SoC Designer Plus
User Guide

Copyright © 2016 ARM Limited. All rights reserved.

Release Information
The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2016</td>
<td>A</td>
<td>Non-Confidential</td>
<td>Update for 8.3</td>
</tr>
<tr>
<td>March 2016</td>
<td>B</td>
<td>Non-Confidential</td>
<td>Update for 8.4</td>
</tr>
</tbody>
</table>

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice 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 implementations infringe any patents.

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, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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 shall prevail.

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.

This document consists solely of commercial items. You shall be responsible for ensuring that any 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.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with * or ° are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the ARM trademark usage guidelines http://www.arm.com/about/trademarks/guidelines/index.php.

Copyright © ARM Limited or its affiliates. All rights reserved.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

In this document, where the term ARM is used to refer to the company it means “ARM or any of its subsidiaries as appropriate.”
Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com
Contents

Preface

Audience ................................................. 13
Conventions ............................................. 13
Terminology ............................................. 14
Further reading ....................................... 14

Chapter 1. Introduction

Introduction to SoC Designer Plus ......................... 15
ESLAPI overview ....................................... 16
CASI ...................................................... 16
CADI ...................................................... 17
CAPI ...................................................... 17
CASIMMI ............................................... 17
SoC Designer Canvas ................................... 18
Canvas Window layout ..................................... 18
Creating a System using Canvas ......................... 21
SoC Designer Simulator ................................ 21
Simulation Features ....................................... 21
Running a Simulation ..................................... 22
Simulator Window Layout ................................ 22
The runtime windows ..................................... 24
  Switching between the diagram and runtime windows .......... 24
  Saving/restoring windows between simulation sessions .......... 25

Chapter 2. SoC Designer Canvas Reference

Starting SoC Designer ..................................... 27
SoC Designer Canvas Application Interface Reference .... 29
  System menu and controls ................................ 30
  Projects tab ............................................ 30
  Title bar ............................................... 30
  Main menu and Toolbar .................................. 31
  The File menu .......................................... 32
  The Edit menu .......................................... 32
  The View menu .......................................... 33
  The Insert menu ........................................ 35
  The Object menu ........................................ 35
  The Tools menu ......................................... 36
  The Simulate menu ...................................... 37
  The Window menu ....................................... 38
  The Help menu ......................................... 38
  Status bar .............................................. 39
Chapter 3. Creating a SoC Designer Component

Component Overview .................................................. 65
Preparing a SoC Designer Component to be Generated .......... 67
  Step 1, Project name and type ..................................... 67
  Step 2, General properties ......................................... 67
  Step 3, Component Ports ........................................... 69
    Importing a port .................................................. 70
  Step 4, Adding Parameters ........................................ 70
    Importing parameters ............................................ 71
    AHB or APB parameters ......................................... 71
  Step 5, CADI debug interface ..................................... 72
Changing the default registers ............................................. 72
Adding new debug registers .................................................. 72
Generating a debug interface for memory .............................. 73
Generating the component .................................................. 73
Modifying the generated files .............................................. 75
General Modifications ....................................................... 75
Slave Port modifications .................................................... 75
CADI modifications .......................................................... 75
CAPI modifications .......................................................... 76
Save/Restore modifications ................................................. 76
Developing components manually ...................................... 76
Defining a factory class for the system ............................... 76
Saving and restoring simulation state .................................. 77
Creating a shared library .................................................... 78
The entry point ............................................................... 78
Creating a project file with the Component Wizard .............. 79
Editing an existing makefile ............................................... 80
Coding and compilation guidelines ..................................... 80
Using your new component in SoC Designer ....................... 81

Chapter 4. Working With Components

Component Description ..................................................... 85
Adding a component to the Diagram window ....................... 86
Displaying component properties using the component info box .................................................. 87
Component context menu ................................................... 88
Port Types, Movement, and Connection ............................. 89
Connecting clock slave ports ............................................. 90
  Connection to master clock [default] ................................. 90
  Connection to no clock .................................................. 91
  Explicit Clock Connection ............................................. 91
Editing memory address spaces for bus-master ports ........... 91
Making connections between components ........................... 91
Working With External ports ............................................. 92
  Adding an external port to a system ................................. 93
Using SystemC components .............................................. 95
Using components based on the ESL APIs .......................... 95
Optimizing components by profiling simulation performance .................................................. 95

Chapter 5. Working With Parameters

Creating new parameters ................................................... 97
Editing Parameters ......................................................... 98
Chapter 6. Working With Memory Maps

Memory Map Overview ....................................................... 101
Hierarchical Systems and Memory Map Editor ................................ 101
  Memory Map Editor dialogs ............................................. 102
Adding an Address Region .................................................. 106
Editing an Address Region .................................................. 107

Chapter 7. SoC Designer Simulator Reference

Starting SoC Designer Simulator ............................................. 109
Starting on Windows .......................................................... 109
Starting on Linux ............................................................. 110
  Command line options .................................................. 110
Opening a Simulation ....................................................... 112
Saving a simulation: mxs, mxr, and mxc files ......................... 114
Application window .......................................................... 116
Diagram window .............................................................. 117
  Context menu for a component in the Diagram window ............ 118
  Context menu for connections in the Diagram window .......... 118
  Context menu for Diagram window ................................ 118
  Diagram window appearance ........................................ 119
Main menu and Toolbar commands ........................................ 119
  The Toolbar ............................................................... 120
  The File menu ........................................................... 121
  The View menu ........................................................... 122
  The Object menu ........................................................ 123
  The Control menu ...................................................... 123
  The Debug menu ........................................................ 125
  The Window menu ...................................................... 126
  The Help menu ........................................................... 126
Status bar ........................................................................... 127
Output window ............................................................... 127
Command Line interface ..................................................... 127
Title bar ............................................................................ 128
Task bar ............................................................................ 128
System menu and controls ................................................... 128
Tools windows ..................................................................... 129
  Parameter window .......................................................... 130
  System Configuration window ........................................ 131
Preferences dialog ............................................................. 132
  General Simulator preferences ......................................... 132
  Simulator preferences for appearance ............................... 133
  Simulator preferences for general simulation behavior ........ 134
  Simulator preferences for Tracer features .......................... 134
  Simulator preferences for profiling configuration ................. 136
  Simulator preferences debug behavior ............................... 137
Memory Map Editor ............................................................ 139
Breakpoint windows .......................................................... 139
Breakpoint Conditions and Dialogs ........................................ 139
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditions for Memory or Register breakpoints</td>
<td>139</td>
</tr>
<tr>
<td>Conditions for Signal or Transaction breakpoints</td>
<td>140</td>
</tr>
<tr>
<td>Breakpoint Manager dialog</td>
<td>141</td>
</tr>
<tr>
<td>Monitor windows</td>
<td>142</td>
</tr>
<tr>
<td>Summary Monitor dialog</td>
<td>142</td>
</tr>
<tr>
<td>Expanded Monitor dialog for signals</td>
<td>143</td>
</tr>
<tr>
<td>Expanded Monitor dialog for generic transactions</td>
<td>143</td>
</tr>
<tr>
<td>Expanded Monitor dialog for AMBA transactions</td>
<td>144</td>
</tr>
<tr>
<td>Trace Manager and the Waveform Viewer windows</td>
<td>145</td>
</tr>
<tr>
<td>Trace Manager dialog</td>
<td>145</td>
</tr>
<tr>
<td>Waveform Viewer</td>
<td>146</td>
</tr>
<tr>
<td>Using the cursors and locators</td>
<td>147</td>
</tr>
<tr>
<td>Synchronization</td>
<td>148</td>
</tr>
<tr>
<td>Using the Zoom functions</td>
<td>148</td>
</tr>
<tr>
<td>Viewing options</td>
<td>148</td>
</tr>
<tr>
<td>Context menu for the Waveform Viewer window</td>
<td>149</td>
</tr>
<tr>
<td>Profiling windows</td>
<td>152</td>
</tr>
<tr>
<td>Profiling Manager dialog</td>
<td>152</td>
</tr>
<tr>
<td>Profiling data dialog</td>
<td>154</td>
</tr>
<tr>
<td>Segment type and size</td>
<td>154</td>
</tr>
<tr>
<td>Zooming</td>
<td>155</td>
</tr>
<tr>
<td>Synchronization</td>
<td>155</td>
</tr>
<tr>
<td>Profiling data dialog context menu</td>
<td>156</td>
</tr>
<tr>
<td>Displaying a profile from the component context menu</td>
<td>157</td>
</tr>
<tr>
<td>Profile Settings Colors dialog</td>
<td>158</td>
</tr>
<tr>
<td>Profiling Settings Groups dialog</td>
<td>158</td>
</tr>
<tr>
<td>Profiling register values and transaction connections</td>
<td>161</td>
</tr>
<tr>
<td>Latency profiling</td>
<td>161</td>
</tr>
<tr>
<td>AHB Bus Legend Entries</td>
<td>162</td>
</tr>
<tr>
<td>AXI Bus Legend Entries</td>
<td>163</td>
</tr>
<tr>
<td>AHB/AXI profiling views</td>
<td>164</td>
</tr>
<tr>
<td>Sample Profile Windows</td>
<td>165</td>
</tr>
<tr>
<td>Software analysis</td>
<td>170</td>
</tr>
<tr>
<td>Working with memory content</td>
<td>170</td>
</tr>
<tr>
<td>Displaying the component memory dialog</td>
<td>170</td>
</tr>
<tr>
<td>Setting a breakpoint on a memory location</td>
<td>170</td>
</tr>
<tr>
<td>Saving and loading a memory image</td>
<td>172</td>
</tr>
<tr>
<td>Viewing registers</td>
<td>173</td>
</tr>
<tr>
<td>Tracing Register Contents</td>
<td>176</td>
</tr>
<tr>
<td>Viewing Disassembled Code</td>
<td>177</td>
</tr>
<tr>
<td>Software Function Profiling</td>
<td>179</td>
</tr>
<tr>
<td>Displaying the Function Profiling Window</td>
<td>179</td>
</tr>
<tr>
<td>Diagram View of the Function Profiling window</td>
<td>180</td>
</tr>
<tr>
<td>Summary View of the Function Profiling window</td>
<td>181</td>
</tr>
<tr>
<td>Configuring Function Profiling</td>
<td>182</td>
</tr>
<tr>
<td>Combined Software and Hardware Profiling</td>
<td>187</td>
</tr>
<tr>
<td>CheckPoint Manager and Saving Checkpoints</td>
<td>189</td>
</tr>
</tbody>
</table>
Chapter 8. Batch Mode Simulation

Overview of Batch Mode Simulation ............................................. 191
Scripted Batch Mode Simulation ................................................. 191
Running a Script from the Command Line .................................. 191
Running a Script from within Soc Designer Simulator ................. 192
Redirecting Output to a Log File ............................................... 192

Chapter 9. Debugging

Component-level Debugging ..................................................... 193
Register windows ................................................................. 194
Register breakpoints ............................................................ 195
Register tracing ................................................................. 196
Memory Overview ............................................................... 198
Full System Coherent Memory Behavior .................................. 198
Requirements for Reading or Reloading a File ......................... 199
Loading Files into Memories ................................................. 199
Sample ASCII file .................................................................. 200
Dumping memory images to a file ......................................... 200
Memory breakpoints .............................................................. 201
Disassembly window ............................................................. 201
Specifying disassembly mode .............................................. 202
Using the Run to debuggable point feature ........................... 202
Debuggers ............................................................................ 202
Synchronization of external debuggers .................................. 202
System-level Debugging and Probes ....................................... 203
Configuring Breakpoint Probes .............................................. 203
Setting breakpoints on memory or register windows ............... 203
Setting breakpoints on transaction or signal connections ......... 204
Setting breakpoints on a disassembly window ...................... 204
Viewing and Managing all breakpoints ................................. 204
Configuring Monitor Probes ................................................... 205
Inserting a monitor on a connection .................................... 205
Selecting the view ............................................................... 206
Enabling transaction dumping .......................................... 206
Clearing the Monitor dialog of data .................................... 206
Configuring Tracer Probes .................................................... 206
Configuring Profiler Probes .................................................. 208
Debugging user-defined components ..................................... 209
Debugging with SoC Designer Simulator ................................. 209
Debugging with Microsoft Visual C++ on Windows ............... 210
Attaching Visual Studio to a running process ........................ 210
Starting the debug version of SoC Designer Simulator ........... 211
Debugging with GNU gdb on Linux ....................................... 212
Setting Breakpoints in Constructors ..................................... 213
Common problems with gdb/ddd ......................................... 214
Chapter 10. The Cycle Accurate Debug Interface

CADI Interface Overview ..........................................................215
Using the CADI interface from SoC Designer Simulator ....................216

Chapter 11. The Cycle Accurate Profiling Interface

CAPI Interface Overview ..........................................................219
Profiling streams and channels ..................................................219

Chapter 12. CASI Memory Map Interface

CASI Memory Map Interface overview ........................................223
SoC Designer Canvas Memory Editor Information Flow ......................224
SoC Designer Simulator Information Flow ....................................225
Using the Bus Master from the Component Wizard ..........................225
MME Map Values in Hierarchical Systems ....................................226
Migrating existing systems to use the MME ..................................226
Automatic conversion of a system to use the MME ..........................227
Interactive conversion of system memory maps ..............................229

Appendix A. SoC Designer Component Library Configuration Files

Configuration File Overview and Content ......................................235
Maxlib.conf files ........................................................................236
Component-level .conf files .........................................................237
How SoC Designer Loads Configuration Files .................................237
.Conf file location and order referenced .......................................237
Handling of duplicate components within a .conf file .........................238
Working with configuration files ...................................................238
Creating a configuration file for a cross-platform environment ..........238
Example of creating a project-specific .conf file ..............................238
Creating a single configuration file per platform (OS) .........................242
Configuration File Syntax .........................................................242

Appendix B. Keyboard Shortcuts

SoC Designer Canvas shortcuts ....................................................245
SoC Designer Simulator shortcuts ................................................246

Appendix C. Creating a SoC Designer Runtime

Overview of SoC Designer Runtime ..............................................249
Creating the binary platform files ...............................................251
Prerequisites .............................................................................251
Preparing the system ...............................................................251
Requesting licenses ....................................................................252
Delivering the system to the end customer ....................................253
Distributing platforms containing third-party models ............................ 253
Support for end customers ................................................................. 254

Appendix D. Third-Party Software

Verilog Parser .................................................................................. 255

Glossary
Preface

Audience

This document has been written for experienced hardware and software developers to create SoC Designer systems from components that have been previously created by modeling development engineers.

SoC Designer is SystemC-based, but strong SystemC or C++ expertise is not required to use SoC Designer. Users must be familiar with the basic concepts of SystemC (such as \texttt{sc\_module} and \texttt{sc\_port}) and basic concepts of C++ (such as classes and inheritance).

Conventions

This book uses the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>courier</td>
<td>Commands, functions, variables, routines, and code examples that are set apart from ordinary text.</td>
<td>\texttt{sparseMem_t SparseMemCreate_New();}</td>
</tr>
<tr>
<td>italic</td>
<td>New or unusual words or phrases appearing for the first time.</td>
<td>\texttt{Transactors} provide the entry and exit points for data ...</td>
</tr>
<tr>
<td>bold</td>
<td>Action that the user performs.</td>
<td>\texttt{Click Close} to close the dialog.</td>
</tr>
<tr>
<td>&lt;text&gt;</td>
<td>Values that you fill in, or that the system automatically supplies.</td>
<td>\texttt{&lt;platform&gt;/} represents the name of various platforms.</td>
</tr>
<tr>
<td>[ text ]</td>
<td>Square brackets [] indicate optional text.</td>
<td>\texttt{$CARBON_HOME/bin/modelstudio [ &lt;filename&gt; ]}</td>
</tr>
<tr>
<td>[ text1</td>
<td>text2 ]</td>
<td>The vertical bar</td>
</tr>
</tbody>
</table>
Terminology

The table below lists the equivalent SoC Designer terms corresponding to the SystemC terms used in the SystemC environment:

<table>
<thead>
<tr>
<th>SystemC term</th>
<th>SoC Designer term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>component</td>
<td>The models for individual devices. For example, CPU core, memory, bus interface, and I/O.</td>
</tr>
<tr>
<td>port</td>
<td>master port</td>
<td>A port that generates transactions or signals</td>
</tr>
<tr>
<td>channel</td>
<td>slave port</td>
<td>SystemC channels are also know as sc_export.</td>
</tr>
</tbody>
</table>

Further reading

This section lists related publications. The following publications provide information relate directly to SoC Designer Plus:

- SoC Designer Plus User Guide
- SoC Designer Plus Installation Guide
- ESL API Developer's Guide
- SoC Designer Plus SystemC Linking Guide
- MxScript Reference Manual
- SoC Designer Plus CoDesign Package HDL Cosimulation Guide

The following publications provide information about ARM® or AMBA®-related architecture. See [http://infocenter.arm.com/help/index.jsp](http://infocenter.arm.com/help/index.jsp) for access to ARM documentation:

- AMBA Specification
- AMBA AHB Transaction Level Modeling Specification
- AMBA AXI Transaction Level Modeling Specification

The following publication provides reference information about ARM® debuggers:

- ARM DS-5™ user documentation
- SoC Designer™ Plus DS-5 Debugger Integration Application Note

The following publications provide additional information on simulation:

Chapter 1

Introduction

This chapter provides an overview of SoC Designer Plus. It contains the following sections:

- *Introduction to SoC Designer Plus*
- *SoC Designer Canvas*
- *SoC Designer Simulator*

**Introduction to SoC Designer Plus**

SoC Designer Plus is a simulation environment for easy modeling and fast simulation of integrated systems-on-chip with multiple cores, peripherals and memories using C++. The cycle-based scheduler and the transaction-based component interfaces enable high simulation speed while retaining full accuracy.

The SoC Designer Plus systems can be used as standalone simulation models or integrated into HW simulation or HW/SW cosimulation tools. Third-party debuggers are easily attached to one or more targets within the SoC Designer Plus systems for full visibility of component details.

SoC Designer Plus is fully based on the SystemC language. While cycle-based simulation is strongly recommended for speed and ease of debugging, SoC Designer Plus supports running any event-driven SystemC simulations.

The SoC Designer Plus suite consists of the following components:

- **SoC Designer Canvas** — A graphical application that enables you to create new SoC Designer Plus systems in addition to loading and modifying existing systems. A SoC Designer Plus system consists of leaf components or other SoC Designer sub-systems. These systems are created and edited using a graphical representation showing the components, their ports, and connections between the ports. You can also add external ports for a system and add labels to annotate the diagram. The graphical representation provides features similar to vector-oriented drawing applications, such as block diagram editors or flow charting programs.

- **SoC Designer Simulator** — A graphical application that enables you to simulate and control SoC Designer Plus systems. It makes the structure and hierarchy of the system more visible, and it enables you to modify parameters for the sub-components of a system.
SoC Designer Simulator provides extensive debugging features on the system and the component level, including a debug interface for attaching and interacting with third party debuggers. The simulated systems can be saved in a file for re-use in simulation (as an .mxs file), and saved information can optionally include the state information (as an .mxr file). (Note that the .mxr option is only available with the Save/Restore Licensing feature.)

Similarly, the SoC Designer Simulator can be run in batch mode, using the MxScript scripting file feature. MxScript is a C language-based interpretive language that can be used to execute commands from a text file that resembles a source code file.

MxScript language can be invoked for the following situations:
- using Simulator batch mode
- using command-line in the SoC Designer Plus graphic user interface: manually launching a script to run or to step through
- using SoC Designer Plus component (such as MxStub) call

For additional information on MxScript, see the MxScript Reference Manual.

- **SoC Designer Plus Component Library** — A repository of elementary components. You can create a library that includes your own custom components (created with SoC Designer Canvas and a C++ compiler) in addition to third-party Component Library components.
- **CoDesign Package** — Includes HDL CoSimulation, which can simulate SoC Designer Plus and HDL components in parallel. This facilitates design for IP re-use, co-development, and verification of the hardware and software.

**ESLAPI overview**

SoC Designer Plus components are based on the ESL API interfaces.

The ESL API consists of the following interfaces:
- The Cycle Accurate Simulation Interface (CASI)
- The Cycle Accurate Debug Interface (CADI)
- The Cycle Accurate Profiling Interface (CAPI)
- The Cycle Accurate Simulation Interfaces-Memory Map Interface (CASIMMI)

**CASI**

The Cycle Accurate Simulation Interface (CASI) defines how components are called by the scheduler and how they communicate with each other. CASI is a SystemC communication library for transaction level communication and cycle-based simulation modeling.

CASI provides the following types of class:
- **Component class** — The CASIModule class is the base class for any component. It provides the functionality for instantiating and configuring the model and sub-components. It also provides the API required for connecting other components. Unlike the interface classes, CASIModule is derived from CASIModuleIF.
- **Interface classes** — The interface classes manage communication between components.
- **Clock classes** — The clock classes manage connection to the simulation clocks.
- **Port classes** — These classes inherit from other basic classes and provide a basis for commonly used objects.
• **Factory classes** — These classes provide functionality for creating, saving, and restoring components. These classes are typically used by SoC Designer Simulator to enable additional functionality during simulation.

• **Save and restore classes** — These classes enable saving the simulation state and resuming it at a later time. This can be useful if you are debugging long simulations and only the final part of the simulation requires replaying.

**CADI**

The CADI interface can be used to:

• Display the contents of registers and memory in the SoC Designer Simulator debug windows.

• Enable interaction between an external debugger and SoC Designer Plus. This can be particularly useful if you are integrating a model of a core that has an established user base that is familiar with the debugger.

SoC Designer Simulator enables the display of register and memory windows for any component that implements a CADI interface. This feature can be used for peripherals and other components that do not have their own dedicated debugger.

**CAPI**

The CAPI interface supports a generic implementation of profiling for the component. The interface enables collecting different types of data such as, for example, addresses, register or memory values, and symbolic values.

To support profiling, a SoC Designer Plus component must:

• implement and register the CAPI interface

• supply the type of information to profile

• gather the information and call `CAPIRecordEvents()`

The collected profile data can be displayed in a SoC Designer Plus profile window or output as a database file.

**CASIMMI**

The CASIMMI interface is used to modify the memory map of components connected to a bus master. SoC Designer Plus contains a Memory Map Editor dialog that simplifies editing the memory maps for components on a master bus.
SoC Designer Canvas

SoC Designer Canvas is a graphical application that enables you to create new systems in addition to load and modify existing systems.

A system can consist of connected components or connected components and previously created systems. These systems are created and edited using a graphical representation showing the components, their ports, and connections between the ports. You can also add external ports for a system in addition to add labels to annotate the diagram. The graphical representation provides features similar to vector oriented drawing applications such as block diagram editors or flow charting programs.

The saved project files have an .mxp extension.

Canvas Window layout

Table 1 describes the graphical user interface elements in the fundamental layout of the SoC Designer Canvas:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Menu</td>
<td>The main menu contains the available options with their corresponding keyboard short cuts.</td>
</tr>
<tr>
<td>Toolbox</td>
<td>The toolbar contains buttons for frequently-used features.</td>
</tr>
<tr>
<td>Component window</td>
<td>The Component window contains a list of all the available components that can be placed over the Diagram window using drag-and-drop.</td>
</tr>
<tr>
<td>System Configuration window</td>
<td>This comprises three tabbed windows, for memory mapping, listing PrimeCells, and a Hierarchy window that shows all the instance names and types of the components and external ports currently in the system.</td>
</tr>
<tr>
<td>Parameters window</td>
<td>The parameters window contains all the parameters of the currently selected component.</td>
</tr>
<tr>
<td>Diagram window (for Canvas)</td>
<td>The Diagram window contains a graphical representation of the system.</td>
</tr>
<tr>
<td>Output window</td>
<td>The output window displays system console output and system check information.</td>
</tr>
<tr>
<td>Status Bar</td>
<td>The status bar displays information about menu items, commands, buttons, component information.</td>
</tr>
</tbody>
</table>
The Component window, System Configuration window, and Parameters window are known collectively as the Tool window (see Figure 1). The tool window group is dockable, so it can be conveniently positioned separate from the main Canvas window. The main Canvas window is known as the Diagram window for Canvas.

Figure 1 SoC Designer Canvas Tool Windows

The Output window displays system console output and system check information (see Figure 2). The Output window is also dockable, for convenient use of your console space.

Figure 2 Output Window
Figure 3 shows a Diagram window, displaying a system that has been constructed in the SoC Designer Canvas tool.

When you open the SoC Designer Canvas tool, the Diagram window, the Tool windows, and the Output window appear together, by default (see Figure 4).
Creating a System using Canvas

The typical procedure to create a system in SoC Designer Canvas is:

1. Specify the component libraries to use by entering their locations in the Preferences dialog (see General preferences, Component Library).
2. Drag components from the Component window to the Diagram window. You can also right-click on a space in the Diagram window and select Add Component from the context menu.
3. Specify the component parameters such as, for example, the memory address (see Editing Parameters).
4. Optionally, specify how the component appears in the Diagram window; for example, by hiding clock ports (see The Object menu).
5. Connect the components ports together (see Making connections between components).
6. Optionally, improve the appearance of the design by moving the component position, rerouting the connections, or adding informative labels (see Port Types, Movement, and Connection).

SoC Designer Simulator

SoC Designer Simulator is a graphical application that enables you to simulate and debug SoC Designer Plus systems. It also provides the ability to look into the hierarchy of the system and modify parameters for the sub-components of a system. Simulated systems can be saved off to a file for repetitive use. SoC Designer Simulator also provides extensive system and component level debugging features.

Simulation Features

SoC Designer Simulator supports the following features:

• controlling the simulation: run, step, stop, reset
• viewing the system structure and hierarchy
• viewing and modifying the parameters of a system
• viewing registers, memories, and disassembly through the CADI interface
• setting breakpoints on registers, memories, disassembly, transactions, and signals
• monitoring of transactions and signals
• viewing signal or transaction activity with the waveform viewer
• collecting statistical information with the profiling analysis windows
• launching and controlling of external debuggers
• automating a simulation in batch mode with scripting support
Running a Simulation

The basic steps to run a simulation are:

1. Load a simulation file (see Opening a Simulation).
2. Optionally attach monitor probes to the system (see the section System-level Debugging and Probes).
3. Run the simulation and observe the results in the runtime windows (see Monitor windows, Trace Manager and the Waveform Viewer windows, and Profiling windows).

Simulator Window Layout

Table 2 describes the graphical user interface elements in the layout of the SoC Designer Simulator:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Menu</td>
<td>The main menu contains the available options with their corresponding keyboard short cuts.</td>
</tr>
<tr>
<td>Toolbar</td>
<td>The toolbar contains buttons to frequently used features.</td>
</tr>
<tr>
<td>Output window</td>
<td>The output window displays information about the state of the simulation and messages sent from the system and its sub-components.</td>
</tr>
<tr>
<td>Command Line</td>
<td>The command line enables the user to enter simulation commands from the keyboard.</td>
</tr>
<tr>
<td>Task Bar</td>
<td>The task bar appears as soon as one or more debug or profiling windows are opened. It shows icons for all open windows.</td>
</tr>
<tr>
<td>Status Bar</td>
<td>The status bar displays information about menu items, commands, buttons, component information.</td>
</tr>
</tbody>
</table>
Figure 5 shows the SoC Designer Simulator application window.

Figure 5  SoC Designer Simulator main window

- Toolbar
- Diagram window
- Cycle counter
- Command line
- Output window
- Tools windows
The runtime windows

There are many debug and profile windows available for the different aspects of simulation and system analysis. These windows (see Figure 6) provide a high degree of visibility into system details and enable debugging and profiling of systems.

Figure 6 SoC Designer Simulator with runtime windows and task bar

Switching between the diagram and runtime windows

Use the SoC Designer task bar to manage the open windows.

All component-related debug windows are color-coded to identify the component that the window belongs to. The i field of a component is assigned a color when the debug or profiling window opens.

To temporarily minimize all runtime windows, use the context menu of the taskbar and select Show Diagram or press Ctrl-D on the keyboard. Repeating the action restores the runtime windows.

To switch between the open runtime windows and select a different window, click on the window or type Ctrl-Tab.
Saving/restoring windows between simulation sessions

If a simulation is saved into an .mxs file (by using the Save feature in SoC Designer Simulator, the state of the open runtimes windows is recorded and the windows are reopened in the same state when the simulation is reloaded.
Chapter 2

SoC Designer Canvas Reference

This chapter provides a reference for SoC Designer Canvas. It contains the following sections:

- Starting SoC Designer
- SoC Designer Canvas Application Interface Reference
- Status bar
- Diagram window
- Output window
- Component Window
- System Configuration window
- Parameter window
- Component icons displayed in the tool windows
- Preferences dialog
- Managing Plugins
- Canvas preferences
- System Properties dialog
- Label Properties dialog

Starting SoC Designer

This section describes how to start SoC Designer Canvas in Windows and Linux.

Windows

Perform one of the following:

- From the Windows Start menu, navigate to Carbon > Carbon SoC Designer > SoC Designer Canvas.
- After setting file associations, double-click on an .mxp file.

Linux

Use the console with the following command line:

> sdcanvas filename
where `filename` represents that the name of the file to open is an optional parameter. If no file is specified, SoC Designer Canvas opens with a blank Diagram window.

SoC Designer Canvas supports the command line options listed in Table 3:

<table>
<thead>
<tr>
<th>Option</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-v/--version</code></td>
<td><code>sdcanvas --version</code></td>
<td>This prints the version of the tool without starting it.</td>
</tr>
<tr>
<td><code>-n/--nomaxlib</code></td>
<td><code>sdcanvas --nomaxlib</code></td>
<td>This starts SoC Designer Canvas without loading any components.</td>
</tr>
<tr>
<td><code>filename</code></td>
<td><code>sdcanvas mySystem.mxp</code></td>
<td>Use this option to start SoC Designer Canvas with the specified file open.</td>
</tr>
<tr>
<td><code>--maxlib component_library</code></td>
<td><code>sdcanvas --maxlib mymaxlib.conf</code></td>
<td>Use this option to start Canvas with the specified library file loaded.</td>
</tr>
<tr>
<td><code>-O MXE_file mxp_file</code></td>
<td><code>sdcanvas -O mySystem.mxe mySystem.mxp</code></td>
<td>Use this option to create a SoC Designer Runtime simulation file (.mxe) from a specified project file (.mxp).</td>
</tr>
<tr>
<td><code>--display display_option</code></td>
<td><code>sdcanvas --display $DISPLAY</code></td>
<td>Use this option to set the X display. The default is <code>$DISPLAY</code>.</td>
</tr>
<tr>
<td><code>--geometry new_geometry</code></td>
<td><code>sdcanvas --geometry new_geometry</code></td>
<td>Use this option to set the client geometry of the main window.</td>
</tr>
<tr>
<td><code>--font new_font</code></td>
<td><code>sdcanvas --font arial</code></td>
<td>Use this option to set the application font.</td>
</tr>
<tr>
<td><code>--background color</code></td>
<td><code>sdcanvas --background 8000</code></td>
<td>Use this option to set the default background color and an application palette. Light and dark shades are calculated based on the new palette.</td>
</tr>
<tr>
<td><code>--foreground color</code></td>
<td><code>sdcanvas --foreground 8000</code></td>
<td>Use this option to set the default foreground color.</td>
</tr>
<tr>
<td><code>--button color</code></td>
<td><code>sdcanvas --button 8000</code></td>
<td>Use this option to set the default button color.</td>
</tr>
</tbody>
</table>
Table 3 Command line options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--visual Truecolor</td>
<td>sdcanvas --visual Truecolor</td>
<td>Use this option to force the application to use TrueColor on an eight-bit display.</td>
</tr>
<tr>
<td>--ncols count</td>
<td>sdcanvas --ncols 216</td>
<td>Use this option to limit the number of colors allocated in the color cube on an eight-bit display. If the count is 216, a 6x6x6 color cube is used (that is, six levels of red, blue, and green). For other values, a cube approximately proportional to a 2x3x1 cube is used.</td>
</tr>
<tr>
<td>--cmap</td>
<td>sdcanvas --cmap</td>
<td>Use this option to install a private color map on an eight-bit display.</td>
</tr>
</tbody>
</table>

SoC Designer Canvas Application Interface Reference

*Figure 7* shows the SoC Designer Canvas application window.

![Canvas application window](image)
This section describes the windows and controls on the application window. It includes the following sections:

- System menu and controls
- Projects tab
- Title bar
- Main menu and Toolbar
- The File menu
- The Edit menu
- The View menu
- The Insert menu
- The Object menu
- The Tools menu
- The Simulate menu
- The Window menu
- The Help menu

**System menu and controls**

SoC Designer Canvas supports the standard system menu and controls: move, size, minimize, maximize, restore, and close. Their appearance and function depends on the operating system.

See the operating system help for more information on how these work.

**Projects tab**

The tabs below the Diagram window lists the project that is currently loaded. If only one project is open, only one tab is displayed. Clicking on a tab displays the Diagram and Tools windows for that project.

**Title bar**

The title bar of application window contains the name of the application, the name of the current system being edited, and the state of the project. If the system has been modified and not yet saved, an asterisk is displayed to the right of the system name.

*Figure 8 Sample title bar for an edited ARM926 system*
Main menu and Toolbar

This section describes the SoC Designer Canvas main menu and toolbar.

Figure 9 Menu and toolbar

You can use the toolbar buttons for frequent operations instead of using the main menus. The Toolbar has the following button groups:

- **New, Open, and Save** — Use these buttons to create, open, or save projects. See the matching entries for the menu entries in *The File menu*.

- **Cut, Copy, Paste, and Del** — Use these buttons to copy or delete components. See the matching entries for the menu entries in *The Edit menu*.

- **Undo and Redo** — Use the **Undo** button to cancel the last editing change done in the Diagram window. Use the **Redo** button to cancel the last Undo. See the matching entries for the menu entries in *The File menu*.

- **Edit and Connect** — Use these buttons to change the mode for the Diagram window. In edit mode, the cursor is used to select and move components. In connect mode, the cursor is used to connect the ports of components together.

- **MemMap** — Use this button to edit the memory map of the selected bus. If the selected component port does not support the CASIMMI interface, the button is not enabled.

- **Comp, Port, and Label** — Use these buttons to specify the type of object that you are adding to the Diagram window.

- **Clock, Signal, and Transaction** — Use these buttons to hide or display the specified ports and connections in the Diagram window.

- **Grid** — Use this button to display or hide the grid in the Diagram window.

- **In, Out, and 100%** — Use these buttons to zoom the view in the Diagram window.

- **Info** — Use this button to display more information about a button on the toolbar or the windows. Click the button and then click the desired button or window to display pop-up information.

