arm

Model Debugger for Fast Models

Version 11.20

User Guide

Model Debugger for Fast Models

User Guide

Release Information

Document history

Issue	Date	Confidentiality	Change
1100-00	31 May 2017	Non-Confidential	Update for v11.0. Document numbering scheme has changed.
1101-00	31 August 2017	Non-Confidential	Update for v11.1.
1102-00	17 November 2017	Non-Confidential	Update for v11.2.
1103-00	23 February 2018	Non-Confidential	Update for v11.3.
1104-00	22 June 2018	Non-Confidential	Update for v11.4.
1104-01	17 August 2018	Non-Confidential	Update for v11.4.2.
1105-00	23 November 2018	Non-Confidential	Update for v11.5.
1106-00	26 February 2019	Non-Confidential	Update for v11.6.
1107-00	17 May 2019	Non-Confidential	Update for v11.7.
1108-00	5 September 2019	Non-Confidential	Update for v11.8.
1109-00	28 November 2019	Non-Confidential	Update for v11.9.
1110-00	12 March 2020	Non-Confidential	Update for v11.10.
1111-00	9 June 2020	Non-Confidential	Update for v11.11.

Issue	Date	Confidentiality	Change
1112-00	22 September 2020	Non-Confidential	Update for v11.12.
1113-00	9 December 2020	Non-Confidential	Update for v11.13.
1114-00	17 March 2021	Non-Confidential	Update for v11.14.
1115-00	29 June 2021	Non-Confidential	Update for v11.15.
1116-00	6 October 2021	Non-Confidential	Update for v11.16.
1117-00	16 February 2022	Non-Confidential	Update for v11.17.
1118-00	15 June 2022	Non-Confidential	Update for v11.18.
1119-00	14 September 2022	Non-Confidential	Update for v11.19.
1120-00	7 December 2022	Non-Confidential	Update for v11.20.

Proprietary Notice

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

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.

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.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction

This document describes how to use the Model Debugger GUI for debugging CADI-compliant models.

1.1 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

Convention	Use
italic	Citations.
bold	Highlights interface elements, such as menu names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u>mono</u>`space`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

1.2 Other information

See the Arm® website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.

- Arm® Glossary.

1.3 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.

Arm® product resources	Document ID	Confidentiality
Component Architecture Debug Interface User Guide	100963	Non-Confidential
User-based Licensing User Guide	102516	Non-Confidential

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com

2. Introduction to 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 and 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.

Figure 2-1: A Model Debugger session: main window with debug windows
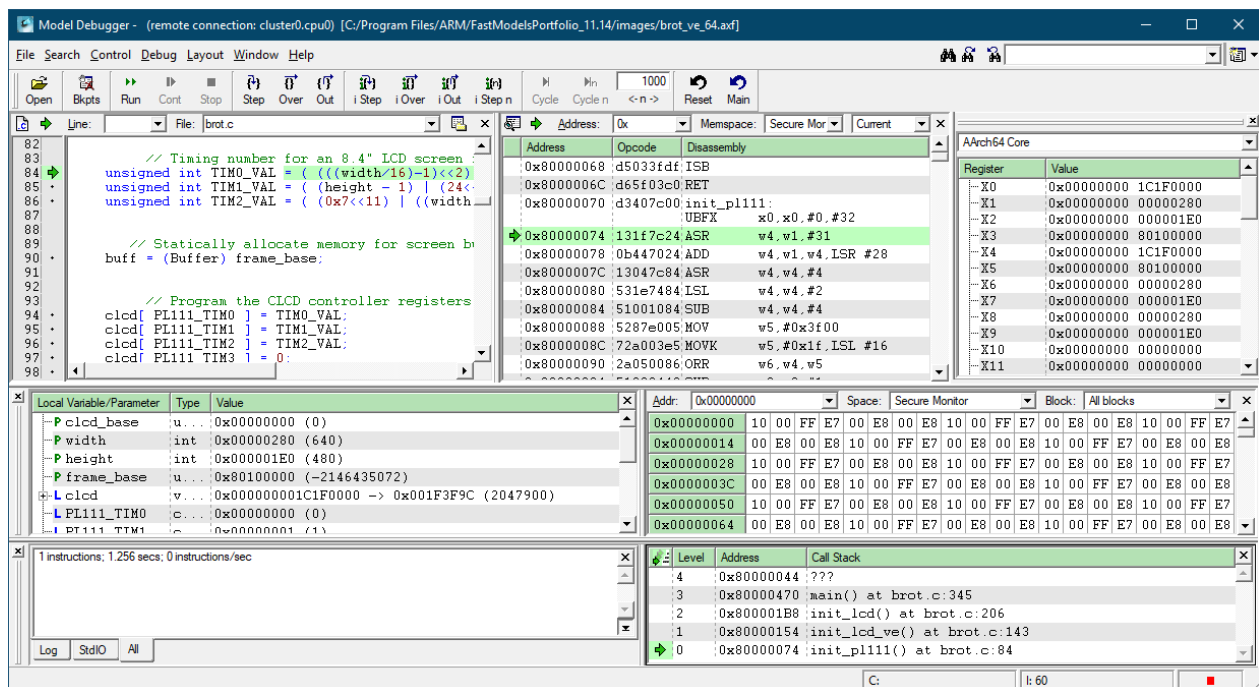
2.1 Key features

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.

Note Fast Models targets do not support all of these features.

2.2 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.

2.3 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 72

3. Installation and configuration

This chapter describes how to install and configure a standalone version of Model Debugger. Model Debugger is automatically installed with Fast Models.

3.1 Linux software requirements

This section describes the software requirements for using Model Debugger on Linux.

Operating system

Red Hat Enterprise Linux 7 or 8 (for 64-bit architectures), Ubuntu 18.04 or 20.04 Long Term Support (LTS).

PDF reader

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

License management utilities

For FlexNet Publisher node-locked or floating licensing, the latest version of the FlexNet software is available for download from License Server Management Software.

For information about user-based licensing, see https://lm.arm.com.

3.2 Linux installation

This section describes how to install Model Debugger.

Unpack the archive and run the setup program:

```
gunzip ModelDebugger_<version>.tgz
tar -xvf ModelDebugger_<version>.tar
cd ModelDebugger_<version>
./setup.bin
```

In the sequence of commands, `<version>` is the version of Model Debugger you are installing.

The installer prompts you for the target installation directory and creates the following subdirectories:

bin

Executables.

doc

Documentation.

etc

Model Debugger setup scripts.

lib
> Libraries and tool-specific files.

3.3 Linux environment configuration scripts

Model Debugger provides setup scripts in the `etc` directory. The appropriate setup script must be executed to configure your environment for Model Debugger.

- For Bourne and related shells, use:

  `. installation_directory/etc/setup.sh`

- For C and related shells, use:

  `source installation_directory/etc/setup.csh`

You might find it more convenient to add a reference to the Model Debugger configuration script to your usual startup script.

The setup script sets the following environment variables:

Table 3-1: Environment variables

Variable	Description
`$PATH`	The `PATH` environment variable is updated to include `installation_directory/bin`.
`$MAXVIEW_HOME`	Set to the Model Debugger installation directory. Model Debugger was previously named MaxView.
`$ARMLMD_LICENSE_FILE`	For FlexNet Publisher node-locked or floating licensing, set to the location of the license file or license server (using `port@host` syntax) for Model Debugger. See License Management for more information. If necessary, you can change this environment variable after installation by editing the `setup.[c]sh` script.

3.4 Windows software requirements

This section describes the software requirements for using Model Debugger on Windows.

Operating system
> Microsoft Windows 10 64-bit.

PDF reader
> Adobe Acrobat Reader.

License management utilities
> For FlexNet Publisher node-locked or floating licensing, the latest version of the FlexNet software is available for download from License Server Management Software.
>
> For information about user-based licensing, see https://lm.arm.com.

3.5 Windows installation

This section describes how to install Model Debugger.

Procedure

1. Open the distribution archive `ModelDebugger_version.zip` and extract the complete contents into a temporary directory. `version` is the version of Model Debugger you are installing.

2. To start the installer, run the `Setup.exe` program in the temporary directory.

3. When prompted by the installer, specify the target directory for the installation or accept the default directory.
   The installer creates subdirectories in the specified installation directory.

   bin

   Executables.

   doc

   Documentation.

> **Note**
>
> The installer configures the environment variables for the user who installed the tools. Other users might need to use Model Debugger on the same computer. Either copy the value of the `%MAXVIEW_HOME%` environment variable to the System variables, or have them define the environment variable themselves in Control Panel. Both operations need administrator privileges.

---

4. Using Model Debugger

This chapter describes how to use Model Debugger.

4.1 Launching Model Debugger

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

Related information

4.1.1 Launching from the command line

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

Table 4-1: Command-line options

Short	Long option	Description
	`--cyclelimit cycles`	Set a limit on the number of system cycles for a simulation in non-GUI mode. Use the `--nogui` option to enable this option.
	`--debug-isim isim_system`	Start `isim_system` and connect Model Debugger to the simulation.
	`--debug-sysc systemc`	Start `systemc` simulation and connect Model Debugger to the simulation.
-T	`--timelimit time`	Set a time limit for a simulation in non-GUI mode. Use the `--nogui` option to enable this option.
-a	`--application filename`	Load the application file `filename`. To target a specific core in cluster systems, give the path to the core instance. For example, `foo.bar.core=dhrystone.axf`. **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 `-c` option.
-C	`--parameter parameter`	Set one model parameter. Specify it as a path naming the instance and the parameter name using dot separators. For example, `foo.bar.inst.parameter=1000`. To set multiple parameters, use `--config-file`.
-c	`--connect simulation_id`	Connect to a remote CADI simulation. `simulation_id` specifies the simulation to connect to. `--list-connections` displays the list of available connections.
-E	`--enable-verbose msgClass`	Use verbose messages if displaying message text for message classes `msgClass`. Without an argument, this option lists all classes.

Short	Long option	Description
-e	--env-connect	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
-F	--stdout-to-file *FILE*	Print all application output to `FILE` instead of the StdIO pane in the Output Window.
-f	--config-file *filename*	Use model parameters from the configuration file `filename`.
-h	--help	Print the available options and exit.
-i	--instance	Specify the instance.
-L	--cadi-log	Log all CADI calls into an XML logfile `CADILog-nnnn.xml`, where *nnnn* is a set of four digits. The logfile is created in the same directory as the model.
	--list-connections	List 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 *filename*	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 *filename*	Load the project file `filename`.
-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 *filename*	Load the layout file `filename`.
-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** check box in the Preferences dialog box to suppress the Select Target dialog box.

