

DesignWare®
ARM® Processors
SystemC™ Library
User Guide

Version U-2003.03-SP1, June 2003

Comments?

E-mail your comments about Synopsys
documentation to doc@synopsys.com

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2003 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

“This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____.”

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CoCentric, COSSAP, CSim, DelayMill, Design Compiler, DesignPower, DesignWare, Device Model Builder, EPIC, Formality, HSPICE, Hypermodel, I, iN-Phase, InSpecs, in-Sync, LEDA, MAST, Meta, Meta-Software, ModelAccess, ModelExpress, ModelTools, PathBlazer, PathMill, Photolynx, Physical Compiler, PowerArc, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SmartLogic, SNUG, SolvNet, Stream Driven Simulator, SiVL, Superlog, System Compiler, Testify, TetraMAX, TimeMill, TMA, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

abraCAD, abraMAP, Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAI, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Expert *Plus*, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler, Direct RTL, Direct Silicon Access, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FormalVera, FoundryModel, FPGA Compiler II, FPGA *Express*, Frame Compiler, Frameway, Galaxy, Gatrane, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, iQBus, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, LRC, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, NanoSim, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLint, OpenVera, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, Progen, Prospector, Proteus OPC, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-Sim XT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-OPC, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, The Power in Semiconductors, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

DesignSphere, MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
AMBA is a trademark of ARM Limited. ARM is a registered trademark of ARM Limited.
All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.

Document Order Number: 37903-000 QA
DesignWare ARM Processors SystemC Library User Guide, vU-2003.03-SP1

Contents

What's New in This Release	v
About This Manual	vi
Related Documentation	vii
Customer Support	x
 1. Setup and Configuration	
The DesignWare ARM Processors SystemC Library	1-2
Setting up and Configuring the Library	1-2
Prerequisites	1-2
Setting Up the ARM Environment	1-3
Obtaining the ARM Development Tools	1-4
Required License Features	1-4
Configuring the Processor Model and the Design	1-5
 2. Using the DesignWare ARM Processor Models	
The DesignWare ARM Processor Model.	2-2
Using the Processor Model	2-3

Constructor Arguments	2-5
DW_arm926ejs Processor Model	2-5
DW_arm946es Processor Model	2-7
Design Parameters	2-9
Using H Clock Enable	2-10
Changing TCM Sizes	2-11
Changing Cache Sizes	2-12
Enabling Semihosting	2-13
Starting a Simulation	2-13
Loading a Program From ADU	2-15
ARM926 MultiLayer AHB and the Interconnect Matrix	2-16
Configuring the Interconnection Matrix	2-16
Using the ARM Software Debugger With System Studio	2-17
Debugger Interface to Memory	2-18
Memory Interface Prioritization for Debug Accesses	2-18
Tightly Coupled Memory Models	2-19
Constructor	2-20
Running armsd or ADU Without an Executable	2-21
Example Designs	2-22
Troubleshooting	2-22

Index

Preface

This preface includes the following sections:

- What's New in This Release
- About This Manual
- Customer Support

What's New in This Release

Information about new features, enhancements, and changes; known problems and limitations; and resolved Synopsys Technical Action Requests (STARs) is available in the *DesignWare ARM Processors SystemC Library Release Notes* in SolvNet.

To see the *DesignWare ARM Processors SystemC Library Release Notes*,

1. Go to the Synopsys Web page at <http://www.synopsys.com> and click SolvNet.

2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)
3. Click Release Notes in the Main Navigation section, find the U-2003.03-SP1 Release Notes, then open the *DesignWare ARM Processors SystemC Library Release Notes*.

About This Manual

This manual describes the ARM926EJ-S and ARM946E-S transaction-level processor models for SystemC.

Audience

The *DesignWare ARM Processors SystemC Library User Guide* is written for system designers, software engineers, and electronics engineers designing systems who use these DesignWare models for modeling and simulation with CoCentric System Studio.

Users who are new to System Studio should read the *Getting Started With CoCentric System Studio* manual. Intermediate-level users will want to read the *CoCentric System Studio User Guide*. Experienced System Studio users or users who are looking for more detailed technical information should also consult the *CoCentric System Studio Reference Manual*.

The System Studio documentation suite consists of the following manuals:

- *Getting Started With CoCentric System Studio*
- *CoCentric System Studio User Guide*
- *CoCentric System Studio Reference Manual*
- *CoCentric System Studio HDL CoSim User Guide*
- *CoCentric System Studio VirSim User Guide*
- *CoCentric System Studio Model Guide*
- *CoCentric System Studio Reference Design Kits Guide*
- *CoCentric System Studio DSP Developer Kits User Guide*
- *CoCentric System Studio Filter Design Tools User Guide*

Related Documentation

You may wish to refer to the following documents for additional information:

- *DesignWare AMBA SystemC Library User Guide*

For additional information about DesignWare ARM Processors SystemC Library, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys Electronic Software Transfer (EST) system
- Documentation on the Web, which is available through SolvNet at <http://solvnet.synopsys.com>

- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at <http://mediadocs.synopsys.com>

The following documents are obtainable from ARM Limited:

- *ARM Developer Suite, Version 1.1, Debuggers Guide.*
ARM Document # ARM DUI 0066C
- *ARM926EJ-S, (Rev 0), Technical Reference Manual.*
ARM Document # ARM DDI0198B
- *ARM946E-S, (Rev 1), Technical Reference Manual.*
ARM Document # ARM DDI0201A
- *ARM Developer Suite, Version 1.2, Debug Target Guide.*
ARM Document # ARM DU10058D

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
<code>Courier</code>	Indicates command syntax.
<i>Courier italic</i>	Indicates a user-defined value in Synopsys syntax, such as <i>object_name</i> . (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
Courier bold	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[]	Denotes optional parameters, such as <i>pin1 [pin2 ... pinN]</i>
	Indicates a choice among alternatives, such as <i>low medium high</i> (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
—	Connects terms that are read as a single term by the system, such as <i>set_annotated_delay</i>
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at <http://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click SolvNet Help in the Support Resources section.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to <http://solvnet.synopsys.com> (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to support_center@synopsys.com.
- Telephone your local support center.
 - Call (800) 245-8005 from within the continental United States.
 - Call (650) 584-4200 from Canada.
 - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

Setup and Configuration

This chapter describes how to install and configure the DesignWare[®] ARM[®] Processors SystemC[™] library models for use with CoCentric[®] System Studio. It contains the following sections:

- The DesignWare ARM Processors SystemC Library
- Setting up and Configuring the Library

For more complete information on using System Studio, see the *CoCentric System Studio User Guide*.