There is also a **Zoom %** toolbar that displays the current zoom level and has a spin control for increasing or decreasing the zoom level. To display this control on the toolbar, select **Zoom Toolbar** from the *Windows* menu. If the toolbar is not displayed, you can change the zoom level by selecting **Zoom In** or **Zoom Out** from the Diagram window context menu.
The File menu

The File menu includes the following elements:

- **New** — You can create a new system at any time. If there are already systems loaded, they remain loaded and can be redisplayed by selecting the system from the project tabs.
- **Open…** — You can open a previously saved system at any time. If there are already systems loaded, they remain loaded and can be redisplayed by selecting the system from the project tabs.
- **Close** — You can close the system at any time. If the system has been modified but not saved, you are prompted to save the current system.
- **Save** — You can save the current system at any time. After a modified system is saved, the title bar represents the saved status by removing the asterisk.
- **Save As…** — You can save the current system under a new name at any time. The standard file Save As dialog is invoked prompting you to enter the name of the new system. After the system has been changed, the title bar is updated to reflect the new name and saved status.
- **Print…** — You can print out the contents of the Diagram window. A Print dialog is displayed to enable you to select the printer.
- **Preferences…** — The Preferences dialog enables configuring and customizing of all features of SoC Designer. This includes generic features and features specific to Canvas, Simulator, and the Component Library. For more information, see Preferences dialog.
- **File list** — A list of all recently opened .mxp project files is displayed near the bottom of the file menu. Select a file to reopen the system.
- **Exit** — Use this menu item to exit SoC Designer Canvas. If there is a modified system and has not been saved, you are prompted to save the current system. If you click Cancel, SoC Designer Canvas remains open and the modified system is not deleted.

Read-only files can be saved under a new name and the newly named system can be loaded and modified.

The Edit menu

The Edit menu includes the following elements:

- **Undo/Redo** — SoC Designer supports the standard undo and redo editing features. The menu item shows what type of undo/redo is available. For example, if you cut a component out of the diagram, the undo menu item shows **Undo – Cut Component**. There are no limits to the number of undo/redo actions other than the operating system limitations. Changes to user preferences cannot be undone or redone.
- **Cut, Copy, Paste, Duplicate, and Delete** — SoC Designer supports the standard cut, copy, paste, duplicate, and delete editing features. Connections are only pasted if the two ends of the connection are part of the pasted cut/copied selection. For example, if one component and its connections are copied, only the component itself can be pasted.
- **Select All** — This option selects all the objects within the diagram.
- **Edit Mode, Movement Mode, Connect Ports Mode** — The application has three different modes:
  - The edit mode is used for normal editing (for example, adding, moving, cut/copy/paste/delete).
  - The movement mode is for moving the system around within the Diagram window using the mouse (as opposed to using scroll bars).
  - The connect mode is used for connecting ports of components and external ports.
To exit connect mode and return to edit mode press the ESC key. To temporarily enter connect mode while in edit mode hold down the Shift key (no component must be selected) and make a connection (quick connect).

The cursor appearance changes to indicate the mode. In connect mode for example, moving the cursor over a valid connection port outlines the port in a green shaded pattern and the cursor changes to the connect icon.

- **Memory Map Editor** — Select this option to display the Memory Map Editor dialog that enables configuring the memory map for bus masters (see Memory Map Overview).
- **System Properties…** — Select this option to display the System Properties dialog box (see System Properties dialog).

### The View menu

This menu controls the viewing options of the SoC Designer Canvas such as zoom, grid and display options:

- **Clock Connections, Signal Connections, and Transaction Connections** — These options toggle the display of the specific connection type. If off, the ports and connections are not displayed. (You can also toggle the display by clicking on the Clock, Signal, or Trans, buttons in the toolbar.)
- **Grid** — This option toggles the display of the grid. (You can also toggle the display by clicking on the Grid button in the toolbar.)

  **Note:** The movement of objects is bound by the snap to grid flag, not the display of the grid.

- **Grid Options…** — This option displays the Grid Options dialog that contains the following options:
  - **Show Grid [default = on]:** — Show the grid in the diagram with the specific options.
  - **Snap to Grid [default = on]:** — When moving or resizing, snap the object to the grid boundary.
  - **Grid Horizontal Spacing / Grid Vertical Spacing [default = 10]:** — The horizontal/vertical distance between grid points in pixels with a zoom of 100%.
  - **Grid Point Diameter [default = 1]:** — The diameter of the grid points in pixels with a zoom of 100%.
  - **Select Point Color… [default = gray]:** — The color of the grid points.

There is also a button to reset all the values back to the factory default settings.

*Figure 10 shows the Grid Options dialog with the default values.*

![Grid Options dialog](image)
• **Layout** — Displays three submenu entries that you can use to control the display of the Diagram window:
  
  • **Reset Default Layout To Factory Settings** changes the layout of the currently selected component back to the factory layout, that results in all slave ports being on the left hand side and all master ports being on the right hand side in an alphabetical order.
  
  • **Save Current Layout as Default** saves the layout of the currently selected component as the default layout for this component. This default layout is used for all components of that type that are dragged and dropped into the diagram from this point on.
  
  • **Reset to Default Layout** reloads the default settings. This is the factory settings unless you have saved a new default setting.
  
  • **Zoom In/Zoom Out** — These options zoom the diagram in/out. The step size changes depending on the current zoom level.
  
  • **Set Zoom to 100%** — This option resets the zoom level to the default of 100% (this is the normal zoom).
  
  • **Zoom to Fit** — This option sets the zoom level so that the entire diagram fits into the Diagram window.
  
  • **Set Zoom...** — This option displays a dialog that enables you to set the amount of zoom. The zoom can be in the range of 10% to 500%.
  
  • **Center on Selected Object(s)** — This option centers the Diagram window over the selected object or objects.
  
  • **Center Diagram** — This option centers the Diagram window over the center of the system model.
The Insert menu

The Insert menu includes the following elements:

- **Add Component…** — Displays a dialog that contains all the available components that can be added to the diagram (see *Adding a component to the Diagram window*).
- **Add External Port…** — Displays the External Port dialog used to specify the type of port to add. SoC Designer remembers the last port and type selected so that the next time the dialog opens, they are selected (see *Adding an external port to a system*).
- **Add Label** — Sets the mouse cursor to a default label. To add a label, move the label to the location desired and click the left mouse button. The label properties dialog is displayed to enable you to change the label text and properties (see *Label Properties dialog*).

The Object menu

The Object menu includes the following elements:

- **Auto Route Connection** — This option removes any manually-placed points or manually-moved line segment end points from the connection and then reroutes the connection using the auto-routing feature of SoC Designer.
- **Mirror External Port** — This option switches the direction in which the external port is drawn. It does not reverse the signal direction (a master port remains a master port). See *Working With External ports*.
- **Enable/Disable Port** — In some cases, certain ports might not be used or might not require connection. Since SoC Designer checks that all ports are connected, these ports cause an error or warning message. This option enables you to disable such ports so that no error or warning message is displayed. Disabled ports appear gray, and enabled ports appear black.

To disable all unconnected ports at the same time, right-click on the component and select Disable All Unconnected Ports from the context menu.

- **Hide Port/Unhide Port** — Use this option to disable the selected port and make it invisible in the Diagram window. To recover a hidden port select **Object → Unhide Port** or **Object → Unhide All Ports**.
- **Unhide All Ports** — This option shows all ports that have previously been hidden. This might result in overlapping port symbols if there is not enough space for displaying all of the hidden ports. SoC Designer does not automatically increase the size of the component if the unhidden ports do not fit into the boundaries of the component.

- **Component Display Options…** — The following options are available for the display of components:
  - **Display Title Bar**
  - **View Model Name**
  - **View Type**
  - **View Logo**

The model name and type options affect how the title bar portion of the component appears.
Figure 11 shows an ARM component with all options enabled.

![Figure 11 All component display options on](image)

Figure 12 shows the ARM component with all display options disabled.

![Figure 12 All component display options off](image)

- **Open Sub-System**... — If the selected component is a hierarchical component and the system description is available, selecting this option opens the sub-system in another tab of SoC Designer Canvas.

- **Rename**... — This option displays a dialog for you to enter a name for the object. Each object in the system must have a unique name identifier. The system checks that the name of the object is unique and notifies you if it is not.

- **Enable/Disable Component** — Use this entry to enable or disable a component in the model. You might do this, for example, to test the system design.

- **Edit Parameters**... — This option displays the Edit Parameters dialog to view and edit the object parameters (see Editing Parameters). You can also access the Edit Parameters dialog by double-clicking on the component, or right-clicking and selecting **Edit Parameters**... from the context menu.

The Tools menu

The Tools menu includes the following elements:

- **Auto Route All Connections** — Performs the autoroute for every connection in the diagram.

- **Check System**... — Checks if the system being developed has any errors or missing information. For example, warnings or errors are displayed if:
  - A signal or transaction port of a component is not connected but is enabled.
  - A port is connected but disabled.
  - A signal or transaction master port is not connected.
  - The system has no version set.
  - The system has no description.
The System Check tab in the Output window outputs two sections:

- **Error section** — Contains problems with the system that might prevent it from running correctly. For example, if a component has no connections, it might not simulate correctly.

- **Warning section** — Contains information that might be an error such as “the system being developed has no description.”

**Generate Platform for SoC Designer Runtime Simulator (mxe)…** — Generates an encrypted version of the configuration (.mxe) file for IP proliferation. This encrypted file can be shipped to third parties and run with the SoC Designer Runtime. See the section *Overview of SoC Designer Runtime* for instructions on generating and delivering platforms.

*Note:* Licensed third-party libraries must be delivered in the package supplied by the library owner. You must only directly distribute libraries that you have created yourself.

**Component Wizard…** — Starts the SoC Designer Component Wizard. The Component Wizard facilitates efficient component creation. The wizard can also produce the project files for importing SystemC or CASI components.

**Add Current System to Component Library…** — Saves the current system and invokes the SoC Designer librarian to add the current system to the library. A wizard guides you through a series of steps to add the system to a Component Library file.

**Add Component to Component Library…** — Invokes a wizard that guides you through a series of steps to add a library (DLL or shared object) or a SoC Designer system (.mxp or .mxe file) to a Component Library file. See the section *Using your new component in SoC Designer* for more information.

**Manage Component Library…** — Opens the Component Library category of the Preferences dialog. See *The File menu* for more details on Component Library administration.

### The Simulate menu

The Simulate menu includes the following elements:

- **Simulate System…** — Opens the simulation window (SoC Designer Simulator) with the current system. Because a temporary simulation file is generated and passed to the simulator, you can simulate while continuing to work on the system. Before SoC Designer Simulator is invoked, the check system function is called (if enabled; see *Preferences dialog*). If there are any errors or warnings, the Check System dialog box appears, indicating you can proceed or cancel the operation.

If your system has the cosimulation option enabled, the simulate system menu item displays the cosimulation type and when called launches the respective cosimulation. If a license for the respective feature is available, this submenu enables the user to start cosimulation with ModelSim.

- **Simulate System (DEBUG)…** — Opens the SoC Designer Simulator and loads the current system using models that have been built with debug symbols.

*Note:* On the Windows platform, it is illegal to mix libraries that link with release and debug runtime libraries in a single application. All SoC Designer models should therefore be linked with the release runtime libraries on Windows.

- **Enable HDL Cosimulation** — Toggles between enabling or disabling the HDL cosimulation feature. For information on HDL, see the *SoC Designer CDP HDL Cosimulation Guide*.

- **HDL Cosimulation** — Initiates HDL Cosimulation. For information on HDL, see the *SoC Designer CDP HDL Cosimulation Guide*. 


The Window menu

The Window menu includes the following elements:

- **Main Toolbar/Zoom Toolbar** — Toggles all toolbars on/off. The **Zoom Toolbar** option toggles the zoom toolbar on/off.
- **Tools/Component/System Configuration/Parameter Window** — Toggles the specific tool window on/off.
- **Tools Window** — Shows/hides all selected tool windows. This option must be on to see any of the three tool windows.
- **Output Window** — This option toggles the output window for the current system on/off.
- **Master Output Window** — This option displays a separate dialog showing all the output of all systems in a single window.
- **Next/Previous Window** — If more than one project is open, use these options to navigate between them.

The Help menu

The Help menu includes the following elements:

- **Help** — Launches the Adobe Acrobat Reader with this document.
- **Show All Pop-Up Help** — Resets all the flags to show all the pop-up help that you have turned off.
- **Don't Show Pop-Up Help** — Sets all the flags to not show any pop-up help.
- **Tip of the Day...** — Opens the Tip of the Day dialog box to the current tip of the day.
- **About...** — Displays the standard About dialog box displaying SoC Designer version information. A **Plugins** button enables you to display the list of currently installed Plugins.
- **About System...** — Displays detailed SoC Designer and System version and license information. Also displays the loaded component libraries.
- **What's This** — Allows you to request information about most controls and objects in the application. Select the menu option and then click a control or area of the application window to display a brief help message. The **What's This** help gives more detailed information than a tool tip but less information than the full help system.
Status bar

The status bar displays information about menu items, toolbar buttons, and objects under the mouse cursor. For example, moving the mouse cursor over a menu item or toolbar button displays a short description of what the action does in the status bar. If the cursor is over a component in the Diagram window, the status bar reflects the instance name of the component, the model name, and the type of component (see Figure 13).

Figure 13 Status bar showing component details

![Diagram of ARM9E-CX example](image_url)

Clicking on the information box (the i at the upper right corner) displays additional information about the component (for example, name, version, and whether the version is locked).

Diagram window

This section describes the contents of the Diagram window (Figure 14).
The Diagram window behaves similarly to a block diagram editor or flow-charting tool. This window is where you:

• place components
• place external ports
• connect the objects together

The view displayed in the Diagram window is a view of the total system diagram. If the system is small, the entire system can be displayed within the window. For large systems, the view is of a small portion of the total system. There is no limit to the size of the system other than the limits imposed by the operating system or computer hardware.

Mouse cursors, tool tips, and status information

Table 4 shows all the cursors used in the Diagram window. The cursor column shows what the actual cursor looks like. The typical usage column shows how the cursor looks when used.

<table>
<thead>
<tr>
<th>Cursor</th>
<th>Typical Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is the standard selection cursor. Clicking on an object selects that object. This cursor is also used for the lassoing of multiple objects.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is the drag-and-drop cursor. If the component is in the Diagram window, the diagram representation of the object is displayed under the cursor.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is the cursor shown when in movement mode.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is the general invalid / not applicable cursor. This is shown if an operation is not valid.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is one of the eight-resize cursors, one for each of the compass points and corners.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is the select port cursor. Clicking on a port of a component selects that port and deselects all other ports.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>This is the start connection cursor. If in connection mode, and prior to starting a connection, this cursor is used when over a valid port.</td>
<td></td>
</tr>
</tbody>
</table>
The standard tool tips and status information are displayed in the status window whenever the cursor is over a menu item or toolbar button. Hovering over an object within the Diagram window displays detailed information in the status bar.

### Scroll bars and scrolling the View port

The scroll bar pucks (the part of the scroll bar that moves) for the Diagram window are proportional to the amount of total system diagram within the view port an amount equal to half the view port width around the edge of the system. You can quickly scroll the system mostly off the edge of the view port and add new objects into the empty portion revealed around the edge.

If you are using the scroll bars to scroll the view port, the view continuously repaints to make it easier for you to obtain the desired view.

### Moving the View port

If SoC Designer Canvas is in movement mode, pressing and holding the left mouse button down enables you to move the view port within the system diagram using the mouse rather instead of using the scroll bars. The limits to the movement are the same as the limits for the scroll bars. The scroll bars track the movement.

If Canvas is in edit mode, press and hold the control key to temporarily switch to movement mode.

### Zooming in and out in the Diagram Window

You can zoom the diagram in or out. The maximum zoom in is 500% and the maximum zoom out is 10%. When zooming out, the contents of the components changes as the size of the component becomes smaller. SoC Designer attempts to display the most important information about objects when zooming out. You can hover the mouse cursor over an object to display its name in the status bar.

To quickly access the zoom feature, press the Shift + Ctrl keys and use the left or right mouse buttons to zoom in or out.

<table>
<thead>
<tr>
<th>Cursor</th>
<th>Typical Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="End Connection Cursor" /></td>
<td>This is the end connection cursor. This is shown if a connection is started (and not yet completed) and the cursor is over a valid end port.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Moving Connection End Point Cursor" /></td>
<td>This is the cursor shown for moving a connection end point. This is used for selecting and placing.</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Move Connection Line Cursor" /></td>
<td>This is the move connection line cursor. This is used for the selection of points and manual placement of points.</td>
<td></td>
</tr>
</tbody>
</table>
Selecting objects in the Diagram Window

To select a single object, left-click on the object in the Diagram window. To select multiple objects, make selections holding down the control key.

You can also use the lasso feature to select objects. Press and hold the left mouse button and then drag the mouse to select all objects within the lasso rectangle. Any object that is partially within the bounding rectangle is selected.

To clear the selection list, click the left mouse button in the background area of the Diagram window.

To remove a single object from the selection list, hold the control key down and left-click on the object.

To select all objects, use the menu option Edit → Select All or lasso the entire system.

Diagram Window Context menus

The context menu displayed if you right-click in the Diagram window depends whether a component is selected or not.

If no component is selected, the general context menu shown in Figure 15 is displayed. The entries duplicate functionality found in the main menus (see Main menu and Toolbar for additional information).

Figure 15 Context menu for Diagram window, no component selected
If a component is selected, the context menu shown in Figure 16 is displayed. See Component context menu for more details.

![Figure 16 Context menu for component in Diagram window](image)

**Display options**

The display of the object type, port list, and sub-component list can be disabled in the SoC Designer Canvas preferences settings. See The Tools menu for more information.

**Updating tool windows**

When you perform an action in one window, all the other windows reflect the action (if applicable). For example, if you select a component within the Diagram window, the System Configuration window is updated with the selection and the Parameter window shows the parameters of the selected component.
Resizing windows

The vertical separator between the Diagram window and the Tool windows can be moved horizontally (see Figure 17) or between the individual windows of the Tool Windows or between the Diagram window and the Output window can be moved vertically.

Figure 17 Moving the window separator

Output window

The Output window contains log information printed by SoC Designer Canvas:

- **Output tab** — Status and error text related to loading systems or Model Libraries is displayed in the window when the Output tab is selected.
- **System Check tab** — Status and error text related to system validation is displayed in the window when the Output tab is selected. Select the tab and then select Check System from the Tools menu to validate the system. Error messages might be displayed if, for example, a port is not connected or a library is missing.
Component Window

This section describes the Component Window, which contains a list of all the available components that can be placed into the Diagram window.

This section includes the following topics:

- Component Window Icon and List View
- “Not Available” symbol
- Drag-and-drop from the Component Window
- Component Window context menu
- Context menu for components in the component window
- Customizing the Component Window tabs
- About component versions

Component Window Icon and List View

Each component has an icon that best represents the component type and the model name of the component. The list is shown as an icon view (Figure 18) or a list view (Figure 19).

![Figure 18 Component Window icon view](image)

![Figure 19 Component Window list view](image)

To switch between icon view and list view, right-click anywhere in the Component Window and select the desired option from the context menu. A scroll bar is provided if the components do not fit in the current window size.
The Component Window contains tabs that group the available components. The standard tabs are:

- **All** — Displays all the available components.
- **Bus** — Displays bus components.
- **Core** — Displays core components.
- **Memory** — Displays memory components.
- **Other** — Displays all components that do not fall into one of the listed categories.
- **PrimChnl** — Displays the primitive channels, provided that include SystemC primitive channel components is selected in the Component Library section of the Preferences dialog.

**“Not Available” symbol**

The **NovP** component in the icon view in Figure 20 has a Not Available symbol around it. This indicates that the component is not available for use. This can occur if the component is missing from the Component Library, or it could not be created.

![Figure 20 Component Window with bad component](image)

**Drag-and-drop from the Component Window**

To drag-and-drop a component from the Component Window:

1. Press and hold the left mouse button over the component you want to add to the Diagram window.
2. Move the component over the Diagram window to the location you want to add the component.
3. Release the left mouse button. The component is added to the diagram. SoC Designer prevents you from dropping a component on top of an object that already exists in the system.
Component Window context menu

Click the right mouse button in the Component Window background (that is, not over a component icon) to display a context-sensitive menu containing the following options shown in Figure 21.

Figure 21 Context menu for Component Window

This menu includes the following options:

- **Component Window Icon View** — Toggles between the icon view and list view for the Component Window.
- **Manage Tabs…** — Displays a dialog that you use to organize the contents of the tabbed windows. See Customizing the Component Window tabs for additional information.
- **Add Current System to SoC Designer Component Library…** — To use the current system as a hierarchical component in other designs, you can add the system to the SoC Designer Component Library.
- **Add Component to SoC Designer Component Library…** — Adds a component to SoC Designer Component Library. The component can be a SoC Designer system, a component DLL, or a shared object. For more information on this feature, see Using your new component in SoC Designer.
- **Refresh Component List** — Clears the current component list in the Component Window, re-parses the Component Library file, and repopulates the Component Window. This is useful if you make Component Library-related changes and do not want to restart SoC Designer.

*Note:* The system diagrams that are currently open are not updated automatically. You must close and reload affected systems to see changes in the SoC Designer Component Library components.
Context menu for components in the component window

Right-click over a component in the Component Window to display the component context menu (Figure 22). Note that this context menu differs from the context menu that appears when the component is on the Canvas.

Figure 22 Component context menu

This menu includes the following options:

- **Show Error Message** — Displays any error messages associated with the component (not able to load the component library, for example).

  The Error Message dialog box displayed contains the file name for the component, the library file, the error text, and paths to the Component Library file containing the component.

  *Note:* This item is not included in the context menu if there are no error conditions for the component.

- **Reload Component** — Reloads only the selected component in the Component Window. This is useful if you have many components in the Component Window and you want to update a specific component.

  *Note:* The instances of this component within an opened system diagram are not updated automatically. You must close and reload the system to see the new component properties.

- **Edit System**… — Allows you to edit hierarchical components. (Each hierarchical component has its own .mxp file).

  *Note:* If the selected component is not a hierarchical component, this option is not included in the context menu.

- **Remove Component from Component Library** — Removes the selected component from its Component Library file but does not delete the model DLL. If multiple components with the same identification have existed within the active Component Library files, only the first one is deleted.

  *Note:* The instances of this component within an opened system diagram are not updated automatically.

- **Edit Component configuration file**… — Starts a text editor and opens the Component Library file that includes the selected component.

- **Component Window Icon View** — Toggles between the icon view and list view for the Component Window.
- **Refresh Component List** — Reloads the component by rescanning the various Component Library files specified in the Preferences dialog.

- **View Component Parameters**… — Displays the Parameter dialog to view the component properties and parameters (see *Editing Parameters* for more information).

### Customizing the Component Window tabs

SoC Designer allows you to customize the Component Window tabs to further organize the components. You can modify or remove existing tabs and add new ones:

1. Right-click next to the tabs and select Manage Tabs from the context menu (*Figure 23*).

*Figure 23 Component Window context menu*

![Component Window context menu](image)

The Component tab manager opens (*Figure 24*).

*Figure 24 Component Window, Tab Manager*

![Component Window, Tab Manager](image)

The Manage Tabs dialog has three list views:

- The Window Tabs list shows the currently-existing tabs. Enabled checkboxes represent active options.
- The Component Types list shows the component types associated with the tab that is active in the Window Tabs list.
- The right-most list shows additional component types. The list is automatically generated by looking at all currently-loaded components.

2. To create a custom tab name, click **Add Tab**. Then use the arrow buttons to specify the component types you want to appear on that tab.
3. To create new component types, click **Add Type**.

4. To change the order of the tabs in the Component Window, use the up/down arrows.

Use the Edit and Delete buttons for any additional modifications. The tabs *All* and *Other* may not be edited or deleted; however, you can disable them so that they do not appear in the Component Window.

Click **Reset** to set the component tabs back to their original configuration. Clicking **Reset** disables custom tabs but does not delete them.

### About component versions

SoC Designer supports multiple versions of the same component. This facilitates switching between different versions for architecture exploration or comparisons and enables backward compatibility.

A version of a component has the same component name but differs in the version string. The differences between different model versions must only be on the level of behavior/implementation, not interfaces and parameters.

The Component Window in list view displays the version information. If multiple versions of a single component exist, they are displayed in a hierarchy level below the component name as shown in Figure 25.

![Figure 25 Component Window with versions shown in list view](image)

If a component has not yet been used in a diagram, you can drag any version into the diagram. SoC Designer distinguishes the default version and the current version:

- The component version displayed in blue represents the default version. This is the version that would be used if no version preference is specified, for example, by dragging the component from the icon view Component Window.
- Once a component has been used in a system, it is not possible to use any other version than the one currently selected. This version is called the current version. All other versions are displayed in a light gray color to represent that they cannot be used in the design.

To use a different version, you must delete and replace the version used in the Diagram window.

### Locking component version

SoC Designer enables locking the component version for each diagram. This means that when a system with a locked component is loaded, SoC Designer loads the locked component version instead of the default one. If the preferred component version is not found, a warning message is displayed.
Components with locked version information are drawn with an asterisk in the information field as shown in Figure 26.

**Figure 26** Locked and unlocked components

To lock a component version:

1. Right-click on a component in the Diagram window.
2. Select Lock Version/Variant from the context menu.

**System Configuration window**

The System Configuration window (Figure 27) contains a list of all the components and external ports in the system displayed in the Diagram window. Each component has a sub-list of all its ports and sub-components. Selecting an object in the Diagram window also selects the object in the System Configuration window.

**Figure 27** System Configuration window

Right-click on an object in the System Configuration window to display a context-sensitive menu of command options specific to the object type.
Parameter window

The Parameter window (Figure 28) contains all the editable parameters of the currently-selected component. If more than one object is selected, or the selected object is not a component, the Parameter window is blank.

![Parameter window](image)

The Parameter window contains the following columns:

- The **Parameter** column lists all the parameters available to be edited.
- The **Value** column contains the current values for all the components.
- The default column contains a button to reset values back to their defaults. If the value currently is the default, the button is disabled. Otherwise, the button displays the Reset icon.
- The double-arrow column indicates parameters that are editable at run-time in the SoC Designer Simulator.
- The **Type** column indicates whether a parameter is a Boolean or numerical variable.

**Note:** To see all of the editable and read-only parameters associated with a particular component, right-click on a component and select **Edit Parameters**… from the context menu.

Refer to **Editing Parameters** for further information; refer to the model guide for your component for parameter descriptions, available settings, and defaults.

Component icons displayed in the tool windows

*Figure 29* shows examples of component icons.

![Component icons](image)

The icons are, from left to right: core, memory, system, clock divider, bus decoder/fan out, other, external port, and port. These small icons are used in the **Component Window** (when it is in list view) and in the System Configuration window.

*Figure 30* shows the icons that are used for the **Component Window** (in icon view) and on the components themselves. Some models display their own icon in the Diagram window.

![Component icons](image)
The external port and port icons are displayed in the System Configuration window, but not in the Component Window.

Preferences dialog

Use the SoC Designer Preferences dialog (Figure 31) to configure SoC Designer preferences. To open the Preferences dialog, select File → Preferences from the main menu of the SoC Designer Canvas or Simulator tool. The Preferences dialog is a standard tree dialog, with major features on the left, and details of each feature on the right.

**Figure 31 SoC Designer Preferences, complete dialog**

General preferences

The preferences in this category affect the SoC Designer Canvas, in addition to the SoC Designer Simulator windows.

**Figure 32 General Preferences, features**

The General group contains items that enable you to modify application options:

- **Display Tool Tips**: Shows tool tips when hovering over elements.
- **Show Information**: Displays information about the current design.
- **Show Splash Screen on Startup**: Shows the splash screen when the application starts.
- **Show Message to advance time during simulations**: Shows a message to advance time during simulations.
- **Use Last Visited Directory for Open/Save**: Use the last directory visited for opening and saving files.

- **Home Directory**: The directory used for opening and saving files.
- **Text Editor**: Editor for modifying text files such as a component configuration file.
- **Browser**: Browser to open HTML document.

The General group contains items that enable you to modify application options:
• **Display Tool Tips** [default = on] — Enables the display of all tool tips within the application.

• **Show Status bar** [default = on] — Toggles the display of the status bar on the bottom of the SoC Designer Canvas and SoC Designer Simulator windows.

• **Show Splash Screen on Startup** [default = on] — Enables you to toggle between showing the splash screen on startup.

• **Show Message to advance time during save/restore** — Applies to Swap & Play functionality. When creating a Swap & Play checkpoint, all components must be at a debuggable point; otherwise, SoC Designer automatically advances the simulation until each component reaches a debuggable point.

By default, SoC Designer requests confirmation before advancing the simulation, proceeding only after you click OK. If you disable the **Show Message to advance time** option, SoC Designer advances the simulation and creates checkpoints without requesting confirmation.

• **Use Last Visited Directory for Open/Save** [default = off]: — Returns you to the last-used directory if you select Open or Save. Selecting this option eliminates the requirement to change directory if you are repeatedly opening, modifying, and saving the same project.

• **Home Directory** — Enables you to specify the path to your home directory. This directory is part of the SoC Designer search path.

• **Text Editor** — Enables you to specify the name of the text editor to be used by SoC Designer. This is the editor that SoC Designer invokes for you to use if you select **Edit Configuration File...** from the context menu of a component. The default value is “notepad.exe” for Windows, and “vi” for Linux.

• **Browser** — Applicable to Linux only. This setting specifies the application to launch for browsing html files. On Windows, the default web browser is used.

**General preferences, Appearance**

The General Preferences Appearance dialog box *(Figure 33)* controls the appearance of text and icons on the application windows.

*Figure 33 General Preferences, Appearance*
• The Toolbar-related options are:
  • **Use Large Icons [default = off]**: Enables using the large version of the toolbar icons instead of the small versions.
  • **Display Text Labels [default = on]**: Displays the name of the toolbar button below the icon of the button.

• The Magnify Object Delay setting specifies that, if the zoom is set to less than 70%, hovering the cursor over a component magnifies the component to its original size after a delay. This setting adjusts the time before the component is shown magnified.

• Use the controls in the **Font** panel to select font type and size for text displayed for the System (Linux only), Output Window, Components, Component Ports, and External Ports. The System font is the font used in the application window. The rest correspond to the fonts used by the objects in the diagram.

  **Note:** Under Windows, the System font cannot be changed using this tool. Modify the Windows System font from the Windows Display Properties dialog that is part of the operating system.

**General preferences, Component Library**

This section describes how configuring setup options affects the component libraries that SoC Designer can locate. **Figure 34** shows the Preferences dialog for Component Library.

**Figure 34 Preferences dialog, Component Library**

This dialog includes the following elements:
• **Working Directory** — Enables you to specify where SoC Designer Canvas searches for the Component Library files. The components in these files are used to fill the Component Window. There are three options:
  • Use the current directory. This is the default, and it is the directory in which SoC Designer was started.
  • Use a specified directory.
  • Set it to the location of the currently-loaded configuration file.

These options are useful if SoC Designer Simulator must load a specific SoC Designer Component Library configuration file, but the configuration file in the current directory has the highest priority.

• **Additional Component Configuration Files** — Manually add and remove Component Configuration files using the Add and Remove buttons. Only those files that are checked (enabled) are searched for components when an SoC Designer model initializes. Use the Enable and Disable buttons to specify the files that to load.

Highlight any file and click the up and down arrows to the right of this section to change the priority of that file. Files near the top of the list have greater priority than those near the bottom.

*Note:* If duplicate components exist, only the first one is available.

• **Include SystemC Primitive Channel components in the Component Library Repository** — Include or exclude SystemC channel components in the component library. By default the loading of channel components is disabled.

**Search order for component library (CONF) files**

The available component library files are searched for components in the following order:

1. Files specified in the script file if SoC Designer is running in batch mode.
2. Files specified on the command line when SoC Designer was started.
3. Files in the specified working directory. By default, this is the current directory. Use the **Working Directory** option to specify the current working directory.
4. Files added in the Preferences dialog.

Use the **Additional Component Configuration Files** options to add or delete search paths or to change the search order.

**Managing Plugins**

You can load or unload plugins using the Plugins panel of the SoC Designer Preferences dialog.

*Note:* Modifications you make to your plugin specifications are not enabled until you restart SoC Designer.

You can control the loading and unloading of plugins with the following features:

• The checkbox Include plugins from the $(MAXSIM_HOME)/etc/Plugin directory is always set to on. This means that plugins stored in this directory are always loaded first by the SoC Designer tools.

• The checkbox Include plugins from the $(MAXSIM_PLUGINS) environment variable enables you to optionally load plugins in this directory after the plugins in the $(MAXSIM_HOME)/etc/Plugin directory load.
• The button **Manage Plugins** opens the **Manage Plugins** dialog for controlling the inclusion or exclusion of the specific DLLs loaded from the $(MAXSIM_HOME)/etc/Plugin and $(MAXSIM_PLUGINS) directories. DLLs in these directories are listed in the Manage Plugins dialog (Figure 35).

**Figure 35 Manage Plugins dialog**

![Manage Plugins dialog](image)

Load or unload any listed DLL by enabling or disabling the checkbox next to its name.

Certain plugins, known as Tools plugins, are required for SoC Designer to load. Tools plugins are not listed in the Manage Plugins dialog. The plugins that can be managed with the Manage Plugins dialog are known as User plugins.

The Manage Plugins dialog does not load plugins stored in the directory $(MAXSIM_PLUGINS)/etc/Plugins, unless the checkbox for Include plugins from the $(MAXSIM_PLUGINS) environment variable is enabled.
Canvas preferences

Figure 36 shows the Canvas general preferences panel.

This panel includes the following elements:

- **Application panel:**
  - **Output SoC Designer Messages to Text File [default = on]:** Enables the logging of SoC Designer messages to the application output file. The name of the file is SoC DesignerMessageOutput.txt.
  - **Perform a System Check prior to Simulating [default = on]:** Enables a system check prior to simulating. If errors or warnings exist, a message box is displayed.
  - **Reload last project on application startup [default = off]:** Reloads the last project (that is, the first project listed in the file menu history list).
  - **Save As dialog adds sub-directory automatically** — Automatically adds a sub-directory to the current directory so that all the project files go into the sub-directory by default. For Windows, this is on by default. For Linux, it is off by default.
  - **Enable Component Registration Cache** — Speeds up the component loading time since the tool uses the cached component information from previous sessions. The option can be turned off to force explicit reloading of all component libraries when the component list is refreshed.

