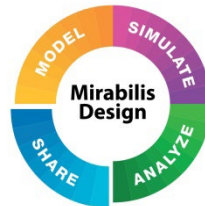


REFERENCE GUIDE

INTRODUCTION TO THE CONCEPTS AND USING VISUALSIM



MIRABILIS DESIGN INC.

2010 El Camino Real, #1061

Santa Clara, CA 95050

www.mirabilisdesign.com

info@mirabilisdesign.com



©2003-2024 Mirabilis Design Inc. All rights reserved.

The information contained herein is subject to change without notice. While every reasonable effort was made to ensure the completeness and correctness of this document, Mirabilis Design Inc. makes no warranty of any kind with regard to this material, including but not limited to any implied warranties. Mirabilis Design Inc. shall not be liable for errors or omissions contained herein or for any damages relating to the use of this material.

VisualSim, VisualSim Architect and Mirabilis Design are registered trademarks of Mirabilis Design Incorporated.

Java and all Java-related titles are trademarks or registered trademarks of Sun Microsystems in the United States and other countries. All other brand or product names may be trademarks of their respective holders.

This document is protected by US and International copyright laws. No part of this document may be reproduced in any manner without prior written consent of Mirabilis Design Inc.

Mirabilis Design Inc.

2010 El Camino Real, #1061

Santa Clara, CA 95050

TABLE OF CONTENTS

1	SCOPE AND PURPOSE OF THE DOCUMENT.....	17
1.1	INTRODUCTION	17
1.2	SCOPE.....	17
2	STARTING VISUALSIM.....	18
2.1	RUNNING VISUALSIM ARCHITECT	18
2.2	RUNNING VISUALSIM CLOUD	18
2.3	RUNNING VISUALSIM POST PROCESSOR	19
3	VISUALSIM PRODUCTS, EDITORS, AND TOOLS.....	21
3.1	HOME PAGE	21
3.2	BLOCK DIAGRAM EDITOR	25
3.2.1	<i>Menu Commands</i>	<i>26</i>
3.2.2	<i>Details of Menu and Toolbar Operation</i>	<i>31</i>
3.2.3	<i>Listeners.....</i>	<i>38</i>
3.2.4	<i>Folders and Tree</i>	<i>39</i>
3.3	TEXT EDITOR AND LISTENER WINDOWS	39
3.3.1	<i>Introduction.....</i>	<i>39</i>
3.3.2	<i>Commands</i>	<i>39</i>
3.3.3	<i>Quick Key Commands</i>	<i>39</i>
3.4	PLOTTERS AND VISUALSIM POST PROCESSOR.....	40
3.4.1	<i>Introduction.....</i>	<i>40</i>
3.4.2	<i>Post Processor Plot Window.....</i>	<i>42</i>
3.4.3	<i>Using Plot.....</i>	<i>43</i>
3.4.4	<i>Commands Configuring the Axes</i>	<i>47</i>
3.4.5	<i>Commands for Plotting Data.....</i>	<i>48</i>
3.5	VISUALSIM POST PROCESSOR	51
3.5.1	<i>Introduction.....</i>	<i>51</i>
3.5.2	<i>Key Features</i>	<i>51</i>
3.5.3	<i>Quick Key Command.....</i>	<i>51</i>
3.5.4	<i>Usage.....</i>	<i>52</i>

3.5.5	<i>VisualSim Architect Models</i>	53
3.5.6	<i>Usage</i>	53
3.5.7	<i>Post Processor Activity</i>	53
3.5.8	<i>Packaging and Archiving</i>	55
3.6	ICON EDITOR.....	55
3.7	EXPRESSION EVALUATOR.....	56
3.8	SIMULATION COCKPIT.....	57
3.8.1	<i>Operation</i>	57
3.8.2	<i>Listen to the Manager and Simulator</i>	58
3.8.3	<i>Pause and Resume</i>	58
3.8.4	<i>Simulation Parameters</i>	58
3.8.5	<i>Plot viewers in the Simulation Cockpit</i>	59
4	PARSERS	60
4.1	ARXML PARSER.....	60
4.2	AXI PARSER.....	62
4.2.1	<i>Parameter configuration in model:</i>	62
4.2.2	<i>Python File Update</i>	62
4.2.3	<i>Parser</i>	63
4.3	GEM5 PARSER.....	63
4.4	GCC PARSER.....	65
4.4.1	<i>Binary and disassembly files</i>	65
4.4.2	<i>GCC file parsing</i>	66
4.5	XML COMPARATOR.....	67
5	BLOCKS	69
5.1	BLOCK EXECUTION SEMANTICS.....	69
5.2	BLOCK LAYOUT.....	70
5.3	BLOCK CONTEXT MENU.....	71
5.3.1	<i>Customize</i>	71
5.3.2	<i>Customize Name</i>	74
5.3.3	<i>Get Documentation</i>	74
5.3.4	<i>Send to Back</i>	74
5.3.5	<i>Bring to Front</i>	74

5.3.6	<i>Save Block in Library</i>	74
5.3.7	<i>Listen to Block</i>	74
5.3.8	<i>Set Breakpoint</i>	75
5.3.9	<i>Convert to Class</i>	75
5.3.10	<i>Class Action - Create Instance (Ctrl+N)</i>	75
5.3.11	<i>Class Action - Create Subclass (Ctrl+U)</i>	75
5.3.12	<i>Convert to Instance</i>	75
5.3.13	<i>Open Block</i>	76
5.3.14	<i>Open Instance</i>	76
5.3.15	<i>Appearance - Edit Custom Icon (Icon Editor)</i>	76
5.3.16	<i>Appearance - Remove Custom Icon</i>	76
5.3.17	<i>Appearance – Flip Ports Horizontally</i>	76
5.3.18	<i>Appearance – Flip Ports Vertically</i>	76
5.3.19	<i>Appearance – Rotate Ports Clockwise</i>	76
5.3.20	<i>Appearance – Rotate Ports Counterclockwise</i>	77
5.4	SETTING UP A BLOCK	77
5.4.1	<i>Blocks Overview</i>	77
5.4.2	<i>Updating Parameters and adding additional parameters</i>	78
5.4.3	<i>Updating Ports and adding additional ports</i>	78
5.4.4	<i>Setting Port Types</i>	79
5.4.5	<i>Linking Parameters</i>	80
5.4.6	<i>Block Documentation</i>	81
5.4.7	<i>Icon Display value</i>	81
5.4.8	<i>Tooltip value</i>	81
5.4.9	<i>Creating Generic Port Icon</i>	81
6	HIERARCHICAL BLOCK SEMANTICS	83
6.1	<i>USING HIERARCHICAL BLOCKS</i>	83
6.2	<i>MODEL EXECUTION SEMANTICS</i>	85
6.3	<i>MODEL EXECUTION FLOWCHART</i>	86
6.4	<i>CREATE SUB-MODEL</i>	87
7	DYNAMIC INSTANTIATION	88
7.1	<i>INTRODUCTION</i>	88

7.2	EXAMPLE	88
7.2.1	<i>How To</i>	90
7.2.2	<i>Requirements</i>	90
7.2.3	<i>Example of the Dynamic Instantiation</i>	91
8	BLOCK, SUB-MODELS (CLASSES) AND INHERITANCE.....	93
8.1	INTRODUCTION	93
8.2	EXAMPLE OF CLASSES.....	94
8.2.1	<i>Overriding Parameter Values in Instances</i>	101
8.2.2	<i>Subclass and Inheritance</i>	102
8.2.3	<i>Sub-Models</i>	104
9	CLASSES AND LIBRARIES	106
9.1	CREATE CLASS- SEQUENCE OF OPERATIONS	106
9.2	CREATE LIBRARY AND INSTANTIATE IN USER_LIBRARY	107
9.3	EXAMPLE OF CREATING A NEW CLASS BLOCK.....	107
9.4	ANNOTATING THE CLASS.....	110
9.5	INSTANTIATING THE CLASS IN MODEL.....	110
9.6	TEST THE NEW CLASS BLOCK:	111
10	LIBRARY MANAGEMENT	113
10.1	INTRODUCTION	113
10.2	VERSION CONTROL	113
10.3	LIBRARY STORAGE	113
10.4	USER LIBRARY	115
11	LIBRARY FOLDER ORGANIZATION	116
11.1	MODEL / DOCUMENT / MODELSETUP	116
11.2	TRAFFIC / SOURCE	117
11.3	FAILURE ANALYSER	118
11.4	RESULTS	118
11.5	BEHAVIOR.....	119
11.6	MAPPERS.....	121
11.7	RESOURCES	121
11.8	POWER.....	122

11.9	HARDWARE SETUP	123
11.10	PROCESSOR GENERATOR.....	124
11.11	MICROARCHITECTURE COMPONENTS	124
11.12	MEMORY.....	125
11.13	HARDWARE DEVICES	126
11.14	INTERFACES AND BUSES.....	126
11.15	SYSTEM_LANGUAGE.....	127
11.16	HARDWARE LANGUAGE.....	127
11.17	MATH_OPERATIONS	128
11.18	ALGORITHMIC.....	129
11.19	USERLIBRARY	129
12	AUTO-SAVE.....	130
13	SIMULATION TECHNOLOGY	131
13.1	INTRODUCTION	131
13.2	MODELS OF COMPUTATION	131
13.3	SIMULATORS- HOW TO SELECT THE RIGHT ONE?	132
13.4	SIMULATORS	133
13.4.2	<i>Choosing Models of Computation</i>	<i>137</i>
14	CONCEPT OF TIME	139
14.1	INTRODUCTION	139
14.2	TIME RESOLUTION	139
14.3	SIMULATION TIME	140
14.4	RELATIVE VS CLOCK TIME.....	140
14.5	COMPUTING TIME	140
14.6	CLOCK SYNCHRONIZATION.....	141
14.7	MODEL EVENT.....	141
15	DATA STRUCTURES	143
15.1	INTRODUCTION	143
15.2	LAYOUT	143
15.3	SUPPORTED DATA TYPES	145
15.4	PORT TYPING.....	146

15.5	DATA STRUCTURE TEMPLATE LOCATION.....	146
15.6	DATA STRUCTURE DEFINITIONS AND EXAMPLE.....	146
15.6.1	<i>Text Example</i>	147
15.6.2	<i>Java Example</i>	147
15.6.3	<i>Finding the Data Structure Template</i>	148
15.7	PATH DEFINITION	148
15.8	CONSTRUCTION.....	149
15.8.1	<i>Compiling Java Data Structure</i>	149
16	PROCESSOR_DS.....	150
16.1	INTRODUCTION	150
16.2	USAGE OF DATA STRUCTURE FIELDS IN THE MODEL	154
16.3	PROCESSOR_DS EXAMPLE.....	157
17	PARAMETERS.....	159
17.1	INTRODUCTION	159
17.2	PARAMETER VALUES	159
17.3	CREATION.....	160
17.4	USING PARAMETERS.....	160
17.5	LINKING PARAMETERS UP/ DOWN THE MODEL HIERARCHY	163
17.6	PARAMETERS AS VARIABLES	164
17.7	SHARED PARAMETERS.....	164
17.8	SPECIAL PARAMETERS	165
17.9	PARAMETER VALUE ON THE ICON	165
18	CONFIGURING PORTS AND MAKING CONNECTIONS	167
18.1	INTRODUCTION	167
18.2	EDITING	167
18.3	ADDING A PORT	169
18.4	REMOVING PORTS	170
18.5	POLYMORPHIC PORT TYPES.....	170
18.6	PORTS.....	171
18.7	RELATION AND LINK.....	171
18.8	POINT-TO-POINT BLOCK CONNECTIONS.....	172
18.9	MAKING CONNECTIONS.....	173

18.10	VIRTUAL CONNECTIONS	173
18.11	ADDING PARAMETERS TO LINKS AND PORTS	173
18.12	CREATING GENERIC PORT ICON	174
19	MODEL VARIABLE	176
19.1	INTRODUCTION	176
19.2	VARIABLE TYPES	176
19.3	VARIABLE DATA TYPES	177
19.4	VARIABLE NAMES	177
19.5	INITIALIZE VARIABLE	178
19.6	CHECKERS.....	179
19.7	ACCESSING LOCAL VARIABLES	180
19.8	THINGS TO REMEMBER	180
19.9	POINTER / REFERENCE TO A VARIABLE	181
20	FILE PATHS.....	183
21	REGEX LANGUAGE	184
21.1	COMPUTATION USING REGEX.....	184
21.2	CASTING AND CONVERSIONS	185
21.3	REGEX FUNCTIONS- QUEUE AND SYSTEMRESOURCE; METHODS.....	186
21.3.1	<i>Mapping Functions to Blocks for getBlockStatus.....</i>	<i>186</i>
21.3.2	<i>getBlockStatus Functions</i>	<i>187</i>
21.3.3	<i>getBlockStatus format and example.....</i>	<i>188</i>
21.4	MATH FUNCTIONS	190
22	ACCELERATING MODEL SIMULATION PERFORMANCE	195
23	PAUSE AND RESUME FUNCTION WITH DEBUGGING FACILITY.....	197
23.1	HOW TO USE PAUSE AND RESUME FUNCTION	197
23.2	STEPS:	197
24	DATA TYPES.....	198
24.1	SCALAR AND NUMERICAL VALUES	198
24.2	STRING CONSTANTS.....	198
24.3	ARRAYS.....	199

24.4	MATRICES	203
24.5	DATA STRUCTURE OR TRANSACTION OR RECORD TOKEN	206
24.6	INVOKING METHODS	208
24.7	DEFINING FUNCTIONS	209
24.8	FIXED POINT NUMBERS.....	212
24.9	SPECIAL FUNCTIONS.....	214
24.10	DISTRIBUTIONS	215
24.11	PROPERTY()	216
24.12	REMAINDER().....	216
24.13	POWER AND MODULO	217
25	MODELING LANGUAGES	219
26	DATA STRUCTURES EXPRESSION LANGUAGE	223
26.1	FORMAT FOR THE STATEMENT BLOCKS.....	226
27	DIFFERENCES BETWEEN SCRIPT AND C PROGRAMMING	229
28	SELECTING THE RIGHT BLOCK FOR YOUR MODEL.....	230
28.1	TRAFFIC GENERATION	230
28.2	VARIABLES.....	231
28.3	PLOTTING	233
28.4	STATISTICS	235
28.5	POWER STATISTICS	236
28.6	RESOURCES AND HARDWARE	237
28.7	UTILITIES AND FUNCTIONS	240
29	ERROR MESSAGES	241
30	MODEL-LEVEL DEBUGGING.....	242
30.1	MODEL ANIMATION	243
30.2	DISPLAY AND STATISTICS.....	243
30.3	LISTEN TO SIMULATOR.....	244
30.4	LISTEN TO PORT.....	244
30.5	LISTEN TO BLOCK	244
30.6	SCRIPT PROFILER FOR THE SCRIPT BLOCK AND SMART CONTROLLER.....	245

30.7	LISTEN TO FILE FOR SCRIPT BLOCK AND SMART CONTROLLER	245
30.8	DEBUGGING VARIABLE.....	246
30.9	DEBUGGING A NEW MODEL.....	247
30.10	DIGITAL DEBUGGING.....	248
30.11	SETTING BREAKPOINTS.....	249
30.11.1	<i>Set Breakpoints context menu choice</i>	249
30.12	PAUSE AND RESUME FUNCTION WITH DEBUGGING FACILITY	250
30.12.1	<i>How to Use Pause and Resume Function</i>	250
30.12.2	<i>Steps:</i>	251
30.13	TESTING THE RTL	251
30.14	HOW TO DEBUG- USING THE METHODOLOGY PROVIDED.....	251
30.14.1	<i>INTRODUCTION</i>	251
30.14.2	<i>Animate</i>	252
30.14.3	<i>Listen To Simulator</i>	252
30.14.4	<i>Listen To Port</i>	253
30.14.5	<i>Listen To Block</i>	253
30.14.6	<i>Tracer</i>	253
	<i>Difficulties in debugging any system and how it is different from debugging a regular piece Java or C/C++ code.</i>	254
30.14.7	254
30.14.8	<i>Sequence for debugging</i>	254
31	SOFTWARE MODELING.....	256
31.1	TASK GRAPH APPROACH	256
31.2	TASK GENERATOR APPROACH	257
31.3	TRACE APROACH.....	257
31.4	MICROARCHITECTURE APPROACH	258
32	INTRODUCTION TO FINITE STATE MACHINE	259
32.1	FINITE STATE MACHINE.....	259
32.1.1	<i>FSM-Controller</i>	259
32.1.2	<i>Guard Expressions</i>	259
32.1.3	<i>Actions</i>	260
32.2	EXECUTION	260

32.3	FINITESTATEMACHINE	261
33	VISUALSIM CLOUD AND FILE EXPORT TO WEB	262
33.1	HOW TO ACCESS VISUALSIM CLOUD	262
33.2	EXPORT MODEL TO WEB. (DISABLED IN THE CURRENT VERSION. IT WILL BE AVAILABLE IN THE FUTURE RELEASE)	262
34	VISUALSIM® INTERFACE WITH FPGA.....	264
34.1	VISUALSIM FPGA INTERFACE APPLICATIONS	264
34.2	HARDWARE AND SOFTWARE REQUIREMENTS	265
34.3	VISUALSIM FPGA INTERFACE KIT.....	265
34.4	HOW INTERFACE WORKS	265
34.4.1	<i>Procedure to Run Example</i>	<i>266</i>
34.5	CONFIGURING FPGA AND SOURCE FILES	268
34.6	POSSIBLE ERRORS	271
34.7	APPENDIX	272
35	QEMU GUIDE.....	274
35.1	STEPS TO INSTALL QEMU ON UBUNTU	274
35.2	DOWNLOAD BENCHMARK TO RUN THE PROGRAM	275
35.3	STEPS TO COMPILE AND RUN C PROGRAM FOR X86 AND ARM	275
36	VISUALSIM GEM5-ARM INTEGRATION.....	283
36.1	INSTALLATION STEPS FOR UBUNTU OS	283
	<i>ASSUMPTION: THE USER HAS INSTALLED JAVA VERSION 14 OR LATER AND VISUALSIM ARCHITECT.</i>	
	283	
36.2	INSTALLATION STEPS FOR CENTOS 8.3	283
	<i>ASSUMPTION: THE USER HAS INSTALLED JAVA VERSION 14 OR LATER AND VISUALSIM ARCHITECT.</i>	
	283	
36.3	RUN THE GEM5 WITH VISUALSIM	284
36.4	TRACE GENERATION FROM GEM5:.....	285
36.5	DEBUG GEM5 USING GDB:.....	285
36.6	DEBUG GEM5 USING GRAPHICAL GDB:	288
36.7	RUNNING THE GEM5 IN SYSTEM CALL EMULATION (SE) MODE.....	291
36.8	RUN THE GEM5 (SE MODE) IN MULTI-CORE	292

36.9	RUNNING THE GEM5 IN FULL SYSTEM (FS) MODE	293
37	GEM5 WITH RISCV	299
37.1	FOR UBUNTU 18.04 INSTALLATION STEPS	299
37.2	RUN THE GEM5 <-> VISUALSIM (NORMAL RUN WITHOUT DEBUGGER).....	299
37.3	RUN WITH DEBUGGER -COMMAND LINE:.....	301
37.4	RUN WITH DEBUGGER - GRAPHICAL GDB:.....	304
37.5	RUNNING THE GEM5 IN SYSTEM CALL EMULATION (SE) MODE.....	306
37.6	RUNNING THE GEM5 (SE MODE) IN MULTI-CORE	307
37.7	RUNNING THE GEM5 IN FULL SYSTEM (FS) MODE	308
38	SPIKE TO VISUALSIM	309

TABLE OF FIGURES

FIGURE 1.	VISUALSIM PLOT APPLICATION WINDOW.....	41
FIGURE 2.	APPLETVIEWER.....	44
FIGURE 3.	ZOOM IN.....	45
FIGURE 4.	PLOT	46
FIGURE 5.	SIMULATION COCKPIT	57
FIGURE 6.	BLOCK EXECUTION SEMANTICS	70
FIGURE 7.	BLOCK ATTRIBUTES	70
FIGURE 8.	BLOCK CONTEXT MENU	71
FIGURE 9.	PORTS THAT CANNOT BE EDITED.....	72
FIGURE 10.	SET PORTS TO BLOCK	79
FIGURE 11.	LINKING PARAMETER	80
FIGURE 12.	HIERARCHICAL BLOCK -> OPEN BLOCK.....	83
FIGURE 13.	ADDING BLOCKS TO A HIERARCHICAL BLOCK	84
FIGURE 14.	MODEL EXECUTION SEMANTICS	86
FIGURE 15.	MODEL EXECUTION FLOWCHART	86
FIGURE 16.	A MODEL USING THE DYNAMIC INSTANTIATION BLOCK	88
FIGURE 17.	CONFIGURE PARAMETERS FOR DYNAMIC INSTANTIATION	89
FIGURE 18.	DYNAMIC INSTANTIATION EXAMPLE IN VISUALSIM	91
FIGURE 19.	LIBRARY FOLDER ORGANIZATION.....	116

FIGURE 20.	BLOCKS IN DOCUMENT/ MODEL / MODELSETUP LIBRARY FOLDERS	117
FIGURE 21.	BLOCKS IN SOURCE LIBRARY	118
FIGURE 22.	BLOCKS IN RESULT LIBRARY	119
FIGURE 23.	CONTROL FLOW BLOCKS	120
FIGURE 24.	MAPPERS.....	121
FIGURE 25.	BLOCKS IN RESOURCE LIBRARY	122
FIGURE 26.	POWER	123
FIGURE 27.	HARDWARE SETUP	123
FIGURE 28.	PROCESSOR GENERATOR	124
FIGURE 29.	MEMORY	125
FIGURE 30.	BLOCKS IN HARDWARE DEVICES.....	126
FIGURE 31.	INTERFACES AND BUSES	127
FIGURE 32.	INTERFACE BLOCKS	128
FIGURE 33.	MATH OPERATIONS LISTING	128
FIGURE 34.	CONFIGURE PORTS	95
FIGURE 35.	HIERARCHICAL MODEL TO CREATE A CLASS	96
FIGURE 36.	A POOR DESIGN OF A DIVERSITY COMMUNICATION SYSTEM.....	97
FIGURE 37.	CREATING AND USING A CHANNEL CLASS.....	100
FIGURE 38.	THE ICON CHANGED FOR THE CLASS	100
FIGURE 39.	MODEL WITH THE ICONS OF THE INSTANCE	101
FIGURE 40.	THE MODEL WITH A SUBCLASS OF THE CHANNEL WITH NO OVERRIDES	102
FIGURE 41.	THE SUBCLASS WITH OVERRIDES THAT ADD SINUSOIDAL INTERFERENCES	103
FIGURE 42.	A MODEL USING THE SUBCLASS FORM AND A PLOT OF THE EXECUTION	104
FIGURE 43.	CREATE A BLOCK DIAGRAM.....	109
FIGURE 44.	BLOCK ATTRIBUTES	109
FIGURE 45.	MAKING THE CONNECTION AND SELECT+ALL	109
FIGURE 46.	CREATE A CLASS.....	110
FIGURE 47.	INSTANTIATE CLASS TO IMPORT A CLASS	111
FIGURE 48.	VISUALSIM RECOMMENDED DIRECTORY STRUCTURE	114
FIGURE 49.	LIBRARY INTEGRATION BETWEEN CENTRAL AND LOCAL SYSTEM.....	114
FIGURE 50.	SIMPLE EXAMPLE	115
FIGURE 51.	SIMPLE EXAMPLE OF A VISUALSIM MODEL EXECUTION CONTROL WINDOW	115
FIGURE 52.	VISUAL NOTATION AND TERMINOLOGY	115

FIGURE 53.	RELATIONSHIP GROUPS	115
FIGURE 54.	EXAMPLE TOPOLOGY.	115
FIGURE 55.	SINE WAVE GENERATOR TOPOLOGY.....	115
FIGURE 56.	VERTEX EXAMPLE.....	115
FIGURE 57.	EXAMPLE OF A FUNCTION BEING PASSED FROM ONE ACTOR TO ANOTHER.....	211
FIGURE 58.	ELABORATE USE	211
FIGURE 59.	USING PARAMETERS IN EXPRESSIONS	161
FIGURE 60.	USING PARAMETERS TO SET BLOCK ATTRIBUTES.....	162
FIGURE 61.	LINKING PARAMETERS	164
FIGURE 62.	USING PARAMETERS TO SET THE ICON COLOR.....	166
FIGURE 63.	CONFIGURE PORTS FOR THE BLOCK.....	167
FIGURE 64.	PORTS THAT CANNOT BE EDITED.....	167
FIGURE 65.	BLOCK-TO-BLOCK CONNECTIONS	172
FIGURE 66.	VISUALSIM MODELING LANGUAGES	219
FIGURE 67.	LIST OF APPLICATION INTERFACE BLOCKS.....	221
FIGURE 68.	POST PROCESSING LATENCY	ERROR! BOOKMARK NOT DEFINED.
FIGURE 69.	POST PROCESSING HISTOGRAM	ERROR! BOOKMARK NOT DEFINED.
FIGURE 70.	MODEL PLOT.....	ERROR! BOOKMARK NOT DEFINED.
FIGURE 71.	HARDWARE MODELING LIBRARY	ERROR! BOOKMARK NOT DEFINED.
FIGURE 72.	LIST OF HARDWARE BUSES	ERROR! BOOKMARK NOT DEFINED.
FIGURE 73.	CYCLE-ACCURATE CACHE AND MEMORY MODELS...	ERROR! BOOKMARK NOT DEFINED.
FIGURE 74.	MODEL ANIMATION	243
FIGURE 75.	ANNOTATION WITH LINEREADER PLUS PROCESSING BLOCK	258
FIGURE 76.	FINITESTATEMACHINE ILLUSTRATION.....	261
FIGURE 77.	EXPORT TO WEB DIALOG WINDOW	ERROR! BOOKMARK NOT DEFINED.
FIGURE 78.	VISUALSIM INTERFACE WITH FPGA	264
FIGURE 79.	VISUALSIM MODEL.....	267
FIGURE 80.	CONFIGURING DATAGRAM WRITER BLOCK	267
FIGURE 81.	CONFIGURING DATAGRAM READER.....	268
FIGURE 82.	GENERATE BITSTREAM	269
FIGURE 83.	GENERATE BITSTREAM	269
FIGURE 84.	EXPORT HARDWARE	270
FIGURE 85.	LAUNCH SDK	270

FIGURE 86.	PROJECT EXPLORER.....	271
FIGURE 87.	CANNOT EXPORT HARDWARE.....	272
FIGURE 88.	OPEN BLOCK DESIGN	273
FIGURE 89.	SHOW IP STATUS	273
FIGURE 90.	UPDATE SELECTED	273
FIGURE 91.	RECOMMENDATION ENGINE BLOCK.....	ERROR! BOOKMARK NOT DEFINED.

1 Scope and Purpose of the Document

1.1 Introduction

Mirabilis Design provides systems engineering solutions for performance analysis, power estimation, and architecture exploration of electronics and real-time software. Mirabilis' product **VisualSim** is used for performance, functional, and power exploration of network of systems, large systems, sub-systems, components (IC, SoC, FPGA, and boards) and real-time software. VisualSim models are constructed in a graphical block diagram editor using a parameterized modeling blocks library.

1.2 Scope

This document provides the information required to enable the reader to construct system models, execute simulations, and analyze the results with an intent to optimize the specification. This document explains the basic components in this framework, the simulator, modeling language, and provides an overview of the libraries.

The intended audience for this document is an engineer or researcher who wishes to construct models in VisualSim, debug these models, and conduct analysis. The goal of the exploration is to architect the right product, that is one which minimizes product failures and has not been over or under designed.

2 Starting VisualSim

2.1 Running VisualSim Architect

To start VisualSim Architect, do the following:

1. The user needs to start the License Server located in the Web Server or local system.
2. To start the License Manager, locate the install directory and navigate to VS_LM.
 - a. From a Command-line, system-prompt or shell window, type StartServer.bat (Windows) or StartServer.sh (UNIX) once you changed the directory to VS_LM. You can also start the License Server by double clicking the file StartServer.bat (Windows).
3. To start Architect, do one of the following:
 - a. Double-click Architect VisualSim Architect in the Application Menu.
 - b. Double-click VisualSim.bat (WINDOWS) or VisualSim.sh (UNIX) in the VisualSim directory viewer.
 - c. From a Command-line or system-prompt or shell window, go to the VisualSim install and type VisualSim.bat (Windows) and VisualSim.sh (UNIX).

2.2 Running VisualSim Cloud

Prerequisite for VisualSim Cloud:

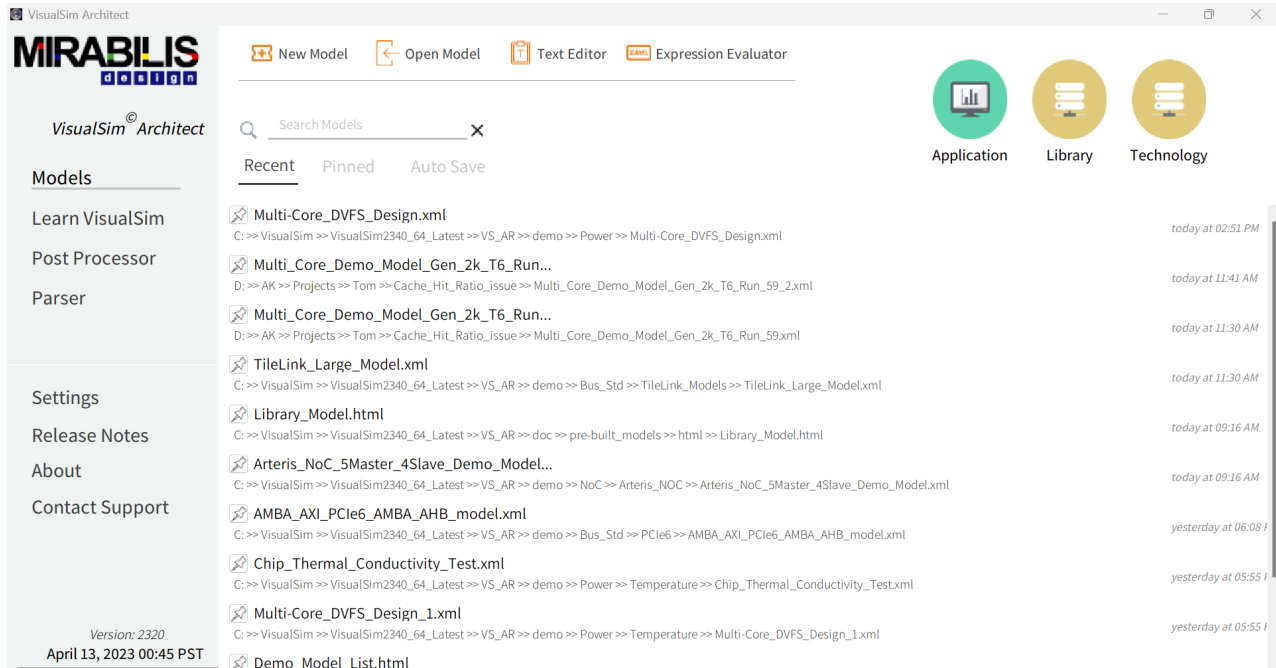
OpenWebStart - <https://openwebstart.com/download/>

Java 17 - <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

Download Startermodel.jnlp using VisualSim Cloud login.

To start VisualSim Cloud, do the following:

- ⇒ Double-click on Startermodel.jnlp from your local drive. Click yes on java security warnings.

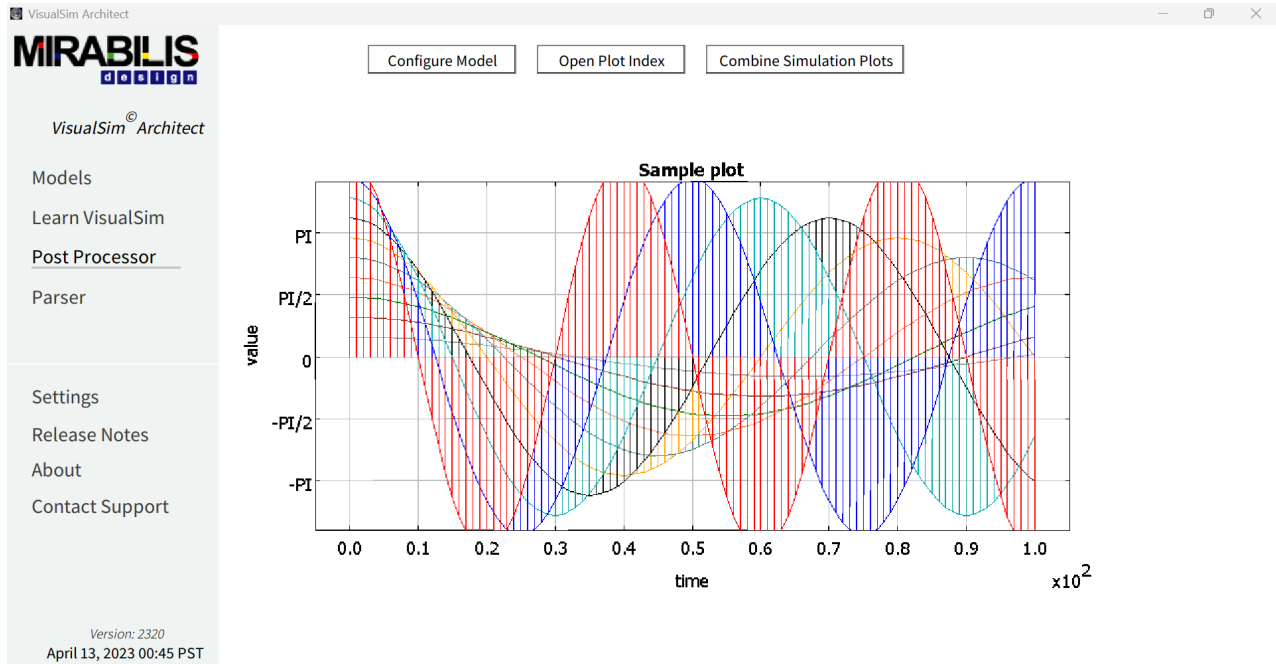


2.3 Running VisualSim Post Processor

To start the Post Processor, do the following:

- ⇒ Double-click PostProcessor.bat (WINDOWS) or PostProcessor.sh (UNIX) in the Post Processor directory.

From a Command-line, system-prompt or shell window, go to the Post Processor install and type PostProcessor.bat (Windows) or PostProcessor.sh (UNIX).



3 VisualSim Products, Editors, and Tools

3.1 Home Page

The home page of the tool provides the graphical interface to the user to access all the features in VisualSim. The following image shows the overview of the home page and the features available in the tool.

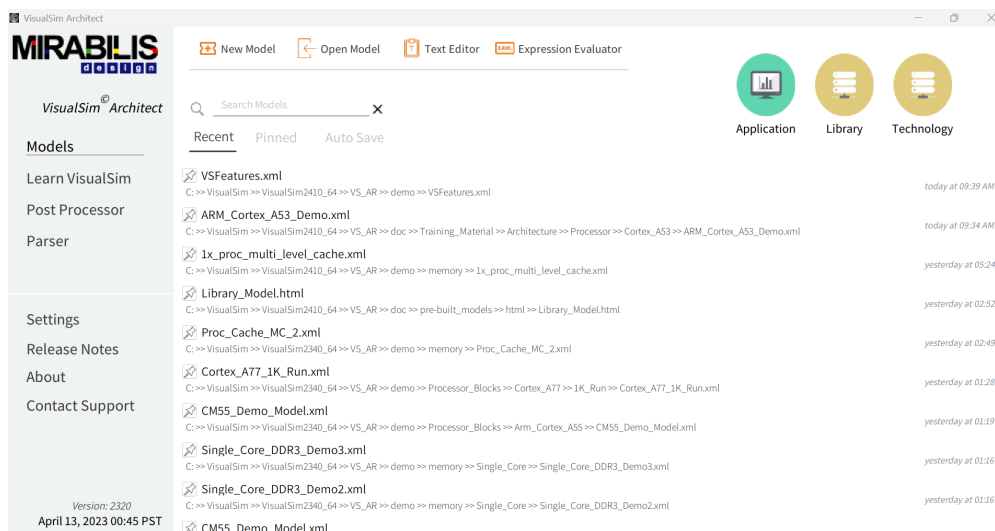


Figure 1. Tool Home page

The top section of the home page provides the following access:

1. **New Model** : Opens a Block Diagram Editor, where user can drag and drop library blocks and create their own design, and utilize the complete modeling features of VisualSim.
2. **Open Model**: Allows user to open an existing model in the system. By default it will go to the VS_AR (Home directory of the tool) folder, where user can access the demo folder which contain demo models for different applications and protocols. User can also navigate to other directories where they can find VisulaSim models, text file or plot files that can be opened by VisualSim.
3. **Text Editor**: Opens the Text editor window which allows user to write scripts , algorithms, and documentation. But ultimately this feature is used to write codes in VisualSim script language for any logic.