The DesignWare ARM Processors SystemC Library

The DesignWare ARM processor SystemC library makes it possible for System Studio users undertaking ARM926 and ARM946 processor-based designs to run unmodified ARM target code (software) in conjunction with their SystemC platform model. It is not necessary to make any special provisions in the software coding.

The following features are supported:

- ARM926EJ-S rev 0
- ARM946E-S rev 1
- ARM Debugger for UNIX (ADU), on Solaris and HP-UX
- ARM Symbolic Debugger (armsd), on Solaris and HP-UX

Setting up and Configuring the Library

The following sections describe how to set up and configure the DesignWare ARM processor SystemC library models.

Prerequisites

To use the DesignWare ARM processor SystemC library you must also install the DesignWare AMBA SystemC library. This library contains the AMBA AHB infrastructure used in the example designs.

The processor model distribution includes an installation of the ARM ADU software debugger.

You must set the SYNOPSISYS_CCSS and SNPS_ARCH environment variables. The SYNOPSISYS_CCSS environment variable should be set to the location of your System Studio installation, and the SNPS_ARCH environment variable should be set to one of the following platform values: sparcOS5, gccsparcOS5, or hp32. Both of these environment variables are used by the DesignWare ARM Processors SystemC Library.

Setting Up the ARM Environment

The installation includes files that you can source to set up the correct ARM environment. These files are platform specific and are located in the dw_arm_processors_sc/arm_ccm directory. Both Bourne shell (.sh) and C-shell (.csh) versions are included; source the file appropriate for your platform and choice of shell.

The setup files must be sourced from the directory in which they are located. If you source them from a different directory, some of the environment variables will be corrupted.

The following is an example of the commands to use if you are working on the Solaris platform with a C-shell environment:

```
% cd install-directory/dw_arm_processors_sc/arm_ccm
% source env_sol.csh
```

The environment setup files assume that the DesignWare ARM Processor SystemC Library is installed in the standard location: \${SYNOPSISYS_CCSS}/../../ccss/models/architectural. If you do not install the DesignWare ARM Processors SystemC Library in this location, you must modify the environment setup files accordingly.

The processor model distribution includes an installation of the ARM ADU software debugger. You must use this ADU installation with the ARM processor models. If you do not set up the ARM environment correctly, and as a result you use an ADU from a different installation on your system, the simulation behavior will be erratic.

Add the ADWU license feature provided to the license file containing your ADS1.2 license features:

- Edit the license file and add the adwu “FEATURE” line to the armlmd section of the license file.
- Use the command **Imutil Imreread** to update the license daemon with the new feature.
- Verify that the feature can be checked out by entering the command **Imutil lmdiag -f adwu**.

Obtaining the ARM Development Tools

If you do not have access to the ARM Development Suite (ADS), contact your local ARM sales office. The tools can also be purchased online from:

www.arm.com/websales/Online_Catalogue_Software_Development_Products_1.html

Required License Features

Simulations that instantiate an ARM processor model require licenses for the ARM software debugger, the ARM processor model, and the System Studio SystemC simulator.

The ADWU license key is included with the ADS1.2 toolset. For normal licensing of the ADWU, contact ARM. However, Synopsys can provide you with a temporary license. If you need a temporary license, submit a support request.

You must modify your `$LM_LICENSE_FILE` environment variable to make the following license manager license features available.

Synopsys license features:

- DesignWare-ARMCORES-tlm
- CoCentric-SYS-Simulator

ARM license features:

- adwu

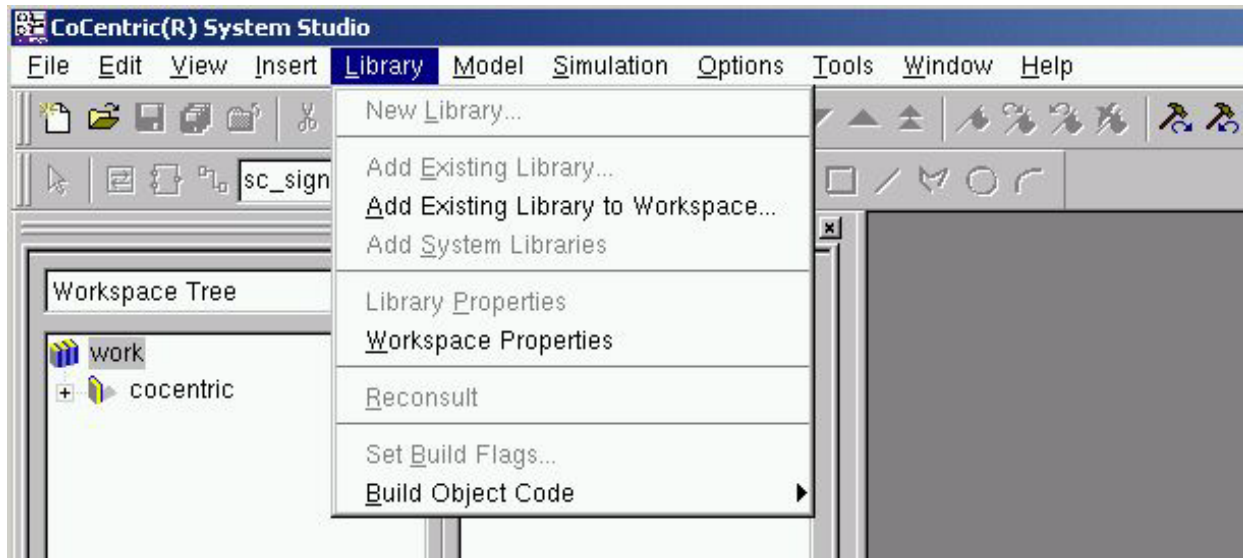
Configuring the Processor Model and the Design

The processor model is instantiated with System Studio just like any other model. The model itself is contained in a System Studio library named `dw_arm_processors.ssl`.