- **Save panel:**
  - **Generate Backup File [default = on]:** If this option is checked, every time a system is saved and there exists a system under the same name, the existing system is renamed to filename~bc in the same directory.
  - **Auto Save [default = on]:** Causes the application to auto save the current system every x minutes. If the application was not properly shut down, the next time it is run, the user is prompted to load the auto save file.
  - **Auto Save Every: x Minutes [default = 5]:** Specifies the interval for auto save. Minimum time is one minute.
  - **Clear Recent File History List** button — Clears the recent file list that is displayed under the main File menu.

*Note:* This option cannot be cancelled after the button is pressed.
Canvas preferences, Appearance

*Figure 37* shows the Canvas Appearance Preferences dialog.

*Figure 37* SoC Designer Canvas Appearance preferences

This dialog includes the following elements:

- **Hierarchy View Options** panel:
  - **Show Sub-Components [default = on]** — Displays the list of sub-components for all system type components currently in the diagram.
  - **Show Ports [default = on]** — Shows the list of ports for all components that contain ports.
  - **Show Types [default = on]** — Shows the type of each object next to the name of the object.

- **Component Window Options** panel:
  - **Icon View or List View [default = icon view]** — Specifies the default view of components in the Component Window. You may prefer the List view if component names are long.
  - **Show unavailable [default = on]** — Toggles the display of components that are part of the SoC Designer Component Library, but are not currently available.
Canvas preferences, Diagram window

Figure 38 shows the Canvas Diagram Preferences dialog.

This dialog allows you to set preferences for the behavior of the Diagram window:

- **Double-clicking on objects** panel — Specify that double-clicking on a component either:
  - Launches the Edit Parameters dialog. This only works for objects that have parameters.
  - Allows you to rename the object.

- **Grid options** panel:
  - **Show Grid** — Selects toggling the display of the grid.
  - **Snap to Grid** — Toggles whether the cursor and objects must snap to the grid.
  - **Grid Spacing** — Sets the horizontal and vertical spacing between grid points.
  - **Grid Point Diameter** — Sets the diameter of grid points in pixels.
  - **Select Point Color**... — Sets the color of grid points.

- **Miscellaneous**:
  - **Draw Diagram with Color [default = on]** — Enables drawing the diagram only using black, gray, and white.
  - **Select Background Color [default = white]** — Enables the background color to be modified.
  - **Auto arrange ports when resizing component [default = on]** — Automatically moves the ports if the component is resized to be smaller. Ports move back to their original positions when resized to be larger.
System Properties dialog

Use the System Properties dialog for viewing and modifying system information. To open this dialog, select the Edit > System Properties... from the main toolbar. The dialog contains three tabs (see Figure 39):

**Figure 39 System Properties dialog**

![System Properties dialog](image)

**Properties tab**

The Properties tab contains the version, description, and timing settings for the system. The version and description and system clock period are editable.

The system’s top level clock period can be set in femto, pico, nano, micro, or milliseconds. You can enter any decimal value that can be expressed using whole femto seconds. The selected unit is used for display purposes in the waveform viewer. Entering $10\text{ps}$ results in the same frequency as entering $0.01\text{ns}$, but in the latter case the waveform viewer displays the time in nanoseconds.

The time resolution and default time unit for the SystemC event-driven simulation can be set also from the same dialog

*Note:* The SystemC time resolution and default time units are only useful if event-driven SystemC modules are present.

**Ports tab**

The Ports tab contains a read-only list of all the external ports of the system.
Parameters tab

The Parameters tab contains a list of all the parameters that are available (Figure 40).

**Figure 40 System Properties dialog, Parameters**

You can select parameters of the components within the system to be editable, make individual parameters visible, or group the parameters. You can also create system parameters that can be used as a link or alias to parameters of individual components in the system.
Label Properties dialog

Use the Label Properties dialog shown in Figure 41 to control how labels appear in the Diagram window.

Figure 41 Label Properties dialog

The dialog includes the following elements:

- **Label** — Enter the text for the label in the multi-line edit box. Carriage returns are accepted.
- **Font** — Select the font for the entire label.
- **Alignment** — The horizontal alignment enables left, center, and right alignment of the entire text. The vertical alignment enables top, center, and bottom alignment for the label. The default is horizontal and vertical center alignment.
- **Rotation** — The text can be rotated 90, 180, or 270 degrees. Default is no rotation.
- **Colors** — The user can select the colors for the text in addition to the background. Check the transparent background option for no background color.
• **Frame and Shadow** — The label has a frame border in addition to a shadow. Specify the thickness of the border by entering values from 1 to 99 (or 0 for no border). Enter the shadow amount values of 1 to 99 (or 0 for no shadow). The shadow is below and to the right of the label.

The default values are 1 for the border and 0 for the shadow.

• **Use these settings as default** — Saves the current settings for all subsequent labels.
Chapter 3

Creating a SoC Designer Component

This chapter describes different methods for creating SoC Designer components in the following sections:

• Component Overview
• Preparing a SoC Designer Component to be Generated
• Generating the component
• Modifying the generated files
• Developing components manually

Component Overview

SoC Designer components are based on the ESL API interfaces. For overview information on ESL API interfaces, see ESLAPI overview.

SoC Designer Canvas provides a component wizard that simplifies creating a new SoC Designer component. You can also create a component manually.

SoC Designer systems are hierarchical representations of hardware systems. A system can be just a single component or consist of a hierarchical structure of subsystems and elementary components. SoC Designer does not distinguish between subsystems and elementary components, they have the same representation with respect to their boundaries. Components can instantiate subcomponents and interconnect them. Components can also have their own behavior, in addition to the behavior of their subcomponents.

A SoC Designer component or system can be designed by creating a shared library, containing the following three elements:

• A component class, defining the component/system behavior
• A factory class, used to create the component/system
• An entry point for the shared library
Figure 42 shows the encapsulation of a SoC Designer component within a given shared library or DLL. A SoC Designer component can be a top-level system, a subsystem, or an elementary component.

Figure 42 The three elements of any SoC Designer component

```
lib MyModel.so

1
C-style entry point
CASInit
{
    // Create MyModel Factory
}

2
class MyModel Factory
    :public CASIFactory
MyModelFactory:~MyModelFactory()
{
    // Register Component Factory
}
MyModelFactory::createInstance()
{
    // Create MyModel object
}

3
class MyModel Factory
    :public CASIFactory
MyModel::MyModel()
{
    // Create Subcomponents
}
MyModel::communicate()
{
    // Behavior first simulation phase
}
MyModel::update()
{
    // Behavior second simulation phase
}
MyModel::getName() { return "MyModel";
```

Note: ARM recommends using the SoC Designer Canvas Component Wizard to create new components. The entry point and factory are created automatically and the component can be immediately used in SoC Designer.
Preparing a SoC Designer Component to be Generated

Use the SoC Designer component wizard to generate SoC Designer components, as described in the following sections:

- Step 1, Project name and type
- Step 2, General properties
- Step 3, Component Ports
- Step 4, Adding Parameters
- Step 5, CADI debug interface

Step 1, Project name and type

1. From the SoC Designer Canvas Tools menu, select Component Wizard…
2. Enter the name of the component and the project location in the first dialog of the wizard (see Figure 43).

Figure 43 New component dialog

3. Check the Standard SoC Designer Component option under Generate. The other options are only used for SystemC or for importing existing components that use the CASI simulation API:
   - Project Files for SystemC Import is for creating the project files (but not the component files) for an existing SystemC component.
   - SystemC Primitive Channel is for creating SystemC interconnect channels.
   - Project files for CASI Import is for importing CASI 1.1 components. See the documentation supplied with the CASI 1.1 package for additional information.

Step 2, General properties

On the Component Attributes dialog (Figure 44), edit the following component properties:
• **Component Description** — This is the text that is displayed in the Component Properties dialog for the component.

• **Component Type** — The type is used by the Component window of SoC Designer Canvas to group the components by type. For example, all components of type Bus are displayed when the **Bus** tab is clicked in the Component window.

• **Save/Restore Support** — Check this control if the component must support the save and restore feature. The code to support save and restore is automatically generated, but might require modification for special cases.

• **Load Application Files** — Check this control if the component supports loading of application files. If selected, the **File Extension** text box is enabled. Entering an extension filters the files that are displayed by the Load Application File dialog.

**Figure 44 Component attributes dialog**
Step 3, Component Ports

On the Add Port Definition dialog (Figure 45), create the ports that are part of the component.

Figure 45 Port information dialog

You can create a new port or import one by copying it from an existing component that is present in the SoC Designer Diagram window.

Click New on the wizard to display the Port Information dialog and specify the following properties:

- **Port Name** — A descriptive name for the port.
- **Port Type** — The port type must be one of the types in the drop down list.
- **Count (1-8)** — The number of ports of this type to add. This option applies to clock slave and signal ports only.
- **Block Size** — Block size (bus master ports only).
- **Memory Size** — The amount of memory associated with the port (bus master ports only).

By default, components are not clocked. To clock the component, add a Clock Slave Port port. You can add multiple clock ports if desired; refer to Connecting clock slave ports.

If you create the wrong type of port or enter the wrong values, use the Delete or Edit buttons to delete or change the port.
Importing a port

If one or more components are present in the Canvas Diagram window, you can use the Import button to import one or more ports from a component in the Diagram window:

Click Import to display the Select Port List dialog (see Figure 46).

**Figure 46 Importing a port from an existing component**

![Select Port List dialog](image)

The default behavior is to import the port and set its configuration to be identical to the configuration used for the port in the existing component.

To copy the port characteristics, but change the direction of the port, use the Invert port direction check box (for example, inverting an existing signal master creates a signal slave).

**Step 4, Adding Parameters**

On the Add Parameter Definitions dialog (Figure 47), specify the component parameters. Parameters can be used to make a component configurable without requiring recompilation. This enables creating generic components for architecture exploration.

**Figure 47 Parameter information dialog**

![Parameter information dialog](image)
Click **New** to add a new parameter and display the Parameter Information dialog used to specify the following properties:

- **Parameter Name** — This name appears in the Parameter window.
  
  *Note:* The name displayed in the **Variable** text box is the parameter name converted to a legal C-variable name. Illegal characters are replaced by underscores and a _p_ prefix is added. For example the `use cache` parameter results in a variable `p_use_cache`.

- **Value** — The default value for the parameter. A Boolean variable, for example, might have `false` as the default.

- **Type** — The data type must be one of the types in the drop down list.

- **Run-Time Parameter option** — Check this box if you require the component parameter to be modified at run time (that is, during simulation).

If you create the wrong parameters or enter the wrong values, use the **Delete** or **Edit** buttons to delete or change the parameter.

Click **OK** to add the new parameter or **Cancel** to close the Enter Parameter Information dialog without creating a parameter.

**Importing parameters**

If one or more components are present in the Diagram window, you can use the **Import** button to import a parameter from a component in the Diagram window.

After importing the parameter, use the **Edit** button to change the parameter type or default value.

**AHB or APB parameters**

If you chose **AHB Transaction Slave** or **APB Transaction Slave** as the type for a new port, the Add Parameter Definitions step of the wizard displays parameters that have been automatically generated to support the AHB or APB interface:

- **PortName_baseAddress** — This is the base address for the component.

- **PortName_addrSize** — This is the address size. If a transaction generates an address between `PortName_baseAddress` and `PortName_baseAddress + PortName_addrSize`, the component is active.
Step 5, CADI debug interface

On the Add CADI Registers and Memory Regions dialog (Figure 48), create CADI registers and add a memory interface. CADI support is optional, but highly recommended.

Figure 48 Debug information dialog

Select CADIInterface to enable the automatic generation of debug code for accessing registers and memory.

Changing the default registers

The Component Wizard generates two debug registers. To add your registers, delete the default registers or modify them as required. Select a register from the list and use the Delete or Edit buttons to delete or change the register.

Adding new debug registers

Click New to display the Register Information dialog used to specify the following properties:

- **Register Name** — The name of the register to be used in the generated code
- **Bit Width** — The width of the register. This is typically 16 or 32 bits.
- **Display Type** — The default numerical format used to display the register contents.
- **Memory Mapped** — Selecting this option maps the registers onto the memory address space of a transaction slave port of the component (this option is only available if transaction slave ports have been created).

If this option is selected, use the **Slave Port** and **Address Offset** boxes to specify the associated slave port and the offset from the start of the slave port’s memory space.

- **Description** — A brief description of the component. This is displayed in the Description box of the Parameters window.
Generating a debug interface for memory

By default, no debug code is generated for accessing memory. Select the Memory check box to generate an example CADI memory space for this component.

Note: You must modify the generated code to return additional information about the component memory. See Using the CADI interface from SoC Designer Simulator for additional information on how to implement the interface.

Generating the component

After completing the instructions in Step 5, CADI debug interface, a summary is displayed (see Figure 49). The component is ready to be generated.

Figure 49 Component summary

Click Finish to start the generation. This creates a SoC Designer Component project. The project includes classes for the component and for each of the slave ports.

As part of the project build, the following are created:

• A DLL to be used for simulation in SoC Designer (optional).
• A Linux make file that enables building this component as a shared object for Linux.

**Figure 50** Save options

You can use the radio buttons shown in *Figure 50* to select whether the component is compiled to a DLL and whether the component is added to the Component Library Repository. If the component is not compiled, you can edit the source and compile it later.

If the component has not been added to the repository, you can do this by manually editing a component configuration file or by using the **Add Component** menu option from SoC Designer Canvas.

After the component is added to the library, an icon is present in the Component window. Dragging the component to the Diagram window displays the new component and its ports (*Figure 51*).

**Figure 51** Example of a component created with the component wizard
Modifying the generated files

The Component Wizard generates C++ source files for the components and compiles them. Next, you must provide the actual functionality of the new component, such as reacting to signals and updating registers. Edit the generated C++ files and recompile them to add your own custom modifications to the created component shell.

General Modifications

To add custom modifications to the created component shell, modify the main files for the component (NewComponent.cpp and NewComponent.h, where NewComponent is the name you specified in the Component Wizard).

The .h file contains declarations for functions found in the .cpp file. The .cpp file contains the following features:

• entry point function (creates the factory class)
• component factory class (creates an instance of the component)
• component constructor and destructor functions
• component initialization functions
• component reset and cleanup functions
• communicate and update functions (for clocked components)
• parameter get and set functions (for component configuration)
• component save and restore functions (if specified in the wizard)
• port initialization functions such as init Signal Port() and init Transaction Port().

Slave Port modifications

The main files for slave ports are Name_Type.cpp and Name_Type.h, where:

• Slave ports are defined in separate source files, where Name is the name of your new slave port as specified in the Component Wizard
• Type is TS for a transaction slave or SS for a signal slave (see Step 3, Component Ports).

The .cpp files contain:

• the port constructor (a blank destructor is in the .h file)
• access functions for the signal or transaction slave. Signal slaves use driveSignal() and readSignal() functions and transaction slaves use read() and write() functions.

These functions must be updated to access the private variables in the component class.

CADI modifications

The main files that handle the CADI debug interface are NewComponent_CADI.cpp and NewComponent_CADI.h (where NewComponent is the name you specified in the Component Wizard.) These files contain code that enables the CADI debug interface to read or set memory or register values. The .cpp file contains the following features:

• structures that hold register or memory information
• NewComponent_CADI class constructor and destructor
• register or memory access functions.
These functions require updating when you manually add additional registers or memory to the component or modify a memory array. See *Using the CADI interface from SoC Designer Simulator* for more information on CADI interface functions.

**CAPI modifications**

The wizard does not generate the `NewComponent_CAPI.cpp` and `NewComponent_CAPI.h` files that provide the profiling interface. If you require profilling for your new component, you must create these files manually and modify the component code to reflect the new additions. See *Profiling streams and channels* for additional information on how to implement the interface.

**Save/Restore modifications**

If *Save/Restore Support* was selected, the Component Wizard generates basic code that can save and restore the simulation state. If you require more advanced Save/Restore functionality, see the *ARM ESL API Developer’s Guide* for additional information about the save and restore classes.

**Developing components manually**

The *ARM ESL API Developer’s Guide* provides details on the classes and data structures in the API. See that manual for instructions on writing component code that is used in a SystemC environment.

This section provides some additional information that is relevant if you are manually creating a component for use in SoC Designer.

If you require extended functionality for debugging, Save/Restore, or profiling, you must manually create (or edit) the source files and extend the functionality:

- Save and restore is described in *Saving and restoring simulation state*.
- The debug interface is described in *Using the CADI interface from SoC Designer Simulator*.
- The profiling interface is described in *Profiling streams and channels*.
- If your component is a bus master or bus slave, you can define the memory map as described in *Memory Map Overview*.

**Defining a factory class for the system**

In addition to the component itself, a factory class is required that can be used by the SoC Designer system to obtain access to the model.

The factory class must inherit from `CASIFactory`. The only functions that must be defined are:

- the constructor. This is named `MyModelFactory()` for the class in Example 1-1.
- the `createInstance()` function.

The following is an example of a Factory class:

```cpp
class MyModelFactory : public CASIFactory
{
public:
    MyModelFactory();
    CASIModule *createInstance(CASIModule *c, const string &id);
};
```
Note: The component factory code generated by the SoC Designer component wizard is sufficient in most cases. The factory class is not required if you are using the models in a pure SystemC environment.

Saving and restoring simulation state

SoC Designer supports saving state information in an .mxr file to enable resuming the simulation from the last-reached cycle count. Note that the .mxr option is only available with the Save/Restore Licensing feature.

In addition to the simulation information that is stored in an .mxs file, the .mxr file contains the state information such as register and memory contents. The state information is specific to the system configuration at the time the state was saved. It is not possible therefore to load a state file on top of a different system configuration.

The data in an .mxr file is stored as a binary stream of encoded information that is not dependent on the operating system, CPU, or byte order of the host computer. A stream written with the Windows operating system can therefore be read by a computer running Linux (assuming the models are also available on the two platforms).

Saving the state for an entire system requires that every component in that system supports the interface. ARM recommends implementing this interface for all new components.

For a component to support the advanced Save/Restore feature:

1. The component must also inherit from CASISaveRestore:

   ```cpp
class MyModel : public CASIModule, public CASISaveRestore {...
2. The `getProperty()` function must return `yes` for the `CASI_PROP_SAVE_RESTORE` property. This informs SoC Designer Canvas about the Save/Restore capability of the component.

3. The Save/Restore functionality must be initialized by calling `initStream(this)` from the component constructor as shown in the following example. If the initialization method is not called, the component state cannot be saved:

   ```cpp
   // MyModel’s constructor
   MyModel::MyModel( ... )
   {
       // Create MyComponent
       ...  

       // Call method inherited from CASISaveRestore
       // Which is required to enable saving and restoring state information
       initStream( this );
   }
```

4. The two pure virtual methods inherited from CASISaveRestore must be implemented. The following example shows a shell implementation of the two methods that implement saving and restoring component state:

   ```cpp
   // Implementation of CASISaveRestore interface to save state
   bool MyModel::saveData( CASIODataStream &data )
   {
       // return save was successful
       return true;
   }

   // Implementation of CASISaveRestore interface to restore state
   bool MyModel::restoreData( CASIODataStream &data )
   {  ```
// return restore was successful
return true;
}

The implementation shown is valid for a component that has no state information to save. The basic
implementation must be provided for the component to enable complex systems that include this
model to save state. The complex components have more extensive save and restore functions.

It is not unusual for components to have no state to save. A Fan Out component (FOUT) that
distributes a signal to several components, for example, completes its task in one cycle and has no
state to save.

Creating a shared library

The SoC Designer scheduler dynamically loads SoC Designer components. The component classes
must therefore be compiled into a shared library or DLL. Figure 52 shows the encapsulation of a SoC
Designer component within a given shared library or DLL. A SoC Designer component can be a
top-level system, a subsystem, or an elementary component.

Figure 52 Accessing the component class through a shared library

lib MyModel.so

1

C-style entry point
CASInit
{
    // Create MyModel Factory
}

2

class MyModel Factory
class MyModel Factory
    :public CASIFactory
    :public CASIFactory
MyModelFactory::MyModelFactory()
{
    // Register Component Factory
}
MyModelFactory::createInstance()
{
    // Create MyModel object
}

3
class MyModel Factory
    :public CASIFactory
MyModel::MyModel()
{
    // Create Subcomponents
}
MyModel::communicate()
{
    // Behavior first simulation phase
}
MyModel::update()
{
    // Behavior second simulation phase
}
MyModel::getName()
{
    return "MyModel";
}

The entry point

To create a shared object that can be loaded by the SoC Designer back end, an entry point must be
defined. The entry point for any SoC Designer shared object is the CASIInit() function. This function
is declared as extern "C" to facilitate linking.

The following is an entry point definition example:

extern "C" void CASIInit()
{
    new MyModelFactory();
}

The init() function creates an instance of the component factory object for the components in the
shared library. The factory then registers itself with SoC Designer.
Creating a project file with the Component Wizard

Compiling your system as a shared library is simplified if you have a project file or make file. You can use the Component Wizard to create a project file for Microsoft Visual Studio and a makefile for use by GCC:

1. From the SoC Designer Canvas Tools menu, select Component Wizard…

2. Enter the name of the component and the project location in the first step of the wizard as shown in Figure 53.

   **Figure 53 Component Wizard project file generation**

   ![Component Wizard project file generation](image)

   3. In the Generate box, select Project files for CASI Import.

   4. Click Next to proceed to the file generation step. The Step 2 dialog displays (Figure 54):

      **Figure 54 Select files to generate**

      ![Select files to generate](image)

5. Click Next to display a summary of the files you have requested.
6. Click **Finish** to generate the top-level files, makefile, and Visual Studio project file for the component.

7. Edit the makefile (or project files) and top-level files to use the code that you manually created.

**Editing an existing makefile**

The Component Wizard creates a Microsoft Visual Studio project file and a makefile. You can, however, also manually edit a makefile to match your project files. Use the makefile template provided in the `etc` directory of the SoC Designer installation directory. Copy the template to your source directory and modify the following entries:

```makefile
PACKAGE_SOURCES = COMPONENT_NAME.cpp
```

Use this makefile to build the library `component_name.so` that contains your SoC Designer component.

**Coding and compilation guidelines**

The following compilation flags must be set to successfully compile components for SoC Designer:

- **Visual Studio** (see the *SoC Designer Installation Guide* for supported versions):
  - Include directories: `${MAXSIM_HOME}/include`
  - Enable C++ Exceptions: YES
  - Runtime library: Multi-threaded DLL or Multi-threaded Debug DLL
  - Enable Run-Time Type Info (RTTI): YES
  - Library directories: `${MAXSIM_HOME}\lib\([WindowsDirectory]\)\Release`
  - Linker: Additional Dependencies: `maxsimcore.lib`
  - `/vmg: C/C++ > Command Line > Additional Options`

- **Red Hat Enterprise Linux** (see the *SoC Designer Plus Installation Guide* for supported versions). The gcc provided with SoC Designer is found at `$CARBON_SDCXX_PATH/bin`; you must use this gcc when building your models.
  - Include directory: `${MAXSIM_HOME}/include`
  - Dynamic library: `-shared`

*Note:* If you use the SoC Designer Canvas Component Wizard, the makefile (for gcc) or `.vcproj` (for Visual Studio) is generated automatically and already contains the appropriate flags.
Using your new component in SoC Designer

To use your component in the SoC Designer Canvas or SoC Designer Simulator tools, the model library must be registered in a library configuration file and the library configuration file must be included in the Model Library section of the Preferences dialog.

To register the model in a configuration file:

1. Select Add Component to Component Library from the SoC Designer Canvas Tools menu. The Model Library Repository Wizard is displayed as shown in Figure 55. Enter the name of the dll or so object in the Library field:

   Figure 55 Model Library Repository Wizard dial

   ![Model Library Repository Wizard dial](image)

2. Click Next to display the next step of the wizard as shown in Figure 56. Select the repository to hold the details of this component.

   ![Select library](image)

   *Note:* To create a new Component Library configuration file, click Browse and specify the name for the component library in a directory you specify. The conventional name for this file is component library, but you can specify a different name.

3. Select Absolute Path to Component Library if the library object is always in a specific path. Otherwise select Component Library Path is Relative to use a subdirectory specified by its relative path to the directory that contains the library configuration file.
4. Click **Next** to display a summary of the details you have entered as shown in *Figure 57*.

*Figure 57 Summary of new component*

5. If the model object you selected is not usable by SoC Designer, an error dialog is displayed as shown in *Figure 58*. Otherwise the object is added to the repository.

*Figure 58 Error dialog*
The following example shows the entry in the library configuration file for a component in the current directory:

```plaintext
component "Tracer"
{
    type = "Other";
    path = "tracer.dll";
}
```

The component type depends on the component type defined in your model. If none is defined, the type `Other` is assumed. The path points to a shared library object (Linux) or a DLL (Windows).

Note: The name of the model must be equivalent to the name that is returned by the component factory and by the `getName()` function of the component. You can also open a library configuration file with a text editor and enter the component name, type and path directly into the file.

Make the model available in the Components window by selecting Manage Model Library from the Tools menu to display the Preferences dialog. Add the configuration file by performing one of the following actions:

- Use the Add button to add the library configuration file to the list Additional Library Configuration Files. Use this option if the component must be available to all systems.
- Select Use the directory that contains the SoC Designer project file. Use this option if the component is only used by a single project.
Chapter 4

Working With Components

This section describes components and how to use them in a system model. It has the following sections:

- **Component Description**
- **Adding a component to the Diagram window**
- **Displaying component properties using the component info box**
- **Component context menu**
- **Port Types, Movement, and Connection**
- **Editing memory address spaces for bus-master ports**
- **Making connections between components**
- **Working With External ports**
- **Using SystemC components**
- **Using components based on the ESL APIs**
- **Optimizing components by profiling simulation performance**

**Component Description**

*Figure 59* shows a component as it is displayed in the Diagram window.

*Figure 59 Component description*

The title bar shows the instance name of the component followed by the component (model) name in parentheses. The title bar also shows the type of the component in italics on the right hand side. Component type is not shown by default.
Figure 60 shows an unselected and a selected component. The selected version has a different colored title bar area and resize control points.

**Figure 60 Component and Selected Component**

Adding a component to the Diagram window

To add a component to the Diagram window:

1. Perform one of the following actions:
   - Drag a component from the Component Window to the Diagram window.
   - Copy and paste an existing component in the Diagram window.
   - Click the Comp button.
   - Right-click on the Diagram window and select Add Component from the context menu.
   - Select Add Component from the Insert menu.

The last three methods display the Select Component dialog (Figure 61).

**Figure 61 Select Component dialog**

2. Enter the instance name of the component in the Instance Name field. If no name is entered, SoC Designer creates a unique name for the component.

3. Select the component you want to add and click OK.
4. The Select Component dialog closes, and if the cursor is inside the Diagram window, the mouse cursor shows a representation of the component. Place the component by clicking the mouse button. SoC Designer does not allow you to place a component on top of another object within the system.

**Displaying component properties using the component info box**

When the mouse hovers over the i box in the top right hand corner of a component, the cursor changes to a question mark (*Figure 62*). This simplifies finding the info box in large diagrams at small zoom factors.

*Figure 62 i box focus*

Clicking on the info box displays the main properties of the component (*Figure 63*).

*Figure 63 Component info box*

The information in the info box is displayed with the same font size regardless of the zoom factor of the diagram.
Component context menu

When a component is on the Canvas, right-clicking on it displays a context menu with the following elements:

- **Cut, Copy, and Delete** — These editing entries function the same as the corresponding entry from the **Edit** menu.

- **Rename…** — Each object in the system must have a unique name identifier. This option displays a dialog for you to enter a name for the object. The system checks to make sure that the name of the object is unique and notifies you if this is not the case.

- **Enable/Disable Component** — Enables or disables the selected component.

- **Open Sub-System…** — If the selected component is a hierarchical component and the system description is available then this option opens the sub-system in another tab of SoC Designer Canvas.

- **Unhide All Ports** — Shows all ports that have previously been hidden. This might result in overlapping port symbols if there is not enough space for displaying all of the hidden ports. SoC Designer does not automatically increase the size of the component if the unhidden ports do not fit into the boundaries of the component.

- **Disable All Unconnected Ports** — Disables every unconnected ports for the component. If **Unhide All Ports** is selected, the port shows as gray.

- **Hide All Disabled Ports** — Hides disabled ports in the component. You can override this by selecting **Unhide All Ports**.

- **Reset to Factory Layout** — Changes the layout of the currently selected component to the factory layout. This results in all slave ports being on the left hand side and all master ports being on the right hand side in an alphabetical order.

- **Save As Default Layout** — Saves the current component view as the default. This configuration is used until a new default layout is saved or the **Reset to Factory Layout** option is selected.

- **Component Display Options…** — The following options are available for the display of components: **Display Title Bar**, **View Model Name**, **View Type**, **View Logo**. The model name and type options affect how the title bar portion of the component appears. Figure 64 shows a component with all display options enabled.

Figure 64 All component display options on

![Component display options](image-url)
Figure 65 shows the component with all display options disabled.

Figure 65 All component display options off

- **Replace Selected Components…** — Enables replacing the selected object(s) with another similar object. The tool attempts to reconnect all connections to the same ports of the new component. If the same port does not exist, the connection is deleted. The size of the component is preserved but the name is not.

  Note: Use this option with care. It might adversely affect the design of your system.

- **Lock Version/Variant** — Selecting this option for a component enforces the currently selected version to be loaded first. A component with a locked version is shown with an asterisk (*) next to the i in the info field.

- **Unlock Version/Variant** — Selecting this option for a component enables any version to be loaded. Whichever version is found first is loaded.

- **Edit Parameters…** — Displays a dialog for viewing and editing the object parameters (see Editing Parameters).

**Port Types, Movement, and Connection**

Figure 66 shows a component that has all six port types.

Figure 66 Component with all port types

Master ports always point out of the component and slave ports always point into the component. You can move master and slave ports around the edge of the component as illustrated in Figure 67.

Figure 67 Master and Slave Ports along the component edges
Transaction connections are represented with a thick line while signal and clock connections are displayed with thin lines as shown in Figure 68. Transaction port outputs must be connected to transaction port inputs and signal port outputs must be connected to signal port inputs.

**Figure 68 Connection Types**

Connecting clock slave ports

This section describes the ways in which a clock slave port of a component can be connected.

If a clock slave port of a component is not connected to a clock driver component explicitly, then the component is implicitly connected to the master clock. To not connect a component to a clock, disable the clock slave port of the component.

Note: If a component has more than one clock port, then clk_in should be left unconnected (i.e., implicitly connected to the master clock). If a component’s only clock port is clk_in, then you may connect clk_in to a CDIV.

Connection to master clock [default]

Connecting to the master clock means that the component is clocked at the rate used by the master clock scheduler of SoC Designer Simulator. In this state, the clock port (clk_in) is displayed with a small black triangle within it (Figure 69). This is the default.

**Figure 69 Connection to Master Clock**
Connection to no clock

Disabling the clock slave port means that the component is not clocked. The component does not receive clock ticks during simulation. Notice that in Figure 70, the clock port name is grayed out, indicating that it is disabled.

**Figure 70 Connection to No Clock**

Explicit Clock Connection

Clock slave ports can also be connected explicitly. Explicitly-connected clock slave ports must be connected to a clock master port of another component (see Figure 71).

**Figure 71 Normal Clock Connection**

Editing memory address spaces for bus-master ports

You can edit the memory address space for a bus-master port from SoC Designer Canvas. Click on the **MemMap** button in the toolbar (or select **Memory Map Editor** from the **Edit** menu) to display the Memory Map Editor dialog. See **Memory Map Editor dialogs** for more information on using the dialogs.

Making connections between components

**Figure 72** shows a connection being drawn between ports. To connect components this way:

1. Click the **Connect** button on the SoC Designer toolbar to enter connection mode.
2. Use the cursor to connect component ports.

**Figure 72 Making a new connection**

![Diagram showing new connection between AHB Bus (MxAHB2) components](image)

**Working With External ports**

External ports are used for hierarchical systems. A system that is designed to be instantiated within a higher-level system must have external ports. A top-level system or a flat system typically does not have external ports.

*Figure 73* shows the external port types.

**Figure 73 External port types**

![Diagram illustrating external port types](image)

External port types may be SystemC generic and transaction ports (solid black), signal ports (shaded), or clock ports (white). Master ports are identified by the addition of the triangular port symbol within the rectangle portion of the external port symbol.
If a system is instantiated as a hierarchical component within another system, each external port is displayed as a port symbol inside the parent system component as shown in Figure 74.

![Figure 74 External ports](image)

If a system like the one above is added to SoC Designer Component Library, it appears in the Component Window with a system icon and the name the system was saved under. Figure 75 shows the above system saved as BridgeSys.mxp.

![Figure 75 Saved subsystem in Component Window](image)

If this system is dragged over the diagram within another system, it looks like the image shown in Figure 76. Any port names result from the names of the corresponding external ports in the underlying system.

![Figure 76 Bridge component in a system](image)

**Adding an external port to a system**

To add an external port to a system that matches a selected component:

1. Select a port on a component.
2. Right-click to display the context menu.
3. Select **Create External Port**. An external port with the same type, name, and orientation is created in the Diagram window.
4. Go into connect mode (press the **Connect** button or select **Connect Ports Mode** from the **Edit** menu) and connect the new external port to the component port.

To add a new port to a system:

1. Click on the **Port** button, select **Add External Port** from the **Insert** menu, or right-click in the Diagram window and select **Add External Port** from the context menu.
The External Port dialog is displayed.

Figure 77 External port dialog

2. Select the type of port (Clock Port for example) from the radio buttons at the left of the dialog.

3. Select the direction for the port (Master Port for example) from the radio buttons at the right of the dialog.

4. Enter a name for the port in the Instance Name text box.

5. Click OK to create the port and display it in the Diagram window.

6. If the created port is facing in the wrong direction (for example, toward the right instead of toward the left), right-click on the external port and select Mirror External Port from the context menu.

For example, Figure 78 is the original slave master port and Figure 79 is the mirrored version.

Figure 78 Original external port

Figure 79 Mirrored external port

7. To change the name of an external port, right-click on the port and select Rename from the context menu. Enter the new name in the Rename Object dialog that is displayed.

8. Go into connect mode and connect the new external port to a component in the system.

Note: You can also use the normal cut, copy, and paste actions to duplicate ports.
Using SystemC components

SystemC components can be directly used in a SoC Designer system and do not require the creation of special wrapper components. Drag a SystemC component from the Component Window to the Diagram, as is done for any other component.

Note that, although SystemC components do not require wrappers, they do require the use of a channel component to interconnect their ports. If you use a SystemC component with sc_in/sc_out type ports, you must also drag a SystemC channel from the Component Window and place it between the SystemC ports to be connected. If multiple ports are used, multiple channel components are also required.

See the SoC Designer SystemC Linking Guide for more information on using SystemC components with SoC Designer.