4. Expression Evaluator: Opens a window where user can try different RegEx and mathematical operations supported by VisualSim. User can use this feature to evaluate their logic, verify the RegEx format/usage, and try experimenting with the regex to see if it can be used in your Script or Expression List logic.
5. Search Bar: Search the models that you accessed before by typing their name.
6. Recent: This section list the recent model that are accessed by the user. It avoids the need to open the frequently used model by navigating to the specific directory each time.
7. Pinned: This section allows the user to pin specific models to easily access them whenever they need.

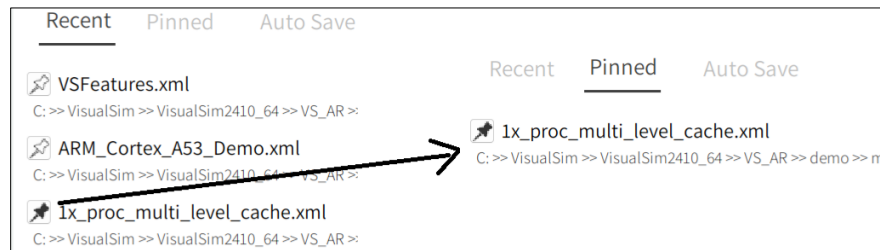


Figure 2. Model Pinning in Home page

8. Auto Save: This section will contain the list of models that are auto saved while the user was using it. This auto save feature save the model periodically if the user makes some changes to the model but not saved in the local directory. It helps the user to have a backup if the lost the design by accident, like closing the model without saving it.

The top right section provides access to the demo models based on the application, library and technology. User can access this section to navigate to the model that best suits for their requirement.

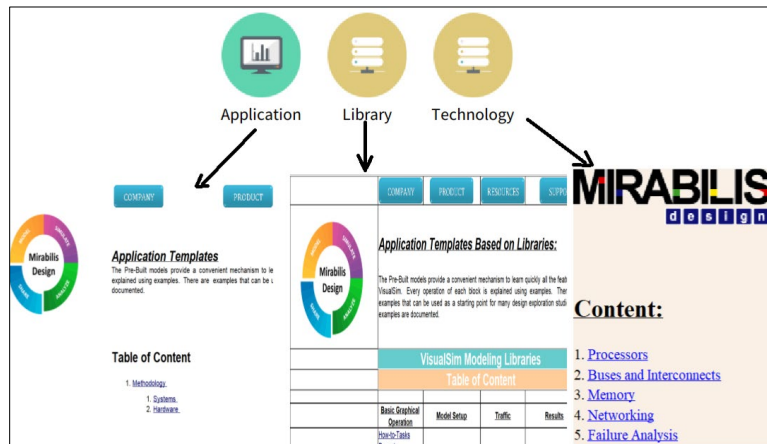


Figure 3. Demo model classifications

The left section of the tool home page contains some of the important features and setting of VisualSim.

1. **Models:** This section allows the user to access the Recent model list, Pinned model list and auto saved model list.
2. **Learn VisualSim:** This section provides access to the documents, training materials, and tutorials of VisualSim, where users can learn about VisualSim features, usage of the tool and usage of internal functionality.
3. **Post Processor:** This allows users to perform post analysis of a design, which includes creating batch file to run multiple combination of parameters and comparing the results of multiple runs.
4. **Parser:** The user can use this section to generate files for a specific application which can be used as an input file in the model. It supports parsing for ARXML, AXI, Gem5 and GCC and XML comparator. For more details visit the Parser section in this document.
5. **Settings:** This section contains the VisualSim Settings and the user can modify certain attributes for the correct function. (Default values works in most cases, modify it only if it is necessary). The settings will be updated once the tool is closed and reopened.

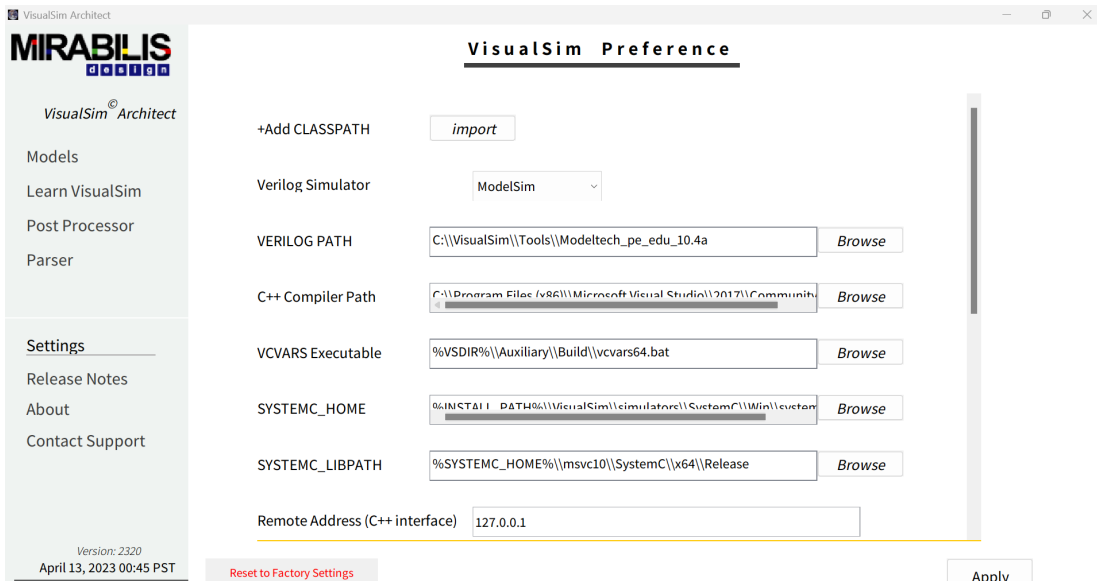
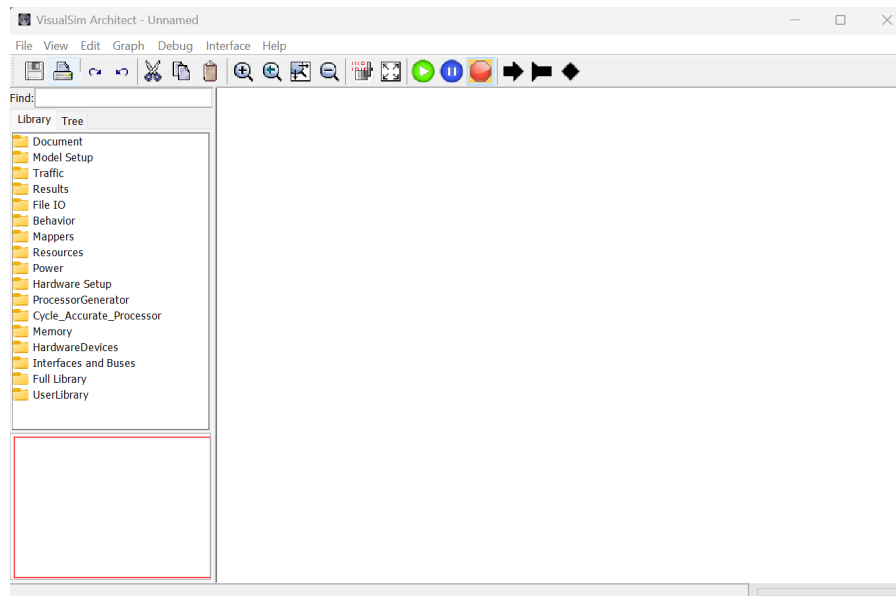


Figure 4. VisualSim Settings configuration

1. User can import C if the class files are outside of VS_AR directory. This allows the model to load up the class files without causing any error.
 2. The default locations and additional parameters for the simulator files are provided, such as Verilog, C++ compiler and System C.
 3. User can select the default browser to open the web pages.
 4. The Auto Save Frequency parameter defines the auto save time for the models being built or being open in VisualSim.
 5. Users can select the language of their choice.
 6. License server type of the current system can be selected.
 7. License server name of this system will be displayed.
2. Release Notes: User can view the latest updates and features supported in the current version VisualSim.
 3. About: opens the company website for more detailed information.
 4. Contact Support: Allows users to contact the support team through a mailing service.

3.2 Block Diagram Editor

The Block Diagram Editor (BDE) is the main editor in the VisualSim Architect environment. Users construct, debug, validate, and optimize the models of their system in this environment. Simulations to validate the model and get early feedback on the architecture are conducted from the Editor. After the model is stable and simulations are restricted to parameter changes, the Batch Mode can be used for large scale system analysis.



In the BDE, select blocks from the core libraries, place them on the BDE canvas, and then connect the blocks to represent the functions of the system. Group the various functions of the system to form a hierarchical model representing a sub system. Make use of hierarchy and aggregation to manage the complexity of large systems. Include adjustable parameters in any level so as to easily modify the system performance as per the requirement. Users can alter random number seeds to investigate system stability through analysis of variance. Prior to simulation, assign each parameter a value to execute a simulation over a range of conditions. Moreover, easily incorporate measured characteristics of the system, such as traffic traces, into VisualSim models.

3.2.1 Menu Commands

Menu	Sub Menu	Explanation	Shortcut
File	Open File	Select and open an existing model, text or html file on a local or networked disk.	Ctrl+O
	New	Open a new Block Diagram Editor (BDE), Icon Editor, FSM-Controller, FiniteStateMachine, Text Editor, Expression Evaluator or Tcl Evaluator.	
	Open Template List	Select and open existing models that are listed in the template page for different applications.	
	Save	Save the active file with its current file name, location, and format.	Ctrl+S
	Save as	Save by opening a window to change the file name, location or format.	
	Print	Print the active file. User can change the print options.	Ctrl+P
	Import	Import an existing file.	
	Export	Export to png, gif Page (Different from the dynamic Export to HTML), and Save in Library (Saves the entire model content as a library in User_Library).	
	Recent Files	View a list of the last 10 files that are saved for you to easily select.	

Menu	Sub Menu	Explanation	Shortcut
	Close	Close the active window.	Ctrl+W
	Exit	Close the application.	
View	Block Diagram Editor	Switch from the existing window to Block Diagram Editor window. This feature is disabled.	
	Simulation Cockpit	View Simulation Cockpit of the model to modify the parameters and to run the model. This feature is being disabled from the next release.	
	Zoom In	View a smaller part of the diagram in the screen view.	Ctrl+Shift+ Equal Symbol
	Zoom Reset	Reset the screen view to the original size.	Ctrl+M
	Zoom Fit	Fit the content of the block diagram into the viewable page.	Ctrl+Shift+ Minus Symbol
	Zoom Out	View a larger part of the diagram in the screen view.	Ctrl+Minus Symbol
	Full Screen	Hide the menu and folder bar to display the model to cover the full screen.	
Edit	Undo	Last action disappears from the screen.	Ctrl+Z
	Redo	Repeats your last action.	Ctrl+Y

Menu	Sub Menu	Explanation	Shortcut
	Cut	Removes the selection from the active file and places it on the clipboard.	Ctrl+X
	Copy	Copies the selection to the clipboard.	Ctrl+C
	Paste	Inserts the selected contents to the clipboard.	Ctrl+V
	Send to Back	When the blocks are overlapped, select the required block, and click Send to Back to place the block behind the adjacent block.	Ctrl+B
	Bring to Front	When the blocks are overlapped, select the required block and click Bring to Front to place the block in front of the adjacent block.	Ctrl+R
Graph	Find	Search for a keyword among the model blocks, parameters, expression, and memories.	Ctrl+F
	Automatic Layout	Rearranges the layout of the blocks, connectivity, and relations in the model. DO NOT USE. THIS IS A PROTOTYPE	Ctrl+T
	Configure Layout	Automatic Layout configuration. DO NOT USE. THIS IS A PROTOTYPE.	
	Instantiate Class	Select the class file to be placed in the BDE. This can be a Java class or an XML class. Use the File Selector to select the file. The file must be located in a directory that is within the CLASSPATH. Example: VisualSim.actor.lib.AbsoluteValue	

Menu	Sub Menu	Explanation	Shortcut
	CreateHierarchy	Select the blocks, connections, and parameters and create a Hierarchical_Block. Note: All parameters that are used in the selected blocks must be included in the selection; else multiple errors are generated. In some cases, the creation may have errors. All the connected wires must be selected so that the input/output ports for this hierarchical block will be generated automatically. All the created ports are consolidated at the top-left of the screen within the hierarchical block.	
	Xml Comparator	Compares the current Model with another model file.	
	New input port	Places an input port in the BDE.	
	New output port	Places an output port in the BDE.	
	New input/output port	Places an input/output port in the BDE.	
	New input multiport	Places an input multiport in the BDE.	
	New output multiport	Places an output multiport in the BDE.	

Menu	Sub Menu	Explanation	Shortcut
	New input/output multiport	Places an input/output multiport in the BDE.	
	New Relation	Places a Relation (black diamond) in the BDE.	
Debug	Listen to Simulator	This is a dialog window that is linked to the Digital Debug setting of the Digital Simulator. This displays the block activity and usage statistics.	
	Animate Execution	Set time to view the dynamic operation of the system model. Current window in the model is animated by highlighting the executing block.	
	Stop Animating	Turns-off the animation.	
Interface	Generate Wrapper	This is used with the C, Application Interface, SystemC, and Verilog links. This generates the required code files to interface with the third-party code and simulator. A single selection generates a wrapper for all the links.	
	Compile Wrapper	This goes through a sequence to compile the code generated by “Generate Wrapper” along with the user code. A single selection compiles all the blocks that have wrappers generated.	

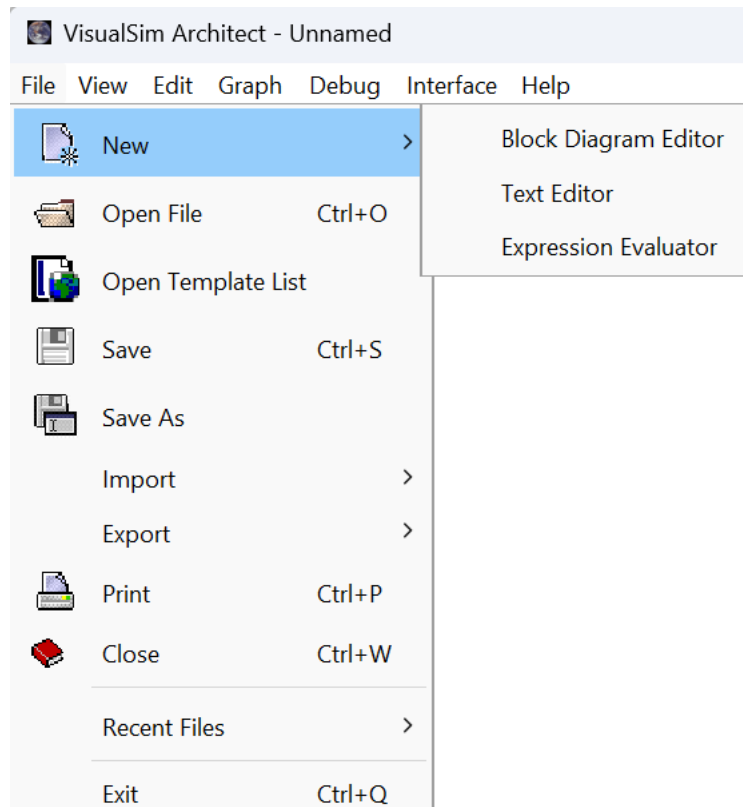
Menu	Sub Menu	Explanation	Shortcut
	Export to Html	This Menu item enables the user to open any model and generates an html document. The tool generates a separate page for each Hierarchical level. The page contains all the parameters, images, blocks, and any documentation described using Annotations. At the bottom of each page, the Applet containing the model is included. On the top page of the model, the simulation is executable. For all other pages, the view of the model alone is provided. When this export occurs, a separate html page is created for each Hierarchical block. The model that shows on these pages will not simulate. The top-level BDE that is exported will have the simulation ability.	
Help	About	View a brief description about the company and product.	
	Help	View a brief description about the company and product.	

3.2.2 Details of Menu and Toolbar Operation

3.2.2.1 Open a New Window

(Includes Block Diagram, Expression Evaluator, and Text Editor)

1. In your Mirabilis Design VisualSim program, click **File > New**.
2. From this menu, select the window of choice – Block Diagram, Expression Evaluator, or Text Editor.
3. A new window is displayed.



3.2.2.2 Existing Window (Model, Hierarchical Block, and URL)

1. In your Mirabilis Design VisualSim program, click File, and then click either **Open File**.
 - a) In the **Look in** list, click the drive or folder location that contains the files.
 - b) In the folder list, locate and open the folder that contains the file.
 - c) Select the file, and then click **Open**.

In addition, quick keys can be used:

1. CTRL+O – To open a file.

3.2.2.3 To open recently opened models – Recent Files




You can select from a list of most recently saved models.

3.2.2.4 Saving Options (Model, Block Diagram, FSM, and Text Editor)

1. To quickly save a document, click **Save** on the Standard Toolbar or CTRL+S or click File and then click Save.
2. To save the models in a different location, use the procedure below.
 - a. On the File menu, click **Save As**.
 - b. In the File name box, enter a new name for the file.
 - c. Click **Save**.

3.2.2.5 Editing Options (Model, Block Diagram, FSM, and Text Editor)



To move or copy a single item:

1. Select the item you want to move or copy.
2. Do one of the following:
 - ⇒ To move the item, click  **Cut** on the Standard toolbar.
 - ⇒ To copy the item, click  **Copy** on the Standard toolbar.
3. If you want to move or copy the item to another document, switch to the document.
4. Click where you want the item to appear.
5. Click  **Paste** on the Standard toolbar.

Alternate methods:

- Use the Cut, Copy, and Paste functions in the Edit Menu.
- Use Cut (CTRL+X), Copy (CTRL+C) and Paste (CTRL+V).

To undo or redo an operation on a model:





1. Click  undo or  redo in the Edit Menu or from the Toolbar.

To delete a block or link:

1. Select the block, state, link or transition you want to delete.





2. Press DELETE on keyboard, or click Delete or on the Edit menu, click Delete.

3.2.2.6 Zoom In or Out of a model

1. You can “Zoom In” to get a closer view of the model or FSM or “zoom out” to see more of the hierarchy at a reduced size.
2. Click **Zoom In**  and **Zoom Out**  on the standard toolbar.
3. In addition, you can **Zoom To Fit** to view the entire model or FSM on the screen.
4. Click **Zoom To Fit**  on the standard toolbar
5. Finally you can also **Zoom Reset** to bring the view back to the previously saved state.
6. Click the **Zoom Reset**  on the standard toolbar.
7. In addition to the standard toolbar option, you can also click **View** on the Menu and select the item.
8. Finally, you can also use the shortcut keys – Zoom in (CTRL+SHIFT+=), Zoom Out (CTRL+-), Zoom Reset (CTRL+M), and Zoom to Fit (CTRL+SIFT+-) to achieve the same result.

3.2.2.7 Stop and Resume

The simulation can be executed from the BDE window also. These operations are similar to the operations in the Simulation Cockpit:

- ⇒ Click **GO**  to start the simulation.
- ⇒ Click **STOP**  to terminate the simulation and not continue further.
- ⇒ Click **PAUSE**  to temporarily halt the simulation.
- ⇒ Click **RESUME**  to restart the simulation from the point where it was paused earlier.

The simulation can be stopped at any time. The displays can be viewed. The simulation can be resumed from that point forward.

3.2.2.8 Editorial Functions

VisualSim provides the following Editorial functions:.

Automatic Layout: This modifies the layout of the blocks and parameters in the current window. Do not use this feature as it incorrectly applies the layout for certain types of models.

Find: Search for a word among all the items in the model including the block names, expressions, parameters, memories, and annotations. This does not find the keyword in class files.

Import a Library: If you want to import a library acquired from another install of VisualSim or hand coded XML file, select Import Library from the File -> Import. Now, select the XML file.

Save in Library: The current model and its entire hierarchical list can be saved in the local library as a block. This is done by clicking the “Save in Library” on File -> Export.

Create Hierarchy: A section of the current window can be made into a hierarchical block by selecting the set of blocks and the relations and clicking “Create Hierarchy” in the Graph Menu. This function creates a new block with all of these selected blocks inside. All the required port for the interfaces are automatically created. It is important to select the links connected to the ports at the periphery of the area selected. This ensures that the correct ports are generated.

Instantiate Entity: This instantiates a sub-model or class as a block in the BDE window.

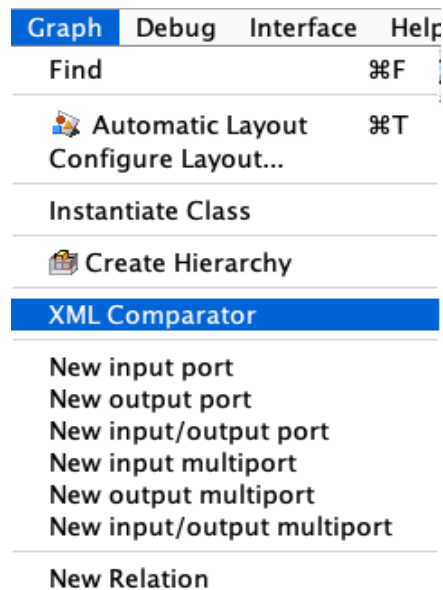
3.2.2.9 Xml Comparator

Xml Comparator can be used to compare two model XML files. The user selects a second model xml file to compare with the current open model file. The results of the comparison are stored in the separate folder in the same directory with the name- (Model Name)_XML _Comparator.

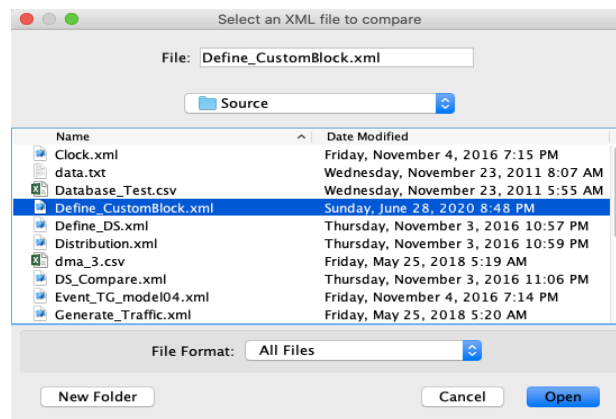
NOTE: A file path or file name containing hyphen or special character should not be selected for comparison. If selected, an Illegal Action Exception will be thrown.

Steps to perform the comparison

- Select the Xml Comparator for the Graph menu.



- Select another xml file for comparison.



- Go to the location of the Model xml file.

- Go into the folder (Model Name)_XML_Comparator created in the location.
- Open the .txt file contained in the folder to view the result.

```

XML_Comparator Results

Valid Relations === [Threads_and_Cores_relation11, Threads_and_Cores_relation10, Threads_and_Cores_relation9, Threads_and_Cores_

Valid Ports ===== [SystemResource.task_plot, Waveform_Plot.input, SystemResource2.task_plot, Waveform_Plot.input, SystemResour

Invalid Relations === [relation4, relation5, relation6]

Invalid Ports === [Temperature3.power_input, Script.input, PowerTable2.state_change, Temperature3.State_Change_input, Script.t

Actor and parameter comparison =====
Parameter value has been modified in Multi-Core_DVFS_Design.xml#Threads_and_Cores . Parameter Name = Num_Cores and new Parameter
Parameter value has been modified in Multi-Core_DVFS_Design.xml#Threads_and_Cores . Parameter Name = DVFS_Multiplier and new Par
Parameter value has been modified in Multi-Core_DVFS_Design.xml#Script . Parameter Name = Block_Name and new Parameter Value ="M
Parameter value has been modified in Main Block Diagram Editor window . Parameter Name = Num_Cores and new Parameter Value =4. C
Hierarchical block has been removed. Name = Temperature3 , Location = Multi-Core_DVFS_Design_1.xml#Temperature3
Hierarchical block has been removed. Name = Heat , Location = Multi-Core_DVFS_Design_1.xml#Heat
Hierarchical block has been removed. Name = Temperature Stats , Location = Multi-Core_DVFS_Design_1.xml#Temperature Stats

File = C:/VisualSim/VisualSim2340_64_Latest/VS_AR/demo/Power/Multi-Core_DVFS_Design.xml , All relations and Ports connected :
-----
* Threads_and_Cores_relation11
  SystemResource.task_plot
  Waveform_Plot.input
* Threads_and_Cores_relation10
  SystemResource2.task_plot
  Waveform_Plot.input
* Threads_and_Cores_relation9
  SystemResource3.task_plot




```

Figure 1

- Figure 1 shows the result of the comparison between two models

3.2.2.10 Model Hierarchy:

The model hierarchy can be viewed in two ways.

- ⇒ Click the Tab on the left section of BDE that says Tree. You can now select and view any Hierarchical block in the model.
- ⇒ Click the  icon to see more details about inside of each hierarchical block.
- ⇒ You can drag and drop entities from the hierarchy diagram on to the canvas.
- ⇒ Click the  or  icons to see the characteristics of the entity.

3.2.2.11 Find

This resides above the Library folder list. This is used to search all the blocks in the library folder list.

3.2.2.12 System Functions

Compile and Generate Wrapper in ModelBuilder

The Generate Wrapper option generates all the required interface blocks to construct a VisualSim block out of existing Custom C, Custom C++, SystemC, and Verilog wrapper. A note on the Command-line indicates that the operation has completed.

The Compile option compiles all the custom-coded block in the model. Current support is for C, C++, SystemC, and Verilog. The output window indicates when the compile is completed.

Memory Management or Garbage Collect

Operation: Select the Garbage Collect icon () in the standard toolbar.

Details: This is useful when there are custom-coded blocks. Use the Garbage collection at regular intervals and before starting a simulation.

Export to Html

Operation: Generate a document from the model.

Details: This Menu item enables the user to open any model and generate a html document. The tool generates separate pages for each Hierarchical level. The page contains all the parameters, blocks, and any documentation described using Annotations. At the bottom of each page, the Applet containing the model is included.

3.2.3 Listeners

- Listen to Block: Right click the required block and select Listen to Block. The Pop up text window displays the progression of execution in the block. Then run the model. This feature is not available for Hierarchical blocks and instantiated hierarchical classes. If you need to see the details of operation within a Hierarchical block, you need to Open Block and select the block to view the execution details. If you need to view the detailed operation of a Instantiated class, you need to select Open Instance and select the block to view the execution details.

- **Listen to Port:** Right click the required port and select Listen to port. The Pop up text window displays each Token passing through the port. You can also add a “relation” in front of the port then connect a Text_Display to this relation to view the activity.

3.2.4 Folders and Tree

Folders: The list of pre-built and user library are provided in the Library folder on the left-side of the Block Diagram Editor. Double-Click on any of the names to see the library blocks listed below. To use a block, simply select the block name in the Library and drag-n-drop the block into the Editor space.

Tree: The Tree tab allows the user to view the list of Hierarchical blocks in the model. The user can click on a name and the respective hierarchical block opens. After the block is open, go back to the original Window, select, and drag-n-drop the hierarchical into the canvas.

Find: The Find function in the Folder can search the entire library table and find the block that matches the search word. Currently this only searches for an exact match.

3.3 Text Editor and Listener Windows

3.3.1 Introduction

The text editor is used extensively for authoring code in the Script. It is also used to view text files. The Listener windows use the Text Editor. So, all the commands listed below are supported by those Listener also.

3.3.2 Commands

All the File, Edit, and Help commands match the Block Diagram Editor.

3.3.3 Quick Key Commands

The Text Editor does not have the Menu Bar. The user will have to know that CTRL+C, X, V, CTRL+F,H are for copy, cut, paste, find, and Find & Replace.

3.4 Plotters and VisualSim Post Processor

3.4.1 Introduction

VisualSim Plot provides comprehensive charting and analysis capability that enables the modeler to remain within the VisualSim environment for all complex analysis. The Plot graphs are instantiated in a model and are displayed as independent viewers when a simulation is started. The independent Post Processor application allows the user to set the view vs. save options for the plots and the plot file name. An associated index file allows for the plots from multiple runs to be combined and viewed in a single display. The graphs can be generated using various Display blocks in VisualSim and by importing XML data files from an external source. The graphs can also be embedded in an HTML executable specification as a Java Applet. Plot includes all the format functions required to modify the view. A variety of charts including, XY, bar, histogram, timeline, and signal are available.

VisualSim Plotting is a set of two-dimensional signal plotters that enhance the capability of the VisualSim simulation platform. Plotting is sometimes viewed as a "post" modeling activity. You can use the VisualSim Plot and the Post Processor application to graphically display and analyze data collected from the simulation. The Plotters can organize results into a variety of x-y graphs and histogram plots for graphical display. An XML extension language for plot data has been created and is available for the user to extend.

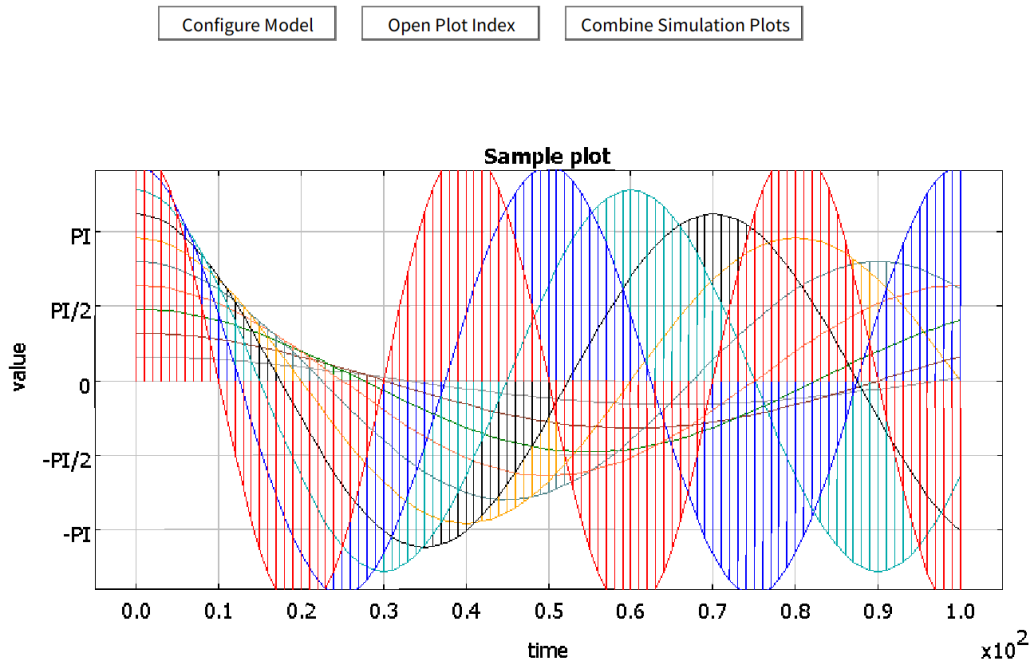


Figure 5.

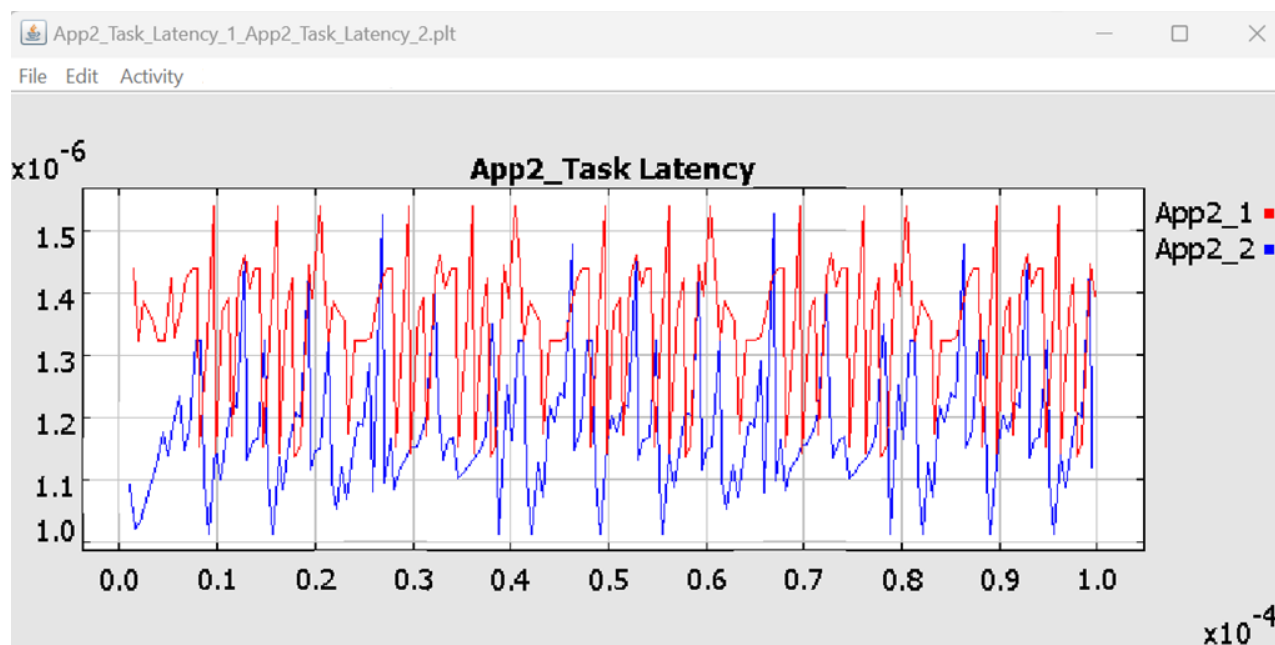
VisualSim Post Processor Window

The following are the key properties of the product:

- Embeddable in applets or applications
- Auto-ranging
- Automatic or manual labeling of axes
- Automatic or manual tick marks
- Logarithmic axes
- Live, animated plots
- Zooming upto the resolution
- Various plot styles: connected lines, scatter plot, bars, and so on
- Various point styles: none, dots, points, and unique marks
- Multiple data sets and a legend

- Color or black and white plotting
- Error bars
- Editable plots
- XML language for specifying plots
- Within a folder if there are multiple files having same name with consecutive numbering like myfile_1, myfile_2, myfile_3, then on selection, all 3 plots will be combined into a single plot and thus a single metric can be compared for different test cases easily.

3.4.2 Post Processor Plot Window



File	Open, Save, SaveAs, Export, Print and Close	
Edit	Format	Modify the layout of the plot.
Edit	Clear	Empties the plot.
	Fill	Populate the plot.

	Reset Axis	Reset axis to include all values.
Activity	About	View a brief description about the company and product.
	Configure Model	Configure a model to generate batch file and index file.
	Open Plot Index	Select an index file to view plots associated with each run.
	Combine Simulation Plots	Creates a plot with all the combinations of particular plot in a single window.

3.4.3 Using Plot

3.4.3.1 Zooming and filling

To zoom in, drag the left mouse button down and to the right to draw a box around an area that you want to see. To zoom out, drag the left mouse button up and to the right. To just fill the drawing area with the available data, type Control-I, invoke the fill command.

3.4.3.2 Printing and exporting

The File menu includes a Print and Export command. The Print command works as you expect. The export command produces a png, gif, and black-and-white encapsulated PostScript file (EPS) suitable for inclusion in word processors and Web Pages.

Note: At this time, the EPS file does not include preview data. This can make it somewhat awkward to work with in a word processor, as it will not be displayed by the word processor while editing (however it prints correctly). It is easy to add the preview data using the freely available program Ghostview¹. Just open the file using Ghostview and, under the edit menu, select "Add EPS Preview." Export facilities are also available from a small set of key bindings, which permits them to be invoked from applets (which have no menu bar) and from the invoked VisualSim Plot.

Note further that with applets, you may find it best to click near the title rather than clicking inside the graph itself and then type the command.