Add the processor library to the System Studio workspace as follows:

- Choose Library › Add Existing Library to Workspace from the System Studio main menu (see Figure 1-1).

Figure 1-1 Adding the Processor Model Library



- Navigate to the dw_arm_processors library, and double-click the file icon. The dw_arm_processors library will now appear in the Workspace Tree.

Once the library has been added to the workspace, the processor model can be instantiated and used like any other System Studio model; just select the model in the Workspace Tree, and drag and drop it into the schematic view.

2

Using the DesignWare ARM Processor Models

This chapter describes how to use the DesignWare ARM processor models in CoCentric System Studio. It contains the following sections

- The DesignWare ARM Processor Model
- Using the Processor Model
- Starting a Simulation
- Loading a Program From ADU
- ARM926 MultiLayer AHB and the Interconnect Matrix
- Using the ARM Software Debugger With System Studio
- Tightly Coupled Memory Models

- Running armsd or ADU Without an Executable
- Example Designs
- Troubleshooting

For more complete information on using System Studio, see the *CoCentric System Studio User Guide*.

The DesignWare ARM Processor Model

The DesignWare ARM processor model is constructed when the SystemC simulation elaborates. When the simulation starts, the processor model starts the ARM debugger process. Once the debugger is running, the processor model emulates the behavior of the real ARM processor: going through reset, fetching, decoding and executing instructions, and responding to asynchronous interrupts.

Because the processor model is a transaction-level model, it only has a few ports:

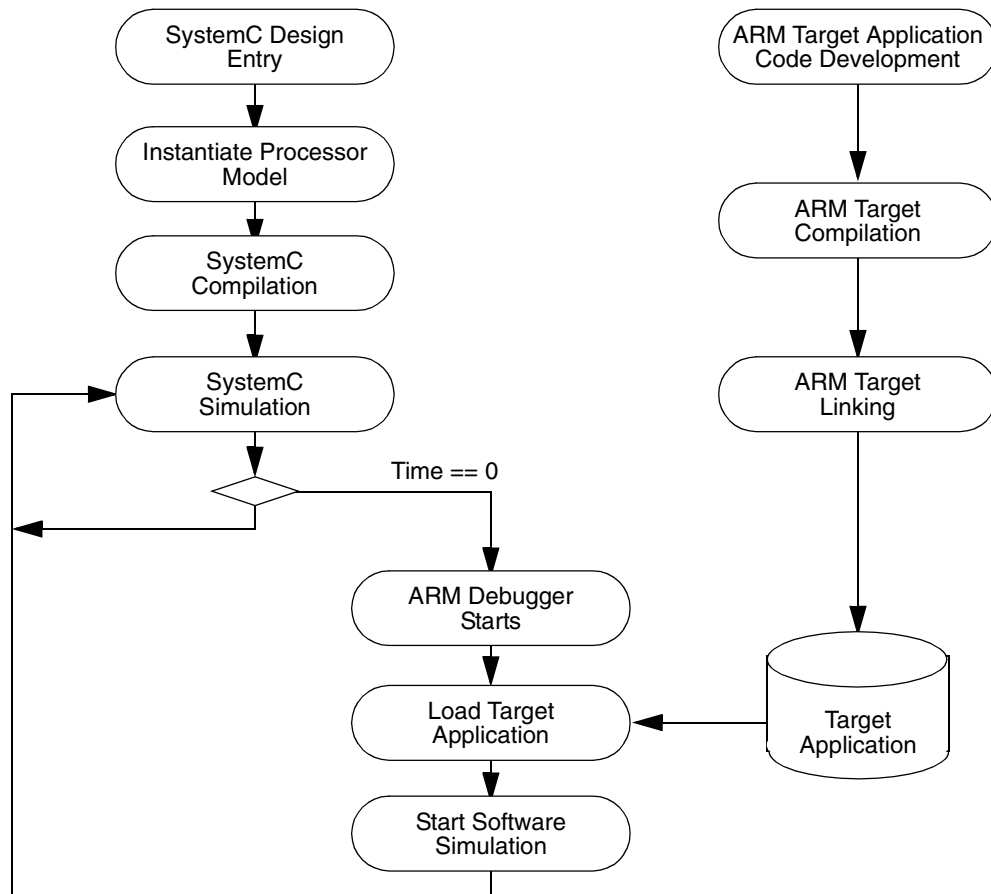
- An instruction AHB bus port, `ibiu`, for connection to the `DW_AHB` instruction bus (`DW_arm926ej`s only)
- A data AHB bus port, `dbiu`, for connection to the `ABM_AHB` data bus
- An instruction TCM port, `itcm`, for connection to an instruction TCM memory
- A data TCM port, `dtcm`, for connection to a data TCM memory
- Interrupt and reset ports (`nIRQ`, `nFIQ`, `nRESET`)
- Initialization ports (`VINITHI` and `INITRAM`)

- A clock port
- H Clock Enable input ports (DHCkEn and, for the DW_arm926ejs only, IHCkEn)

Using the Processor Model

The typical usage scenario for the processor model is shown in Figure 2-1.

Figure 2-1 Typical Usage Scenario



Simulation startup using the processor model proceeds as follows:

- The ARM software debugger is invoked on the first clock edge applied to the processor model. As a result, you will not get a debugger prompt until after simulation is initiated. In the case of the ADU, the debugger window will not appear until the simulation is initiated. If you start the simulation in pause mode, either by invoking the simulation executable with the `-pause` switch or by selecting the “Paused” run option in the System Studio simulation control panel, you must subsequently start the simulation before the debugger prompt will appear.
- Once the debugger is ready you can load the target program and control the software environment (see “Starting a Simulation” on page 2-13 for a description of the indications that the debugger is ready to load the target code).