4.1.1.1 String syntax

Filenames and similar strings that are specified when starting Model Debugger from the command line must be within double quotes if there is white space in the string.

For example:

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

There is, however, 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"
```

4.1.1.2 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 17

4.1.1.3 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.

4.1.2 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
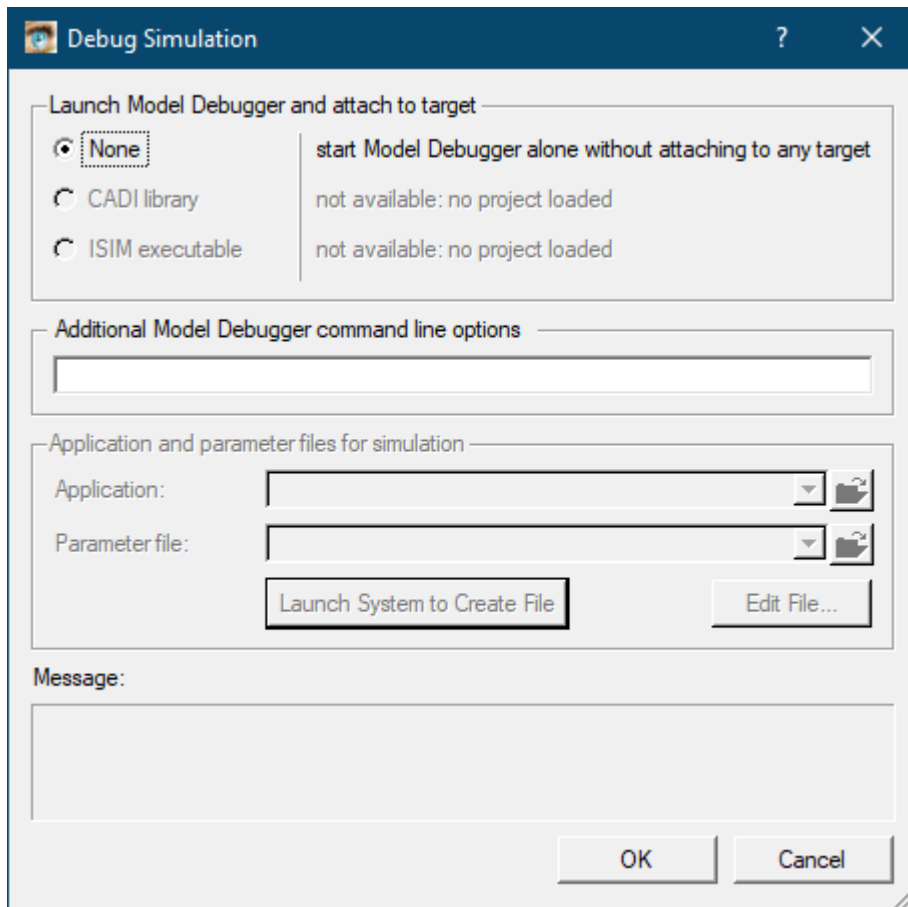
2. Click **OK**.
   Model Debugger starts.

4.1.2.1 Configuring model parameters

This section describes how to configure models.

If you had not yet loaded a model at the time that 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
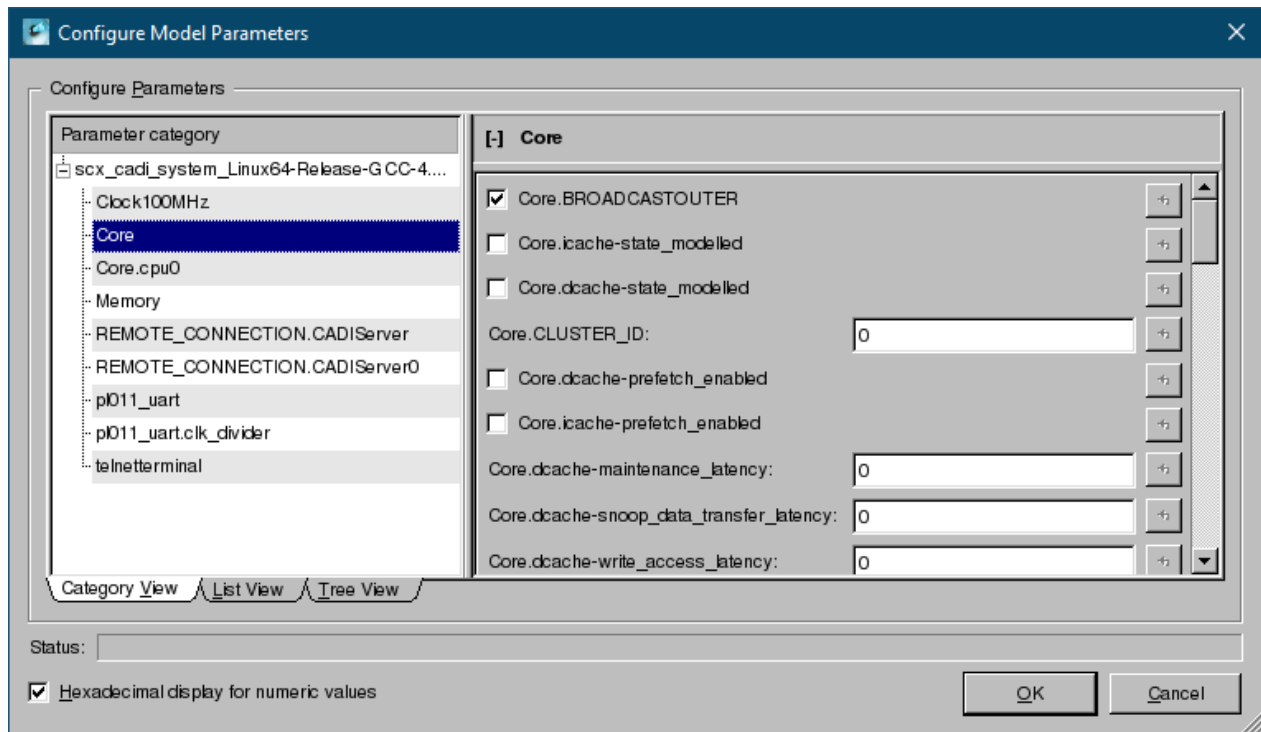
If you had already loaded a model before Model Debugger starts, Model Debugger checks the available components and opens a similar dialog box.

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 the view, click the **+** symbol. To collapse the view, click the **-** 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.

> **Note**
>
> 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
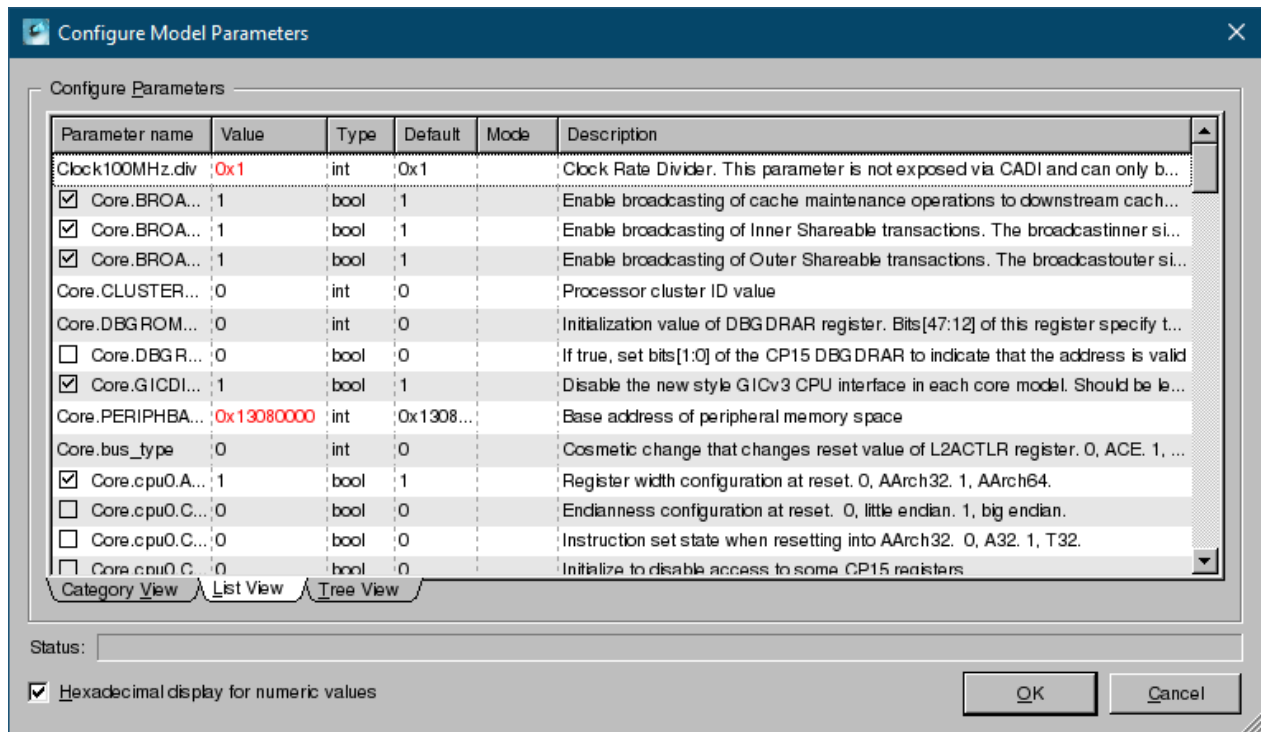
Set the parameters for the model and click **OK** to close the Configure Model Parameters dialog box.

4.1.2.2 Selecting target components

After closing the Configure Model Parameters dialog box, the Select Targets dialog box appears. Select those components that can be debugged in the model.

Figure 4-4: Select Targets dialog box
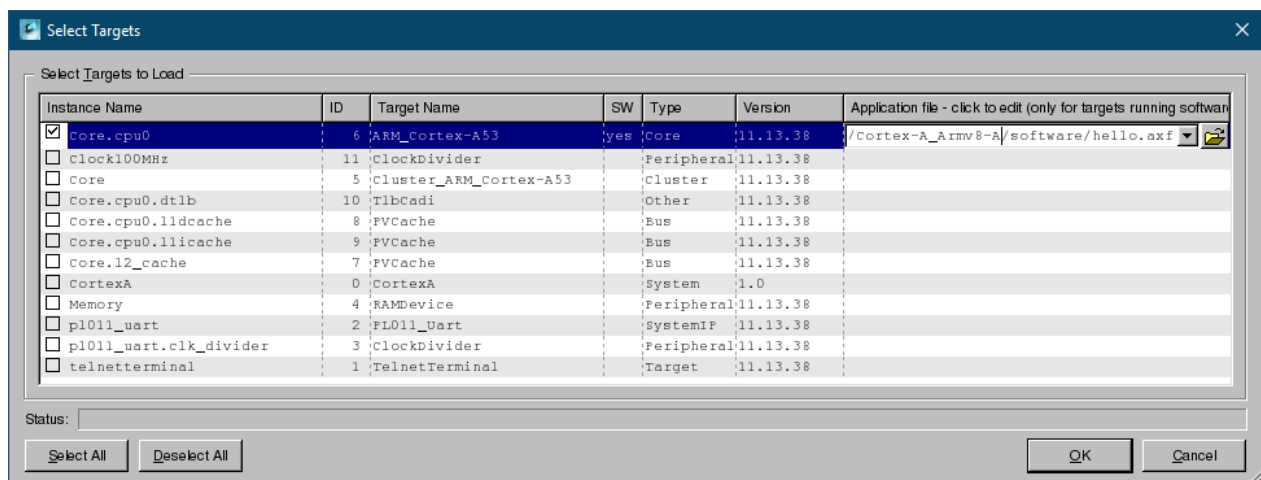
1. Click the box next to the components that are to load.

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.

4.1.2.3 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
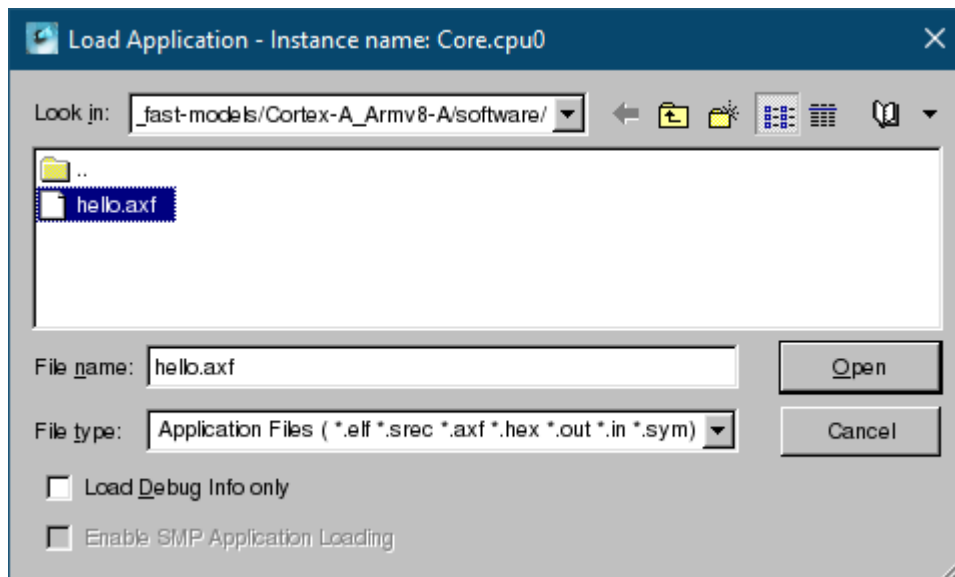
Reset the component after loading the source.

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

Note

4.2 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
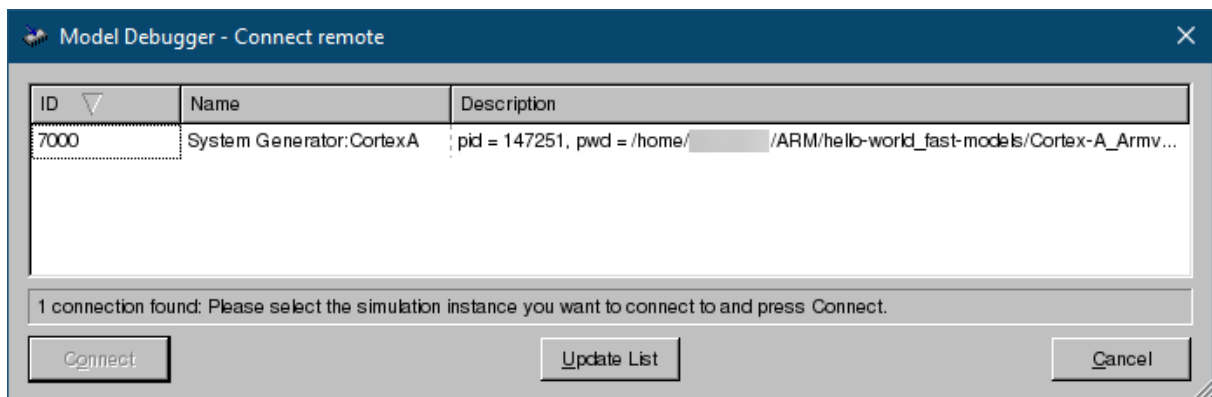
4. Select the required simulation instance and click **Connect**.
5. Select a target, for example the processor, and click **Connect** to connect to the specific component instance in the simulation.
6. If no application loads, select **File** > **Load Application Code**. Select the application image from the Load Application dialog box and click **Open**.

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

4.3 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
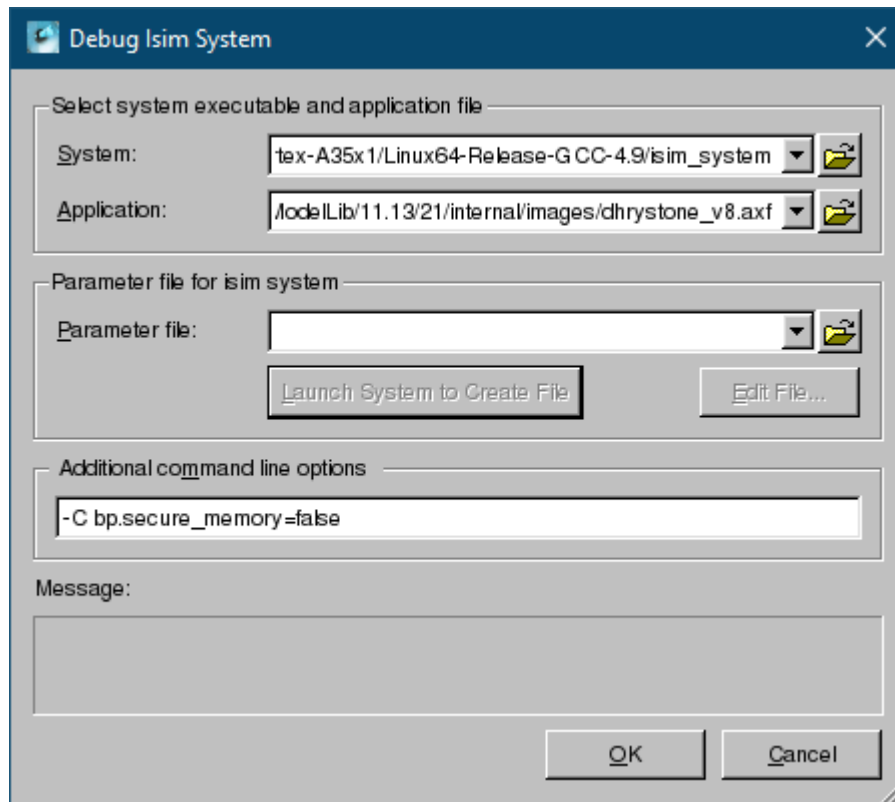
2. Select a simulation and optionally an application (and parameter file for an ISIM only). If a file is missing, an error message appears. Click **OK** to start the simulation and connect.
   The **Select Targets** dialog box appears.

Related information

4.4 Connect Model Debugger to a model running on another machine

You can connect Model Debugger to a Fast Model that is running on a different 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 the CADI server listens on.

> **-C REMOTE_CONNECTION.CADIServer.port=<*port*>**
>
> Specify the port the CADI server listens 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 and an ISIM running on separate hosts

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
```