Exporting to the clipboard and to standard output, in theory, is allowed for applets, unlike writing to a file. Thus, these key bindings provide a simple mechanism to obtain a high-resolution image of the plot from an applet, suitable for incorporation in a document. However, in some browsers, exporting to standard out triggers a security violation. You can use Sun's *appletviewer* instead.

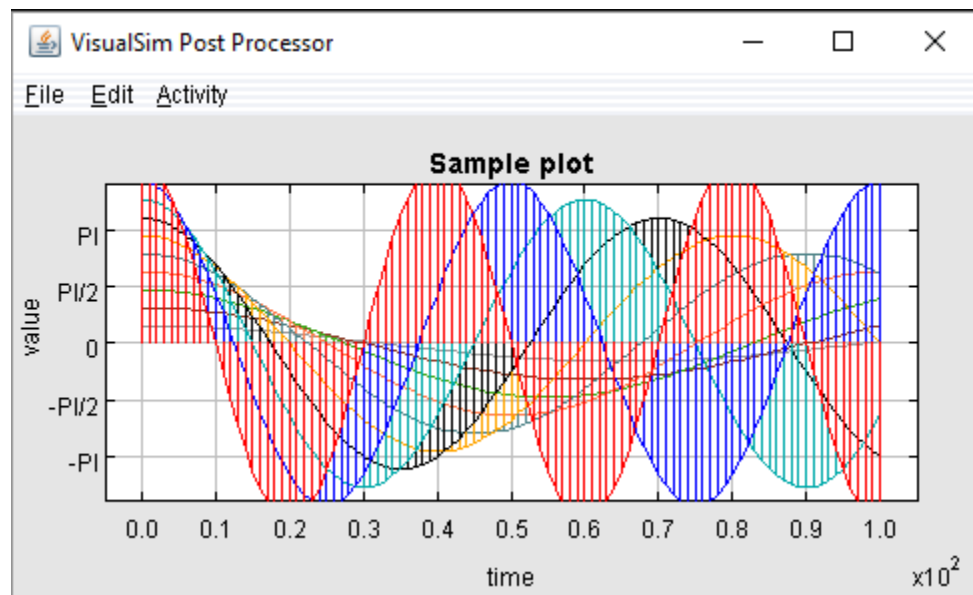


Figure 6.

AppletViewer

¹ Ghostview is available in <http://www.cs.wisc.edu/~ghost>.

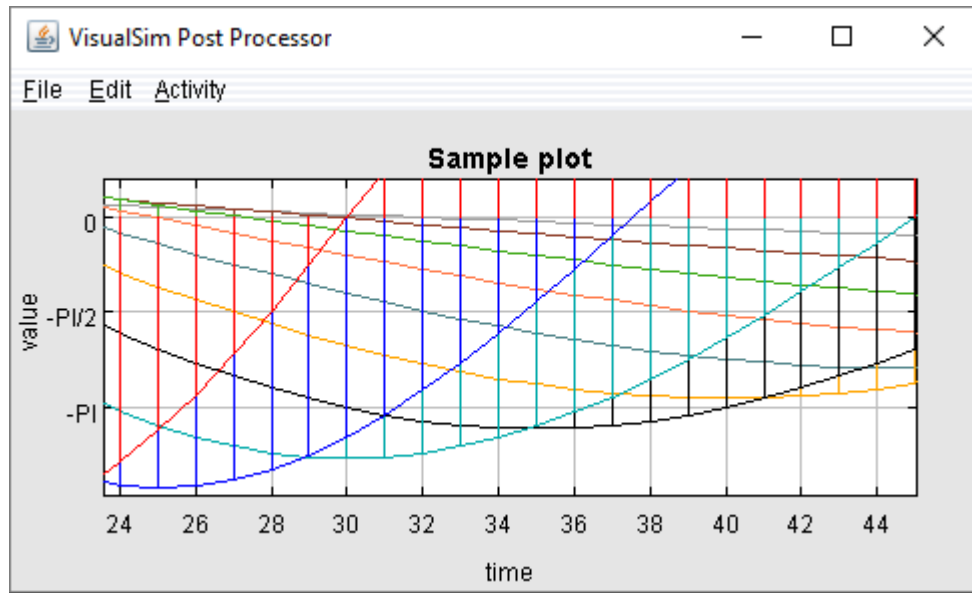


Figure 7.

Zoom in.


To zoom in, drag the left mouse button down and to the right to draw a box around the region you wish to see in more detail.

1. Control-c: Export the plot to the clipboard.
2. Shift-d: Dump the plot to a standard output.
3. Shift-e: Export the plot to a standard output in EPS format.
4. Shift-f: Fill the plot.

3.4.3.3 Editing the data

You can modify the data that is plotted by editing the plot file.

3.4.3.4 Modifying the format

You can control how data is displayed by invoking the Format command in the Edit menu or the Format Icon () on the Toolbar. This brings up a dialog like the one in Figure 4. At the

left, are the dialog and the plot before changes are made, and at the right are after changes are made. In particular, the stems are added, the lines connecting the data points are added, and the data points are rendered with points. Use the **Save** or **Save As** command in the File menu to save the modified plot. More sophisticated control over the plot can be had by editing the Plot file, which is a text file.

The entries in the format dialog are all straightforward to use except the "X Ticks" and "Y Ticks" entries. These are used to specify how the axes are labeled. The tick marks for the axes are usually computed automatically from the ranges of the data. Every attempt is made to choose reasonable positions for the tick marks regardless of the data ranges (powers of ten multiplied by 1, 2, or 5 are used).

To change what tick marks are included and how they are labeled, enter into the "X Ticks" or "Y Ticks" entry boxes a string of the following form:

`label position, label position, ...`

A label is a string that must be surrounded by quotation marks if it contains any spaces. A position is a number giving the location of the tick mark along the axis. For example, a horizontal axis for a frequency domain plot may have tick marks as follows:

XTicks: `-PI -3.14159, -PI/ 2 -1.570795, 0 0, PI/ 2 1.570795, PI 3.14159`

Tick marks could also denote years, months, days of the week, and so on.

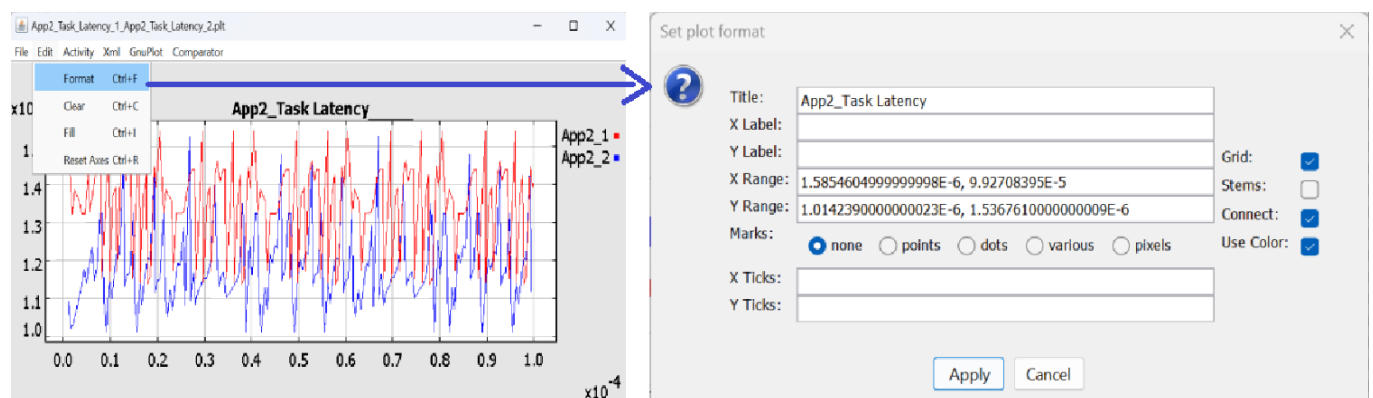


Figure 8.

Plot

The above plot displays the Control data using the Format command in the Edit menu.

3.4.4 Commands Configuring the Axes

The following commands are understood by the Application. These commands can be placed in a file and then read as a local file or via a URL. The recognized commands include:

- **TitleText:** string
- **XLabel:** string
- **YLabel:** string

These commands provide a title and labels for the X (horizontal) and Y (vertical) axes. A string is simply a sequence of characters, possibly including spaces. There is no need here to surround them with quotation marks, and in fact, if you do, the quotation marks will be included in the labels.

The ranges of the X and Y axes can be optionally given by commands like:

- **XRange:** min, max
- **YRange:** min, max

The arguments min and max are numbers, possibly including a sign and a decimal point. If they are not specified, then the ranges are computed automatically from the data and padded slightly so that datapoints are not plotted on the axes.

The tick marks for the axes are usually computed automatically from the ranges. Every attempt is made to choose reasonable positions for the tick marks regardless of the data ranges (powers of ten multiplied by 1, 2, or 5 are used). However, they can also be specified explicitly using commands such as:

- **XTicks:** label position, label position, ...
- **YTicks:** label position, label position, ...

A label is a string that must be surrounded by quotation marks if it contains any spaces. A position is a number giving the location of the tick mark along the axis. For example, a horizontal axis for a frequency domain plot might have tick marks as follows:

```
XTicks: -PI -3.14159, -PI/ 2 -1.570795, 0 0, PI/ 2 1.570795, PI  
3.14159
```

Tick marks could also denote years, months, days of the week, and so on.

By default, tick marks are connected by a light grey background grid. This grid can be turned off with the following command:

- **Grid:** off

It can be turned back on with the grid.

- **Grid:** on

Also, by default, the first ten data sets are shown each in a unique color. The use of color can be turned off with the command:

- **Color:** off

It can be turned back on with a check box in the Format window.

- **Color:** on

3.4.5 Commands for Plotting Data

The following set of commands are understood by the Application support:

- **Specification** of data to be plotted
- **Control** over how the data is shown

The style of marks used to denote a data point is defined by one of the following commands:

- ◆ **Marks:** none
- ◆ **Marks:** points
- ◆ **Marks:** dots

- ◆ **Marks: various**
- ◆ **Marks: pixels**

Here, points are small dots, while dots are larger. If various is specified, then unique marks are used for the first ten data sets, and then recycled. If pixels are specified, then a single pixel is drawn.

Using no marks is useful when lines connect the points in a plot, which is done by default. If the above directive appears before any DataSet directive, then it specifies the default for all data sets. If it appears after a DataSet directive, then it applies only to that data set.

To disable connecting lines, use:

- ◆ **Lines: off**

To re-enable them, use

- ◆ **Lines: on**

You can also specify "impulses", which are lines drawn from a plotted point down to the x axis. Plots with impulses are often called "stem plots." These are off by default, but can be turned on with the command:

- ◆ **Impulses: on**

or back off with the command

- ◆ **Impulses: off**

If that command appears before any DataSet directive, then the command applies to all datasets. Otherwise, it applies only to the current data set.

To create a bar graph, turn off lines and use any of the following commands:

- ◆ **Bars: on**
- ◆ **Bars: width**
- ◆ **Bars: width, offset**

The **width** is a real number specifying the width of the bars in the units of the x axis. The **offset** is a real number specifying how much the bar of the i-th data set is offset from the

previous one. This allows bars to "peek out" from behind the ones in front. Note that the forward-most data set is the first one. To turn off bars, use:

♦ `Bars: off`

To specify data to be plotted, start a data set with the following command:

♦ `DataSet: string`

Here, `string` is a label that appears in the legend. It is not necessary to enclose the string in quotation marks.

To start a new dataset without giving it a name, use:

♦ `DataSet:`

In this case, no item appears in the legend.

If the following directive occurs:

♦ `ReuseDataSets: on`

then datasets with the same name are merged. This makes it easier to combine multiple data files that contain the same datasets into one file. By default, this capability is turned off, so datasets with the same name are not merged.

The data itself is given by a sequence of commands with one of the following forms:

♦ `x, y`
♦ `draw: x, y`
♦ `move: x, y`
♦ `x, y, yLowErrorBar, yHighErrorBar`
♦ `draw: x, y, yLowErrorBar, yHighErrorBar`
♦ `move: x, y, yLowErrorBar, yHighErrorBar`

The **draw** command is optional, so the first two forms are equivalent. The **move** command causes a break in connected points, if lines are being drawn between points. The numbers `x` and `y` are arbitrary numbers as supported by the Double parser in Java (e.g. "1. 2", "6.39e-15", and so on.). If there are four numbers, then the last two numbers are assumed to be the lower and upper values for error bars. The numbers can be separated by commas, spaces or tabs.

3.5 VisualSim Post Processor

3.5.1 Introduction

VisualSim Post Processor application is a network-based application that can be shared across a division or a corporation. VisualSim Post Processor is platform and OS-independent, enabling different users to maintain the same experience. In addition, a mobile provision enables the Post Processor to be used offline. This enables flexible usage during travel, design review, and remote presentations.

3.5.2 Key Features

1. Uses an open XML DTD for data management with a .plt extension.
2. Graphical user interface with built-in error checking.
3. Manages the plot selection for viewing and display.
4. Configure a model to save the plots in a specific destination directory.
5. Creates an index for all the simulation runs of the model along with the associated plot.
6. Open plots from different simulation runs simultaneously for easy comparison.
7. Combine datasets from multiple plots into a single graph (Future release).

3.5.3 Quick Key Command

File	Same as the Block Diagram Operations	
	Export	Export plot image as a eps, gif, and png.
Edit	Format	Modify the layout of the plot.
	Clear	Empties the plot.

	Reset Axis	Reset axis to include all values.
Activity	About	View a brief description about the company and product.
	Configure Model	Create an index for the plot and select the simulation parameters to save.
	Open Plot Index	Select plots from multiple simulations to combine and view.
	Combine Simulation Plots	Plot all the individual plots in a directory in a single window.

3.5.4 Usage

To understand the usage of the Post Processor, let us look at the example available in the Post Processor install under Model_Plots.

- **Step 1:** From the VisualSim Architect Block Diagram Editor, add the PlotManager block (Results > PlotManager) to VME_Bus_Model1.xml. The Plot_Path is updated to the destination of the output plot files (make sure the path is added within "" and the slashes are backward slashes).
- **Step 2:** From the Post Processor window, select Configure Model. Now, select the VME_Bus_Model1.xml. Click the parameters that are varied to create the different unique simulation runs of the model. Also specify which plots need to be viewed and which one must be saved. Click OK or Apply + OK. This creates an index file and a Batch file in the destination directory.
- **Step 3:** Now run the simulation in batch or interactive mode by varying the selected parameters. If parameters not selected in Step 2 are used, these runs are not to be treated as unique. Other runs could overwrite the plots from these runs.

- **Step 4:** Now return to the Post Processor. Select Open Plot Index, select the VME_Bus_Model1_Index.xml that is located in the destination directory. For Plot configuration file, select any plot in the destination directory related to the model. The list of simulations runs for the model and the associated plots are listed. Select any number of runs and any plot. Click View Plot. The plots are now displayed as separate windows. Click View Selected Traces. The plots are displayed in a single window for comparison.

3.5.5 VisualSim Architect Models

VisualSim Post Processor works with two Display (Unbuffered and Buffered), TimeDataPlotter, Histogram Plotter, XYPlotter, DS_TimeDataPlotter, DS_XYPlotter, ArrayPlotter and BarGraph blocks of the VisualSim model. The Display blocks are supported for the purpose of setting View and Save only. They can not be opened with VisualSim Post Processor. The graphs are stored in XML files with a .plt extension. Simulation results from a single model are stored together and are tagged with the simulation run ID and the plot name. A simulation index file maintains the connection between the simulation run and the graphs. The simulation runs are uniquely identified by the variation of the parameter.

3.5.6 Usage

VisualSim provides all the standard graphical user interface capability. These include edit functions such as open a new plot file (single file), print, save, and export to eps format. The format functions provide the ability to modify the view of a single plot. The Activity menu configures simulation model and opens the index file containing references to plot data.

3.5.7 Post Processor Activity

Configure Model- This is used to configure a model to generate Batch file and Index file. Step-by-step instruction:

1. Add **PlotManager** block in VisualSim Block Diagram of the model to be configured.

2. Specify output directory path in the PlotManager block parameter Plot_Path (path within double quotes and the slashes in backward slash).
3. Select **Configure Model** from the main Post Processor application window.
4. From the parameter list, select the parameters that are modified during this batch run of simulation results. Leave out parameters that are not modified during this run. You can always go back and redo this listing for a different run.
5. Format for parameter Range and step: Range can support range of parameter as start and end range with colon eg: 10:100. Range also support all possible values eg: 10, 25, 50, 100. For Range format the Step must be set with the a value that increase for every run. For all possible values the Step must have None.

Parameters

Parameter Name	Range	Step
<input checked="" type="checkbox"/> Latency	1:20	5
<input checked="" type="checkbox"/> Sim_Time	2.0,5.0,10.0	None

In this case the Latency range is from 1 to 20 and the step is 5, so the bath mode simulation will create runs as 1, 6, 11 and 16 for latency. For Sim_Time the defined values will be used for each run.

6. For each plot in the model, you can view (during the simulation), save or both. Select the radio button to enable this.

Parameters

Plots

TimeDataPlotter	<input type="checkbox"/> View	<input type="checkbox"/> Save	
-----------------	-------------------------------	-------------------------------	--

7. Click **OK**. You will get a pop window that says Batch File Generated.

Open Plot Index - This opens the reference to the simulations and their respective plots.

1. After completing the simulation runs, return to the Post Processor.
2. Select **Open Plot Index**.
3. Select the index file . (Model name _Index.xml)
4. Select any one of the plot file in Plot configuration file selection window.
5. A list of simulations runs identified by the values of the selected parameters are displayed on the top-half of the window. A list of the plots is displayed in the bottom.
6. Select one or more simulation runs and one or more plots (all is also an option).
7. Click View Plots, the selected plots open in separate windows.
8. Click View Selected Traces to get the plots in a single window.
9. Sometimes, a blank plotter screen may appear. Click on Format → Fill to view the Files.

Note: All plot windows appear one on top of the other.

3.5.8 Packaging and Archiving

To package and ship the simulation results, simply tar or zip the output directory. This can be shipped to the end-user. The end-user then unzips the content. The model index is contained in the package and can be used for viewing the graphs.

3.6 Icon Editor

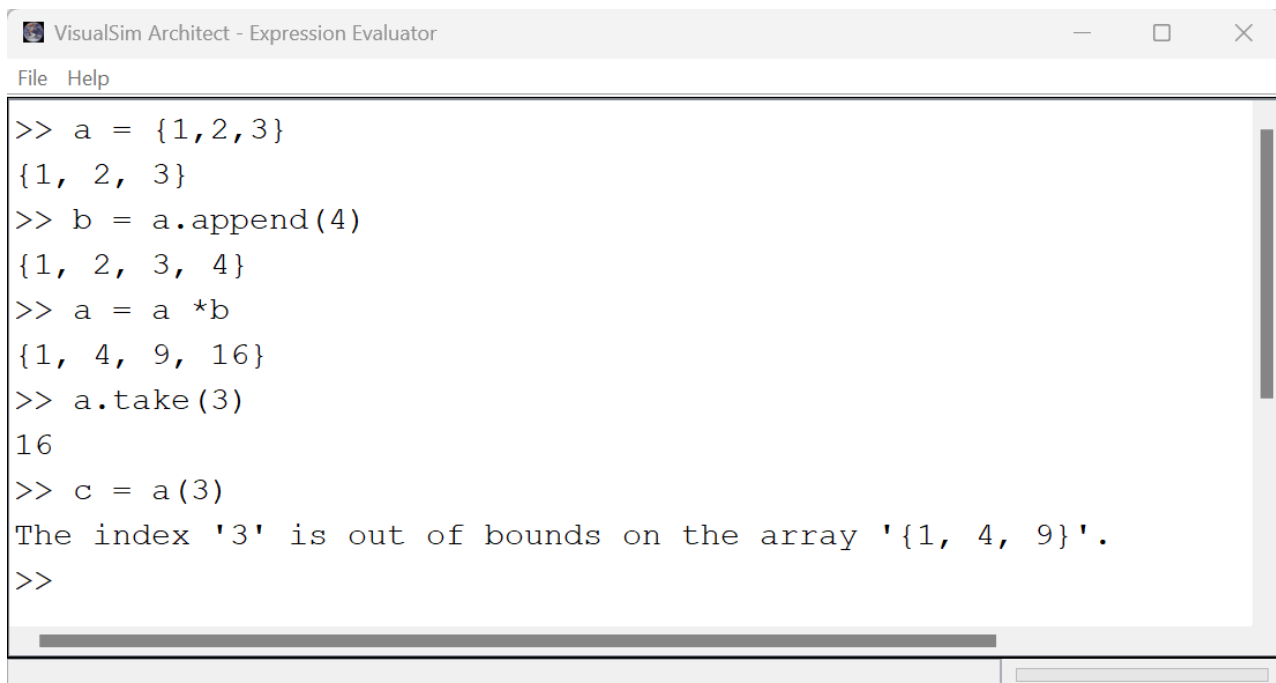
This Editor is used to create custom icons for the blocks. This editor has a number of standard shapes, lines, colors, and functions to access block parameters.

If editing a block in a model, then the resulting icon details are stored in the model XML file. If a new icon is being constructed, the resulting icon is stored in an XML format.

An icon can be constructed using shapes such as lines, circles, and rectangles. Alternately, there can be an image used for the icon. The one restriction is that the icon must be in the same directory as the XML file containing the reference.

3.7 Expression Evaluator

This uses the extensive expression language available in VisualSim to evaluate equations and algorithm. This can be used for conducting quick studies that are algorithm in nature and do not require simulation. Also, this can be used to test a sequence of expressions and see if they work before using in the Script Block. This supports variables defined in the environment and can be used with any of the functions in the RegEx language. Below is an example:



```
>> a = {1,2,3}
{1, 2, 3}
>> b = a.append(4)
{1, 2, 3, 4}
>> a = a *b
{1, 4, 9, 16}
>> a.take(3)
16
>> c = a(3)
The index '3' is out of bounds on the array '{1, 4, 9}'.
>>
```

Notice that the errors in the expressions are also displayed here. This can be used to fine tune your sequences. This window does not support C language functions such as while, if-else, and switch-case.

To learn more about the RegEx language, refer to the RegEx Expression documentation in Learn VisualSim section of tool front page.

3.8 Simulation Cockpit

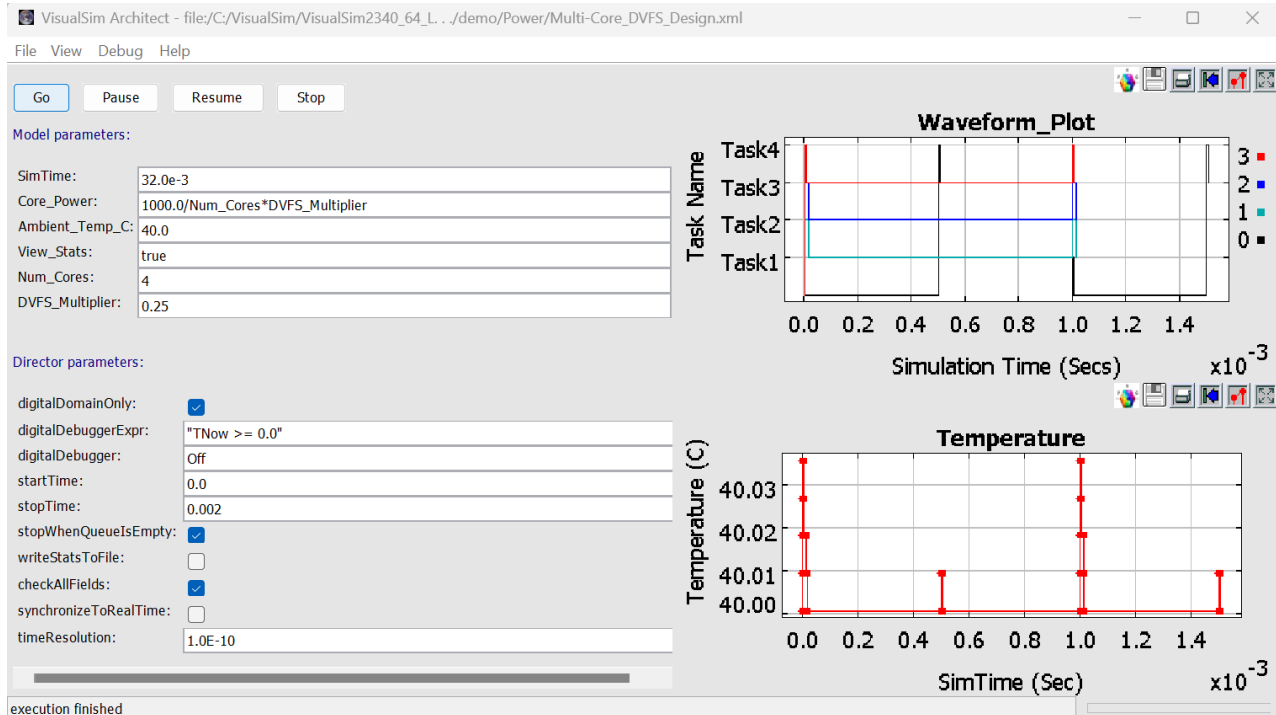

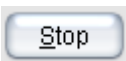




Figure 9. Simulation Cockpit

The simulation cockpit provides a single window to set parameter values, simulate, listen to simulator activity, and view the plotted result. The Simulation Cockpit has been included with VisualSim as a legacy support.

3.8.1 Operation

These functions are the buttons in the Simulation Cockpit.

- ⇒ Click **GO**  to start the simulation.
- ⇒ Click **STOP**  to terminate the simulation and not continue further.
- ⇒ Click **PAUSE**  to temporarily halt the simulation.

⇒ Click **RESUME**  to restart the simulation from the point where it was paused earlier.

3.8.2 Listen to the Manager and Simulator

Operation: Click **Debug** in the Menu of the Simulation Cockpit and then click **Listen to Manager/Simulator**.

Details: The **Listen to Manager** has been disabled to speed up the simulation. Future applications of this interface are being considered. The **Listen to Simulator** can report the sequence of block execution and the model profile statistics at the end of the simulation. The setting of the Digital Simulator determines the output. Other simulators do not provide any data to this window.



Note

The **Listen to Simulator** is used in conjunction with the Digital debugger. Refer to the Digital debugger under Model debugging sections.

3.8.3 Pause and Resume

The simulation can be paused at any time. The display can be viewed and the simulation can be resumed from that point forward.



Note


For the Pause and Resume to work effectively, the Listen to Manager Dialog window must be started. The simulation is faster than the keyboard access and the Pause request is lost.


3.8.4 Simulation Parameters


Modify the values of the parameters in the Simulation Cockpit and restart the simulation. There are two types of parameters that can be finalized prior to simulation - Simulation parameters


and Model parameters. The Simulation parameters modify the actions of the kernel while the model parameters modify the operating of the simulation.


3.8.5 Plot viewers in the Simulation Cockpit


Export : Click this  icon to export the plot to different file type. Specify location and file name and click **Export**.

Save : Click this  icon to save the data into a file. Specify location and file name and click **OK**.

Print : Click the  icon to print the window. In the Print window, select printer, and then click **OK**.

Rescale : Click the  icon to rescale the plot to fit the data.

Format : Click the  icon, specify the various attributes, and click **OK**.

Reset : Click the  icon to reset X and Y axis to their original values.

4 Parsers

The VisualSim Architect provides different parser for specific purpose that can be used in the VisualSim environment. For example, if the user has to generate a file for a model input and it requires aspecific format then these parser helps user to obtain the files in the required format by reducing the amount of time spend on modifying and updating the files. Other caser are like generating a file for analysis such as axi parser and XML comparator.

4.1 ARXML parser

The parser takes a ARXML (Autosar XML) file as input which contains the data in a package format. The parser extract the data information using the tag list file (arxml_parser_tag_list.csv) that user kept in the folder where the .arxml fie is located, and then it provide the output files that can be loaded to a model such as AUTOSAR.

How to use it:

Select Parser in tool home page and select Arxml Parser option.

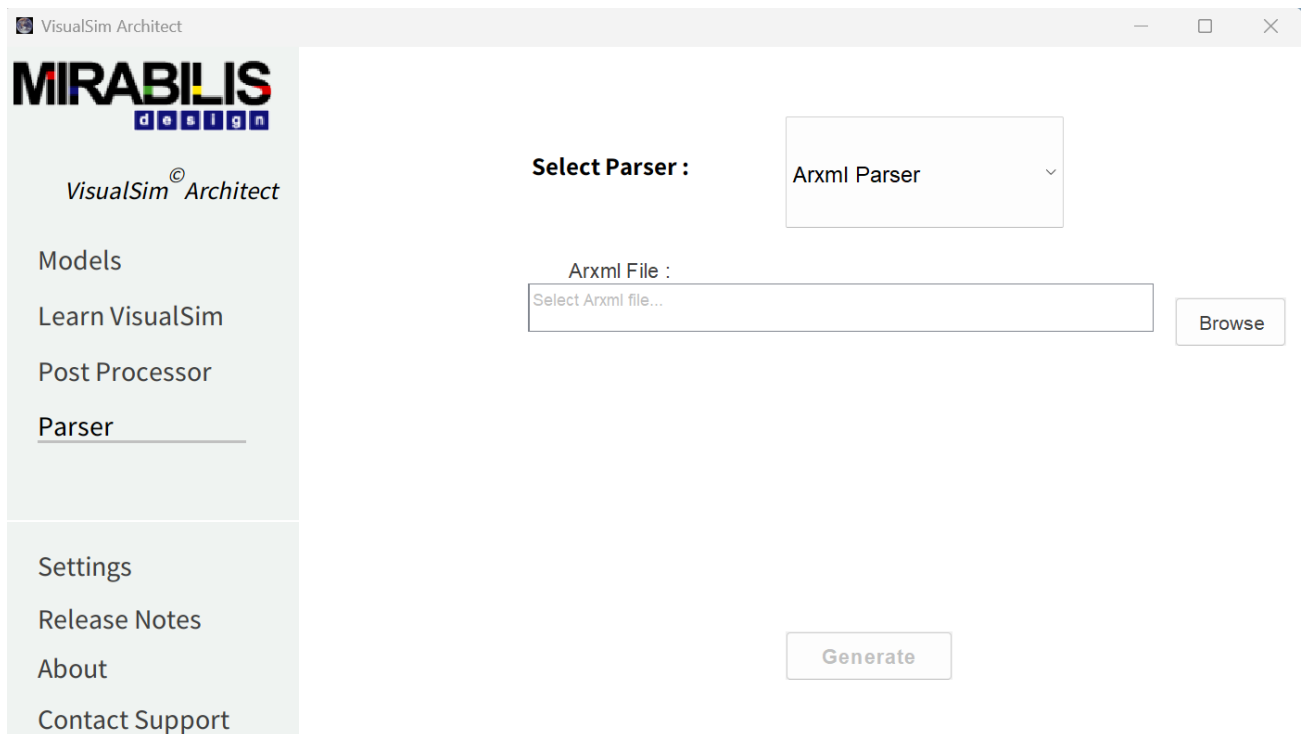
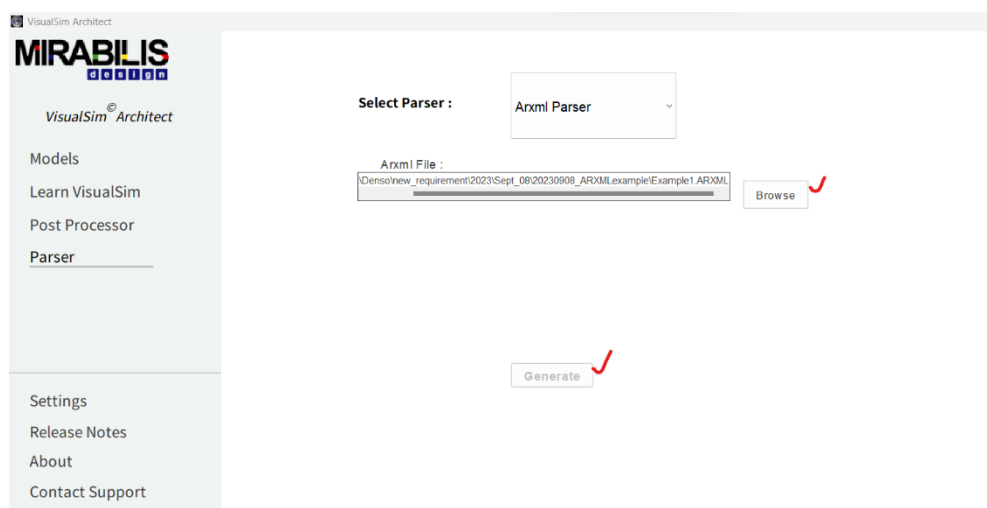


Figure 10. ARXML parser Window

Select Browse and choose the ARXML file that you want to parse. Once the file is selected, Click generate.



The screenshot shows the VisualSim Architect web interface. On the left is a sidebar with the Mirabilis logo and navigation links: Models, Learn VisualSim, Post Processor, Parser (selected), Settings, Release Notes, About, and Contact Support. The main area has a 'Select Parser' dropdown set to 'Arxml Parser'. Below it is an 'Arxml File' input field containing a file path, with a 'Browse' button and a red checkmark. At the bottom center is a 'Generate' button with a red checkmark.

The format of the arxml_parser_tag_list.csv file to extract the data from the arxml file.

	A	B
1	Tags	
2	</Port>	
3	</SHORT-NAME>	
4	</LENGTH>	
5	</UNUSED-BIT-PATTERN>	
6	</FRAME-LENGTH>	
7	</PACKING-BYTE-ORDER>	
8	</PDU-REF>	
9	</START-POSITION>	
10	</COMM-CONTROLLER-REF>	
11	</COMMUNICATION-DIRECTION>	
12	</SOURCE-I-PDU-REF>	
13	</TARGET-I-PDU-REF>	
14	</IDENTIFIER>	
15	</CATEGORY>	
16	</DATA-CONSTR-REF>	
17	</IMPLEMENTATION-DATA-TYPE-REF>	
18	</DEFINITION-REF>	
19	</VALUE>	
20	</I-SIGNAL-REF>	
21	ALL	

Check for the string “AXI_” in the file and replace the word AXI with the bus name (11 locations uses bus name in the python file). For example, if bus name is “AX_Top”, replace AXI_ to AXI_Top_. Once the changes are made save the file and open the tool.

4.2.3 Parser

Go to the tool page and select the Parser, then select AXI Parser in the pull down option.

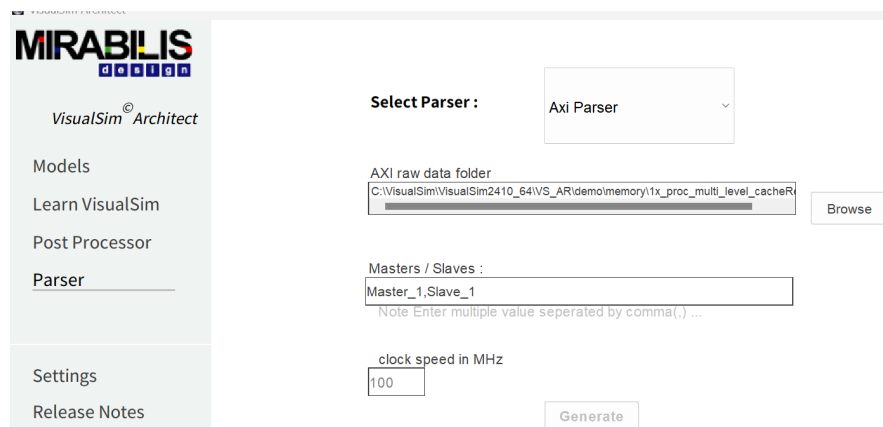


Figure 12. AXI pasreser window

Select the folder which contain the data files in AXI raw data folder.

Update the Masters/Slaves option according to the number of master and slave configured in the model. Format is Master_1,Master_2, Slave_1,Slave_2 .

If user entered wrong name like Master 14 but that is not configured in the model during simulation it will throw an error. Please update the master salve name correctly.

Select the speed at the bus was simulated. Eg: 100, which is 100Mhz.

Once the parmaters are configured correctly click Generate. You will get a gnu plot file in the result directory where the data files are stored. User can view the plot in GNU plot software.

4.3 GEM5 parser

The Gem5 parser helps the user to generate the trace file that can be used to run an application program in Hybrid processor. Select parser from tool home page and select Gem5 Parser option.

