	Parameter name	Default	Description
COMPILER	Compiler	-	The compiler to use to build this configuration. One of the following: vc2019 Microsoft Visual Studio 2019. gcc The first gcc version in the Linux search path. gcc-7.3 GCC 7.3. gcc-9.3 GCC 9.3. gcc-10.3 GCC 10.3.
CONFIG_DESCRIPTION	Configuration description	-	Description of the configuration.
CONFIG_NAME	Configuration name	-	Name of the configuration. For example <code>Linux64-Debug-GCC-9.3</code> .
DISABLE_SYMBOL_SUPPRESSION	Disable suppression of symbols	0	If true, stop SimGen from adding the <code>-fvisibility=hidden</code> flag to mark symbols as hidden.
ENABLE_DEBUG_SUPPORT	Enable model debugging	0	Use implementation-defined debug support.
ENABLE_NAMESPACE_STD	Enable namespace std	1	Use namespace std: 1 (true) Generate using <code>namespace std</code> and place in the code. 0 (false) Specify the namespace. This setting might reduce compilation time.
ENABLE_PRECOMPILE_HEADER	Enable precompiling	0	Precompile headers if true.
GENERATE_LINEINFO	Source reference	"LISA Code (incl headers) "	Control line redirection in the generated model source code: "LISA Code" Source code. "LISA Code (incl. headers) " Source and header. "Generated Code" No line redirection at all.
INCLUDE_DIRS	Include directories	-	Additional include directories. Separate multiple entries with semicolons. Non-absolute paths are relative to the sgproj file.
MAKE_OPTIONS		-	Additional options to pass to <code>make</code> . Linux only.
MODEL_DEBUGGER_COMMAND_LINE	Model Debugger	-	Additional command line options to pass to Model Debugger.
MODEL_SHELL_COMMAND_LINE	Model Shell and ISIM	-	Additional command line options to pass to Model Shell or ISIM.

Parameter ID	Parameter name	Default	Description
PLATFORM	Platform/linkage	-	Host platform for which this configuration is valid. One of: Linux64 Linux x86-64. Linux64_armv81 Arm® AArch64 Linux. Win64 64-bit Microsoft Windows, linked against release runtime library. Win64D 64-bit Microsoft Windows, linked against debug runtime library.
POST_BUILD_EVENT	Postbuild actions	-	Commands to execute after building the model. Separate multiple entries with semicolons.
POST_CLEAN_EVENT	Post-clean actions	-	Commands to execute after the standard <code>clean</code> has completed.
PRE_COMPILE_EVENT	Precompile actions	-	Commands to execute before starting compilation. Applies to Microsoft Windows only. Separate multiple entries with semicolons.
PREPROCESSOR_DEFINES	Preprocessor defines	-	Preprocessor defines. Separate multiple entries with semicolons.
PRE_LINK_EVENT	Prelink actions	-	Commands to execute before starting linking. Applies to Microsoft Windows only. Separate multiple entries with semicolons.
SIMGEN_COMMAND_LINE	SimGen options	-	Additional command line options to pass to SimGen.
SIMGEN_WARNINGS_AS_ERRORS	Warnings as errors	"0"	If 1 (<code>true</code>), treat LISA parsing and compiler warnings as errors.
SYSTEMC_COMMAND_LINE	SystemC arguments	-	Command-line arguments for SystemC executable.
SYSTEMC_EXE	SystemC executable	-	Name of custom executable running exported SystemC model.
TARGET_SYSTEMC	SystemC component	0	If 1 (<code>true</code>), build a library with an exported SystemC component.
TARGET_SYSTEMC_AUTO	SystemC component with auto-bridging	0	If 1 (<code>true</code>), build a library with a SystemC component with auto bridging.
TARGET_SYSTEMC_ISIM	SystemC integrated simulator	0	If 1 (<code>true</code>), build a SystemC ISIM executable. Note: It is invalid to select both <code>TARGET_SYSTEMC_MAXVIEW</code> and <code>TARGET_SYSTEMC_ISIM</code> .
TARGET_SYSTEMC_MAXVIEW	SystemC integrated CADI library	0	If 1 (<code>true</code>), build a CADI system dynamic library with the SystemC scheduler for running from Model Debugger or Model Shell. Note: It is invalid to select both <code>TARGET_SYSTEMC_MAXVIEW</code> and <code>TARGET_SYSTEMC_ISIM</code> .

Parameter ID	Parameter name	Default	Description
USER_SYSTEMC_DYNLIB	User specified SystemC shared library path	NULL	Set this parameter to the full path of a static or dynamic SystemC library. This parameter overrides the default location where SimGen looks for it, which is specified by the <code>SYSTEMC_HOME</code> environment variable. For more information, see Linking against the SystemC library in the Fast Models User Guide.
USER_SYSTEMC_MAIN	User defined SystemC main file path	NULL	Set this parameter to a file that defines a custom <code>sc_main()</code> function, for an ISIM target (<code>TARGET_SYSTEMC_ISIM</code>). SimGen uses this function instead of generating a default one. This parameter must include the path relative to the build directory.
USE_STATIC_SYSTEMC_LIB	Link the set target to static SystemC library.	0	Link to a static rather than dynamic SystemC library. By default, the <code>SYSTEMC_HOME</code> environment variable specifies the location of the library, but you can override this using parameter <code>USER_SYSTEMC_DYNLIB</code> . For a CADI library, whose build target is <code>TARGET_SYSTEMC_MAXVIEW</code> , this parameter is set by default and cannot be unset.
VERBOSITY	Verbosity	"Off"	Verbosity level: "Sparse", "On", or "Off".

Related information

[Auto-bridging](#)

3.2.6.18 Protocol Properties dialog

This dialog displays the properties of protocols.

Select a protocol from the **Protocols** list, right-click on it and select **Properties** to display the properties.

Protocol name

The name of the protocol.

File

The file that defines the protocol.

Repository

The repository that contains the reference to the file path.

Description

A description dating from the addition of the file to the project.

Methods

A panel that displays the LISA prototypes of methods, or behaviors, available for the protocol. The values are for reference only. They are not editable.

Properties

A panel that displays the properties for protocol. The values are for reference only. They are not editable.

3.2.6.19 Run dialog

This dialog specifies the actions that execute to run a selected target.

There are actions for different targets, and additional options.

To display the dialog, click **Run** from the **Project** menu.

Select command to run

Select the executable to run.

Full command line

Adjust the command line that System Canvas generates, for example, add parameters or change the location of the application to load onto the executable.

Effective command line

Shows the complete command line with expanded macros and environment variables, ready for execution.

Model Debugger

Run the model in Model Debugger. The initial command line options come from project settings and user preferences.

Model Shell

Run the model with Model Shell. The initial command line options come from project settings and user preferences.

ISIM system

Run the model as an ISIM system. The initial command line options come from project settings and user preferences.

Custom

Specify the command line in **Full command line**.

Recent

Select a recent command.

Insert Placeholder Macro

Insert a macro or environment variable from drop-down list at the current cursor position in **Full command line**. System Generator expands them to build the complete command line.

%CADI%

The full absolute path of the CADI dynamic library.

%ISIM%

The full absolute path of the ISIM executable.

%BUILD_DIR%

The relative path to the build directory (relative to project path).

%DEPLOY_DIR%

The relative path to the deploy directory (identical to %BUILD_DIR%).

%PROJECT_DIR%

The full absolute path to the directory of the project.

Launch in background

Run an application asynchronously in a separate console window. Use this if the application requests user input or if the output is long.

Clear History

Remove all the recent entries from command history. This also removes corresponding items from the System Canvas main menu.

3.2.6.20 Self Port dialog

Use this dialog to add a port to the top-level component.

To display the dialog, without having anything selected in the **Block Diagram** view, click **Add Ports**, or click **Add Port** from the **Object** menu.

Instance name

The name of the port.

Array size

The number of ports, for a port array. Leave the box empty, or enter 1, for normal ports.

Protocol

The name of the protocol for the port. To display a list of protocols, click **Select...**

Type

Master port or **Slave Port**.

Attributes

- **Addressable** for bus ports.
- **Internal** for ports between subcomponents. The port is not visible if the component is added to a system.

Create LISA method templates according to selected protocol

Select an option from the drop-down list to create implementation templates for methods, or behaviors, for the selected protocol:

- Do not create method templates.
- Create only required methods. This is the default.
- Create all methods, including optional behaviors.

This creates only methods corresponding to the selected port type, that is, for either master or slave.

Editing the existing port might create new methods, but does not delete existing methods.

Mirror port image

Reverse the direction of the port image.

4. Model Debugger

Model Debugger for Fast Models is a fully retargetable debugger for scalable cluster software development. It is designed to address the requirements of SoC software developers.

Model Debugger has an easy to use GUI front end which supports:

- Source-level debugging.
- Complex breakpoints.
- Advanced symbolic register display.
- Customized window layout.

Model Debugger can connect to any *Component Architecture Debug Interface* (CADI) compliant model.

It supports full cluster debugging, and multiple instances of Model Debugger stay fully synchronized while debugging different cores running within a single system.

Model Debugger is included in the Fast Models package and in the Fast Models FVP Standard Library package. For installation information for the Fast Models package, see [Installation](#) in the *Fast Models User Guide*.

4.1 Key features

This section describes the key features of Model Debugger.

- Full simulation control on C-statement and instruction levels.
- C-source level display with syntax highlighting.
- Integrated variable browser.
- Low-level disassembly display.
- Call stack and backtrace.
- Complex register display with unlimited register groups and compound registers.
- Memory windows with support for multiple memory spaces and bit widths.
- Breakpoints on register and memory locations with complex conditions.
- Advanced search capabilities.
- Intuitive GUI with fully customizable window layout.
- Project management to store debugging sessions including window layout, open files, and breakpoints.



Fast Models targets do not support all of these features.

4.2 Retargetable debugger

Model Debugger supports completely retargetable debugging of any target that supports the CADI debug interface.

All target-related information, such as the disassembly and resources like register files and flags, is contained in the target model library. Model Debugger communicates with the target using CADI to retrieve the static target-specific information, for example a register file. It can then determine the target state and control execution.

Model Debugger can attach to and debug target components that are part of Fast Models systems. It can also debug any stand-alone target model library that has a CADI interface.

4.3 Cluster debugging

Model Debugger supports cluster debugging and can be attached to an arbitrary number of core targets in a cluster system.

If attached to a processor model, Model Debugger automatically loads the debug information for the respective target processor and colors all views.

Model Debugger can save the appearance for each target that is based on project files. Information that can be saved and restored includes:

- Debugger geometry.
- Complete layout and geometry of all views.
- Breakpoints.

Related information

[Model Debugger sessions](#) on page 103

4.4 Using Model Debugger

This chapter describes how to use Model Debugger.

4.4.1 Launching Model Debugger from the command line

This section describes how to launch Model Debugger, and how to start models and connect to them from it.

Related information

[Preferences dialog box](#) on page 147

4.4.1.1 Model Debugger command-line options

To launch Model Debugger from the command line, type `modeldebugger`, with options and arguments.

Table 4-1: Command-line options

Short	Long option	Description
	<code>--cyclelimit</code> <code><CYCLES></code>	Set a limit on the number of system cycles for a simulation in non-GUI mode. Use the <code>--nogui</code> option to enable this option.
	<code>--debug-isim</code> <code><ISIM_SYSTEM></code>	Start <i>ISIM_SYSTEM</i> and connect Model Debugger to it.
	<code>--debug-sysc</code> <code><SYSTEMC></code>	Start <i>SYSTEMC</i> simulation and connect Model Debugger to it.
-T	<code>--timelimit</code> <code><TIME></code>	Set a time limit for the simulation in non-GUI mode. Use the <code>--nogui</code> option to enable this option.
-a	<code>--application</code> <code><FILE></code>	Load the application <i>FILE</i> . To target a specific core in cluster systems, give the path to the core instance. For example, <code>foo.bar.core=dhrystone.axf</code> . Note: For the application file to be displayed in the Model Debugger Select Targets dialog, you must also specify the remote CADI simulation. See the <code>-c</code> option.
-C	<code>--parameter</code> <code><PARAMETER></code>	Set a single model parameter. Specify it as a path naming the instance and the parameter name using dot separators. For example, <code>foo.bar.inst.parameter=1000</code> . To set multiple parameters, use <code>--config-file</code> .
-c	<code>--connect</code> <code><SIMULATION_ID></code>	Connect to a remote CADI simulation. <i>SIMULATION_ID</i> specifies the simulation to connect to. <code>--list-connections</code> displays the list of available connections.
-E	<code>--enable-verbose</code> <code><MSGCLASS></code>	Use verbose messages if displaying message text for message classes <i>MSGCLASS</i> . Without an argument, this option lists all classes.
-e	<code>--env-connect</code>	Connect to remote CADI simulation using the following environment variables: CADI_CLIENTPORT_TCP Port number. CADI_INSTANCEID Component instance name. CADI_APPLICATIONFILENAME Application file name.

Short	Long option	Description
-F	--stdout-to-file <FILE>	Print all application output to FILE instead of the StdIO pane in the Output Window .
-f	--config-file <FILE>	Use model parameters from the configuration file <i>FILE</i> .
-h	--help	Print the available options and exit.
-i	--instance	Specify the instance.
-L	--cadi-log	Log all CADI calls into an XML logfile <i>CADILog-nnnn.xml</i> , where <i>nnnn</i> is a set of four digits. The logfile is created in the same directory as the model.
	--list-connections	List all possible connections to remote CADI simulations on the local machine and exit afterwards. Each simulation has a unique simulation ID.
	--list-instances	List target instances.
-l	--list-params	List target instances and their parameters.
-m	--model <FILE>	Load the specified model.
-N	--nogui	Run the simulation without displaying the GUI.
-n	--no-params-dialog	Do not display the parameter configuration dialog at startup.
-O	--stdout-to-stdout	Print all application output to stdout instead of the StdIO pane in the Output Window .
-p	--project <FILE>	Load the project file <i>FILE</i> .
-q	--quiet	Suppress all Model Debugger output.
	--plugin	Load specific trace plug-ins. The equivalent environment variable is FM_TRACE_PLUGINS .
-V	--verbose	Equivalent to --enable-verbose "MaxView".
-v	--version	Print the tool version and exit.
-x	--force-reg-hex	Force registers with initial integer display to be hexadecimal format instead.
-Y	--layout <FILE>	Load the layout file <i>FILE</i> .
-y	--no-target-dialog	Suppress the Select Target dialog box that normally appears when a model is loaded. Model Debugger automatically connects to targets that have the executes_software flag set. From the GUI, you can use the Other Settings checkbox in the Preferences dialog box to suppress the Select Target dialog box.

4.4.1.2 String syntax

Filenames and other strings that are specified when starting Model Debugger from the command line must be within double quotes if they contain whitespace.

For example:

```
modeldebugger -a "cluster0.cpu0=my application.axf"
```

There is no requirement to use quotes if the string has no spaces. Both of these forms are valid:

```
modeldebugger -a cluster0.cpu0=dhrystone.axf
```

```
modeldebugger -a "cluster0.cpu0=dhrystone.axf"
```

4.4.1.3 Configuration file syntax

You can configure a model that you start from the command line with Model Debugger by including a reference to an optional plain text configuration file. Each line of the configuration file must contain the name of the component instance, the parameter to be modified, and its value.

Use this format:

instance.parameter=value

The *instance* can be a hierarchical path, with each level separated by a dot "." character. If *value* is a string, additional formatting rules might apply.

You can include comment lines beginning with a # character in your configuration file. Boolean values can be set using either *true/false* or *1/0*, for example:

```
# Disable semihosting using true/false syntax
coretile.core.semihosting-enable=false
#
# Enable VFP at reset using 1/0 syntax
coretile.core.vfp-enable_at_reset=1
#
# Set the baud rate for UART 0
baseboard.uart_0.baud_rate=0x4800
```

Related information

[String syntax](#) on page 90

4.4.1.4 Running Model Debugger without a GUI

Model Debugger can be run without a Graphical User Interface (GUI). This mode is triggered by the command-line option `--nogui`.

To limit the duration of a simulation in non-GUI mode, specify the amount of seconds or system cycles using the command-line options:

- `--timelimit time_in_seconds`
- `--cyclelimit number_of_system_cycles`

The `--timelimit` and `--cyclelimit` options are only available in `--nogui` mode.

4.4.2 Launching Model Debugger from System Canvas

This section describes how to launch Model Debugger from System Canvas.

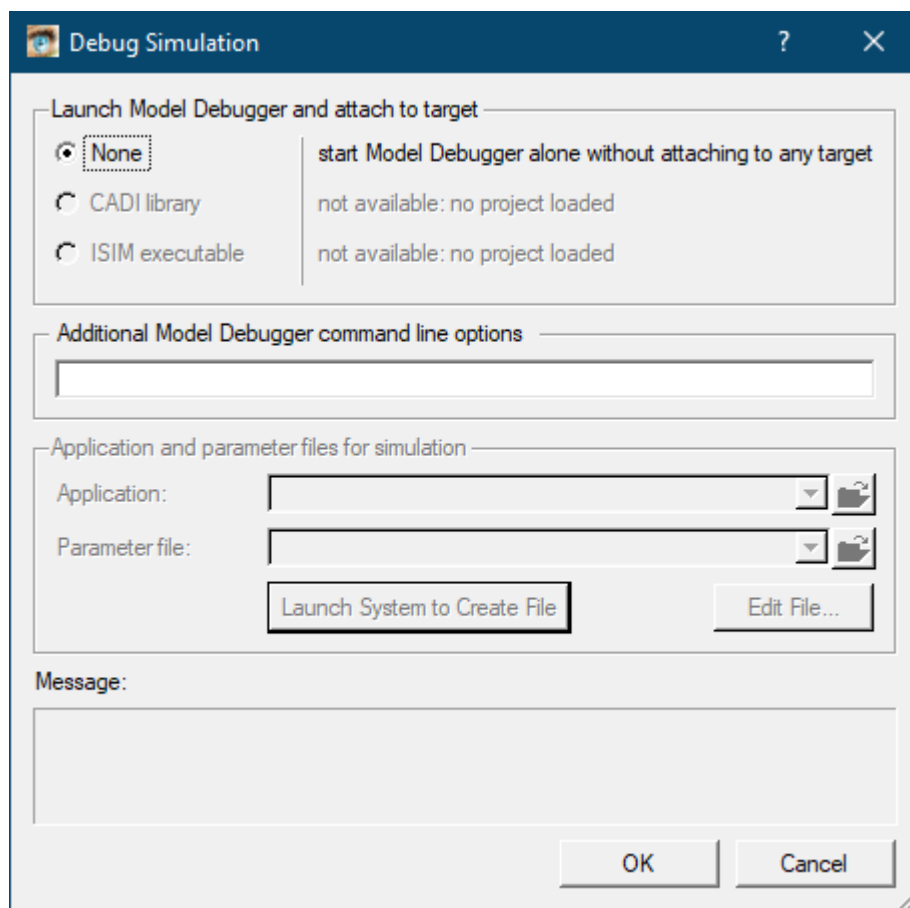
Procedure

1. Open the **Debug Simulation** dialog box in either of the following ways:
 - Click the **Debug** button on the toolbar.
 - Select **main menu > Project > Launch Model Debugger**.