4.5 Model Debugger application windows

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

Related information
Preferences dialog box on page 73
Saving the window layout on page 39

4.5.1 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.

Figure 4-8: Default layout for Model Debugger
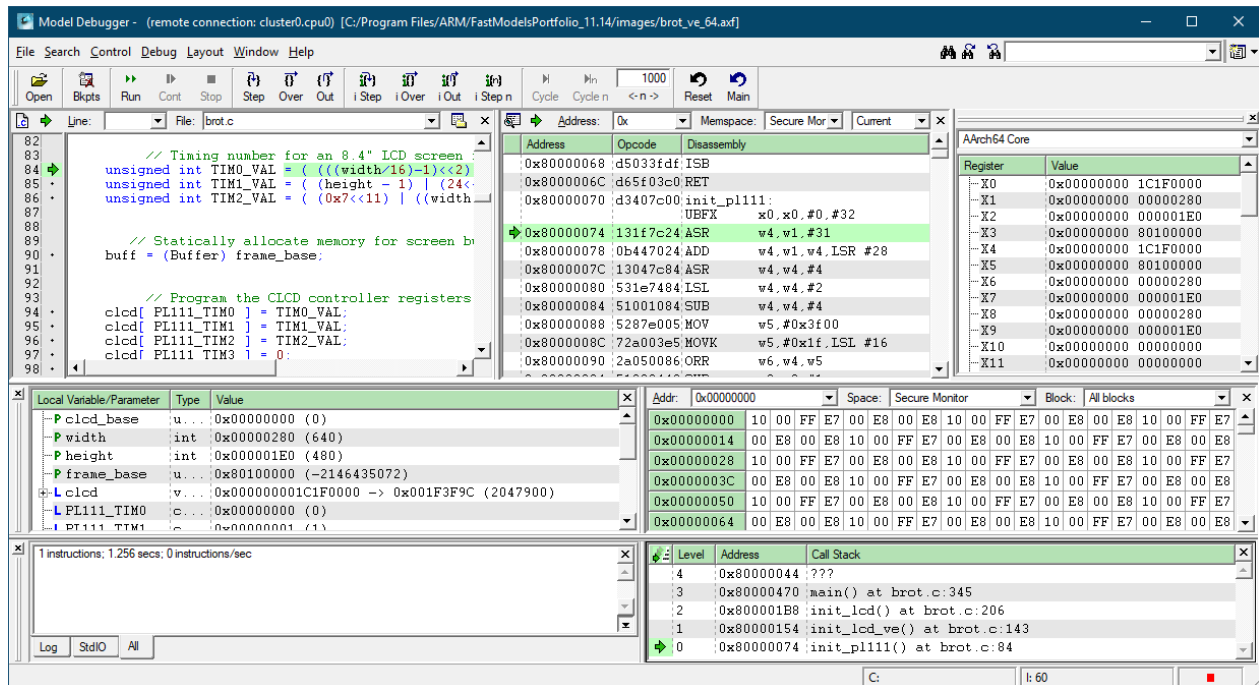
4.5.2 Main toolbar

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 4-9: Main toolbar
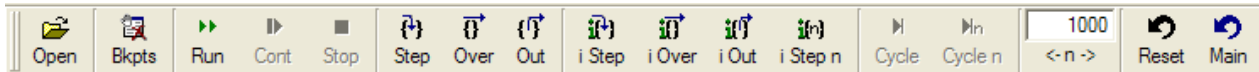
Open

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.