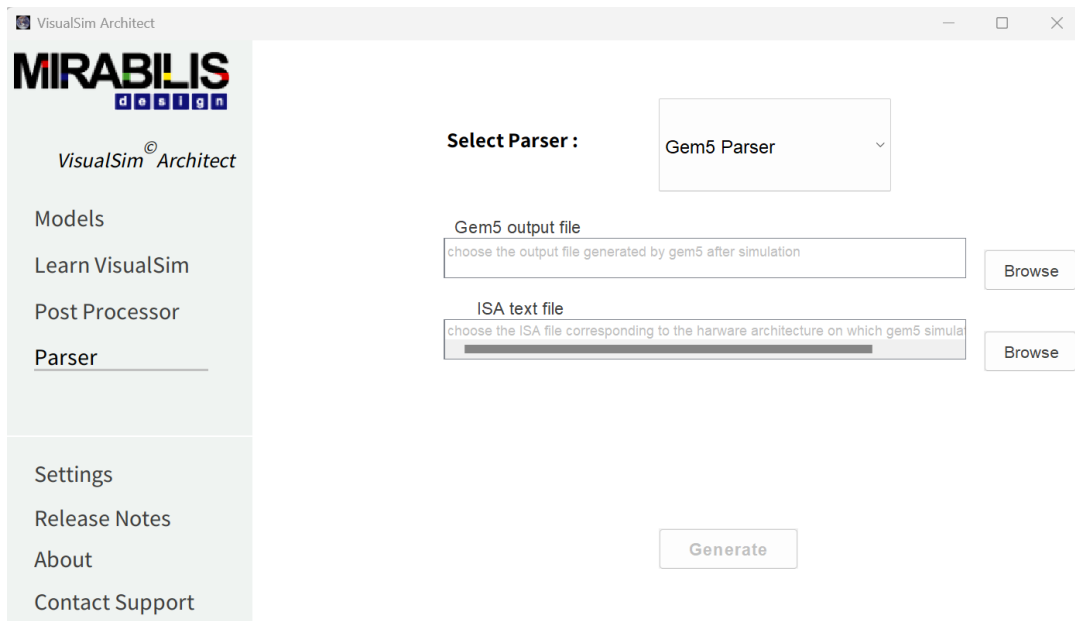


Figure 13. Gem5 Parser window

Select the gem5 executed trace in Gem5 Output file section, and select the instruction set file for the target processor in ISA text file section.

```

80919: system.cpu_cluster.cpus: T0 : 0x400be4 @_start : movz x29, #0, #0 : IntAlu : D=0x0000000000000000
80919: system.cpu_cluster.cpus: T0 : 0x400be8 @_start+4 : movz x30, #0, #0 : IntAlu : D=0x0000000000000000
81252: system.cpu_cluster.cpus: T0 : 0x400bec @_start+8 : orr x5, xzr, x0 : IntAlu : D=0x0000000000000000
153846: system.cpu_cluster.cpus: T0 : 0x400bf0 @_start+12 : ldr x1, [sp] : MemRead : D=0x0000000000000001 A=0x7f
153846: system.cpu_cluster.cpus: T0 : 0x400bf4 @_start+16 : add x2, sp, #8 : IntAlu : D=0x00000007ffffefe78
154179: system.cpu_cluster.cpus: T0 : 0x400bf8 @_start+20 : add x6, sp, #0 : IntAlu : D=0x00000007ffffefe78
154179: system.cpu_cluster.cpus: T0 : 0x400bfc @_start+24 : movz x0, #0, #48 : IntAlu : D=0x0000000000000000
154845: system.cpu_cluster.cpus: T0 : 0x400c00 @_start+28 : movk x0, #0, #32 : IntAlu : D=0x0000000000000000
155178: system.cpu_cluster.cpus: T0 : 0x400c04 @_start+32 : movk x0, #64, #16 : IntAlu : D=0x0000000000040000
155511: system.cpu_cluster.cpus: T0 : 0x400c08 @_start+36 : movk x0, #1312, #0 : IntAlu : D=0x00000000000400520
155844: system.cpu_cluster.cpus: T0 : 0x400c0c @_start+40 : movz x3, #0, #48 : IntAlu : D=0x0000000000000000
156177: system.cpu_cluster.cpus: T0 : 0x400c10 @_start+44 : movk x3, #0, #32 : IntAlu : D=0x0000000000000000
156510: system.cpu_cluster.cpus: T0 : 0x400c14 @_start+48 : movk x3, #64, #16 : IntAlu : D=0x0000000000040000
156843: system.cpu_cluster.cpus: T0 : 0x400c18 @_start+52 : movk x3, #5968, #0 : IntAlu : D=0x00000000000401750
157176: system.cpu_cluster.cpus: T0 : 0x400c1c @_start+56 : movz x4, #0, #48 : IntAlu : D=0x0000000000000000
157509: system.cpu_cluster.cpus: T0 : 0x400c20 @_start+60 : movk x4, #0, #32 : IntAlu : D=0x0000000000000000
157842: system.cpu_cluster.cpus: T0 : 0x400c24 @_start+64 : movk x4, #64, #16 : IntAlu : D=0x0000000000040000
158175: system.cpu_cluster.cpus: T0 : 0x400c28 @_start+68 : movk x4, #6160, #0 : IntAlu : D=0x00000000000401810
158508: system.cpu_cluster.cpus: T0 : 0x400c2c @_start+72 : bl <_libc_start_main> : IntAlu : D=0x00000000000400c30
262737: system.cpu_cluster.cpus: T0 : 0x401090 @ _libc_start_main : stp

```

Figure 14. Gem5 executed trace format

```

/* Instruction Set or File Path. */

Mnew Ra Rb Rc Rd Re Rf Rg Rh ; /* Label */
ARM BCH ALU ALUMUL FPNEOSIMD LDSTR STRDT ;

BCH      INT_1 INT_2 ; /* BRANCH */
ALU      INT_3 INT_4 ; /*Single cycle*/
ALUMUL   INT_5 INT_6 ; /* Multi cycle*/
FPNEOSIMD FP_1 FP_2 ; /*Floating point/NEON/ASIMD instructions*/
LDSTR    INT_7 INT_8 ; /*Load/Store instructions*/
STRDT    INT_9 INT_10 ; /*Store Data*/

begin size_config
Read 11 32 INT_7[1:198] ;
Write 11 32 INT_7[199:500] ;
end size_config

begin execUnit_config
Queue_Size INT 1 16 ;

```

Figure 15. Instruction set file format

Once you selected the file, select Generate.

4.4 GCC Parser

The GCC parser helps the user to convert the compiled disassembly file into the trace file that can be loaded to the micro- architecture processor model. The binary file and the disassembly file must be generated for a target processor in linux.

4.4.1 Binary and disassembly files

The following command is an example for a arm cortex A77 processor.

```
aarch64-none-linux-gnu-gcc -mtune=cortex-a77 -mcpu=cortex-a77 -O3 -static -o dpt dpt.c -lm
```

This will provide a binary file which must be used to generate the disassembly. The following command convert the compiled binary code into disassembly.

```
aarch64-none-linux-gnu-objdump -d dpt | tee cmdPrint_dpt.txt
```

In this case the binary file name is dpt and the disassembly file name is cmdPrint_dpt.txt

Once the disassembly file is generated use that file in the GCC parser. The following image shows the disassembly file format.

```

1
2 dpt: file_format elf64-littlearch64
3
4
5 Disassembly of section .init:
6
7 000000000400258 <.init>:
8 400258: a9bf7b7d stp    x29, x30, [sp, #-16]!
9 40025c: 910003fd mov    x29, sp
10 400260: 940000f9 bl     400644 <call_weak_fn>
11 400264: a8c17bfd ldp    x29, x30, [sp], #16
12 400268: d65f03c0 ret
13
14 Disassembly of section .plt:
15
16 000000000400270 <.plt>:
17 400270: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
18 400274: f9400211 ldr     x17, [x16]
19 400278: 91000210 add     x16, x16, #0x0
20 40027c: d61f0220 br     x17
21 400280: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
22 400284: f9400611 ldr     x17, [x16, #8]
23 400288: 91002210 add     x16, x16, #0x8
24 40028c: d61f0220 br     x17
25 400290: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
26 400294: f9400a11 ldr     x17, [x16, #16]
27 400298: 91004210 add     x16, x16, #0x10
28 40029c: d61f0220 br     x17
29 4002a0: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
30 4002a4: f9400e11 ldr     x17, [x16, #24]
31 4002a8: 91006210 add     x16, x16, #0x18
32 4002ac: d61f0220 br     x17
33 4002b0: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
34 4002b4: f9401211 ldr     x17, [x16, #32]
35 4002b8: 91008210 add     x16, x16, #0x20
36 4002bc: d61f0220 br     x17
37 4002c0: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
38 4002c4: f9401611 ldr     x17, [x16, #40]
39 4002c8: 9100a210 add     x16, x16, #0x28
40 4002cc: d61f0220 br     x17
41 4002d0: f0000450 adrp   x16, 48b000 <_GLOBAL_OFFSET_TABLE_+0xc0>
42 4002d4: f9401a11 ldr     x17, [x16, #48]
43 4002d8: 9100c210 add     x16, x16, #0x30
44 4002dc: d61f0220 br     x17

```

Figure 16. Disassembly file format for GCC parser

4.4.2 GCC file parsing

Select Parser in the tool page and select GCC Parser in the pull down option.

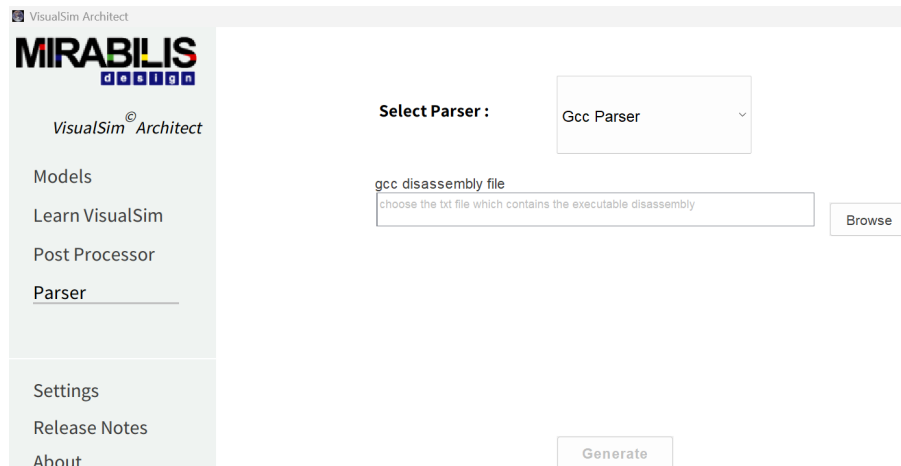


Figure 17. GCC parser window

Select the disassembly file generated using the commands mentioned in the previous sub section in the gcc disassembly file option. Click Generate, once the parsing is completed a file named Trace_1.txt will be generated in the same folder.

1	Address	Instruction	Registers;
2	4197348	mov	{'x29', '#0'};
3	4197352	mov	{'x30', '#0'};
4	4197356	mov	{'x5', 'x0'};
5	4197360	ldr	{'x1', '[sp]'};
6	4197364	add	{'x2', 'sp', '#8'};
7	4197368	mov	{'x6', 'sp'};
8	4197372	movz	{'x0', '#0', 'lsl #48'};
9	4197376	movk	{'x0', '#0', 'lsl #32'};
10	4197380	movk	{'x0', '#64', 'lsl #16'};
11	4197384	movk	{'x0', '#1312'};
12	4197388	movz	{'x3', '#0', 'lsl #48'};
13	4197392	movk	{'x3', '#0', 'lsl #32'};
14	4197396	movk	{'x3', '#64', 'lsl #16'};
15	4197400	movk	{'x3', '#5968'};
16	4197404	movz	{'x4', '#0', 'lsl #48'};
17	4197408	movk	{'x4', '#0', 'lsl #32'};
18	4197412	movk	{'x4', '#64', 'lsl #16'};
19	4197416	movk	{'x4', '#6160'};
20	4197420	bl	{'4198544'};
21	4197424	bl	{'4195072'};
22	4198544	stp	{'x29', 'x30', '[sp, #-384]!'};
23	4198548	adrp	{'x7', '4587520'};
24	4198552	mov	{'x29', 'sp'};
25	4198556	ldr	{'x7', '[x7, #2312]'};
26	4198560	stp	{'x19', 'x20', '[sp, #16]'};
27	4198564	mov	{'x19', 'x3'};
28	4198568	stp	{'x21', 'x22', '[sp, #32]'};
29	4198572	mov	{'x20', 'x4'};
30	4198576	mov	{'x21', 'x5'};
31	4198580	stp	{'x23', 'x24', '[sp, #48]!'};

Figure 18. GCC parser output file

4.5 XML comparator

The XML comparator performs comparison between two xml files to observe the difference between two models in different aspects, such as parameters, variables, etc. Select Parser in tool home page and select XML comparator option.

Select the xml files that you want to compare in XML file 1 and XML File 2 oprions. Click generate.

```
XML_Comparator Results
```

```
Valid Relations === [relation6, relation7, relation, relation9, relation11, relation13, relation12, relation10]
```

```
Valid Ports ===== [Traffic.output, ExpressionList.input, Queues.input, Fork.output, Queues.pop_input, Flow.output]
```

```
Invalid Relations === [relation61, relation16, relation59, relation21, relation53, relation54, relation69, relation18]
```

```
Invalid Ports === [BusArbiter.input, BusInterface.child_out, BusInterface.input1, Integrated_Cache11.to_neighbour]
```

```
Actor and parameter comparison =====
```

```
A new parameter has been added to lx_proc_multi_level_cache.xml#PowerTable . Parameter Name = State_Plot_Enable
```

```
A new parameter has been added to lx_proc_multi_level_cache.xml#AMBA_AXI2 . Parameter Name = Enable_Plots_and_Flow
```

```
New Hierarchical block has been added. Name = AMBA_AXI , Location = lx_proc_multi_level_cache.xml#AMBA_AXI2
```

```
Parameter value has been modified in lx_proc_multi_level_cache.xml#Integrated_Cache2 . Parameter Name = Cache_Size
```

```
Parameter value has been modified in lx_proc_multi_level_cache.xml#Integrated_Cache . Parameter Name = Cache_Size
```

```
Parameter value has been modified in lx_proc_multi_level_cache.xml#Integrated_Cache . Parameter Name = Cache_Size
```

```
Parameter value has been modified in lx_proc_multi_level_cache.xml#Integrated_Cache . Parameter Name = Cache_Size
```

```
Parameter value has been modified in lx_proc_multi_level_cache.xml#Integrated_Cache . Parameter Name = Miss_Count
```

```
Hierarchical block has been removed. Name = BusArbiter , Location = 4x_proc_common_L2.xml#BusArbiter
```

```
Hierarchical block has been removed. Name = BusInterface , Location = 4x_proc_common_L2.xml#BusInterface
```

Figure 20. XML comparator result

5 Blocks

A block is a basic modeling component available in VisualSim. Every icon in VisualSim is considered a block. A block contains an icon, ports, parameters, and logic inside. The user can modify the parameters. Associated with each are a standard documentation and the ability for the user to create custom documentation. Some of the blocks have the ability to add additional ports. The user can change the graphic image of the icon using the Custom_Icon option. If an attribute such as a rectangle is covering a Block, right-click on the attribute and "Send to Back".

5.1 Block Execution Semantics

VisualSim supports a common set of block execution semantics. Here we focus on the Digital domain. A Digital library block can optionally execute the following internal methods:

initialize ()	Initialize Block parameters, attributes, memories, and so on
prefire()	Pre-Fire Block once per simulation iteration
fire()	Fire Block, input Tokens processed, outputs generated
postfire ()	Post-Fire Block, iterations, scheduled events
wrapup ()	Wrap-Up Block, free memory, collect ending stats

It is not necessary to become familiar with the details of internal block execution, unless one is planning to create their own custom blocks. The typical block that are shipped with VisualSim executes the **initialize()** and **fire()** methods. In the **initialize ()** method, block parameters, and block menu entries are read, as the block is initialized into the modeling space. In the **fire()** method, input port Tokens are processed, and output port Tokens are generated based on the Block functionality. Graphically, block execution looks similar to the following image:

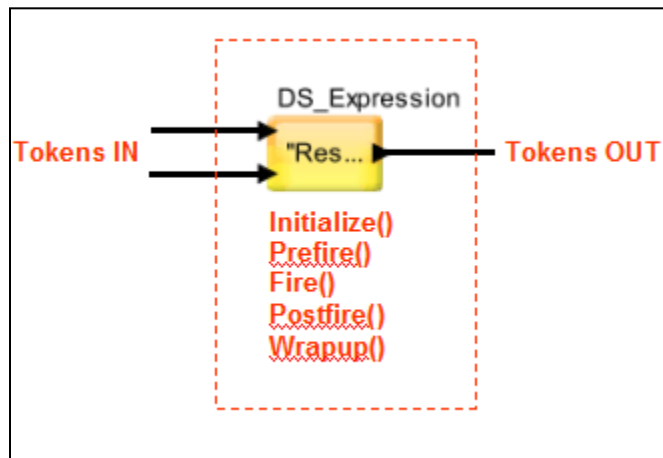


Figure 21.

Block Execution Semantics

5.2 Block Layout

A typical Digital Block contains the following attributes:

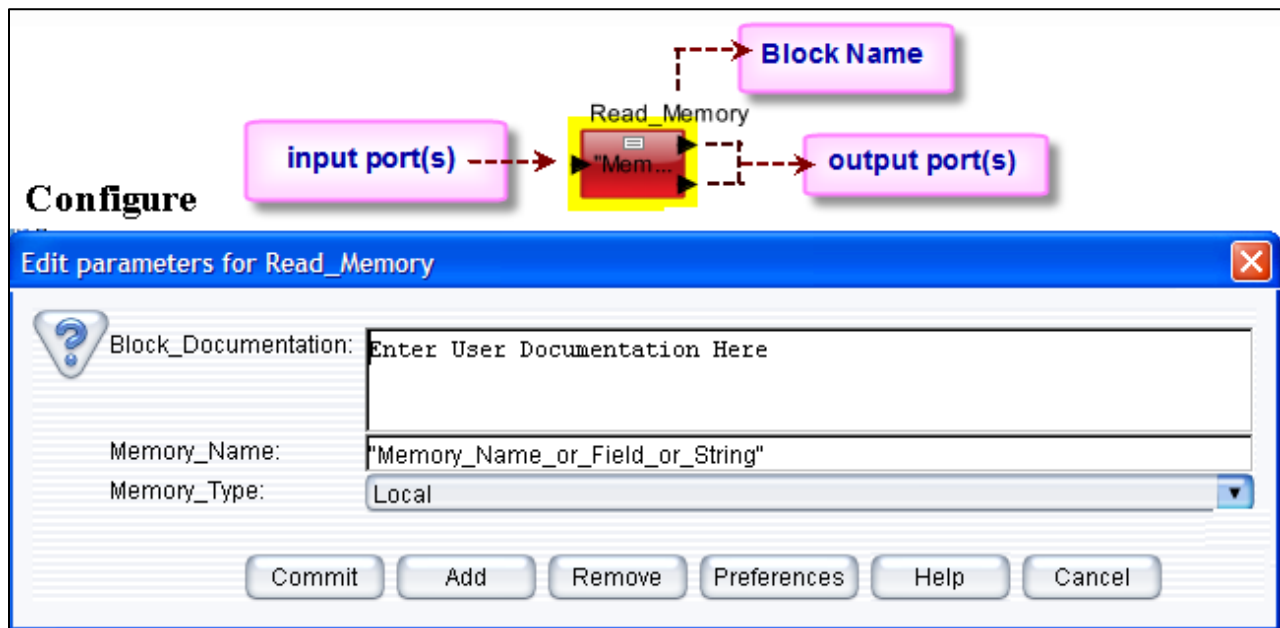


Figure 22.

Block Attributes

The user notices that a “Name” sometimes appears on top of the block symbol. This text typically describes the beginning of the first statement, or other parameter of the block, so the user can quickly identify the block functionality without having to open it.

5.3 Block Context Menu

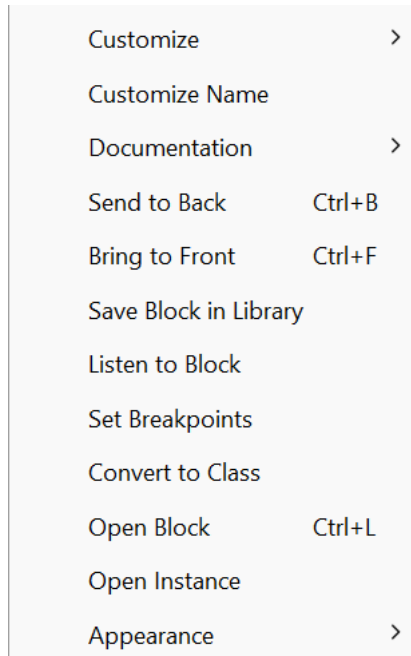
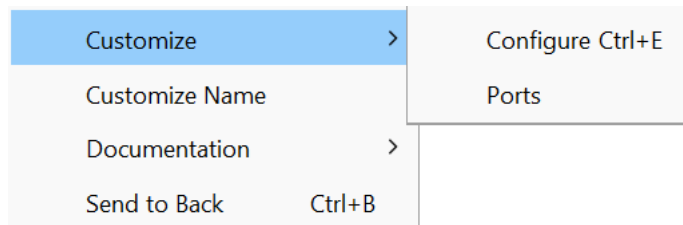


Figure 23. Block Context Menu

The **Block Context Menu** options are explained in the following sections:

5.3.1 Customize



The **Customize Parameters** is available for the parameters, blocks, modules, and ports. This **Configure** option is used to modify or add parameters to the block. After the parameters are modified, click **Commit** to update the model or click **Cancel** to retain the current settings.

The **Ports** is available for the blocks, and modules. Click in the cell to be edited. Some properties may not be editable because they are part of block definition. These cells have a red background and ignore any attempts at editing as shown below.

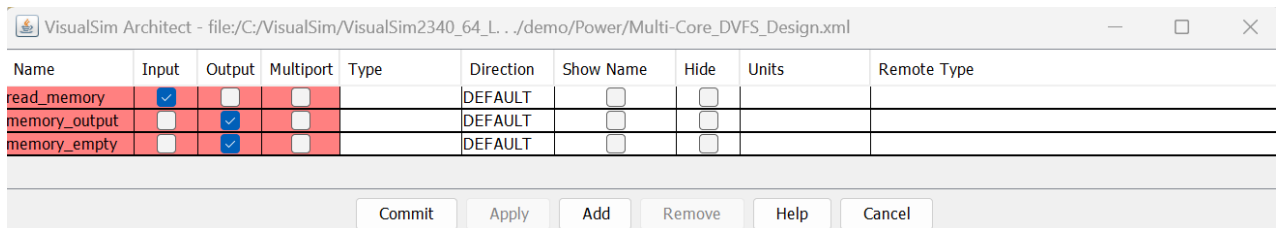


Figure 24. Ports that cannot be edited

Name – Names must not contain a period or a space, must not be blank, and should be unique.

Input, Output, Multipoint – Checkboxes that determines if a port is input, output, or both.

Type – The type can be given by any expression that can be evaluated (this expression is called a “prototype”). For example, you can specify the type to be double by giving the string “0.0” (or any other number with a decimal point). However, this does not look very good as a designator for a type. The VisualSim expression language defines the constant “double” to equal “0.0”. Thus, the preferred way to specify that a port has type double is to give its type as “double”.

The primitive types include:

- Data Structure
- boolean
- complex
- double
- fixedpoint
- float
- general
- int
- long
- matrix
- object
- scalar
- string
- unknown
- unsignedByte
- xmltoken
- {int}
- [double]
- {x = double, y = double}

The constant “unknown” has a rather special behavior, in that it sets the type of the port to be unknown, allowing type resolution to infer it. The constant “matrix” designates a matrix without specifying an element type, in contrast to, for example, “[double]”, which specifies a double matrix (see below). Similarly, the constant “scalar” designates a scalar of any type (double, int, long, etc.). The constant “general” designates any type.

As the type is given by a “prototype”, any data type that can be given in an expression can be specified as a type. For structured types, follow the same syntax as in expressions. For example:

```
{double} - double array
[int]     - int matrix
{field1 = string, field2 = int} - For Data Structure
```

Direction – The Direction determines which side of the icon the port is located on. Clicking in this cell brings up a menu that provides the means to select a specific side, that is. NORTH, EAST, SOUTH, and WEST. DEFAULT can also be selected. Below are the defaults:

- Input ports are usually on the West side.
- Output ports are usually on the East side.
- Ports that are both an input and an output are located on the South side.

Show Name – Enabling this option causes the name of the port to be displayed. Disabling it removes the name from the display.

Hide – Disabling this option causes the port to be displayed. Enabling removes the port from the display.

Units – The units specification for the port. This is currently not being used.

Remote Types – This column is used to define the data types on the remote system or application. This is used to define the data types for the Verilog, Application Interface, and SystemC only. This is a string field and must be enclosed by “”. The content in this column text depends on the type of block and the interface.

5.3.2 Customize Name

This feature is available for the parameters, blocks, modules, and ports. This function is called to modify the name of this entity. After the name has been modified, click **Commit** to update the model or click **Cancel** to retain the current settings.

5.3.3 Get Documentation

This feature is available for the blocks and modules. This function opens a new window that provides access to the complete documentation associated with that block. If there is no documentation provided for the entity, then a dialog window appears indicating that there is no documentation available or the default documentation for the base component is shown, such as Hierarchical block. The **Customize Documentation** enables the user to override the default documentation and add their own information. This is associated with that block in the model. Note that once the user selects the Customize Documentation item, the documentation shows the new format. To undo this option, the user must go to **Remove Custom Documentation**.

5.3.4 Send to Back

This moves the selected block behind whatever is in front. For example, it can be a Annotation or a Rectangle.

5.3.5 Bring to Front

This moves the selected block to the front.

5.3.6 Save Block in Library

The selected block can be saved in the local library. This is a useful way to store the frequently used blocks in a local list.

5.3.7 Listen to Block

This feature is available for the blocks and modules. A new text window opens. This feature enables the user to view the progression of the execution through the selected block. This

feature is not available for Hierarchical blocks. It is supported for regular blocks and instantiated hierarchical classes. If you need to see the details of operation within a Hierarchical block, you need to Open Block and select the block to view the execution details. If you need to view the detailed operation of a Instantiated class, you need to select Open Instance and select the block to view the execution details.

5.3.8 Set Breakpoint

This feature allows the user to set the breakpoints at any block. This feature is available only for Digital and Untimed Digital Domains. This can be used to move the simulation to a particular point in the simulation and then debugging operation such as a listen to block.

5.3.9 Convert to Class

This feature converts this block into a class. A class is a master version of the block. This class can be instantiated multiple times in the model. Changes made to this class block (identified by a blue halo), are replicated to all linked instances. The information in the XML for the instances are all linked to this class block.

5.3.10 Class Action - Create Instance (Ctrl+N)

This creates an instance of the class block.

5.3.11 Class Action - Create Subclass (Ctrl+U)

This creates a subclass of the class block. Instances can also be created of the sub-classes. Changes made to this block propagate to the instances. Changes made to the class block are propagated to this block.

5.3.12 Convert to Instance

This converts the class or subclass back to an instance.

5.3.13 Open Block

This feature is available for hierarchical blocks and those shipped with source code. All other blocks show up with an error message.

5.3.14 Open Instance

This opens the details of the instance and not the original class. This is very important for models constructed with XML classes. **The instance code must not be modified.** Modify the code in the block definition. If the instance code is modified, the reference to the original class is disconnected. Use the Instance to do Listen to Block/Port for debugging in the model.

5.3.15 Appearance - Edit Custom Icon (Icon Editor)

This menu item is used to replace the current icon for the selected block with a custom drawn icon. This item opens the Icon editor. The user can create a custom icon in the Editor and save it in the XML file along with the current model.

5.3.16 Appearance - Remove Custom Icon

This restores the original icon and removes the custom icon.

5.3.17 Appearance – Flip Ports Horizontally

The ports will be reordered by flipping the ports in horizontal axis.

5.3.18 Appearance – Flip Ports Vertically

The ports will be reordered by flipping the ports in Vertical axis.

5.3.19 Appearance – Rotate Ports Clockwise

The ports will be reordered by Rotating the ports of the block in clockwise direction.

5.3.20 Appearance – Rotate Ports Counterclockwise

The ports will be reordered by Rotating the ports of the block in counter clockwise direction.

5.4 Setting up a Block

5.4.1 Blocks Overview

1. All Data Structures enter on the top-left (input port) and exit on the top-right (output port). Blocks have parameters that can be populated by the user. There is a format for filling the values for every block, most of the block have default values or comments about the parameter. The block documentation has example values for each parameter
2. The traffic is the stimulus that goes through from block to block. The input into a block causes the block to fire, meaning that the block takes the content of the input; does modification to the content or does some action based on the content; and then places it on the output port.
3. Block Parameter entry rules:
 - a. Any parameter field that is shaded blue is a string expression. All values placed within this string do not require "" around them.
 - b. If the entry is an expression and the background of the parameter space is not blue, the complete expression must be enclosed in a "".
 - c. If the entry is a number or a parameter, then no quotes are required.
 - d. Memories and field names must be always within "", if the parameter field is not shaded blue.
 - e. ExpressionList, Script, and Smart Controller blocks support multiple input ports. Hence the field names on any input port must be identified with the prefix of "port_name.". For example, input.INDEX is the INDEX field in the Data Structure arriving on the port called "input".
 - f. The Script block also supports a generic port, called port_token, that represents any input port, and the current executing thread.

- g. In the Basic Processing and Resource blocks, there can be only a single input port. Here, the field name does not contain the “port_name” prefix.

5.4.2 Updating Parameters and adding additional parameters

To set up a block in VisualSim, double click the library and select the required block. Drag and drop the block into the BDE and double click the block to set the parameters. You can also right click and select **Customize > Configure**. The Edit Parameter window is displayed. Set values for the predefined parameters. You can create a new parameter for the block. To add a new parameter, click **Add**, enter the parameter name, default value and class. Click **OK**.

5.4.3 Updating Ports and adding additional ports

You can add ports to block. By default, every input port is located on the left hand side of the block and the output port is on the right hand side of the block. To add a port, right click the block and select **Customize > Ports**. Click **Add** and a new entry is created. Enter the name of the port and select the type of the port. The type of the port is described in detail as follows:

If **input** is selected, the new port is created on the left side (default-West) of block.

If **output** is selected, the new port is created on the right side (default-East) of the block.

If **multiport** is selected, the new port is created on the left side (default-West) of the block.

If both **input** and **output** are selected, the new port is created at the bottom (default-South) of the block.

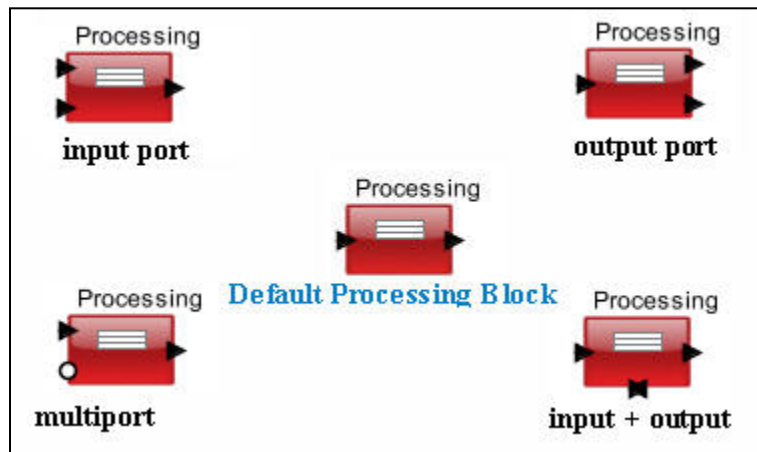


Figure 25. Set Ports to Block

User can also change the direction of the port to North, East, South or West. There are options to display the port name or hide it. Units or the Remote type can also be set.

Port data type- general (Default). In most cases, the port type does not have to be changed. It can be maintained in the default state. There are certain instances when the connected input port has a fixed data type. In that case, the output port connected to it must be modified.

Show Name – To display the port name

Hide - To hide the port name.

Remote Types- This column is used to define the data types on the remote system or application. This is used to define the data types for Verilog, Application Interface, and SystemC blocks only. This is a string field and must be fully contained within "". The content in this column text depends on the type of block and the interface.

5.4.4 Setting Port Types

Some block ports may have pre-defined data types. For example, the plotter input and the pop input of the Queues. In these cases, the ports connected to these fixed data types must be modified. If the ports are not correctly specified, an exception is reported. To modify the port type, right-click on the block to "**Customize > Ports**". To see the current type prior to modifying, context display by passing the mouse pointer over the specific ports. If the connected ports

have type General, then no change is required. If either the input or output type has a specific type set, then the other one must also match this value. If a port has type Unknown, and is connected to a type of General, then set the Unknown to General.

5.4.5 Linking Parameters

Block Parameter is the default parameter available in a block. User can also add or remove (added parameter) block parameter. **Model Parameter** is available in the VisualSim library: **Model Setup > Parameter=**. How to set a model parameter to the model is explained below.

To set a new parameter to model, drag and drop the model parameter from **Model Setup > Parameter=**. Right click the model parameter and select **Customize Name**. Enter the name of the parameter and click **Commit** to display the name in BDE. Now, double click the **Parameter:** to set the value for the parameter. Copy the model parameter name and paste it in the block parameter. Now you have successfully linked a model parameter with a block parameter.

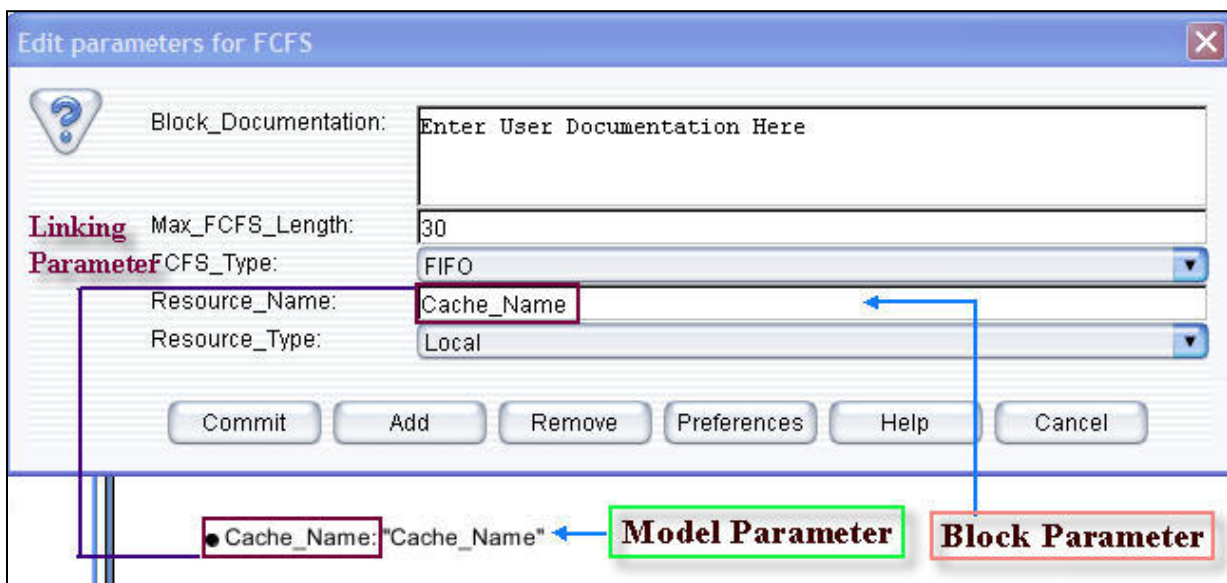


Figure 26. Linking Parameter

5.4.6 Block Documentation

Every block in VisualSim contains block documentation. To view the block documentation right click the block and select **Documentation > Get Documentation**.

User can create documentation for a custom block or hierarchy based block. To create a new document, right click the block and select Open Block. A new BDE is opened. Right click in the white space and select **Documentation > Customize Documentation**. Enter the new documentation details and click **Commit**. Save the model.

5.4.7 Icon Display value

By default, the block icon has the name of the block instance. This is unique in the model window. It is possible to display something different by adding a line to the XML as follows:

```
<display name="This is a test of the display system."/>
```

Where the item in "" is the value to be displayed on top of the icon. In this case, the Block Instance name is replaced with this value.