The **Debug Simulation** dialog box appears.

If you have loaded a project, the target option and the **Application** field are available for use.

Figure 4-1: Debug Simulation dialog box



2. Click **OK**.
Model Debugger starts.

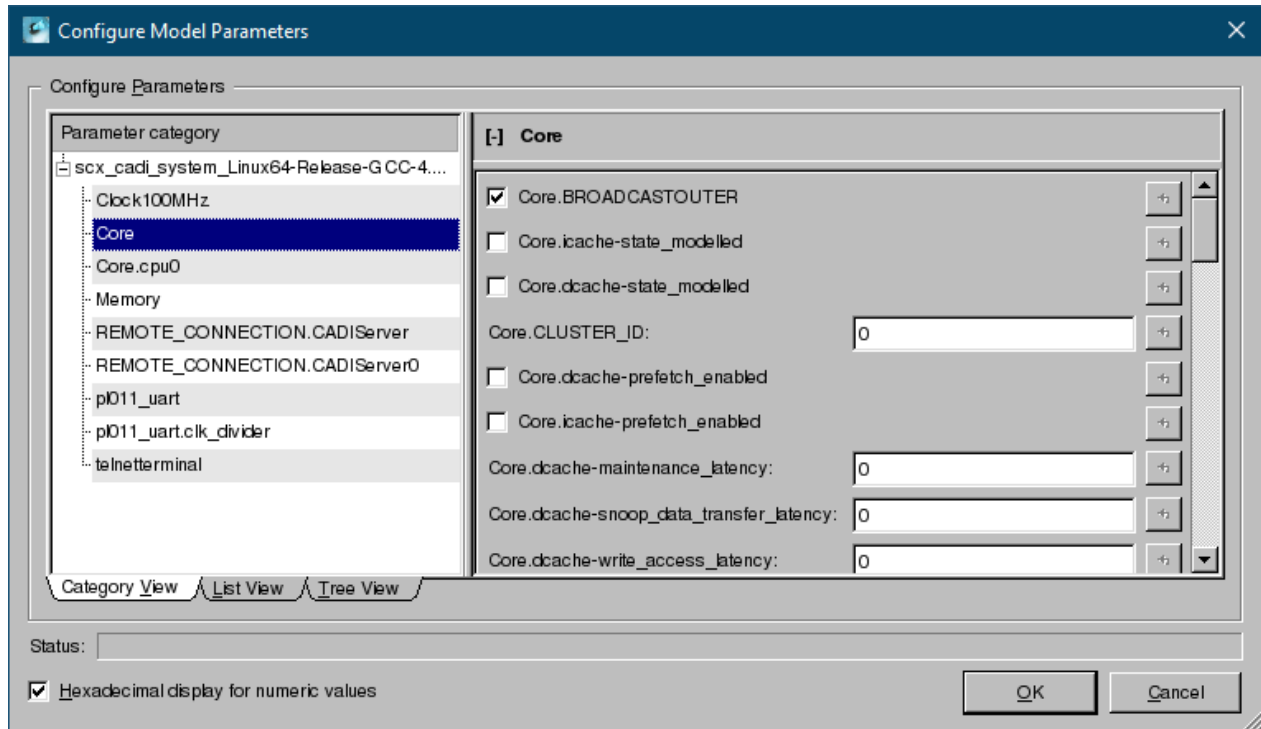
4.4.3 Configuring model parameters

This section describes how to configure models.

If you had not yet loaded a model when Model Debugger starts:

1. Select **File > Load Model**.
2. In the **Load Model** dialog box that appears, locate and select the required model, then click **Open**.
3. A dialog box appears:

Figure 4-2: Configure Model Parameters dialog box



If you had already loaded a model before Model Debugger starts, Model Debugger checks the available components and opens a similar dialog box.

The exact contents and titles of the panes vary depending on the model.

The **Configure Model Parameters** dialog box has the following sections:

Parameter category

This pane contains a hierarchical list of the component parameters by category. To expand or collapse the view, click the **+** or **-** symbol.

Parameter setting

The parameter values are displayed in the right-hand pane.

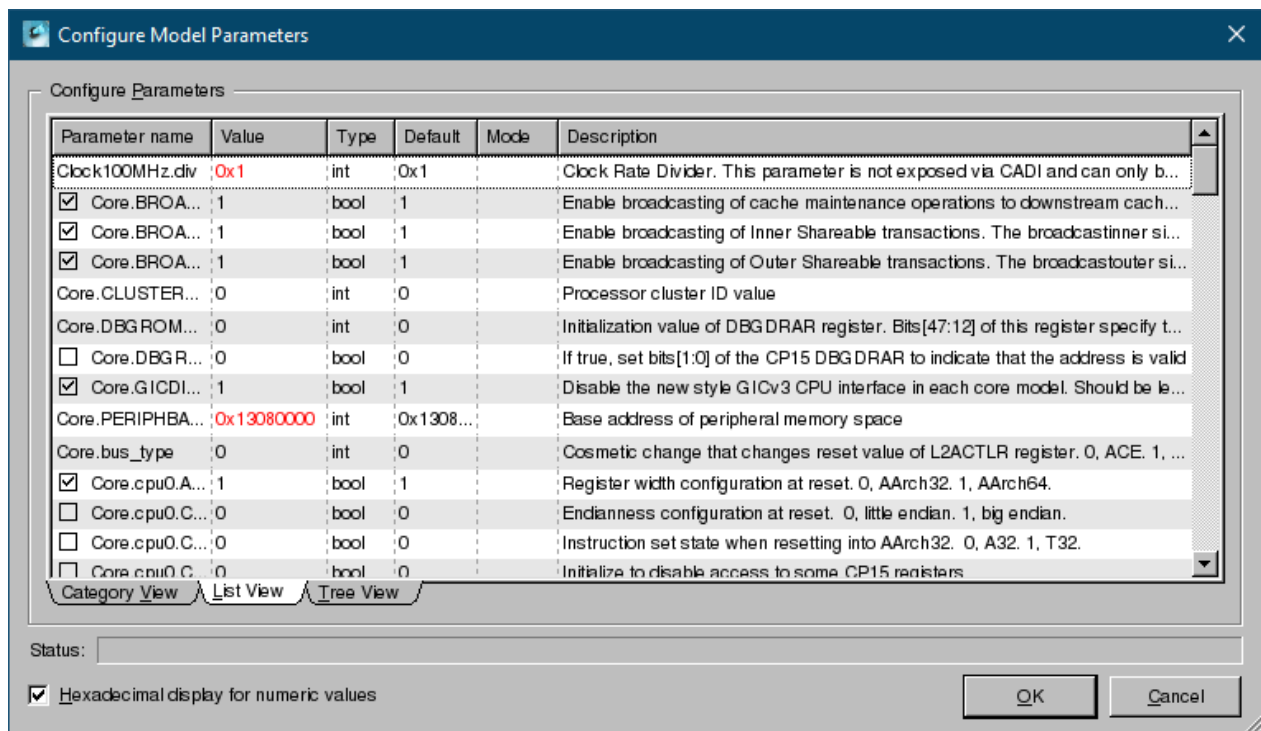
To toggle between hexadecimal or decimal views, use the box in the lower left corner of the window.



The name of this pane varies, depending on the model, but its purpose and usage is the same in all cases.

To view the configuration parameters as a single list, click the **List View** tab. The **Tree View** is similar, but shows the same parameters in a grouped hierarchy.

Figure 4-3: Configure Model Parameters dialog box, List View tab

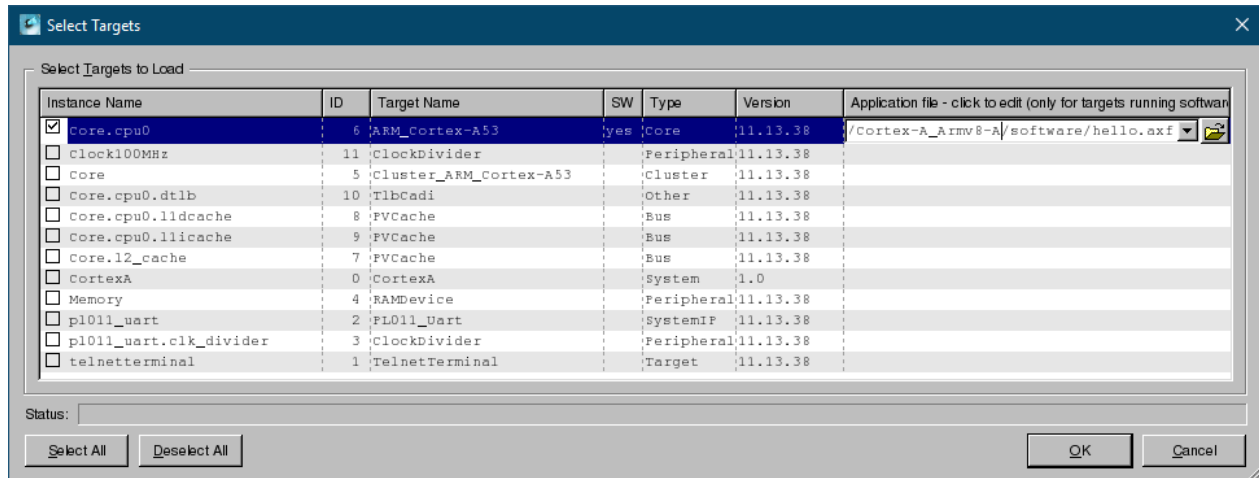


Set the parameters for the model and click **OK** to close the dialog box.

4.4.4 Selecting target components

After closing the **Configure Model Parameters** dialog box, the **Select Targets** dialog box appears. Select the components to be debugged in the model.

Figure 4-4: Select Targets dialog box



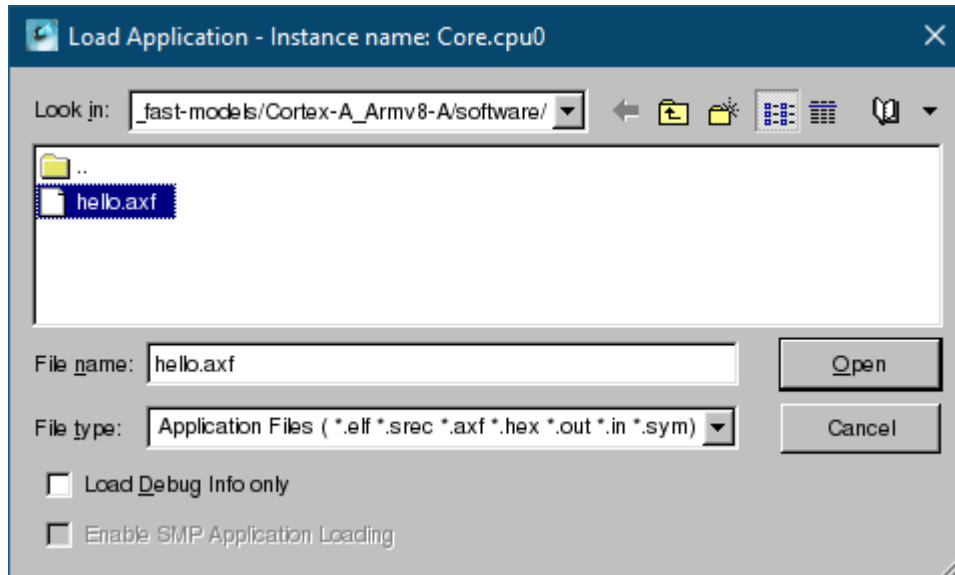
1. Click the checkbox next to the components to load.
2. The **Application file** column displays applications to load for the processors. If the correct application is not selected, click in the field and enter the name of the file.
3. If the application name in the list is the same as the application that was already loaded, the debug information is automatically loaded to the debugger.
4. Click **OK** to close the dialog box.
5. One or more instances of Model Debugger are created, depending on how many targets you selected to load.

If the application is loaded and the source code can be found from the debug information in the specified file, Model Debugger displays the code in the source window.

4.4.5 Loading an application

If the application is not loaded automatically, select **File > Load Application** to locate and load the application manually.

Figure 4-5: Load Application dialog box



Reset the component after loading the source.



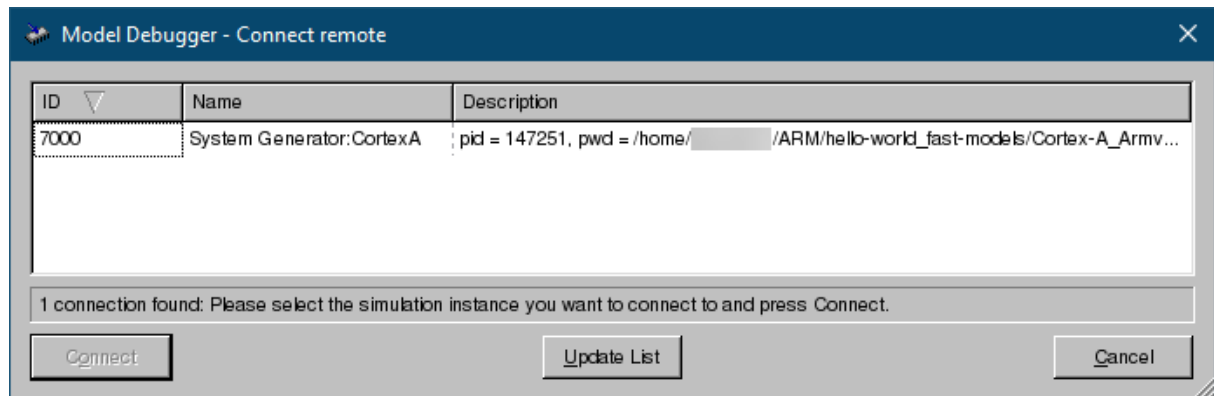
The Load Application dialog box only displays if you have loaded a model from Model Debugger.

4.4.6 Connecting Model Debugger to a running ISIM

You can use Model Debugger to connect to and debug a remote *Integrated SIMulator* (ISIM) that has a CADI interface.

Procedure

1. Launch your ISIM executable with the `-s` option to start a CADI server.
2. Launch Model Debugger.
3. To display the Model Debugger **Connect remote** dialog box, select **File > Connect to Model** :

Figure 4-6: Connect remote dialog box

4. Select the required simulation instance and click **Connect**.
5. Select a target, for example the processor, and click **Connect** to connect to the specific component instance in the simulation.
6. If no application loads, select **File > Load Application Code**. Select the application image from the Load Application dialog box and click **Open**.



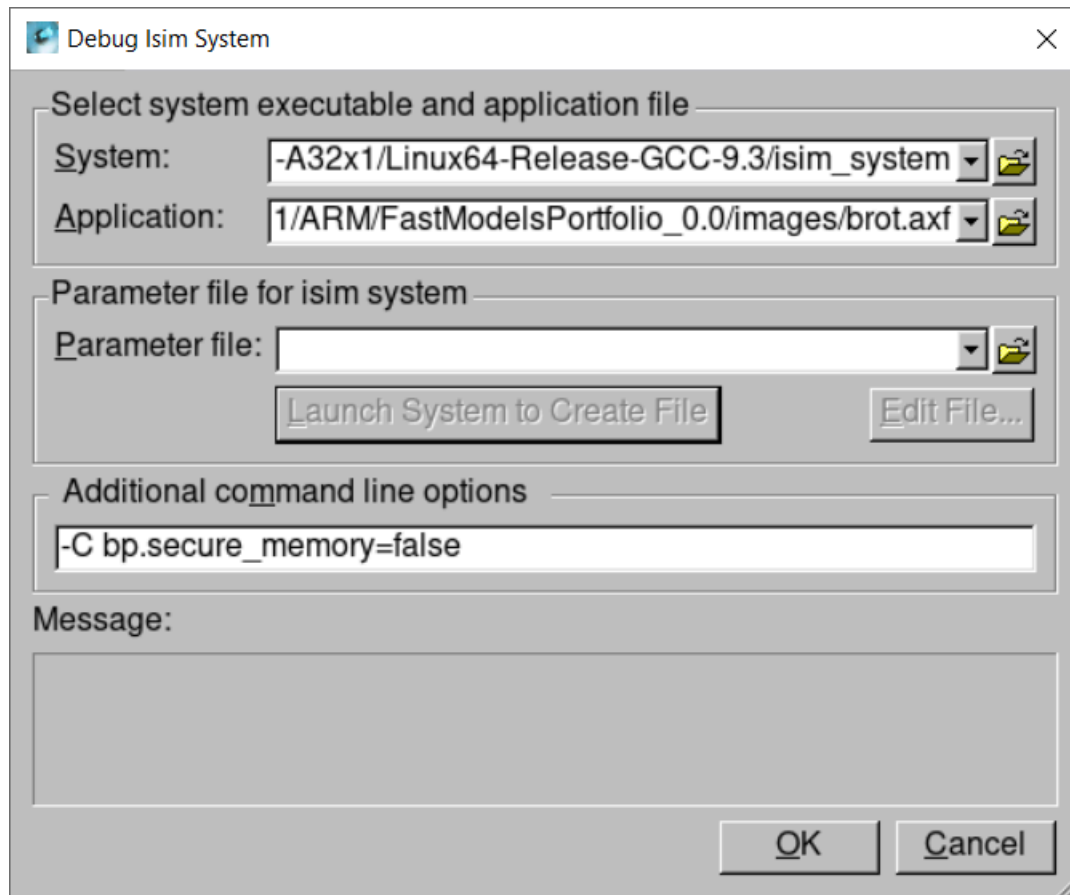
If the application loads and the debug information in the application file allows it, Model Debugger displays the source code in the source window.

4.4.7 Starting a simulation within Model Debugger

In Model Debugger, you can start a SystemC or *Integrated SIMulator* (ISIM) model simulation and then connect to it.

Procedure

1. Select either:
 - **File > Debug Isim System ...** to display the **Debug Isim System** dialog box.
 - **File > Debug SystemC Simulation ...** to display the **Debug SystemC Simulation** dialog box.

Figure 4-7: Debug Isim System dialog box

2. Select a simulation and optionally an application (and parameter file for an ISIM only). If a file is missing, an error message appears. Click **OK** to start the simulation and connect. The **Select Targets** dialog box appears.

Related information

[Connecting Model Debugger to a running ISIM](#) on page 96

4.4.8 Connect Model Debugger to a model running on another host

You can connect Model Debugger to a Fast Model that is running on a different host machine by using the environment variable `CADI_TARGET_MACHINE` and some extra arguments when launching the model.

Procedure

1. Start the model with the arguments:

-s

Start the CADI server.