> **Note**
>
> 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.

> **Note**
>
> Not all model targets support this command.

Back

Click to step the simulation backwards by one cycle.

> **Note**
>
> Not all model targets support this command.

Reset

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

> **Note**
>
> 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.

> **Note**
>
> - 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

4.5.3 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:

> **Note**
> 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.

> **Note**
>
> 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.

> **Note**
>
> Not all model targets support this command.

Instruction Step Out

Advance the simulation until a return instruction is executed.

> **Note**
>
> 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.
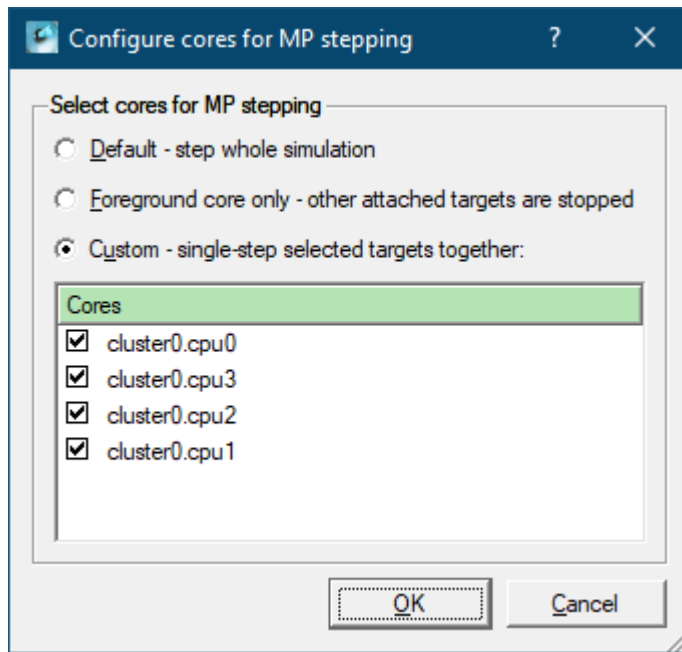
Back n

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 4-10: 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.

> **Note**
>
> 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.

> **Note**
>
> 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.

> **Note**
>
> Fast Models does not use the profiling options.

View Profiling ...

Display the Profile Information dialog box.

> **Note**
>
> 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.

4.5.4 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.

4.5.5 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 4-11: Moving a window
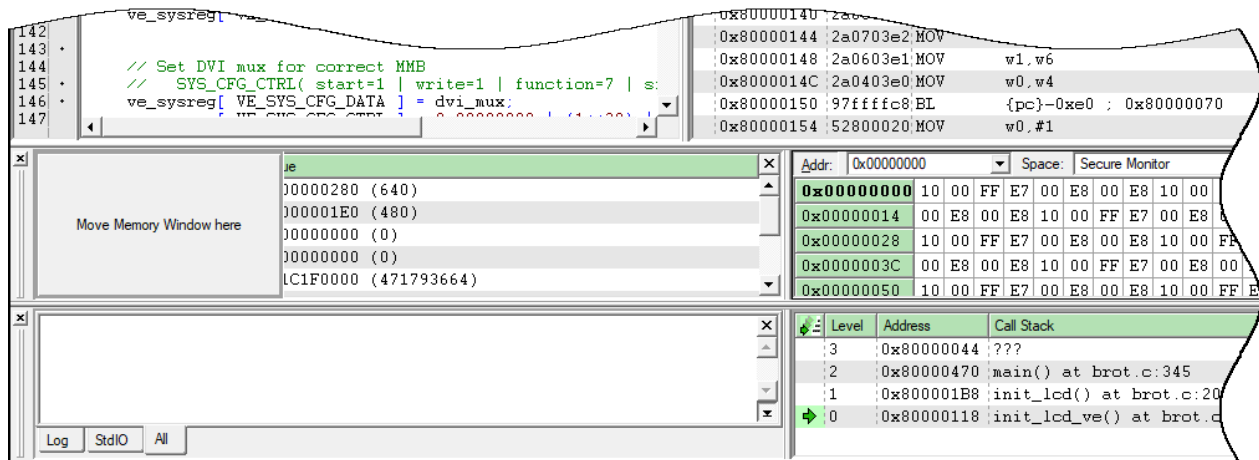
To copy a window, press **Ctrl**+**F9**. This action effectively duplicates the existing view. A gray box near the center of the Model Debugger window indicates the location for the duplicate view for the Local Variables window. Releasing the mouse button creates the duplicate window in the target location.

Figure 4-12: Duplicating a window
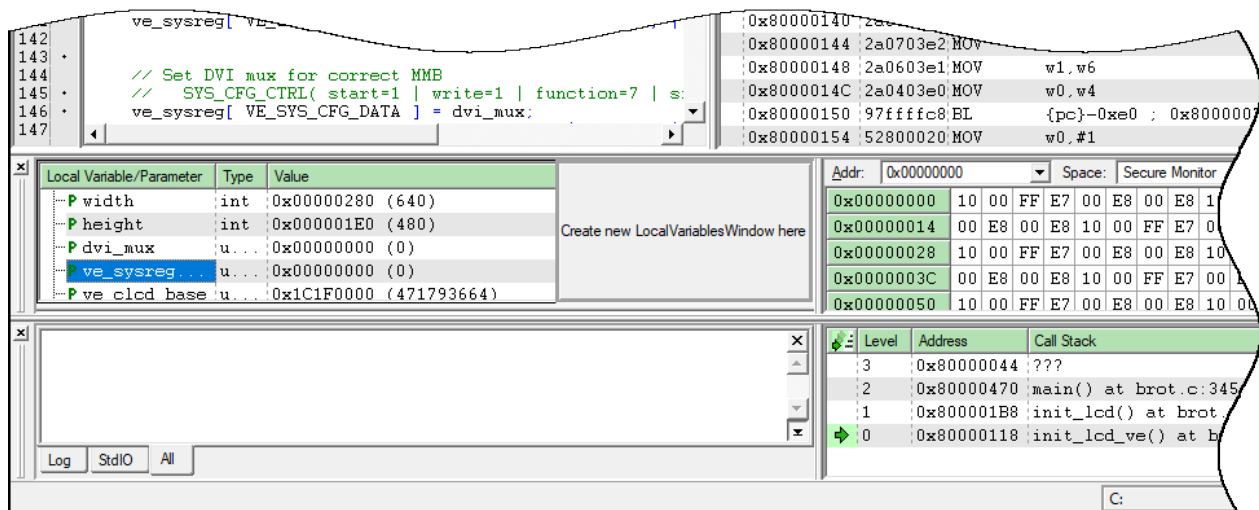
> **Note**
>
> The windows might be difficult to place into the required position. To force the window to dock to a particular location, select the window handle and right-click to display the context menu. The options are: **Dock Bottom**, **Dock Left**, **Dock Right**, and **Dock Top**. It might take several moves to force the window to the required location.
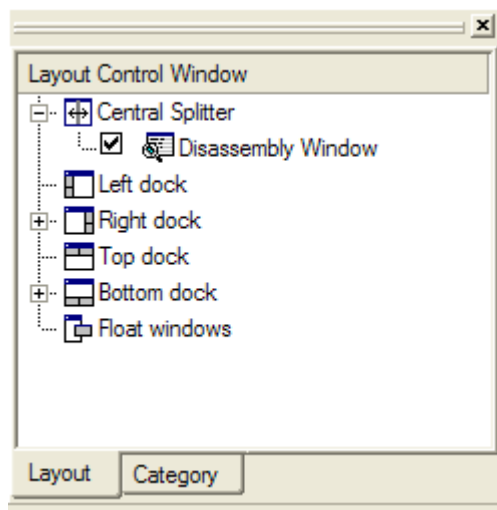
4.5.6 Saving the window layout

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

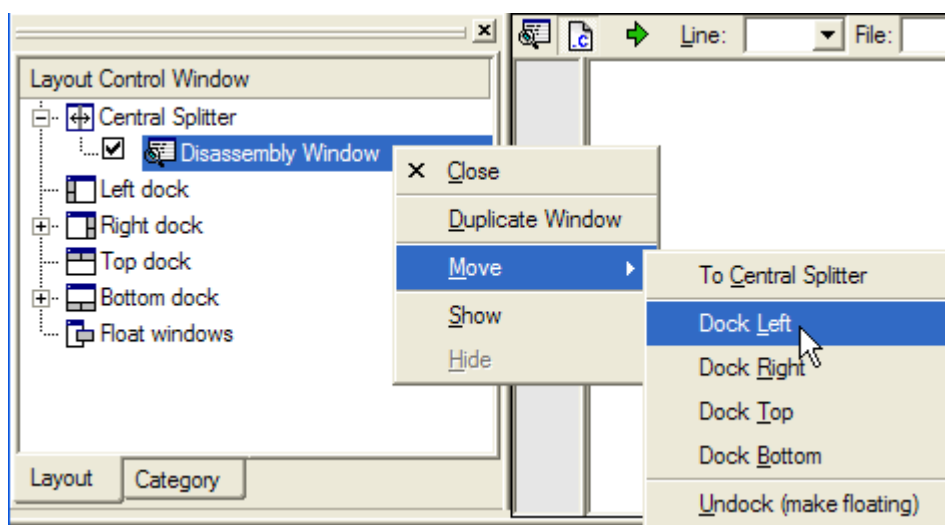The **Layout** menu has the following entries:

Layout Control Window

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

Figure 4-13: Layout Control window

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

Figure 4-14: Layout Control context menu

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

Load Layout

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

Figure 4-15: Load Layout dialog box
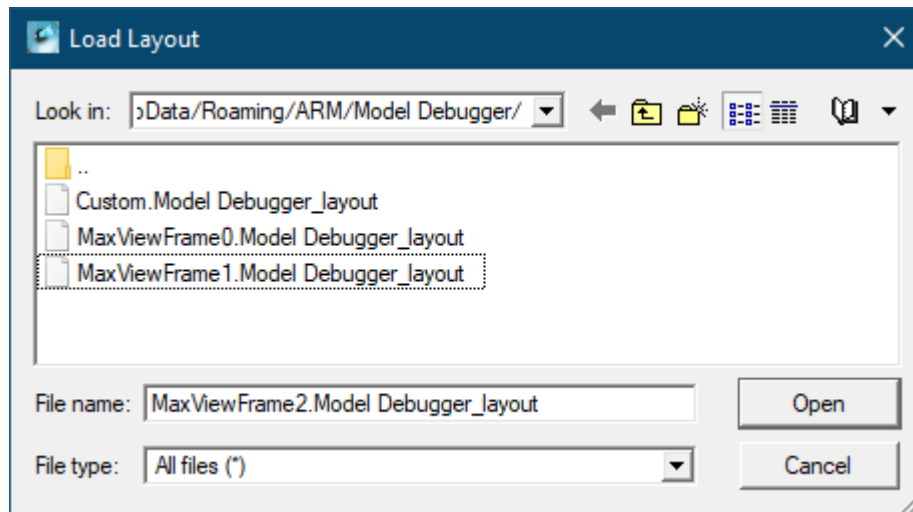
Save Layout

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.