5.4.8 Tooltip value

It is possible to customize the tooltip, which is the string that appears when you move the mouse over a block. To do this, add a parameter called `_explanation`, with a default value without any "" and class as `VisualSim.kernel.util.StringAttribute`. The default value is displayed as the tooltip.

5.4.9 Creating Generic Port Icon

Port icons are dynamically created in VisualSim during the block instantiation on the Block Diagram Editor. The icon and color for Input, Output, Input/Output, and none port types are fixed and cannot be modified by the user. For a number of reasons, the user might want to keep the look of the port different from the standard view. A generic icon of rectangular shape and green fill is provided. The port icon looks the same for input, output, and input/output. The users can modify their port icon to this generic type by doing the following:

1. Drag a block into the BDE.
2. Right-click on the port and select "Configure".
3. Now click Add.
4. Enter the following information.
 1. Name: `_icon` ; Value: `true`; Class: `VisualSim.kernel.util.StringAttribute`
5. Click OK.
6. Click OK.
7. Now, you see that the port icon has changed to a green rectangle.

You can do the same thing by updating the xml file:

```
<port name="input" class="VisualSim.actor.TypedIOPort">  
  <property name="input"/>  
  <property name="_icon" class="VisualSim.kernel.util.StringAttribute" value="true">  
    </property>  
  </port>
```

You need to add the property name `_icon` with the associated class and value to each port that needs to be converted to this generic icon.

6 Hierarchical Block Semantics

6.1 Using Hierarchical blocks

A Hierarchical block is a container of a number of basic and other Hierarchical blocks. A hierarchical block is used when the Editor Window runs out of space, or combines blocks that perform a specific function. Each level of a Hierarchy must have the same simulator. Levels below and above this Hierarchy can have a different simulator, as far as they conform to the simulator rules.

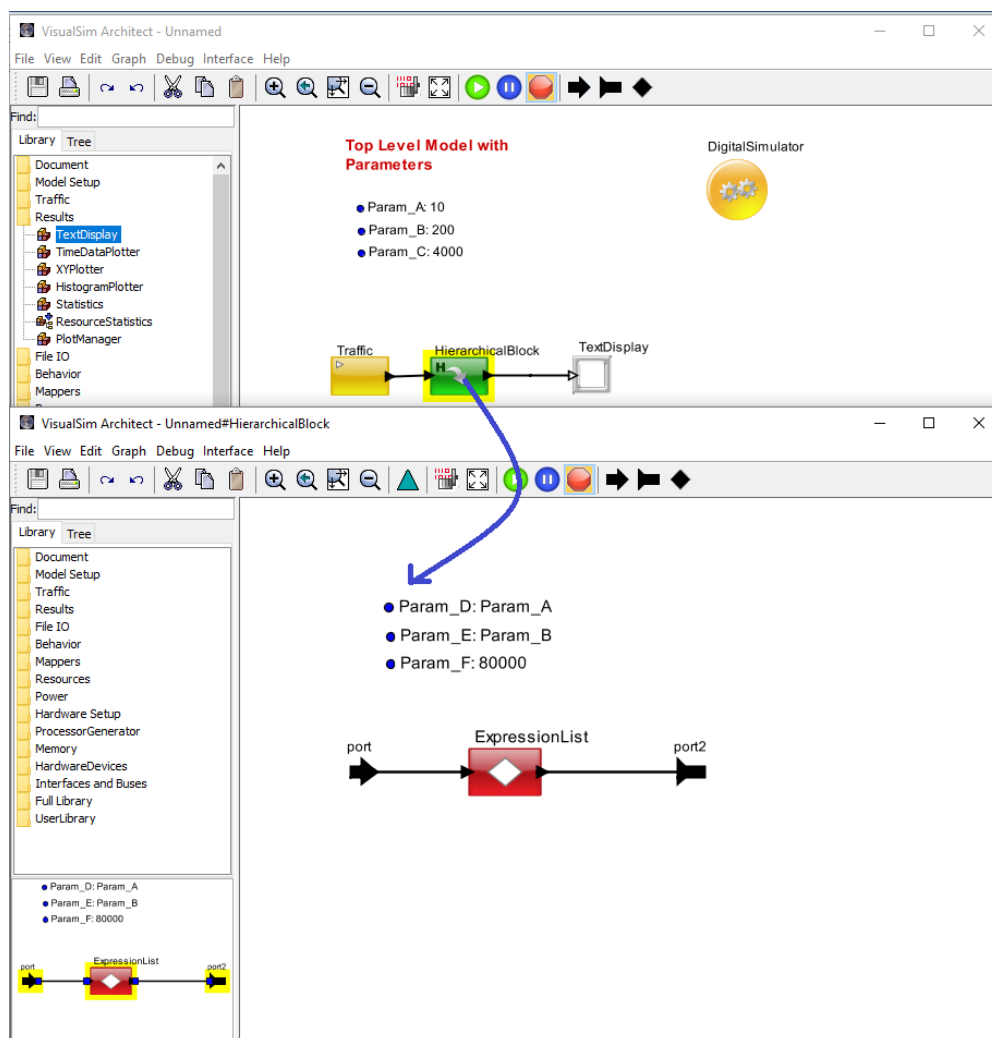


Figure 27.

Hierarchical Block -> Open Block

To create a hierarchical block, drag "Hierarchical Block" from the Library Full Library > Model> Hierarchical Blocks into an existing BDE window. To view the inside of the Hierarchical block, right-click the icon and select "Open Block". Select the input and output Toolbar Icons to add input and output ports. These are displayed as input/output ports at the parent level model. Example of a simple Hierarchical Block is shown in Figure 13.

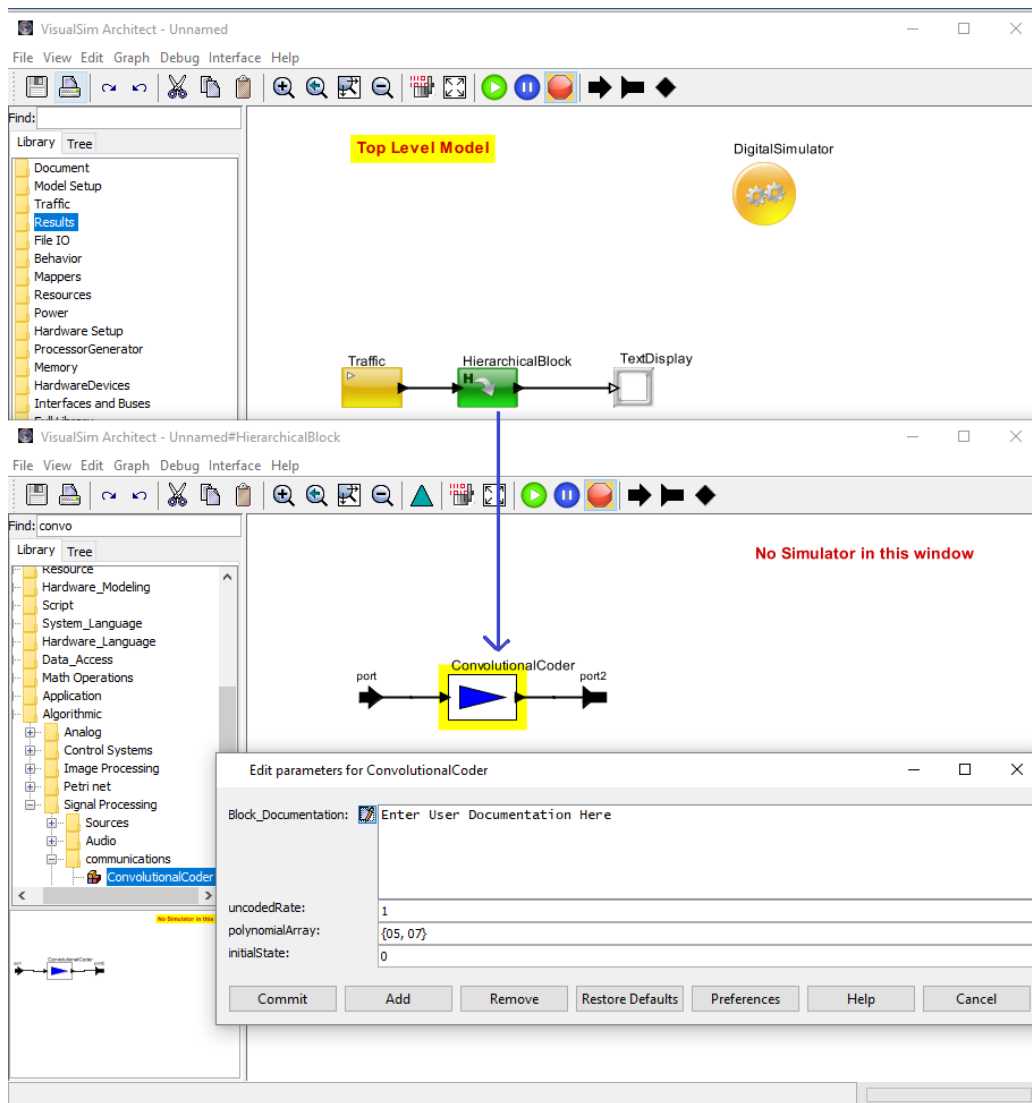


Figure 28.

Adding Blocks to a Hierarchical block

Port Type (input (input), output (output)) are set by right-click in white space or by selecting the icons from Menu/Toolbar. See notes on "Configure Ports" for more details.

Simulator Type None is required if the Hierarchical Block is using the same time semantics as the parent window. For example, this block is a Digital and parent is Digital. Mixed domain modeling requires a simulator to be defined with the hierarchical block and is discussed as separate topics. For example, a DSP Algorithm (Untimed Digital Simulator) can exist within a Digital.

In terms of model execution, Hierarchical blocks initialize, the first time they receive an input Token. In terms of animation, one can animate a single hierarchical block, using the same procedure as a flat model topology.

6.2 Model Execution Semantics

The Digital domain execution is a sequence of synchronous block execution between asynchronous time. This means that blocks with no delay are all executed at the same time (synchronous time) in the flow order. When a flow encounters a relation or a branch, it returns to the last branch and picks another flow to execute. It continues to do this until no more blocks can be executed at that time. At that point, it looks at the Master calendar and determines the next time point where there are asynchronous events. VisualSim schedules all delays and events on a master calendar queue in the Digital domain. All events executed on the master calendar queue are deterministic, meaning that the simulation executes the events, whether synchronous or asynchronous, in the same order each time the simulation is run. This is important for model development, repeatability during model validation, and obtain the same timing results for every run.

Synchronous Events: Synchronous events occur at the same simulation time, meaning more than one event is scheduled on the calendar queue for the same simulation time.

Asynchronous Events: Asynchronous events occur at a future simulation time, relative to the current simulation time and by implication there are no outstanding synchronous events.

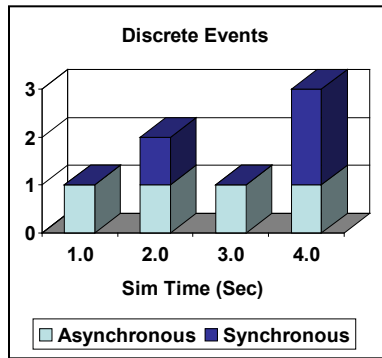


Figure 29.

Model Execution Semantics

6.3 Model Execution Flowchart

This figure depicts the overall model execution flowchart which combines the Model Execution semantics and Block Execution semantics, shown in the previous two pages.

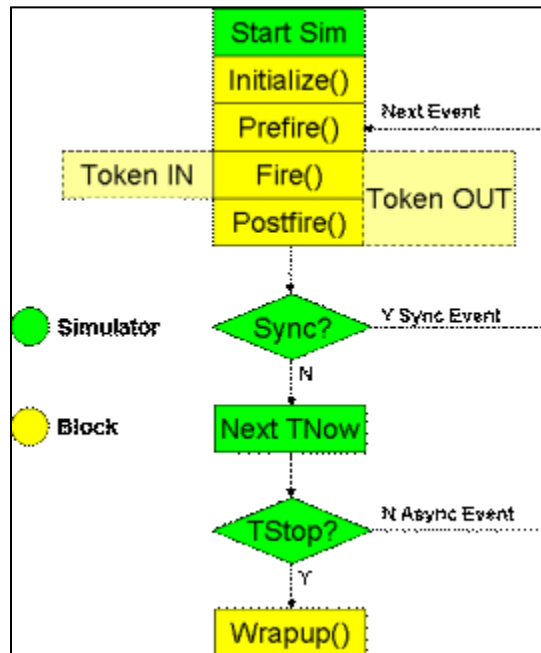


Figure 30.

Model Execution Flowchart

6.4 Create Sub-Model

This feature converts this block into a Sub-model or class. A class is a master version of the block. This class can be instantiated multiple times in the model. Changes made to this class block (identified by a blue halo), are replicated to all linked instances. The information in the XML for the instances are all linked to this class block. More information on this is provided by the Sub-Model documentation page.

7 Dynamic Instantiation

7.1 Introduction

VisualSim includes a number of *Advanced Function Blocks*, that operate on the structure of the model rather than on data. These higher-order components help significantly in building large designs where the model structure scales. The most important of them is the Dynamic Instantiation. All other higher-order or Hierarchical blocks are considered experimental and are not guaranteed to work all the time. Using this block, the model can scale to any number of instances of an identical component by varying a parameter value. Each instance can have parameter values that are unique and can be accessed uniquely by an instance parameter.

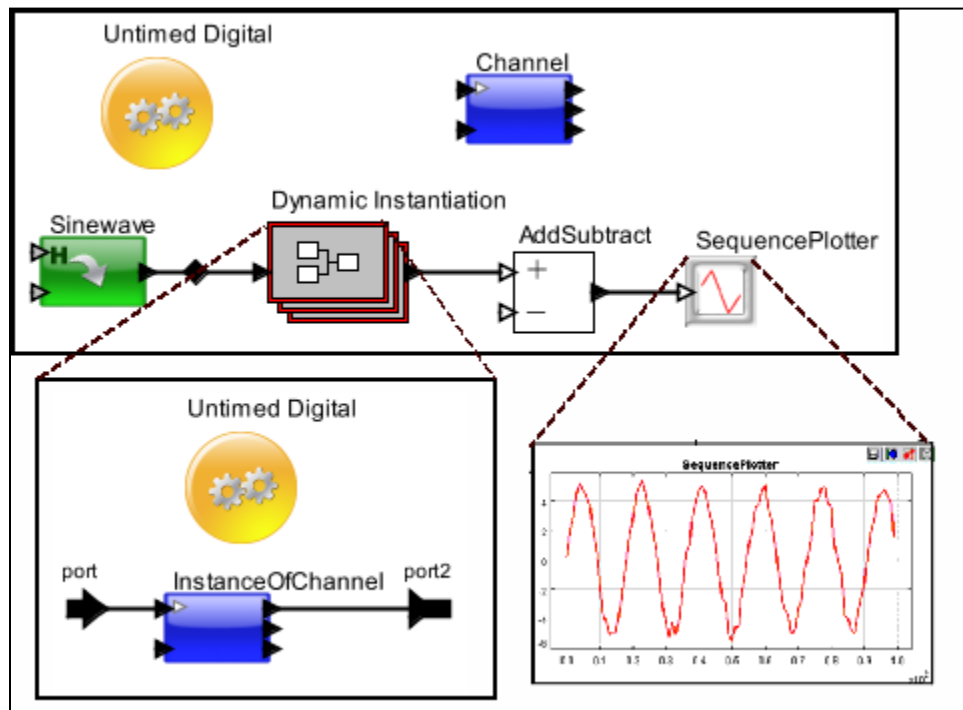


Figure 31.

A model using the Dynamic Instantiation block

7.2 Example

Consider a model which has the five instances of the Channel class wired in parallel. This model has the feature that the number of instances is hardwired into the diagram. It is awkward,

therefore, to change this number, and particularly awkward to create a larger number of instances. This problem is solved by the Dynamic Instantiation block. A model equivalent to that is created using the Dynamic Instantiation block as shown in Figure 16.

The above figure shows a model using the Dynamic Instantiation block which permits the number of instances of the channel to change by simply changing one parameter.

The Dynamic Instantiation is a hierarchical block into which we have inserted a single instance of the Channel.

The Dynamic Instantiation block has two parameters, *nInstances* and *instance*, as shown in the below figure.

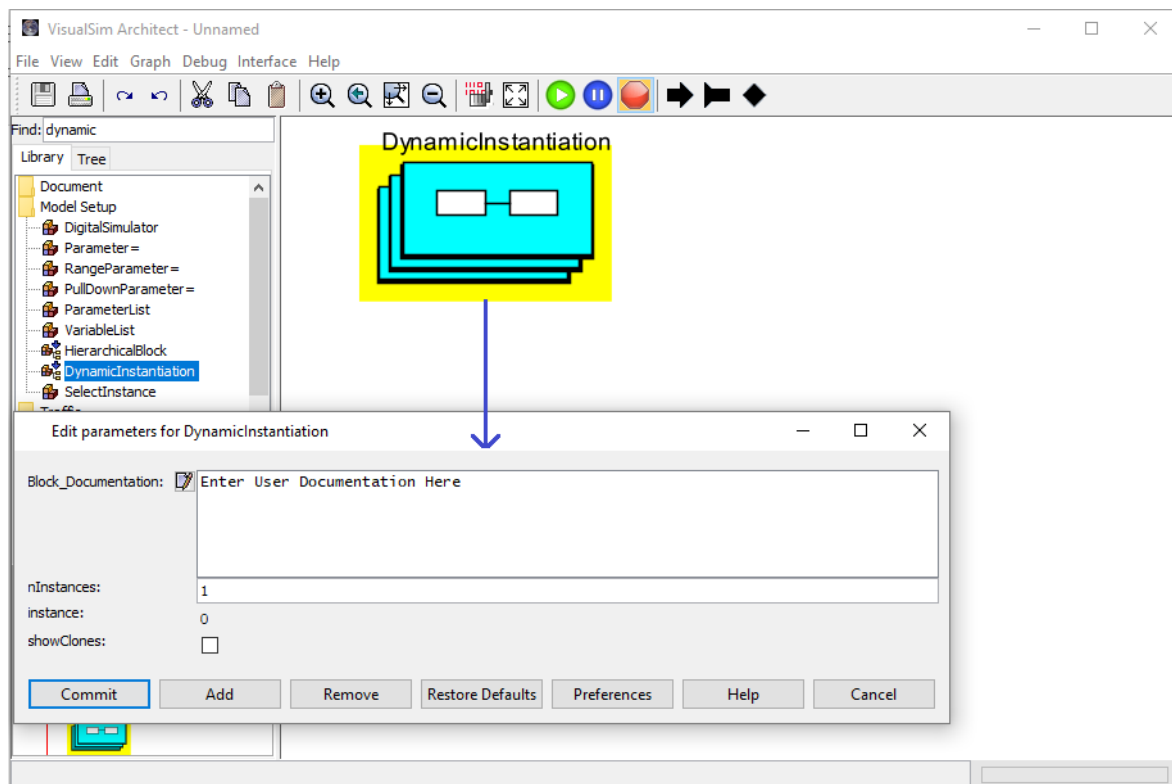


Figure 32. Configure Parameters for Dynamic Instantiation

The first parameter of the Dynamic Instantiation block specifies the number of instances while the second parameter is available to the model builder to identify the individual instance.

The first of these specifies the number of instances to create. At run time, this block replicates itself *nInstances* parameter value, connecting the inputs and outputs to the same sources and destinations as the first (prototype) instance. In Figure 16, notice that the input of the Dynamic Instantiation is connected to a relation (the black diamond), and the output is connected directly to a multiport input of the AddSubtract block. As a consequence, the multiple instances are wired in a manner similar to the above figure, where the same input value is broadcast to all instances, but distinct output values are supplied to the AddSubtract block.

The model is better in Figure 16 because now we can change the number of instances by changing one parameter value. The instances can also be customized on a per-instance basis by expressing their parameter values in terms of the *instance* parameter of the Dynamic Instantiation block. Try, for example, making the *noisePower* parameter of the InstanceOf Channel actor depend on *instance*. For example, set it to "instance * 0.1" and then set *nInstances* to 1. You see a clean sine wave when you run the model.

7.2.1 How To

Dynamic Instance (DI) block is located in the **Full Library > Model Setup**. Use this block in the same way that a Hierarchical Block is used. The SelectInstance is located in the **Full Library > Model Setup** library. The SelectInstance is required on the input side and the output side of the DI block to indicate the specific instance to be accessed.

7.2.2 Requirements

To access each instance of the Dynamic Instantiation (DI) block, a SelectInstance block must be placed on the input and output of the DI block. On the input, this block picks the instance from a value in the incoming Field or Variable, please select Conversion Type as DS(0)_to_DS(n). On the output, the block combines multiple instances to an output, please select Conversion Type as DS(n)_to_DS(0).

7.2.3 Example of the Dynamic Instantiation

To view an example of a Dynamic Instantiation model in VisualSim, view the Dynamic Instantiation Usage Page, locate the following file in VisualSim directory VS_AR\doc \Training_Material\How_to_tasks\DI\DI.htm. Examples models are available in the same directory. One of these examples is described here.

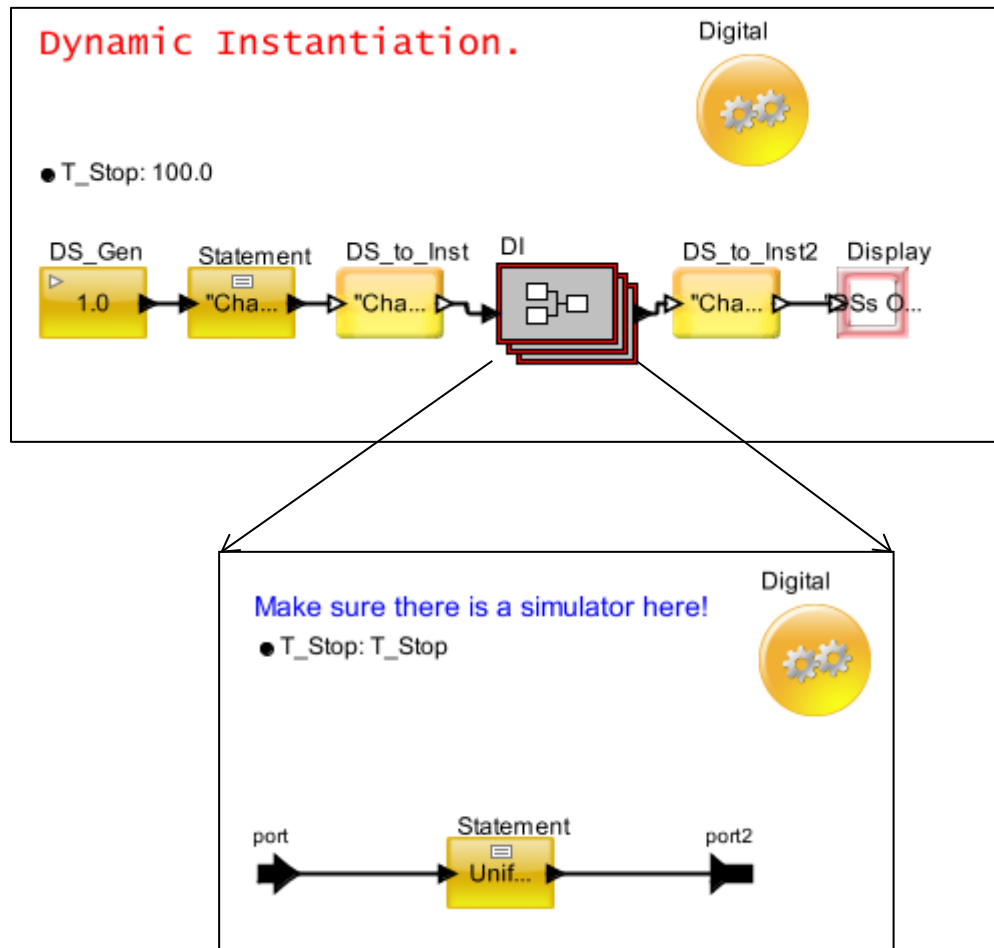


Figure 33. Dynamic Instantiation Example in VisualSim

In this simple model, there are 4 instances of the DI block. A traffic generator creates DS. The field "Channel" of the Stop_n_Wait DS provides a random number between 0 to 3. In the SelectInstance block, the value of the Channel field specifies the instance that this DS is sent to. The output of the DI also has a SelectInstance block. This combines the DS coming from all the

instances and sends them out on a single instance link, as is visible from the Display output. Also notice that any parameters can be added just as in the Hierarchical Block.

8 Block, Sub-Models (Classes) and Inheritance

8.1 Introduction

Blocks are the basic elements in VisualSim, used to construct a model. Blocks can be defined using Java, C/C++ enclosed in a Java Wrapper, Hierarchical block within a model, and a hierarchical block saved as a separate file called sub-model or class. Class and Sub-Model are the same and are used interchangeably.

One of the major capabilities in VisualSim is the ability to define block-oriented classes with instances and subclasses along with inheritance. The key idea is that you can specify that a component definition is a class, in which case all instances and subclasses inherit its structure. This improves modularity in designs. We illustrate this capability with an example.

Classes are especially useful when you need to instantiate a block multiples times in a model or used repeatedly in many models. This can be a component in your designs or a reporting utility.

Classes are the base entities. When the class is instantiated, the user can only modify the values of the parameters and not the topology or the existing list of parameters. The user can add additional details within the Instance. These details are maintained only for that particular model and does not affect the base Class.

If these new changes become the basis for another reusable block, then you can define this new block as a sub-class. The sub-class refers to the class for the basic details. Only the incremental details are stored in the sub-class.

A class or sub-class can be easily converted into a instance for use in a particular model. This is useful when a model needs to be shipped to another person that does not have access to the libraries.

The class also plays a significant role in the development of a user library. The following section discusses the construction of classes, use of the class in models, creation of libraries, and making the library in the Library Folder.

Before we get into the details, here is an example of the classes and the role they play in VisualSim modeling.

8.2 Example of Classes

Follow the steps to create the model:

1. Drag the Digital Simulator block from **Model Setup > Digital Simulator**.
2. Drag the Sinewave block from **Full Library > Source > Clock**.
3. Drag the SequencePlotter block from **Full Library > Result > Plotter**.
4. Create a Hierarchical block.
 - a) Drag the AddSubtract block from Math_Operations > Math and Trig.
 - b) Drag the Gaussian block from Math_Operations > Distributions.
 - c) Drag a parameter from Model Setup > Parameter=. Right click the parameter and select **Customize Name**. The Rename Parameter window is displayed. Now enter the name of the parameter as noisePower and enable Show Name.
 - d) Double click the parameter named noisePower and enter the value 0.01.
 - e) Now select all the blocks (Mentioned in points a to d). The selected blocks are highlighted in yellow color. Now click Graph > Create Hierarchy. The Hierarchical block is displayed. (To view the blocks, right click the Hierarchical block and select Open Block.)
5. Right click the Hierarchical block and select Customize Name. Rename the Hierarchical block as Channel.



Hierarchical block is referred as Channel block in following sections.

6. Configure the Ports for the Hierarchical as described below:

- a) Right click the Channel block and select Customize > Ports. The Configure Ports window is displayed as shown below:

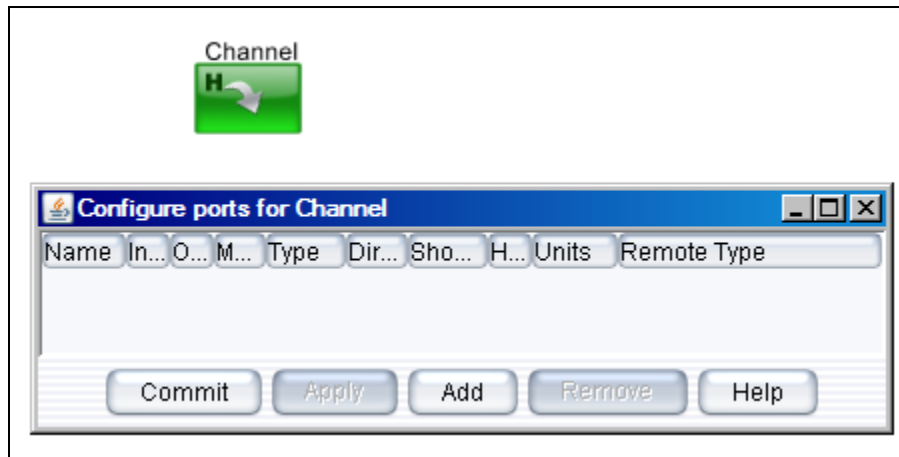


Figure 34.

Configure Ports

- b) Click **Add** to add a new port.
 - c) Enter the name of the port in **Name**. For example, IN, OUT.
 - d) Enable the Input option in case of input port and Output option in case of Output port. Select the type if necessary.
 - e) Click **Commit** to submit.
7. Now connect the output of the Sinewave block to the input of Channel block.
 8. Connect output of the Channel block to the input of the SequencePlotter.
 9. The Channel Hierarchical block icon can be changed by doing a right click on channel->Appearance->Edit Cuostom Icon

Refer the following image for more details:

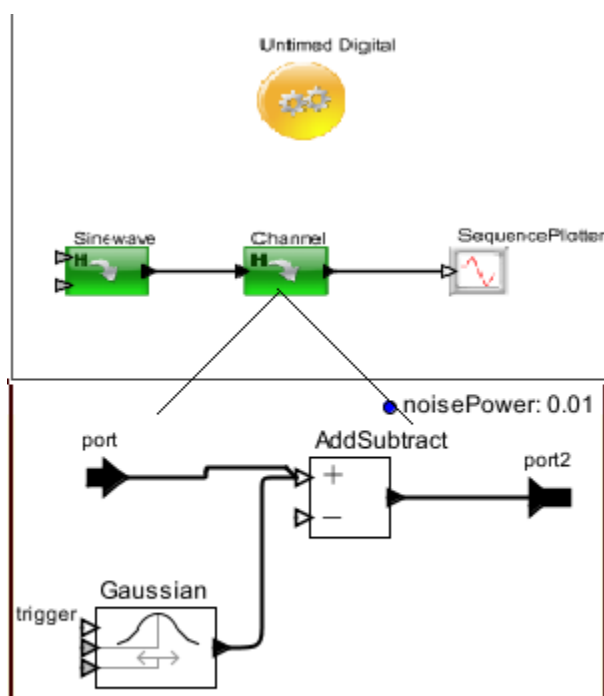


Figure 35.

Hierarchical Model to create a Class

Consider the model in Figure 35. Suppose that we wish to create multiple instances of the channel, as shown in figure 36.

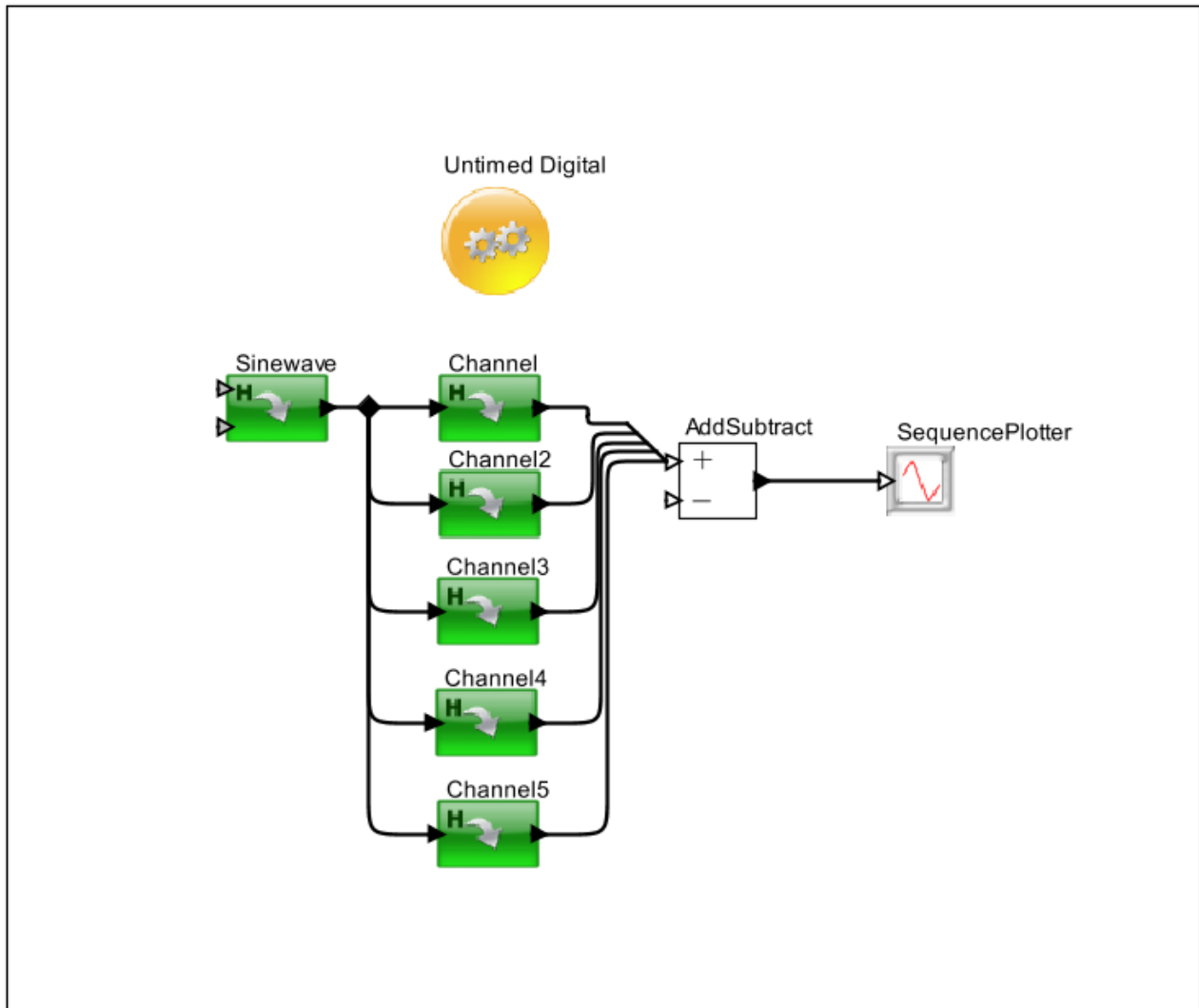


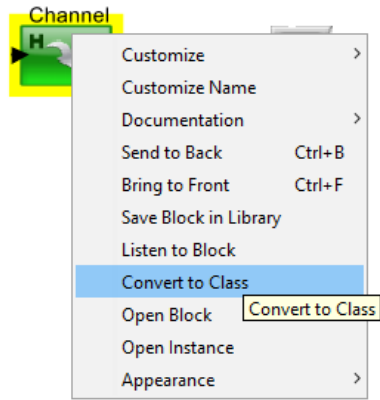
Figure 36. A poor design of a diversity communication system

The above is a poor design of a diversity communication system that has multiple copies of the channel that are defined.

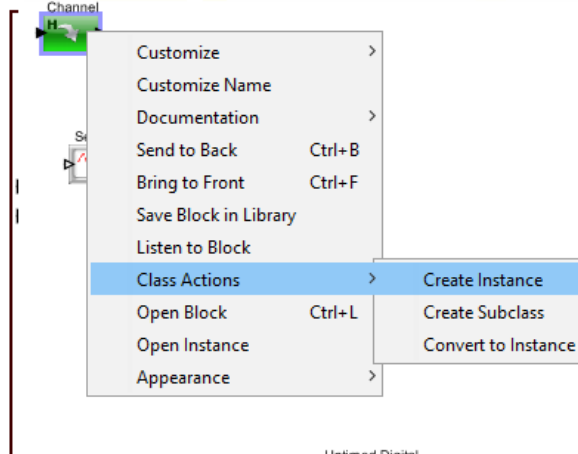
In the above figure, the Sinewave signal passes through five distinct channels (note the use of a relation to broadcast the same signal to each of the five channels). The outputs of the channels are added together and plotted. The result is a significantly cleaner sinewave than the one that results from one channel alone. However, this is a poor design, for two reasons.

First, the number of channels is hardwired into the diagram. We deal with that problem in the next section. Second, each of the channels is a copy of the hierarchical block in figure 36. This results in a far less maintainable or scalable model than we would like. Consider, for example, what it would take to change the design of the channel. Each of the five copies would have to be changed individually.