The memory loading process utilizes the DW_AHB direct memory interface and as a result the simulation time does not advance during this process. If you use a custom memory implementation that does not support the `ahb_direct_if` interface, you must provide an alternative mechanism for loading the target program into memory.

- After the application has been loaded you may set software breakpoints before starting execution of the software. The processor model will hold the SystemC simulation until you start program execution. At that point the model returns control to the SystemC scheduler and the simulation begins to advance.

During simulation you may interact with the ARM software debugger or with the System Studio simulation control facilities. Note that these tools are completely independent: the ARM software debugger is unaware of System Studio, and System Studio is unaware of the

debugger. Refer to “Using the ARM Software Debugger With System Studio” on page 2-17 for a description of the interaction between the ARM software debugger and System Studio.

The ARM ADU and armsd debuggers are described in the ARM document: *ARM Developer Suite, Version 1.1, Debuggers Guide*.
ARM Document # ARM DUI 0066C.

Constructor Arguments

The DW_arm926ejs processor and DW_arm946es processor models have different sets of constructor arguments as described in the following sections.

DW_arm926ejs Processor Model

The DW_arm926ejs processor model has eight constructor arguments:

name_

The module name.

ibiu_priority

This is the priority used by the ibiu port for arbitrating on the instruction bus. If no value is supplied, the default value is 1.

ibiu_default_grant

This argument is passed to the instruction bus during construction and determines whether or not the ibiu port will be the default master for the bus. If not specified, the default value is “true”.

dbiu_priority

This is the priority used by the dbiu port for arbitrating on the data bus. If no value is supplied, the default value is 1.

dbiu_default_grant

This argument is passed to the data bus during construction and determines whether or not the dbiu port will be the default master for the bus. If not specified, the default value is “true”.

BigEndian

This argument configures the ARM debugger in the appropriate byte order. If not specified, the default value is Little Endian (BigEndian = false). This argument has to be consistent with the byte order of the design’s memory system, and the target application must be compiled in that same byte order.

ProgName

This argument allows you to specify the name of the default target application. If not specified, the default value is “” and you must explicitly load a program through the debugger. This feature is used mostly in batch operations or regression tests.

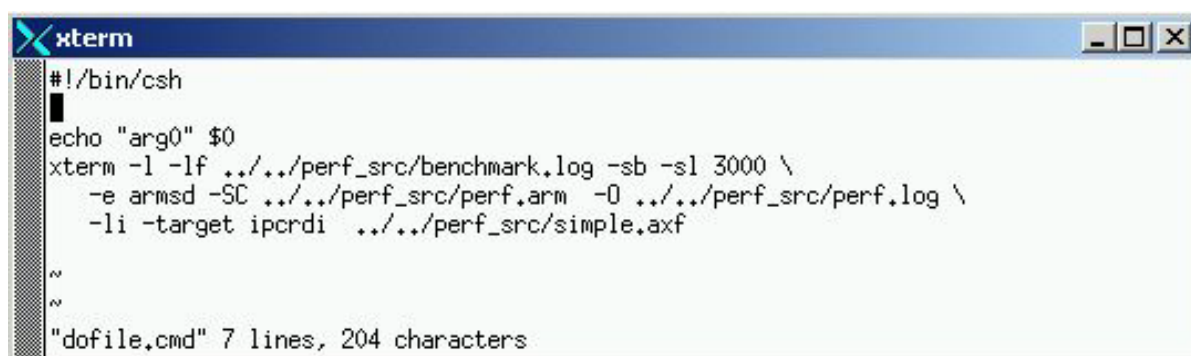
debugger

The processor model invokes an ARM debugger at the beginning of the SystemC simulation. This constructor argument determines which debugger will be invoked; the default setting is “adu”. You may specify adu, armsd, or the name of a shell script. The specified value will be passed as a string to the processor model, which will attempt to locate the debugger via the \$PATH environment variable. If the model cannot resolve the path, it will report an error and terminate the simulation. If the string specifies a shell script, the script is responsible for invoking the debugger

and consuming any parameters passed to it by the processor model, as well as passing the appropriate arguments to the debugger.

If you specify a script file, make certain that its file permissions are correctly set so that it is executable. An example of a script is shown in Figure 2-2.

Figure 2-2 An Example Debugger Script File

A screenshot of an xterm window titled "xterm". The window contains a script file with the following content:

```
#!/bin/csh
echo "arg0" $0
xterm -l -lf ../../perf_src/benchmark.log -sb -sl 3000 \
  -e armsd -SC ../../perf_src/perf.arm -O ../../perf_src/perf.log \
  -li -target ipcrdi ../../perf_src/simple.axf
~
~
"dofile.cmd" 7 lines, 204 characters
```

To see how a constructor is configured for a specific instance, double-click the processor model instance in the schematic view. This opens the Configure Objects dialog box. Click on the constructor tab to see the values passed to the constructor.

DW_arm946es Processor Model

The DW_arm946es processor model has six constructor arguments:

name_

The module name.

priority

This is the priority used by the ibiu port for arbitrating on the bus. If no value is supplied, the default value is 1.

default_grant

This argument is passed to the bus during construction and determines whether or not the dbiu port will be the default master for the bus. If not specified, the default value is “true”.

BigEndian

This argument configures the ARM debugger in the appropriate byte order. If not specified, the default value is Little Endian (BigEndian = false). This argument has to be consistent with the byte order of the design’s memory system, and the target application must be compiled in that same byte order.

ProgName

This argument allows you to specify the name of the default target application. If not specified, the default value is “” and you must explicitly load a program through the debugger. This feature is used mostly in batch operations or regression tests.

debugger

The processor model invokes an ARM debugger at the beginning of the SystemC simulation. This constructor argument determines which debugger will be invoked; the default setting is “adu”. You may specify adu, armsd, or the name of a shell script. The specified value will be passed as a string to the processor model, which will attempt to locate the debugger via the \$PATH environment variable. If the model cannot resolve the path, it will report an error and terminate the simulation. If the string specifies a shell script, the script is responsible for invoking the debugger and consuming any parameters passed to it by the processor model, as well as passing the appropriate arguments to the debugger.