4.5.7 Opening new debug views

This section describes how to open new debug views.

About this task

To open a new window:

- Select **New View** from the **Window** menu and selecting the required type of debug view.

- Click the **View** icon at the right of the menu bar and select a view from the list.

Figure 4-16: Icons for selecting a new debug view
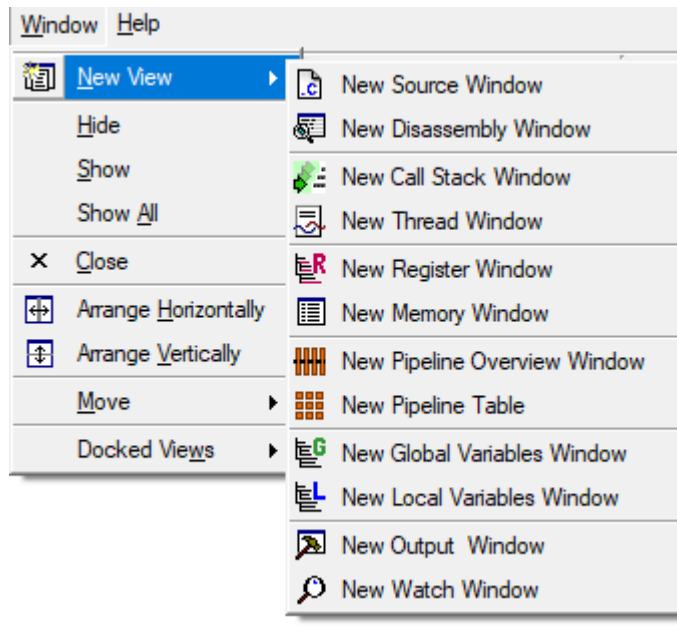
4.5.8 Closing windows and views

This section describes how to close windows and views.

About this task
You can close a dock window, and all views in the window, by clicking 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.

4.5.9 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.

4.6 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.

4.6.1 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 4-17: 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.

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

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 4-18: Source view
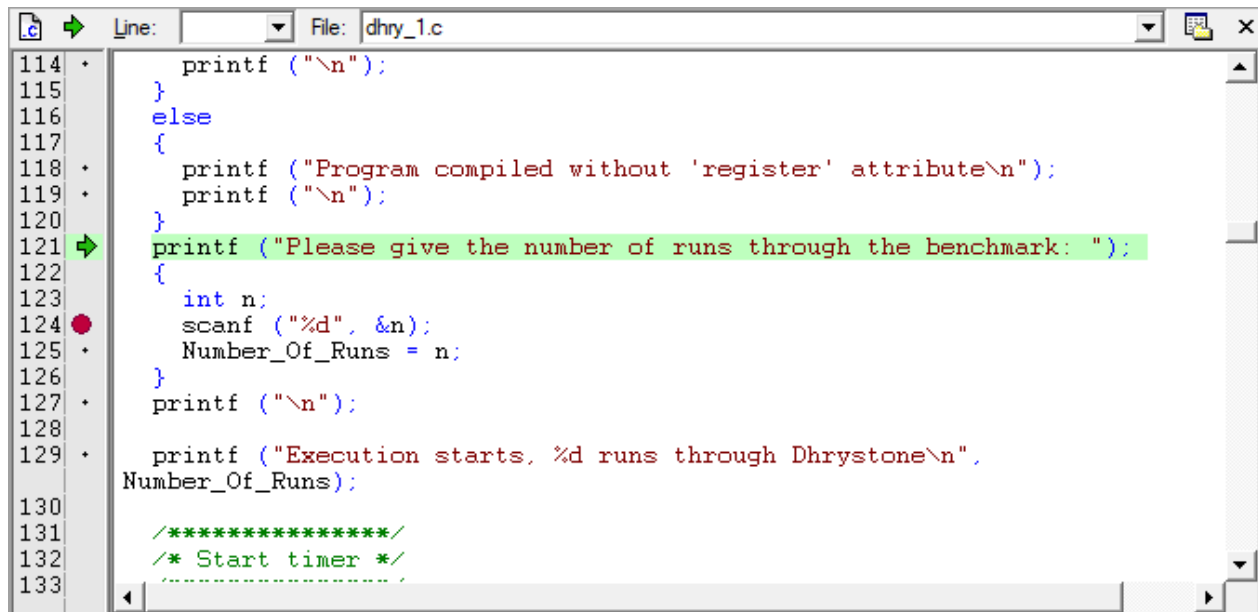
The **Source view** title bar has controls for:

- 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

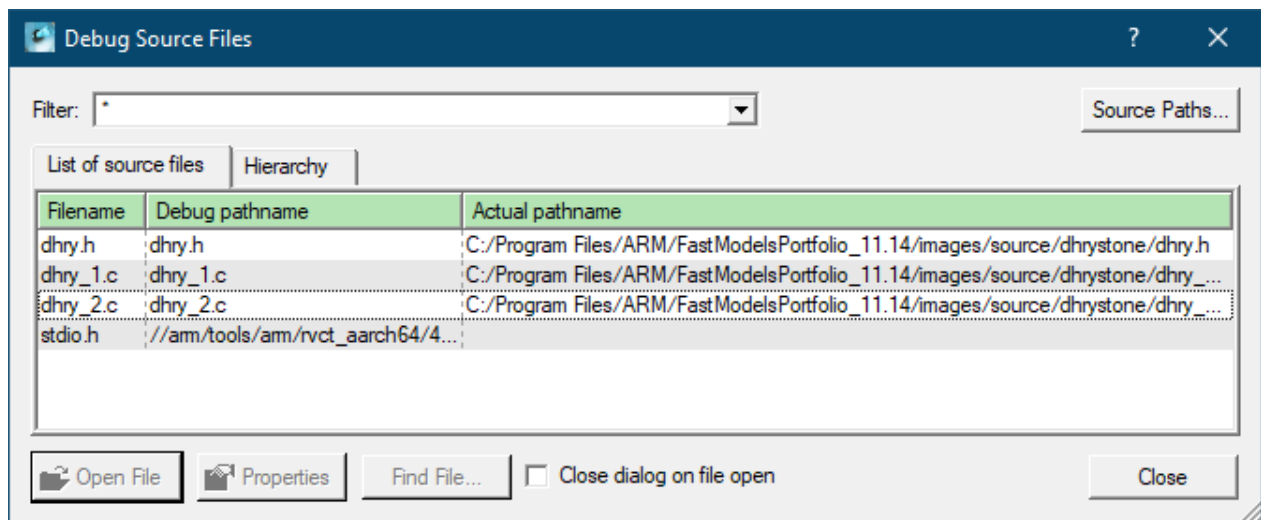Display the filename and path for the file.

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**.

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

Figure 4-19: Debug Source Files dialog box

Filename	Debug pathname	Actual pathname
dhry.h	dhry.h	C:/Program Files/ARM/FastModelsPortfolio_11.14/images/source/dhrystone/dhry.h
dhry_1.c	dhry_1.c	C:/Program Files/ARM/FastModelsPortfolio_11.14/images/source/dhrystone/dhry_...
dhry_2.c	dhry_2.c	C:/Program Files/ARM/FastModelsPortfolio_11.14/images/source/dhrystone/dhry_...
stdio.h	//arm/tools/arm/rvct_aarch64/4...	

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 4-20: Find Source File dialog box
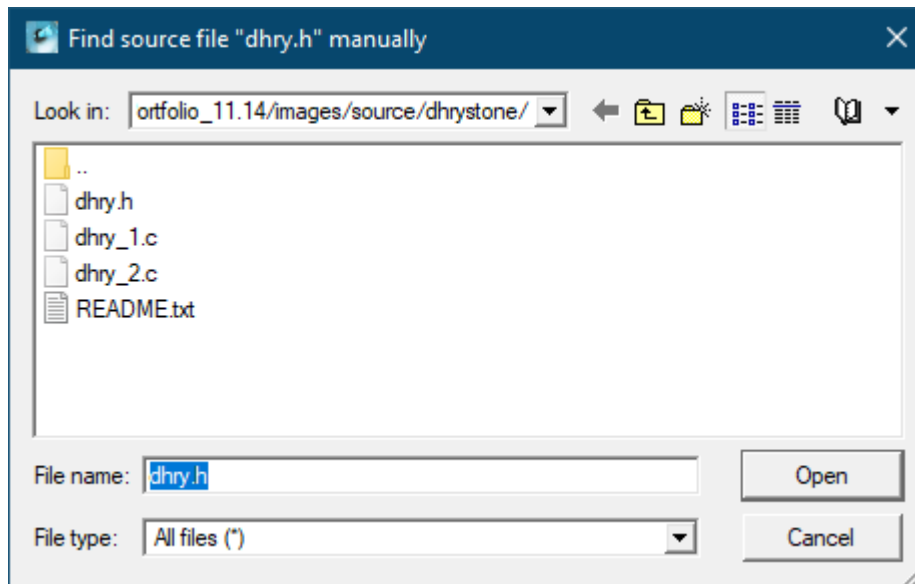
Click **Properties** to display the **File Properties** dialog box for the selected file. You can also use the **Find File** button in the **File Properties** dialog box to locate the file.