Using components based on the ESL APIs

SoC Designer supports components built with the ESL APIs (composed of CADI, CAPI, and CASI APIs). For more information, refer to the ARM ESL API Developer’s Guide.

Optimizing components by profiling simulation performance

On the Linux platform, SoC Designer ships with gperftools for profiling. This tool is used for profiling the simulation itself, not for profiling the simulated system. This is useful for identifying models in the simulation that could be optimized to improve simulation performance.

For instructions on using gperftools, refer to the readme using_gperftools.txt, located in the directory $(MAXSIM_HOME)/deps/gperftools.

Complete details on the gperftools suite can be found at the gperftools project site: https://github.com/gperftools/gperftools.
Creating new parameters

To create a new parameter for the system:

1. Click the **New Parameter** button in the **Parameters** section of the dialog box. This invokes the New System Parameter dialog box (*Figure 80*).

*Figure 80 New System Parameter Dialog*  
![New System Parameter Dialog](image)

2. Enter the name of the system parameter in the **Name** field.

3. Enter the default value for the parameter. This default value is overridden if an individual component sets the parameter differently than the default value.

4. From the **Links** list, select the parameters that this system parameter sets as default.

A system parameter can link to multiple component parameters. If you change the value of the system parameter, the linked component parameters updates automatically.
Editing Parameters

Use the Edit Parameters dialog for viewing component object properties and ports, and for modifying object parameter information. Refer to the model guide for a particular component for parameter descriptions, available settings, and defaults.

To open this dialog:

1. Select an object from the Canvas or System Configuration Window
2. Select Object > Edit Parameters… from the main toolbar. The Edit Parameters dialog opens (Figure 81).

![Figure 81 Edit Parameters dialog]

The dialog contains three panels:

- **Properties panel** — Contains all the properties of the component object. Depending on the state of the object, some fields might be read-only. Reported properties include:
  - component library name and location
  - configuration file where the component library has been specified
  - model type, version, and description
  - file extension of support object files (if any)
  - supported debuggers
  - name of the company who provided the component intellectual property (IP)
• Port List panel — Lists the name and type of all the component object’s ports.
• Parameters — Contains all the editable parameters of the component object.

To edit a parameter:

1. Double-click in the Value column of the parameter to be edited
2. Enter the new value.
3. Press Enter or click in the background of the list box to accept the change.

If a parameter value has been modified, the button to the right of the value field changes to show the Reset icon. Clicking this button resets the value to the factory-set default value.

Parameters that are editable at run-time in the SoC Designer Simulator are represented by a checkmark in the fourth column.

The Documentation button at the bottom of the Edit Parameters dialog displays detailed documentation for the component, if this link was defined by the developer of that component. See Configuration File Overview and Content for information about providing this link in components.
Chapter 6

Working With Memory Maps

Memory Map Overview

This section describes the dialogs associated with editing the Memory Map for components with bus master ports that support the CASIMMI API.

Note that for slave components, the Memory Map Editor dialog only supports editing of address regions of slave components that:

- Return non-NULL values from the getMappingConstraints() function
- Return casiVersion 6 (or higher) from the transaction port properties casiVersion field
- Return an empty string when the bus master port calls the component’s setAddrRegion() for the last address region.

This is the default for new components. If you modify a component that has an older slave to enable support for the Memory Map Editor, you must implement these features.

For more advanced, in-depth information about the Memory Map Interface, see CASI Memory Map Interface overview.

Hierarchical Systems and Memory Map Editor

For SoC Designer Canvas, the memory map is always stored in the .mxp file that contains the slaves attached to that bus:

- **top level m xp file** — Contains the memory map information for any top-level slaves connected to a bus at top level.
- **lower level m xp file** — Contains the memory map information for lower level slaves. For a bus at the lower level, the lower-level .mxp file also contains the memory map information for any slaves on the same level as the bus.

For SoC Designer Simulator, the Memory Map Editor shows the union of all the regions for all slaves (from top and lower levels).
Memory Map Editor dialogs

You can view and edit the memory address spaces for bus-type components from SoC Designer Canvas. To view a Memory Map Editor (MME) dialog:

- Select Memory Map Editor from the Edit menu, to open a modal Memory Map Editor dialog. Or,
- To display the non-modal MME dialog, click the MemMap tab that is in the System Configuration Window. By default, the System Configuration Window is docked and located to the right of the main SoC Designer Canvas window (see Figure 83). You can move this non-modal window by dragging it, after toggling the docking status (see Figure 82).

To toggle docking status on or off, right-click on the System Configuration Window and select Undock Tools Window (displayed when the window is docked) or Dock Tools Window (displayed when the window is undocked).

Note that when the System Configuration Window is undocked, it displays within the Tools Window dialog box (Figure 82).

Figure 82 Memory Map Editor, non-modal, being docked
The Memory Map Editor dialogs display the memory address regions for slave components of the bus master. The non-modal Memory Map Editor dialog in the MemMap tab (see Figure 83) is simpler and has fewer functions than the modal dialog that is invoked from the Edit menu.

Figure 83 Memory Map Editor, non-modal, docked, for a bus master port

The memory address space can also be viewed or edited from SoC Designer Simulator if a memory map has been defined for the system. There is a basic conversion utility within the Simulator Memory Map Editor that enables generation of a new .mxp file with memory map information.

Note: The minimum and maximum number of available maps for the component is determined by the slave component and is set by the designer of the component. Depending on the functionality of the component, however, the map to use for a simulation can be selected and configured at run time.

To view the memory address regions for slave components of the bus master with the non-modal Memory Map Editor, you must select the bus master component in the SoC Designer Canvas Window.

The non-modal Memory Map Editor dialog has the following controls:

- **Memory Space** — Select the bus component from the drop-down list. The number of memory spaces depends on the number of CASIMMI interfaces defined in the constructor of the bus components in the system.

- **Map** — Select the map that corresponds to the memory space.

  Note: There can be one or more maps available. The number of maps is determined by the designer of the component with the bus master and cannot be modified by the system creator. Some bus masters might have multiple memory regions to, for example, use different maps in reset or normal operating states.

- **Address Columns** — The non-modal Memory Map Editor dialog can modify values in the **Start Addr** and **End Address** columns of the display box. To modify an address value, simply double-click on it. The value to be changed highlights, indicating you can type in a new value (see Figure 84).
The display box lists all of the address regions that are connected to the selected bus master.

*Note:* There can be one or more address regions for each component slave. The minimum and maximum number of regions for a slave is determined by the designer of the component.

**Figure 84 Memory Map Editor, non-modal, modifying an address**

![Memory Map Editor](image)

To use additional memory map editing features, display the modal Memory Map Editor dialog (see Figure 85) by selecting **Edit > Memory Map Editor**.

**Figure 85 Memory Map Editor for a bus master port**

![Memory Map Editor](image)

The Memory Map Editor dialog has the following controls:

- **Memory Space** — Select the bus component from the drop-down list. The number of memory spaces depends on the number of CASIMMI interfaces defined in the constructor of the bus components in the system.

- **Map** — Select the map that corresponds to the memory space.

  *Note:* There can be one or more maps available. The number of maps is determined by the designer of the component with the bus master and cannot be modified by the system creator. Some bus masters might have multiple memory regions to, for example, use different maps in reset or normal operating states.

- **Master Ports** — Represents the name of the bus master port that controls this address region. There may be multiple bus master ports defined for a single memory spaces depending on the value of the `masterPortList` parameter:
registerInterface(CASIMMI * interface,CASIModuleIF *component,
        string name,uint32_t numMasterPorts,
        CASIPortIF **masterPortList);

Multiple bus master ports might, for example, be used in interconnect matrices such as AXI.

There is typically only one master port for an address region.

- **Address Regions** — The box lists all of the **Address regions** that are connected to the selected bus master.

  *Note: There can be one or more **Address regions** for each component slave. The minimum and maximum number of regions for a slave is determined by the designer of the component.*

- **Add** — Click this button to add a new memory region if the component slave supports additional regions. (See *Adding an Address Region.*)

- **Edit** — Select an address region from the list and click this button to edit an existing region. (See *Editing an Address Region.*) You can also display the edit dialog by double-clicking on an address region in the list.

- **Remove** — Select an address region from the list and click this button to delete the region. If the component slave does not support having fewer defined regions, an error message is displayed and the selected region is not deleted.

- **Copy all** — Click to copy the current **Memory Space** and **Map**. All the regions in the current map is copied and is available for a subsequent paste

- **Paste** — Click to paste **Address regions** information saved by a previous **Copy all** into the displayed **Memory Space** and **Map** display. The paste removes all existing memory regions from the current map, before pasting the previously copied items into this memory map.

- **Export** — Click to save **Address regions** information for the specified **Memory Space** and **Map** as a text file, formatted to match the dialog’s **Address regions** display box.

  *Note: No import for such files is currently available.*

- **Clear All** — Deletes all MME related information from the .mxp file. The next time the MME dialog is opened, default address region values is created, based on the constraints for the connected slave ports. The default **Address regions** values are represented by a * next to the name in the **Memory Space** field.
Adding an Address Region

To use the Memory Map Editor to add a new region to the map:

1. Select an existing region in the current map. The add dialog enables you to add a new region to the same slave component as the selected one.

2. Click Add to display the Add Address Region dialog (Figure 86).

Figure 86 Add Address Region dialog

If the error message shown in Figure 87 is displayed (or the Add button is disabled), the constraints for the component prevent adding a new region.

Figure 87 Add Address Region Error message

3. Enter the new values:
   - Name — Enter the name for the new region.
   - Start — Enter the start address for the new region.
   - End/Size — Enter the address of the end of the region or the region size. The other field is automatically calculated based on the value entered.

   You can use the underline character “_” to separate groups of digits. For example, 0xabcd_1234 is the same as 0xabcdef1234.
4. Click **OK** to save the new region with the specified values.

Changes to the address regions are checked against the constraints of that slave. If there are conflicts with the constraints, an error dialog is displayed and the changes are rejected.

**Editing an Address Region**

To use the Memory Map Editor to edit the values for an existing region:

1. Select a region from the list in the from the Memory Map Editor dialog.

2. Click **Edit** to display the Edit Address Region dialog (Figure 88).

**Figure 88 Edit Address Region dialog**

![Edit Address Region dialog]

Enter the new values:

- **Start** — Enter the start address for the region.
- **End/Size** — Enter the end of the region or the region size. The other field is automatically calculated based on the value entered.

You can use the underline character “_” to separate groups of digits. For example, 0xabcd_1234 is the same as 0xabcd1234.

- **Name** — Enter the name for the region.

**Note:** The constraints from the slave port and the slave component parameters are displayed in the text box at the bottom of the dialog. This information cannot be changed from the dialog, but you can copy the slave address regions (parameters in the slave component) and paste them into the **Address Region** text boxes.
3. Click **Ok** to save the region with the specified values.

Any changes to the address regions are checked against the constraints of that slave. If there are conflicts with the constraints, an error dialog like the one in *Figure 89* is displayed and the changes are rejected.

*Figure 89 Edit Address Region Error message*

![Edit Address Region Error message]

Minimum Region Size does not satisfy the Memory Constraint Value
Chapter 7

SoC Designer Simulator Reference

This chapter provides a reference for SoC Designer Simulator. It contains the following sections:

• Starting SoC Designer Simulator
• Application window
• Tools windows
• Preferences dialog
• Memory Map Editor
• Breakpoint windows
• Monitor windows
• Trace Manager and the Waveform Viewer windows
• Profiling windows
• Software analysis
• CheckPoint Manager and Saving Checkpoints

Starting SoC Designer Simulator

From within SoC Designer Canvas, you can start SoC Designer Simulator by pressing F5. SoC Designer Simulator opens with a simulation model of the system that was being designed in SoC Designer Canvas.

You can also invoke SoC Designer Simulator directly to load and simulate an existing system (see Starting on Windows and Starting on Linux).

Starting on Windows

There are several ways to invoke SoC Designer Simulator on Windows:

• From the Windows Start menu, navigate to Carbon > Carbon SoC Designer > SoC Designer Simulator.
• After setting up file associations, double-click an .mxs, .mxr, or .mxe file. (The .mxr extension is only available with the Save/Restore Licensing feature.)
• After setting the MAXSIM_HOME environment variable, select Start > Run and enter: %MAXSIM_HOME%\Bin\Release\SDsim.bat
Starting on Linux

On Linux platforms, invoke SoC Designer Simulator from the console with the following command:

```bash
> sdsim options filename
```

*filename* opens an `.mxp` or `.mxe` system, an `.mxs` or `.mxr` simulation, or an `.mxscr` script file. Refer to the following section for a complete list of options. (The `.mxr` extension is only available with the Save/Restore Licensing feature.)

Command line options

Table 5 describes the command line options that SoC Designer Simulator supports.

<table>
<thead>
<tr>
<th>Option</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
</table>
| filename, loaded with options | sdsim <options> root_filename.mxe | Name of file to load; it must be one of the following file types: `.mxc / .mxe / .mxp / .mxr / .mxs / .mxscr`
  
  *Note: The SoC Designer Runtime Only Environment does not support loading `.mxp` or `.mxr` files.* |
| -dbg (Linux only.)   | sdsim -dbg          | Loads the release_with_debug component libraries. Optionally, use with the `-debugger` option to simultaneously specify the debugger; otherwise, the system prompts you for the debugger to be attached. |
| -debugger= (Linux only.) | sdsim -dbg -debugger=none | In conjunction with `sdsim -dbg`, specifies a debugger on the command line or specifies using the debug version without a debugger. |
| -G                   | sdsim -G filename.mxp | Loads the release_with_debug component libraries. |
| -h                   | sdsim -h / --help   | Prints the command line help with a list of options. |
| -I / --runuiscript filename | sdsim -I / --runuiscript | Causes Simulator to enter batch mode and execute the supplied script file in GUI mode. The GUI is under control of a script, but the program remains in batch mode.
  
  *Note: You must run this command in the path where the `.mxp/.mxs` system resides.* |
| -v                   | sdsim -v / --version | Prints the version info of the tool without starting it. |
| -i or --script filename.mxscr | sdsim --script myscript.mxscr | Starts Simulator and executes it in batch mode.
  
  *Note: You cannot combine these options with command-line options like `-s 1000 that control simulation execution.* |
| -s x                 | sdsim mySys.mx$s -s 1000 | Runs the simulation for the specified number of cycles. SoC Designer Simulator starts in batch mode (no GUI) and all messages are directed to stdout.
  
  *Note: You cannot combine these command line options with the `--script` option that specifies a script file.* |
### Table 5 Command line options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-r)</td>
<td>sdsim mySys.mxs -r</td>
<td>Has the same effect as <code>-s</code>, except that the simulation is run indefinitely.</td>
</tr>
<tr>
<td>(-l) or <code>--logfile</code> f</td>
<td>sdsim -l logfile.txt</td>
<td>Causes all output to stdout and stderr to be redirected to that file. This option is most useful on Windows where output redirection is not possible.</td>
</tr>
<tr>
<td>(-t) or <code>--statefile</code> filename</td>
<td>sdsim mxSys.mxs -r -t statefile.bin</td>
<td>Causes Simulator to save the simulation state (.mxr) file to the name specified, overwriting any existing file with the same name. The <code>.mxr</code> file ending is added if omitted.</td>
</tr>
<tr>
<td>(-b) or <code>--maxlib</code> filename</td>
<td>sdsim mxSys.mxs --maxlib mylib.conf</td>
<td>Starts Simulator with a specific component configuration file loaded.</td>
</tr>
<tr>
<td>(-n) or <code>--nomaxlib</code></td>
<td>sdsim --nomaxlib</td>
<td>Starts Simulator with no component configuration files loaded.</td>
</tr>
<tr>
<td>(-e) or <code>--silent</code></td>
<td>sdsim --script myscript.mxs --silent -l logfile.txt</td>
<td>Directs Simulator not to open a console terminal window, but run silently instead. This can be used, for example, in combination with the <code>-l logfile</code> command line option to run batches of multiple simulations.</td>
</tr>
<tr>
<td>(--display new_display)</td>
<td>sdsim --display $DISPLAY</td>
<td>Sets the X display. The default is <code>$DISPLAY</code>.</td>
</tr>
<tr>
<td>(--geometry new_geometry)</td>
<td>sdsim --geometry new_geometry</td>
<td>Sets the client geometry of the main window.</td>
</tr>
<tr>
<td>(--font new_font)</td>
<td>sdsim --font arial</td>
<td>Sets the application font.</td>
</tr>
<tr>
<td>(--background color)</td>
<td>sdsim --background 8000</td>
<td>Sets the default background color and an application palette. Light and dark shades are calculated based on the new palette.</td>
</tr>
<tr>
<td>(--foreground color)</td>
<td>sdsim --foreground 8000</td>
<td>Sets the default foreground color.</td>
</tr>
<tr>
<td>(--button color)</td>
<td>sdsim --button 8000</td>
<td>Sets the default button color.</td>
</tr>
<tr>
<td>(--visual Truecolor)</td>
<td>sdsim --visual Truecolor</td>
<td>Forces the application to use TrueColor on an eight-bit display.</td>
</tr>
</tbody>
</table>
Opening a Simulation

If you invoke SoC Designer Simulator from SoC Designer Canvas, the selected system is automatically loaded. To simulate a different system (or if SoC Designer Simulator was invoked directly):

1. Select **File > Open**, select the system file (*Figure 90*), and click **Open**.

*Figure 90 Open simulation*
Some systems (for example, the ARM926EJ-S system example) contain components that require an application file to be loaded for the core. If this is the case, the Select Application Files dialog is displayed (Figure 91).

**Figure 91 Select Application Files dialog**

![Select Application Files dialog](image)

2. Select a component from the list and click **Select File** to launch a standard file-browsing dialog to select the input file for the core (Figure 92).

**Figure 92 Selecting the input file**

![Select The Application Code](image)
Saving a simulation: mxs, mxr, and mxc files

You can save a loaded simulation at any time.

If you load or create a simulation and run it for $x$ number of cycles, you can then save the simulation as an .mxr or .mxc file. If you reopen the file, the simulation is in exactly the same state as when you saved it. For example, if you had previously run it for 1000 cycles, then after loading, it indicates that it has already run 1000 cycles.

1. In SoCD Simulator, select File > Save or File > Save As... The Save As dialog displays (Figure 93).

![Figure 93 Save As dialog](image)

2. Select the type of simulation file to save. If the simulation was created from a SoC Designer project file (.mxp), the following file types are available:

   - **Simulator Configuration (.mxs file)** — This is the default file type, which stores information about the simulation and system configuration, including any attached probes or parameters you modified. It does not save the state of the simulation or models.

   - **Save/Restore Checkpoint (.mxr file)** — Available only with the Save/Restore Licensing feature. The .mxr file stores the same details as the .mxs file as well as a complete binary snapshot of all model state. This state information enables you to restore the simulation at exactly the same cycle count as when the simulation was stopped. Not all components support this advanced feature.

     The .mxr file stores state information, but discards previously-recorded profile or monitor information. Any Profiling data dialogs that are displayed are empty. Record new profile information to the windows by continuing to run the system.

     To save collected profiling information for later analysis, use the Export Raw Data menu option on the Profiling data dialog context menu or the Dump to CSV checkbox on the Profiling Manager window. You can not reload previously-saved data into a Profiling data dialog.

   - **Swap & Play Checkpoint (.mxc file)** — Available only with the Swap & Play Licensing feature. The .mxc file is a checkpoint that stores the state of all the components in a system at a point in a simulation (architectural state of abstract models and complete binary state of other models).

     Save .mxc files with the Save and Restore View command, or manage them with the CheckPoint Manager. Note that the CheckPoint Manager is intended for use between Fast Models and Cycle Accurate systems. You can save .mxc files in a Cycle Accurate system and restore them into that Cycle Accurate system, or save them in a Fast Model system and restore them into the corresponding Cycle Accurate system. The usage between Fast Models and Cycle Accurate systems is referred to as Swap & Play; refer to the SoC Designer Fast Models System Creator User Guide for more information.
Note: When you save a Swap & Play checkpoint (.mxc file), a second file is automatically created, which contains all memory data for that checkpoint. This file has the extension .mem and is stored in the same location as the .mxc file. If you copy or move the .mxc file, ensure that you also copy or move its associated .mem file. When you load a checkpoint, its associated .mem file is loaded automatically provided the two files are in the same location.

Note: Saving with state works properly only if all of the components in your system implement the Save State feature. If not, then the file can only be saved as an .mxs file.

Note: .mxs, .mxc, and .mxr files contain the path to the original .mxp file that contains the system. To use an .mxc, .mxs or .mxr file, the original .mxp file must be present.

Click the **Save** button to display the Save Simulation As dialog (Figure 94).

**Figure 94 Select Location**
Application window

After starting SoC Designer Simulator, the application window shown in Figure 95 is displayed.

The application window has the following sections:

- **Toolbar** — The Toolbar (see Main menu and Toolbar commands) provides a fast, intuitive way to perform common actions on the simulation.
- **Cycle counter** — The cycle counter shows how many cycles have elapsed in the current simulation run.
- **Diagram window** — This window shows a graphical view of the system under simulation. By selecting components or connections in this window, it is possible to set breakpoints, add monitors, perform waveform tracing, and collect profile information (see Diagram window).

Only one simulation at a time can be loaded, but you can open a sub-system of the simulation (if it contains any). A new subsystem appears in a new tab, at the bottom of the diagram window.

- **Tool windows** — The System Configuration and Parameters windows display information about the system and the selected component (see Tools windows).
- **Output window** — The Output window displays log messages from the simulation (see Output window).
- **Command line** — The command line is used to enter commands as a text string. (See Command Line interface.)
• **Task bar and status bar** — The task bar and status bar display additional information about the system. See *Task bar* and *Status bar*.

During a typical simulation run, several additional windows might be displayed:

• **Breakpoint Manager** — Once a simulation is started, it runs until stopped by the user or until a breakpoint is reached. The Breakpoint Manager window displays all breakpoints that have been set in the system and controls whether or not they are enabled. (See *Breakpoint windows*.)

• **CheckPoint Manager** — This option is only available with the Swap&Play Licensing feature.

  The Checkpoint Manager window displays the checkpoints that have been created and allows you to restore those checkpoints, spawn simulation, or save checkpoints as .mxc files.

  **Note:** The CheckPoint Manager is intended for use between Fast Models and Cycle Accurate systems.

For details about .mxc files, refer to *Saving a simulation: mxs, mxr, and mxc files*.

For details about Fast Models, refer to the *SoC Designer Fast Models System Creator User Guide*.

• **Trace Manager** — The Trace Manager window controls whether activity on a connection or bus is logged. The recorded information can be displayed in a Waveform Viewer window. (See *Trace Manager and the Waveform Viewer windows*.)

• **Profiling Manager** — The Profiling Manager window shows all system objects that can record profile information and controls whether a profile log file is created. The recorded information can be displayed in a profiling window. (See *Profiling windows*.)

• **Probes and monitors** — Connecting a monitor probe to a component or connection displays information about the current state of the object. The Monitor window can be changed to history view to show previous activity on the object. The following types of monitor probes are available:
  
  • Connection monitors (transaction, signal, or bus)
  • Register or memory monitors (for example, for a cache)
  • Disassembly monitors (for example, for a CPU component)
  • Communication probes are used to test interfaces.

  For more information, see *Monitor windows*.

• **Debuggers** — You can connect an external debugger to enable source-code level debugging of an application running on the simulated system. This is called Target debugging. See *Component-level Debugging*.

  You can also connect an external debugger to enable source-code level debugging of a component design. This is called host-level debugging. See *Debugging user-defined components*.

### Diagram window

The Diagram window contains a graphical representation of the system model. The following sections describe the actions that can be controlled from the window:

• *Context menu for a component in the Diagram window*
• *Context menu for connections in the Diagram window*
• *Context menu for Diagram window*
• *Diagram window appearance*
Context menu for a component in the Diagram window

Select a component in the system and then right-click to display the context menu for the component. The menu contains the following entries:

- **View Subsystem** — Opens the components in the subsystem and displays them in the Diagram window. (Disabled if the component is not a system.)
- **View Registers** — Displays the Registers window for this component. (Disabled if the component does not contain registers.)
- **View Memory** — Displays the Memory window for this component. (Disabled if the component does not contain memory.)
- **View Disassembly** — Displays the Disassembly window for this component. (Disabled if the component is not a processor.)
- **View Dependency Graph** — Displays dependencies for components that use static scheduling.
- **Profiling** — This entry contains a submenu of streams that can be enabled for profiling for this component. If the CAPI interface is not implemented for the component, this entry is not present.
- **Launch Debugger** — Launches the debugger for the processor component. The debugger to launch is determined by the type of component. If the component is not a processor, this entry is not present.
- **Attach Host Level Debugger (Visual C++)** — Launches the Host level debugger for that platform: Microsoft Visual C++ for Windows, and gdb/ddd for Linux. This selection also loads design code for the component.
- **Edit Parameters** — Displays the Edit Parameters dialog for the component.

For more information on manipulating components, see *The Object menu* and *The Debug menu*.

Context menu for connections in the Diagram window

Select a connection between components in the system and then right-click to display the context menu for connections.

- Insert/Remove Breakpoint
- Enable/Disable Breakpoint (disabled if no breakpoint has been set)
- Edit Breakpoint Properties (disabled if no breakpoint has been set)
- Insert/Remove Monitor
- Enable/Disable Tracer
- Profiler (has two submenu components, to Enable and to Display).

For more information on manipulating connections, see *The Object menu*, *Breakpoint windows*, and *Monitor windows*.

*Note:* Use the View menu to hide or display connections between components (see The View menu).

Context menu for Diagram window

Right-click on a blank portion of the Diagram window to display its context menu:

- **Remove All Connection Breakpoints** — Select this entry to remove all breakpoints on signals, registers, memory, and code.
- **Remove All Probes** — Select this entry to remove all monitor, communication, and tracer probes. Any windows relating to the probes close.
• **Zoom In, Zoom Out, Zoom to Fit, and Zoom to 100%** — Select this entry to zoom the system model shown in the Diagram window.

• **View Parent System** — Select this entry to return to the parent system after you have finished examining a component within a subsystem. If you are not currently viewing a subsystem, this entry is disabled.

**Diagram window appearance**

Use the **View** menu to control the appearance of the Diagram window. From this menu you can hide connections, set grid options, and zoom the view of the model in the window.

Select **View > Grid Options** to display the Grid Settings dialog shown in *Figure 96.*

**Figure 96 Grid Options dialog**

The dialog has the following controls:

• **Show Grid [default = off]**: — Show the grid in the diagram with the specific options.

• **Snap to Grid [default = on]**: — Moving or resizing objects snap the object to the grid boundary.

• **Grid Horizontal Spacing / Grid Vertical Spacing [default = 10]**: — Set the horizontal/vertical distance (in pixels) between grid points to use if the window is zoomed to 100%.

• **Grid Point Diameter [default = 1]**: — The diameter of the grid points in pixels with a zoom of 100%.

• **Select Point Color… [default = gray]**: — The color of the grid points.

There is also a button to reset all the values back to the factory default settings.

**Main menu and Toolbar commands**

This section describes the SoC Designer Plus toolbar buttons and menus.

• *The Toolbar*
• *The File menu*
• *The View menu*
• *The Object menu*
• *The Control menu*
• *The Debug menu*
• *The Window menu*
• *The Help menu*
The Toolbar

The Toolbar buttons (Figure 97) provide a fast way to access functionality that is available from the menu items. The Toolbar status indicators display the current system state.

**Figure 97 Toolbar**

The Simulation Toolbar contains the following items:

- **Open, Save, and Close** — Click on these buttons to open, save, or close a simulation model.
- **Brkpts, Checkpts, Profile, and Trace** — Click on these buttons to toggle displaying the Breakpoint Manager window, the Checkpoint Manager Window, the Profiling Manager window, and the Trace Manager window. (The CheckPtMgr option is only available with the Swap/Play Licensing feature.)
- **MemMap** — Click on this button to display a read-only version of the Memory Map Editor dialog and view the memory map configuration for the bus masters (see Memory Map Overview).
- **Wave** — Click on this button to display the Waveform Viewer window and view data for tracing-enabled ports.
- **Analyzer** — Launches System Analyzer for the currently-loaded simulation; this is only active when “Use external profile database” is enabled on the Simulator Profile Preferences dialog (see Simulator preferences for profiling configuration).
- **Run, Stop, Step, Step n** — Click on these buttons to run and stop the simulation.
- **Reset** — Click on this button to reset the simulation and set the cycle count to 0.
- **Sync All** — Click on this button to synchronize open viewer windows for software and hardware profiling, and waveform features.

*Note:* Segment Size, Zoom Size, and Start Cycle data is not always possible to synchronize. For Segment Size, ensure that all open windows have the same x-axis display format.

- **System state** — The state indicator shows whether the system is running or stopped.

To display the Simulation Toolbar, select **Simulation ToolBar** from the **Window** menu.

The following additional controls can be displayed in the Toolbar area:

- **Cycle counter** — The cycle counter displays the current processor cycle. To display this button, select **Cycle Counter** from the **Window** menu.
- **Zoom %** — The Zoom text box displays the current magnification of the system in the Diagram window. Use the controls at the side of the display to increase or decrease the zoom factor. To display this button, select **Zoom ToolBar** from the **Window** menu.
- **Step N** — The Step N control displays the currently specified step size. Click on the button below the step size value to advance the simulation that number of steps. To display this button, select **Step N ToolBar** from the **Window** menu.
The File menu

Use the File menu to open, save, and exit simulations. The menu has the following entries:

- **Open…** — Select this item to open a dialog that enables you to select a system project (.mxp or .mxe) file, a batch file (.mxscr), or an existing simulation (.mxs or .mxr) file.

  **Note:** If your models are linked dynamically to external libraries, such as a high precision math library, you must make sure that the required library is in a place that the system’s dynamic loader can find it. On Linux systems this means putting the path of the library in your LD_LIBRARY_PATH. On Windows, the path to the location of the library must be in the PATH environment variable.

If a system project file containing programmable cores is loaded the Select Applications dialog is displayed, where you can pick application files to be loaded into the core(s).

- **Close** — Select this item to close the currently loaded simulation. If the simulation is a new or modified simulation, you are prompted to save the simulation.

- **Save** — If the current simulation was loaded from an existing file, selecting Save writes the simulation to the same file (and in the same format as it was read). The .mxs and .mxr files have different formats. If the simulation is newly created, clicking Save displays the Save As dialog.

- **Save As…** — Select this item to save a newly-created simulation or to save an existing simulation to a new file. The Save As dialog supports the following file types:
  - Simulator Configuration (.mxs file)
  - Save/Restore Checkpoint (.mxr file)
  - Swap & Play Checkpoint (.mxc file)

Refer to Saving a simulation: mxs, mxr, and mxc files for information about these file types.

After you choose your file type, click Save to display a standard file browser.

- **Print…** — Select this item to print the system as displayed in the Diagram window. The Print dialog is displayed to enable you to select a printer.

- **Preferences…** — Select this item to change the look and feel of SoC Designer Simulator. There are three sections for SoC Designer Simulator preferences.

Selecting OK and Save saves all of the currently set preferences.

- **Restore debug view…** — Select this item to display a dialog for loading the debug context from a previously saved version of the .mxs simulation file. Any breakpoints or probes that were present when the simulation was saved to file are restored when the file is opened. The simulation, however, is not reset.

- **File list** — A list of all recently opened files is displayed near the bottom of the file menu. Selecting a file from the list opens the file.

- **Exit** — Select this item to exit SoC Designer Simulator. If a simulation is running, it is stopped before closing.

If you have made any modification to the simulation, a prompt asks if you want to save before closing the simulation.

If the simulation window has been launched from SoC Designer Canvas, only the simulation window is closed. If SoC Designer Simulator has been started directly, the application exits.
The View menu

The View menu is similar to the one in the SoC Designer Canvas window. The following menu items are present:

- **Simulation Hierarchy…** — Select this item to display the Hierarchy Of dialog shown in Figure 98. Use the Properties, Ports, Parameters, and Debuggers tabs to display detailed information on a selected component in the System list.

![Figure 98 Hierarchy Of dialog](image)

- **Clock Connections, Signal Connections, and Transaction Connections** — Select these items to toggle the display of the specific connection type. If off, the ports and connections are not displayed.

- **Grid** — Select this item to toggle display of the grid.

  *Note: Movement of objects is bound by the snap to grid flag not the display of the grid.*

- **Grid Options…** — Select this item to display the dialog box described in Diagram window appearance.

- **Zoom In, Zoom Out** — Select these items to zoom the diagram in/out. The step size changes depending on the current zoom level.

- **Set Zoom to 100%** — Select this item to reset the zoom level to the default of 100% (this is the normal zoom).

- **Zoom to Fit** — Select this item to set the zoom level so that the entire diagram fits into the Diagram window.

- **Set Zoom…** — Select this item to display a dialog box that enables to set the amount of zoom. The zoom can be in the range of 10% - 500%.

- **Center on Selected Object(s)** — Select this item to center the Diagram window over the selected object or objects.
• **Center Diagram** — Select this item to place the center of the system diagram at the center of the Diagram window.

• **Clear Output Window** — Select this item to clear the output window. All output is lost after you use this item, unless logging to file is enabled.

**The Object menu**

The **Object** menu contains items that are typically found in context menus for components or connections:

• **Edit Parameters…** — Select this item to display a dialog box in which:
  - The first section contains all the read-only properties of the component: type, version, load file extension, description, and debuggers supported.
  - The second section contains a list of all the ports of the component.
  - The third section contains all the editable parameters of the component.

  To change the value of a parameter:

  1. Double-click the left mouse button in the **value** column under the specific row of the parameter to be edited.
  2. Type in the new value.
  3. Click in the background of the list box to accept the change.

    *Note: Notice that the button to the right of the value field has changed to reset. Clicking this button resets the value to the factory set default value.*

• **Insert/Remove Breakpoint** — Select this item to insert a breakpoint on the selected connection. If the breakpoint already exists, selecting this menu option removes it.

• **Insert/Remove Monitor** — Select this item to insert a monitor on the selected connections. If the probe already exists, selecting this menu option removes it.

• **Enable/Disable Tracing** — Select this item to enable or disable tracing on the selected connections.

**The Control menu**

Use the **Control** menu to start, run, or stop the simulation. The menu has the following menu items:

• **Restart Simulation…** — Select this item to cause a hard reset in the simulation.

  If application files are associated with any of the components, the Select Application Files dialog is displayed. The default files to load are the currently loaded application files. Change the files if required and click **Proceed**. (If no application files are associated with any components, the dialog is not displayed.)

• **Reset** — Select this item to cause a soft reset in the simulation. It resets the cycle count to zero and resets all models in the system. Unlike **Restart Simulation**, the application files are not reloaded.