-C REMOTE_CONNECTION.CADIServer.enable_remote_cadi=1

Enable a remote client, for instance Model Debugger, to connect to the CADI server.

-C REMOTE_CONNECTION.CADIServer.listen_address=<IP_address>

Specify the network address for the CADI server to listen on.

-C REMOTE_CONNECTION.CADIServer.port=<port>

Specify the port for the CADI server to listen on.

-p

Optional parameter to print the port that the CADI server is listening on.

2. On the machine that Model Debugger will run on, set the environment variable `CADI_TARGET_MACHINE` to the IP address and port number from step 1. Use the following syntax:
`CADI_TARGET_MACHINE=<IP_address>:<port_number>`
3. Launch Model Debugger.

Example 4-1: Connect Model Debugger to an ISIM running on a separate host

This example connects Model Debugger running on host 2 to an ISIM running on host 1:

1. Run the following command on host 1:

```
./isim system -S -p -C REMOTE_CONNECTION.CADIServer.enable_remote_cadi=1 \  
-C REMOTE_CONNECTION.CADIServer.listen_address=12.34.56.78 \  
-C REMOTE_CONNECTION.CADIServer.port=7000
```

2. Run the following command on host 2:

```
env CADI_TARGET_MACHINE=12.34.56.78:7000 modeldebugger
```

4.4.9 Using the cache view registers

The Register view displays registers related to the cache when you select a `pvcache` target to load.

Procedure

1. To update the data values displayed in these registers, double click on the value of the `CACHE_SEL` register and press **Enter**:

Figure 4-8: Update the cache view register values

INFO	
Register	Value
⊕ CACHE_INFO	0x00000000 00001FFF
⊖ CACHE_SEL	0x00000000 00000000
⊖ CACHE_SEL_INDEX	0x00000000 00000000
⊖ CACHE_SEL_METHOD	ByIndex
⊕ CACHE_RET_INDEX	0x00000000
⊕ CACHE_RET_TAG	0x00000000
⊖ CACHE_RET_PADDR	0x00000000 00000000
⊖ CACHE_RET_DATA_0	0x00000000 00000000
⊖ CACHE_RET_DATA_1	0x00000000 00000000
⊖ CACHE_RET_DATA_2	0x00000000 00000000
⊖ CACHE_RET_DATA_3	0x00000000 00000000
⊖ CACHE_RET_DATA_4	0x00000000 00000000
⊖ CACHE_RET_DATA_5	0x00000000 00000000
⊖ CACHE_RET_DATA_6	0x00000000 00000000
⊖ CACHE_RET_DATA_7	0x00000000 00000000

2. You can view the cache line values using either the `ByIndex` or `BySetWay` cache selection method. To change between them, double click on the `CACHE_SEL_METHOD` value and edit the value to either `BySetWay` or `ByIndex`, then press **Enter**:

Figure 4-9: Change the cache selection method

INFO	
Register	Value
⊕ CACHE_INFO	0x00000000 00001FFF
⊖ CACHE_SEL	0x00000000 00000000
⊖ CACHE_SEL_INDEX	0x00000000 00000000
⊖ CACHE_SEL_METHOD	BySetWay
⊕ CACHE_RET_INDEX	0x00000000 00000000
⊕ CACHE_RET_TAG	0x85000000 00000040
⊖ CACHE_RET_PADDR	0x00000000 03200000

Related information

[Register views](#) on page 129

4.4.10 Setting and removing breakpoints

This section describes how to work with breakpoints.

Related information

[Breakpoint dialog boxes](#) on page 145

[Breakpoint Properties dialog box](#) on page 145

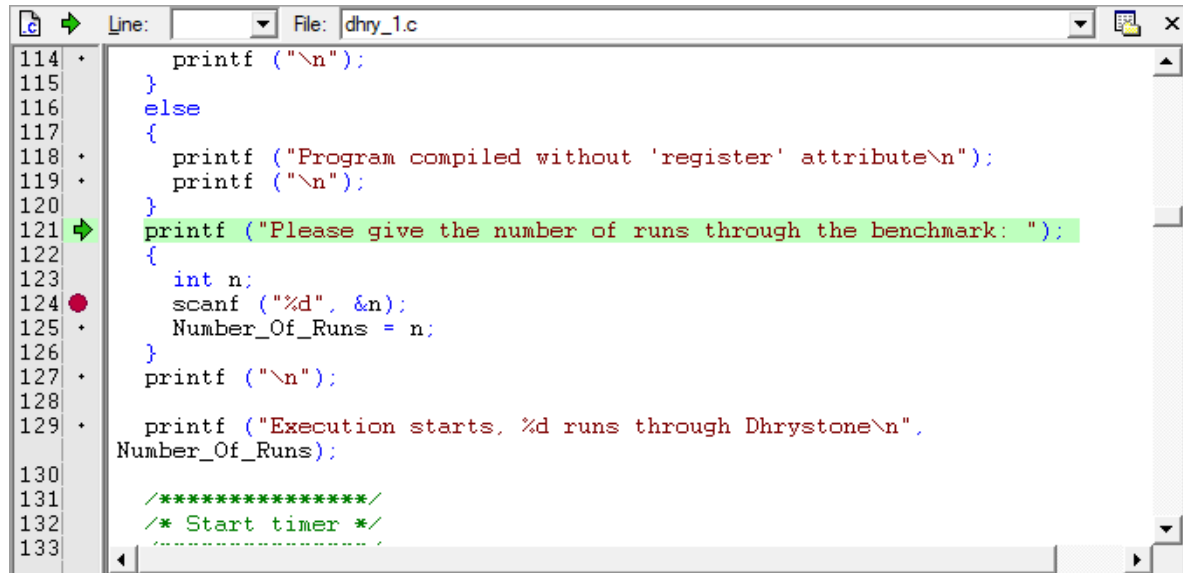
4.4.10.1 Setting breakpoints in the debug views

This section describes how to set different kinds of breakpoint in the debug views.

Source code view

The second column contains small bullets for each source line. To set a breakpoint, double click on a bullet.

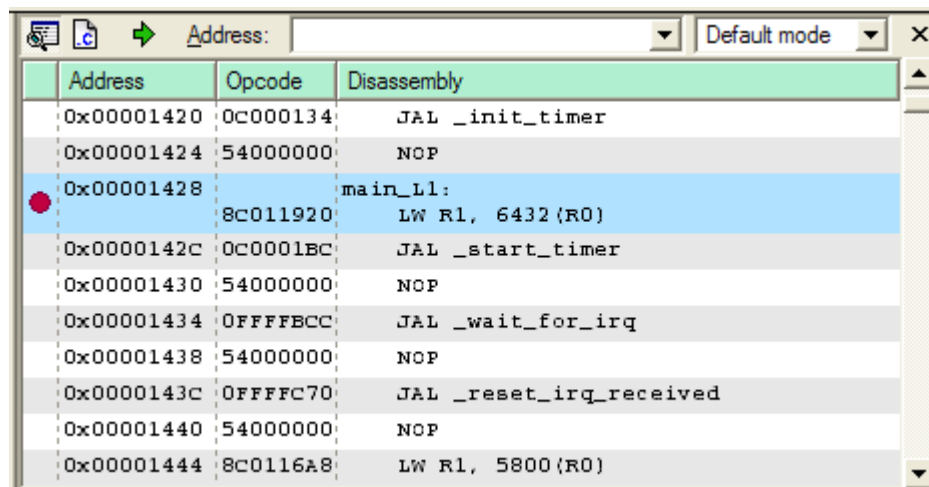
Figure 4-10: Source view breakpoint



Disassembly view

To set a breakpoint on a line, double click on any column.

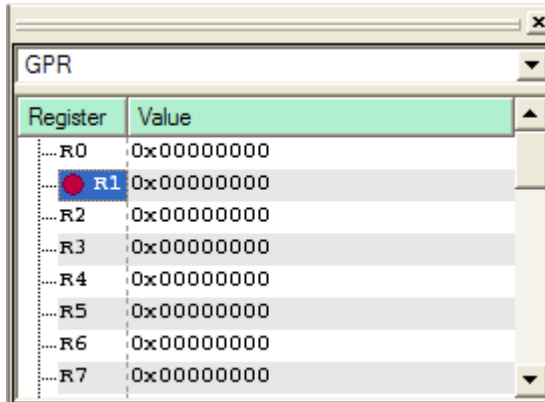
Figure 4-11: Disassembly view breakpoint



Register view

To set breakpoints, double click on the first column, the register name column.

Figure 4-12: Register view breakpoint



Memory view

To set breakpoints, select **Insert Breakpoint** from the context menu. It is not possible to set a memory breakpoint by double clicking on an address.

Local variables view

It is not possible to set these breakpoints.

Global variables view

It is not possible to set these breakpoints.

Call stack view

To set breakpoints, double click on items in the first column.



To use this view, the architecture must have a definition of the DWARF register mapping and the model must have DWARF register mapping too. The loaded application must be an ELF file that contains a `.debug_frame` section.

Pipeline Table

To set breakpoints, double click on the name in the first column.

Figure 4-13: Pipeline table breakpoint

IF		ID		EX		MEM		WB	
PC	0x00001634	PC	0x00001630	PC	0x0000162c	PC	0x00001628	PC	0x00001624
CON	3	CON	3	CON	3	CON	3	CON	3
opc	CAFBC0000	opc	23DE0004	opc	03C0E820	opc	AFDDFFFC	opc	00000000
dis	SW 0(R29) , R2	dis	ADDI R30, R30,	dis	ADD R29, R30,)	dis	SW -4(R30) , R	dis	SYNCP
it	0x00000000	it	0x00000002	it	0x00000001	it	0x00000004	it	0x00000000
a	0x00000000	a	0x00001848	a	0x00001844	a	0x00001840	a	0x00000000
b	0x00000000	b	0x00000000	b	0x00000000	b	0x00000000	b	0x00000000
d	0x00000000	d	0x00000000	d	0x00000000	d	0x0000172c	d	0x00000000
i	0x00000000	i	0x00000004	i	0x00000000	i	0xFFFFFFFF	i	0x00000000

Watch view

If you copy an item from another view into the Watch view, you can set breakpoints in either the original view or the Watch view.

4.4.10.2 Setting conditional breakpoints

Some breakpoint objects support conditional breakpoints.

About this task

To create a conditional breakpoint:

Procedure

1. Set an unconditional breakpoint.
2. Set the conditions for the breakpoint with the Breakpoint Properties dialog box.

Related information

[Breakpoint Properties dialog box](#) on page 145

4.4.10.3 Removing and disabling breakpoints

This section describes how to inactivate breakpoints.

You can quickly remove a breakpoint by double clicking on it. To inactivate a breakpoint without removing it, disable the breakpoint by right-clicking on the breakpoint and selecting **Disable breakpoint** in the resulting context menu. A disabled breakpoint is shown as a gray, rather than red, circle symbol. Other breakpoint dialog boxes and menus also permit you to configure your breakpoints.

4.4.11 Model Debugger sessions

Model Debugger session files enable saving and restoring debugging sessions and provide a convenient way to specify the session parameters.

Session files have the extension *.mvs.



The session files are only available for directly loaded models. They cannot be used for connections to Model Shell or SystemC simulations.

The information that can be saved and restored includes:

- Debugger main window geometry.
- Layout of all debug views.
- Target model being loaded.
- Application file.
- Breakpoints.

Session files also enable configuring the individual layout of debugger windows for cluster systems. You could, for example, then use the project with SoC Designer.

To save a session, select **Save Session**, or **Save Session As**, from the **File** menu.

To load a session and restore the original model connection and window layout, select **Load Session** from the **File** menu.

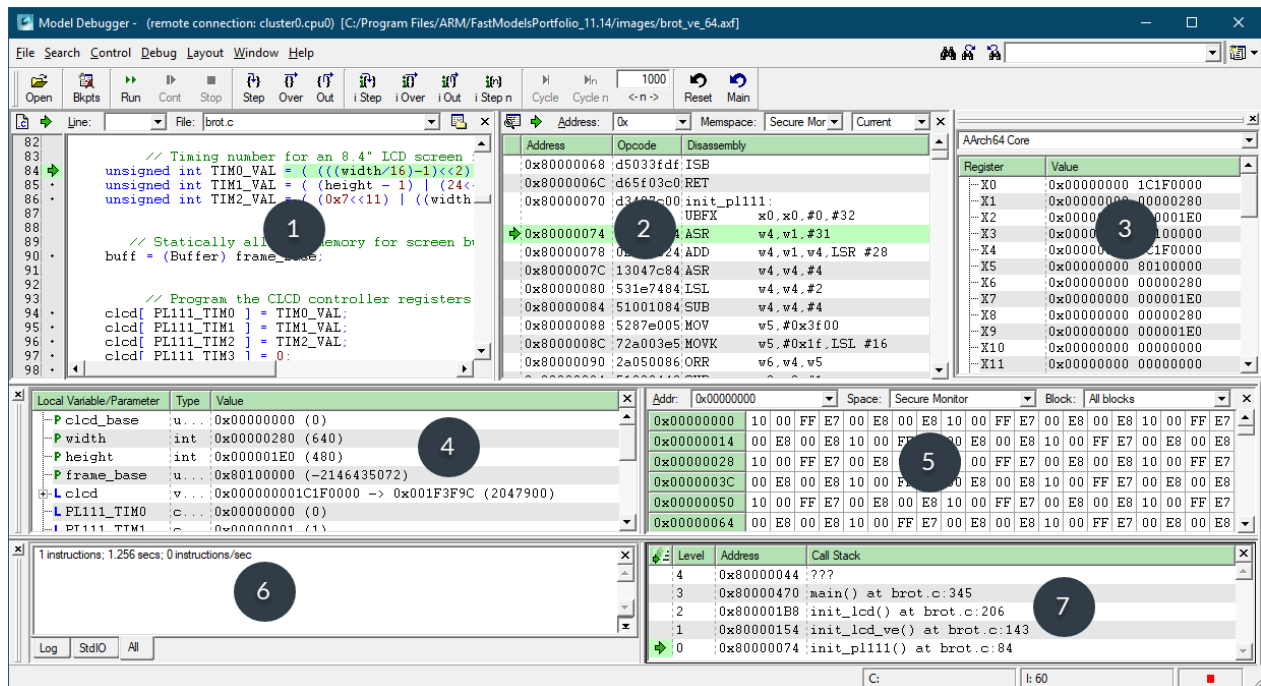
4.5 Model Debugger GUI

This chapter describes the windows, menus, dialogs, and controls in Model Debugger.

4.5.1 Application windows

The Model Debugger GUI consists of the main menu, the toolbar, and the workspace with dock windows.

This screen capture shows the default layout, but you can configure it using the **Layout** menu:

Figure 4-14: A Model Debugger session: default window layout

1. **Source window** shows the source code with line numbers and bullets that represent lines of executable code, on which you can set breakpoints.
2. **Disassembly window** shows indicators for breakpoints and the PC, and address, opcode, and disassembly strings.
3. **Register window** shows registers and their values.
4. **Local variables window** shows all local variables (indicated by the letter L) and parameters (indicated by the letter P) that are valid at the current PC, with their type and value.
5. **Memory window** shows the contents of a range of memory addresses starting from the base address specified in the address field (**Addr:**). Other fields allow for selection of the address space (**Space:**) and physical memory block (**Block:**).
6. **Output window** displays debugger messages and debug target output.
7. **Call stack window** displays the call stack.

Related information

[Preferences dialog box](#) on page 147

[Saving the window layout](#) on page 117

4.5.1.1 Workspace

The workspace can contain various view types.

- Source code.

- Disassembly.
- Call stack.
- Thread.
- Register.
- Memory.
- Global variables.
- Local variables.
- Output.
- Watch.

By default, the layout does not contain the thread, global variable, or watch windows.

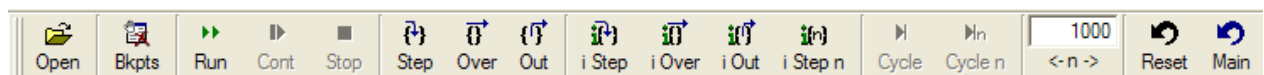
The workspace layout can be customized by opening views, closing views, or specifying options in the Preferences dialog box.

All views can be moved or resized. Project files enable saving and restoring the customized layout. The files can give each processor target type, and even each target instance, a unique appearance.

4.5.1.2 Main toolbar

The main toolbar provides buttons for frequently used functions. If the functionality is not available in the current context, the buttons are grayed out.

Figure 4-15: Main toolbar



Open

Click to open a model library and application file. When the button is clicked:

1. If a model library is not already open, a dialog box is displayed to enable you to select a model library to load.

Select the model library and click **OK**.

2. If an application is not already open, a dialog box opens to enable you to select the application file to load into the target.

Select the application file and click **OK**.

3. If a model library and application are already open, a dialog box is displayed to select the source file for the application.

Select the source file and click **OK**.



Note

You might use a Symmetric MultiProcessing (SMP) model with more than one processor, such as one based on the Cortex®-A9 processor. In this case, Model Debugger only loads one image that is run on all processors. All Model Debuggers that are attached to the SMP model load the debug information for that image. This feature is called SMP awareness.

In certain circumstances, you can switch SMP awareness on or off by using the Model Debugger Preferences dialog box.

Bkpts

Click to open the breakpoint manager.

Run

To run the simulation until a breakpoint is hit or some exception occurs, click this button. Encountering a simulation halt is an example of an exception that stops simulation.

Pause/Cont

Click to pause or continue the current high-level simulation step command. An example would be a source-level step. The button text and icon changes depending on whether the simulation is running (**Pause**) or stopped (**Cont**).

You can interrupt high-level simulation control commands with breakpoints before completion. These commands can be completed by clicking the **Cont** button.

Stop

Click to stop the execution of the model being debugged.

Step

To execute until the simulation reaches a different source line, click to cause a source-level step.

Over

To execute the simulation and step over any function calls, click to cause source-level steps.

Out

To execute control command until the current function is exited, click to cause source-level steps.

i Step

Click to advance the simulation by executing one source-level instruction.

i Over

Click to advance the simulation by one source-level instruction without following any call instructions.



Note

Not all model targets support this command.

i Out

Click to advance the simulation until a return instruction is executed.



Not all model targets support this command.

i Step n

Click to advance the simulation by executing the number of source-level instructions that are specified in the **<-n->** control.

Cycle

Click to advance the simulation by a single cycle.

Cycle n

Click to advance the simulation by the number of cycles that are specified in the edit box. The default is 1000 cycles.

<-n ->

Enter the number of cycles to step if the **Cycle n** or **Back n** buttons are clicked. The default is 1000 cycles.

If the **i Step n** button is clicked, this control indicates the number of instructions to step.

Back n

Click to step the simulation backwards by the number of cycles that are specified in the edit box. The default is 1000 cycles.



Not all model targets support this command.

Back

Click to step the simulation backwards by one cycle.



Not all model targets support this command.

Reset

Click to cause a reset of the target model. The application is reloaded.