Figure 4-21: Source File Properties dialog box
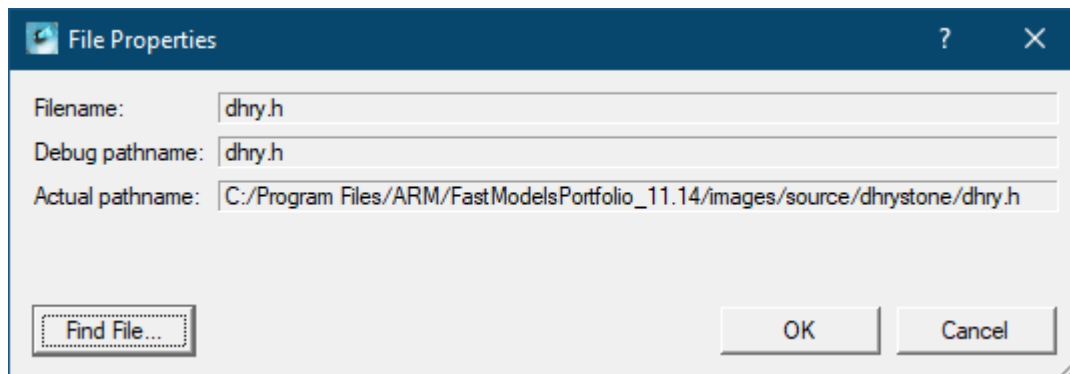
Model Debugger has an automatic mechanism to add replacement paths that are invoked every time you are prompted to find a source file. If the source file is found, an automatic source path replacement is calculated.

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.

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

Figure 4-22: Source Path Replacement dialog box
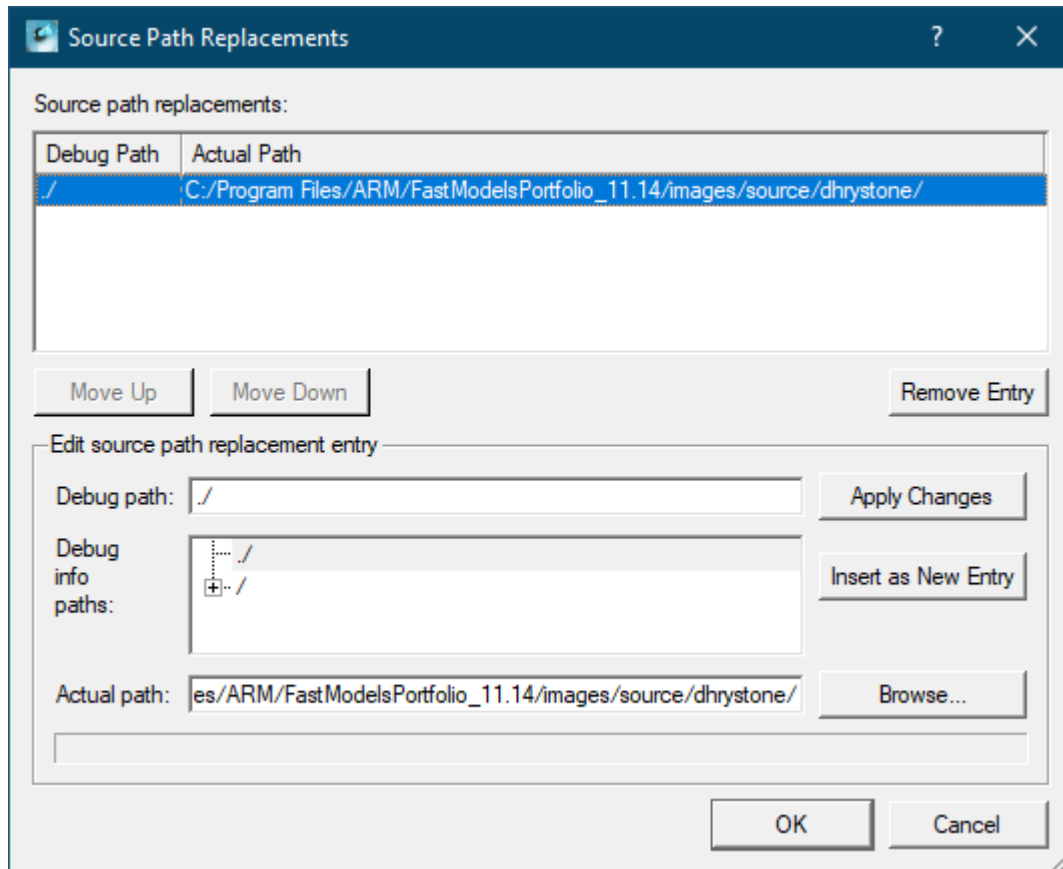
Existing source file replacements are displayed in the top part of the **Source Path Replacement** dialog box. You can remove or reorder paths by highlighting an entry and clicking one of the following buttons:

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.

4.6.2 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 4-23: Disassembly view

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.

Note

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

Figure 4-24: Matching source and disassembly
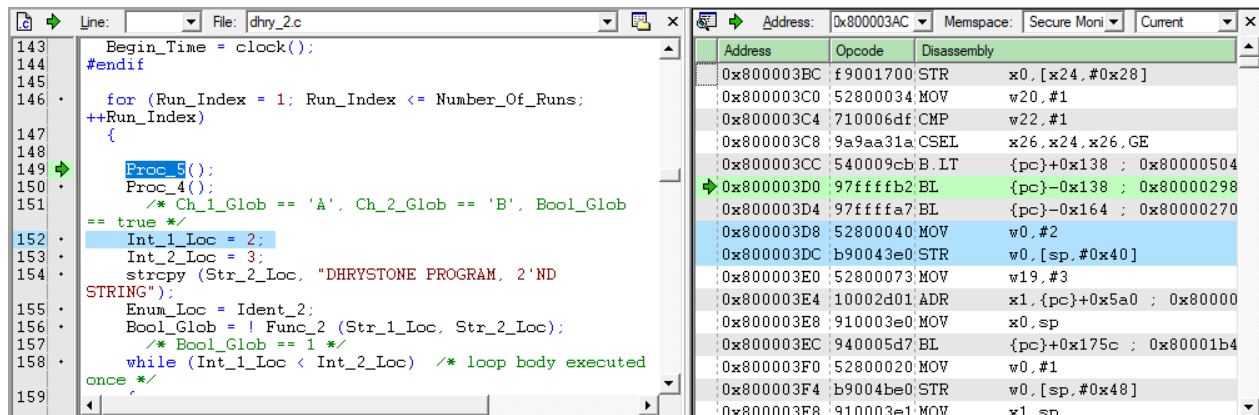
Context menu for Disassembly view

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.

4.6.3 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 4-25: 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.

4.7 Debug views for registers and memory

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

4.7.1 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 4-26: Select register group
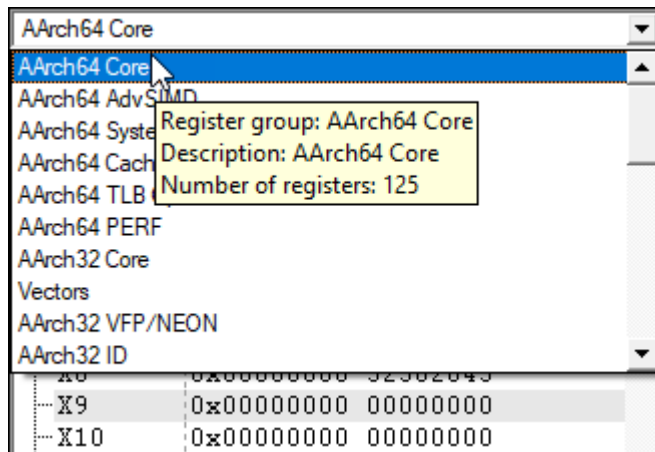
For each register, a buffered state of the register (previous value) is stored. To view the contents, either:

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

    Figure 4-27: Register view showing current and previous contents

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

    Figure 4-28: Register view contents at cursor

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.

4.7.2 How to use 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 4-29: Update the cache view register values

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 4-30: Change the cache selection method

4.7.3 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 4-31: Memory view

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 4-32: Load File to Memory dialog box
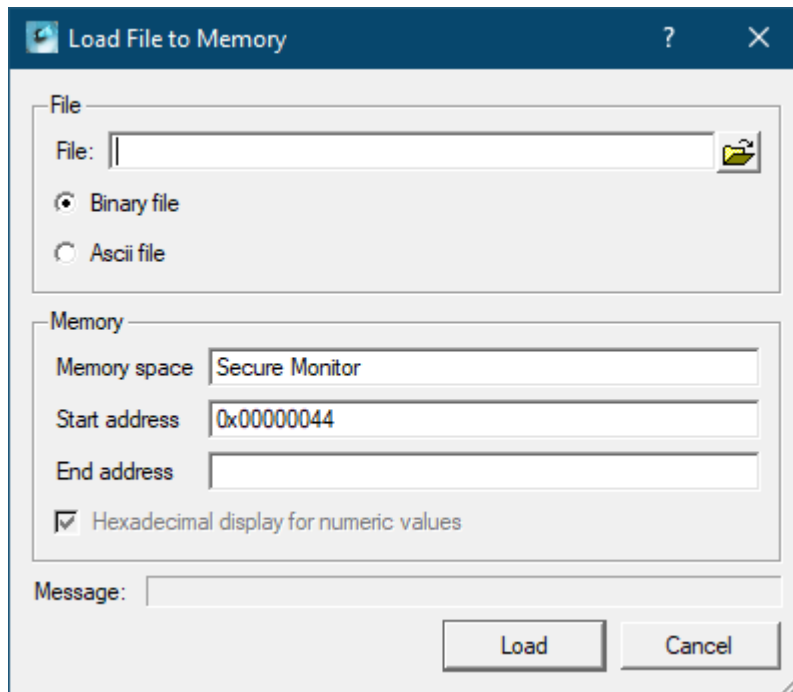
Enter the file name into the top field of the dialog box. You can use the button next to it to browse for the file. When loading or storing a binary or ASCII file, select the correct button. The Memory Space and Start Address fields are filled automatically from the memory view where you opened the dialog 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.

4.7.4 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 4-33: Local Variable view

Global Variables window

This window shows the global variables with their types and values. A green letter G marks them.

Figure 4-34: Global Variable view

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.

4.8 Debug views for pipelines

Model Debugger provides options for viewing the pipeline.