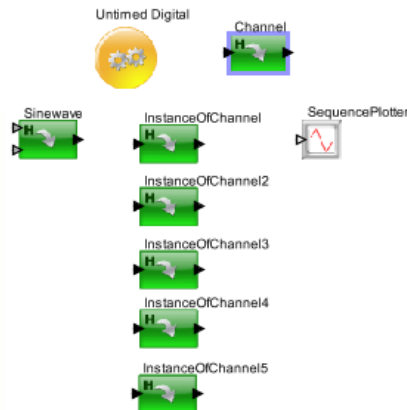
A better solution is to define a channel class or Sub Model. To do this, begin with the design in figure 35 and remove the connections to the channel, as shown in figure.



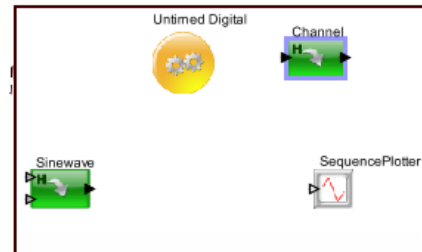
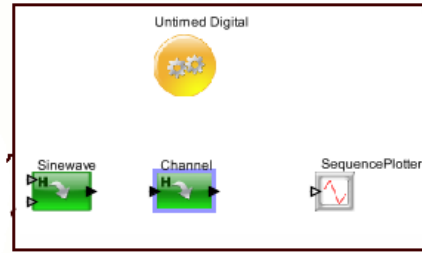
1. Model with connections removed



4. Create Instance



2. Model after channel is converted to class



3. Model after Class is moved to the top

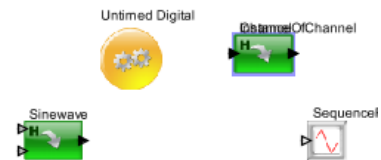


Figure 37.

Creating and using a Channel Class

Then right click and select "**Convert to Class**". (Note that if you fail to first remove the connections, you get an error message when you try to convert to class. A class is not permitted to have connections.) The block icon acquires a blue halo, which serves as a visual indication that it is a class, rather than an ordinary block (which is an instance). Classes play no role in the execution of the model, and merely serve as definitions of components that must then be instantiated. By convention, we put classes at the top of the model, near the simulator, as they function as declarations.

After you have a class, you can create an instance by right clicking and selecting "Class Actions > Create Instance" or press Control-N. Do these five times to create five instances of the class, as shown in figure 37. It is in fact, a much better design and to verify this, try making a change to the class. For example, by creating a custom icon for it, as shown in figure:

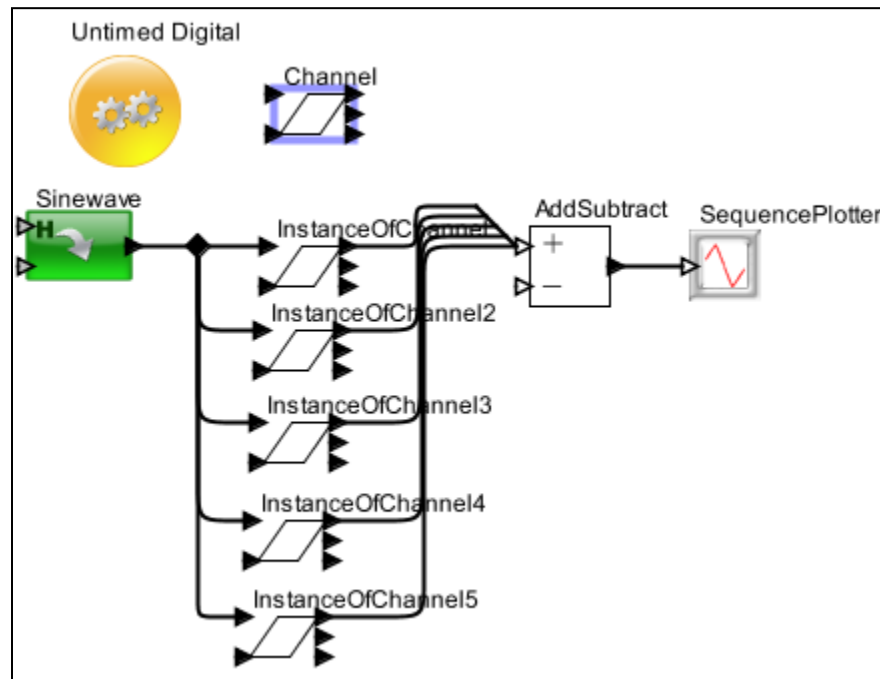


Figure 38.

The icon changed for the class

Note that changes to the base class propagate to the instances of the class. A more subtle advantage is that the XML file representation of the model is much smaller, since the design of the class is given only once rather than five times.

If you look inside any of the instances (or the class), you find the same channel model. In fact, you see the class definition. Any change you make inside this hierarchical model will be automatically propagated to all the instances. Try changing the value of the noisePower parameter, for example.

8.2.1 Overriding Parameter Values in Instances

By default, all instances of a Channel have the same icon and the same parameter values. However, each instance can be customized by overriding these values. In the below figure:

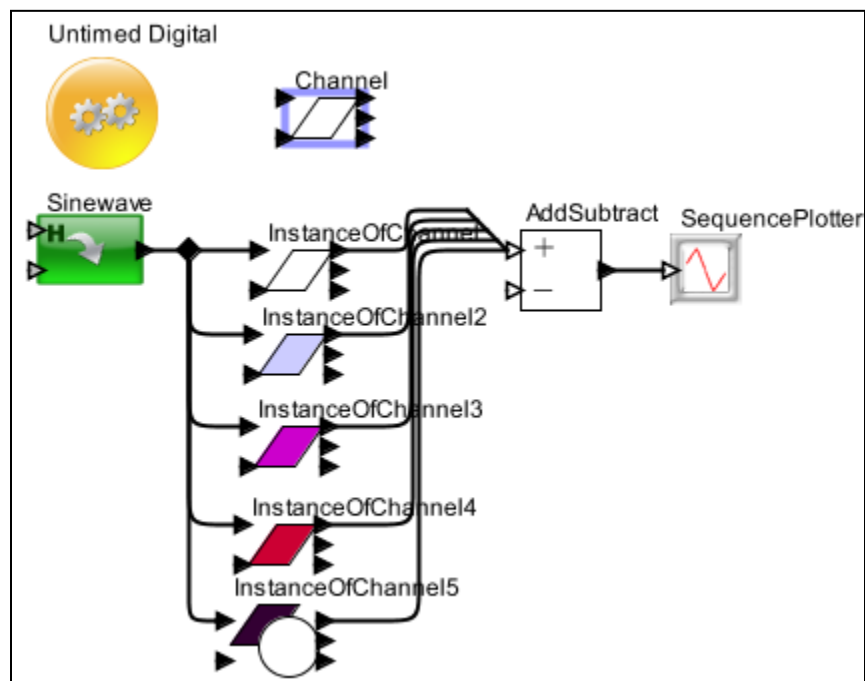


Figure 39.

Model with the icons of the instance

The above figure shows a Model with the icons of the instance changed to override parameter values in the class.

For example, we have modified the custom icons so that each has a different color, and the fifth one has an extra graphical element. To do this, just right click the icon of the instance and select "**Appearance > Edit Custom Icon.**"

8.2.2 Subclass and Inheritance

Suppose now that we wish to modify some of the channels to add interference in the form of another sinewave. A good way to do this is to create a subclass of the Channel class, as shown in figure.

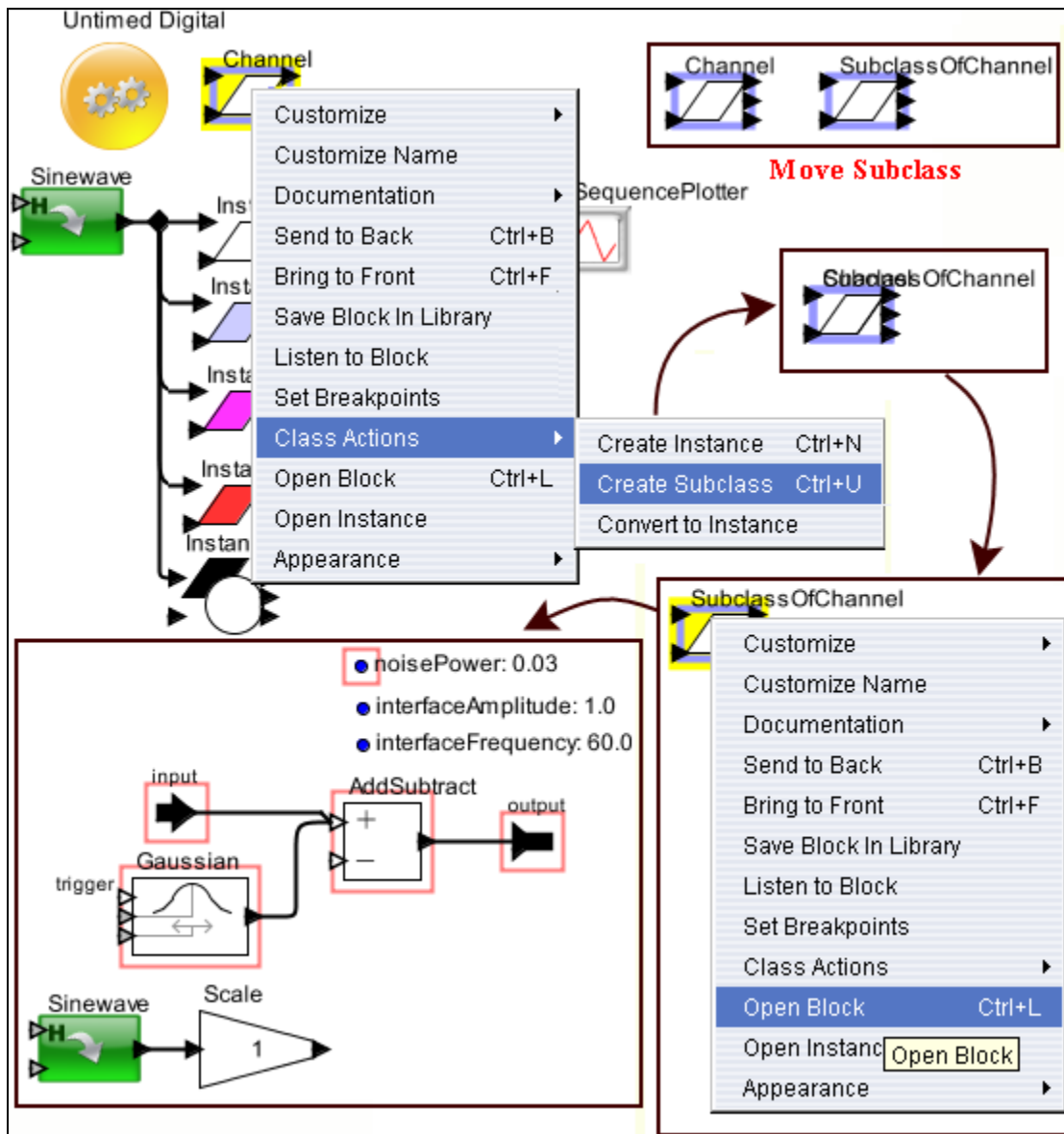


Figure 40.

The model with a subclass of the channel with no overrides

A subclass is created by right clicking the class icon and selecting "**Class Actions > Create Subclass.**" The resulting icon for the subclass appears right on top of the icon for the class, so it needs to be moved over, as shown in the figure.

Looking inside the subclass reveals that it contains all the elements of the class, but with their icons now surrounded by a dashed pink outline. These elements are inherited. They cannot be removed from the subclass (try to do so, and you get an error message). You can, however, change their parameter values and add additional elements. Consider the design shown in below figure.

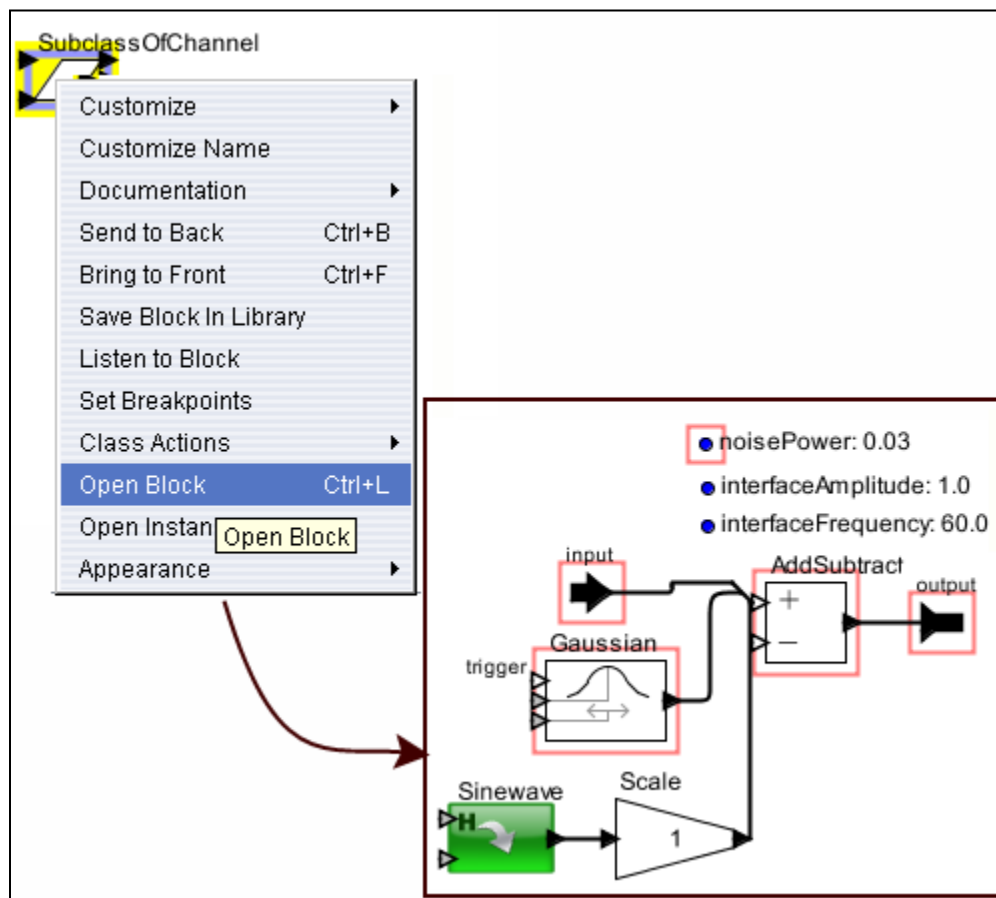


Figure 41. The subclass with overrides that add sinusoidal interferences

The above figure adds an additional pair of parameters named "interferenceAmplitude" and "interferenceFrequency" and an additional pair of actors implementing the interference. A

model that replaces the last channel with an instance of the subclass is shown in the below figure.

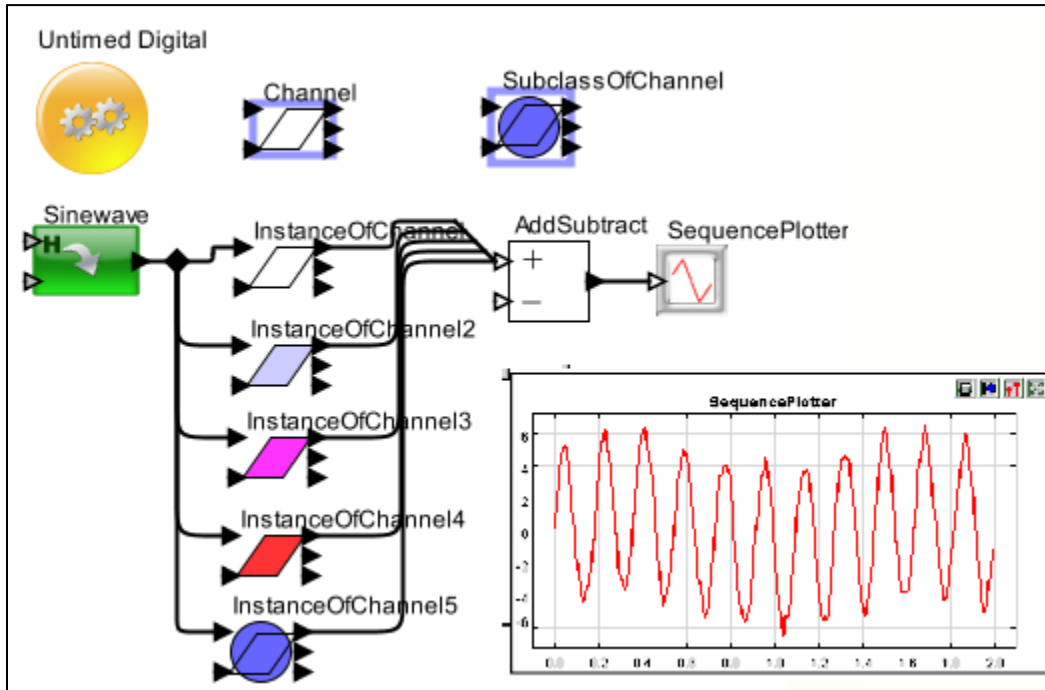


Figure 42.

A model using the subclass form and a plot of the execution

The above model is along with a plot where you can see the sinusoidal interference.

An instance of a class may be created anywhere in a hierarchical model that is either in the same composite as the class or in a composite contained by that composite. To put an instance into a submodel, simply copy (or cut) an instance from the composite where the class is, and then paste that instance into the composite.

8.2.3 Sub-Models

Sub-models are an extension to the above class concept. This is a mechanism to provide a reuse and sharing of modeling components developed by team members. These are classes that are usually well understood and standard elements that require changes to parameter values only.

Classes are maintained within the model where they are used. Sub-models are stand-alone XML files that contain the model definition and a list of parameters. The procedure is still the same. The list of these sub-models can be maintained on an html page with links that can be opened from within VisualSim. Similar to Classes, multiple instances of the same class can exist in a model and each of these instances can have different values for the parameters and icons.

9 Classes and Libraries

Before constructing classes, there are certain procedures to follow. If constructing a class using Java or C++, visit the respective documentation for details. For all blocks, there are some basic rules:

- Edit the VS_Model_Library variable in the VisualSim.bat (.sh) file. This is located in the VS_AR directory. The location of this directory becomes the baseline for all your libraries and custom blocks. The blocks are referenced relative to this base directory.
- If you have C/C+/SystemC blocks, you should also configure the VS_C_Library variable. This is the baseline for all the C++ files and the associated compiled code.
- If you have Java or custom XML class blocks, make sure the base path is included in the CLASSPATH of VisualSim.bat (.sh). All Java and XML classes are accessed relative to this CLASSPATH directory.

9.1 Create Class- Sequence of operations

1. Assemble the initial block diagram

Use the library blocks to assemble the model.

Create a Hierarchical block of this block diagram.

2. Create a class

Convert the Hierarchical block into a Class.

Save as a sub-model.

3. Instantiate a new class for use in a model

To use the Class, you need to instantiate the block in the model using Graph> Instantiate.

Entity. Make sure the Class is located within the VS_Model_Library directory.

4. To test the class

Test the class by constructing a simple model around it.

5. Save in Library. Right-click the block and select Save Block in Library.

9.2 Create Library and Instantiate in User_Library

1. Create a Library

Right-click the User_Library in the Folder and select 'Open for Editing'.

Using Graph > Instantiate Class and entering VisualSim.moml.EntityLibrary, create a new folder.

Note: Make sure the Library name does not end in a numeral.

Convert to Class

Save as sub-model

Do a File > Open and select this file for editing.

Add the blocks that are part of this library.

Note: Make sure there is a Digital Simulator in this library folder. This is needed or you receive an error each time you open VisualSim.

2. Import Library into the User_library

Right-click the User_Library and select 'Open for Editing'.

Using Graph > Instantiate Class, select the library created above.

Select Save.

3. View the Library

Restart VisualSim

The new library is visible in the User_Library.

Rules:

1. Make sure that the library blocks and the library file are located in the same directory. This becomes easy to ship to others.
2. When shipping the library, make sure to zip the file relative to the VS_Model_Library directory. Similarly, when the file is extracted, make sure to extract at the VS_Model_Library level.

A tutorial on creating the Class is provided.

9.3 Example of Creating a New Class Block

Open the above saved Class and assemble the following sequence.

- Open a New Block Diagram Editor

- Instantiate a Digital Simulation and a new Parameter (SimTime).
- Add blocks – Source >Traffic > Traffic, Defining_Flow > Processing and an output port into the model.
- Edit the Parameters of the Processing block as shown in figure 44.
- Select all the items in the block diagram and select Graph->Create Hierarchy.
- A new Hierarchical block appears with a port.
- Remove the wire between the block and the port.
- Select the Hierarchical and Right-click to 'Convert to Class'.
- Open the Hierarchical 'class' and do a File->Save As.
- Select the directory under VS_Model_Library folder to save the file. Make sure to click "Save as sub-model" option. Name the file Traffic_Gen.xml.

Notes:

The "Open Block" shows the original version; whose content should not be modified by the user after it has been qualified. When you do an "Open Instance" for a Hierarchical block in a model, you are opening an instance of the block in the model. You have a local instance where you can add additional parameter or blocks to the model. You cannot modify the original block content. Any changes to the original Class is propagated to all models that use the block.

Notes:

- When you have global variables, Smart_Controllers and Schedulers, it is essential to have their names tied to the Block_Name such that they stay unique in the model. Otherwise the user gets an error when multiple instances are assembled.
- When you have a Script block such as a **Script** or **Smart Controller**, do not modify the Expression Script inside the Instance. This detaches the code from the original. Thus, any changes to the original are not reflected here.
- When creating a new variation of a class; do not open a model containing this block and try to save it as a new class + sub-model. This creates a class of an instance of a class and makes the solution very messy. Use the sub-class approach for doing this.

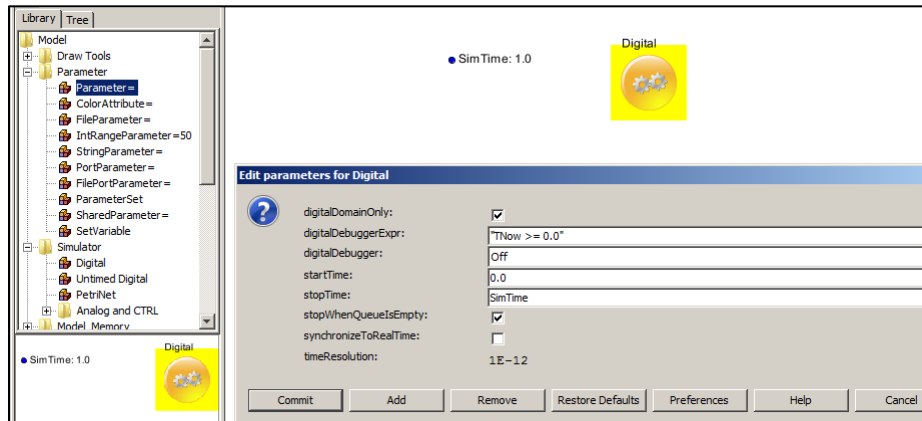


Figure 43.

Create a Block Diagram

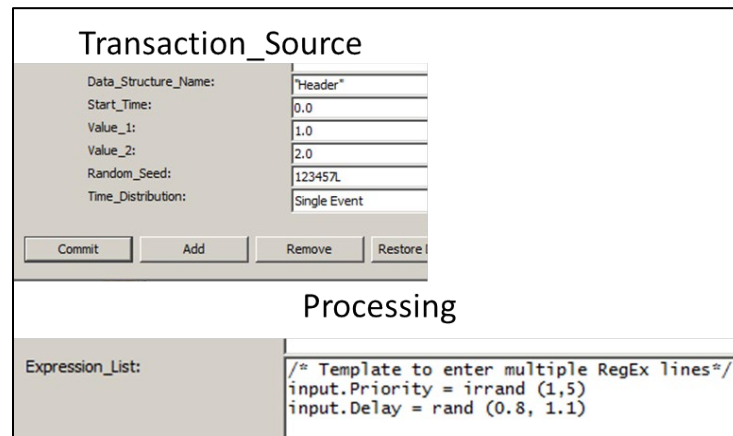


Figure 44.

Block Attributes

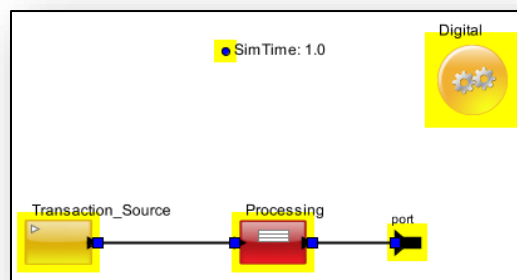


Figure 45.

Making the Connection and Select+All

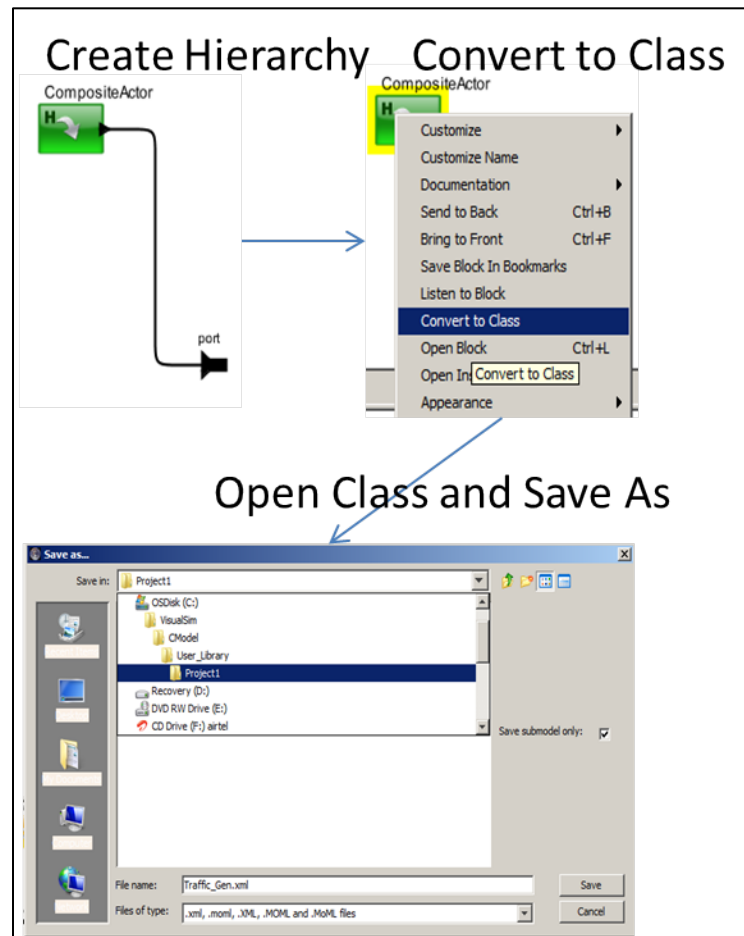


Figure 46.

Create a Class

9.4 Annotating the Class

There are two things to make the Class block interesting. The first is the tooltip and the second is the block icon. The tooltip is done by adding a parameter with name `_explanation`, value in string without "" and class as `VisualSim.kernel.util.StringAttribute`. The icon can be updated by a right-click on the white space and selecting Edit Custom Icon. Make sure to Save the model after this.

9.5 Instantiating the Class in Model

A block that has been compiled to a .class (Java code version) file or a hierarchical block that is contained in a .xml file may be instantiated from ModelBuilder via Graph > Instantiate Entity and used in the model.

To instantiate an entity, follow these steps

1. Start up VisualSim and do File > New > Block Diagram Editor.
2. In the Block Diagram Editor, select Graph > Instantiate Class. Select the class file by traversing through the directory folders. The class must be in a Hierarchy within a Classpath location. It is best to place it within the VS_Model_Lib variable.
3. Click OK.
4. The new block is displayed in the BDE.

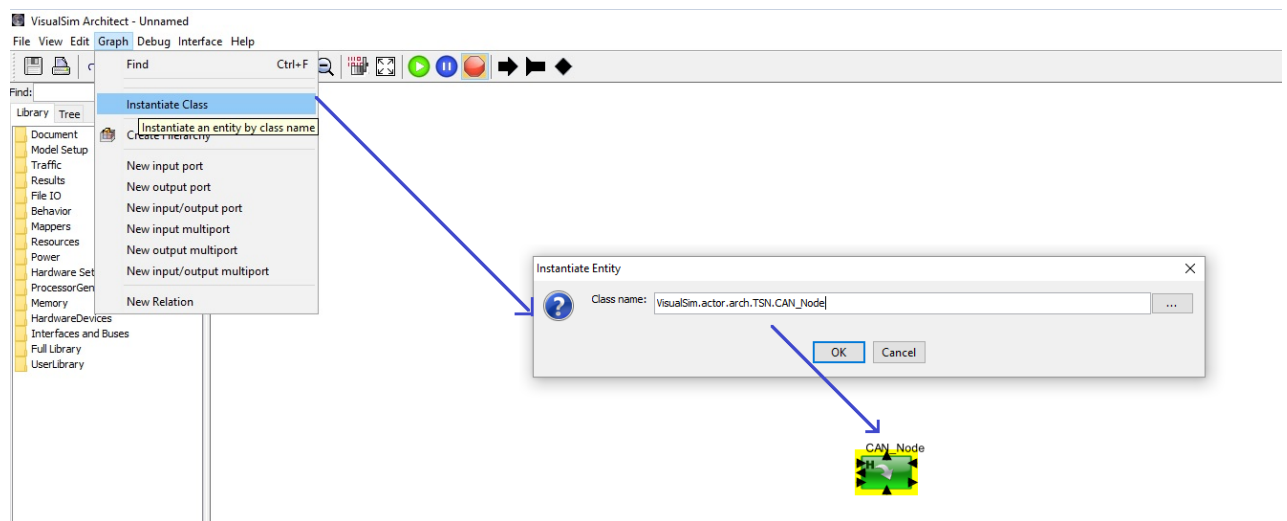


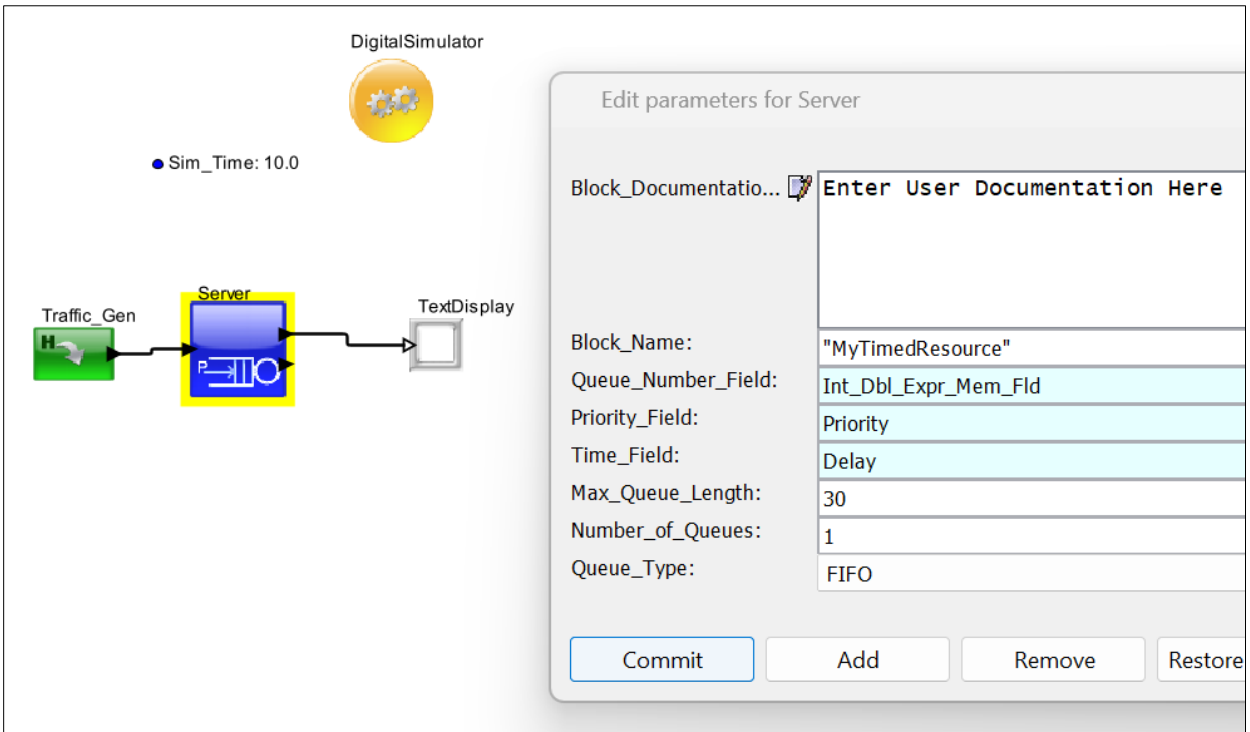
Figure 47.

Instantiate Class to import a Class

9.6 Test the New Class Block:

1. After instantiating the Traffic block, drag the Digital Simulator, parameter, Server, and a TextDisplay to the BDE.
2. Set the parameter to SimTime and assign a value of 10.0.
3. Link the SimTime parameter to the stopTime of the Digital Simulator.
4. Set the parameters of the Server.
5. Connect all the blocks as shown in the figure.

6. Select GO toolbar button to start the simulation.
7. You should see the Header Data Structure in the display.



10 Library Management

10.1 Introduction

Library management is an important enabler of IP reuse and sharing across the corporation. Library management is the organization of the key modeling components and system that are assembled with these systems. The library management system contains the following parts:

- Version control
- Library storage and Directory structure
- Accessing from the Graphical Editor

10.2 Version Control

The version control typically is the corporate storage environment. The solution utilizes a server and client solution. Typical VisualSim components and models are worked on by a single user. So, the version control, as in maintaining revisions, is not very critical. Users can manually maintain various versions in the environment.

10.3 Library Storage

The establishment of a centralized library management structure for a company is important and essential. Mirabilis Design prescribes a standard approach. The central library is accessible by all and can be stored in a version control system. There is a base directory which is <VisualSim_Top>. Within this top directory, there are two sub-structures - components and models.

- Components contain the basic components such as network node, traffic, processor, bus, and statistics creator. The Components can have sub-directories within them for different types of devices.
- Model contains platforms, boards, sub-systems, systems and networked systems. These can be organized within the model folder based on projects.

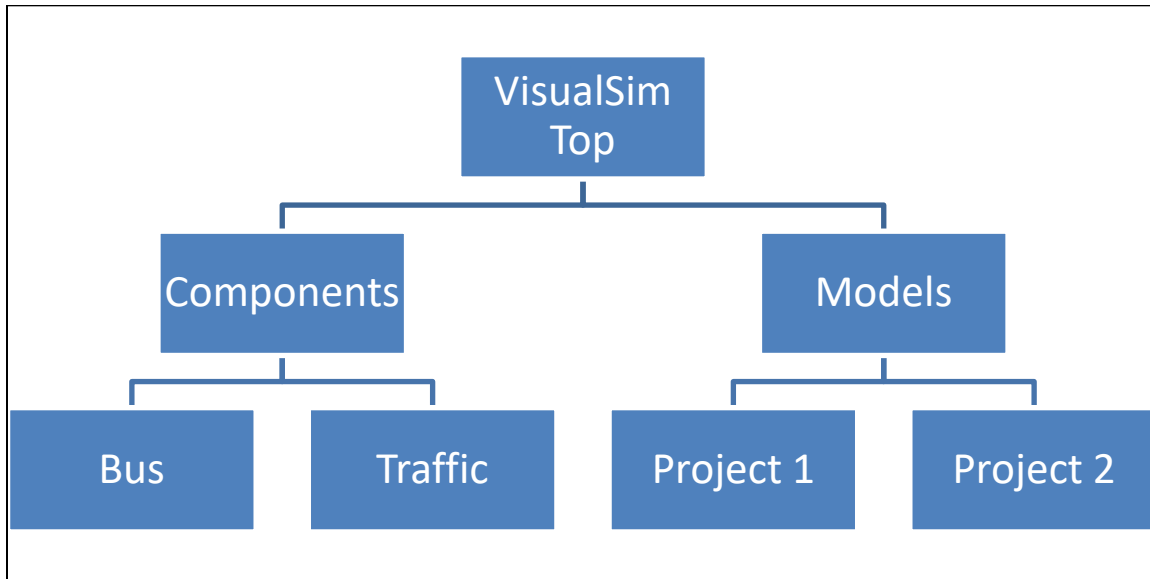


Figure 48. VisualSim Recommended Directory Structure

It is best to create a local copy of this directory for use within VisualSim. This ensures that connectivity losses do not stall creation. The top directory and the names of the directory organization with this top is identical between the central and local structure. Users can depend on version control tools to check in and check out the latest revisions of the files.