• **Run** — Select this item to start the simulation running and continues running until you stop the simulation or a breakpoint is hit.

• **Stop** — Select this item to stop the simulation if it is currently running.

• **Step** — Select this item to simulate one system cycle.
Note: One simulation cycle may not be equivalent to one instruction in the instruction set simulators. For instruction stepping, use the step button in the debugger attached to the processor.

- **Step n …** — Select this item to display the dialog shown in Figure 98 and step the simulation a fixed number of cycles:
  1. Enter the number of cycles to step in the Cycles text box.
  2. Optionally, specify a delay between cycles by entering a value between 1 and 99 in the Delay text box. The delay is in 1/10ths of a second and delay time therefore varies between 0.1 and 9.9 seconds.
  3. Click Step to start the simulation for the specified number of cycles.
  4. Click on the Iterate button to continuously run the specified number of cycles, stop for the specified delay and then run again.
  5. Use the Stop button in the toolbar (or the Stop entry on the Debug menu) to stop the simulation before the specified number of cycles has been reached (for Step mode).

![Figure 99 Step n dialog](image)

- **Cosimulation Turbo Control** — This item is shown only when a system is configured for HDL cosimulation with the CoDesign Package. For specific information, see the SoC Designer CDP HDL Cosimulation Guide.
The Debug menu

Use the Debug menu to control how debug information for components is displayed. The menu has the following entries:

- **Display Messages** — Select this item to toggle displaying of messages generated by the components in the currently running simulation. The components still generate the messages however. If this option is turned off, the simulation performance is improved at the expense of diagnostic information.

- **Log Messages To File** — Select this item to send all output to a file in the preferences directory:
  - On Linux this is usually a `${HOME}/.maxsim/mxexplorer/logs` directory.
  - On Windows, the `.maxsim/mxexplorer/logs` directory branch is within your profile directory.

If this option is turned off, the simulation performance is improved at the expense of diagnostic information.

- **Remove All Probes** — Select this item to remove all probes from the current diagram.

- **Breakpoint Manager** — Select this item to display the Breakpoint Manager window that is used to administer all breakpoints (including component level breakpoints and system level breakpoints).

- **Checkpoint Manager** — This option is only available with the Swap&Play Licensing feature.

  The Checkpoint Manager window displays the checkpoints that have been created and allows you to restore those checkpoints, spawn simulation, or save checkpoints as .mxc files.

  *Note:* The CheckPoint Manager is designed for use between Fast Models and Cycle Accurate systems.

  For details about .mxc files, refer to [Saving a simulation: mxs, mxr, and mxc files](#).

  For details about Fast Models, refer to the [SoC Designer Fast Models System Creator User Guide](#).

- **Profiling Manager** — Select this item to open a dialog that enables you to control the collection of profiling data from components that implement the CAPI interface. See [Profiling windows](#).

- **Trace Manager** — Select this item to open a dialog that enables you to administrate all traces in the current simulation.

- **Memory Map Editor** — Select this item to display the Memory Map Editor dialog and view the memory map configuration for the bus components.

- **Waveform Viewer** — Select this item to open the Waveform Viewer. The viewer enables you to display traces without leaving the SoC Designer Simulator application. See [Waveform Viewer](#) for additional information.

- **View Host Level Performance Profiling** — Select this item to open the Host Level Performance Profiling Viewer. The viewer enables you to display accumulated run time for a module, a function, or a source code line.

- **Reset Host Level Performance Profiling** — Forces the accumulated time that displays in the Host Level Performance Viewer to reset to 0.

- **Launch Debugger for** — Select this item to display a submenu containing all the components that have support for at least one of the debuggers supported by SoC Designer. Clicking on one of the components in this sub menu launches the default debugger associated with that component. The debugger to be launched is specified in brackets next to the component’s instance name.
• **View Registers for** — Select this item to display a submenu listing all components that implement an CADI interface. These components enable you to view and modify the component’s registers. Selecting an item from this menu opens a dialog displaying the registers and their current values for the selected component.

For more details see *Component-level Debugging*.

• **View Memory for** — Select this item to display a submenu listing all components that implement a CADI interface. The components permit you to view and modify the component’s memory. Selecting an item from this menu opens a dialog displaying the current values held in the memory for the selected component.

For more details on using this window see *Component-level Debugging*.

• **View Disassembly for** — Select this item to display a submenu listing all the components that implement a CADI disassembly interface. These components enable you to view the disassembly and to step through the instructions.

For more details on using this window see *Component-level Debugging*.

• **Refresh External Debuggers** — Select this item to send synchronization information to all attached debuggers.

### The Window menu

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

• **Simulation Toolbar** — Select this item to toggle display of the Simulation Toolbar.

• **Cycle Counter** — Select this item to toggle display of the cycle counter.

• **Zoom Toolbar** — Select this item to toggle display of the zoom control.

• **Step N Toolbar** — Select this item to toggle display of step n control.

• **Tools Window** — Select this item to toggle display of the System Configuration and Parameter tools windows.

• **System Configuration Window** — Select this item to toggle display of the System Configuration window.

• **Parameter Window** — Select this item to toggle display of the Parameter window.

• **Output Window** — Select this item to toggle display of the Output window. If the Output window is not displayed, the taskbar and command line are also not displayed.

• **Command Line** — Select this item to toggle display of the command line.

• **Taskbar** — Select this item to toggle display of the taskbar.

### The Help menu

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

• **Help** — Select this item to open the *<Product name> <Subtitle, revision, or version>* document using Acrobat Reader.

• **Tip of the Day** — Select this item to display helpful hints about using SoC Designer Simulator.

• **About** — Select this item to display version information.

• **About System** — Select this item to display detailed system and session setup information.

• **What’s this** — Select this item to change the cursor to a question mark. Clicking on a control or window in SoC Designer Simulator displays general information about that item.
**Status bar**

Move the mouse cursor over an menu item, toolbar button, or component to display additional information about the object.

**Output window**

Most of the menu items and buttons log information in the Output window.

Right-click in the Output window to display the context menu:

- **Copy to clipboard** — Select this item to copy the contents of the Output window to the clipboard. The information can then be pasted into a text file for later analysis.
- **Clear** — Select this item to delete all of the text from the Output window.
- **Select Font** — Select this item to display the Select Font dialog that enables you to select the font used to display messages. Select the new font and style from the dialog and click **OK**.

**Command Line interface**

Use the Command Line text box (located at the bottom of the Output window) to enter most of the commands that are available from the graphic user interface. For example, typing `step` into the Command Line and pressing **Enter** advances the simulation by one step.

If you are writing a SoC Designer script, you can use the Command Line to try out the commands before including them in the script. For more information on scripting, see the *MxScript Reference Manual*.

Right-click inside the text box to display the context menu that enables you to edit, copy, or paste text.
Title bar

The title bar of SoC Designer Simulator contains the name of the application in addition to the current system being simulated.

If there is no system loaded, the title bar contains the application name **No Simulation Loaded**. If the simulation has been modified and not yet saved, an asterisk is placed to the right of the system name.

Task bar

If user windows (such as monitor windows for example) are open, a task bar is displayed above the status bar and displays the name of each of the user windows. Click on the icon in the task bar to bring the selected window to the front.

System menu and controls

SoC Designer Simulator supports the standard system menu and controls: move, size, minimize, maximize, restore, and close.

The appearance and function of these controls vary depending on the operating system. See the operating system help for more information.
Tools windows

The System Configuration and Parameters windows are collectively called the Tools windows. These windows display information about the system and its components. To display the windows, use the **Window** menu (see *The Window menu*).

To adjust the area occupied by the windows, move the dividing bar between the Diagram window and the Tools windows (see Figure 100).

**Figure 100 Changing the size of the Tools windows**
The Tools windows are typically displayed as part of the main application window. The windows can be undocked by selecting the bar at the top of the windows and dragging the panel off the application window (see Figure 101) or by selecting Undock from the context menu.

Figure 101 Undocking the Tools windows
Left-click over a blank part of one of the tools windows to display a context menu (Figure 102). You can use the context menu to toggle display of some or all of the tools windows.

**Figure 102 Tools menu context menu**

![Tools menu context menu](image)

The Parameter window (Figure 103) contains all the editable parameters of the currently-selected component. If more than one object is selected, or the object is not a component, the Parameter window is blank.

**Figure 103 Parameter window**

![Parameter window](image)

The window contains the following columns:

- The **Parameter** column lists all the parameters available to be edited.
- The **Value** column contains the current values for all the components.
- The **default** column contains a button to reset the value back to the default value. If the value currently is the default, the button is disabled. Otherwise, the button displays the reset icon.
- Parameters that are editable are indicated by a check mark in the fourth column.

*Note:* Double-click the left mouse button over the editable value that you wish to modify. This invokes an in-place text edit field. Modify the value and then press the **Enter** key (or click elsewhere in the application) to accept the value. Press the **Esc** key to cancel the change.

- The **Type** column indicates whether the parameter is a Boolean or numerical variable.
System Configuration window

The System Configuration window contains a list of all the components that have been used to create the displayed system model.

Selecting an object in the System Configuration window also selects the object in the Diagram window.

Click on the + symbol next to a component to display its contents:

- A system component in the list expands to show all the components that it is constructed from.
- A basic component expands to show all the ports that are present on the component.

The context menu displayed when you right-click on an entry in the System Configuration window is the same context menu displayed when you right-click on a component or port in the Diagram window.

Preferences dialog

This section describes the configuration settings in SoC Designer Simulator sections of the Preferences dialog. For information on how to use the Canvas-related sections of the Preferences dialog, see Preferences dialog.

General Simulator preferences

The General Simulator preferences dialog (*Figure 104*) allows you to specify general Simulator behavior.

*Figure 104 General preferences for Simulator dialog*

This dialog has the following panels:

- **Application Options panel** — Specifies the:
  - Number of recently-loaded simulations to remember
  - Number of commands issued on the command line kept in the history
  - Number of output lines kept in the main window
- **Restore Window Geometry For Simulator check box** — Enables or disables restoring of the simulation geometry for each simulation session. It restores the window size and position of SoC Designer Simulator to the specifications in the simulation file when it is reloaded. These details are written to the simulation file every time you save it. Note that under Linux, this option is called *Restore All Windows on mxs Load*; the functionality is the same.
• **Double Click on Components panel** — Specifies how a component reacts to a mouse double-click. **Edit Parameter** opens the Parameter window. Opens Sub-System opens the subsystem (applies only to subsystem hierarchical components).

• **Remember panel** — Specifies whether previously-opened application files and previously-set breakpoints are used when an `.mxp` file is reloaded.

• **Clear All** buttons clear the list of remembered files and breakpoints.

• **Clear Recent File History List button** — Empties the list of files that have recently been opened.

**Simulator preferences for appearance**

The options in this section affect the appearance of the SoC Designer Simulator.

**Figure 105 SoC Designer Simulator preferences for appearance**

This dialog has the following panels:

• **Miscellaneous panel** — Controls the display of windows or controls as follows:
  
  • **Show Cycle Counter** — Toggles the cycle counter in the toolbar.
  
  • **Show Commandline** — Enables or disables the command line window that is displayed at the bottom of the SoC Designer Simulator window.
  
  • **Display external ports** — Displays external ports in the Diagram window.

• **Font size panel** — Selects the font sizes for the register, memory and disassembly windows and enables increasing/decreasing the font used in the CADI register, memory and disassembly view windows.

• **Window Manager panel** — Specifies whether minimizing of windows is managed by the **Taskbar** or by explicitly minimizing the windows.

• **Advanced panel** — Displays the Advanced Preferences dialog, in which you may choose a different font for text in CADI probe windows. The default font for text associated with probes is 10-point Courier New.
Simulator preferences for general simulation behavior

The preferences in this section apply only to the SoC Designer Simulator.

Figure 106 Preferences dialog for general simulation behavior

This dialog includes the following panels:

- **Miscellaneous panel** — Includes:
  - **Mode To Display Cycle Count In** — Controls the cycle count display format.
  - **Display Performance** — Enables display of the performance data in the output window when the simulation is stopped. This requires the execution of more than 1000 cycles.
- **Unconnected Ports panel** — Allows you to display or suppress warning messages when these ports are accessed.
- **Memory View panel** — Allows you to enable or disable modification of read-only memory from the CADI view.
- **Verbosity panel** — Controls the level of errors to display.

Simulator preferences for Tracer features

You can set preferences for **Tracer** Simulation features using the Preferences dialog for Tracer features (Figure 107).

Figure 107 Preferences dialog for Tracer features
This dialog has the following panels:

- **Trace Probes panel** — Use this option when setting a trace to include multiple signals. When you set a trace, the Tracer Properties dialog displays, in which you enable or disable particular channels to trace.

  **Don’t prompt** is disabled by default. In this state, SoC Designer displays the Tracer Properties dialog for each connection, allowing you to modify each connection’s channels to trace. If you enable Don’t prompt, SoC Designer applies the settings from the initial Tracer Properties dialog to each connection you included in the trace.

This option is also available on the Tracer Properties dialog. Refer to *Configuring Tracer Probes*.

- **Memory Usage panel** — The Memory Usage bar indicates how much of the allocated memory has been used.

  The Usage Settings button opens the Memory Usage dialog (see *Figure 108*), where you can adjust the maximum memory used for tracing, and configure the page size used for each trace channel.

  ![Figure 108 Memory Usage dialog](image)

  If the buffer for this page becomes full, the buffer size is increased by the amount specified (subject to the total limit imposed by the value in Max Memory Usage). The page size values determine the compromise between tracing performance and memory consumption:

  - The default values are suitable for standard computer configurations in the most common cases, for example, 256MB of RAM.
  - If a very large number of trace channels are required, or if a lot of simulation time is to be covered by the trace (for example, much greater than one million cycles) you may want to reduce the page size.
  - If you are using a workstation that has a lot of available memory, you can sometimes improve performance by increasing page size.

- **Export to VCD panel** — The options in this section are for backward compatibility.
Simulator preferences for profiling configuration

You can set profile buffer characteristics in the Profile Settings dialog (Figure 109):

Figure 109 Profile Settings preference settings

- **Use External profiling** — Indicates that you want monitored connections to log transaction data to a .profile file for use by System Analyzer. Profiler probes continue to generate profiling data for SoC Designer standard profiling. See the *ARM System Analyzer User Guide* for more information.

- **Total raw data buffer size** — Changes the default buffer size allocated for the profiling streams. For example, if you are recording a large number of streams, you may want to increase this value.

- **Total data buffer per view** — Changes the maximum memory allocation for displaying the profiling data. The default value is generally adequate and does not usually require changing.

- **Page size when buffered to disk** — Changes the size of the pages displayed for profiling data buffered to disk. The default value is generally adequate and does not usually require changing.
Simulator preferences debug behavior

This section explains component debugging options.

Figure 110 SoC Designer Simulator preferences for debug features

The page is divided into the following panels:

- **Debug Messages panel** — Provides debug options that help you to identify problems in a model. This panel includes:
  - **Enable Messages Before** — Enables or disables the display of all messages related to the selected functions. When Enable Messages Before is disabled, components still generate messages, but do not display them; if logging is enabled, logging still occurs. Selecting Enable Messages Before forces the simulation to print out a debug message for the functions you specify. For example, checking communicate() results in the display of a debug message before the communicate() function is called for each of the clocked components in the simulation, and for every step executed.
  - **Interactive Mode** — When enabled, after each debug message is displayed, you are prompted to continue.
  - **Stop Simulation on Warning** — When enabled, the simulation stops if a warning message is generated.
  - **Monitor panel** — When Show debug transactions in monitors is enabled, this includes lines for ReadDbg and WriteDbg events in monitor history dialogs.

*Note:* In batch mode simulation, debug message output can be controlled via the environment variable ENABLE_DEBUG_MESSAGES.
• **Logging panel** — Changes settings for logging. This panel includes:
  
  • **Enable Logging to File** — Enables and disables logging.
  
  • **Clear Log File on Startup** — If checked, the log file is cleared on startup of the tool. By default, this option is enabled, so that a log file from a previous run is lost the next time you start the tool.
  
  • **Log File** — Enter the path for logging. If you enter a new file, it is used instead of the default file, the new file is used.
  
• **Component Calling Order panel** — Contains controls for modifying the order in which components communicate and updates are called during a simulation. Use these controls to verify that components do not have order dependency problems. The system must behave correctly regardless of the order in which components communicate and updates are called.
  
  • **Default Calling Order** — Causes `Communicate()` and `Update()` to be called on each component in the default order for the tool.

  *Note: The order in which the tool calls the components is not guaranteed. The components must be developed in such a way that they are not order-dependent.*

  • **Reverse Calling Order** — Causes the components to be called in reverse order instead of the default order that is normally used.
  
  • **Progressive Permutation Ordering** — Changes the order in which the components are called for every cycle. (A progressive permutation algorithm is used so that every possible order of calling the clocked components is used, one per cycle. After all possible combinations have been exhausted, the order restarts at the beginning.)

• **Advanced panel** — Contains the following controls:

  • **Enable Crash Handling** — Causes the system to do the following when an exception occurs in the code of a model: Stop the simulation, record system information relevant to the reproduction and analysis of the erroneous session, record system information relevant to the reproduction and analysis of the erroneous session, and display an error message (see Figure 111) advising the user where to locate and send the recorded file and other information related to the crash.

  ![Figure 111 SoC Designer exception dialog](image)

  *Note: If Enable Crash Handling is disabled, SoC Designer Simulator does not catch the exception. The exception is passed to the external debugger. On Windows, for example, the Microsoft Visual Studio debugger catches the exception and, if debug information is available, point to the line in the source code that triggered the exception.*

  • **Generate Core Dump** — When enabled, generates a core dump file in the event of a SoC Designer crash. ARM recommends using his feature when working with your support contact to debug tool issues.
**Memory Map Editor**

The Memory Map Editor, accessible from the SoC Designer Simulator Debug menu, lists all memory regions for all slave components. If the system contains hierarchical elements, the memory regions for all slaves in the subsystem are also displayed.

Use the Memory Map Editor, accessible from the Edit menu in SoC Designer Canvas to create and edit new memory maps; see *Memory Map Editor*.

*Note: You can view the memory address space from SoC Designer Simulator if a memory map exists.*

For additional information on migrating a system that does not use memory maps, see *CASI Memory Map Interface*.

**Breakpoint windows**

This section describes the windows related to setting or viewing breakpoints. Refer to *Configuring Breakpoint Probes* for procedures related to breakpoints.

Red dots on a connection indicate that a breakpoint has been set on that connection (see *Figure 112*).

**Breakpoint Conditions and Dialogs**

Breakpoint conditions differ depending on the type of connection. This section describes:

- **Conditions for Memory or Register breakpoints**
- **Conditions for Signal or Transaction breakpoints**
- **Breakpoint Manager dialog**

**Conditions for Memory or Register breakpoints**

The Breakpoint Properties dialog for Memory and Register breakpoints features a pulldown menu used to set your conditions. *Figure 113* shows the available menu options.
Possible breakpoint conditions for a memory location or register are:

- **changes** — Stop when the register/memory value changes
- **equals** — Stop when the register/memory equals a specific value
- **not equals** — Stop when the register/memory differs from a specific value
- **greater than** — Stop when the register/memory is greater than a specific value
- **greater than or equals** — Stop when the register/memory is greater than or equal to a specific value
- **less than** — Stop when the register/memory is less than a specific value
- **less than or equals** — Stop when the register/memory is less than or equal to a specific value
- **within** — Stop when the register/memory is between the specified values. This option is available for memory breakpoints only.

_Note:_ The CADI interfaces do not enable breakpoints on read/write of a register or memory location.

**Conditions for Signal or Transaction breakpoints**

Breakpoint conditions are based on AXI properties, which vary from protocol to protocol. Refer to the appropriate SoC Designer _Protocol Bundle User Guide_ for details about the breakpoint conditions available for the protocol.

*Figure 114* shows an example of the Breakpoint Condition dialog for AXIv2 transactions.

*Figure 114 Breakpoint Condition dialog for an AXIv2 transaction*
Breakpoint Manager dialog

The Breakpoint Manager (Figure 115) dialog allows you to view all breakpoints — on connections, registers, and memory locations — and manage them individually or as a group. This dialog is accessible by selecting **Debug → Breakpoint Manager…** or by clicking the **Brkpts** button in the SoC Designer Simulator toolbar.

**Figure 115 Breakpoint Manager dialog**

The Breakpoint Manager dialog includes the following buttons:

- **Remove** — Removes the selected breakpoint.
- **Enable** — Enables the selected breakpoint.
- **Disable** — Disables the selected breakpoint.
- **Locate** — For connection-based breakpoints, brings the connection on which the selected breakpoint is set to the center of your SoC Designer Diagram window.

For register- and memory-based breakpoints, displays the Register dialog or Memory dialog that contains the code, register, or memory for which the selected breakpoint is set.

Double-clicking on a breakpoint in the Breakpoint Manager dialog performs the same functions.

- **Properties** — Displays the breakpoint conditions set for the selected breakpoint.
- **Remove All** — Removes all breakpoints in the list.
- **Enable All** — Enables all breakpoints in the list.
- **Disable All** — Disables all breakpoints in the list.
Monitor windows

Monitors allow you to view the activity that occurs during each cycle on a connection or bus. Monitor icons on a connection indicate that a monitor has been configured on that connection (Figure 116).

Figure 116 Monitor on a connection

Note: Monitored CHI data is viewable only with the System Analyzer; these results are not visible in the SoC Designer profiling windows. Use SoC Designer to create monitors; during simulation, content is exported to the System Analyzer database. See the SoC Designer System Analyzer User Guide for more information.

This section describes the:

• Summary Monitor dialog
• Expanded Monitor dialog for signals
• Expanded Monitor dialog for generic transactions
• Expanded Monitor dialog for AMBA transactions

Summary Monitor dialog

By default, a condensed version of the Monitor dialog appears when you configure a monitor on a connection (Figure 117 shows an AXIv2 Summary Monitor dialog). Monitor dialogs are customized according to connection type; for this reason, the button to expand the Monitor dialog may be:

• a History button
• a chevron (>>) button

Figure 117 Summary Monitor dialog for AXIv2
Expanded Monitor dialog for signals

For signals, monitors display values passed during the current cycle. If changes occurred during this cycle then the information is displayed in red; if no change occurred, then the information is displayed in gray.

*Figure 118* shows a sample Monitor dialog for a signal.

The Monitor dialog for a signal includes the following features:

- **Summary button** — Condenses the Monitor dialog.
- **Clear Events button** — Clears the events currently listed in the Monitor dialog.
- **Elapsed Cycles view** — Number of cycles that have been executed since the monitor probe was attached.
- **Activity/Cycles view** — Percentage of transactions per cycle.
- **Log transactions to a file checkbox** — Enables logging of transactions to the file you specify in the text field.
- **Browse button** — Allows you to browse to the desired path of the log file.
- **OK button** — Activates your changes and closes the Monitor dialog.

Expanded Monitor dialog for generic transactions

A *generic transaction* is a transaction that occurs on a non-AMBA connection, such as a connection between an MxStub component and an MxBus component.

Monitors allow you to view the transactions that take place during each cycle. If changes occurred during this cycle then the information is displayed in red; if no change occurred, then the information is displayed in gray.
Figure 119 shows a sample expanded Monitor dialog for a generic transaction.

**Figure 119 Monitor dialog for generic transactions**

The Monitor dialog for a generic transaction includes the following features:

- **Summary button** — Condenses the Monitor dialog.
- **Clear Events button** — Clears the events currently listed in the Monitor dialog.
- **Elapsed Cycles view** — Number of cycles that have been executed since the monitor probe was attached.
- **Activity/Cycles view** — Percentage of transactions per cycle.
- **Log transactions to a file checkbox** — Enables logging of transactions to the file you specify in the text field.
- **Browse button** — Allows you to browse to the desired path of the log file.
- **OK button** — Activates your changes and closes the Monitor dialog.

**Expanded Monitor dialog for AMBA transactions**

SoC Designer contains customized monitors for AMBA transactions. SoC Designer recognizes AMBA transaction connections and uses the customized monitors for AXI4, AXIv2 (ARM AXI3), AHB, and APB, rather than the monitor for generic connections.

Figure 120 shows a sample expanded Monitor dialog for an AMBA transaction; in this case, AXIv2.

*Note:* Monitored data varies depending on the protocol; refer to the appropriate SoC Designer Protocol Bundle User Guide for details on the Monitor dialog specific to that protocol.

**Figure 120 Monitor dialog for AXIv2 transactions**
The Monitor dialog for the AXIv2 connection shown above includes the following features:

- **Chevron buttons (<< / >>)** — Condense and expand the Monitor dialog.
- **Channels pulldown menu** — View the active channels, current transactions, or closed transactions.
- **Start Logging button** — Toggle logging activity.
- **... button** — Browse to specify the path of the log file.
- **Append box** — When selected, specifies that SoC Designer appends logging data to the end of a single log file, rather than creating a new log file per instance.

### Trace Manager and the Waveform Viewer windows

Tracing records changes that occur in a register or are transmitted over a connector. Recorded trace information can be displayed in a Waveform Viewer window. You can trace items regardless of whether they are displayed in the waveform viewer.

This section describes:

- **Trace Manager dialog**
- **Waveform Viewer**

*Note: For information on tracing registers, see Tracing Register Contents.*

### Trace Manager dialog

The Trace Manager dialog displays all traces currently set, and allows you to perform actions on them individually or as a group. To display the Trace Manager dialog, select **Debug → Trace Manager** in SoC Designer Simulator. *Figure 121* shows a sample Trace Manager dialog.

*Figure 121 Trace Manager Dialog*
The Trace Manager dialog includes the following features:

- **Enable column** — When a box is selected, records trace information for the associated stream name.
- **Name column** — When a name is selected, actions performed in this dialog apply to that stream.
- **Remove button** — Removes the selected trace from the list and stops the recording of trace information for this stream.
- **Enable/Disable Tracing buttons** — Enables or disables tracing for the stream. If the stream is currently enabled, the Disable button is functional and the Enable button is grayed out. If the stream is currently disabled, the Enable button is functional and the Disable button is grayed out.
- **Locate button** — Moves the connector matching this stream to the center of the Diagram window.
- **Properties button** — Displays the Properties dialog for this trace. The contents of the dialog depend on the type of trace selected.
- **Remove All button** — Removes all traces from the list and stops the recording of trace information for the streams.
- **Enable All/Disable All buttons** — Enables or disables tracing for all streams in the list.
- **Display Wave button** — Displays the Waveform Viewer window (refer to Waveform Viewer). The window contains all the traces that are currently enabled.
- **Dump in VCD format checkbox** — When selected, dumps the tracing data to a file in VCD format. By default, the log file is set to the system name with .vcd as the extension.
- **Browse button** — Allows you to set the location of the VCD trace log file and change its name if necessary.

**Waveform Viewer**

The Waveform Viewer allows you to view and analyze the events on connections, and the values of registers and memories over time (see Figure 122). To display the Waveform Viewer, in SoC Designer Simulator:

- Select **Debug > Waveform Viewer**.
- Click the **Wave** button in the toolbar.
- Click the **Display Wave** button in the Trace Manager dialog (see Trace Manager dialog for more information).

Figure 122 Waveform Viewer window
The Waveform Viewer provides powerful features for analyzing traced waveforms:

- Choice between cycle mode and time mode (not available on all traces)
- Zooming and zoom history
- Dynamic auto-scrolling at runtime
- Transaction awareness, including display of access type (read or write)

Note: Transactions that are split across multiple cycles, such as AHB or AXI bus transactions, have the address, data, and status information correctly represented in the appropriate cycle.

- The Channel column shows the names of each channel and the active channel
- The Value column shows the values of each channel for the cycle under the cursor
- Hierarchical channels enable the display of single bits of any value
- A cursor and left and right locators for detailed event analysis
- Placing the mouse cursor over a waveform displays the value at that point
- Keyboard shortcuts for all features.

Using the cursors and locators

The waveform viewer cursors enable you to conveniently locate events and measure the time between events. The following cursors are used:

- main cursor (blue)
- left cursor (green), also called a locator
- right cursor (red), also called a locator.

When the Waveform Viewer window is first displayed, none of the cursors are visible and the channel in the top row is the active channel. Left-click with the mouse to set the current cursor in the viewer.

To set the left and right cursors, press the Shift key and left or right-click. (You can alternatively use the L and R keys.) Figure 123 shows the window and cursors.

Figure 123 Waveform Viewer window in time mode

Use the left/right arrow keys to move the cursor to the previous/next event in the active channel. The active channel is highlighted in red and can be changed using the arrow up/down keys or by clicking on an entry in the Channel or Value columns.
**Synchronization**

You can synchronize multiple hardware (CAPI) profiling, software profiling and waveform viewer windows by clicking the **SyncAll** button on the toolbar.

To exclude individual windows from resynchronization, click the **Sync** button on the title bar of the windows.

**Using the Zoom functions**

The SoC Designer waveform viewer supports a wide range of zoom features and dedicated buttons.

**Table 6 Zoom Functions**

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Zoom in out" /></td>
<td>Zoom in or out (factor is 2 or 2.5)</td>
</tr>
<tr>
<td><img src="image" alt="Zoom to fit" /></td>
<td>Zoom to fit (zooms all the way out)</td>
</tr>
<tr>
<td><img src="image" alt="Zoom to normal" /></td>
<td>Zoom to normal (zooms to a level where 1 pixel represents 1 cycle)</td>
</tr>
<tr>
<td><img src="image" alt="Zoom to cursors" /></td>
<td>Zoom to cursors (zoom to area defined by the left and right locators)</td>
</tr>
<tr>
<td><img src="image" alt="Zoom history undo/redo" /></td>
<td>Zoom history undo/redo (a circular buffer remembers the previous five zoom levels)</td>
</tr>
</tbody>
</table>

- Zoom in/out at mouse cursor (Ctrl-key + left/right mouse button)
- Zoom using the mouse wheel (while holding down the Ctrl-key)

All buttons have tool-tips that appear when the mouse pointer hovers over the button for more than a second.

**Viewing options**

The waveform of each channel can be displayed in one of the following ways:

- values
- boolean
- graph (with discrete or interpolated values).

The waveform viewer supports graph displays a graph with connected event-dots instead of a printed value. The standard graph displays a line at the level of the last event until the next event occurs. The interpolated graph simply interconnects consecutive events with a straight line.

To select one of the waveform options right-click on a trace in the **Channel** list and choose the desired option from **View waveform as** item on the context menu.
The values in the displayed waveforms can be displayed in the following ways:

- hexadecimal
- full hexadecimal (with leading zeros)
- unsigned decimal
- signed decimal
- floating point (only for floating point CADI registers).

To select one of the view options for values, right-click on a trace in the Channel list and choose the desired option from the context menu View values in item.

When choosing the graph view option, the Y-axis needs to be configured. By default the channel is 64 pixels high and the scale shows a range from 0 to the maximum value permitted by the bitwidth of the channel (for example in many cases 32bit). This can be changed by selecting Configure Graph View… from the context menu.

**Context menu for the Waveform Viewer window**

Right-click on a Waveform Viewer window to display the general context menu or right-click on an individual channel to display the context menu for that channel. The menu has the following entries:

- **Add trace to view** — Select one of the available traces from the list in the Trace Manager window to add it to the Waveform Viewer window.
- **Remove trace from view** — Removes the trace from the list. This option is disabled on the general context menu.
- **Move trace** — Select up, down, top, or bottom to move the position of the trace in the trace list. This option is not present on the general context menu.
- **Show waveform for channel** — Select a channel from the list to toggle display. This option is not present on the general context menu for the waveform display area.
- **Hide/Show SubChannels** — Select the Show/Hide SubChannels dialog. Check or uncheck the individual channels from the dialog and click OK. Double-click on a subchannel name to edit the name as shown in Figure 124.
This option is not present on the general context menu.

**View waveform as** — Select a display type (Values, Boolean, Graph, or Graph Interpolated) from the list. This option is not present on the general context menu.

**View values in** — Select a numeric display type from the list. This option is not present on the general context menu.

**Simulate to cursor** — Runs the simulation until the cycle count indicated by the cursor is reached. If a breakpoint occurs before the cursor is reached, the simulation stops. This option is only present on the context menu for the cursor.

**View values in** — Select a display type (Values, Boolean, Graph, or Graph Interpolated) from the list.

**Configure Graph View**… — Displays the dialog specifying the scale and value range for the graph associated with the channel. **Channel Height** determines the amount of height available for this channel. If the height is too small, the name of the channel might not be visible. If values are outside of the Minimum Value and Maximum Value range, the graph uses the range value instead.
This option is only present on the context menu displayed for a channel that is currently in graph view.

**Figure 125 Configure Graph View dialog**

- **Open new window** — Displays a new Waveform Viewer window with no traces displayed.
- **Duplicate this window** — Displays a new Waveform Viewer window with the same traces displayed as the original.
- **Open trace manager** — Displays the Trace Manager (see *Trace Manager and the Waveform Viewer windows*).
- **Configure waveform view** — Displays a dialog for configuring the Waveform Viewer refresh rate and the X axis scale (see *Figure 126*). If **Autoscroll to Simulation Cycle** is checked, the window always displays the current cycle.

**Figure 126 Configure Waveform Viewer**
Profiling windows

SoC Designer Simulator provides support for profiling and visualization at runtime. Components use the CAPI interface to provide profiling streams that the user can choose to activate and display.

This section contains the following subsections:

- **Profiling Manager dialog**
- **Profiling data dialog**
- **Displaying a profile from the component context menu**
- **Profile Settings Colors dialog**
- **Profiling Settings Groups dialog**
- **Profiling register values and transaction connections**
- **Latency profiling**

Note: When you reload an .mxr file, Profiling data dialogs that you subsequently display are empty. This is because an .mxr file stores state information, but discards previously-recorded profile and monitor information. Continue to run the system to record new profile information.

Profiling Manager dialog

Use the **Profiling Manager** dialog (shown in **Figure 127**) to view the profiling streams that are available and to activate the streams of interest:

**Figure 127 Profiling Manager dialog**

1. Select the **Streams** tab to display the streams that can be profiled.
2. Select the **Enable** checkbox next to the desired streams to record profile information for that stream.
3. Some streams can be displayed with different parameters for the X and Y axis. If the **X axis** or **Y axis** entries contain a dropdown list, select the values for the axis from the list.
4. To save the stream to a CSV (comma-separated value) file, click the checkbox under **Dump to CSV** for the stream. This feature is present for hardware streams only. (To save software profile data, use the **Export raw data** menu option on the Profiling data dialog context menu — see [Profiling data dialog context menu](#)).

5. Profile information is typically recorded to a buffer in memory. If the buffer becomes full, older information is discarded to make room for the new data. To avoid having the older information discarded, click the checkbox under **Buffer to disk** for the stream. If the memory buffer fills, the hard disk is used to hold the overflow data. This feature is present for hardware streams only.