The options are:

- The Pipeline Overview window.
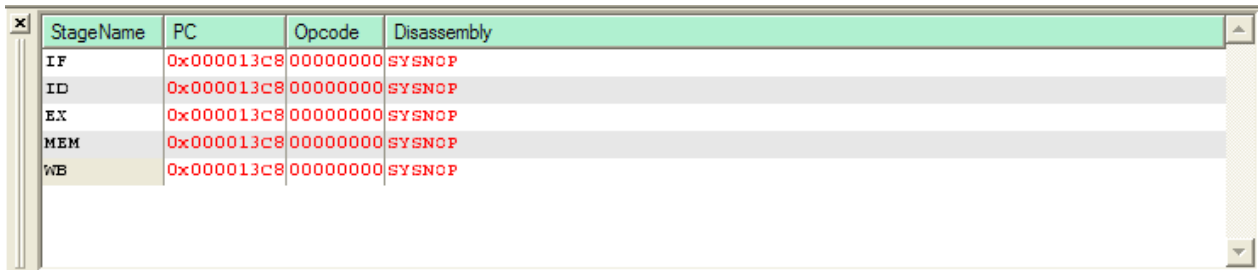
- The Pipeline Table.

Pipeline views are only available if your model supports them. If the pipeline icons are gray, not orange, then you cannot view pipeline information.

4.8.1 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.
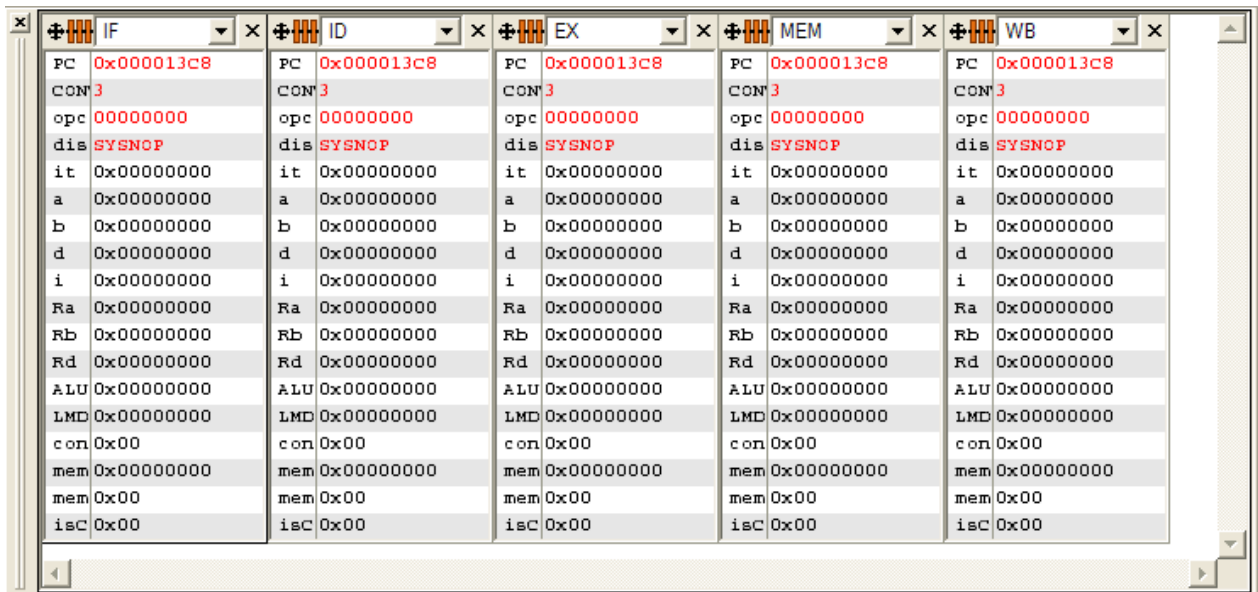
Figure 4-35: Pipeline Overview window

4.8.2 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 4-36: Pipeline Table window

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 4-37: Pipeline Table icons
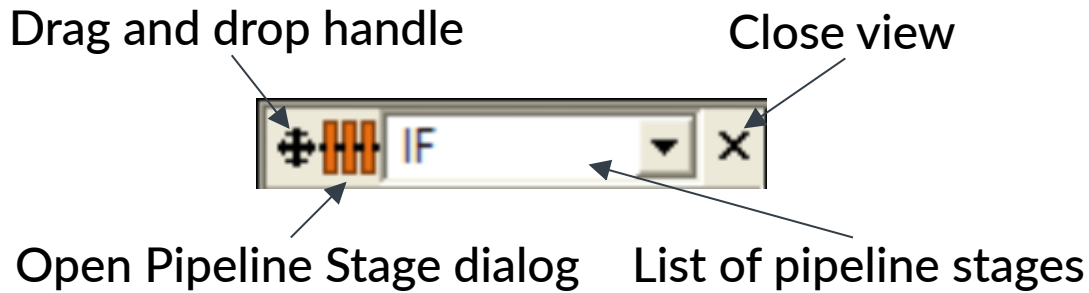
Double click the first column of a view to set or remove a breakpoint on the field.

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 4-38: Pipeline Table context menu
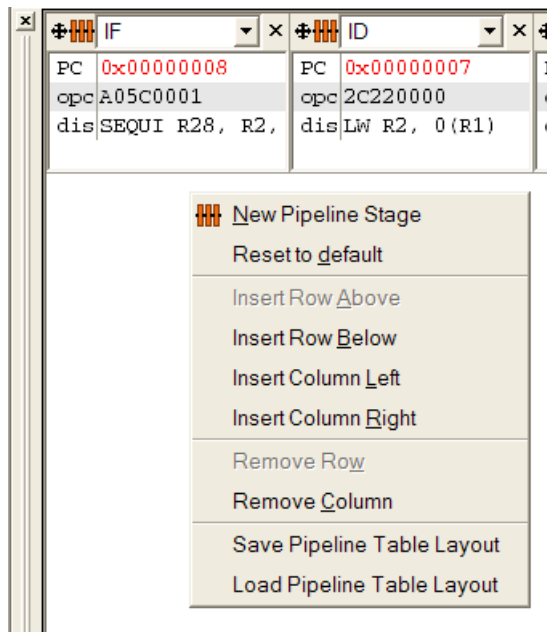
The context menu has the following entries:

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 4-39: Pipeline Stage Properties dialog box
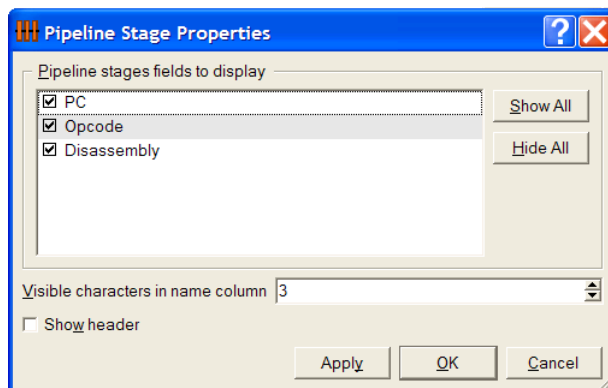
Choose the fields to be displayed in the pipeline stage list by checking the boxes or clicking the **Show All** or **Hide All** buttons.
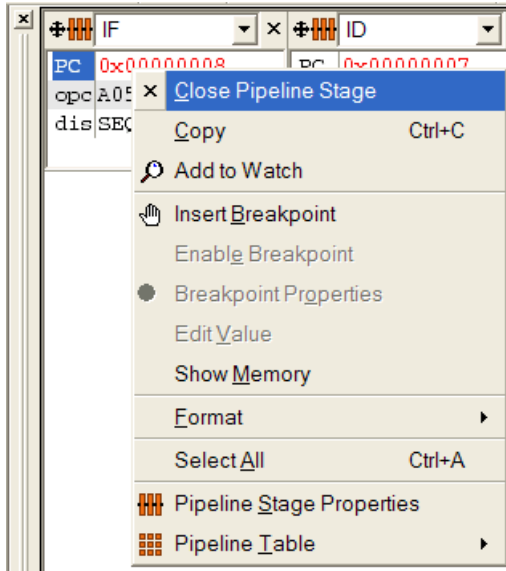
The size of the first column can be set in the **Visible characters in the name column** control.

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 4-40: 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 4-41: Submenu for display format
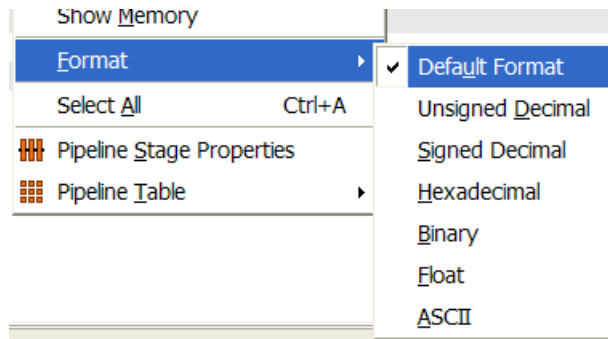
Select All

　　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 71

4.9 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 4-42: Watch window

There are two types of entry in the **Watch window**:

System variables

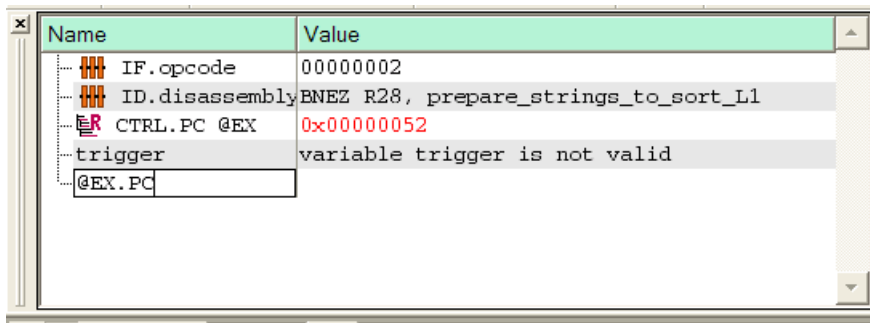Entries in this group are marked with small icons to the left of their name to indicate their origin. They can be manipulated in the **Watch window** in the same way as in their original view. Items in this category include:

- 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

4.9.1 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.