For best results, Arm recommends the syncLevel of the model should be 1 or higher when using this command.

Main

Click to cause a reset of the target model. The application is reloaded. The model runs until the `main()` function of the application source code is reached.



- This command is only available if a `main()` function can be found in the debug information of the application file.
 - For best results, Arm recommends the syncLevel of the model should be 1 or higher when using this command.
-

Related information

[Preferences dialog box](#) on page 147

4.5.1.3 Menu bar

The main menu bar provides access to most Model Debugger functions and commands.

File menu

The **File** menu has the following options:

Open Source ...

Opens the source code for the application.

Source File Manager ...

Displays the Source File Manager dialog box.

Load Application Code ...

Loads application code to the model.

Load Application Code (Debug info only) ...

Loads debug information only.

Load Model ...

Loads a model.

Connect to Model ...

Displays the Connect to Target dialog box to connect to a model file.

Debug Isim System ...

Displays the Debug Isim System dialog box to start and debug an isim system.

Debug SystemC Simulation ...

Displays the Debug SystemC Simulation dialog box to start and debug a SystemC simulation.

Close Model

Closes the currently open model. If Model Debugger is connected to a CADI server, the connection is closed but the simulation continues to run.

Open Session ...

Opens a previously saved session.

Save Session

Saves the current debug session.

Save Session As

Saves the current debug session to a new location and name.

Preferences

Displays the Preferences dialog box to enable you to modify the user preferences.

Recently Opened Models

Displays a list of the most recently opened model files. To open the file, click a list entry. By default, the last 16 files are displayed in the list. The number of files to display can be set in the Preferences dialog box.

To remove a file from the list, move the mouse cursor over the file name. Press the **Delete** key or right click and select **Remove from list** from the context menu.

Recently Opened Applications

Displays a list of the most recently opened applications. To open the application, click a list entry. By default, the last 16 applications are displayed in the list. The number of applications to display can be set in the Preferences dialog box.

To remove an application from the list, move the mouse cursor over the application name. Press the **Delete** key or right click and select **Remove from list** from the context menu.

Recently Opened Sessions

Displays a list of the most recently opened sessions. To open the session, click a list entry. By default, the last 16 sessions are displayed in the list. The number of sessions to display can be set in the Preferences dialog box.

To remove a session from the list, move the mouse cursor over the session name. Press the **Delete** key or right click and select **Remove from list** from the context menu.

Exit

Ends Model Debugger. If you have modified files or sessions, a dialog box prompts you to save your changes.

Search menu

The **Search** menu has the following options:

Find ...

Opens a dialog box that enables searching for a string in a currently active window.

Find Next

Repeats the last defined search to find the next occurrence.

Find Previous

Repeats the last defined search, but the search direction is backwards in the document.

Control menu

The **Control** menu has the following options:



When using options that reset the model, Arm recommends the `syncLevel` should be 1 or higher, for best results.

Hard Reset

This option resets the target model without reloading the application.

Reset

Click to cause a reset of the target model. The application is reloaded automatically.

Goto Main

Cause a reset of the target model. The application is reloaded. The model runs until the `main()` function of the application source code is reached.



This command is only available if a function `main()` can be found in the debug information of the application file.

Run

Run the simulation until a breakpoint is hit or some exception occurs. An example would be simulation halt.

Pause/Continue Source Step

Pause or continue the current high-level simulation step command. An example would be a source-level step.

Source Step Over

Cause a source-level step to execute until the simulation reaches a different source line.

Source Step Out

Cause source-level steps to execute control command until the current function is exited.

Instruction Step

Advance the simulation by executing one source-level instruction.

Instruction Step Over

Advance the simulation by one source-level instruction without following any call instructions.



Not all model targets support this command.

Instruction Step Out

Advance the simulation until a return instruction is executed.



Not all model targets support this command.

Instruction Step n

Advance the simulation by the number of instructions in the **<- n ->** edit box. The default is 1000 cycles.

Cycle Step

Advance the simulation by a single cycle.

Cycle Step n

Advance the simulation by the number of cycles in the edit box. The default is 1000 cycles.

Enable/Disable Step Back

Enable or disable stepping back by cycles.



Not all model targets support this command.

Back

Step the simulation backwards by one cycle.



Not all model targets support this command.

Back n

Step the simulation backwards by the number of cycles in the edit box. The default is 1000 cycles.

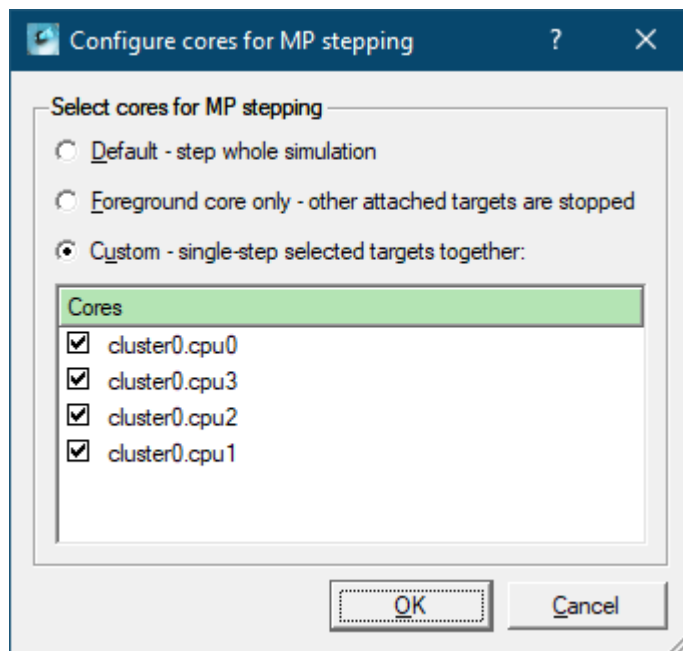


Not all model targets support this command.

Configure cores for MP stepping ...

To enable independent execution of cores, that is, targets, use the **Configure cores for MP stepping** dialog box.

Figure 4-16: Configure cores for MP stepping dialog box



In cluster (multiprocessor) debugging, each Model Debugger window is connected to a particular target, and the controls in that window apply only to that target. It is the simulation that determines how other connected targets behave when you click **Stop**, **Step**, or **Run** within a window. Typical behavior is to stop and run the whole simulation.

You use the **Configure cores for MP stepping** dialog box to enable Model Debugger to override the default behavior. Model Debugger can control each target to which it is connected. It can force that target to stop executing code while the simulation is running or stepping. In that instance, Model Debugger does not stop any target to which it is not connected. To stop during independent stepping, connect to a target, even if you do not specifically want to view or control that target.



The **Configure cores for MP stepping** dialog box is only enabled if you have loaded a model.

The available MP stepping modes are as follows:

- Use **Default - step whole simulation** to place all execution control with the simulator. In this mode, Model Debugger does not explicitly stop any targets.
- **Foreground core only - other attached targets are stopped** enables the foreground target to run, and to stop all other targets to which it is connected.



The foreground target is the target that is associated with the window that you have selected to run.

-
- **Custom - single-step selected targets together** enables a fixed set of targets to run, and to stop all other targets to which Model Debugger is connected. This mode disables step and run controls for deselected targets.

Debug menu

The **Debug** menu has the following options:

Display Messages

Display debug messages.

Clear Log

Clear the log of debug messages.

Clear Model Output

Clears all output messages from the model.

Clear Output Summary

Clear the summary output messages.

Breakpoint Manager ...

Display the Breakpoint Manager dialog box.

Profiling Manager ...

Display the Profiling Manager dialog box.



Fast Models does not use the profiling options.

View Profiling ...

Display the Profile Information dialog box.



Fast Models does not use the profiling options.

Save Model State ...

Save the current model state. If reloaded, simulation continues from the point where the model state was saved.

Restore Model State ...

Reload a previously saved model state.

Load Debug Info for Module

Load debug information for the module.

Set Parameters

Set parameter values for the model.

Select Targets

Select the execution target within the model.

Layout menu

The **Layout** menu has the following options:

Layout Control Window

To set layout options such as tiling, display this window.

Load Layout ...

Load a previously saved window layout.

Save Layout ...

Save the current layout. Model state is not saved.

Load Recent Layout

Use a recently used window layout.

Restore Default Layout

Restore the window layout to the defaults. This option is useful if the layout has become disorganized.

Window menu

The **Window** menu has the following options:

New View

Display a new debug view.

Hide

Hide an existing debug view.

Show

Display view that was most recently hidden.

Show All

Displays all previously hidden views.

Close

Close the window in focus.

Arrange Horizontally

Tile all view windows horizontally.

Arrange Vertically

Tile all view windows vertically.

Move

Move a view to the new position specified on the submenu.

Docked Views

Dock or undock the view list on the submenu.

Help menu

The **Help** menu has the following options:

Help ...

Opens this book in Adobe Acrobat Reader.

About ...

Displays the standard About dialog box displaying version and license information.

About Model ...

Opens the text file that contains the release notes.

4.5.1.4 Dock windows

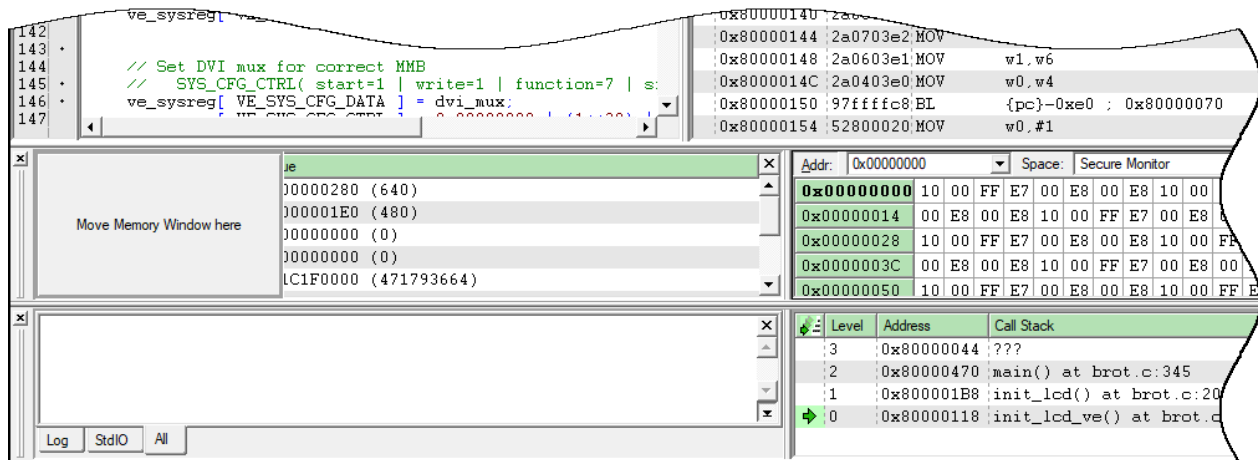
Model Debugger provides dock windows that can be docked inside the main workspace or floated as a top-level window. To toggle between the docked and floating state, double-click on the dock window handle or the title bar of the floating window.

4.5.1.5 Moving or copying views

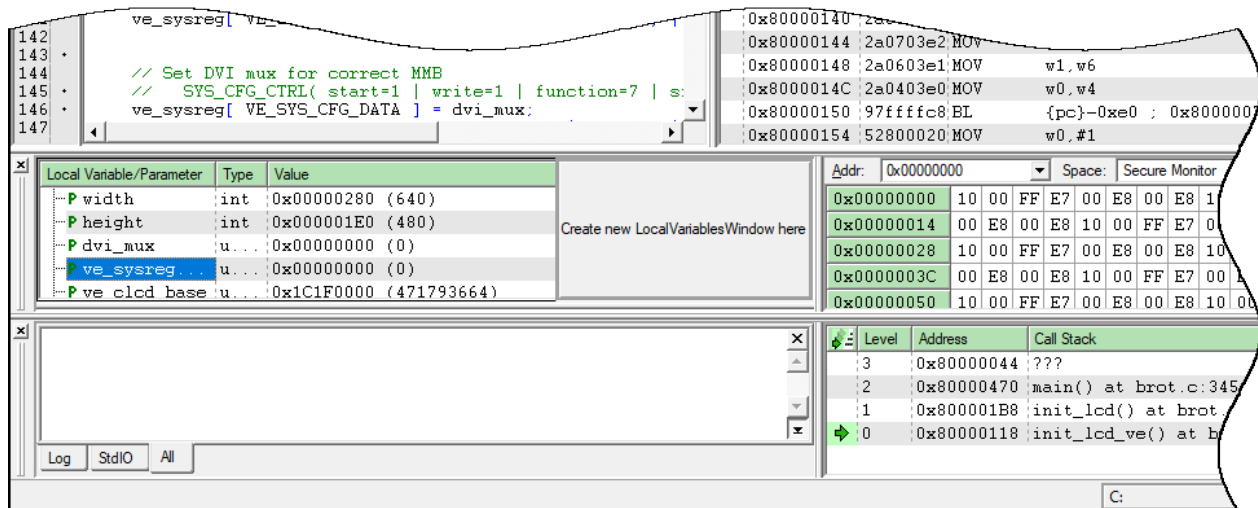
Move or copy debug views within the same dock window or copied by dragging and dropping into another dock window.

To start a drag-and-drop operation, left-click the debug view and, while holding the mouse button down, press the **F9** key.

A gray box on the left edge near the bottom of the Model Debugger window indicates the target location. Releasing the mouse button drops the window into the gray box.

Figure 4-17: Moving a window

To copy a window, press **Ctrl+F9**. This action effectively duplicates the existing view. A gray box near the center of the Model Debugger window indicates the location for the duplicate view for the Local Variables window. Releasing the mouse button creates the duplicate window in the target location.

Figure 4-18: Duplicating a window**Note**

The windows might be difficult to place into the required position. To force the window to dock to a particular location, select the window handle and right-click to display the context menu. The options are: **Dock Bottom**, **Dock Left**, **Dock Right**, and **Dock Top**. It might take several moves to force the window to the required location.

4.5.1.6 Saving the window layout

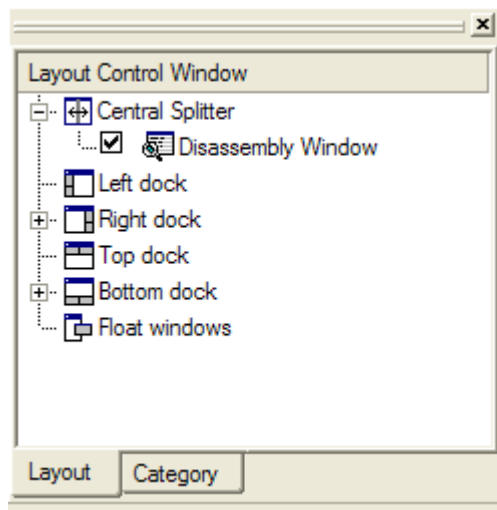
If you use different debug windows and views for different models, you can save and later reload layouts to simplify reorganizing the views.

The **Layout** menu has the following entries:

Layout Control Window

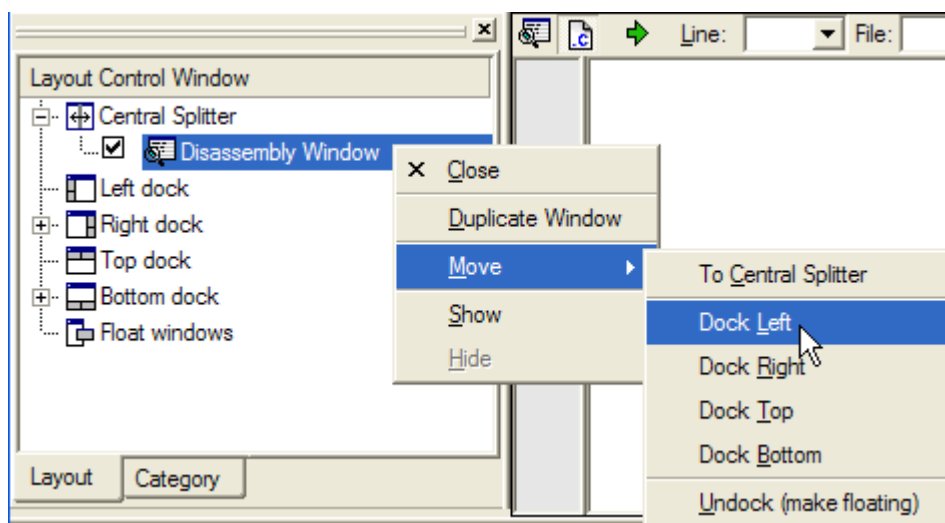
Displays the window. To change focus to the selected window, click an entry.

Figure 4-19: Layout Control window



Right-click to display a context menu for moving or duplicating windows.

Figure 4-20: Layout Control context menu





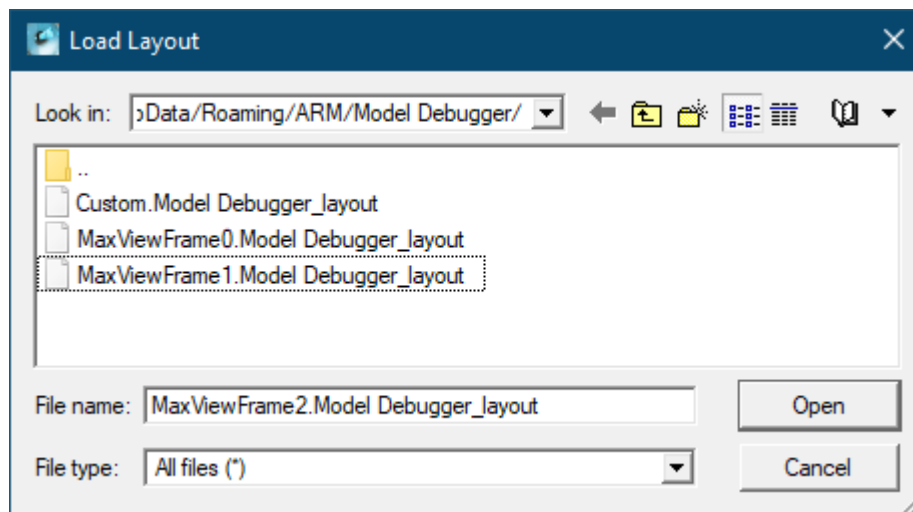
Note

You can also use drag-and-drop within the Layout Control window to change the location of the windows.

Load Layout

Load a previously saved layout file. The window positions match the window configuration present when the layout was saved.

Figure 4-21: Load Layout dialog box



Save Layout

Save the current window arrangement to a layout file.

Load Recent Layout

Load the last saved layout. If you have modified the current layout, a prompt asks whether to save the current layout.