If you specify a script file, make certain that its file permissions are correctly set so that it is executable.

To see how a constructor is configured for a specific instance, double-click the processor model instance in the schematic view. This opens the Configure Objects dialog box. Click on the constructor tab to see the values passed to the constructor.

Design Parameters

In the example designs, design parameters are passed as constructor arguments for both processor models. The parameters control the values of the BigEndian, program_name, and Debugger constructor arguments.

There are two ways in which you can set the values for the parameters in System Studio:

1. By entering a value in the default value field in the Parameter page of the design's interface view. The default values for the BigEndian, program_name, and Debugger parameters are the same as the default values of the constructor arguments.
2. By specifying the values in a simulation control file. The values specified in a simulation control file will override the default values set in the Parameter page of the interface view.

Figure 2-3 shows an example of how to set a parameter value in a simulation control file. Note that you can change parameter values between simulation runs without having to rebuild the design. However, if you change the design's default values in the Parameter page of the interface view, the change will not be reflected in your simulation until you rebuild the design. Note that the BigEndian, program_name, and Debugger parameters are used only during model construction, so that changing a parameter's value in System Studio after a simulation has started will have no effect on the behavior of the ARM processor model.

Figure 2-3 Using a Simulation Control File to Set Parameter Values

A screenshot of an xterm window with a blue title bar and standard window controls. The terminal displays the contents of a simulation control file named 'example.scf'. The file contains several lines of text: 'set SIM_NAME \$env(SIM_NAME)', 'set_parameter /\${SIM_NAME}/Debugger "../perf_src/perf.cmd"', 'set_parameter /\${SIM_NAME}/BigEndian false', 'set_parameter /\${SIM_NAME}/ProgName "../perf_src/simple.axf"', a comment line '# run a simulation iteration', 'run_iteration', two tilde '~' characters, and a status line '"example.scf" 9 lines, 243 characters'.

```
xterm
set SIM_NAME $env(SIM_NAME)
set_parameter /${SIM_NAME}/Debugger "../perf_src/perf.cmd"
set_parameter /${SIM_NAME}/BigEndian false
set_parameter /${SIM_NAME}/ProgName "../perf_src/simple.axf"

# run a simulation iteration
run_iteration

~
~
"example.scf" 9 lines, 243 characters
```

You can also change the design parameters using the Parameters page in the Interface view.

Using H Clock Enable

Each processor model must be clocked on each core clock. You must also indicate to the processor mode for which of these core clocks each AHB is active. This indication is provided via the H clock enable input ports on each DesignWare processor model.

- The DW_arm926ejs model has two H clock enable input ports (IHClkEn and DHClkEN), controlling the instruction bus and the data bus, respectively.
- The DW_arm946es model has one H clock enable input port (DHClkEN), controlling the data bus.

The input pins are active high.

A specific application for utilizing the H clock enable ports is when the processor model core is run at a higher frequency than the external AHB bus.

In the example designs directory of this release (dw_arm_processors_examples) the DW_arm926_example is setup to demonstrate different core and H clock/enable timing. H clock and H clock/enable creation is done by the module HClkGen (found in dw_arm_processors_aux in this release).

Different H clock/enable frequencies can be created using the constructor parameters dbus_ratio and ibus_ratio.

Note:

To clock the processor model core and its buses at the same frequency, just tie the H clock enables high, as shown in the example design DW_arm946_example.

The Technical Reference Manual (TRM) of the ARM core modeled by the CCM documents the clock interface in full detail. Refer to the TRM for full details of the clocking modules supported by each ARM core.

Changing TCM Sizes

Normally the TCM sizes are set in the ARM coprocessor 15 register 9, bits 5:1. However, the ARM CCM ignores the setting of these bits (while still honoring the enable/disable and start address settings). To modify the size of the TCM, you must either modify the coveracs.dsc file provided or add a new .ami file with the desired settings. For example, place the following (sets ITCM and DTCM sizes to 4K) into a file, mycfg.ami :

```
;; ARMulator configuration file type 3
```

```

{ Memories
  { ARM926EJ
    IRamSize=0x1000
    DRamSize=0x1000
  }
}

```

Add the directory containing this file to the ARMCONF environment variable setting, for example:

```

setenv ARMCONF <path>/dw_arm_processors_sc/arm_ccm/
armconfig:<mycfg.ami_dir>

```

For further information on the ARMulator configuration files, see the “Armulator configuration files” section of the ADS 1.2 Debug Target Guide.

Changing Cache Sizes

It is possible to change the cache sizes in a way similar to the way that you change the TCM sizes. The fields to modify for cache sizes are ICache_Lines and DCache_Lines.

For example, to set both the ICache and DCache sizes to 4K, either modify the ICache_Lines and DCache_Lines settings in the coveracs.dsc file provided with the model, or add the following lines to a custom configuration file:

```

;; ARMulator configuration file
type 3 { Processors
  {ARM926EJS_rev0
    ICache_Lines=128
    DCache_Lines=128
  }
}

```

The cache line length is 8 (32-bit) words.

For further information on the ARMulator configuration files, see the “Armulator configuration files” section of the ADS 1.2 Debug Target Guide.