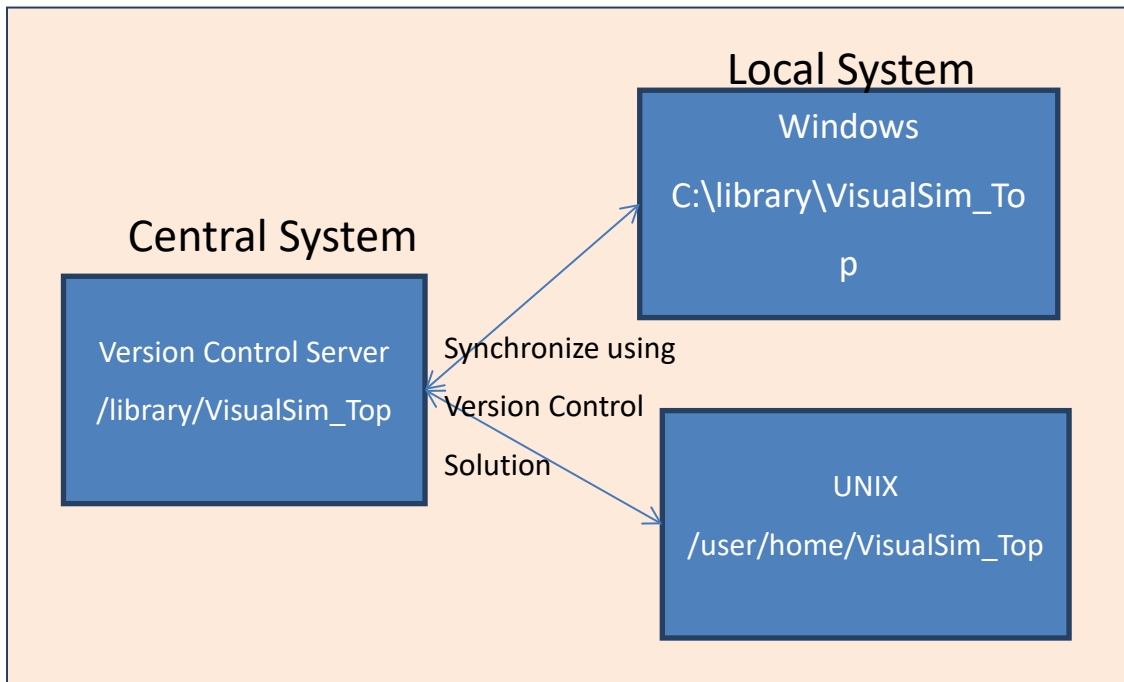


Figure 49. Library Integration between Central and Local System

10.4 User Library

Each user has a User Library, stored in `$HOME/.VisualSim/UserLibrary.xml`.

In the Block Diagram Editor, the User_Library is visible at the bottom of the library palette on the left. After a class has been instantiated, to add it to the User Library, right click the block, and select "Save block in Library".

11 Library Folder Organization

In VisualSim, blocks are grouped together based on functionality and arranged in the Library. VisualSim contains the following libraries:

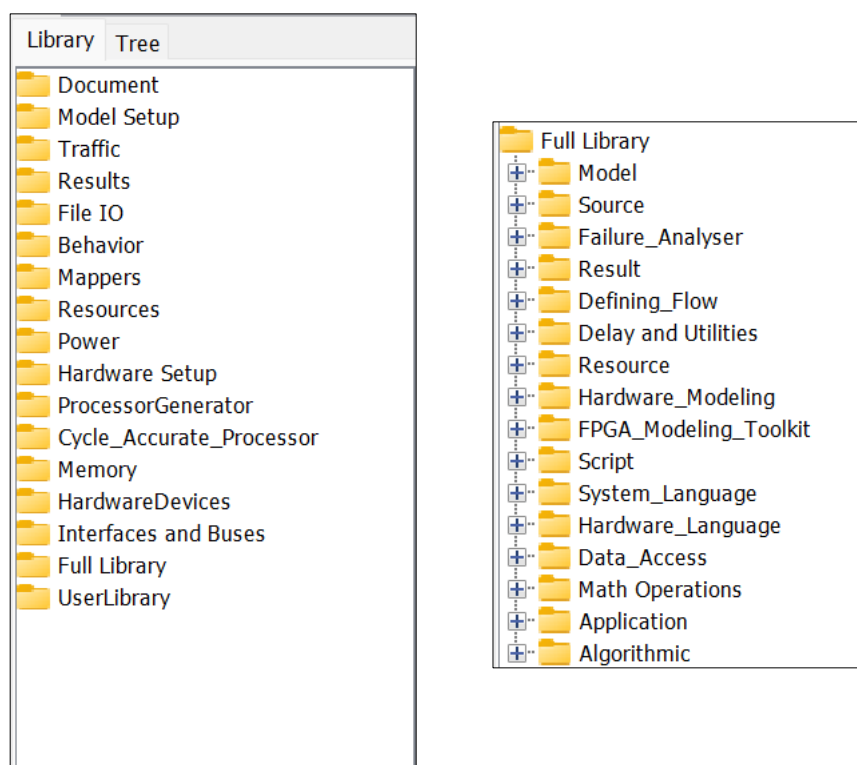


Figure 50.

Library Folder Organization

11.1 Model / Document / ModelSetup

Model / Document / ModelSetup are overhead components used to setup the model. These include drawing tools, simulators, Variable, parameters, hierarchical, and FSM blocks.

Draw Tools – Contains Annotation, Arc, Arrow, Image, Line, Polygon, Rectangle, Eclipse, ModelID.

Parameter – Used for analyzing the performance of the system by fixing their value for a simulation run.

Simulator – Translates the graphical depiction of the system into a form suitable for simulation execution, and executes simulation of the system model.

Variable – Variable that is initialized and updated during the simulation.

Hierarchical – Grouping a set of functional blocks that combine to define a function or device.

FSM – Basic block for Finite State Machine operation.

Utility – Allows the user to determine the current values of Variable and the list of virtual connections. It also has content for checking the type and the units.

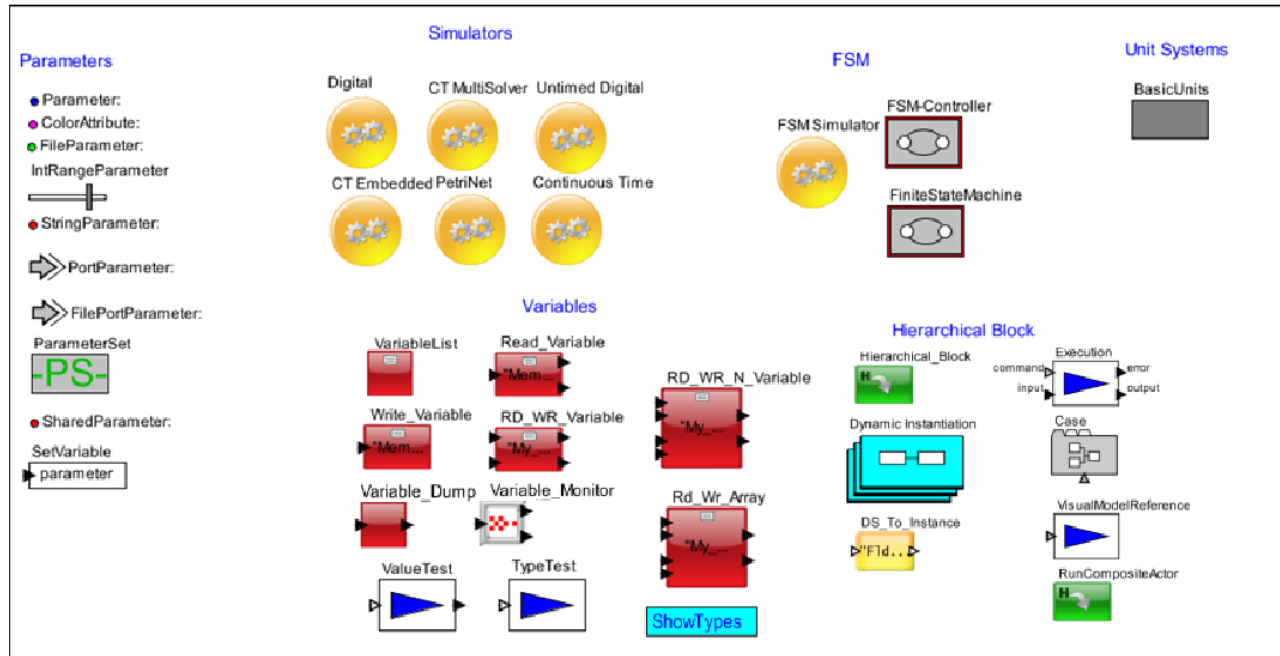


Figure 51. Blocks in Document/ Model / ModelSetup Library Folders

11.2 Traffic / Source

User can generate traffic using a standard distribution, sequences, trace files, clocks, and custom distributions.

Clock – Generate all possible clock inputs.

Event – Generate the input according to the Event set.

File Import – Input can be generated according to the file input,

Traffic – Generate input as per the specified DS and time distribution.

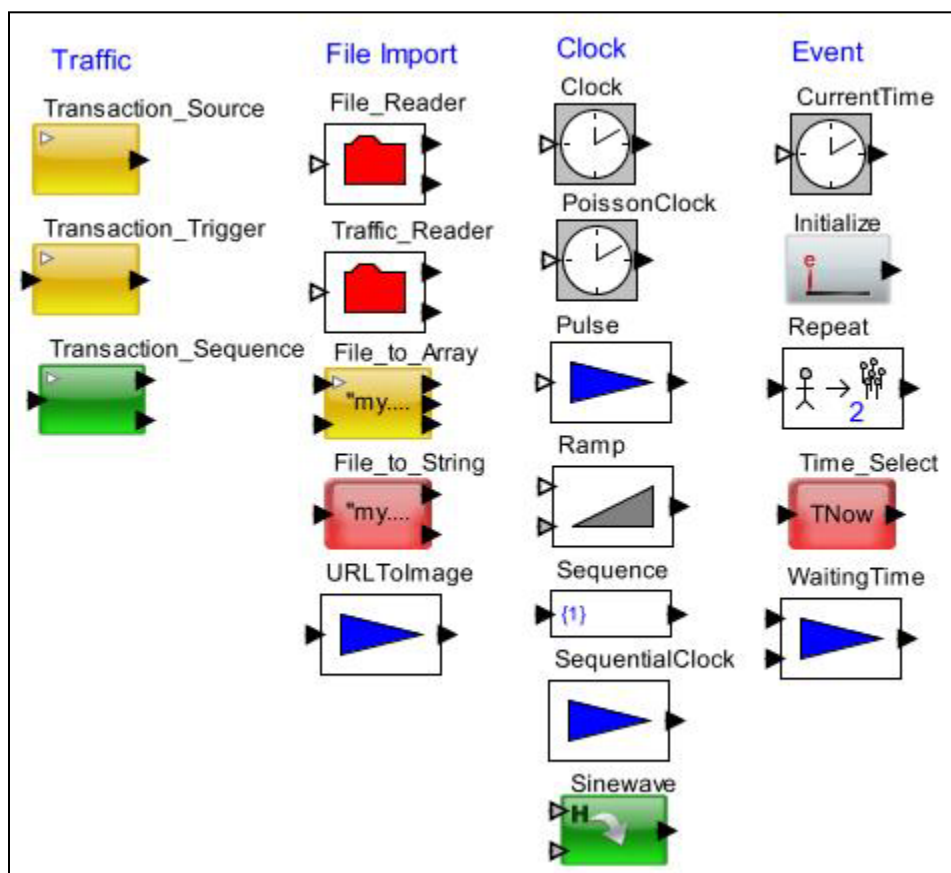


Figure 52.

Blocks in Source Library

11.3 Failure Analyser

The Network Failure block allows user to inject different type of failure in the network to observe the stability of the design.

11.4 Results

The Results Library is used to evaluate the effectiveness of the system model and to estimate the performance.

Plot Manager- Link to the Post Processor.

3D Interactive Creator- Create visual and animated displays.

Statistics Generator - Generate statistics on all resources and hardware blocks.

Plotter - A complete collection of Array plotter, Bar Graph, Histogram plotter, matrix Viewer, Real-time plotters, and so on.

Text – A complete collection of text displays, monitor value, file writers, and so on.

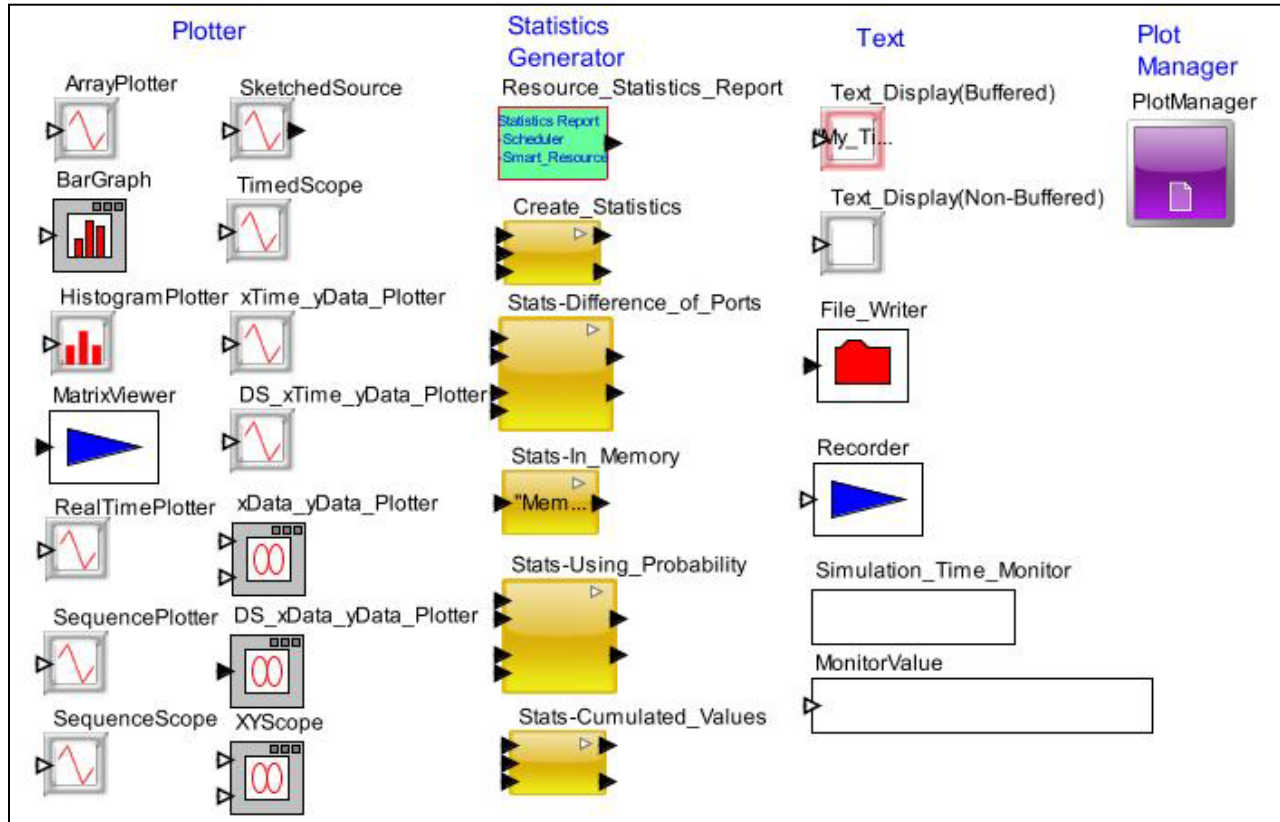


Figure 53.

Blocks in Result Library

11.5 Behavior

Behavior is used in,

- defining expressions,
- evaluating assignments,
- making control decisions and assigning values to a Field or Variable,
- randomizing Field values,
- calculating processing cycles on a scheduler or Queue blocks,

- computing statistics such as latency (TNow - TIME), utilization and throughput and
- creating assumption values.

Control Flow blocks are used to make decisions, create loops and take branches.

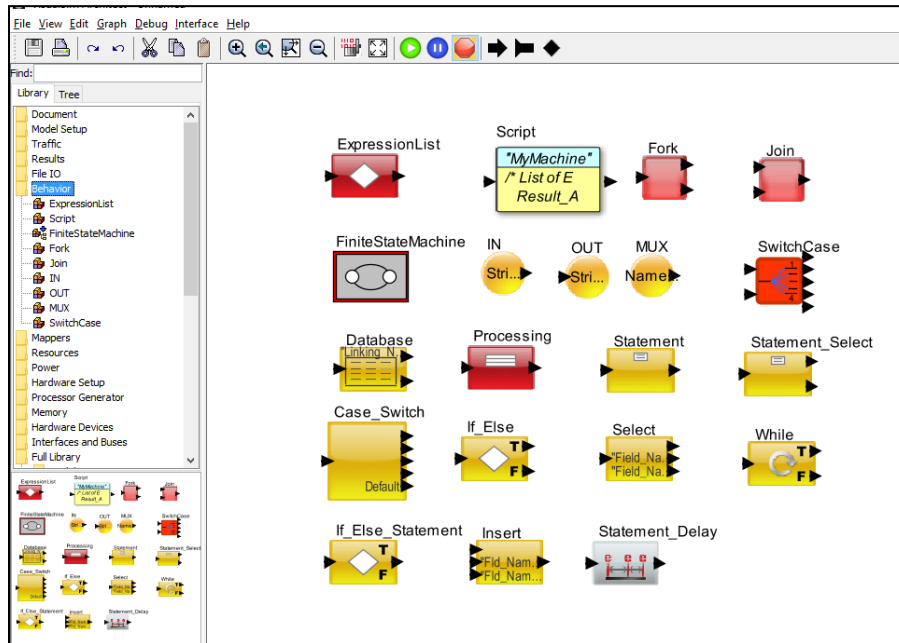


Figure 54.

Control Flow blocks

Access the ExpressionEvaluator from the File->New->ExpressionEvaluator. This window allows the user to experiment with a sequence of math and logic expression.

ExpressionEvaluator- Enter a sequence of assignment statements that are executed in order and the output is controlled by an evaluating an expression.

Script- Programming block to describe behavior using the VisualSim script format and RegEx functions.

Database block - A high-performance lookup table with search, read/write/erase functionality.

Processing- Enter a sequence of assignment statements to be executed in order.

Basic Processing – Contains all decision making and control flow blocks such ExpressionList, Case_Switch, and while.

Virtual Connection – Connections between different parts of the model using names. Two of these Virtual Connection blocks- IN and OUT are in Behavior. The other two blocks are located in the Virtual Connection Folder in Defining Flow Folder.

FSM – Basic block for Finite State Machine operation.

11.6 Mappers

Mapper blocks define the connectivity between the behavior flow and the architecture flow, and within the architecture flow using a named connection. The connection can be static, that is to a single scheduler, or dynamic is based on the value of the input field content.

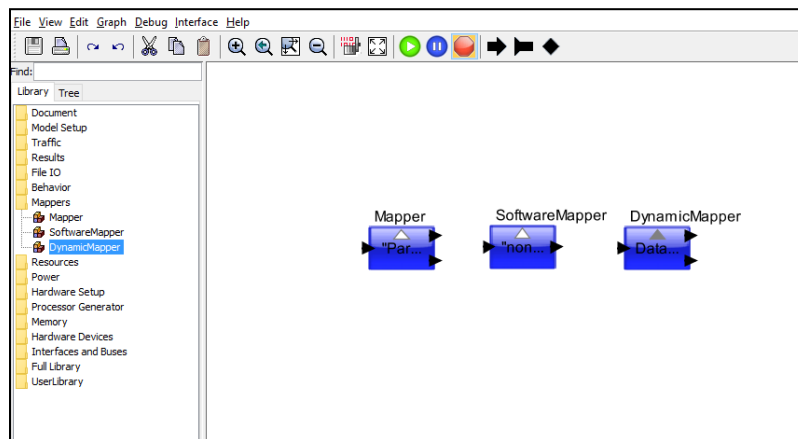


Figure 55.

Mappers

11.7 Resources

- Resources model the physical aspects of the system. In the performance of a system design, a model platform is described using the physical components such as **Queue**, **Schedulers**, **Flow Control**, and **Associated Arbitration Policies**. There are two ways to define resources - **Active** and **Passive**. Active resources consume time while passive resources consume quantity. Passive resources can be created using **Arrays**, **Database** block, or **Quantity-shared Resources**. All other blocks under resources are categorized as Active.
- Queues and Servers** – Queues with advanced content handling for both time-based and event-based. Define complex arbitration schemes that are dependent on external activity for timing and pop.

- **Channel** - The Channel block can be used to model a DMA channel, wireless channel or a bus. The channel block contains a queue and multiple channels.
- **Event Queue** - Queue stores the incoming data structure and pop it out on the arrival of a trigger. The blocks in this folder have single and multiple concurrent queues with priority.
- **Timed Queue** - Queue stores the incoming data structure and pop it out after a pre-determined delay. The blocks in this folder have single and multiple concurrent queues with priority enabled. Two variations of these blocks are also available- one with a single queue and multiple parallel time resources; and another scheduler that supports slot-based time allocation.
- **Quantity Based** - A polymorphic quantity-shared resource for N-identical resources, which enqueues input tokens in either a FIFO, or LIFO, order depending on the "Queuing_Discipline" parameter attribute.
- **SystemResource** – Defines the execution of a process in a data structure in a variety of pre-defined scheduling schemes. This block separates the behavior flow or data flow from the architecture definition. SystemResource block can be used to link multiple concurrent behavior flows into a single block.

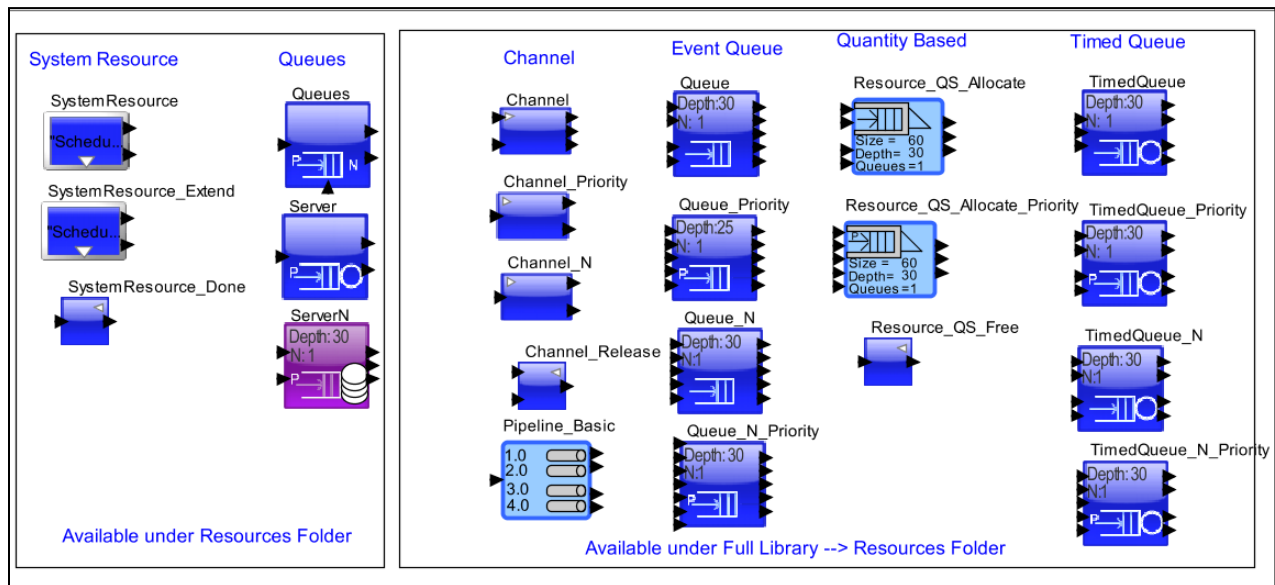


Figure 56.

Blocks in Resource library

11.8 Power

Power Modeling Library blocks are available in this library folder.

PowerTable – Blocks are used to evaluate the following for individual devices - battery discharge, instantaneous power, average power, and so on.

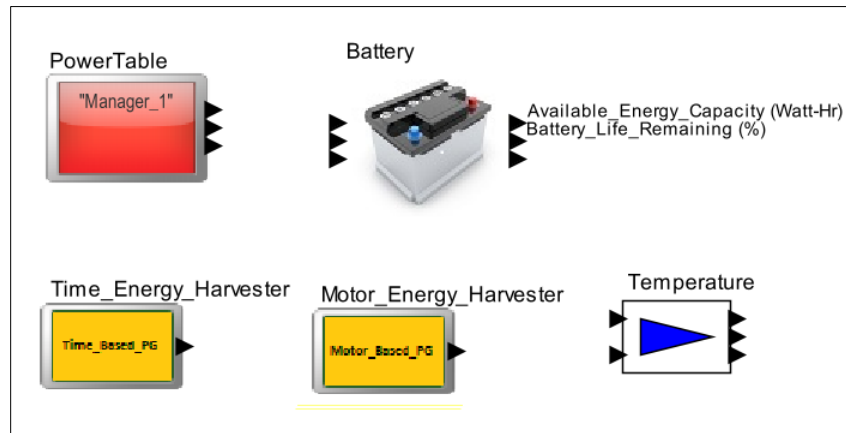


Figure 57.

Power

11.9 Hardware Setup

The Hardware setup provides the list of library blocks that must be or may be included as part of a cycle accurate system model. ArchitectureSetup block handles all the address mapping, routing, plotting, statistics and debugging for the Hardware Modeling components. There can be multiple ArchitectureSetup blocks in a model. Each block instance must have a unique name. All Bus and Hardware blocks must be associated with one of the ArchitectureSetup.

Device Interface block helps the designer to model custom device interfaces and custom hardware components. Hardware setup block set also includes few utility library blocks such as VCD writer, Timing Diagram generator, and clock align.

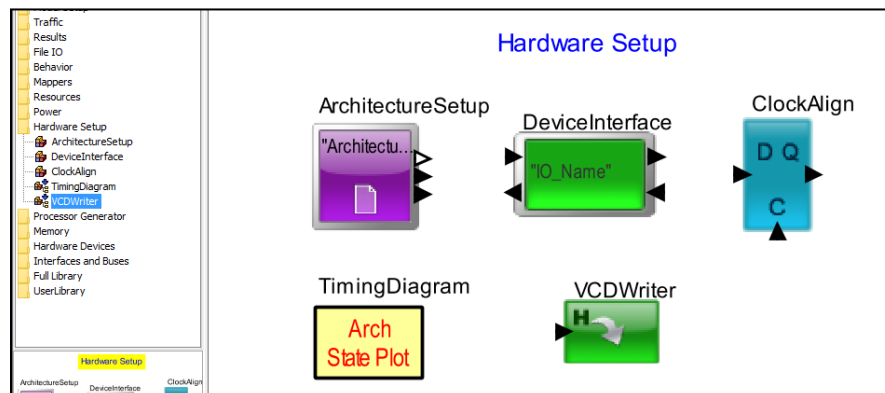


Figure 58.

Hardware Setup

11.10 Processor Generator

The Processor block set is used to model variations of commercial and proprietary processors. The Processor generator block set also includes an Instruction set block. The Instruction_Set block can be used to define a set of instructions for each execution unit of the processor. It can also bunch together execution units with a common reference name. This reference name can be used in the Processor pipeline definition. The Processor block can have cache definitions within and also refer to an external cache or memory blocks. The last block is the Task_Generator block that generates an array of instructions based on the profile provided in the Instruction_Mix_Table.

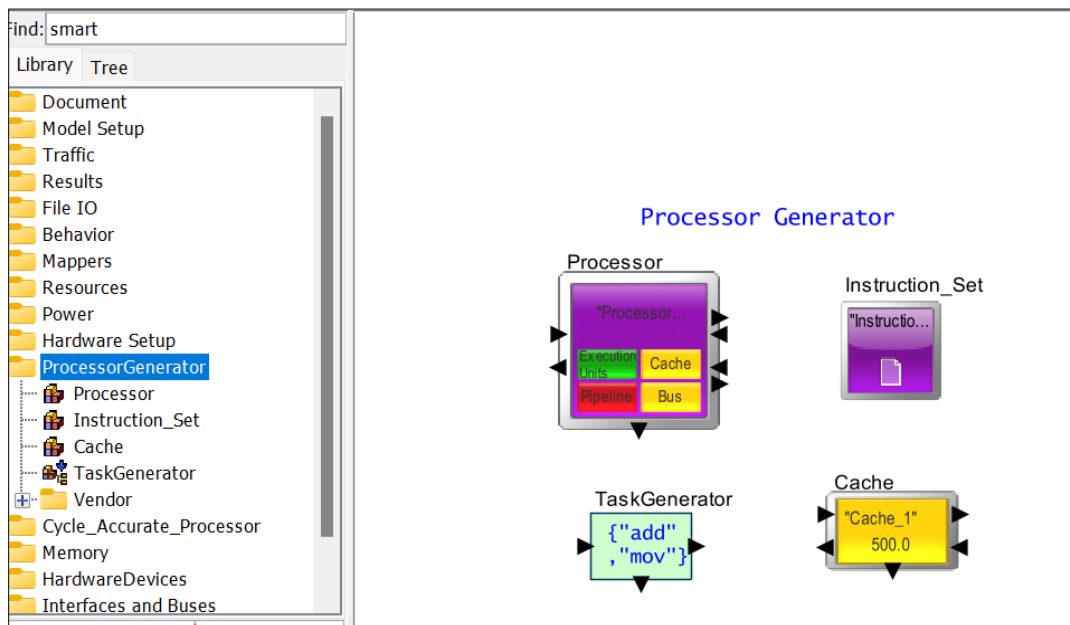


Figure 59.

Processor Generator

11.11 MicroArchitecture Components

Microrarchitecture library allows user to create an existing architecture or a custom architecture in a low level design. The blocks performs behavioral computation, maintain processor states, and perform branch mispredictions and flushes based on the computed values.

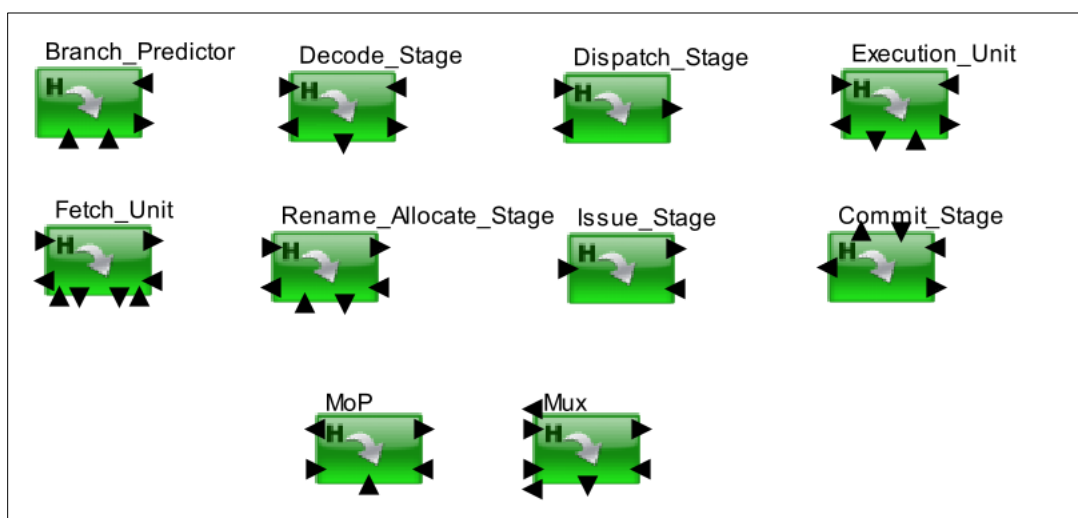


Figure 60. Microarchitecture Components

11.12 Memory

This library folder has the configurable libraries of cache, Memory and Memory controllers. Types of Memory technologies supported are SRAM, SDRAM, DDR, DDR2, DDR3, DDR4, DDR5, LPDDR, LPDDR2, LPDDR3, LPDDR4 and LPDDR5.

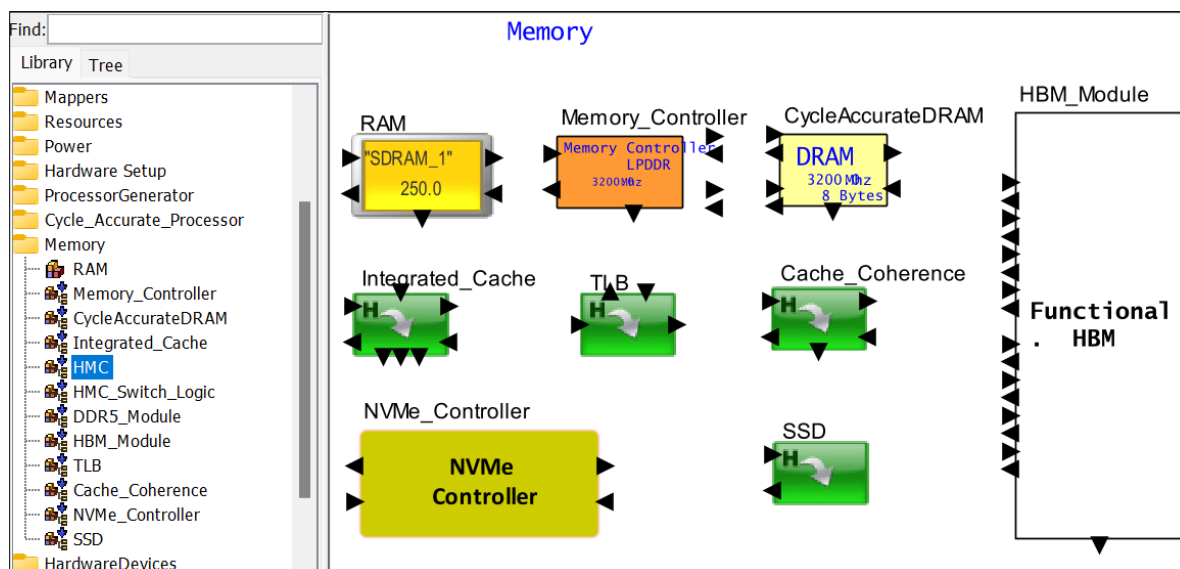


Figure 61.

Memory

11.13 Hardware Devices

Hardware Device blockset includes library blocks of Custom Bus arbiter, Custom Bus Interface, DMA Controller, Blocking Switch, Bridge, Crossbar, and Switch.

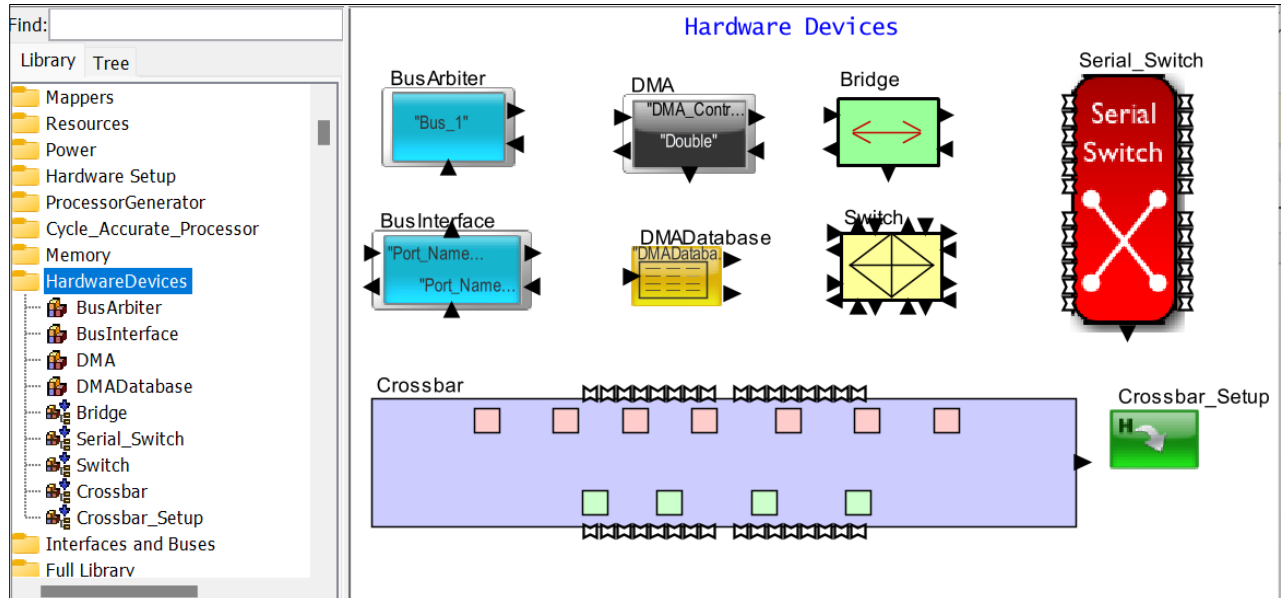


Figure 62.