Restore Default Layout

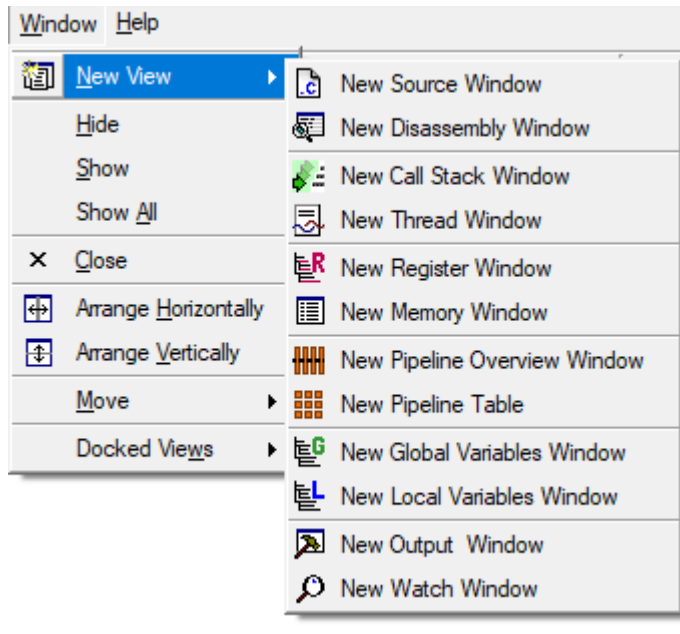
Use the default layout.

4.5.1.7 Debug views

To open a new debug view window, select **New View** from the **Window** menu and select it from the list.

About this task

Selecting a new debug view:



To close a dock window and all views in it, click the close button in the dock handle or title bar. This action closes all views in the window. To close views individually, click the specific close icon.

4.5.1.8 Output window

This window displays messages from Model Debugger and from the debugging targets.

The window has the following tabs:

Log

Debugger messages, such as errors and warnings.

StdIO

Output from the target model.

All

An interleaved view for both the **Log** and **StdIO** messages.

4.5.2 Debug views for source code and disassembly

The Source code and Disassembly views share a common window.

Each view consists of:

- A title bar with controls for selecting a target line or switching between views.
- The actual code browser for source or disassembly.
- Columns for line number or address.

The function of the columns and title bar controls is specific to each view.

4.5.2.1 Source view

This section describes the **Source view**.

The **Source view** on the left contains two columns with a gray background that contain the line number and bullets that represent executable code locations. The right side of the view contains your source code.

The button with the green arrow scrolls the code browser to the location of the statement or instruction that is to be executed next. You can find this button at the top left of the **Source view** window.

Figure 4-23: Arrow button for scrolling code



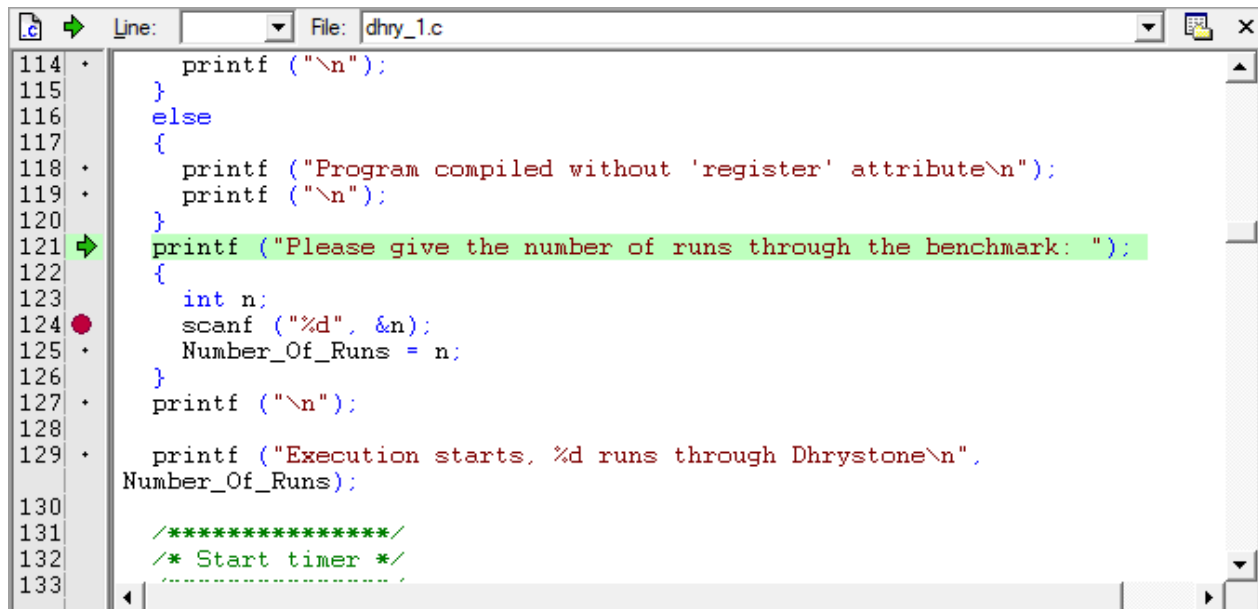
To highlight the corresponding addresses in the disassembly view, click the left-most column in the **Source view**. The highlighting reveals the instructions the source statement maps to.



Note

Highlighting is only available for source lines with a bullet. The bullet indicates that the line is executable.

To set a breakpoint on the source line, double click a bullet. A filled red circle is displayed next to the line to indicate that a breakpoint has been set.

Figure 4-24: Source view

The **Source view** title bar has controls for:

- Selecting a target line in the source using the **Line:** entry box.
- Selecting a source file that has already been loaded using the **File:** drop down list.
- Opening the Debug Source Files dialog box.

Context menu for Source view

Right click in the **Source view** to display the context menu. The menu has the following options:

Insert Breakpoint

Insert a breakpoint at the selected location.

Enable Breakpoint

Enable the breakpoint at the selected location.

Breakpoint Properties

If a breakpoint is present on the selected instruction, selecting this option displays the **Breakpoint properties** dialog box.

Run to here

Run to the selected instruction.


Word wrap

Wrap the text to fit inside the window.

File properties

Display the filename and path for the file.

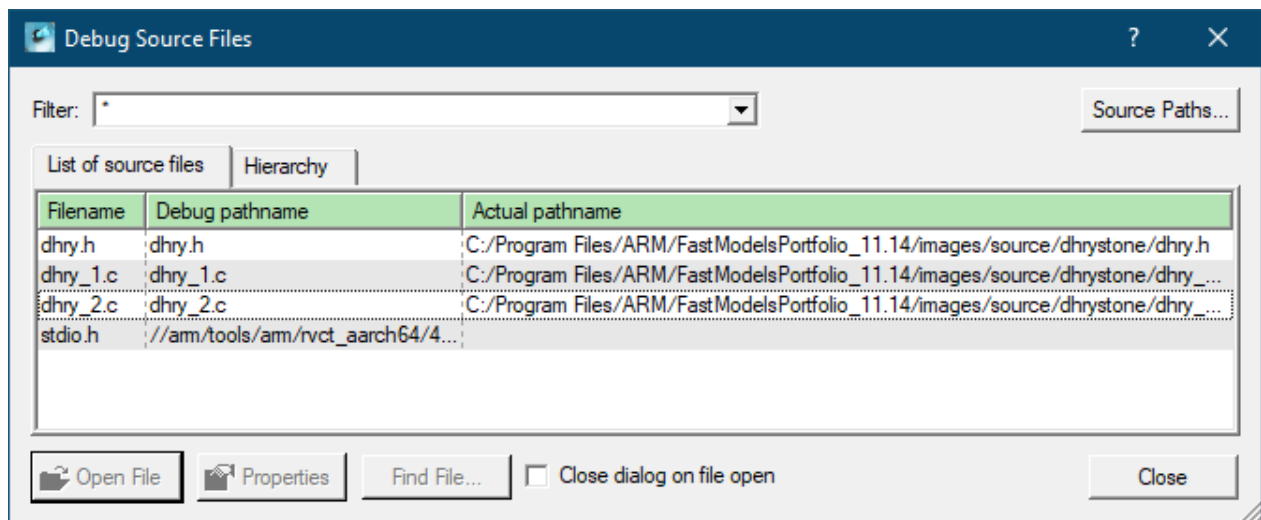
Debug Source Files dialog box

 The **Debug Source Files** dialog box lets you locate source files that are required for debugging an application. To open the dialog box, click the icon in the upper right corner of the **Source view**.



Pathnames appear with slash (/) characters, even on MS Windows. This fact does not affect operation.

Figure 4-25: Debug Source Files dialog box



The tabs switch between two different views that list the properties for the source file:

Filename

This column contains a list of files that the debugged application refers to. This column is not shown in **Hierarchy** view.

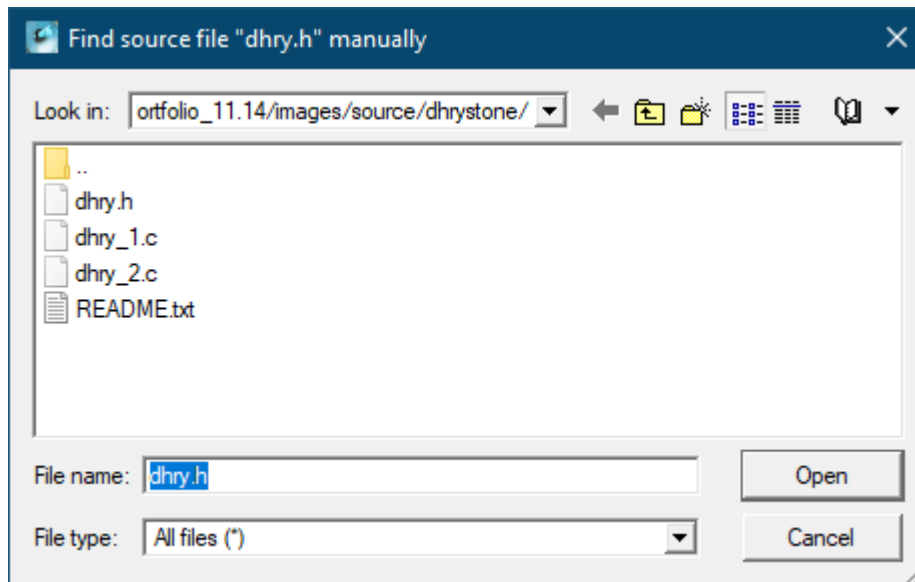
Debug pathname

This column shows the path for the file. The pathname comes from the debug information of the application. This path might be invalid because it refers to the original source file at compilation time. The debug pathname can be absolute or relative to the executable.

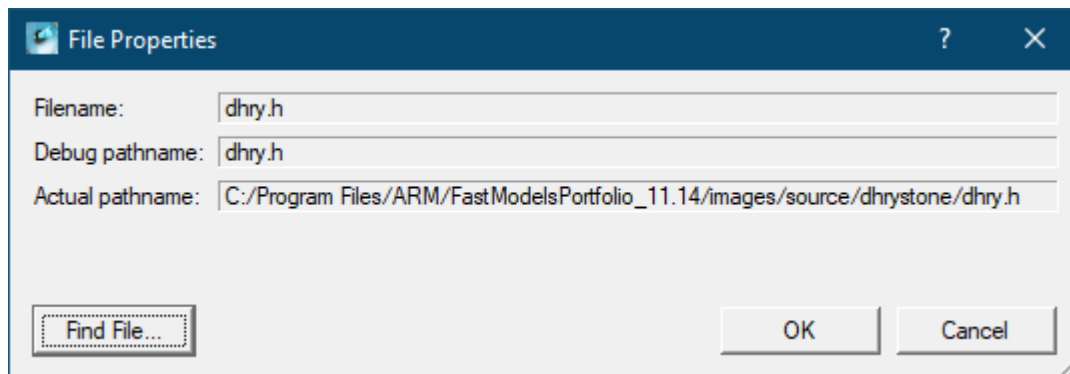
Actual pathname

This column contains the path Model Debugger actually uses to locate the file. You can set the path by double clicking a row or selecting a row and clicking **Open File**. The **File Open** dialog box enables selecting the source file. After selecting the file, the file is opened in the debugger.

Click **Find File** to display the **Find source file** dialog box and navigate to the directory containing the source.

Figure 4-26: Find Source File dialog box

Click **Properties** to display the **File Properties** dialog box for the selected file. You can also use the **Find File** button in the **File Properties** dialog box to locate the file.

Figure 4-27: Source File Properties dialog box

Model Debugger has an automatic mechanism to add replacement paths that are invoked every time you are prompted to find a source file. If the source file is found, an automatic source path replacement is calculated.

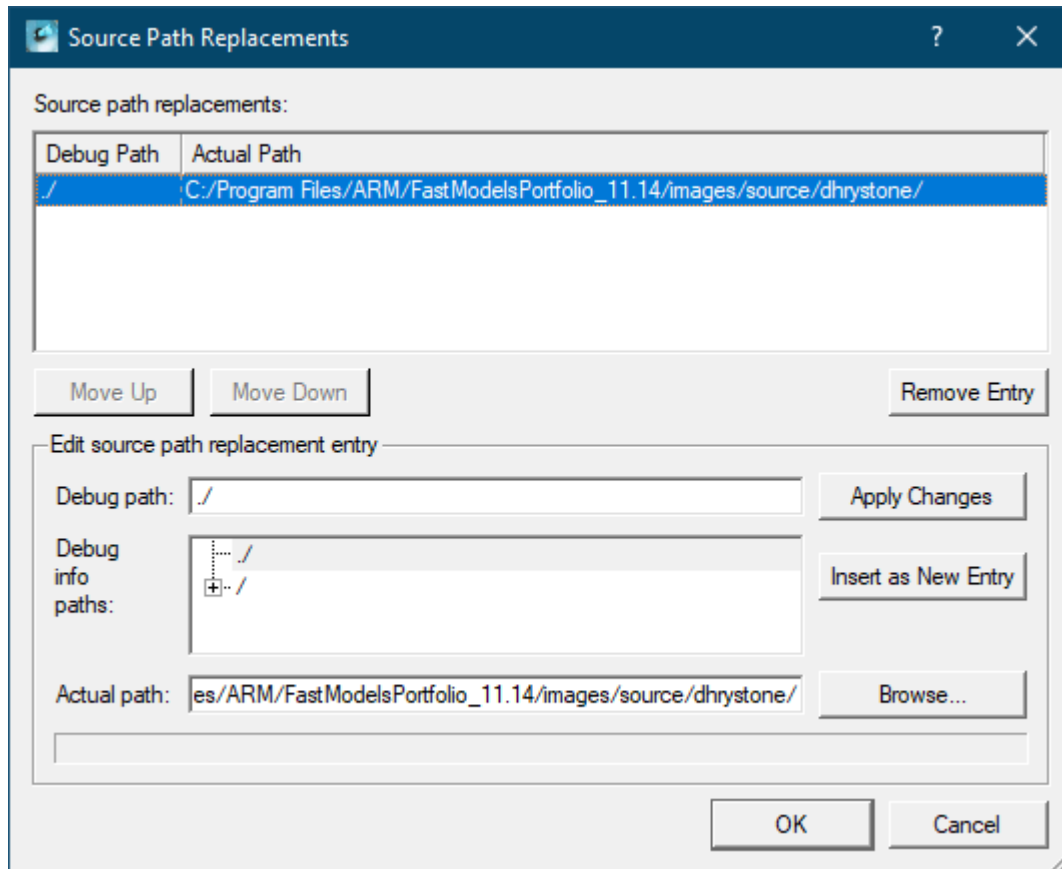
This path might not always be correct. There are situations where you must manually edit source path replacements because the automatic path is wrong for that context. You might, for example, have a header file whose name is common between two different compilers, and Model Debugger chooses the wrong one.

Click **Source Paths...** to open the **Source Path Replacements** dialog box. Use this dialog box to change the path, or priority of the paths, to the source files for the application.



The source path replacements are stored in the Model Debugger session file and not with user preferences.

Figure 4-28: Source Path Replacement dialog box



Existing source file replacements are displayed in the top part of the **Source Path Replacement** dialog box. You can remove or reorder paths by highlighting an entry and clicking one of the following buttons:

Move Up

Move the path up one position in the list.

Move Down

Move the path down one position in the list.

Remove Entry

Delete the path from the list.

Debug Path and **Actual Path** have the same meaning as in the **Debug Source Files** dialog box.

In the lower part of the **Source Path Replacement** dialog box, you can add new source paths or modify existing ones. The additional features are:

Debug info paths

Provides a tree view that simplifies navigation through the debug paths in the debug information of the source file.

Browse

Click this button to select a path with a browser rather than typing in the actual path directly.

Apply Changes

Modify the selected entry using the entered changes.

Insert as New Entry

Adds the new path to the source path replacement list.

Searching in source files

You can search for text in the active window by using the **Find** dialog box. Click **Find** on the **Search** menu to open the **Find** dialog box.

Type the text in the box and click the **Find Next** or **Find Previous** buttons to search upwards or downwards. Re-use previous search terms by clicking the drop-down arrow on the right of the text entry box.

The dialog box is modeless, so you can change views without closing it. The mode is updated automatically.

4.5.2.2 Disassembly view



The Disassembly view provides four columns for breakpoints and PC indicator, address, opcode, and disassembly string.

If your target model has TrustZone® support, disassembly breakpoints from all worlds appear in the first column. The filled red circles indicate a breakpoint in the world in the disassembly view, and unfilled red circles indicate breakpoints in other worlds.

The green arrow indicates the actual position of the PC.

To display the whole disassembly in a help bubble, move the cursor over a disassembly line. This function is useful if the complete disassembly string does not fit horizontally into the view.

Figure 4-29: Disassembly view

  Address: 0x Memspace: Secure Monitor Current			
	Address	Opcode	Disassembly
	0x80000398	b9404ff6	LDR w22,[sp,#0x4c]
	0x8000039C	10002760	ADR x0,{pc}+0x4ec ; 0x80000888
➔	0x800003A0	9400038e	BL {pc}+0xe38 ; 0x800011d8
	0x800003A4	2a1603e1	MOV w1,w22
	0x800003A8	10002d60	ADR x0,{pc}+0x5ac ; 0x80000954
●	0x800003AC	9400038b	BL {pc}+0xe2c ; 0x800011d8
	0x800003B0	d2800019	MOV x25,#0
	0x800003B4	aa1903e0	MOV x0,x25
	0x800003B8	94000341	BL {pc}+0xd04 ; 0x800010bc
	0x800003BC	f9001700	STR x0,[x24,#0x28]
	0x800003C0	52800034	MOV w20,#1

The Disassembly view title bar has the following controls:

Address:

Enter a start address to display the code from.

Memspace:

Select **Secure** (TrustZone®) or **Normal** memory space, if applicable for the processor architecture.