   **Note:** If the **Buffer to disk** checkbox is unchecked, any files previously used to record profile data are deleted.

6. Click on **Display** to display the profiling windows for the selected streams. Each stream is opened in a separate window (see [Profiling data dialog](#)).

7. Select the **Settings** tab (Figure 128) to configure the buffer size and display options for profiles.

   ![Figure 128 Profiling Manager Settings](#)

8. A default buffer size is allocated for the profiling streams. To change the buffer size (for example, if you are recording a large number of streams), enter the new value in the **Total raw data buffer size** text box and click **Apply**.

9. You can also use the **Total data buffer per view** and **Data display range** text boxes to change these profiling characteristics. The default values for these settings, however, are generally adequate and typically do not require changing.

   **Note:** You can also specify Profiling Manager Settings in the Profile sub-dialog of the SoC Designer Preferences menu (see [Simulator preferences for profiling configuration](#)).
Profiling data dialog

The Profiling data dialog (*Figure 129*) shows the profiled information of the selected profiling stream in a graphical form. Each profiling stream can provide multiple channels (represented by different colors within the same diagram). Each channel represents an event type; for example, reads or writes. The display of each channel can be controlled individually by using the appropriate check box in the Legend list.

*Figure 129 Profiling data dialog*

Segment type and size

The settings for the Segment type and Segment size drop-down lists determine how hardware profiling data are summed up in each bar of the Profiling window diagram.

There are the following options:

- **Fixed size segments** — Select the segment type you wish to record and then select the size from the Segment size list.
  
  Fixed size controls how many cycles are summed for each bar. For example, if the value in the Segment size drop-down list is 1000, then the vertical size of the color box inside each bar shows how many times the event has occurred within the 1000 cycles that are represented by the bar.

- **Function-based segments** — This is available only if software profiling has been enabled for a processor. (See *Software Function Profiling*.)
  
  Function-based segments display profiling data on a per-function basis. The Segment size list contains the core components that support software profiling. Each segments' width equals the width of the corresponding function call from the selected core. This permits you to analyze the events that are occurring inside each separate function call. For example, you could use this feature to locate functions that result in cache misses or significant conflicts on the bus.

- **Clock based segments** — This displays profiling data based on a component clock (other than the main simulation clock). The Segment size list contains the clock connections that are enabled for profiling.
  
  To enable profiling, right-click on a clock connection and select Enable profiling from the context menu.

*Note:* You must enable software, clock, or AHB/AXI latency profiling before displaying the profiling window. If you enable profiling after the window is displayed, you must close and reopen the window to have the additional options displayed.
Note: For software profiles, the Segment lists are not displayed.

Zooming

You can zoom in, out, zoom to fit, and zoom all the way in. Figure 130 shows the zoom controls on a hardware Profiling data dialog.

**Figure 130 Zoom buttons**

![Zoom buttons](image)

Note: Software profiles use a window similar to the Waveform Viewer window. See Using the Zoom functions.

You can also use the mouse to zoom profiling and waveform viewer windows:

- Press the Ctrl key and click in the profiling diagram to center the view on the cursor position.
- Press the Ctrl key and use the mouse to select a rectangle in the profiling diagram. The viewer zooms to the selected area.

Synchronization

You can synchronize multiple hardware (CAPI) profiling, software profiling and waveform viewer windows by clicking the SyncAll button on the toolbar.

To exclude individual windows from resynchronization, click the Sync button on the title bar of the windows. The Sync button changes to include a red diagonal line (Figure 131) when a dialog is excluded from Sync All actions.

**Figure 131 Sync button — Dialog excluded from Sync All**

![Sync button](image)
Profiling data dialog context menu

*Figure 132* shows the Profiling data dialog context menu.

**Figure 132 Profiling data dialog context menu**

Right-click on the graph area of a *Profiling data* dialog to display its context menu (see *Figure 132*). The menu entries available depend on the type of window.

- **Export raw data**… — A Save As dialog is displayed to enable you to save the displayed profile information to a text file.

- **Set colors**… — The Profile Color Settings dialog is displayed to enable you to configure the colors used for the different bar graph items (see *Profile Settings Colors dialog*).

- **Set groups**… — The Profile Groups dialog is displayed to enable you to create new display categories (see *Profile Settings Groups dialog*).

- **x Axis Radix**… — The popup selection menu displays the current Radix. To change the current Radix, select *View in decimal format* or *View in hexadecimal format*.

- **y Axis Radix**… — The popup selection menu displays the current Radix. To change the Radix, select *View in decimal format* or *View in hexadecimal format*.
Displaying a profile from the component context menu

If a component supports profiling, you can right-click on it to display a **Profiling** entry in the context menu.

You can use the context menu to quickly display a profile for the selected component:

1. Right-click on the component to display the context menu.
2. Select **Profiling**, the stream to profile, and **Enable** (see Figure 133).

**Figure 133 Profiling entry on the context menu**

![Profiling entry on the context menu](image)

**Note:** AHB or AXI bus latency profiling must be enabled from the context menu before the **Profiling Manager** window for the profile streams to be displayed in the **Profiling Manager** window.

3. Run the simulation to record profiling information.
4. Right-click on the component and select the following from the context menu:
   - Profiling
   - Stream to profile
   - Display

**Note:** The default X and Y axis values are used. To display other values, use the **Profiling Manager** dialog or the **Segment type** and **Segment size** controls (if present) on the **Profiling data** dialog.

**Note:** Selecting **Profile** → **Channel name** → **Display** from the context menu automatically enables the channel if it was not already enabled.
Profile Settings Colors dialog

To change the colors for a Profiling data dialog, right-click on the window and select Set colors from the context menu. The Profile Settings Colors dialog (see Figure 134) is displayed. Double-click on a color to select a new color from the color dialog.

Figure 134 Profile Settings Colors dialog

Select Apply to this stream to use the colors on the selected Profiling data dialog. Select Apply to all streams to create a new color group.

Profiling Settings Groups dialog

The Profile Settings Groups dialog is used to specify recording specific events as a group (for example, recording all of the cache hits in one channel).

Note: Use the Profile Settings Global Groups dialog to specify how software profiling is done (see Software Function Profiling).
To create a new profiling group:

1. In the Profiling data dialog, right-click on the display area and select **Set groups** from the context menu. This displays the Profile Settings Groups dialog (Figure 135).

   **Figure 135 Profile Settings Groups dialog**

   ![Profile Settings Groups dialog](image)

2. Click on the **Add group** button to display the Add Group dialog (Figure 136).

   **Figure 136 Add Group dialog**

   ![Add Group dialog](image)

3. Enter the name of the group and assign a color to the group.

4. Click **OK** to return to the Profile Settings Groups dialog and select the name of the new group from the **Groups** list.

5. Select an event from the column on the left of the dialog and click the right arrow button to add the event to the group (Figure 137).
6. Click **Close** to close the dialog and return to the Profile window.

7. Click on the group names to select the streams to display (see **Figure 138**).

**Figure 137 Profile Settings Groups dialog with events added**

**Figure 138 Profile window with new group displayed**
Profiling register values and transaction connections

You can profile the values assigned to a particular register by enabling tracing for that register. To profile the register values, open the CADI register view, right-click on the register and select **Enable/Disable Tracing**. After running the simulation, open the profiling manager, and select the profiling stream corresponding to that register to be displayed.

It is possible to trace the read/write events for a particular transaction connection:

1. Enable profiling for a transaction connection by right-clicking on the connection and selecting **Enable/Disable Tracing**.
2. Run the simulation to collect data until the simulation finishes.
3. Open the profiling manager.
4. Display the stream corresponding to that connection

*Note:* For AMBA® transaction connections for AXI, AHB, and APB, you can enable round-trip profiling for the connection by right-clicking on the connection and selecting **Profiler** from the context menu.

Latency profiling

To display the latency:

1. Right-click on a connection and select **Profiler** from the context menu.
2. The bus latency icon appears on the connection (*Figure 139*).

*Figure 139 Bus latency profile icon*
3. Right-click on the connection and select **Profiler** from the context menu. The Profiling window appears (*Figure 140*).

**Figure 140 AHB latency profile**

![AHB latency profile](image)

*Note:* The dots displayed indicate transaction endings or any step of a transaction ending.

The positions of the dots in the graph area (see *Figure 141*) indicate the type of transaction and latency by color and position, as follows:

- **Dot color** — The dot color indicates the activity type. If two activities occur at the same point, the dot is split into two differently colored sections.
- **Dot position** — The dot height indicates the length of a transaction or a transaction step. The position on the x axis indicates where a transaction or transaction step ended. The gap between dots shows that no transactions or transaction steps occurred between the two points.

**Figure 141 Dot position indicates AHB latency**

![Dot position indicates AHB latency](image)

**AHB Bus Legend Entries**

The Legend entries have the following meaning for AHB buses:

- **Write Trans** — The total length of the write transaction, starting with `requestAccess` (if arbitration is required) or `address` (if arbitration is not required) until the last beat of data is done.
- **Write-Initial** — The length of the first beat of write data, starting with `requestAccess` (if arbitration is required) or `address` (if arbitration is not required) until the beat of data is done.
- **Write-Burst** — The length of any beat of write data, except the first one, starting with the cycle when the previous beat of data was done.
• **Read Trans** — The total length of the read transaction, starting with `requestAccess` (if arbitration is required) or `address` (if arbitration is not required) until the last beat of data is done.

• **Read-Initial** — The length of the first beat of read data, starting with `requestAccess` (if arbitration is required) or `address` (if arbitration is not required) until the beat of data is done.

• **Read-Burst** — The length of any beat of read data, except the first one, starting with the cycle when the previous beat of data was done.

**AXI Bus Legend Entries**

The Legend entries have the following meaning for AXI buses:

• **Coherency Trans** — The total length of the coherency transaction, starting with accessing the address channel until the response phase is done and all data beats are done.

• **Coherency-Initial** — The length of the first beat of coherency data, starting with address until the beat of data is done.

• **Coherency-Burst** — The length of any beat of coherency data, except the first one, starting with the cycle when the previous beat of data was done.

• **Coherency Response** — The length of response for a coherency transaction, starting with address.

• **Write Trans** — The total length of the write transaction, starting with accessing the address channel until the response phase is done.

• **Write-Initial** — The length of the first beat of write data, starting with `address` until the beat of data is done.

• **Write-Burst** — The length of any beat of write data, except the first one, starting with the cycle when the previous beat of data was done.

• **Write Response** — The length of response for a write transaction, starting with the cycle when the last beat of data was done.

• **Read Trans** — The total length of the read transaction, starting with accessing the address channel until the last beat of data is done.

• **Read-Initial** — The length of the first beat of read data, starting with accessing the address channel until the beat of data is done.

• **Read-Burst** — The length of any beat of read data, except the first one, starting with the cycle when the previous beat of data was done.
AHB/AXI profiling views

Buses can be profiled by latency, events, operation type, or address.

Note: You must enable latency profiling before you open the Profiling Manager for the latency streams to be included in the list of available streams. If you have already opened the Profiling Manager, close and reopen it to refresh the list.

To display the Profiling Manager, either:

• Access a component’s context menu and click the Profile icon
• Select Profiling Manager from the Debug menu

Figure 142 shows the Profile Manager entries for a bus that has Latency and Operation Type options.

The same profiler specification procedures enable profiling for different components, but different component protocols can offer different options for a profiler probe. For example, an AXI latency profiler probe can be inserted on any AXI connection by selecting Profiler from the context menu for the connection.
There are different profiling options for an AXI component. In the Profiling Manager dialog in Figure 143, the AXI profiler streams are selected, and the X-axis is being configured for Address.

Figure 143 Profiling Manager dialog with an AXI component for latency streams

Sample Profile Windows

The Profile window for an AXI Latency stream profile displayed by Cycle and Latency is shown in Figure 144.

Figure 144 Profiling AXI latency by Cycle and Latency
The Profile window for an AHB Latency stream profile displayed by Cycle and Latency is shown in Figure 145.

Figure 145 Profiling AHB latency by Cycle and Latency

Selecting the Latency stream with Cycle for the X axis and Operation Type for the Y axis with an AHB component results in the profile shown in Figure 146. The profile is displaying the operation types related to latency.

Figure 146 Profiling AHB latency by number of operations
Selecting the AHB Events stream with Cycle for the X axis and Operation Type for the Y axis results in the profile shown in Figure 147. The profile is displaying different bus events.

**Figure 147 Profiling AHB bus events by number of operation types**

The profile for AXI Channel activity by Cycle and Channel event is shown in Figure 148.

**Figure 148 Profiling AXI channel events**
Selecting the Events stream with Address for the X axis and Operation Type for the Y axis results in the profile shown in Figure 149. The profile displays the operations that occur in different memory ranges.

**Figure 149 Profiling AHB bus events by address location**

![Figure 149 Profiling AHB bus events by address location](image)

**Figure 150** shows profiling of Latency by Address.

**Figure 150 Profiling AXI Latency by Address**

![Figure 150 Profiling AXI Latency by Address](image)
Figure 151 shows profiling of an AXI Channel stream profiled Cycle and Address.

Zooming in and changing the segment size (for the AHB bus events) results in dialogs similar to those in Figure 152 and Figure 153.
Software analysis

This section describes the tools used to analyze software performance:

- Working with memory content
- Viewing registers
- Tracing Register Contents
- Viewing Disassembled Code
- Software Function Profiling
- Combined Software and Hardware Profiling

Working with memory content

This section describes how to view and modify memory contents.

Displaying the component memory dialog

To display a component’s memory content:

1. Right-click on the component.
2. From the context menu, select View memory... For certain components, the menu choice may display as View child memory for...

The Memory For <component name> dialog appears.

Setting a breakpoint on a memory location

To set a breakpoint on a memory location:

1. In the Memory For <component name> dialog, left-click on a memory location to select it.
2. Right-click and select Insert/Remove Breakpoint from the context menu (see Figure 154).

Figure 154 Insert breakpoint
The cell with a breakpoint is highlighted with a red frame (see Figure 155).

**Figure 155 Breakpoint set**

When the breakpoint is hit, the memory is highlighted with an orange background (see Figure 156).

**Figure 156 Breakpoint hit**

If a breakpoint was hit while the Memory window was closed or in the background, the window is re-opened and brought to the foreground.
To keep a breakpoint location, but disable stopping if the location is hit, select **Enable/Disable Breakpoint** from the context menu. A disabled breakpoint is displayed with a thick black border (see Figure 157).

**Figure 157** Disabled breakpoint

The default breakpoint property is to break whenever the value changes. To change the breakpoint condition:

1. Select **Edit Breakpoint Properties...** from the context menu.

2. In the Breakpoint Properties dialog (**Figure 158**), select the desired condition.

**Figure 158** Editing breakpoint properties

**Saving and loading a memory image**

You can save the current contents of the memory to a file, or load component memory from a file. For more information, refer to the section **Memory Overview**.
Viewing registers

Components that implement a CADI interface permit you to view and modify the registers (see Figure 159). These are displayed in the menu item Debug → View Registers for…. You can also access the register view from the context menu.

Figure 159 Register view

![Register View](image)

Use the tabs on the bottom of the Register window to change the register bank displayed. The list shows values currently held in the selected register bank.

**Note:** Not all components have multiple register banks.

Registers are colored red if the value changes during simulation steps.

To change the value of a register, click on a register value and edit the contents (see Figure 160). You can enter a value in hexadecimal (with prefix 0x) or decimal.
Note: You can use the underline character “_” to separate groups of digits. For example, 0xabcd_1234 is the same as 0xabcd1234.

Figure 160 Changing the register value

To save the current values of the registers, select Copy to Clipboard or Save to File from the context menu for the Register window.

To set a breakpoint on a register value, click on the register and select Insert/Remove Breakpoint from the context menu (see Figure 161).

Figure 161 Register window context menu
A register with a configured breakpoint displays a red dot in front of the register name (Figure 162). When the breakpoint is hit the register is highlighted with an orange background. If a breakpoint is hit while the Register window is closed or in the background, the window is re-opened and brought to the foreground.

Figure 162 Register breakpoint
Tracing Register Contents

To trace the contents of a register:

1. Display the registers for the component.

2. Use the context menu to enable tracing for a single register (or all registers) as shown in Figure 163.

3. The register entry is added to the list of channels displayed in the Trace Manager (see Trace Manager and the Waveform Viewer windows).

   Note: A tracing icon is displayed next to the register to indicate that it is currently being traced.

4. Select Waveform Viewer from the Debug menu (or click on the Wave icon).

5. The Waveform Viewer window is displayed with the register added under the Channel list.
Viewing Disassembled Code

Components that implement a CADI interface for disassembly enable you to view the disassembly at the current program counter (PC) or at any specified address (see Figure 164).

Figure 164 Disassembly view

To view the disassembly for a processor, either:

- Select the processor and select Debug → View Disassembly, or
- Right-click on the component and select View Disassembly from the context menu.

The green arrow always displays the current execution address or the program address of the instruction of the reference stage in the core model.

Symbol information for address labels is also displayed in the code.

Use the control buttons in the Disassembly window to run, stop, and reset the simulation. Use the Step into and Step over buttons to instruction step the simulation.

The instruction step may translate into more than one simulation cycle.

Check the Duration box to display the number of cycles used for the instructions. If an instruction has not yet been reached, the number of cycles is not displayed.

To set a breakpoint:

1. Click on a location and then right-click to display the context menu.
2. Select the option Insert/Remove Breakpoint or Continue Until Here.

**Figure 165 Breakpoint context menu for code**

3. The simulation runs until the breakpoint activates.

4. The current PC (green arrow) is shown at the selected address when the breakpoint occurs (see **Figure 166**).

**Figure 166 Breakpoint hit**
Software Function Profiling

SoC Designer Simulator provides the capability to profile the software running on programmable components. Features include displaying the calling graph over time, and generating statistics for each function.

To take advantage of this powerful feature, components must support the ELF format with debug info. Components supporting this feature have the menu item Profiling/Software in their context menu. Check the Enable submenu item to activate the feature, then open the Profiling window from the Display context sub-menu.

Note: To support software profiling, the ELF/AXF application must be loaded into the processor core through SoC Designer Simulator. This can be done in addition to loading the application through a software debugger.

This section describes:

• Displaying the Function Profiling Window
• Diagram View of the Function Profiling window
• Summary View of the Function Profiling window
• Configuring Function Profiling

Displaying the Function Profiling Window

To display the Function Profiling window (Figure 167) for a core, perform one of the following actions:

• Right-click the core component in the Diagram window and select Profiling → Software → Display from the context menu.
• Click the Profile button to display the Profile Manager window and enable the Software stream for the core.

Figure 167 Function profiling window for CPU
The Diagram panel (see Figure 167) of the Function Profiling window contains two columns:

- On the left the function names of the loaded object files are listed.
- On the right hand side the calling graph is displayed in the time domain.

The Summary panel of the Function Profiling window contains major column groups:

- On the left the function names of the loaded object files are listed.
- On the right hand side the profile information for each category is displayed in a separate column within the panel.

**Diagram View of the Function Profiling window**

If you click on a function to select it, you can then use the keyboard arrow keys to navigate to each call. The selected function is displayed in red (see Figure 168). The cursor can also be placed with the mouse by clicking on the diagram.

![Figure 168 Function Profiling window cursor](image)

By default, the function in the summary row is selected. If the cursor is moved with the keyboard arrow keys, the simulation stops at every function.

You can control which functions are displayed in the window:

- To move a function from the bottom of the window to the top, select the function and press and hold the left mouse button and move the function to a new location in the list.
- To use a filter list to control which functions are displayed, right-click in the Function Profiling window and select **Configure View** from the context menu. Use the Diagram tab of the Configuration dialog to create filter functions (see Configuring Function Profiling).
Right-clicking on a function name displays the context menu:

- **Insert/Remove Breakpoint** — Toggles a breakpoint for the selected function. The breakpoint appears in the Disassembly window. You can also set or reset a breakpoint directly by double-clicking on a entry in the **Function** column.

- **Enable/Disable Breakpoint** — Toggles between enabling or disabling an existing breakpoint for the selected function. A disabled breakpoint is indicated by a gray disk instead of a red disk.

- **Export Raw Data** — Displays a dialog for saving the function trace information to a text file.

- **Configure View** — Displays the configuration dialog.

**Summary View of the Function Profiling window**

Click the **Summary** tab in the Function Profiling window to display a table showing totals for various categories by function. The left panel lists the functions and the right panel contains statistics for the software or hardware activities related to that function.

*Note:* By default, only software statistics (such as Average Duration) are displayed in the **Summary** view. To also display hardware statistics for the component, use the **Summary** tab of the Software Profiling Configuration dialog to select hardware categories.

The Avg Duration column shows the average duration calculated by dividing the function Duration by the number of calls to that function. This is the case for all rows except for the Summary row. For the summary row, the number displayed in the Avg Duration column is the sum of all durations divided by the number of calls.

The row and column order and size can be changed to make the summary more readable:

**To change the functions** that are displayed:

1. Click on the Diagram tab in the Profiling window to display the trace view.
2. Click **Configure View** from the context menu to display the Software Profiling Configuration dialog.
3. Click on the **Diagram** tab and create a filter group.

**To change the order of rows** in the table, click on the column title. Repeated clicking switches between sorting by function and sorting by column value.

**To change the categories** that are displayed:

1. Click **Configure View** from the context menu to display the Software Profiling Configuration dialog.
2. Click on the **Summary** tab to display the current software and hardware streams.
3. Check the streams to record. Streams that are not checked do not have a column present in the Profile window.

**To change the column order** for the listed statistics, select a column title and press and hold the left mouse key while moving the cursor to the left or right.
To change the column size, move the mouse cursor over the column separators in the title bar. The mouse cursor changes into the resize symbol as shown in Figure 169. Press and hold the left mouse button and drag the column border to change the width.

Figure 169 Changing the width of the major column

Right-clicking on a row in the window displays the context menu:

- **Insert/Remove Breakpoint** — Toggles a breakpoint for the selected function. The breakpoint appears in the Disassembly window.

- **Enable/Disable Breakpoint** — Toggles between enabling or disabling an existing breakpoint for the selected function. A disabled breakpoint is indicated by a gray disk instead of a red disk.

- **Copy Summary to Clipboard** — Displays a dialog for copying the summary information to a paste buffer. (This option is only visible from the Summary tab context menu.)

  The copied data is separated by tabs and line feeds so that it can easily be pasted into a spreadsheet application such as Excel.

- **Save Summary to File** — Displays a dialog for saving the summary information to a .csv file. (This option is only visible from the Summary tab context menu.)

  The saved data is separated by commas and line feeds so that it can loaded directly by database tools such as Excel or by a custom application.

- **Configure View** — Displays the Software Profiling Configuration dialog.

**Configuring Function Profiling**

To configure function profiling using the Software Profiling Configuration dialog:

1. Select a function and right-click to display the context menu.
2. Select **Configure view**... from the menu.
The Software Profiling Configuration dialog is displayed (see Figure 170).

**Figure 170 Configuring the Profiling display**

3. Select the display mode as **cycles** or **time** from the **Scale** drop-down list.

4. Use the check boxes in the **Function Name Filter** to select how the functions are displayed in the Profile window.

5. The Filter Function Calls panel enables suppressing the display of functions. This might be useful if, for example, you did not want to see library or initialization functions displayed in the Profile window.

6. The functions to be ignored are entered by clicking on the **New** button in the Configure dialog. The Create New Filter Group dialog is displayed.

7. Enter the file name for the text tile and click **OK** to open the standard editor on the respective platform.
8. Enter the functions you want to include or exclude in a text file (see Figure 171) and save the file.

Figure 171 List of functions in text editor

9. After creating one or more groups, each group can be selected or deselected using the Enable/Disable buttons.

10. Use the Exclude or Include radio buttons in the Filter Action panel to select whether the functions in the group file are excluded from display or whether only the functions in the group files are displayed.
11. Click on the **Summary** tab of the dialog to display the **Summary Columns** information shown in **Figure 172**.

**Figure 172 Software Profiling Configuration dialog, Summary information**

12. Select the details to record for the software and hardware by checking the appropriate boxes:
If, for example, all hardware streams and two software profile entries are chosen from the configuration dialog list, the window displays the two columns for the software (for Figure 173, this is **No Calls** and **Duration**) and a column for each of the hardware streams.

**Figure 173 Function Profiling Summary with only two columns**

<table>
<thead>
<tr>
<th>Function</th>
<th>No. Calls</th>
<th>Duration (Cycles)</th>
<th>SYS_ARM926E-S-Core Instruction Cache</th>
<th>Cache Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>14</td>
<td>2112</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>main</strong></td>
<td>1</td>
<td>1571</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>init</strong></td>
<td>1</td>
<td>1592</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_stackheap</strong></td>
<td>1</td>
<td>377</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_start_main</strong></td>
<td>3</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_init_main</strong></td>
<td>1</td>
<td>84</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_init</strong></td>
<td>1</td>
<td>1047</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>68</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>34</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>625</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>get_tcb_type</strong></td>
<td>1</td>
<td>45</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>33</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If only the core and two software profile entries are chosen from the summary configuration dialog list, the window displays the two columns as shown in Figure 174.

**Figure 174 Function Profiling Summary with only two columns**

<table>
<thead>
<tr>
<th>Function</th>
<th>No. Calls</th>
<th>Duration (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>1282</td>
<td>107518</td>
</tr>
<tr>
<td><strong>main</strong></td>
<td>1</td>
<td>107518</td>
</tr>
<tr>
<td><strong>init</strong></td>
<td>1</td>
<td>107059</td>
</tr>
<tr>
<td><strong>libc_stackheap</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_start_main</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_init_main</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_init</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>libc_ptrace</strong></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

13. Use the Hardware panel of the Software Profiling Configuration dialog to profile hardware accesses and software functions calls together (see **Combined Software and Hardware Profiling**).

14. Select the components (or one or more activities for a component) from the list.
15. From the radio buttons at the bottom dialog, select whether you want to record all details for the hardware streams or only record the selected details.

16. Click **OK** to close the dialog and return to the Profile windows.

**Combined Software and Hardware Profiling**

SoC Designer Simulator can display hardware and software profiles that enable you to relate hardware activity to specific function calls.

To display the profiles:

1. Enable software profiling for the core and enable profiling for the selected component. Use the Profile Manager or right-click to display the context menus for the processor and the individual components.

2. Display the Function Profiling window and the Profiling windows for the hardware activities.

3. Set the Segment type in the component’s Profile window to **Function based**. See also the general setup information in the section *Configuring Function Profiling*.

4. Adjust the zoom and start cycle for each window so that the displays cover the same range (see *Figure 175*).

5. Click the **Sync All** button on the toolbar to synchronize the hardware profiling windows with the software profiling and waveform viewer windows.

6. Examine the windows to find relationships between the two profiles.

For the windows in *Figure 175*, the cache activity is much higher for `memcpy()` than for `wait_for_trigger()`.
Figure 175 Hardware and Function Profiling windows
CheckPoint Manager and Saving Checkpoints

A checkpoint is a file that contains the state of all the components in a system at a point in a simulation. In SoC Designer, checkpoint files have the extension .mxc. These differ from the .mxr simulation state files, which contain the debug view (register and memory windows, breakpoints, etc.) in addition to component state.

Note: For Fast Model systems, the .mxp file associated with the checkpoint is that of the original Cycle Accurate system from which the Fast Model system was created.

Note: The CheckPoint Manager option is available only with the Swap & Play Licensing feature.

The Checkpoint Manager is designed for use between Fast Model and Cycle Accurate systems. It allows you to:

• Create multiple checkpoints. When you create a checkpoint, SoC Designer automatically associates it with the current system (.mxp file).
• Organize your checkpoints.
• Create multiple named workspaces for a particular SoC Designer system.
• Restore a checkpoint and run the simulation from that point. (You can do this from within the Checkpoint Manager and from within SoC Designer Simulator.)

For more about .mxp, .mxc, and .mxr files, refer to Saving a simulation: mxs, mxr, and mxc files.

For detailed instructions on using the Checkpoint Manager, refer to the SoC Designer Fast Model System User Guide.
This chapter describes batch mode simulation. It contains the following sections:

- Overview of Batch Mode Simulation
- Scripted Batch Mode Simulation

Overview of Batch Mode Simulation

SoC Designer provides a batch mode to run simulations in the background or run regression tests without any user interaction. SoC Designer additionally supports extensive scripting features that permit the control of all aspects of simulation and verification of SoC Designer systems.

There are three ways to run a simulation in batch mode:

- **run for a specific number of cycles:** `> sdsim -s 10000 simfile.mxs`
- **run indefinitely:** `> sdsim -r simfile.mxs`
- **run from a script:** When you launch a batch mode script, SoC Designer Simulator executes commands from a text file that resembles a source code file (see Scripted Batch Mode Simulation for details). The scripting language is named MxScript, and it is based on the C language. For information about MxScript, see the MxScript Reference Manual.

Scripted Batch Mode Simulation

You can run a script file from within SoC Designer Simulator, or use the command line to specify that SoC Designer Simulator run a script.

Running a Script from the Command Line

To specify a script file to run from the command line, enter: `> sdsim --script example.mxscr.`

*Note:* When you launch a script from the command line, you cannot combine script options, command options, and runfile options in the same line.
Running a Script from within Soc Designer Simulator

From within the SoC Designer Simulator, open an .mxscr script file from the File menu or toolbar. This displays a Script dialog (Figure 176) with Step, Run, Stop, and Reset button controls to operate the script.

Figure 176 MxScript dialog

To toggle breakpoints in a script, double-click on a line.

If running SoC Designer Simulator indefinitely, the simulation ends only if any of the components calls the halt() command or if an error (or fatal error) occurs. If the option Stop on Warning is set in the SoC Designer preferences, a warning message also causes the simulation to be aborted.

To capture output in batch mode, regular output is directed to stdout. Warnings, errors, and fatal errors go to stderr.

Redirecting Output to a Log File

Use the -l option to redirect output to a log file. For example:

```
sdsim -l logfile.txt
```

This option is for batch mode only. Specifying a log file causes all output to stdout and stderr to be redirected to that file. This option is particularly useful on Windows because other methods of output redirection are not possible.
This section describes debugging components and applications in a C++ environment. There are a variety of facets to debugging a simulation:

- **Target-level debugging** — The embedded software developer debugs the application code that is running on the system. See Component-level Debugging.

- **System-level debugging** — The system designer examines signals and transactions between components to ensure correct system operation. For information see System-level Debugging and Probes.

- **Host-Level Debugging** — An engineer developing a component debugs component behavior and code to ensure that the component operates correctly. One example could be debugging using a host level Visual C++ debugger. For information see Debugging user-defined components.

### Component-level Debugging

SoC Designer offers built-in debug windows for registers and memories of components based on the CADI debug interface. This makes it possible for any component to have its own register and memory windows. Any component with CADI interfaces appears in the list of debuggable components provided by the SoC Designer Simulator.

To see the list of debuggable components, click **Debug View registers / memory for**. You can also right-click on a component in SoC Designer Simulator to see that component's supported debugging options.

See the following sections for more information on the windows and tools that are available for debugging at the component level:

- **Register windows**
- **Memory Overview**
- **Disassembly window**
- **Debuggers**

See also Combined Software and Hardware Profiling.
Register windows

If a component defines registers in its CADI interface, they can be viewed in the SoC Designer Simulator (see Figure 177).

Figure 177 Viewing CADI defined registers

Registers are organized in groups that can be selected using the tabs at the bottom of the register window. Each register can have a different bitwidth. Use the Register Radix submenu to select how the register contents are displayed.

If a register is not read-only, you can modify its value by selecting the current value and entering the new one. You can enter new values in hex, decimal, binary, or octal:

- hex values must be preceded with 0x.
- binary values must be preceded with 0b.
- octal values must be preceded with 0.

Note: You can use the underline character “_” to separate groups of hexadecimal digits. For example, 0xabcd_1234 is the same as 0xabcd1234.

The display format of each register is specified by the debug interface of the component itself. Registers may be displayed as hex or decimal values, floating values, symbolic values, booleans, or strings. By default the display format is chosen according to the definition of the registers in the component. However it is possible to force the display to hex or decimal for all registers by selecting the desired option in the context menu.
Register breakpoints

To set breakpoints on a register, double-click on its name. You can also use the context menu.

A red dot in front of the register name indicates that the breakpoint is set (see Figure 178).

Figure 178 Register breakpoints

A default breakpoint causes the simulation to stop if the respective value changed.

You can modify a breakpoint condition to stop the simulation if the register value is equal to, greater than, or less than a specified value. To do this, access the breakpoint properties dialog from the context menu.
Register tracing

You can trace each register by selecting enable/disable tracing in the context menu (see Figure 179).

Figure 179 Enabling tracing on a register

You can also enable/disable the tracing for an entire register group by selecting the option trace all registers in group in the context menu.

The name of the trace consists of the component name, followed by the register group name and the register name. The generated trace can then be viewed in the waveform viewer (Figure 180) or exported to VCD.

Figure 180 Waveform Viewer window for a register trace
After selecting the registers and signals to display, you can select **View waveform as Graph** to display one or more of the channels as a graph. This can be used to record and display combinations of register statistical information and hardware activity in the same window (see **Figure 181**).

**Figure 181 Graphical view of register values and bus signals**

If the graph is not readable with the size selected, select the channel and then select **Configure Graph View** from the context menu. The **Graph View Settings** dialog appears (**Figure 182**).

**Figure 182 Configure View Settings**

Enter the range in the **Minimum Value** and **Maximum Value** fields. The **Channel Height** field controls the height of the channel graph in the Waveform Viewer window.
Memory Overview

If a component defines memories in its CADI interface then they can be viewed in SoC Designer Simulator (see Figure 183).

Figure 183 Viewing CADI defined memories

A component can have multiple memory spaces that can each have different properties, such as size and bitwidth. Components can define memory blocks that can be used to quickly navigate to a certain address in the memory space.

If a memory is not read-only, its contents can be edited by selecting the desired memory cell and entering the new value. Values are always displayed and entered in hexadecimal format.

Full System Coherent Memory Behavior

For a system with components that support Full System Coherent Memory views, such as the Cortex-A15, Cortex-A53, and Cortex-A57, a debug Read request polls through all connected (and supporting) components containing memories, and returns the valid system value of the address. Similarly, a debug Write updates the data value in all locations that a valid value exists.

Cache-coherent interconnect models, such as the CCI-400, have the capability to backward-propagate debug requests coming on their slave ports, and read/write through all internal memories (typically, CPU caches) upstream from the slave ports. Components that do not support Full System Coherent Memory views do not participate in the coherent views; addresses owned and modified by them may show stale values in memory views from other system components.