Enabling Semihosting

ARM semihosting may be enabled by adding the following fragment to your custom configuration file (refer to “Changing TCM Sizes” on page 2-11 for instructions on creating a custom configuration file):

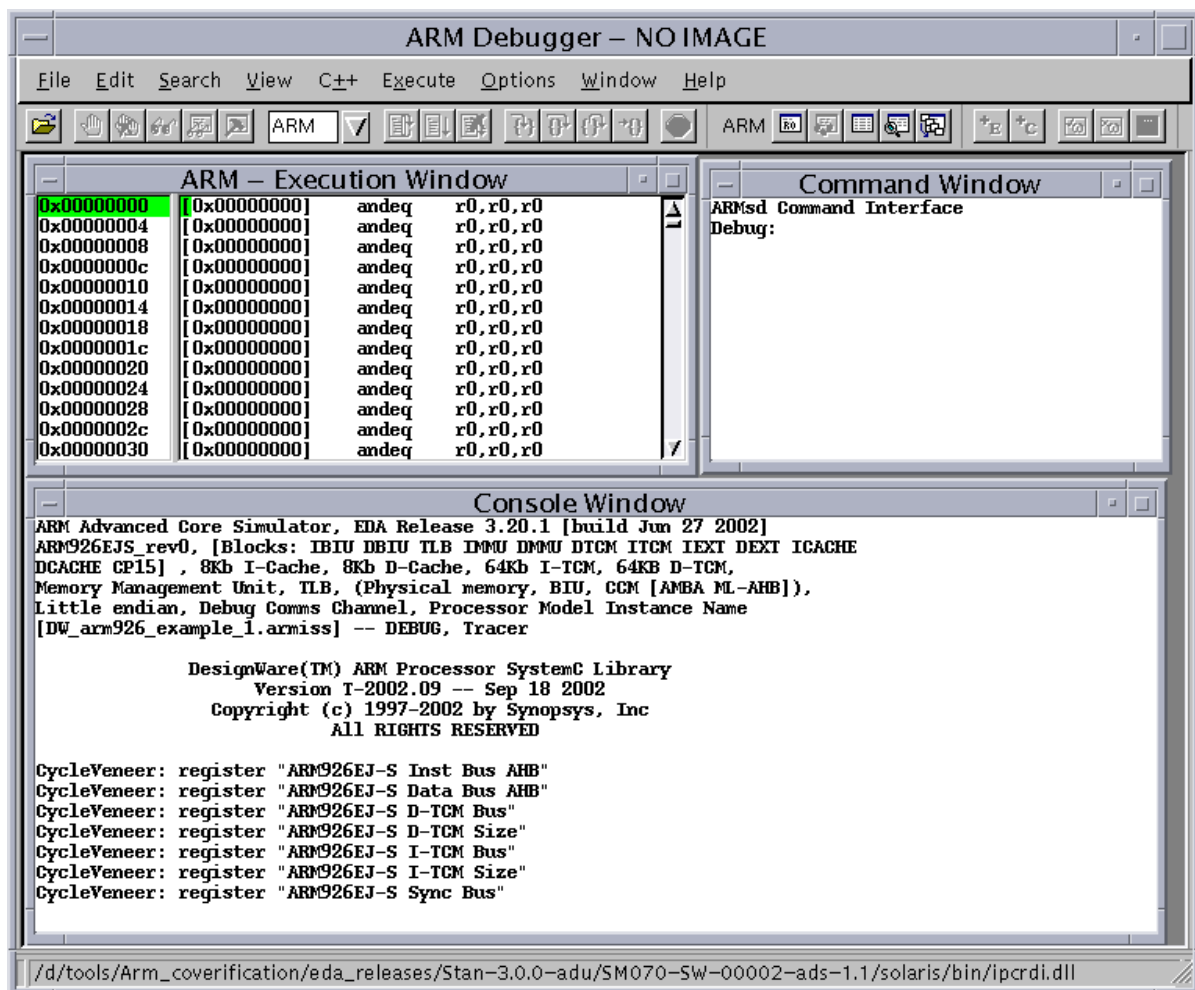
```
{ PeripheralSets
  { Peripherals_Common_ACS
    { SemiHost=Default_Semihost
    }
  }
}
```

For further information on the ARMulator configuration files, see the “Armulator configuration files” section of the ADS 1.2 Debug Target Guide.

Starting a Simulation

Once the design has been compiled in System Studio, start the simulation by clicking Run in the simulation control panel. As described in “Using the Processor Model” on page 2-3, if you start the simulation in paused mode, you must continue the simulation to bring up the ARM software debugger; the debugger will be invoked on the first clock edge of the simulation. By default the example designs will launch the ADU as the debugger, as shown in Figure 2-4.

Figure 2-4 The ADU Window



Note that the ADU Console Window shows which version of the ADU was launched (version 3.20.1 in this example; the first line refers to the ARM ISS to which the ADU is connected) and the name of the processor model instance (DW_arm926_example.armiss). The instance name display is useful when the design contains multiple processors.

Important:

Do not load the target application until the ADU window titled “ARM - Execution Window” displays the contents of memory, as shown in Figure 2-4. This display indicates that the debugger has been fully invoked. Loading the target application before the ARM debugger has been fully invoked will cause the ADU to crash.

Loading a Program From ADU

Target applications can be loaded via the command line, or by using the file navigator:

- Choose File, and then Load Image.
- Use the file navigator buttons to locate the desired target application.
- Select the target program and click Open. The target program is loaded into the simulation’s program memory area.

Note that the memory write accesses associated with the initial loading of a program into the instruction memory do not generate bus cycles (simulation events).

Once the target program has been loaded, you can interact with the ARM software debugger to access registers, variables, and memory locations, set breakpoints, and view the various source files associated with the target program. The program can be viewed as high-level source code, as disassembled machine instructions, or as a combination of both.

Simulation of the target program is initiated with the `step` and `go` commands, either from the command-line interface, or from the ADU graphical interface. Debugging the target program and debugging the SystemC simulation can happen in concert, switching back and forth between the two domains.

ARM926 MultiLayer AHB and the Interconnect Matrix

The DW_arm926ejs model has two independent, asynchronous AHB buses: the instruction bus (accessed through the `ibiu` port) and the data bus (accessed through the `dbiu` port). The instruction bus is used for instruction fetches only. The data bus is used for data reads and writes.

The interconnect matrix allows the instruction and data buses to share a memory region.

Configuring the Interconnection Matrix

The interconnect matrix model is named DW_ahb_icm_tlm, and allows a memory region to be accessed by multiple buses. It is configured via an external file. In the System Studio Design Center, double-click the interconnect matrix model to open the Configure Objects dialog box and use the Constructor page to specify the configuration file. For a description of the file format, see the *DesignWare AMBA SystemC Library User Guide*.