Architecture

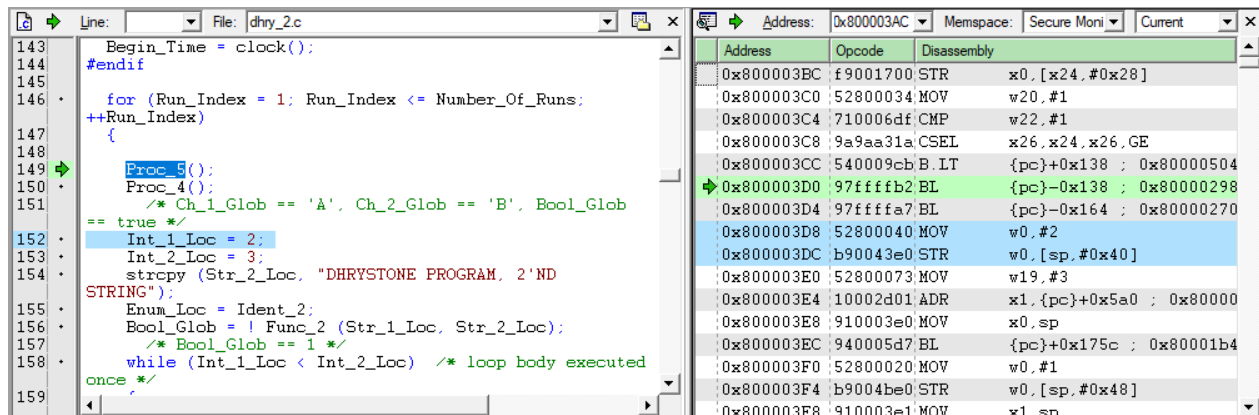
Select the disassembly mode or instruction set for the opcodes, such as **A32** or **T32**.

Mapping source lines to the disassembly listing

To highlight in blue the corresponding addresses in the disassembly view, click the left-most column in the source view. The highlighting indicates the disassembly instructions to which the respective source statement maps.



This action is only possible for source lines with a bullet point.

Figure 4-30: Matching source and disassembly

Context menu for Disassembly view

Right-click one in the Disassembly view to display the context menu. The menu has the following options:

Insert/Remove Breakpoint

Insert/Remove a breakpoint on the selected location only in the current shown memory space (TrustZone® world). The same can be achieved with a double click in the first column.

Insert/Remove Breakpoint into /from all Program Memories

Insert /Remove a breakpoint on the selected location in all program memory spaces.

Enable/Disable Breakpoint

Enable/Disable the breakpoint at the selected location.

Breakpoint Properties

If a breakpoint is present on the selected location, selecting this option displays the Breakpoint properties dialog box.

Show memory

Select a memory space and update the Memory view to display the memory contents at the address specified corresponding to the instruction location.

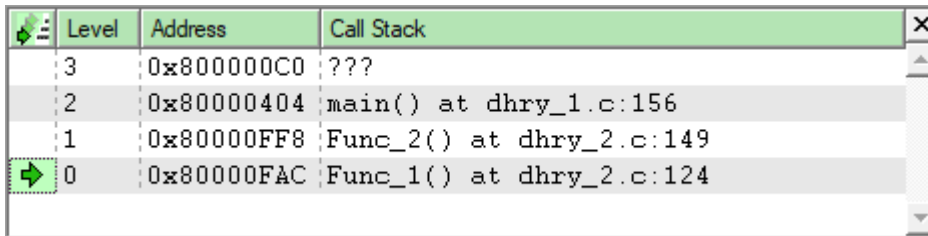
Run to here

Step the code until the selected location is reached.

4.5.2.3 Call Stack view

The Call Stack view displays the call history.

To use the Call Stack view, DWARF register mapping must be defined for the architecture and provided in the model.

Figure 4-31: Call Stack view

	Level	Address	Call Stack
	3	0x800000C0	???
	2	0x80000404	main() at dhry_1.c:156
	1	0x80000FF8	Func_2() at dhry_2.c:149
➡	0	0x80000FAC	Func_1() at dhry_2.c:124

**Note**

The loaded application must be an ELF file that contains a `.debug_frame` section. No other type of debug information is supported for the call stack view. The `.debug_frame` section must contain valid DWARF debug information that matches the DWARF 2 or DWARF 3 specification. The C compiler provides this information to describe all necessary information to unwind the stack:

- The stack pointer.
- How to retrieve the previous frame pointer.
- All registers that are involved in the unwinding process.

Only the frame section can supply this information.

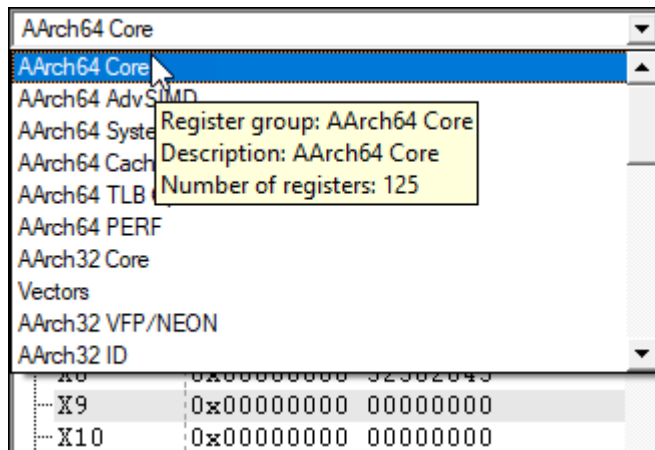
4.5.3 Debug views for registers and memory

This section describes the views that relate to register or memory contents.

4.5.3.1 Register views

This section describes the register debug views.

The **Register view** displays registers and their values and organizes them into multiple groups. A combo box enables switching between the groups that the target model predefines.

Figure 4-32: Select register group

For each register, a buffered state of the register (previous value) is stored. To view the contents, either:

- Use the context menu in the **Register view** and select **Show Previous Values**.

Figure 4-33: Register view showing current and previous contents

AArch64 Core	
Register	Value
X0	0x00000000 0000000E
prv:	0x00000000 00000009
X1	0x00000000 00000002
X2	0x00000000 00000001
X3	0x00000000 00000003
X4	0x00000000 49525453

- Or place the cursor over the respective register. The buffered state is updated every time the model execution stops.

Figure 4-34: Register view contents at cursor

AArch64 Core	
Register	Value
X0	0x00000000 0000000E
X1	0x00000000 00000002
X2	0x00000000 00000001
X3	0x00000000 00000003
X4	0x00000000 49525453
X5	0x00000000 DF00474E
X6	0x00000000 59524844

Context menu for Register view

Right-click one of the registers in the **Register view** to display the context menu. The menu has the following options:

Copy

Copy the contents of the selected register.

Add to Watch

Add the selected register to the Watch view.

Insert Breakpoint

Insert a breakpoint on the selected register.

Enable Breakpoint

Enable the breakpoint at the selected register.

Breakpoint Properties

If a breakpoint is present on the selected register, selecting this option displays the Breakpoint properties dialog box.

Edit Value

Edit the contents for the selected register.

Select and show memory at *nnn*

Select a memory space and update the **Memory view** to display the memory contents at the address that the register contents specify.

Show memory at *nnn*

Update the **Memory view** to display the memory contents at the address that the register contents specify.

Format

Choose the number base to use to display the register contents. The options are **Default Format**, **Unsigned Decimal**, **Signed Decimal**, **Hexadecimal**, **Binary**, **Float**, or **ASCII**.

Show Previous Value

Display the current value and the previous value for the selected register.

Select All

Select the registers in the Register view.

4.5.3.2 Memory view

This section describes the memory debug view.

The **Memory view** displays a range of memory starting from the base address that the address field (**Addr:**) specifies. Enter base addresses as decimal or hexadecimal numbers. Other fields allow for selection of the address space (**Space:**) and physical memory block (**Block:**).

Figure 4-35: Memory view

Addr:	0x00000044	Space:	Secure Monitor	Block:	All blocks												
0x00000000	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10
0x00000011	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10	00
0x00000022	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF
0x00000033	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7
0x00000044	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00
0x00000055	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8
0x00000066	00	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00
0x00000077	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00000088	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10

Context menu for Memory view

To display the context menu, right click one of the cells in the **Memory view**. The menu has these options:

Insert Breakpoint

Insert a breakpoint on the selected memory location.

Enable Breakpoint

Enable the breakpoint at the selected memory location.

Breakpoint Properties

If a breakpoint is present on the selected memory location, selecting this option displays the Breakpoint properties dialog box.

Edit Value

Edit the contents for the selected memory location.

Select and show memory at *nnn*

Select a memory space and update the **Memory view** to display the memory contents at the address that the contents of the memory location specify.

Show memory at *nnn*

Update the **Memory view** to display the memory contents at the address that the contents of the memory location specify.

Show disassembly at *nnn*

Update the disassembly view to display the disassembly contents at the address that the contents of the memory location specify.

Copy

Copy the contents of the selected memory location.

Add to Watch

Add the selected memory location to the Watch view.

Endian

Select the memory model to use to display memory contents. The options are: **Default Endian**, **Little Endian**, and **Big Endian**.

Format

Choose the number base to use to display the memory contents. The options are **Default Format**, **Unsigned Decimal**, **Signed Decimal**, **Hexadecimal**, **Binary**, **Float**, or **ASCII**.

Fixed column count

Display a fixed number of memory values per row. The width of the memory window determines the number to display.

Increment column count

Increment the number of memory values to display per row.

Decrement column count

Decrement the number of memory values to display per row.

Increment current address

Increment the start address that is used for each memory row.

Decrement current address

Decrement the start address that is used for each memory row.

Increment MAU per cell

Increase the size of the word, that is, the Minimum Addressable Unit (MAU), to be displayed in each memory cell. This option also changes the memory access size. If the chosen access size is not supported, Model Debugger defaults to a size of a single MAU.

Decrement MAU per cell

Decrease the size of the word, meaning MAU, to be displayed in each memory cell. This option also changes the memory access size. If the chosen access size is not supported, Model Debugger defaults to a size of a single MAU.

Load File to Memory ...

To load a binary or ASCII file into memory, use the Load File to Memory dialog box.

Save Memory in a File ...

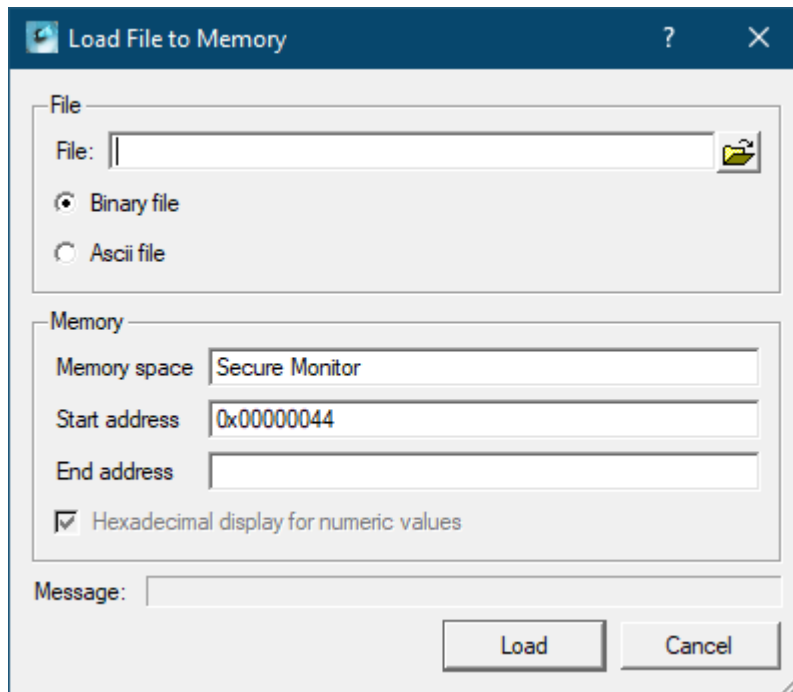
To save the contents of memory in a binary or ASCII file, use the Save Memory in a File dialog box.

Memory Display Options

To enable setting column count, view format, endian mode, and MAU per cell, use the Memory Display Options dialog box.

Load File to Memory and Store File to Memory dialog boxes

To load or store the memory contents of the target model, use these dialog boxes.

Figure 4-36: Load File to Memory dialog box

Enter the file name into the top field of the dialog box. You can use the button next to it to browse for the file. When loading or storing a binary or ASCII file, select the correct button. The Memory Space and Start Address fields are filled automatically from the memory view where you opened the dialog box. You can change the values. Put an end address into the bottom field. When loading a file, unless you enter a value here the maximum address of the memory is used.

If any problems occur, a message appears in the Message field.

4.5.3.3 Variables view

This section describes the Variables debug view.

Variables are displayed in these windows:

Local Variables window

This window shows all local variables and parameters that are valid for the current PC value, with their type and value.

- A blue letter L before the variable name indicates a local variable.
- A green letter P indicates a parameter.

Figure 4-37: Local Variable view

Local Variable/Parameter	Type	Value
P width	int	0x00000280 (640)
P height	int	0x000001E0 (480)
L __result	int	0x00000280 (640)
L VE_A_SYSREG_BASE	const unsigned int	0x1C010000 (469827584)
L VE_A_CLCD_BASE	const unsigned int	0x1C1F0000 (471793664)
L VE_A_FRAME_BASE	const unsigned int	0x80100000 (-2146435072)
L VE_DVI_MUX	const unsigned int	0x00000000 (0)

Global Variables window

This window shows the global variables with their types and values. A green letter G marks them.

Figure 4-38: Global Variable view

Global Variable	Type	Value
G buff	ch...	0x00100000 -> ""
*buff	char	0x00 { 0 } '\x00'
G cx	long	0xFFEE2B74
G cy	long	0x00000000 {0}
G dx	long	0x00003236 (12854)
G dy	long	0x0000191B (6427)

Complex values such as structs and arrays or pointers can be expanded by clicking the small cross before the variable name.



Note

To use the variables windows, the loaded application must be an ELF file that contains `.debug_info` and `.debug_abbrev` sections. No other type of debug information is supported for this view. The `.debug_info` section must contain valid DWARF debug information that matches the DWARF 2 or DWARF 3 specification. The model must provide a PC register to enable locating local variables.

For applications that have more than one compilation unit, the units are only loaded when the PC reaches the respective context.

The loading of these compilation units can be triggered manually by selecting **Load Debug Info for Module** from the **Debug** menu. Right-click on one of the variables windows and select **Load Debug Info for Module**.

The displayed dialog box lists the compilation units that can be loaded.

Context menu for the Variable view

To display the context menu, right click one of the items in the Global or Local Variable view. The menu has these options:

Copy

Copy the contents of the selected variable.

Add to Watch

Add the selected variable to the Watch view.

Insert Breakpoint

Insert a breakpoint on the selected variable.

Enable Breakpoint

Enable the breakpoint at the selected variable.

Breakpoint Properties

If a breakpoint is present on the selected variable, selecting this option displays the Breakpoint properties dialog box.

Edit Value

Edit the contents of the selected variable.

Show memory

Select a memory space and update the **Memory view**. You can display the memory contents at the address of the value of the variable.

Show Previous Value

Display the current value and the previous value for the selected variable.

Select All

Select the variables in the variable view.

Load Debug Info for Module

Load debug information for the module that contains the selected variable.

4.5.4 Debug views for pipelines

Model Debugger provides options for viewing the pipeline.

The options are:

- The Pipeline Overview window.
- The Pipeline Table.

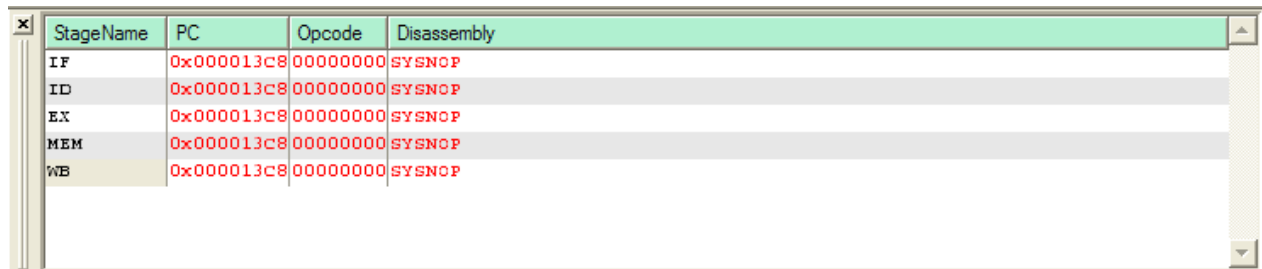
Pipeline views are only available if your model supports them. If the pipeline icons are gray, not orange, then you cannot view pipeline information.

4.5.4.1 Pipeline Overview window

The Pipeline Overview window presents the main details of every pipeline stage.

The Pipeline Overview contains the name, program counter, opcode, and disassembly for the stages.

Figure 4-39: Pipeline Overview window



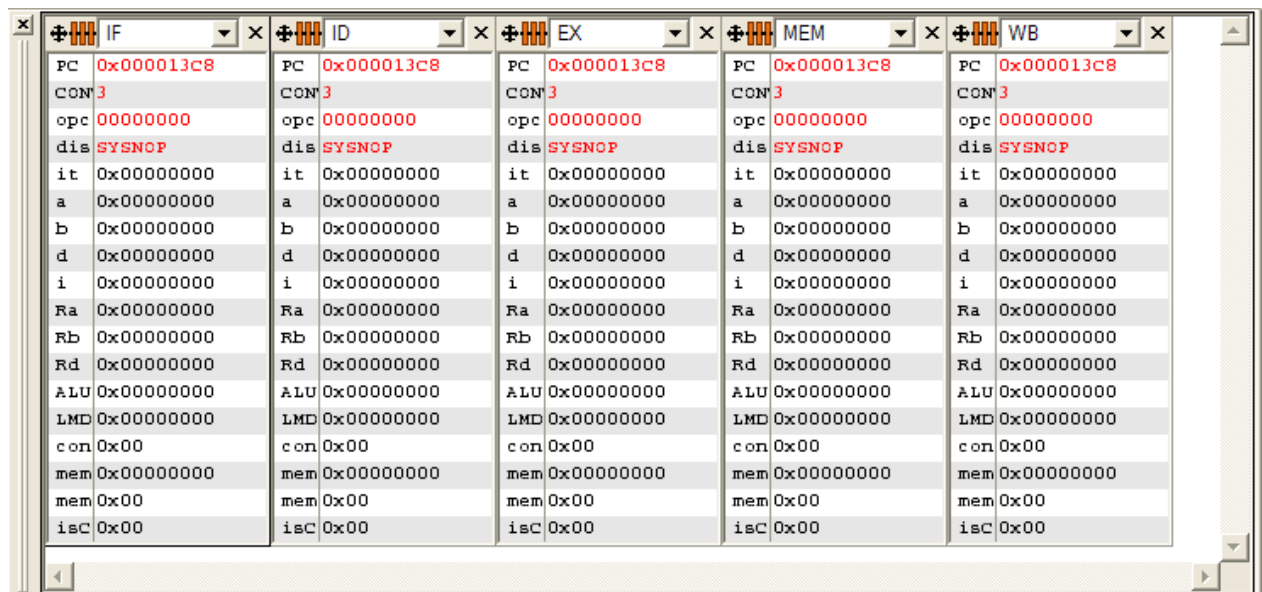
StageName	PC	Opcode	Disassembly
IF	0x000013c8	00000000	SYSNOP
ID	0x000013c8	00000000	SYSNOP
EX	0x000013c8	00000000	SYSNOP
MEM	0x000013c8	00000000	SYSNOP
WB	0x000013c8	00000000	SYSNOP

4.5.4.2 Pipeline Table window

The Pipeline Table gives a detailed view of the pipeline stages.