4.10 Breakpoints in Model Debugger

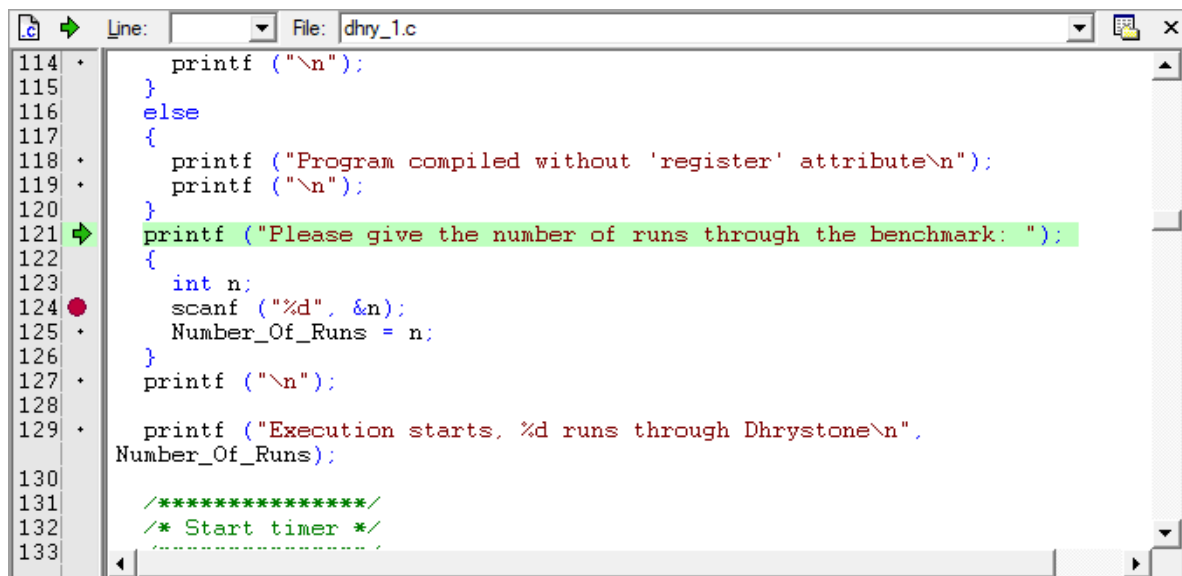This section describes how to work with breakpoints in Model Debugger.

4.10.1 Setting breakpoints in the debug views

This section describes how to set different kinds of breakpoint in the debug views.
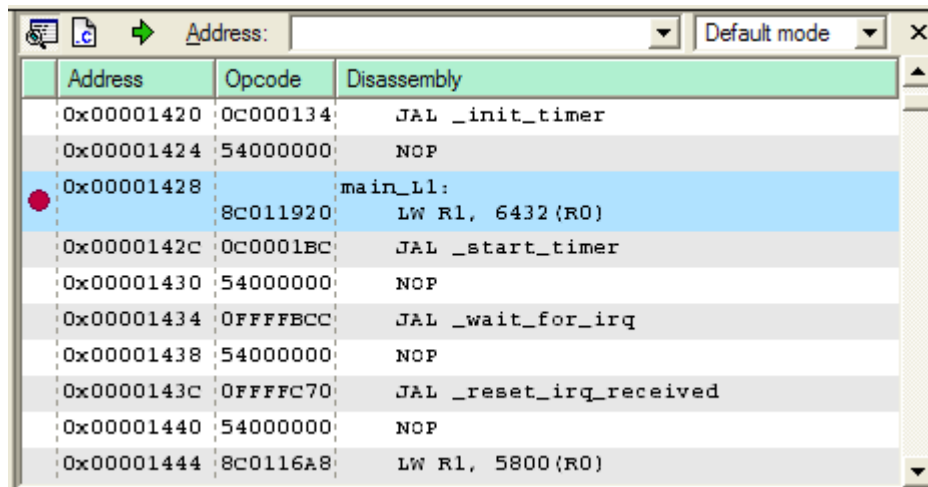
Source code view
>   The second column contains small bullets for each source line. To set a breakpoint, double click on a bullet.

Figure 4-43: Source view breakpoint
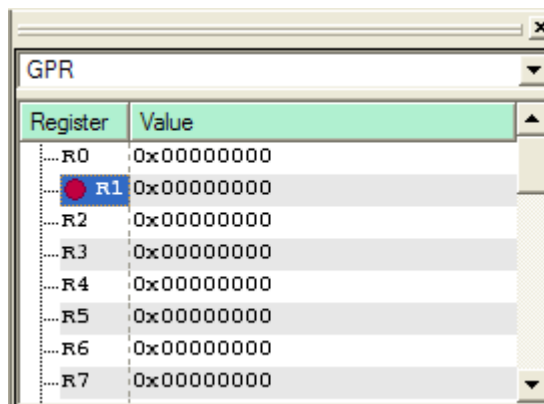

Disassembly view
>   To set a breakpoint on a line, double click on any column.

Figure 4-44: Disassembly view breakpoint

Register view

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

Figure 4-45: 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.

> **Note**
>
> 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 4-46: Pipeline table breakpoint

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.

4.10.2 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 71

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.

4.10.4 Breakpoint Manager dialog box

Control and maintain all breakpoints through the Breakpoint Manager.

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
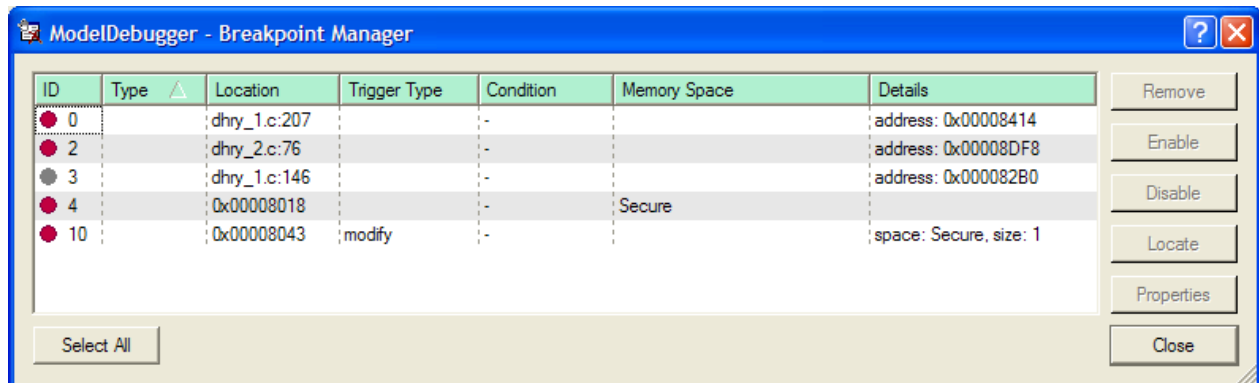
4.10.5 Breakpoint Properties dialog box

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.
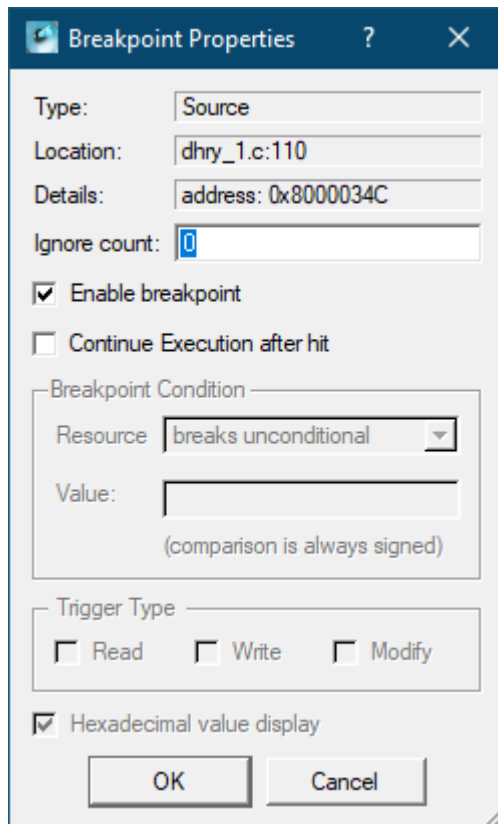
Trigger Type

> 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

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`.

Note

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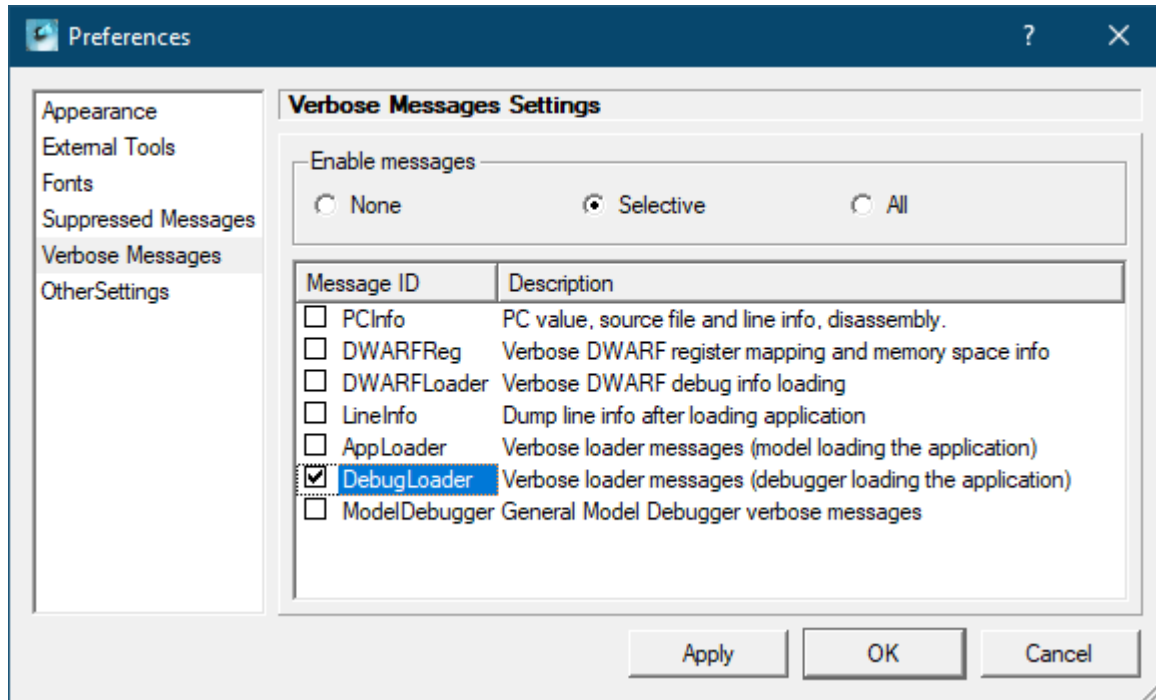
To save a session, select **Save Session**, or **Save Session As**, from the **File** menu.

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

4.12 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.

Note 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.

5. Shortcuts

This chapter describes shortcuts available in Model Debugger.

5.1 Keyboard shortcuts

This section describes the keyboard shortcuts.

Table 5-1: 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 `main()`.
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.

Modifier	Key	Function
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.
Ctrl	T	Select target.
Ctrl	U	User preferences dialog.
Ctrl	W	Close model.