Using the ARM Software Debugger With System Studio

The ARM software debugger is used to debug the software running on the ARM processor model. System Studio provides macrodebugging capabilities for the SystemC simulation, and the host debugger (for example, DDD) provides microdebugging capabilities for SystemC. It is important that you understand the operational paradigm between the ARM debugger and these other debugging facilities.

One situation that is confusing when first encountered arises from how the debugger deals with the pipeline for writes. If the debugger steps over a memory read in the target code, the read will have completed externally. If the debugger steps over a memory write in the target code, the write will not yet have completed externally because of the pipeline for write accesses. If you perform a debug read for an address (by printing the value of memory for that address) immediately after stepping over a memory write to that same address, the debug read will get the correct value even though the external write has not taken place yet. The debug access actually reads the external location (which still has the old data), but the debugger ignores this value and instead returns the correct value from a small cache used for this purpose.

When running with both debuggers active, interaction is intuitive, but there are a few caveats. Either the “hardware” debugger (for example, DDD) or software debugger (ADU) will stop the SystemC simulation (for example, when a breakpoint is reached):

- When a breakpoint is reached in DDD, the SystemC simulation state is completely visible, but you will be unable to view the state of the software using the ARM software debugger.

- When a software breakpoint is reached, you will have full visibility into the software state, but you will not be able to view the state of the SystemC simulation.

This situation may be confusing when you first encounter it. When the ARM software debugger stops the software, it appears to SystemC and System Studio that the processor model is taking a long time to complete its iteration. When System Studio or the host debugger stops the simulation process, it prevents the processor model (which is part of the SystemC process) from communicating with the ARM software debugger, so it is not possible to view the state of the software running on the ARM926 processor.

Another important caveat is that if the hardware debugger, such as DDD, is terminated, the ARM software debugger will be left hanging. It is best to exit the ARM software debugger first and then exit the hardware debugger.

Debugger Interface to Memory

Memory requests made by the ARM software debugger will only be satisfied via the AHB direct memory interface (`ahb_direct_if.h`). If you create your own slave module and want to be able to read or write its memory or registers through the debugger, you must implement the `ahb_direct_if` interface.

Memory Interface Prioritization for Debug Accesses

The processor models support debug accesses to both the AHB and TCM buses. However, because the different memories can have overlapping address ranges, the processor model implements a bus prioritization scheme for satisfying debug memory accesses.

When the model receives a debug request for an address, it checks memory interfaces in the order shown below. It will direct the access to the first memory interface that includes the specified address. If none of the memory interfaces include the specified address, the debugger will report a data abort for the access.

For the DW_arm926ejs model, the order in which it will check the memory interfaces for the specified address is:

1. Instruction TCM
2. Instruction AHB
3. Data TCM
4. Data AHB

For the DW_arm946es model, the order in which it will check the memory interfaces for the specified address is:

1. Instruction TCM
2. Data TCM
3. Data AHB

Tightly Coupled Memory Models

Support for the Tightly Coupled Memory (TCM) interface for the processor models is provided by the DW_TCMemory model included in the dw_arm_processors_aux directory. For a complete description of the supported capabilities of the ARM926EJ-S TCM interface refer to Chapter 5, Tightly Coupled Memory Interface, of the ARM 926EJ-S, (Rev 0), Technical Reference Manual.

The DW_TCMemory module is an sc_module implementing the transaction-level interface defined in tcm_slave_if.h. This module, or hierarchical channel, is connected to the Data or Instruction TCM interface ports (dtcm and itcm respectively).

Note:

Even if your target design does not use the TCM interfaces of the processor models, you must connect a DW_TCMemory to both TCM ports of the processor model: a module may not have unconnected ports. If you do not enable the TCM interfaces in your application software they will not be used.

The DW_TCMemory model supports the following features through parameters:

- wait states for write accesses
- wait states for read accesses
- read-only memory

The DW_TCMemory model supports the following features through constructor arguments:

- memory size configurable from 4 KB to 1 MB
- Endianism selection

Constructor

The DW_TCMemory constructor accepts the following arguments:

- name – module name

- memory size – one of:

```
typedef enum {  
    TCM_4KB    = 0,  
    TCM_8KB    = 1,  
    TCM_16KB   = 2,  
    TCM_32KB   = 3,  
    TCM_64KB   = 4,  
    TCM_128KB  = 5,  
    TCM_256KB  = 6,  
    TCM_512KB  = 7,  
    TCM_1MB    = 8  
} tcm_mem_size_t;
```

- start_address – this must be aligned to the size of the memory
- big_endian – set to true for Big Endian memory byte ordering

Running armsd or ADU Without an Executable

The armsd and adu debuggers will not allow you to enter the go command without either loading an executable image or setting the Program Counter (PC). If you load the ARM executable into memory directly (without using the armsd/adu load command), you must set the PC explicitly before you can start execution.

For either armsd or adu, you need to type the following after invocation

```
let pc=0
```

or replace the 0 with the appropriate start address.

Example Designs

The DesignWare ARM processor SystemC library includes example SystemC designs showing how each processor model is used together with the DesignWare AMBA models. Each design includes target software.

The designs are called DW_arm926_example and DW_arm946_example and are located in the dw_arm_processors_examples directory. Each design contains one processor model as the master, the AHB infrastructure, several memory slaves for program, stack, and heap memory, and two TCM memories. The DW_arm926ejs example also incorporates the interconnect matrix to allow the target software to access program memory using the data bus. This is necessary for self-modifying code and setting up exception vectors.

Troubleshooting

The following section describes problems that you may encounter along with their solutions. Note that many different failures, only some of which are described below, may occur if the ARM environment is not set up correctly. If you encounter a problem that is not described below, verify the following:

- Your DesignWare ARM Processor SystemC Library is installed in the standard location:

```
${SYNOPSYS_CCSS}/../../../../ccss/models/architectural.
```

- You have sourced the correct environment setup file, and have done so from within the dw_arm_processors_sc/arm_ccm directory in the installation.