By default, the table shows views for all pipeline stages. Each of the detailed entries has a name and value field.

Figure 4-40: Pipeline Table window

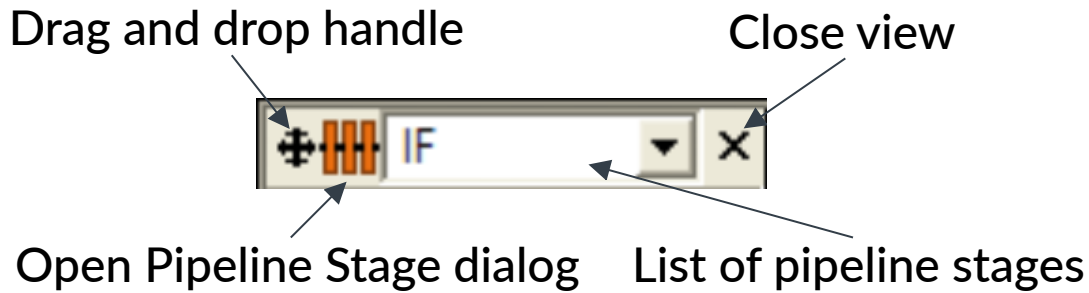


IF	ID	EX	MEM	WB
PC 0x000013c8	PC 0x000013c8	PC 0x000013c8	PC 0x000013c8	PC 0x000013c8
CON 3	CON 3	CON 3	CON 3	CON 3
opc 00000000	opc 00000000	opc 00000000	opc 00000000	opc 00000000
dis SYSNOP	dis SYSNOP	dis SYSNOP	dis SYSNOP	dis SYSNOP
it 0x00000000	it 0x00000000	it 0x00000000	it 0x00000000	it 0x00000000
a 0x00000000	a 0x00000000	a 0x00000000	a 0x00000000	a 0x00000000
b 0x00000000	b 0x00000000	b 0x00000000	b 0x00000000	b 0x00000000
d 0x00000000	d 0x00000000	d 0x00000000	d 0x00000000	d 0x00000000
i 0x00000000	i 0x00000000	i 0x00000000	i 0x00000000	i 0x00000000
Ra 0x00000000	Ra 0x00000000	Ra 0x00000000	Ra 0x00000000	Ra 0x00000000
Rb 0x00000000	Rb 0x00000000	Rb 0x00000000	Rb 0x00000000	Rb 0x00000000
Rd 0x00000000	Rd 0x00000000	Rd 0x00000000	Rd 0x00000000	Rd 0x00000000
ALU 0x00000000	ALU 0x00000000	ALU 0x00000000	ALU 0x00000000	ALU 0x00000000
LMD 0x00000000	LMD 0x00000000	LMD 0x00000000	LMD 0x00000000	LMD 0x00000000
con 0x00	con 0x00	con 0x00	con 0x00	con 0x00
mem 0x00000000	mem 0x00000000	mem 0x00000000	mem 0x00000000	mem 0x00000000
mem 0x00	mem 0x00	mem 0x00	mem 0x00	mem 0x00
isc 0x00	isc 0x00	isc 0x00	isc 0x00	isc 0x00

Columns and rows can be resized by grabbing the lines between them and dragging them.

The cross in the top left corner of every pipeline stage view is the starting point for drag and drop operations. A view can be copied or moved to an empty table cell. If it is dragged beyond the boundaries of the table, a new row or column is added in the direction of the drag.

Figure 4-41: Pipeline Table icons



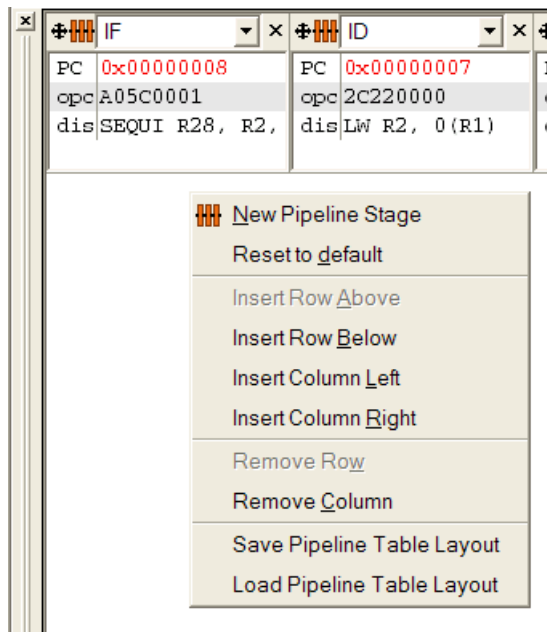
Double click the first column of a view to set or remove a breakpoint on the field.

Double click the second column of a view to open the field for inline edit of the value. You cannot, however, perform an inline edit of an opcode or disassembly.

Pipeline Table context menu

Right click the empty space in the table or in an empty cell to open the context menu.

Figure 4-42: Pipeline Table context menu



The context menu has the following entries:

New Pipeline Stage

Create a stage and add it to the view.

Reset to default

Use the default layout for the Pipeline Table view.

Insert Row Above/Insert Row Below

Insert a new row above or below the current cell.

Insert Column Left/Insert Column Right

Insert a new column to the left or right of the current cell.

Remove Row/Remove Column

Remove the row or column that includes the current cell.

Save Pipeline Table Layout

Save the current layout.

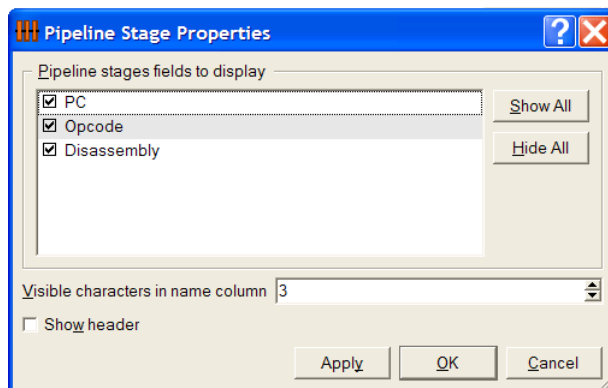
Load Pipeline Table Layout

Load a previously saved layout and use that configuration in the Pipeline Table view.

Pipeline Stage Properties dialog box

To open the Pipeline Stage Properties dialog box, click the orange icon to the right of the cross. You can customize the lists in the Pipeline table with this dialog box. The combo box to the right of the orange icon lists all pipeline stages that are available for the current model. The chosen pipeline stage is displayed in the view underneath. To remove the view from the cell, click the X, located in the right top corner of each list.

Figure 4-43: Pipeline Stage Properties dialog box



Choose the fields to be displayed in the pipeline stage list by checking the boxes or clicking the **Show All** or **Hide All** buttons.

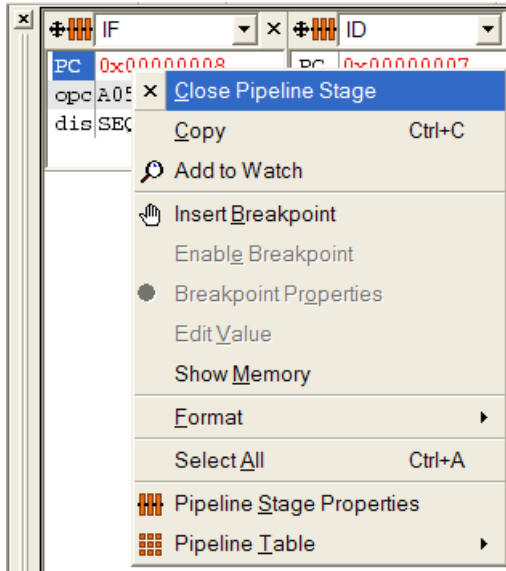
The size of the first column can be set in the **Visible characters in the name column** control.

The optional header can be switched on and off by checking the **Show header** check box.

Context menu for an entry in the Pipeline Table

Right click an item in the Pipeline view lists to open the context menu.

Figure 4-44: Pipeline view context menu



The context menu has the following entries:

Close Pipeline Stage

Select to close the pipeline stage.

Copy

Select to copy the pipeline field. The field can be pasted into the Watch window.

Add to Watch

Select to add the pipeline field to the Watch window.

Remove Breakpoint/Insert Breakpoint

The text that appears depends on whether the selected field already has a breakpoint:

- If a breakpoint is present, select **Remove Breakpoint** to delete it.
- If a breakpoint is not present, select **Insert Breakpoint** to add a breakpoint.

Enable Breakpoint/Disable Breakpoint

If a breakpoint is present:

- Select **Enable Breakpoint** to enable it
- Select **Disable Breakpoint** to retain the breakpoint, but disable it.

Breakpoint Properties

View and set the details and conditions for a particular breakpoint.

This dialog box is also available from the Breakpoint Manager dialog box.

Edit Value

Select to open the chosen field for inline edit.

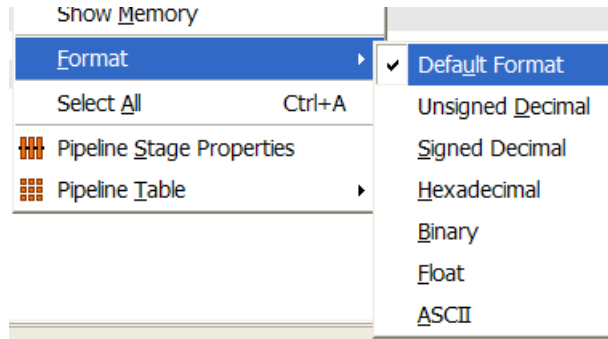
Show Memory

Select to mark the memory address in the Memory Window.

Format

Select to open the submenu. This menu lets you choose the format for number display.

Figure 4-45: Submenu for display format



Select All

Selects all fields in the list.

Pipeline Stage Properties

To change the contents of the Pipeline Stage view, open the Pipeline Stage Properties dialog box.

Pipeline Table

Select to display a submenu for the Pipeline Table.

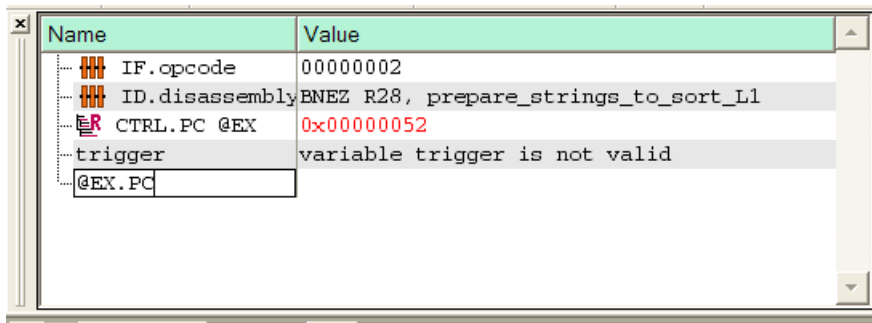
Related information

[Breakpoint Manager dialog box](#) on page 145

4.5.5 Watch window and Expression Evaluator

The Expression Evaluator is located in the **Watch window**.

To display the window, select **Window > New View > New Watch Window**.

Figure 4-46: Watch window

There are two types of entry in the **Watch window**:

System variables

Entries in this group are marked with small icons to the left of their name to indicate their origin. They can be manipulated in the **Watch window** in the same way as in their original view. Items in this category include:

- Registers.
- Memory locations.
- Pipeline fields.
- Variables from source code.

Expressions for evaluation

These items do not have an associated icon because they are not duplicates of an item in a different view. They cannot have breakpoints set and their value cannot be changed. However, you can edit the expression itself by text in the **Name** column.

Double click in the left column of an existing entry to add a breakpoint for that variable.

Double click in the right column of an existing entry to edit the contents.

Double click on the last entry in the left column to enter a new expression. Press the **Enter** key to evaluate the expression.

The following rules apply to the names of the resources in the target:

- Registers must be entered in the form:

```
$registerGroup.registerName
```

If the register name is unique for the whole target, the following shorthand notation can be used:

```
$registerName
```

- Pipeline stage fields must be entered in the form:

```
@pipelineStage.fieldName
```

- Memory locations must be entered in the form:

```
memorySpace:address
```

The content of a memory location is queried with the following expression:

```
*memorySpace:address
```

The delivered pointer can be type cast into any required type:

```
(typeName*)memorySpace:address
```

The variables of the software running on top of the target processor can be entered using an expression as they appear in the software. They do not require a prefix or quotes. Access components of structs or unions with the '.' or '->' operator according to the C syntax.

Numeric values can be entered in the following formats:

Integer values

Integer values can have binary, octal, decimal, or hexadecimal representation. The prefix indicates the representation format:

- Binary numbers have a leading `0b`.
- Octal numbers a leading `0`.
- Hexadecimals have a leading `0x`.
- Literals with no prefix are interpreted as decimals.

Floating-point values

Floating-point values can have decimal and scientific representation.

Enter floating-point values in decimal representation (123.456) or in scientific representation with positive or negative exponent (1.23456e2).

To form complex expressions, combine resources, variables, and literals in the target with operators. The expression evaluator has the same operands as the C language and has the same precedence and associativity of operators. Inside the complex expression, the resources of the target can be used if an integer value would be sufficient in a regular C expression.

Table 4-2: Operator precedence

Precedence	Operators	Associativity
1	[] -> .	Left to right
2	! ~ + - * & (unary) (cast) sizeof	Right to left
3	* (binary) / %	Left to right

Precedence	Operators	Associativity
4	+ - (binary)	Left to right
5	<< >>	Left to right
6	< <= > =>	Left to right
7	== !=	Left to right
8	& (binary)	Left to right
9	^	Left to right
10		Left to right
11	&&	Left to right
12		Left to right
13	?:	Left to right

4.5.5.1 Context menu for Watch window

To display the context menu, right click one of the values in the Watch window.

The menu has these options:

Paste

Insert a copied memory, register, or variable into the Watch window.

Copy

Copy an item and its value from the Watch window.

Insert Breakpoint

Insert a breakpoint on the selected watched item.

Enable Breakpoint

Enable the breakpoint at the selected watched item.

Breakpoint Properties

If a breakpoint is present on the selected watched item, selecting this option displays the Breakpoint properties dialog box.

Edit Value

Edit the contents for the selected watched item.

Select and show memory at nnn

Select a memory space and update the Memory view to display the memory contents at the address that the contents of the watched item specify.

Show memory at nnn

Update the Memory view to display the memory contents at the address that the contents of the watched item specify.

Format

Choose the number base to use to display the watched item. The options are **Default Format**, **Unsigned Decimal**, **Signed Decimal**, **Hexadecimal**, **Binary**, **Float**, or **ASCII**.

Increment number of bytes

Increment the number of memory addresses to display.

Decrement number of bytes

Decrement the number of memory addresses to display.

Select all

Select all of the watched items.

4.5.6 Breakpoint dialog boxes

Control and configure breakpoints using the **Breakpoint Manager** and **Breakpoint Properties** dialog boxes.

Related information

[Setting and removing breakpoints](#) on page 100

4.5.6.1 Breakpoint Manager dialog box

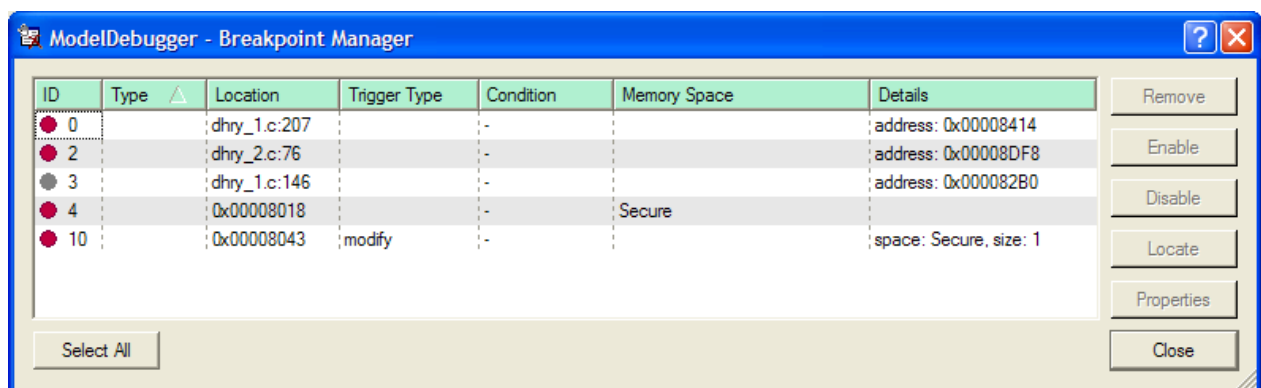
This dialog box lists all breakpoints and provides the breakpoint target location, condition, and target details.

Breakpoints that are hit are highlighted in the breakpoint list with an orange background. The breakpoint is also highlighted in the original view for the item.

In the breakpoint list, select an item to:

- Enable, disable, or remove breakpoints.
- Locate the breakpoint target location in the respective debug view.
- Modify breakpoint conditions using the **Properties** button.

Figure 4-47: Breakpoint Manager dialog box



4.5.6.2 Breakpoint Properties dialog box

Open the Breakpoint Properties dialog box by right clicking on a breakpoint and then selecting **Properties** from the resulting context menu, or with the Breakpoint Manager.

Select the breakpoint criteria with the Breakpoint Properties dialog box:

Ignore count

Enter the number of occurrences to ignore before triggering the breakpoint. Enter 0 to trigger a breakpoint for every occurrence.

Enable breakpoint

To enable the breakpoint, check this box. If unchecked, the breakpoint location and type is stored, but occurrences do not trigger a breakpoint.

Continue Execution after hit

To enable the continuation of execution after breakpoint hit, check this box. If checked the execution of the debugged application does not stop when the breakpoint is being hit.

Resource

Select the condition that results in a breakpoint being triggered. Conditional breakpoints are not supported for some types of breakpoint object.