Refer to the Memory section of your model guide to determine if your model supports Full System Coherent Memory views.
Requirements for Reading or Reloading a File

When reading a file or loading its contents into a component, the file characteristics must match. For example:

- If the file was saved as ASCII, it must be selected as ASCII for reloading.
- If the file was saved with the Space setting of PROG, the Memory window must have PROG selected when the file is reloaded.

Typically, the Block setting must also match unless you are deliberately moving memory from one block to another.

Loading Files into Memories

For debug purposes, it is possible to upload files into a memory space using the component’s CADI interface. The files are treated as memory images.

1. Right-click on the component in SoC Designer Simulator.
2. From the context menu, select View memory... For certain components, the menu choice may display as View child memory for... The Memory For <component name> dialog appears.
3. Right-click the Memory for <component name> dialog.
4. From the context menu, select Read memory image. The Read memory image from file dialog displays (Figure 184).

Figure 184 Loading files into a memory space

Use this dialog to specify the file name to load and the following options:

- **Start Address** — The address where the image is to be loaded (default is 0)
- **End Address** — End Address of the memory range to read (default is 0xffff)
- **Load File as Binary** — The file is read in binary mode
- **Load File as ASCII** — The file is interpreted as containing only ASCII coding of hex values.

Pressing OK loads to the memory area defined by the start address. The number of bytes loaded is whichever is smaller of the file size or the value in **Bytes to Load**.

*Note: You can use the underline character “_” to separate groups of digits. For example, 0xabcd_1234 is the same as 0xabcd1234.*
The binary mode must be used for loading binary images. The ASCII mode provides an easy way to write custom images by hand. The ASCII reader reads the hex values in a file and loads them into the memory as bytes.

**Sample ASCII file**

Loading a file with a contents as specified above into an empty memory (initialized with zeroes) results in the following memory contents:

```
0x0000 00 12 34 55 0a 0b 0c 04 0x0008 05 10
20 30 00 00 00 00
```

**Note:** This feature is intended for debugging purposes only. Applications must be loaded at simulation start from the Load Applications dialog. This is why it is necessary for simulated systems to include a component that contains an object loader.

**Dumping memory images to a file**

You can store the contents of a memory window to a file in binary or ASCII format.

1. Right-click on the component in SoC Designer Simulator.
2. From the context menu, select **View memory...** For certain components, the menu choice may display as **View child memory for...** The Memory For <component name> dialog appears.
3. Right-click the Memory for <component name> dialog.
4. From the context menu, select **Write memory image**. The Write memory image to file dialog displays (Figure 185).

![Figure 185 Writing memory image to a file](image)

Use this dialog to select the file and dump options:

- **Start Address** — Start address of the memory range to write (default is 0).
- **End Address** — End address of the memory range to write (default is 65535 – for 64k).
- **Write File as Binary** — The file is written in binary mode.
- **Write File as ASCII** — The file is written as ASCII.

If the dialog is closed using the **OK** button, the file is written from the start address to the end address as specified in the dialog.
Memory breakpoints

You can set breakpoints on memory locations by double-clicking on the desired memory location or by right-clicking and selecting from the context menu. A red box around the value indicates that the breakpoint is set.

From the context menu you may also define the breakpoint properties that define the break condition. The default break condition is **break on change**. Other options are break if equal to, not equal to, greater than, or less than a specified value.

Disassembly window

If a component implements the disassembly API, the disassembly can be viewed directly in SoC Designer Simulator. To access it, right-click on the component and from the context menu select **View Child Disassembly For...** The Disassembly window displays (*Figure 186*).

*Figure 186 Disassembly for a component*

The green arrow indicates the current execution address or the program address of the instruction of the reference stage in the core model.

The icons on the Disassembly dialog initiate the following actions:

- Run to debuggable point
- Locate current instruction
- Run
- Stop
- Step into
- Step over (if supported by the model)
- Reset

**Note:** The instruction step may translate into more than one simulation cycle. Check the **Cycles** box to display the number of cycles for the instruction step.

You can set breakpoints on a disassembly line by double-clicking on the first column of the window.
To let the simulation run until a certain point, right-click on an instruction and select **Continue Until Here** from the context menu.

**Specifying disassembly mode**

You can select a disassembly mode (for example, 32bit or 16bit mode) from a drop down list box. As soon as the simulation continues, however, the mode switches back to the current mode of simulation (as reported by the model) and all instructions below the current instruction are displayed in the current mode.

**Using the Run to debuggable point feature**

The run to debuggable point feature, the first button on the Disassembly dialog, forces the processor into a coherent state. When debugging, the model is brought to the debug point automatically whenever a software breakpoint is hit (including single stepping). However, if a hardware breakpoint is reached, or if the system is advanced by cycles within SoC Designer, the model can get to a non-debuggable state. In this case, the Run to debug point button advances the processor to the debug state. After the processor reaches debug state, the model causes the system to stop simulating.

The Run to debug point feature is also available as a context menu item (Run to Debuggable Point) from the component within SoC Designer Simulator.

**Debuggers**

To debug the application source for a processor, select the **Launch Debugger** option from the context menu of the processor component or from the **Debug** menu. Use the **Step** or **Run** buttons in SoC Designer Simulator to advance the code pointer in the attached debugger.

SoC Designer supports the ARM Development Suite 5 (DS-5). Refer to the *SoC Designer DS-5 Debugger Integration Application Note* and the DS-5 documentation at the ARM web site for more information.

**Synchronization of external debuggers**

All connected debuggers are automatically synchronized by the debug server built into SoC Designer Simulator. The debuggers are synchronized every time the simulation starts, stops, or steps. In some cases, however, it is desirable to establish an additional synchronization; for example, if one debugger has changed a resource and it has not been noticed by the other debugger. In that case, select the **Refresh External Debuggers** from the **Debug** menu or use the keyboard shortcut **Ctrl+Shift+F12**.
System-level Debugging and Probes

You can attach debug probes to any existing connection between two components. SoC Designer offers several built-in probes and supports the use of user-defined probes.

The following sections describe how to configure the probe types:

- Configuring Breakpoint Probes
- Configuring Monitor Probes
- Configuring Tracer Probes
- Configuring Profiler Probes

You can configure a connection with Breakpoint, Monitor, Tracer, and Profiler probes simultaneously.

For additional information about these probes, refer to Breakpoint windows, Monitor windows, and Trace Manager and the Waveform Viewer windows.

Configuring Breakpoint Probes

Use breakpoint probes to let the simulation run until a certain condition is met; for example, you can set a breakpoint to stop the simulation if the address equals 0.

The procedures for creating breakpoints and setting their conditions differ based on the connection type. This section describes:

- Setting breakpoints on memory or register windows
- Setting breakpoints on transaction or signal connections
- Setting breakpoints on a disassembly window
- Viewing and Managing all breakpoints

For information about software breakpoints, see Software analysis.

Note: To set breakpoint conditions in batch mode, use MxScript. For more information, see the bpConnSetCond(...) function in the MxScript Reference Manual.

Setting breakpoints on memory or register windows

To set a breakpoint on a memory or register window:

1. Right-click on the component and select View Registers or View Memory from the context menu. The Registers dialog or Memory dialog for the component appears.
2. Click on the register or memory location on which to set a breakpoint.
3. Right-click and select Insert/Remove Breakpoint from the context menu. For register breakpoints, the breakpoint icon appears next to the register location. For memory breakpoints, a red box appears around the memory location.
4. Right-click on the register or memory location and select Edit Breakpoint Properties... from the context menu (this option is available only after a breakpoint has been set). The Breakpoint Properties dialog appears.
5. Set the breakpoint condition you want and click OK.

For a description of memory and register breakpoint conditions, see Conditions for Memory or Register breakpoints.
Setting breakpoints on transaction or signal connections

Breakpoint conditions differ from protocol to protocol. Refer to the appropriate SoC Designer Protocol Bundle User Guide for details.

To set a breakpoint on a transaction or signal connection:

1. Double-click on the connection, or right-click on a connection and select Insert/Remove Breakpoint from the context menu (see Figure 187).

![Figure 187 Transaction and Signal component context menu](image)

You can also select the connection and select Insert/Remove Breakpoint from the Object menu.

2. Set the breakpoint conditions by right-clicking on the connection and selecting Edit Breakpoint Properties... from the context menu. The Breakpoint Condition dialog appears.

3. Set the breakpoint condition you want and click OK.

   For a description of signal and transaction breakpoint conditions, see Conditions for Signal or Transaction breakpoints.

Setting breakpoints on a disassembly window

To set a breakpoint on a disassembly window:

1. Select View Disassembly from the processor component context menu.

2. Double-click on the memory address.

An alternate procedure to set a breakpoint on a disassembly window is:

1. Select View Disassembly from the processor component context menu.

2. Click on the location.

3. Right-click to display the context menu.

4. Select Insert/Remove Breakpoint from the context menu.

Viewing and Managing all breakpoints

To view and manage all breakpoints, open the Breakpoint Manager dialog from the Debug menu or by clicking the Brkpts button. For details about this dialog, refer to Breakpoint Manager dialog.
Configuring Monitor Probes

This section describes:

- Inserting a monitor on a connection
- Selecting the view
- Enabling transaction dumping
- Clearing the Monitor dialog of data

Note the following important information related to monitoring:

- **AMBA transactions** — SoC Designer contains customized monitors for AMBA transactions. SoC Designer recognizes AMBA transaction connections and uses the customized monitors for AXI4, AXIv2 (AMBA AXI3), AHB, and APB. Refer to the appropriate SoC Designer Protocol Bundle User Guide for specific information about the monitoring functionality of each protocol.

- **Single Master/Multiple Slave transactions** — You can connect a bus-master port to multiple slaves. Add a monitor to each slave attached to the master.

Inserting a monitor on a connection

To insert a monitor on a connection:

1. Right-click the connection.
2. From the context menu that appears (see Figure 188), select Insert/Remove Monitor.

A monitor symbol is displayed on the two ends of the connection, and the Summary Monitor dialog appears. The example in Figure 189 shows the Summary Monitor dialog for an AXIv2 connection; dialogs for other connection types may look different.

3. Initially, no data is displayed. Run the simulation to populate the Monitor dialog with data.
4. To enlarge the monitor dialog to view a history of the transactions, click the >> button (for other connection types, the expand button may be a History button).

Selecting the view

*This functionality is available only for AXI4 and AXIv2 connections.*

Use the Channels pulldown menu in the Monitor dialog to choose one of the following views:

- **Channels view** — Shows the most recent activity on all channels. The channel with current activity is highlighted in red.
- **Open transactions view** — Shows transactions on read/write or coherency channels that have not yet finished. This is a two-level tree view, where the top level is a summary of the transaction, and the second level shows the steps of the transaction in order. As they complete, transactions move from the Open transactions view to the Closed transactions view.
- **Closed transactions view** — Shows all completed transactions.

Enabling transaction dumping

*For AXI protocols:*

1. With Channels selected in the pulldown menu, select the Start Logging button.
2. To change the name or location of the log file from the defaults, click the ... button and enter the new information.
3. To specify appending logging data to the end of a single log file, rather than creating a new log file per instance, select the Append box.

*For non-AXI connections:*

1. Check the Log transactions to a file box in the Monitor dialog.
2. To change the name or location of the log file from the defaults, click the Browse... button or enter the new information in the text field.

Clearing the Monitor dialog of data

To clear the contents of the Monitor dialog, either:

- Click the Reset button in the toolbar, or
- Close the Monitor dialog. When you re-insert a new monitor, SoC Designer displays the Monitor dialog without data.

Configuring Tracer Probes

Use tracer probes to log selected transactions to memory for:

- Display in the waveform window, or
- Storage in a VCD transaction recording file. You can view VCD files with a standard GTKWave tool or other standard, publicly-available viewers. Refer to Trace Manager dialog for details.

To insert a tracer probe:

1. In SoC Designer Simulator, select a connection or multiple connections.
2. Right-click on the connection, or on one of a set of selected connections.
3. From the context menu, select Enable/Disable Tracing. The trace symbol \( \mathcal{V} \) appears at each end of the connection and the Tracer Properties dialog opens (see Figure 190).

![Figure 190 Tracer Properties dialog](image)

4. By default, all signals are traced. To disable tracing of certain signals, use the checkboxes located to the left of the signal.

   To use the current settings as the default for subsequent traces, enable **Use these settings as defaults**.

   To turn off prompting for selecting individual signals to be traced, and use only the defaults, enable **Don’t prompt**. With Don’t Prompt enabled, no dialog is displayed when a trace is added. **Don’t Prompt** is also available as a Simulator preference; see Simulator preferences for Tracer features.

5. Click **OK** to apply your changes.

By default, each added trace is automatically displayed in the waveform viewer (see Trace Manager and the Waveform Viewer windows).

   **Note:** SoC Designer permits the insertion of tracer probes at any time. Logging begins only after the insertion of a probe. Before the insertion, the state is displayed as unknown.

The set of signals is specific to the protocol on the connection. For AXI4* connections, all ACE signals are shown, even if they are not part of the selected protocol variant.

For descriptions of the signals profiled, refer to the appropriate SoC Designer Protocol Bundle User Guide or Model User Guide.
Configuring Profiler Probes

To configure a Profiler Probe on a connection:

1. In SoC Designer Simulator, right-click a connection or multiple connections.

2. From the context menu, select **Profiler > Enable**. The Profiler Probe icon appears at each end of the selected connection(s).

3. To view the Profiler Probe dialog, right-click on a connection and select **Profiler > Display**. The Profiler Probe dialog appears.

By default, the Profiler Probe for AXIv2 and AXI4 provides a latency display. This combines the latency for all operation types (write trans, write-initial, write-burst, write response, read trans, read-initial, read-burst) into a single display.

For AXI4 protocol variants with coherency channels (AC, CR, and CD), the Profiler Probe latency display shows coherency transaction latency, as well as read and write latency.

You can view a color-coded version of the Profiler Probe display by using the Profiling Manager and selecting cycle vs. latency on the latency profiling stream. Refer to **Profiling windows** for more information.

To disable profiling on a connection:

1. Right-click the connection.

2. From the context menu, select Profiler. Select the Enable menu choice again to toggle off profiling. The Profiler Probe icon is removed from the connection and profiling is disabled.
Debugging user-defined components

This section provides a simple guide on how to debug user-defined components using SoC Designer Simulator:

- **Debugging with SoC Designer Simulator** describes the built in debugging features of SoC Designer Simulator.
- **Debugging with Microsoft Visual C++ on Windows** and **Debugging with GNU gdb on Linux** cover the use of an external debugger.

This section assumes:

- You are using the debug version of Simulator (for example, started using the `-dbg` option).
- The component contains debug information.

*Note:* It is important to distinguish the different levels of debugging components. This section describes how to debug components using native C-debuggers such as Visual C++ and GDB, requiring the availability of debug libraries and source code for those components. Component-level debugging for end users is possible only with embedded debug interfaces such as CADI and third party debuggers or SoC Designer’s built-in debug windows. See Component-level Debugging for more details on CADI-based debugging.

Debugging with SoC Designer Simulator

The debugging features in SoC Designer Simulator are typically used to test the logic implementing a component, but the SoC Designer Simulator can be used to identify implementation problems.

*Note:* Ensure that you note the file name and location of each component library (`.conf` file). This information helps you ensure that the correct version of the library is used. To view a component library’s file name, access the component’s Edit Parameters dialog (note that the Component Library field is shown only for components with an associated `.conf` file).

The Preferences dialog, accessible from the **File** menu, contains a debug section (see Preferences dialog). Use this dialog to:

- Enable/disable various debug messages to be printed out just prior to a crucial events; using a combination of these messages, you can identify suspect areas of the component. For example, if your model faults during the execution of a cycle, you can enable the debug messages for the `communicate()` and `update()` calls. Now when you step a cycle, you see the order in which each component is called. If your model crashes after a call to update in a subcomponent, the last message displayed informs you of the component `update()` that was called.
- Select file logging options. All messages are logged to the file specified. If the crash in a component brings down the tool, you can see the last message call made to the log file. All messages are logged to the file even though the displaying of messages may be turned off.
- Change the order that their components are called. This is useful for ensuring that the logic or implementation behind your model is not dependent on the default order in which the components are called. Modifying the calling order has no effect on the behavior of the model.
• Enable crash handling. If a crash occurs, an SoC Designer Exception message appears (see Figure 191).

**Figure 191 SoC Designer exception dialog**

The SoC Designer exception error message tells you where to locate a text file that you can send to ARM Support.

### Debugging with Microsoft Visual C++ on Windows

To debug user components under MSVC++, two approaches can be used. You can attach Visual Studio to a running process, or have Visual Studio launch the Debug version of SoC Designer Simulator. Both methods are described below.

#### Attaching Visual Studio to a running process

1. Load a simulation project into the debug version of SoC Designer Simulator.
2. Right-click on a component to display the context menu and select **Attach Host Level Debugger**.
3. Select **Yes** at the confirmation dialog asking if you want to use the selected debugger.
4. The Attach to Process dialog is displayed. Select the **Native** option.

**Figure 192 Attach options**

5. Use the debugger to open source files and set breakpoints.
Starting the debug version of SoC Designer Simulator

1. Open the project that contains the source code for the DLL.

2. From the Project menu, select Project Properties. The Project Properties dialog is displayed (Figure 193).

Figure 193 Project Property dialog

3. In the Configuration: pulldown menu, ensure that Release_with_Debug is selected.

4. In the left panel, under Configuration Properties, select Debugging.

5. In the right panel, ensure the following fields are configure correctly:
   - **Command** text field — Contains the path to the sdsim executable:
     $(MAXSIM_HOME)\Bin\Release\SoCDesigner.exe.
   - **Command Arguments** text field — Contains -G <.mxp file>.
     *Note:* The -G option loads the release_with_debug component libraries; <.mxp file> is your SoC Designer platform file.
   - **Working directory** text field — Set to the directory path where your .mxp file resides.
   - **Enviroment** text field — Sets the PATH enviroment variable to the following:
     PATH=%MAXSIM_HOME%\Bin\Release;%MAXSIM_HOME%\etc\mono\bin;%MAXSIM_PROTOCOLS%\TIM2\amba_socket\lib;%MAXSIM_PROTOCOLS%\Common\lib;%MAXSIM_PROTOCOLS%\AHBv2\lib;%MAXSIM_PROTOCOLS%\AHBv2\xactors\lib;%MAXSIM_PROTOCOLS%\AXIv2\lib;%MAXSIM_PROTOCOLS%\AXIv2\xactors\lib;%MAXSIM_PROTOCOLS%\AXI4\lib;%MAXSIM_PROTOCOLS%\AXI4\xactors\lib;%MAXSIM_PROTOCOLS%\AXI4\xactors\lib;%PATH%

6. Click **OK** to save your changes.

7. From the Visual Studio Debug menu, select Start Debugging. This invokes SoC Designer Simulator.
Note: The following alert may appear:
"Debugging information for 'socdesigner.exe' cannot be found or does not match. Symbols not loaded. Do you want to continue debugging?"

If this occurs, Click Yes.

SoC Designer Simulator opens. The simulation breaks when a breakpoint in the loaded model library’s code is reached, or when the SoC Designer Simulator initialization completes.

Continue debugging normally.

Debugging with GNU gdb on Linux

Debugging on Linux platforms is not restricted to gdb; you can use whichever debugger is available and supports GNU compiled code. The following instructions cover only gdb using ddd as the front end, but these instructions are also applicable to other debuggers.

To start the debug version of SoC Designer Simulator:

1. Specify the following command on the command line:
   > sdsim –dbg

2. Before SoC Designer Simulator starts, it prompts for the command to start a debugger. Enter ddd or specify the debugger on the command line as follows:
   > sdsim –dbg –debugger=ddd

DDD (or the specified debugger) opens with SoC Designer Simulator loaded as the executable to debug (Figure 194 shows the Program menu).

Figure 194 DDD debugger console Program menu
3. Select **Run** or type `run` in the gdb command window. SoC Designer Simulator opens.

4. Before you load components, check Preferences to ensure that the extra debug messages for constructor calls are enabled and that interactive mode is set (see Figure 195). These are requirements to debug the model’s constructor.

![Figure 195 Setting debug preferences](image)

5. Load your components.

Because SoC Designer Simulator contains no debugging information, breakpoints cannot be set until the debug environment is specified.

**Setting Breakpoints in Constructors**

Performing the following steps before loading your components specifies that you be prompted before the constructors of the components are called. This gives you time to place breakpoints in the components’ constructors if necessary:

1. When the prompt informing you that the constructor for component `xyz` is about to be called, the simulation is halted until you click **Next** or **Continue**.

2. Pass control of the execution to gdb by pressing the **Interrupt** button on the **Program** menu.

3. Open the source files for the models and set breakpoints. To open source files select **File > Open Source**.

4. Now set breakpoints in the source as usual. After all breakpoints have been set, select **Program > Continue** to return control to SoC Designer Simulator.

5. Now when a breakpoint is hit, control passes back to the debugger and you can continue to debug as you would for a normal project.
Common problems with gdb/ddd

Ensure that you use correct versions of gdb and ddd. The required gdb version is version 6.0 or above.

The version of ddd is not as critical. ARM recommends version 3.3, which is considered reliable.

- **Problem:** Starting ddd or gdb with sdsim as an argument results in the error message no executable from the debugger.
  **Possible Cause:** The file sdsim in your path is a front-end script and not the actual executable. Use the sdsim -dbg option to run with a debugger.

- **Problem:** When trying to run within gdb/ddd, SoC Designer Simulator cannot be started because a shared library cannot be found.
  **Possible Cause:** The gdb debugger typically starts a new shell that launches the execution. If the shell startup script (a .cshrc for c-shell users) resets the LD_LIBRARY_PATH environment variable then the shared libraries necessary to run SoC Designer Simulator cannot be found. Check your startup script. (For example start a new shell and verify whether the LD_LIBRARY_PATH variable is set.) You may be required to ask your system administrator how to fix this problem.

- **Problem:** When trying to open a source file using File → Open Source from DDD, no files appear in the list.
  **Possible cause:** You have not interrupted the execution. Press the Interrupt button in DDD’s command window.
  Another possible cause: The source files of the shared libraries have not been loaded. Press the button Load shared libraries in the Open Source dialog.

- **Problem:** Breakpoints in constructor or in the init() function are ignored.
  **Possible cause:** Breakpoints set in source files of shared libraries are not preserved from one run to another, therefore it is necessary to set them again. To set a breakpoint in the constructor it is possible to let SoC Designer Simulator first load the shared library and then prompt you before the constructor gets called. At that moment a breakpoint can be set.

- **Problem:** Every time DDD is started a dialog appears complaining about the version number.
  **Solution:** This problem can be fixed by modifying your init file in the .ddd directory in your home directory.
Chapter 10

The Cycle Accurate Debug Interface

CADI Interface Overview

The *Cycle Accurate Debug Interface* (CADI) provides debug access to memory, registers, or processors in a component. The CADI interface can be used to:

- Display the contents of registers and memory within SoC Designer Simulator for any type of component.
- Enable the cooperation of an existing debugger with SoC Designer. This can be particularly useful when integrating a model of a core with an established user base familiar with the debugger’s front end or if other third party debuggers are not available for this architecture.

SoC Designer Simulator enables the display of register and memory windows for any component that implements a CADI interface. This feature can be used for peripherals and other components that do not have their own dedicated debugger.

If, however, features like source-level debugging or stack view are required, a separate embedded software debugger must be used. If such a debugger exists, it can probably be used within SoC Designer. Contact support for more details on embedded software debuggers.

See the CADI section in the *ARM ESL API Developer’s Guide* for additional information on the classes and data structures used by the CADI API.
Using the CADI interface from SoC Designer Simulator

This section describes how provide debug access to simple components.

The Component Wizard step shown in Figure 196 enables you to specify registers and memory that are exposed to the debug interface.

Figure 196 Debug information dialog

See the ARM ESL API Developer’s Guide, and Chapter 3 for more details on component creation.

Special readDbg and writeDbg methods are automatically implemented by the Component Wizard. The following example shows how these generated methods enable access to memory-mapped registers associated with a transaction slave port:

**Register access from the debug functions**

```c
CASIStatus apb_TS::readDbg(uint64_t addr, uint32_t* value, uint32_t* ctrl) {
    uint64_t offset = addr - port_base_address;
    switch (offset) {
    case 0x0:
        value[0] = owner->reg0;
        break;
    case 0x4:
        value[0] = owner->reg1;
        break;
    default:
        return CASI_STATUS_NOTSUPPORTED;
    }
```
The values of the registers can be accessed from the debug functionality built into the SoC Designer Register Window. Figure 197 shows the Register Window for a component.

Figure 197 Viewing register contents from SoC Designer

Similar CADI functionality is provided for reading and writing memory in a component. SoC Designer has a Memory Window that uses the CADI interface to display memory contents.
Chapter 11

The Cycle Accurate Profiling Interface

CAPI Interface Overview

The *Cycle Accurate Profiling Interface* (CAPI) is used to gather customized profiling data and to enable the user to visualize that data in SoC Designer Simulator. It contains the following section:

The CAPI interface supports a generic implementation of profiling and enables the collection of different types of data that are organized around streams and channels of information.

To support profiling, a SoC Designer component must:

- implement and register the CAPI interface
- supply the type of information to profile
- gather the information and call `CAPIRecordEvents()`
- display the collected information in profile windows.

Profiling streams and channels

The CAPI interface implemented by a component has one or more profiling streams. Each stream contains one thread of information. For instance, a bus component allocates a profiling stream to each master to collect:

- grants
- conflicts
- reads
- writes

Each profiling stream is composed of one or more profiling channels. Each channel represents one item of information to be collected and placed in the stream.

The Cache component, for example, has a stream with a single channel that records the address and a second stream with a single channel that can record the operation type as one of:

- Read misses
- Read hits
- Write misses
- Write hits
The operation channels can be displayed graphically. Figure 198, for example, shows the SoC Designer Cache Profile window.

**Figure 198 Example of a profile window showing multiple channels**

Each profiling channel has:

- **name** — A string that contains the name for the recorded data.
- **type** — Describes the type of data to record (such as `CAPI_CHANNEL_TYPE_U16`).
- **description** — A string containing descriptive text on the channel.
- **symbol inf** — A pointer to a structure that gives the number of symbol values and the symbol names and colors to use when the profiling data is displayed. If a channel does not record symbolic data, this pointer is null.

Each stream has:

- **name** — a string that contains the name of the stream.
- **enabled flag** — a Boolean value that indicates whether the stream must record data for each of its channels.
- **event width** — a number indicating the total size of the data block that is used to hold the events for all channels that are part of the stream.
- **trace head and tail** — pointers to the head and tail of the linked list of trace segments for the stream.

Each trace segment has a pointer to the next trace segment in the stream and a pointer to the data array (see the CAPI section in the *ARM ESL API Developer’s Guide* for additional information of the profiling data structures). New trace segments are added if the current trace segment data array is filled.

- **trace size** — a number indicating the size of the array holding the channel data. If the maximum size is exceeded, a new trace segment is created and added to the linked list.

The component uses the `CAPIRecordEvent()` function to record events for the stream during run time. Each stream is independent and the recording proceeds as often as it required. For example, the first cycle might record three events for stream1 and the next cycle could record one event for stream2.

*Note:* Although streams are recorded independently, each event recorded for a stream must contain a data element for every channel included in that stream. If stream1 is com-
posed of channel 1 and channel 2, for example, every event recorded at runtime must provide a value for channel 1 and channel 2 in that stream.

Streams must be constructed to record events that occur at the same time. You could, for example, create a stream that records transaction type and transaction address because these are always related in time. Creating a stream with events from two different bus masters would not be practical because each master has events occurring independently of the other.
CASI Memory Map Interface overview

You can use the Cycle Accurate Simulation Interface - Memory Map Interface (CASIMMI) to modify the memory map of components connected to a bus master. Use the Memory Map Editor dialog from SoC Designer Canvas to edit the CASIMMI memory regions connected to a bus master port.

*Note:* See the ESL API Developer’s Guide for more details about the CASI MMI classes and interfaces.

*Figure 199* shows a master component named *CompBus* (with bus master port *bm1*) and slave components named *compslave1* (with slave port *ps1*) and *compslave2* (with slave port *ps2*) in the Memory Map Editor.

If the CASIMMI interface for a component is not detected, or if any of the components do not match the requirements, the editor functions are disabled and memory map changes must be done by setting the parameters of each individual component that is connected to the bus master.
The requirements for a slave component supporting the Memory Map Editor are:

- the CASI version string must be \texttt{CASI\_VERSION\_1\_1}
- the \texttt{getMappingConstraints} function must not return \texttt{NULL}

\textbf{SoC Designer Canvas Memory Editor Information Flow}

The following steps occur in SoC Designer Canvas to detect and support the memory editor:

1. At initialization SoC Designer Canvas calls \texttt{getMappingConstraints()} for all slave ports. SoC Designer Canvas uses the returned values to compute a set of initial regions to use for mapping slaves connected to the bus.

2. The bus master component registers itself with the Memory Map Editor by calling \texttt{CASIMMI::registerInterface()}. If this does not happen, SoC Designer Canvas assumes that this component does not support CASIMMI and the old style interface is used by calling \texttt{getAddressRegions()}.

3. On opening the Memory Map Editor dialog, the dialog sets up the memory map information:
   - If the memory map information is already present in the \texttt{.mxp} project file, the Memory Map Editor dialog uses that information.
   - If the information is not in the \texttt{.mxp} file, Memory Map Editor dialog calls \texttt{CASIMMI::requestBusAddressMaps()} on the registered MMI interface. The bus component (or bus master port) implements \texttt{requestBusAddressMaps()} to tell the Memory Map Editor dialog what are the map number and names it supports.

4. The user edits the supplied memory constraints from the connected slave port in the Memory Map Editor dialog. The editor validates the entries against any supplied constraints.

   The Memory Map Editor dialog shows multiple maps with the name of each map provided by the bus. Each such map can be edited by the user. The bus can be modeled so that it can support one map or another at run-time based, for example, on the state of the bus, on input signals, or writes to control registers.

5. The edited values are saved to the \texttt{.mxp} file.
**SoC Designer Simulator Information Flow**

The following steps occur in SoC Designer Simulator to detect and support the memory editor:

1. The bus component/port constructor calls `CASIMMI::registerInterface()`.
   
   If this does not happen, SoC Designer Simulator assumes the CASIMMI interface is not supported.

2. Simulator calls `setBusAddressMaps()` to tell the component and bus what information is present in the `.mxp` file. This function is called after the configure stage of simulation and before the initialization stage.

3. From the values passed from `setBusAddressMaps()`, the Bus Master Port sets up its Memory Table and also calls the `setAddressRegions()` function of the slave ports. The size of the last address region is 0 to indicate the end of the address region array.

   The functions are called every time the bus has a change in state.

4. If the information is not present in the `.mxp` file, the bus component calls the `getAddressRegions()` function in the slave (using the legacy style memory mapping based on parameter values).

**Using the Bus Master from the Component Wizard**

If you use the Component Wizard in SoC Designer Canvas to generate a component that contains a bus master or slave port, there is no requirement to modify the generated code unless you are using multiple memory maps for the same bus.

The CASIMMI base implementation contains default implementations of `getCurrentMemoryMaps()` and `setCurrentMemoryMap()`.

`getCurrentMemoryMaps()` and `setCurrentMemoryMap()` are advanced features and are reserved for use by the Simulator to manage components that contain multiple maps.

There are two typical usage scenarios for using for using CASIMMI:

- **Single map (default)**
  
  You do not have to modify the code if you are using the BusMasterPort that is supplied with SoC Designer.

  Instantiate the BusMaster Port class in SoC Designer:

  ```
  bm1_BMaster = new sc_port< CASITransactionIF, 0 > (this, "bm1", 0x1000, 0x10000 );
  ```

  This is what is done in the component wizard.

  The CASIMMI is predefined in the SoC Designer BusMaster Port and is called in the constructor of the BusMasterPort. This predefined CASIMMI has only one Map called `MapX`.

  The component user can add the different values for this Map for the different connected slaves from the Memory Map Editor. This is typically sufficient for most implementations.

- **Multiple maps**
  
  If you require a bus component with more than one Map (for example, Map1, Map2, and Map3), you must create a new class that inherits from `CASIMMI` class:

  ```
  CompMaster_CASIMMI : public CASIMMI
  ```
You must also implement the following functions:

- **Constructor** — This is where you call `registerinterface()`.
- **Destructor** — This is where the instance (and any allocated memory) is removed.
- **requestMemoryMaps** — This function modifies the referenced `CASIMMIMemoryMapRequest` structure.

```cpp
struct CASIMMIMemoryMapRequest {
    // num of addr maps supported
    uint32_t numMaps;
    // array of uint32_t w/ IDs of addr maps supported.
    uint32_t* memoryMapID;
    // array of strings naming addr maps supported
    std::string* mapNames;
    CASIMMIMemoryMapRequest& operator=(CASIMMIMemoryMapRequest& puRef);}
```

This is where you can specify the different maps:

```cpp
req->mapNames = new string[3];
req->mapNames[0] = "Map1";
req->mapNames[1] = "Map2";
req->mapNames[2] = "Map3";
```

### MME Map Values in Hierarchical Systems

If the bus master port is an external port (present at the top level and in the subsytem), the memory map information for the slave port is saved in the `.mxp` file that contains the slaves component. Since there is only one CASIMMI for that bus master port in `setMemoryMaps()`, SoC Designer sets the Memory Map Editor Maps values to include all maps from the sub-system and top-level system.

### Migrating existing systems to use the MME

This section describes how to migrate existing systems to use the Memory Map Interfaces that support the Memory Map Editor.

You can convert an existing system (.mxp file) to use the Memory Map Editor in two ways: automatically (see *Automatic conversion of a system to use the MME*) or interactively (see *Interactive conversion of system memory maps*).
Automatic conversion of a system to use the MME

To automatically convert an existing system (mxp file) to use the Memory Map Editor feature of SoC Designer:

1. Open your system in SoC Designer Simulator (see Figure 200).

![Figure 200 Example system](image)

Note: The automatic migration wizard in Simulator does not enable migrating designs that have use MME set to false and are not runtime editable. For these designs, you must open the mxp project file in Canvas and change the configuration there.

2. Press the MemMap button to automatically generate the mxp file (see Figure 201).

![Figure 201 MemMap button on main toolbar](image)

3. A dialog is displayed prompting to automatically generate a memory map (Figure 202). Click Yes.

![Figure 202 No Memory Map information error message](image)
4. The Save As dialog is displayed. Enter the name and destination of the newly created file and click **OK** (see Figure 203).

**Figure 203 Save as dialog**

![Save As dialog](image)

Note: If the constraints for the attached components do not permit a valid memory map to be generated automatically, you must use the manual migration process or use the Properties dialog for each of the individual attached components and change the memory start and size values.