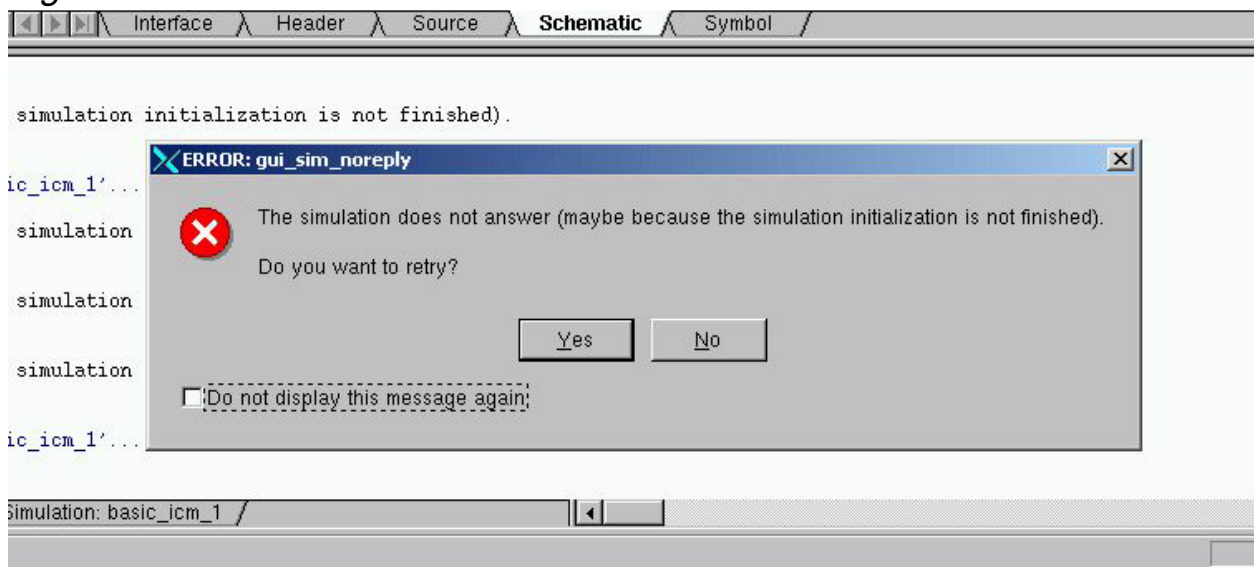
- Your path picks up the ADU from the DesignWare ARM Processor SystemC Library.

The following are problems that you may encounter along with their solutions:

- When starting a simulation, a “gui_sim_noreply” error is reported (see Figure 2-5).

The ARM debugger is launched in a separate process from the SystemC Simulation. The debugger may take several seconds to start. The time taken may be longer than the connection timeout used by the SystemC simulation control panel. If it is, the error dialog box is displayed.

Figure 2-5 Connection Timeout Error



Simply click “Yes” to allow the simulation launcher to retry the connection. Often, the ADU will pop up on its own if you wait a few more seconds (but click Yes anyway to keep the simulation launcher synchronized).

Caution!

Do not suppress this error message (do not check the “Do not display this message again” check box). Doing so will cause System Studio to retry automatically and, if there is a problem with the simulation initialization, System Studio will become locked in the retry loop.

- The ADU appears to load and execute target code, but no bus traffic is generated.

This is probably the result of using the wrong ADU. Make sure that your ARM environment is set up correctly. See “Setting Up the ARM Environment” on page 1-3 for more information.

- The simulation terminates at startup with the following message:

```
ARM debugger and CCM have exited
../../../../src/systemc/kernel/sc_event.h:198:failed assertion
'm_notify_type==NONE'
time:command terminated abnormally
```

This and other similar errors can occur in environments in which a different ADU was previously installed and used, and are caused by an incompatibility between your ADU registry and the ADU you are using. To correct this, check for a process called `windu_registry*` on your workstation. If it is present, terminate it and then remove the registry directory from your workstation. If the name of your workstation were “foo”, you would execute the following command:

```
% rm -rf $HOME/.windu.foo
```

- The simulation terminates with the following message:

```
ld.so.1: adu: fatal: libole43.so: open failed: No such file or
directory
```

This error is caused by an environment variable problem in your ADU setup. See “Setting Up the ARM Environment” on page 1-3 for more information.

- At the beginning of the simulation the debugger error dialog box shown in Figure 2-6 appears.

Figure 2-6 “No Configuration” Debugger Error Message



Note:

This is the result of an incorrect setting of the ARMCONF environment variable. Make sure your ARM environment is set up correctly. See “Setting Up the ARM Environment” on page 1-3 for more information.

Index

A

- ADU, ARM ADU, ARM debugger 1-4
- ADWU license 1-4
- ahb_direct_if interface 2-18
- ARM environment 1-3
- ARM setup files 1-3

B

- BigEndian 2-6, 2-8

C

- clock port 2-3
- constructor arguments 2-5
- custom memory implementation 2-4

D

- data AHB bus port 2-2
- data TCM port 2-2
- dbiu_default_grant 2-6
- dbiu_priority 2-6
- debug memory access prioritization 2-18
- debugger argument 2-6, 2-8
- debugger script file 2-7

- default_grant argument 2-8
- design parameters 2-9
- DW_TCMemory constructor 2-20

E

- environment variables 1-3
- exiting ADU 2-18

G

- gui_sim_noreply error 2-23

I

- ibiu_default_grant 2-5
- ibiu_priority 2-5
- initialization ports 2-2
- instruction AHB bus port 2-2
- instruction TCM port 2-2
- interconnect matrix model 2-16
- interrupt port 2-2

L

- library location 1-3

N

name_ 2-5, 2-7

P

ports 2-2

priority argument 2-7

ProgName 2-6, 2-8

R

reset port 2-2

S

simulation control file 2-9

simulation startup 2-4

simulation startup, pause mode 2-4

SNPS_ARCH 1-3

supported features 1-2

SYNOPSYS_CCSS 1-3

T

tightly coupled memory models 2-19

troubleshooting, ADU registry incompatibility
2-24

troubleshooting, no bus traffic 2-24