Value

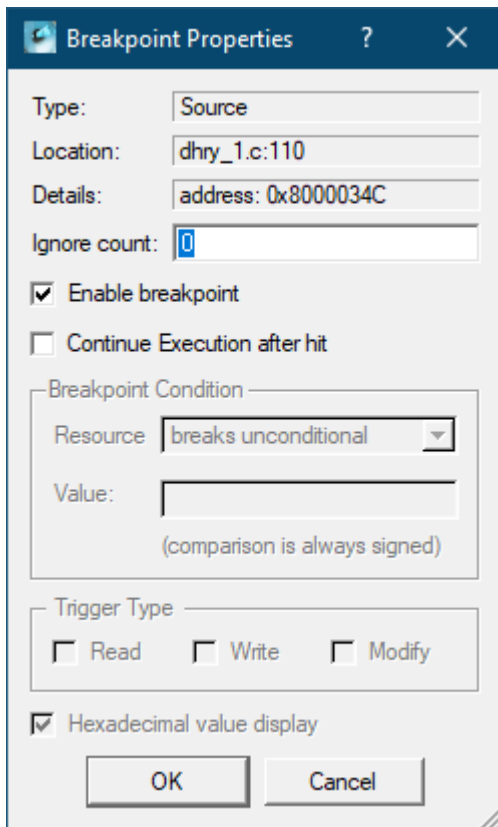
If the **Resource** type is not **breaks unconditional**, select the comparison value that is to trigger the breakpoint.

Trigger Type

Select whether a **Read**, **Write**, or **Modify** operation triggers the breakpoint. These check boxes are not enabled for some types of breakpoint object.

Hexadecimal value display

Check to display the contents of the **Value** field in hexadecimal format. If unchecked, decimal format is used.

Figure 4-48: Breakpoint Properties dialog box

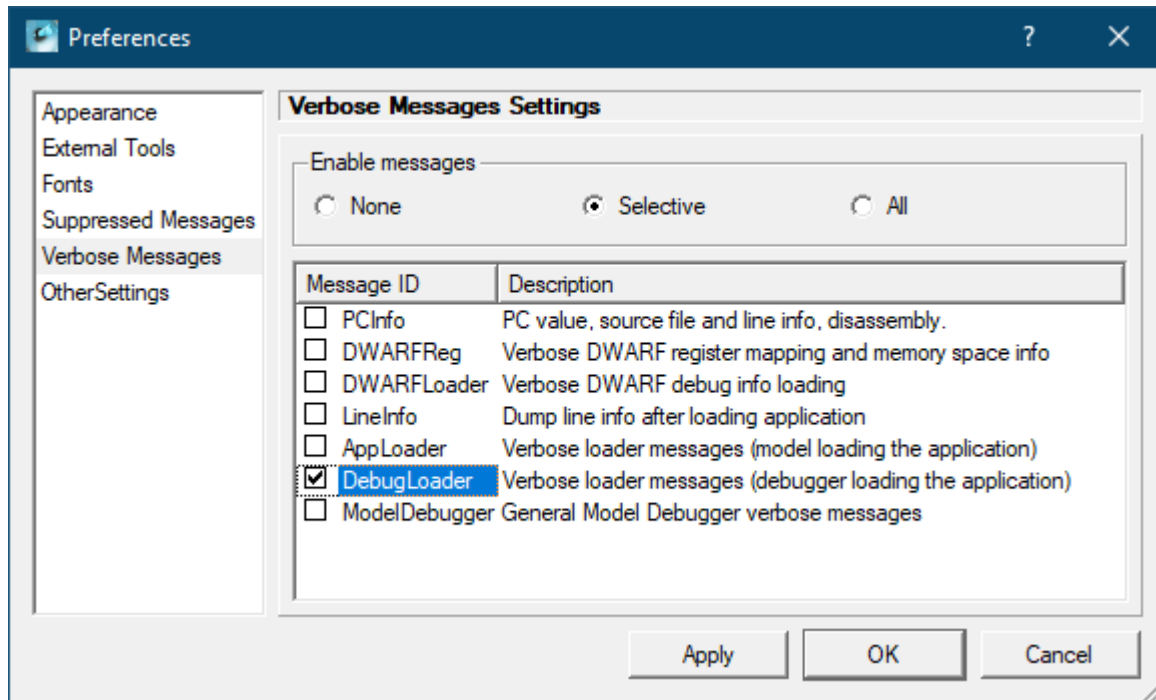
Related information

[Setting and removing breakpoints](#) on page 100

4.5.7 Preferences dialog box

Configure the behavior of Model Debugger with the Preferences dialog box.

Figure 4-49: Preferences dialog box



To display the options for that category, select an entry in the list on the left side of the dialog box:

Appearance

Each option has a checkbox:

- **Show tool tips** enables display of pop-up help for a control.
- **Display toolbar text labels** displays text below the icon.
- **Word wrap in source window** wraps long lines to fit the window.
- **Show splash screen on startup** displays the information screen.
- **Reload recent layout on startup** keeps your last used layout.

Use the controls in the **Recent files and directories** to control how many previously used files and directories are displayed.

External Tools

The **use operating system file associations** checkbox is only available on Windows, and is selected by default. This setting inactivates the external tool edit fields and buttons. To activate these fields, clear the checkbox.



The default external tools are different on Linux.

Configure the display of the documentation with these settings. Access the documentation through:

- The **Help** menu item. You access the PDFs for the Model Debugger and Fast Models this way.
- The `documentation_file` property in a component property listing. This property might point to a PDF file, a text file, an HTML file, or `http://` link.

You can change the default external tools. Click the folder icon to open a browser, or use the drop-down list to choose a previously selected executable.

Fonts

You can specify the fonts for each of the windows.

To control the fonts with the `$DISPLAY` variable, check **Fonts depend on \$DISPLAY variable**.

Suppressed Messages

Lists the suppressed messages and enables you to specify an action for each message.

Verbose Messages

Turn on or off verbose message setting for the message IDs. To turn on or off individual messages, click **Selective**.

Other settings

Each option has a checkbox:

- **Load all Compilation Units at Startup:** load all required files.
- **Show Parameter Dialog at Startup:** display the dialog box to configure model parameters.
- **Show Target Dialog at Startup:** display dialog that normally appears when a model is loaded. If unchecked, Model Debugger automatically connects to targets that have the `executes_software` flag set.
- **Enable SMP Application Loading:** have Model Debugger load the application once into memory and load only debug information for all processors. The PC is set to the value of the first processor in all processors.

After displaying the dialog box and modifying preferences:

- Click **Apply** to apply the settings and keep the dialog box open.
- Click **OK** to apply the settings and close the dialog box.
- Click **Close** to close the dialog box. Unapplied settings changes are lost.

4.5.8 Keyboard shortcuts

This section describes the keyboard shortcuts.

Table 4-3: Keyboard shortcuts

Modifier	Key	Function
-	Esc	Close the current dialog.
-	F1	Help.
-	F3	Find next.
Shift	F3	Find previous.
-	F5	Run target.
Shift	F5	Stop target.
-	F10	Source-level step over.
Ctrl	F10	Source-level step out. This action leaves the current function.
-	F11	Source-level step into.
Shift	F11	Instruction-level step.
Shift	F10	Instruction-level step over.
Ctrl +Shift	F11	Instruction-level step out.
Ctrl	F11	Cycle step.
Ctrl +Shift	F11	Cycle step N.
Ctrl	B	Open the breakpoint manager.
Ctrl +Shift	C	Connect to model.
Ctrl +Shift	D	Load debug information from application code.
Ctrl	F	Search (find) operation.
Ctrl +Shift	L	Load model library.
Ctrl +Shift	M	Go to function <code>main()</code> .
Ctrl	O	Multi-functional open. If no target is loaded, a dialog is displayed to select the model library and application code. If a target is loaded, the source file is opened.
Ctrl +Shift	O	Load application code.
Ctrl	P	Open the Model Parameter dialog.
Ctrl +Shift	P	Pause or continue source step.
Ctrl	Q	Close Model Debugger.
Ctrl	R	Reset target only.
Ctrl +Shift	R	Reset target and reload application.
Ctrl	S	Save the current session.

Modifier	Key	Function
Ctrl	T	Select target.
Ctrl	U	User preferences dialog.
Ctrl	W	Close model.

5. Model Shell

Model Shell is a command-line tool for configuring and running Component Architecture Debug Interface (CADI)-compliant models.



Arm deprecates Model Shell. Use Integrated SIMulators (ISIMs) instead.

Model Shell launches CADI-compliant models and provides:

- Semihosting stdio.
- CADI logging.
- A launch platform for debuggers, profilers, and operating environments.

Model Shell can start a CADI server to enable other debuggers to connect to the model in the following ways:

- The simulation is initialized, but does not run. An external debugger must control the simulation. This is the default behavior.
- The simulation is initialized and runs immediately. An external debugger can connect to the simulation after it starts.

Model Shell provides semihosting input and output only for standard streams:

- When a CADI server is started, semihosting output goes to the Model Shell console and to all debuggers.
- If a debugger is attached, it performs semihosting input. If not, Model Shell provides the input.

Model Shell is included in the Fast Models package and in the Fast Models FVP Standard Library package. For installation information for the Fast Models package, see [Installation](#) in the Fast Models User Guide.

5.1 ISIM targets

An Integrated SIMulator (ISIM) target is a simulation executable that SimGen generates from a LISA + model description. It runs standalone, without the need for Model Shell or Model Debugger. SimGen generates an ISIM by linking the executable against the SystemC shared library.

All Model Shell command-line options, except `--mode1`, can be used with an ISIM target. `--mode1` is not required because the model is integrated into the ISIM.

Related information

[Fast Models User Guide](#)

5.2 Model Shell command-line syntax

To start Model Shell from the command line, type `model_shell` with any options.

```
model_shell -m <model> [<options>]
```

model

File name of the model, including `.so` or `.dll` extension.



Build a `.so` or `.dll` library from the `.sgproj` file for the model, using System Canvas.

options

List of command-line options.

Related information

[Model Shell command-line options](#) on page 153

[Configuration file syntax for specifying model parameters](#) on page 155

[Fast Models User Guide](#)

5.3 Model Shell command-line options

Use these options to control Model Shell behavior from the command line.

Table 5-1: Model Shell command line options

Long	Short	Description
<code>--application <file1> <file2></code>	<code>-a</code>	Load application list. Use <code>-a [INST=] file</code> to load an application into a system instance: <pre>model_shell \$MODEL -a cluster0.cpu0=app1.axf -a cluster0.cpu1=app2.axf</pre> Use <code>*</code> to load the same application image into all the cores in a cluster: <pre>model_shell \$MODEL -a "cluster0.*=app.axf"</pre> Unless you specify a core, Model Shell loads the image into the first cluster or instance that can run software.
<code>--break <address></code>	<code>-b</code>	Set program breakpoint at the address. Use <code>-b [INST=] [threadid:] address</code> to set a breakpoint for the system instance. Multiple <code>--breaks</code> set multiple breakpoints.
<code>--cadi-log</code>	<code>-L</code>	Log all CADI calls to an XML log file.
<code>--cadi-server</code>	<code>-S</code>	Start a CADI server. This enables attaching a debugger to debug targets in the simulation. To shut down the server, return to the command window that you used to start the model and press Ctrl+C . The Model Shell process must be in the foreground before you can shut it down.

Long	Short	Description
--cadi-trace	-t	Enable diagnostic output from CADI calls and callbacks.
--config-file <file>	-f	Use model parameters from a configuration file.
--cpulimit <n>	-	Specify the maximum number of host seconds for the simulation to run, excluding simulation startup and shutdown. Fractions of a second can be specified, but the remaining time is only tested to a resolution of 100ms. If <i>n</i> is omitted, the default is unlimited.
--cyclelimit <n>	-	Specify the maximum number of cycles to run. If <i>n</i> is omitted, the default is unlimited.
--data <file>@address	-	Specify raw data to load at this address. The full format is: --data [INST=] file@[memspace:] address
--dump <file>@<address>,<size>	-	Dump a section of memory into a file at model shutdown. Multiple --dumps are possible. The full format is: --dump [INST=] file@[memspace:] address,size
--help	-h	List the Model Shell command-line options, and then exit.
--keep-console	-K	Keep console window open after completion. Microsoft Windows only.
--list-instances	-	Print list of target instances to standard output.
--list-memory	-	Print memory information for the model to standard output.
--list-params	-l	Print list of target instances and their parameters to standard output. Use this to help identify the correct syntax for configuration files, and to find out what the target supplies.
--list-regs	-	Print model register information to standard output.
--model <file>	-m	Model to load. Optional.
--output <file>	-o	Redirect output from the --list-instances, --list-memory, --list-params, and --list-regs commands to a file. The contents of the file are formatted correctly for use as input by the --config-file option.
--parameter [<inst>.]<param>=<value>	-C	Set a parameter to this value. For hierarchical systems, specify the complete name of the parameter.
--plugin <file>	-	Specify plugins. These plugins or those in environment variable FM_PLUGINS are loaded.
--prefix	-P	Prefix semihosting output with the name of the target instance.
--print-port-number	-	Prints the port number on which the CADI server is listening.
--quiet	-q	Suppress Model Shell output.
--run	-R	Run simulation on load, with a CADI server: -S --run. The default is to start the simulation in a stopped state.
--start <address>	-	Initialize the PC to this application start address, overriding the .axf start. The full format is: --start [INST=] address
--stat	-	Print statistics at the end of the simulation.

Long	Short	Description
<code>--timelimit <n></code>	<code>-T</code>	Specify the maximum number of wall clock seconds for the simulator to run. If <i>n</i> is omitted, the default is unlimited. If <i>n</i> is specified as 0, Model Shell initializes the system, loads all applications and data, sets breakpoints and PC, and exits immediately without running the model. Use this option to convert applications to raw binary. For example: <code>model_shell --timelimit 0 -m mymodel.dll -a app.axf --dump app.raw@0x8000,0x10000</code>
<code>--verbose</code>	<code>-V</code>	Enable verbose messages.
<code>--version</code>	<code>-v</code>	Print the Model Shell version number, then exit.

Related information

[Automatic Model Shell shutdown](#) on page 157

5.4 Configuration file syntax for specifying model parameters

Text files can configure models for Model Shell from the command line, thus setting many parameters at once.

```
model_shell --config-file my_configuration_file.txt ...
```

Each line of the configuration file must have the same *instance.parameter=value* syntax as used for command-line assignments.

Include comment lines and blank lines in configuration files with a # character before the comment or blank text.

To generate a configuration file, use the `--list-instances` and `--list-params` options on the command line. The command line can also include parameter assignments.

Example

```
model_shell --list-params --list-instances -C top-mm=0x3 -o file.config -m model.so
```

might generate:

```
# Instances:
# Instance id: instance name (SW: y/n, component, type, version) : description
# instance.parameter=value # (type, mode) default = 'value' : description
# : [min..max]
#-----
# Instance 0: (SW: no , NoCore, , 1.0) : Regression test system without PVLIB
usage.
  top-p=0x2 # (int , init-time) default = '0x2' : test display name
  top-str="empty" # (string, init-time) default = 'empty' : test string param
  top-mm=0x3 # (int , init-time) default = '0x6' : test min(2) max(6)
  param : [0x2..0x6]
# Instance 1: a1 (SW: no , A, , 1.0) :
```

```

a1.p1=0x2          # (int , init-time) default = '0x2' : A parameter p1
a1.p2=0            # (bool , run-time ) default = '0'   : A parameter p2
# Instance 2: a1.b  (SW: no , B, , 1.0) :
a1.b.p1=0x2        # (int , init-time) default = '0x2' : B parameter p1
a1.b.p2=""         # (string, run-time ) default = ''   : B parameter p2
# Instance 3: a2    (SW: no , A, , 1.0) :
a2.p1=0x2          # (int , init-time) default = '0x2' : A parameter p1
a2.p2=0            # (bool , run-time ) default = '0'   : A parameter p2
# Instance 4: a2.b  (SW: no , B, , 1.0) :
a2.b.p1=0x2        # (int , init-time) default = '0x2' : B parameter p1
a2.b.p2="test"     # (string, run-time ) default = ''   : B parameter p2
#-----

```

This is another way of specifying run-time parameters:

```

# Disable semihosting using true/false syntax
coretile.core.semihosting-enable=false
#
# Enable VFP at reset using 1/0 syntax
coretile.core.vfp-enable_at_reset=1
#
# Set the baud rate for UART 0
baseboard.uart_0.baud_rate=0x4800

```

5.5 SMP support

Model Shell provides Symmetric MultiProcessing support. It can be simple or standard.

Simple

This is only suitable for model systems that have one SMP cluster. The same application is loaded in all cores.

```
model_shell -m smp_model.so -a app.axf
```

Standard

This is suitable for all cases and uses the `-a` option to list the applications for each core.

Use the full instance name of each core.

```
model_shell -m smp_model.so -a multiprocessor.processor0=app1.axf -a
multiprocessor.processor1=app2.axf
```

In addition to loading individual applications for each core, the `-a` option also enables loading the same application in all cores.

Replace the index of the core with `*`.

```
model_shell -m smp_model.so -a multiprocessor.processor*=app.axf
```

```
model_shell -m smp_model.so -a "multiprocessor.*=app.axf"
```



On Unix, the * character requires escape quotes.

5.6 Manual Model Shell shutdown

User actions that stop Model Shell.

Press **ctrl+c**.

The program starts shutting down the simulator and exits after shutdown is complete. On a second press, Model Shell terminates immediately. Some models can assign their own **ctrl+c** handlers that override Model Shell behavior.

Press **ctrl+Break**.

Model Shell terminates immediately. Windows only.

Close an LCD window.

The simulation stops.

5.7 Automatic Model Shell shutdown

Command-line options that control when to stop Model Shell. The first fulfilled condition stops Model Shell. If none are specified, Model Shell shuts down when the simulation ends.

--break

Breakpoint. `--cadi-server` overrides this option.

--cyclelimit

Cycles > cycle limit. `--cadi-server` overrides this option. This option might reduce execution speed. It ignores breakpoints.

--timelimit

Time > running time limit.

--cpulimit

Time > process time limit. Tested to a granularity of 0.1s to avoid performance loss. Elapsed processor time includes user time and kernel time.

5.8 License checking messages from Model Shell and ISIM systems

The license checking messages appear in the `stderr` and `stdout` outputs, and are useful for the detection and diagnosis of licensing issues.

The `model_shell` and `isim_system` executables return a status value when they exit:

0

no error (for example, clean simulator shutdown).

1

error (for example, license checking or file not found).

For exit status 1, parse the `stderr` output. A message might, for example, appear in the GUI with other `WARNING`, `ERROR` or `Fatal Error` messages. See the lines that follow for more information from the license checking module. When a license is about to expire, Model Shell prints a warning message to `stdout`, but the simulation still starts correctly.

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.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

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
1126-00	19 June 2024	Non-Confidential	Update for v11.26.
1125-00	13 March 2024	Non-Confidential	New document for v11.25.

For technical changes to this documentation, 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
<i>italic</i>	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 <u>underline</u>	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: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . 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.



Tip

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



Remember

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.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Table 1: Arm publications

Document name	Document ID	Licensee only
Component Architecture Debug Interface User Guide	100963	Non-Confidential
Fast Models Reference Guide	100964	Non-Confidential
Fast Models User Guide	100965	Non-Confidential
User-based Licensing User Guide	102516	Non-Confidential