You cannot generate a memory map if the busmaster does not have CASIMMI or the slaves do not satisfy the CASIMMI requirements.

5. If you use models that by default do not use the memory map (for example, AHB_casi, APB_casi, AXI2APB, and PL300):
   - Load the newly created system in the SoC Designer Canvas and set every **use MME** parameter to true. The **use MME** parameter only exists in bus master port components.
   - Save the modified system as an .mxs simulation file.

6. Run the generated system in the SoC Designer Simulator.
Interactive conversion of system memory maps

The Memory Map Editor dialog can be used to generate and configure an entire system memory map:

1. Open your system in SoC Designer Canvas.
2. Click the MemMap button to display the Memory Map Editor dialog.
3. The Memory Space drop-down list shows all components that support the MME in their BMP (see Figure 204). For the example shown, these are the AHB-Bus and APB-Bus.

![Figure 204 Memory space list](image)

4. Select one of the components in the Memory Space list. The Master Port field shows the bus master for the component.

5. The Address regions list shows slaves connected to the selected master port (see Figure 205). Select one of the slaves from the Address regions list.

![Figure 205 Memory Map Editor dialog with default values](image)
6. Click the **Edit** button to display the Edit Address Region dialog (see *Figure 206*). Use this dialog to change the start, end, size, and region name for the address region belonging to the selected component.

*Figure 206 Edit Address Region dialog for APB_Bus*

7. The text box at the bottom of the dialog lists the slave port constraints (see *Figure 207*). If your settings do not meet these constraints, the MME produces an error message.

*Figure 207 Parameters for the APB_Bus component*

8. Scroll the text to display additional parameters for the connected slave. You can copy and paste the base and size values from the text area or enter custom values.

To use the default values:

a) Select the value next to the **Base Port 1** parameter and copy it into the **Start** field in the **Address Region** box.
b) Select the value next to the Size parameter and copy it into the Size field in the Address Region box. The completed dialog resembles Figure 208.

**Figure 208 Copying the Start and Size parameters**

If you are entering custom values, use the base and size constraints in the text area to ensure that the values you enter are valid.

*Note:* You can use the underline character “_” to separate groups of digits. For example, 0xabcd_1234 is the same as 0xabcd1234.

9. Click OK to close the Edit Address Region dialog.

10. From the Memory Map Editor dialog, select the next region in the Address Region list and click Edit to display the Edit Address Region dialog again.

   Repeat to specify the base and size values for the remaining regions.
11. After editing the regions, the Memory Map Editor dialog looks similar to the one in Figure 209.

**Figure 209 Memory Map Editor with new values for AHB_Bus space**

12. After all Address Regions for the selected component have been assigned, select the next component from the Memory Space drop-down list.

13. Repeat to set the base and size values for each slave in the Address region list.

14. Click OK to close the Memory Map Editor.

If an error message like the one in Figure 210 is displayed, the address regions have not been correctly set. Examine the start and end address and modify them so that they do not overlap.

**Figure 210 Error message indicating overlapping address regions**

If an error message like the one in Figure 211 is displayed, the use MME parameter for the buses have not been set to true.

**Figure 211 Error message indicating that the use MME parameter is not set**
15. Activate the **use MME** parameter for each of the components listed in the Memory Map Editor **Memory Spaces** list. Select a component and then change the **use MME** value from the Parameter Window (see *Figure 212*).

**Figure 212 use MME parameter**

If you cannot see the **use MME** parameter, scroll down the parameter list. If the parameter is not available, check whether you’re using an up-to-date model.

16. After you have changed all **use MME** parameters to true, the system is ready. It now uses the MME information instead of the base and size values from the component parameters.

17. Verify that everything works by enabling the debug messages for the components. Every component now outputs the regions it is assigned to.

18. Save the system.

19. Simulate the system to verify that the migration has been completed successfully.
This chapter contains the following sections:

- Configuration File Overview and Content
- How SoC Designer Loads Configuration Files
- Working with configuration files
- Configuration File Syntax

Configuration File Overview and Content

SoC Designer Plus uses component library configuration (.conf) files to locate components. Your installation includes numerous configuration files, such as those for CPAKs, protocol models, and other components. You can also create your own configuration files.

As shown in Figure 213, a project’s top- and mid-level .conf files may use @include directives to point to component-level .conf files, which include the data needed to load the components.
Maxlib.conf files

Maxlib.conf files are top-level component configuration files created by the SoC Designer Runtime Generator or by Carbon Model Studio when creating a SoC Designer component. Maxlib.conf typically include references to the .conf files of each model in a system.

Note: ~/.carbon/SoCDesigner/<SoCD version>/maxlib.conf files are generated by SoC Designer. Their syntax is different from other configuration files, so do not use them as templates for modifications to other .conf files.
Component-level .conf files

Component-level .conf files have one or more component specifications with the following general structure:

```component "MyModel"
{
  type = "Other";
  version="1.0.0";
  path_linux = "./lib/libMyModel.so";
  release_with_debug_path_linux = "./lib/libMyModel.so";
  path_win32 = "./lib/Release/libMyModel.dll"
  release_with_debug_path_win32 = "./lib/Release_with_Debug/DebuglibMyModel.dll"
  documentation_file = ".../doc/guide_name.pdf"
}
```

All attributes specified for the component are optional, but a valid path must be specified in order to load the component.

How SoC Designer Loads Configuration Files

When you launch SoC Designer Canvas or SoC Designer Simulator, these programs automatically reference and load configuration files in a particular order. This section describes:

- The location of referenced .conf files and the order in which they are loaded.
- How SoC Designer handles duplicate references to the same component across .conf files.

.Conf file location and order referenced

**Order of .conf files referenced for Linux:**

1. Configuration files specified at the command line when launching SoC Designer Canvas or SoC Designer Simulator (`sdsim/sdcanvas -b <config_1> -b <config_2> ... -b <config_n>`) take precedence over other Configuration files:

2. The `maxlib.conf` file in the SoC Designer installation directory (`~/.carbon/SoCDesigner/<SoCD version>/maxlib`).

3. The `maxlib.conf` file located in the directory in which you execute SoC Designer Canvas or SoC Designer Simulator (`./maxlib.conf`).

4. The `maxlib.conf` file at `~/.maxlib.conf`.

**Order of .conf files referenced for Windows:**

On Windows, SoC Designer Canvas and SoC Designer Simulator load the components in the `maxlib.conf` file located in same directory as the `.mxf` file you are launching. The configuration file location is specified in the following Windows Registers location:

- `HKEY_CURRENT_USER>Software/Carbon>Carbon Design Systems > SoC Designer > <SoCD version> > maxlib`.
Fine-grained path specifications take precedence over more general path specifications. For example, when using SoC Designer for Visual C++ 2013 on Windows, the value specified by `path_win32_2013` overrides any value specified by `path` or `path_win32`.

**Handling of duplicate components within a .conf file**

SoC Designer Canvas behavior

If the same component is specified in more than one configuration file, SoC Designer Canvas loads only the first occurrence of the component. If SoC Designer Canvas encounters a second occurrence of a component in the configuration files, it generates a warning of the following type, and does not load the file:

```
WARNING :: Component 'AXIv2_Slave' from 'C:\Program Files (x86)\Carbon\SoCDesigner\version>\Protocols\AXIv2\etc\AXIv2Components.conf' will be ignored - duplicate exists
```

SoC Designer Simulator behavior

If SoC Designer Simulator encounters a second occurrence of a component in the configuration files, it does not load the file. No warning is generated.

**Working with configuration files**

There are many approaches to configuring your development environment so that your project loads the appropriate components. This section presents two approaches:

- **Creating a project-specific .conf file** – in other words, a consolidated configuration file that loads all components required for a particular project (.mxp file). This approach is intended for development environments that support both Linux and Windows. See the section "Creating a configuration file for a cross-platform environment."

- **Creating an operating system-specific .conf file**. This method is intended for development environments that are exclusively Linux, or exclusively Windows. See the section "Creating a single configuration file per platform (OS)."

**Creating a configuration file for a cross-platform environment**

This section describes creating a single configuration file that loads the components required by a project, such as an ARM CPAK, that runs on both Linux and Windows.

To create a single configuration file:

1. Determine which .conf files include components that are essential to your project.
2. Concatenate the essential configuration files into a single configuration file.
3. Edit the consolidated configuration file so that only required content (type, version, variant, paths, and documentation reference) is included.

The following section provides an example of this procedure.

**Example of creating a project-specific .conf file**

The example project used in the section (My_Project.mxp) includes the following components:

- Cortex™-A9 Processor
- NIC-301 interconnect
• Generic memory

Step 1: Determining essential files: The project My_Project.mxp includes the following .conf files for the three components:

MODELS/CortexA9MP_r2p2/CortexA9MP4/gcc/maxlib.libCORTEXA9MP4.conf
MODELS/CortexA9MP_r2p2/CortexA9MP4/vc2013/maxlib.libCORTEXA9MP4.conf
MODELS/CortexA9MP_r2p2/CortexA9MP4/vc2013/maxlib.libCORTEXA9MP4.windows.conf
MODELS/nic301/nic301_pv_c/gcc/maxlib.libnic301_pv.conf
MODELS/nic301/nic301_pv_c/vc2013/maxlib.libnic301_pv.conf
MODELS/nic301/nic301_pv_c/vc2013/maxlib.libnic301_pv.windows.conf
MODELS/memory/mem/gcc/maxlib.genericmem.conf
MODELS/memory/mem/vc2013/maxlib.genericmem.conf
MODELS/memory/mem/vc2013/maxlib.genericmem.windows.conf

Inspection of these files indicates that the following versions of the .conf files reference components required by My_Project.mxp:

gcc/maxlib.component.conf
vc2013/maxlib.component.windows.conf

However, the environment is set up so that the memory component is included by protocols.conf in the user preferences. This means that the memory component .conf files can be excluded from the single configuration file you are creating.

Step 2: Concatenating essential content: Concatenate the essential versions of the .conf files for the Cortex-A9 and NIC-301:

cat \ MODELS/CortexA9MP_r2p2/CortexA9MP4/gcc/maxlib.libCORTEXA9MP4.conf \ MODELS/CortexA9MP_r2p2/CortexA9MP4/vc2013/maxlib.libCORTEXA9MP4.windows.conf \ MODELS/nic301/nic301_pv_c/gcc/maxlib.libnic301_pv.conf \ MODELS/nic301/nic301_pv_c/vc2013/maxlib.libnic301_pv.windows.conf \ > My_Project.conf

The content of the resulting My_Project.conf file is as follows:

component "CORTEXA9MP4"
{
  type = "Core";
  version = "4.1.1-r2p2";
  variant = "";
  release_with_debug_path = "./libCORTEXA9MP4.mx_DBG.so";
  path = "./libCORTEXA9MP4.mx.so";
  documentation_file = "CML_CortexA9_User.pdf";
}

component "CORTEXA9MP4"
{
  type = "Core";
  version = "4.1.1-r2p2";
  variant = "";
  release_with_debug_path = "./libCORTEXA9MP4.mx_DBG.dll";
  path = "./libCORTEXA9MP4.mx.dll";
  documentation_file = "CML_CortexA9_User.pdf";
}
Step 3: Editing the consolidated .conf file: Edit My_Project.conf, consolidating all path data into a single component section for each component. Type, version, variant, and documentation_file are the same across files, so include them only once in each section.

In addition to consolidating path data, you must also:

- Prepend the directory path (for example, ./MODELS/CortexA9MP_r2p2/CortexA9MP4/gcc) to each .so and .dll path.
- Prepend the directory path to each documentation_file path. Model guide PDFs are the same across platforms, so you may use either the gcc or vc2013 path.
- Add the name of the operating system as a suffix to the keyword for path. For example:

  path = becomes:
  - path_linux =,
  - path_win32_vc2013 =, and
release_with_debug_path = becomes:

- release_with_debug_path_linux =
- release_with_debug_path_win32_2013 =

After you make the above modifications, the final content of My_Project.conf is as follows. Modifications are highlighted in red:

component "CORTEXA9MP4"
{
    type = "Core";
    version = "4.1.1-r2p2";
    variant = "";
    release_with_debug_path_linux = "./MODELS/CortexA9MP_r2p2/CortexA9MP4/gcc/libCORTEXA9MP4.mx_DBG.so";
    path_linux = "./MODELS/CortexA9MP_r2p2/CortexA9MP4/gcc/libCORTEXA9MP4.mx.so";
    documentation_file = "./MODELS/CortexA9MP_r2p2/CortexA9MP4/vc2013/CML_CortexA9_User.pdf";
}

cOMPONENT "nic301_pv"
{
    type = "Bus";
    version = "4.0.0-r2p0";
    variant = "";
    release_with_debug_path_linux = "./MODELS/nic301/nic301_pv_c/gcc/libnic301_pv.mx_DBG.so";
    path_linux = "./MODELS/nic301/nic301_pv_c/gcc/libnic301_pv.mx.so";
    release_with_debug_path_win32_vc2013 = "./MODELS/nic301/nic301_pv_c/vc2013/libnic301_pv.mx_DBG.dll";
    documentation_file = "./MODELS/nic301/nic301_pv_c/vc2013/CML_NIC301_User.pdf";
}

When SoC Designer loads My_Project.conf, the components for Linux and Windows vc2013 are loaded automatically.
Creating a single configuration file per platform (OS)

The examples below are for a sample SoC Designer project (.mxp) that includes the following components:

- Cortex™-A9 Processor
- NIC-301 interconnect
- Generic memory

Create a .conf file for the project that uses @include directives for each required component.

The sample environment is set up so that the memory component is included by protocols.conf in the user preferences. This means that you can exclude the @include for the memory component .conf file from the configuration file you are creating.

Configuration file for Linux

#This is a Linux .conf file named MyProject_Linux.conf
@include "path/to/maxlib.libCORTEXA9MP4.conf"
@include "path/to/nic301.conf"

When SoC Designer loads MyProject_Linux.conf, the Linux versions of the Cortex-A9 and NIC-301 components are loaded automatically.

Configuration file for Windows

#This is a Windows .conf file named MyProject_Windows.conf
@include "path/to/maxlib.libCORTEXA9MP4.windows.conf"
@include "path/to/nic301.windows.conf"

When SoC Designer loads MyProject_Windows.conf, the Windows versions of the Cortex-A9 and NIC-301 components are loaded automatically.

Configuration File Syntax

When working with .conf files, refer to the following syntax rules:

- Comment lines begin with the pound (#) sign.
- The @include directive can reference a full path or a relative path to the location of the configuration file. It may also reference environment variables. For example:
  
  @include "$MAXSIM_PROTOCOLS/etc/Protocols.conf"

The component section of the configuration file supports the path options described in the following tables:

Table 7 Config file component path directive order for non-debug simulations

<table>
<thead>
<tr>
<th>Windows / VS2013</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>path_win32_2013</td>
<td>path Linux</td>
</tr>
<tr>
<td>path_win32</td>
<td>path</td>
</tr>
<tr>
<td>path</td>
<td></td>
</tr>
</tbody>
</table>
Table 8 Config file component path directive order for debug simulations

<table>
<thead>
<tr>
<th>Windows / VS2013</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>release_with_debug_path_win32_2013</td>
<td>release_with_debug_path_linux</td>
</tr>
<tr>
<td>release_with_debug_path_win32</td>
<td>release_with_debug_path</td>
</tr>
<tr>
<td>release_with_debug_path</td>
<td></td>
</tr>
</tbody>
</table>

*a. SoC Designer may use the non-debug simulation paths if a “release with debug” path is not specified.*
Appendix B

Keyboard Shortcuts

This appendix lists keyboard shortcuts. It has the following sections:

- **SoC Designer Canvas shortcuts**
- **SoC Designer Simulator shortcuts**

### SoC Designer Canvas shortcuts

Table 9 lists keyboard shortcuts for SoC Designer Canvas.

**Table 9** SoC Designer Canvas shortcuts

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + N</td>
<td>File → Create new System</td>
</tr>
<tr>
<td>Ctrl + O</td>
<td>File → Open existing System</td>
</tr>
<tr>
<td>Ctrl + S</td>
<td>File → Save the current system</td>
</tr>
<tr>
<td>Ctrl + F4</td>
<td>File → Close the currently open system</td>
</tr>
<tr>
<td>Ctrl + Q</td>
<td>File → Exit</td>
</tr>
<tr>
<td>Ctrl + Z</td>
<td>Edit → Undo</td>
</tr>
<tr>
<td>Ctrl + Y</td>
<td>Edit → Redo</td>
</tr>
<tr>
<td>Ctrl + X</td>
<td>Edit → Cut</td>
</tr>
<tr>
<td>Ctrl + C</td>
<td>Edit → Copy</td>
</tr>
<tr>
<td>Ctrl + V</td>
<td>Edit → Paste</td>
</tr>
<tr>
<td>Ctrl + D</td>
<td>Edit → Duplicate</td>
</tr>
<tr>
<td>Del</td>
<td>Edit → Delete selected object</td>
</tr>
<tr>
<td>Ctrl + A</td>
<td>Edit → Select All</td>
</tr>
<tr>
<td>Ctrl + Shift + S</td>
<td>Edit → Switch to Edit Mode</td>
</tr>
<tr>
<td>Ctrl + Shift + C</td>
<td>Edit → Switch to Connect Ports Mode</td>
</tr>
<tr>
<td>Ctrl + G</td>
<td>View → Toggle Grid Display</td>
</tr>
<tr>
<td>Ctrl + ‘+’</td>
<td>View → Zoom in</td>
</tr>
</tbody>
</table>
### Table 9: SoC Designer Canvas shortcuts (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + '-'</td>
<td>View → Zoom out</td>
<td></td>
</tr>
<tr>
<td>Ctrl + 1</td>
<td>View → Zoom to 100%</td>
<td></td>
</tr>
<tr>
<td>Ctrl + F</td>
<td>View → Zoom to Fit</td>
<td></td>
</tr>
<tr>
<td>Ctrl + R</td>
<td>View → Center on Selected Objects</td>
<td></td>
</tr>
<tr>
<td>Ctrl + I</td>
<td>View → Center Diagram</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>Simulate System</td>
<td></td>
</tr>
<tr>
<td>Ctrl + Shift + F5</td>
<td>Simulate System in Debug Mode</td>
<td></td>
</tr>
</tbody>
</table>

### SoC Designer Simulator shortcuts

Table 10 lists keyboard shortcuts for SoC Designer Simulator.

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + O</td>
<td>File → Open existing System</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + S</td>
<td>File → Save the current system</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + F4</td>
<td>File → Close the currently open system</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + Q</td>
<td>File → Exit</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + H</td>
<td>View → Simulation Hierarchy</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + G</td>
<td>View → Toggle Grid Display</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + '+'</td>
<td>View → Zoom in</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + '-'</td>
<td>View → Zoom out</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + 1</td>
<td>View → Zoom to 100%</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + F</td>
<td>View → Zoom to Fit</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + R</td>
<td>View → Center on Selected Objects</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + I</td>
<td>View → Center Diagram</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + Shift + F5</td>
<td>Control → Restart Simulation (Hard Reset)</td>
<td>Any window</td>
</tr>
<tr>
<td>Ctrl + Shift + R</td>
<td>Control → Reset Simulation</td>
<td>Any window</td>
</tr>
<tr>
<td>F5</td>
<td>Control → Run</td>
<td>Any window</td>
</tr>
<tr>
<td>Shift + F5</td>
<td>Control → Stop</td>
<td>Any window</td>
</tr>
<tr>
<td>Ctrl + F11</td>
<td>Control → Step one Cycle</td>
<td>Any window</td>
</tr>
</tbody>
</table>
### Table 10: SoC Designer Simulator shortcuts (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + Shift + F11</td>
<td>Control → Step n Cycles</td>
<td>Any window</td>
</tr>
<tr>
<td>Ctrl + B</td>
<td>Debug → Breakpoint Manager</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + W</td>
<td>Debug → Toggle Waveform Viewer</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + Shift + F12</td>
<td>Debug → Refresh External Debuggers</td>
<td>Any window</td>
</tr>
<tr>
<td>Ctrl + A</td>
<td>Select All</td>
<td>Main window</td>
</tr>
<tr>
<td>Ctrl + D</td>
<td>Show Diagram/Show Open Windows</td>
<td>Any window</td>
</tr>
<tr>
<td>Ctrl + E</td>
<td>Bring Focus To Main Window</td>
<td>Any window</td>
</tr>
<tr>
<td>Ctrl + Tab</td>
<td>Bring next Runtime Window in Foreground</td>
<td>Any window</td>
</tr>
<tr>
<td>F10</td>
<td>Instruction Step Over (skipping calls)</td>
<td>Disassembly window</td>
</tr>
<tr>
<td>F11</td>
<td>Instruction Step Into</td>
<td>Disassembly window</td>
</tr>
<tr>
<td>Ctrl + F5</td>
<td>Run to Debuggable Point</td>
<td>Disassembly window</td>
</tr>
<tr>
<td>Left/Right Arrow</td>
<td>Move cursor to previous/next event</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Home/End</td>
<td>Move cursor to first/last event</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl + Left/Right</td>
<td>Move cursor to previous/next cycle</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl + Home/End</td>
<td>Move cursor to first/last cycle</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Up/Down Arrow</td>
<td>Select previous/next channel</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Alt + Up/Down Arrow</td>
<td>Move selected channel up/down one position</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Plus/Minus</td>
<td>Show/Hide sub-channels (of selected channel)</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Del</td>
<td>Remove current channel or sub-channel from waveform viewer</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>L/R</td>
<td>Insert left/right locator at cursor position</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Shift + L/R</td>
<td>Remove left/right locator</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Key</td>
<td>Action</td>
<td>Scope</td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Ctrl + L/R</td>
<td>Move cursor to left/right locator</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl +/Minus</td>
<td>Zoom in/out</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl+Shift +/-Minus</td>
<td>Zoom to next/previous setting (zoom redo/undo)</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl + F</td>
<td>Zoom to fit</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl + A</td>
<td>Zoom to range defined by the left and right locators</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>Ctrl + 1</td>
<td>Reset zoom to 1 cycle per pixel</td>
<td>Waveform Viewer</td>
</tr>
<tr>
<td>C</td>
<td>Center view on cursor</td>
<td>Waveform Viewer</td>
</tr>
</tbody>
</table>
Appendix C

Creating a SoC Designer Runtime

This appendix contains the following sections:

- Overview of SoC Designer Runtime
- Creating the binary platform files

Overview of SoC Designer Runtime

You can use SoC Designer to generate a platform that can be distributed to customers to run with the SoC Designer Run Time Only Environment Simulator. The option to generate platforms for IP proliferation that can be delivered to customers allows customers to run a simulation of this platform by purchasing only the SoC Designer Run Time Only Environment, rather than the full SoC Designer Suite.

Embedded software developers can use the SoC Designer Run Time Only Environment for platform configuration, simulation, and debug of the application executing on the processors.

The SoC Designer Run Time Only Environment enables full system-level and component-level debugging on the platform.

The SoC Designer Run Time Only Environment supports all the features of SoC Designer Simulator except for host-level debugging of components. Host-level debugging is not supported because runtime platform customers develop software, not systems. They have no requirements, therefore, to debug component models. Component design must be done with SoC Designer Canvas.

A FLEXnet license is required to run the SoC Designer Run Time Only Environment and any third-party models.

Figure 214 provides a graphical view of the process of developing a platform model.
The steps to create and distribute a new platform are:

1. If necessary, obtain required third-party models and licenses from your supplier.

2. Generate the Runtime system from SoC Designer. The system might use a mixture of locally-created and third-party models.

3. Send the generated platform and the associated tools and platform licenses to the end customer.

4. If some of the models are licensed directly from the supplier, the customer must also obtain the models and licenses from the third-party supplier.

5. The customer can now use SoC Designer Runtime to run the encrypted simulation file.
Creating the binary platform files

This section describes the steps required to create and distribute the binary platform files. It contains the following sections:

- Prerequisites
- Preparing the system
- Requesting licenses
- Delivering the system to the end customer
- Distributing platforms containing third-party models
- Support for end customers

Prerequisites

To use the platform option of SoC Designer, the platform generator must be licensed. The platform generator license is a group license and can be used by any SoC Designer user.

Preparing the system

To create a directory containing the platform files:

1. From the Tools menu in SoC Designer Canvas select SoC Designer Generator. This creates an .mxe file with the complete system information. The .mxe file is equivalent to an .mxp file, except that it is in an encrypted form.

   A prompt is displayed to enter the directory in which to place the SoC Designer Run Time Only Environment files.

2. Enter the directory name.

3. The Platform Generation dialog lists all items that are copied into the IP proliferation directory (see Figure 215).

   Figure 215 SoCD Runtime Generation dialog

4. Use the check box to select or deselect particular items.

5. Click OK and SoC Designer copies the selected items to the specified directory.
Note: You can also create the .mxe file using a command line option with SoC Designer. The -O option allows you to generate an .mxe file from a specified .mxp file. For example:

    sdcanvas -O project.mxe project.mxp

Using this method, there is no way to deselect particular items; all the libraries in the system are included in the .mxe file.

A summary dialog is displayed showing where the information was placed. Click OK to close the dialog.

The IP proliferation directory contains:

• a release sub-directory that contains the specified libraries (.so or .dll files)
• a Component Library file (.conf files)
• an .mxe file for the system

Note: The platform package must only include libraries for non-licensed components that are delivered directly to the end-customer. The list displayed in the summary dialog must not include any licensed models that have been provided by ARM or a third-party in a separate package. The customer must obtain the licensed models from the model owner.

**Requesting licenses**

If the end customer does not yet have licenses for the SoC Designer Run Time Only Environment and the required Component Library models, contact your sales representative for additional information on licensing models in the platform space.

Note: Unless new third-party (or ARM) models are being used, no new licenses are needed if a changed platform is delivered to a customer that has already used the SoC Designer Run Time Only Environment.
Delivering the system to the end customer

To use the generated system, the following components must be delivered to the end customer:

- SoC Designer Run Time Only Environment installation package. You can download the latest version from the ARM support web site.
- licenses
- MXE system file (.mxe extension)
- component libraries (.so or .dll files)
- the Component Library library file

The last three items are generated in the directory specified by the designer. If additional files are required, they must be bundled with the package.

*Note:* ARM recommends that you create a combined package (for example, a .zip file) that contains all of the model files and the associated tools. This ensures that the distributed platform runs properly on other computers.

Distributing platforms containing third-party models

When proliferating platforms that contain third-party models (for example MIPS or ARM models), you must distribute these separately to the end customer. (This is in addition to the IP proliferation package and the SoC Designer Run Time Only Environment software).

The platform proliferation license does not include licenses for Component Library processor models or third-party tools:

- Some models must be delivered to the customer directly from the model creator rather than including the model with the platform package.
- The required third-party debuggers must be delivered separately.
- The IP licenses must be obtained separately.
Support for end customers

The SoC Designer licensee (the platform provider who generated the system file for the platform) provides first-line technical support.

The support division of ARM provides second-line technical support.

Note that SoC Designer provides a simple way to produce a document with essential information for a specific error, for use by technical support, as follows:

1. In SoCD Canvas, select Help → About System… to display the About System dialog.
2. Select the entire contents of the About System text box that displays, and copy it into an empty document.
3. Save the document.
4. Contact your first-line support person, and after you are authorized to contact technical support at ARM, send a copy of the document you have saved.

This procedure is designed to improve communication and speed up the processing of the reported error.
Appendix D
Third-Party Software

ARM acknowledges and thanks the respective owners for the following software that is used by our product:

• **Verilog Parser**

**Verilog Parser**

```
/*
 * Copyright (c) 1992, Bakul Shah <bvs@BitBlocks.com>
 * All Rights Reserved.
 *
 * Permission to use, copy, modify, distribute or sell this software and
 * its documentation for any purpose is hereby granted without fee, subject
 * to the following conditions.
 *
 * 1. The above copyright, this list of conditions and the following
 *    disclaimer must appear UNCHANGED in all copies of the software and
 *    related documentation, and their derivative works or modified
 *    versions
 *
 * 2. Binary distribution must be accompanied by documentation that
 *    reproduces the above copyright, this list of conditions and the
 *    following disclaimer.
 *
 * 3. Any modifications to the source code must be clearly marked as such.
 *
 * 4. The author's name may not be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 */
```
* This software is provided by the author AS IS. The author DISCLAIMS
* any and all warranties of merchantability and fitness for a particular
* purpose. In NO event shall the author be LIABLE for any damages
* whatsoever arising in any way out of the use of this software.
*/
This glossary describes some of the terms used in SoC Designer documentation.

**Advanced eXtensible Interface (AXI)**

A bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure. The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

**Advanced High-performance Bus (AHB)**

A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

*See also* Advanced Microcontroller Bus Architecture and AHB-Lite.

**Advanced Microcontroller Bus Architecture (AMBA®)**

A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

**Advanced Peripheral Bus (APB)**

A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

**AHB**

See Advanced High-performance Bus.
**AHB-Lite**

A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.

**Aligned**

A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

**AMBA**

See Advanced Microcontroller Bus Architecture.

**APB**

See Advanced Peripheral Bus.

**AXI**

See Advanced eXtensible Interface.

**AXI channel order and interfaces**

The block diagram shows:

- the order in which AXI channel signals are described
- the master and slave interface conventions for AXI components.
**AXI Terminology**

The following AXI terms are general. They apply to masters and slaves:

- **Active read transaction** — A transaction for which the read address has transferred, but the last read data has not yet transferred.

- **Active transfer** — A transfer for which the xVALID handshake has asserted, but for which xREADY has not yet asserted. The letter x in the signal name denotes an AXI channel as follows:
  - AW — Write address channel.
  - W — Write data channel.
  - B — Write response channel.
  - AR — Read address channel.
  - R — Read data channel.

- **Active write transaction** — A transaction for which the write address or leading write data has transferred, but the write response has not yet transferred.

- **Completed transfer** — A transfer for which the xVALID/xREADY handshake is complete.

- **Payload** — The non-handshake signals in a transfer.

- **Transaction** — An entire burst of transfers, comprising an address, one or more data transfers and a response transfer, writes only.

- **Transmit** — An initiator driving the payload and asserting the relevant xVALID signal.

- **Transfer** — A single exchange of information. That is, with one xVALID/xREADY handshake.

The following AXI terms are master interface attributes. To obtain optimum performance, they must be specified for all components with an AXI master interface:

- **Combined issuing capability** — The maximum number of active transactions that a master interface can generate. This is specified instead of write or read issuing capability for master interfaces that use a combined storage for active write and read transactions.

- **Read ID capability** — The maximum number of different ARID values that a master interface can generate for all active read transactions at any one time.

- **Read ID width** — The number of bits in the ARID bus.

- **Read issuing capability** — The maximum number of active read transactions that a master interface can generate.

- **Write ID capability** — The maximum number of different AWID values that a master interface can generate for all active write transactions at any one time.

- **Write ID width** — The number of bits in the AWID and WID buses.

- **Write interleave capability** — The number of active write transactions for which the master interface is capable of transmitting data. This is counted from the earliest transaction.

- **Write issuing capability** — The maximum number of active write transactions that a master interface can generate.
The following AXI terms are slave interface attributes. To obtain optimum performance, they must be specified for all components with an AXI slave interface.

- **Combined acceptance capability** — The maximum number of active transactions that a slave interface can accept. This is specified instead of write or read acceptance capability for slave interfaces that use a combined storage for active write and read transactions.
- **Read acceptance capability** — The maximum number of active read transactions that a slave interface can accept.
- **Read data reordering depth** — The number of active read transactions for which a slave interface can transmit data. This is counted from the earliest transaction.
- **Write acceptance capability** — The maximum number of active write transactions that a slave interface can accept.
- **Write interleave depth** — The number of active write transactions for which the slave interface can receive data. This is counted from the earliest transaction.

**Beat**

Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

*See also* Burst.

**Burst**

A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AMBA are controlled using signals to indicate the length of the burst and how the addresses are incremented.

*See also* Beat.

**Byte**

An 8-bit data item.

**Byte lane strobe**

A signal that is used for unaligned or mixed-endian data accesses to determine which byte lanes are active in a transfer. One bit of this signal corresponds to eight bits of the data bus.

**Component**

An individual core, memory, bus, etc. A component can also be an entire subsystem. In this case, the subsystem is placed into another system as a component.

**Connection**

A link between two components by using one master and one slave port. There are three types of connections: clock, signal, and transaction.
Direct Memory Access (DMA)
An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.

DMA
See Direct Memory Access.

External port
A port that is used to connect the current system within another system.

Halfword
A 16-bit data item.

High-Performance Matrix (HPM)
A highly configurable auto-generated AMBA 3 bus subsystem, based around a high-performance AXI cross-bar switch known as the AXI bus matrix, and extended by AMBA infrastructure components.

Host
A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

HPM
See High-Performance Matrix.

Internal port
A port of an individual component, used to establish connections between the component and other components/external ports within the system.

IP-XACT
A standard provided by the SPIRIT consortium. IP-XACT-compliant IP descriptions store configuration and interconnection information for the IP block in XML files.

Label
An annotation in the Diagram window (part of SoC Designer Canvas application).

Master
A component or port that initiates a transaction or drives a signal.

CADI
ESL API Debug Interface, enables reading and writing memory and register values and also provides the interface to external debuggers.
CASIMMI
SoC Designer Memory Map Interface, used to define and use memory maps for bus master components.

CAPI
ESL API Profiling Interface, enables collecting historical data from a component and displaying the results in various formats.

CASI
ESL API Simulation Interface, is based on the SystemC communication library and manages the interconnection of components and communication between components.

SoC Designer Component Library (MaxLib)
The repository of SoC Designer components.

Mega Model
A SoC Designer system consisting of one or more components.

Object
An object is a component, a connection, an external port, or a label.

Probe
Object inserted on a connection for the purpose of, for example, tracing, monitoring, breakpoints.

Region
A partition of instruction or data memory space.

Remapping
Changing the address of physical memory or devices after the application has started executing. This is typically done to permit RAM to replace ROM when the initialization has been completed.

Runtime
SoC Designer runtime is an environment for running binary versions of system models. These models cannot be modified by the user and the source for the component libraries is not provided.

Signal connection
The signal-based interface is very close to hardware simulators in that it simulates every signal independently.

Slave
A component or port that responds to a transaction or signal.
SPIRIT

(Structure for Packaging, Integrating and Re-using IP), a consortium that promotes the IP-XACT standard for IP interconnection.

Simulation run

This term is used for the combination of loading, running, and closing a system.

System

A system is a collection of components connected to collaborate in a simulation.

Transaction connection

The transaction-based interface encapsulates a group of signals into one read or write transaction.

Unaligned

A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

Word

A 32-bit data item.