Blocks in Hardware Devices

11.14 Interfaces and Buses

Interfaces and Buses blockset includes list of advanced bus and interface technologies. This includes library blocks of AFDX, AMBA, AVB, Autosar, CAN, Switched Ethernet, Fibre Channel, FireWire, FlexRay, PCI, PCIe, RapidIO, Spacewire, TTEthernet, Wireless, and networking library blocks.

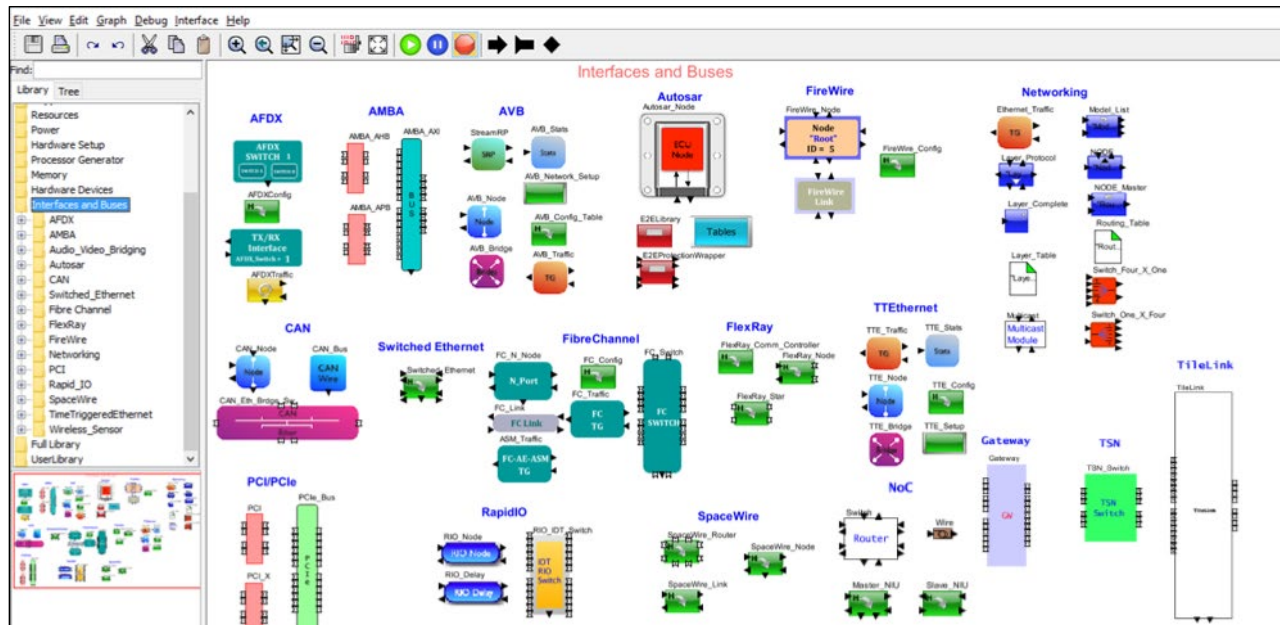


Figure 63.

Interfaces and Buses

The Wireless Sensor networks library contains a separate simulator that allows communication wirelessly between the channel and the operating blocks. This library contains different antennas and channels.

11.15 System_Language

The System_Language blocks are available under Full Library > System_Language folder. These blocks provide an interface to easily import C code, applications written in C or C++, Python, MatLab, and Satellite Toolkit. The Interfaces require the licenses for MatLab and Satellite Toolkit.

11.16 Hardware Language

The Hardware Language blocks are available under Full Library > Hardware Language library folder. Interfaces are provided to run VisualSim in co-simulation with Verilog and SystemC. The Interfaces require the licenses for Verilog.

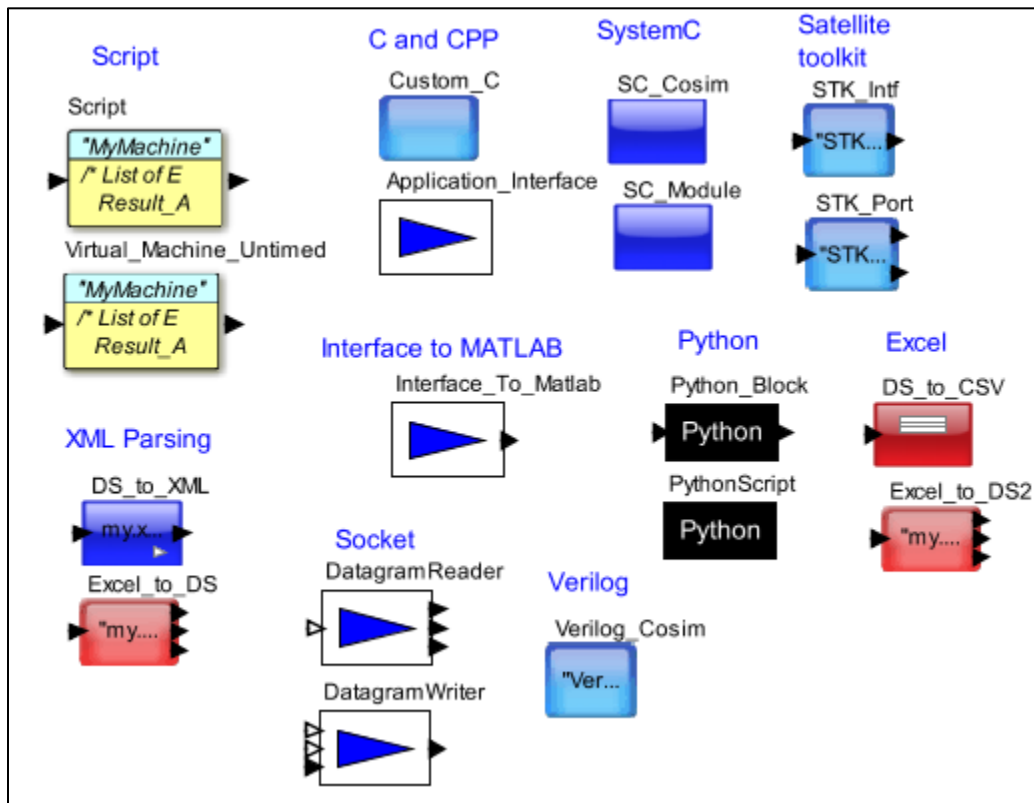


Figure 64. Interface Blocks

11.17 Math_Operations

The Math_Operations blocks are available in Full Library > Math Operations. This library contains Array, Boolean and Logic blocks that are required for the operation of system model.

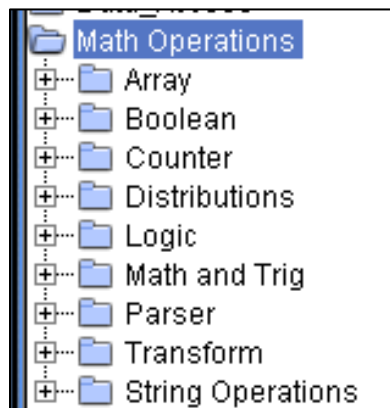


Figure 65. Math Operations Listing

11.18 Algorithmic

The Algorithmic blocks are available under Full Library > Algorithmic. This library contains blocks to generate events and wave-forms. It also contains image processing, Petri net, and signal processing blocks.

11.19 UserLibrary

The User Library is maintained in the \$User Home/.VisualSim/User_Library.xml file. Users can add Hierarchical folders with this Library. In addition, the user can add libraries using the Block Diagram Editor.

12 Auto-Save

VisualSim provides the ability for the user to continuously save the currently open model, if they are modified. The interval between the saves is set in the VSconfig.properties. The format is:

Auto_Save_Time=2

Where the number on the RHS is the time interval between saves.

The intermediate xml files are saved in the <User Home>/.VisualSim directory. The format is as follows:

Scheduler_SW_FCFS_Preempt_20151127_143903.xml

Where

File Name: Scheduler_SW_FCFS_Preempt

Date: 20151127- Here is it 2015 November 27

Time: 143903 is 2:39:03 PM

Note: The intermediate files are not deleted when the model is saved or VisualSim is closed.

13 Simulation Technology

13.1 Introduction

The simulators implement various models of computation. Most of these models of computation can be viewed as a framework for component-based design, where the framework defines the interaction mechanism between the components. VisualSim consists of four major simulators-

- Digital
- Digital Untimed
- Continuous Time
- Finite State Machine
- SystemC

Except for the Finite State Machine, all the simulators implement their own scheduling between blocks and do not rely on threads. These usually results in a highly efficient execution. The FSM simulator is in a category by itself, as the components are not producers and consumers of data, but rather are states.

13.2 Models of Computation

A Model of Computation gives an interaction mechanism for components to deal with concurrency and time. The utility of a model of computation stems from the modeling properties that apply to all similar models. For many models of computation these properties are derived through formal mathematics. Depending on the model of computation, the model may be deterministic, statically schedulable, or time safe. Because of its modeling properties, a model of computation represents a style of modeling that is useful in any circumstance where those properties are desirable. In other words, models of computation form design patterns of component interaction, in the same sense that Gamma and others. describe design patterns in object oriented languages. For a particular application, an appropriate model of computation does not impose unnecessary constraints, and at the same time constrained enough to result in useful derived properties.

13.3 Simulators- How to select the right one?

1. What are Models of Computation: Govern interaction of components in a model. The simulators share kernel, event queue, and some libraries. The transfer of data and the exchange of timing events between simulators are transparent to the user.
2. Single Kernel vs. Co-Simulation: Synchronization has to be done externally and by each independent kernel. Having independent threads running. SDF + DE in VisualSim vs. VisualSim + VHDL are the difference.
3. Discrete-Event: Execution of synchronous events between asynchronous times. The Calendar Queue maintains sequences of events on a time line. This is similar to Hyperformix Workbench, OPNeT, BONEs, Verilog, and VHDL.
4. Finite State Machine: Case statement in a graphical form defining sequential control logic. The entities represent states and connections represent transitions between states. The execution is a strictly ordered sequence of state transitions. This is similar to Statemate, OPNeT, and Summit Visual FSM.
5. Synchronous Data Flow: Can specify *a priori* the number of input samples consumed on each input and the number of output samples produced on each output each time the block is invoked. The execution is statistical and determined prior to execution. This similar to SPW. Cossap or Synopsys System Studio is a small variation to this in the sense that Cossap evaluates a large number of inputs at every port before moving to the next port/block. This is extremely fast if the model is flow-through. This slows down if there are control elements such as loop-back.
6. Continuous Time: Simulator finds a fixed-point, i.e., a set of continuous-time functions that satisfy all the relations. This solves algebraic or differential relations between inputs and outputs. This is similar to Spice, Simulink and AMS. To determine the best variation of the Continuous Time simulator for your application, refer to Chapter 1, Section 2.4.
7. FSM Hierarchical: A FSM Hierarchical is one that switches between these simple models when the system transitions between regimes or states. A CT + FSM creates a modal model.
8. Mixed-Signal: Embedded systems frequently contain components that are best modeled using differential equations, such as MEMS and other mechanical components, analog circuits, and

microwave circuits. These components, however, interact with an electronic system that may serve as a controller or a recipient of sensor data. This electronic system may be digital. Joint modeling of a continuous subsystem with digital electronics is known as mixed signal modeling.

13.4 Simulators

In this section, we describe models of computation that are implemented in VisualSim simulators.

13.4.1.1 Continuous Time (CT)

In the CT simulator (continuous time), actors represent components that interact via continuous-time signals. Actors typically specify algebraic or differential relations between inputs and outputs. The job of the director in the simulator is to find a fixed-point, that is, a set of continuous-time functions that satisfy all the relations.

The CT simulator includes an extensible set of differential equation solvers. The simulator, therefore, is useful for modeling physical systems with linear or nonlinear algebraic/differential equation descriptions, such as analog circuits and many mechanical systems. Its model of computation is similar to that used in Simulink, Saber, and VHDL-AMS, and is closely related to that in Spice circuit simulators.

Embedded systems frequently contain components that are best modeled using differential equations, such as MEMS and other mechanical components, analog circuits, and microwave circuits. These components, however, interact with an electronic system that may serve as a controller or a recipient of sensor data. This electronic system may be digital. Joint modeling of a continuous subsystem with digital electronics is known as mixed signal modeling. The CT simulator is designed to interoperate with other VisualSim simulators, such as Digital, to achieve mixed signal modeling. To support such mixed-modeling, the CT simulator models the discrete events as Dirac delta functions. It also includes the ability to precisely detect threshold crossings to produce discrete events. Physical systems often have simple models that are only valid over a certain regime of operation. Outside that regime, another model may be appropriate. A modal model is one that switches between these simple models when the system transitions between regimes. The CT simulator interoperates with the FSM simulator to create modal models.

13.4.1.2 Digital (DE)

In the discrete-event (DE) simulator, the actors communicate via sequences of events placed in time, along a real time line. An event consists of a value and time stamp. Actors can either be processes that react to events (implemented as Java threads) or functions that fire when new events are supplied. This model of computation is popular for specifying digital hardware and for simulating telecommunications systems, and has been realized in a large number of simulation environments, simulation languages, and hardware description languages, including VHDL and Verilog.

DE models are excellent descriptions of concurrent hardware, although increasingly the globally consistent notion of time is problematic. In particular, it over-specifies (or over-models) systems where maintaining such a globally consistent notion is difficult, including large VLSI chips with high clock rates. Every event is placed precisely on a globally consistent time line.

The DE simulator implements a sophisticated discrete-event simulator. Other DE simulators in general need to maintain a global queue of pending events sorted by time stamp (this is called a priority queue). This is expensive, as inserting new events into the list requires searching for the right position at which to insert it. The VisualSim DE simulator uses a calendar queue data structure for the global event queue. A calendar queue may be thought of as a hash-table that uses quantized time as a hashing function. As such, both enqueue and dequeue operations can be done in time that is independent of the number of events in the queue.

In addition, the DE simulator gives deterministic semantics to simultaneous events, unlike most competing discrete-event simulators. This means that for any two events with the same time stamp, the order in which they are processed can be inferred from the structure of the model. This is done by analyzing the graph structure of the model for data precedence so that in the event of simultaneous time stamps, events can be sorted according to a secondary criterion given by their precedence relationships. VHDL, for example, uses delta time to accomplish the same objective.

13.4.1.3 Finite State Machine (FSM)

The finite-state machine (FSM) simulator is radically different from the other VisualSim simulators. The entities in this simulator represent not block/actor but rather state and the connections represent transitions between states. Execution is a strictly ordered sequence of state transitions.

The FSM simulator leverages the built-in Expression Language in VisualSim to evaluate guards, which determine when state transitions can be taken.

FSM models are excellent for control logic in embedded systems, particularly safety-critical systems. FSM models are amenable to in-depth formal analysis, and thus can be used to avoid surprising behavior.

FSM models have a number of key weaknesses. First, at a very fundamental level, they are not as expressive as the other models of computation described here. They are not sufficiently rich to describe all partial recursive functions. However, this weakness is acceptable in light of the formal analysis that becomes possible. Many questions about designs are decidable for FSMs and undecidable for other models of computation. A second key weakness is that the number of states can get very large even in the face of only modest complexity. This makes the models unwieldy. Both problems can often be solved by using FSMs in combination with concurrent models of computation. This was first noted by David Harel, who introduced Statechart formalism.

Statecharts combine a loose version of synchronous-reactive modeling (described below) with FSM. FSMs have also been combined with differential equations, yielding the so-called hybrid systems model of computation.

The FSM simulator in VisualSim can be hierarchically combined with other simulators. We call the resulting formalism “*charts” (pronounced “starcharts”) where the star represents a wildcard. As most other simulators represent concurrent computations, *charts model concurrent finite state machines with a variety of concurrency semantics. When combined with CT, they yield hybrid systems and modal models. When combined with SDF (described below), they yield something close to Statecharts.

13.4.1.4 Untimed Digital or Synchronous Data Flow

The synchronous dataflow (SDF) simulator handles regular computations that operate on streams. Dataflow models are popular in signal processing. Dataflow models construct processes as sequences of atomic actor firings. Synchronous dataflow (SDF) is a particularly restricted special case with the extremely useful property that deadlock and boundedness are decidable. Moreover, the schedule of firings, parallel or sequential, is computable statically, making SDF extremely useful specification formalism for embedded real-time software and for hardware.

Certain generalizations sometimes yield to similar analysis. Boolean dataflow (BDF) models sometimes yield to deadlock and boundedness analysis, although fundamentally these questions are undecidable. Dynamic dataflow (DDF) uses only run-time analysis, and thus makes no attempt to statically answer questions about deadlock and boundedness.

13.4.1.5 SystemC

SystemC is an event-driven simulator providing hardware-oriented constructs within the context of C++ as a class library implemented in standard C++. SystemC use spans design and verification from concept to implementation in hardware and software. SystemC provides an interoperable modeling platform which enables the development and exchange of very fast system-level C++ models. It also provides a stable platform for development of system-level tools.

The library provides a set of data types implementing various data representations needed for hardware modeling and certain types of software programming. These include 2-valued and 4-valued bit vectors of arbitrary width, and fixed-point representations. Also included in the core language are modules and ports for representing structure, as well as interfaces and channels that describe communication. Finally, the library includes a set of built-in primitive channels that have wide use such as signals and FIFOs.

A SystemC system consists of a set of one or more modules. Modules provide the ability to describe structure. Modules may contain processes, ports, internal data, channels, and instances of other modules. All processes are conceptually concurrent and can be used to model the functionality of the module. Ports are objects through which the module communicates with other modules. The internal data and channels provide for communication between processes and maintaining module state. Module instances provide for hierarchical structures. The interface, port, and channel structure provides for great flexibility in modeling communication and in model refinement.

13.4.1.6 Wireless Sensor networks

Modeling of wireless sensor networks requires sophisticated modeling of communication channels, sensor channels, ad-hoc networking protocols, localization strategies, media access control protocols, energy consumption in sensor nodes, and so on. This modeling

framework is designed to support a component-based construction of such models. It is intended to share models of disjoint aspects of the sensor nets problem and to build models that include sophisticated elements from several aspects.

13.4.2 Choosing Models of Computation

The rich variety of concurrent models of computation outlined in the previous section can be daunting to a designer faced with having to select them. Most designers today do not face this choice because they get exposed to only one or two.

Embedded software and digital electronics designers exclusively utilize the Discrete-Event simulator while DSP and image processing architects utilize the Synchronous Data Flow. Protocols are best done using the combination of Discrete-Event and Finite State Machine while MEMS and control systems prefer Continuous Time and Finite State Machine. A special case of combining simulators is the Continuous Time and Finite State Machine. This combination is called a Hybrid System.

There are three CT Simulators:

- CTMultiSolverSimulators - Top-level director only
- CTSimulators - Top-level or inside a composite actor
- CTEmbeddedSimulators - Contained only in a CTCompositeActor

In terms of mixing models of computation, all the directors can execute composite actors that implement other models of computation, as long as the composite actors are properly connected. Only CTMixedSignalDirector and CTEmbeddedDirector can be contained by other domains. The outside domain of a composite actor with CTMixedSignalDirector can be Discrete Event. The outside domain of a composite actor with CTEmbeddedDirector must also be CT or FSM, if the outside domain of the FSM model is CT.

An essential difference between concurrent models of computation is their modeling of time. Some are very explicit by taking time to be a real number that advances uniformly, and placing events on a time line (Discrete-Event) or evolving continuous signals along the time line (Continuous Time). Others are abstract and take time to be discrete (Synchronous Data Flow).

A grand unified approach to modeling would seek a concurrent model of computation that serves all purposes. This could be accomplished by creating a *melange*, a mixture of all of the above, but such a mixture would be extremely complex and difficult to use. Another alternative would be to choose one concurrent model of computation, say the Continuous-Event, and show that all the others are subsumed as special cases. This is relatively easy to do, in theory. Most of these models of computation are sufficiently expressive to be able to subsume most of the others. However, this fails to acknowledge the strengths and weaknesses of each model of computation. Thus, to design interesting systems, designers need to use heterogeneous models.

The approach used in VisualSim is to provide in the infrastructure an *abstract semantics*, rather than a unifying model of computation. It is “abstract” in the sense that it is not a complete model of computation. For example, the abstract semantics of the block package asserts that blocks “fire,” but it says nothing about when they fire. This makes it possible to define blocks that can operate in several simulators (we call these *simulator polymorphic* blocks).

14 Concept of Time

14.1 Introduction

The **Simulator** controls the concept of time and the advancement of the simulations schedule. As VisualSim is a multi-domain simulator, the time is synchronized across multiple simulations. Some Simulators such as the un-timed digital have no concept of time. These Simulators move the data token from one output port to the next input port.

During model initialization, a topology map of the entire model is constructed. Each **Hierarchical** block can contain a **Simulator** icon instance. This partitions the model structure for the local simulators to handle the processing and the master simulator to manage the central communication and the **Master Clock**. For time-based **Simulators** such as **Discrete-event Simulator**, it is important to add a **Simulator** icon to each level of the Hierarchy for large models. This keeps the events occurring at that level of hierarchy local and maintains a smaller calendar of events, thus increasing simulation performance.

Sub-models or classes must always have a Simulator instantiated in them.

14.2 Time Resolution

Time Resolution of the **Simulator** is provided as an attribute of the **Simulator** icon. This is a **double** value which defaults to **1.0e-12**. The **Time Resolution** of the **Simulator** must be smaller than the smallest delay in the model. **Zero** is not a possible value. If the smallest time in your model is **0.1** ns, then the **Time Resolution** must be smaller than **0.05** ns. The Time Resolution is used to resolve any contention occurring at the same time.

Note: If you have identical results for different time resolutions, the time resolution must always be of the form $1.0e(x)$ which x is a positive or negative integer. The time resolution cannot be $0.5e-11$ or something similar. It must always be $1.0e(x)$.

Blocks in different levels of the hierarchy or even within the same hierarchy can be fired in synchronous time. This means that there is no advance in time between the individual firing. This is called concurrent operation. In real life this would be a processor which is acting on

instructions in two **Execution Units**, while another **Execution Unit** is waiting for a **Cache** access.

14.3 Simulation Time

The simulation time depends on the number of events in the model and the time resolution. The StopTime in the Continuous and Digital simulator depend on the time resolution. The largest simulation time can be derived using the following formula:

$$\text{TimeResolution} = 1.0\text{e-}12$$

$$\text{Bits} = \text{floor}(-1 * \log_2(\text{TimeResolution})) + 1 = 40.0$$

$$\text{maximumGain} = 52 - \text{Bits} = 12$$

$$\text{stopTime} = 0.90 * \text{pow}(2.0, \text{maximumGain}) = 3686.4$$

The simulation stops at this point and any remaining events are not completed.

If all the activities in the model finish prior to the StopTime and there are no more items remaining, the clock simply jumps to the StopTime and the simulation ends.

14.4 Relative vs Clock Time

The **Digital Simulator** operates on “Relative Time” and “Clocks”. In Relative Time, time advances in the model using a delay value. The delay value has a base of seconds. If the model is clock-driven, there are inherent clock cycles and clock edges. This means that every block in the model is synchronized to a clock and fires at the clock edge. Models can be a combination of clock and Relative Time.

14.5 Computing Time

One unit of time is one second and one unit of data size is one byte. One nanosecond is denoted by 1.0e-9 while, 8 bits is denoted as 1 byte. The clock generated and any math calculations can be accurate to a resolution of **1.0e-14**. So, the clock generated can be at least as accurate as the hardware clock.

Values are stored in VisualSim at a resolution of 1.0e-20. Calculations are performed at the accuracy of 1.0e-14. When you use a time value in any delay block, the expression is first

calculated with the calculation accuracy and then rounded to time resolution. If you are using multiple of a clock value for the delay, it is best to create another clock with the required value.

For example, the time resolution is $1.0\text{e-}13$. $1.0\text{e-}13$ is the resolution of the number inside the WAIT(x) statement or Delay block. x is rounded internally according to time resolution to be used by WAIT(x). The value of x in other parts of the model has the original resolution of $1.0\text{e-}14$. When x is multiplied with N, and the $N*x$ is used inside WAIT($N*x$), the result $N*x$ is rounded again according to $1.0\text{e-}13$. But in the calculation of $N*x$, the resolution is $1.0\text{e-}14$.

14.6 Clock Synchronization

The WAIT (time) and TIMEQ function in the Script support clock synchronization for an integer and long value. If the time value is a double, then the simulator is delayed by this value. If the WAIT() uses an integer, then the value is treated as a clock speed. This means that function will delay to the clock boundary. For example, if the value is 1000000. This is considered as the speed of the clock in hertz. In this case, the delay value is $(1.0/1000000)$ or $1.0\text{e-}6$ or 1 micro-second. The use of an integer or long value as the clock value in Hertz for the WAIT() or TIMEQ() delay argument causes the statement to execute at exactly the clock rate based on the simulation time. There is no need to perform any other clock synchronization.

14.7 Model Event

A Model Event is similar to a time event, except generated by the user, and can occur at any time between blocks. In other words, a model condition triggers a Model Event for synchronizing between blocks, or within clock-driven processes. The most common use is to generate clocks from a single block to multiple blocks that are waiting for a clock event. Model Events can be generated from any RegEx block using the following RegEx function:

newEvent ("MyEvent")

The argument of newEvent must evaluate to a string name. The string argument can be composed of newEvent(MyParameter + "_MyEvent") notation for events inside a hierarchical block. This function can be applied in ExpressionList, Script or Smart_Controller Blocks. The

Script and Smart_Controller blocks also support the following syntax which executes locally in the Script or Smart_Controller block.

EVENT ("MyEvent")

This notation is preferred over the **newEvent()** RegEx notation, for faster execution. In a corresponding Script or Smart_Controller block, the user needs a matching syntax as follows:

WAIT ("MyEvent")

This statement continues processing on the next statement after it receives a newEvent() or EVENT(). If the newEvent() or EVENT() is generated before the WAIT() on event, characterized as a string argument, then the WAIT() immediately goes to the next statement. Finally, if two newEvent()s or EVENT()s are generated prior to a WAIT ("MyEvent") only one will execute similar to an 'OR' of pending events. This is done to prevent the unintentional build up of events in a model that may not execute as expected.

15 Data Structures

15.1 Introduction

Data Structures are the signals or transactions that propagate from block to block along the wires between ports. Data Structures are the equivalent to a packet on a network or a collection of wires/pins on a chip or a message between methods in software. The Data Structure contains a number of fields with unique names.

Every block in a model operates and makes decisions on the content of the Data Structures. For example a Traffic Generator would create a Data Structure with certain content, at a certain time and transfer it to the next block, which could be a queue. The queue block could reorder the queue storage based on the field of this data structure called "Priority".

Data Structure fields can be manipulated by the **ExpressionList**, **Script**, and **Custom-coded** blocks. All other blocks use the field values to make decisions such as routing, execute, Read or Write operations and implement delays.

15.2 Layout

A data structure contains a number of rows and each row contains a Field name and a Value. A Data Structure in VisualSim looks like this:

Field Name	Field Value
{BLOCK	"Traffic"
DELTA	0.0
DS_NAME	"DS_Traffic"
Field1	1.0
Field2	"str"
Field3	True
ID	1
INDEX	0
TIME	1.0E-10}

All data structures comprise six (6) header rows. These are:

Field Name	Field Description	Type
BLOCK	Name of Source Block	String
DS_NAME	Name of Data Structure Template	String
DELTA	User Settable Field. This field can be used to store intermediate values that might be needed for a calculation elsewhere in the model.	Double
TIME	Simulation time when DS was created. This is a time stamp that can be used to compute the latency and/or to determine deadline triggers.	Double
INDEX	User Settable Field. This field can be used to store intermediate values that might be needed for a calculation elsewhere in the model.	Integer
ID	Unique ID of each Data Structure from each source. This is a increasing sequence number.	Integer

The remaining are user-defined fields. There can be any number of user-defined fields. The types of the user-defined fields are inferred from the initial value. The three user-defined fields from the above example are:

Data Structure Field Name	Field Type	Field Value
Field1	Double	1.0
Field2	String	str
Field3	Boolean	true

15.3 Supported Data Types

The fields of the data structure support the following data types:

Data Type	Example
Integer	20
Long	2L
Double	1.0
String	"L2_Cache"
Boolean	true or false
Data Structure	{FldA=1, Fld2=3}
Binary String (Available only in the Data Structure blocks and is being deprecated.)	4'b100 (string for the Verilog format)
Array	{1.0, 2.0} or {{1.0,2.0},{3.0,4.0}}
Matrix	[1, 2; 3, 4]
Complex	4 + 2j
Fixed Point	fix(.37665, 6, 2)
Embedded Data Structure	Data_Structure "Processor_DS" or Data_Struct "C:.VisualSim.VS_AR.VisualSim.data.Processor_DS"


15.4 Port Typing

The “**Type System**” in VisualSim follows a hierarchical pattern that determines the automatic casting in an expression. All block ports are polymorphic, meaning that their type is defined at run-time based on the arriving data token. The default polymorphic setting for the port is 'unknown' or 'general'.

Some blocks have preset port types for the input and output ports. These types cannot be modified by the user. The ports connected to these ports must be modified. The data type of the ports connected to these preset port must match the preset type. To change the port data type, right-click on the block and select “**Configure Ports**” in the context menu of the block. Common ports requiring this modification includes the “**priority**” ports of the **TimedQueue** and the “**input**” ports of the **TimeDataPlotter** blocks.

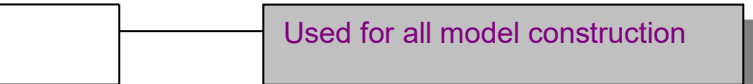
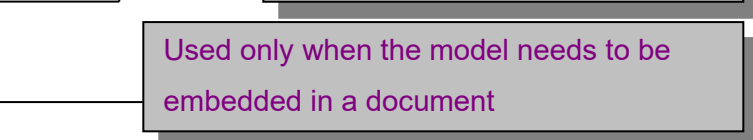
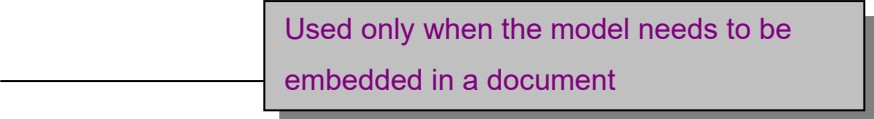
15.5 Data Structure Template Location

Text-based Data Structure can be located anywhere in the File System that can be accessed by VisualSim. If the txt file is located in <VS Home>/VisualSim/data, else the path must be provided. Java class definition of Data Structure must always be located in the <VS Home>/VisualSim/data directory.

 Note	<p>The data structures shipped with VisualSim are either located in the <VS Home>/VisualSim/data directory or in the directory containing the model.</p>
--	--

15.6 Data Structure Definitions and Example

There are three ways of constructing a Data Structure definition:

- Text or CSV format file 
- Traffic Block Window 
- Java class file 

15.6.1 Text Example

File Name: Stop_n_Wait.txt

Rules: Each row is a field. Spaces are required between columns. The line must end with a semi-colon. The advantage of the text version of the Data Structure is that it can be dynamically modified and used without recompiling.

// Name	Type	Value	Comment
Uniform_Rand_Length	int	1 ;	/* Field Comment */
Uniform_Rand_Noise	int	2 ;	/* Field Comment */
Channel	int	3 ;	/* Field Comment */
Packet_Bytes	int	4 ;	/* Field Comment */
Time_Stamp	double	5.0 ;	/* Field Comment */
Packet	Data_Struct	"Processor_DS" ;	/* Field Comment */

15.6.2 Java Example

File: Stop_n_Wait.java-

Rule: Data Structure fields are concatenated like a string.

//	Type	Name	Value	Comment
private String	Transaction_Name	= "App_0" ;	/* Transaction Name (IORead, etc.)	*/
private int	Transaction_Bytes	= 0 ;	/* Transaction Size in Bytes	*/
private String	Source_Name	= "P0" ;	/* Source Name, used in routing	*/
private int	Source_ID	= 1 ;	/* corresponds to Processor ID	*/
private String	Destination_Name	= "Bus" ;	/* Destination Name, used in routing	*/
private int	Destination_ID	= 1 ;	/* Destination Processor ID	*/
private String	DMA_Channel_Name	= "DMA_Channel" ;	/* DMA Channel Name	*/
private int	DMA_Channel_ID	= 1 ;	/* DMA Channel ID	*/



Note

The last line must end with a ";" outside the "", as this indicates the end of the string.

Save this file in the \$VS/VisualSim/data directory only.

To compile, use javac in the Java JDK bin folder

15.6.3 Finding the Data Structure Template

The blocks first look for a class definition in the VisualSim/data directory and then a text file version of the Data Structure. It accepts the first file with the matching name of the Data Structure that it finds in the CLASSPATH. The CLASSPATH is defined in the VisualSim.bat/sh file.

15.7 Path Definition

There are three ways to define the path to the Data Structure definition.

1. **Data_Structure_FileName** (Definition is in VisualSim/data directory of the install)
2. **User_Library.Data_Structure_FileName** (Definition is in <VS Home>)
3. **C:.\VisualSim.Data_Structure_FileName** (External to VisualSim- Windows)

or

User.VisualSim.Data_Structure_FileName (External to VisualSim- UNIX)

The Data Structure definition location is required for the following blocks:

- Traffic
- TriggeredTraffic
- VariableList- Initial Value
- Script Block
- ExpressionList

- RegEx function- newToken

15.8 Construction

The Data Structures are generated by the **Source** blocks (**Traffic**, **TriggeredTraffic**, **Transaction_Sequence**, and **Script**).



Note

It is recommend to define all the data structures in a single location. New fields can be created in the ExpressionList and Script blocks.

The sequence of events is as follows:

1. Create a Data Structure template as a txt or a Java file.
2. Enter the name and, optional path, in the traffic generator block.
3. Customize field values to create scenarios.
4. Use the values to make decisions or conduct operations in the model.

15.8.1 Compiling Java Data Structure

To compile the java file, run "javac VisualSim\data\MyDS.java" from the root of the VisualSim install. If javac is not recognized, make sure that the jdk/bin directory is included in the PATH.

16 Processor_DS

16.1 Introduction

The Processor_DS is the Data Structure Template used by the Hardware Modeling blocks. The following are the fields.

Field Name	Type	Value	Comment
A_Address_Min	int	1	Can store Address. This is not being used by any IP block.
A_Address_Max	int	100	Can store Address. Used in the Memory Controller for the Bank Number.
A_Addr_Ctrl_Flag	Boolean	true	Address or Control Flag; Used to distinguish between Read and Write in the Memory Controller. True is read.
A_Address	Long	0L	Address value in long data type, used by processor, cache and memory.
A_Branch	Boolean	false	Instruction Branch; Used in the processor block to determine whether the branch has been taken or not.
A_Bytes	int	8	Data size in Bytes. This is the total size of the transaction. Used in all the IP blocks for the size of the transaction.
A_Bytes_Remaining	int	4	Remaining data in Bytes when transaction is fragmented.

Field Name	Type	Value	Comment
A_Bytes_Sent	int	4	Transaction data size in Bytes for this fragment.
A_Command	String	Read	Commands are typically Read, Write, Prefetch, and so on. The user can put any value in it as described for the customer blocks.
A_D_Addr	Long	0L	Used for data cache address
A_Data	String	MyData	User Data. This is not being used by Ip block currently.
A_First_Word	boolean	true	First word for a transaction. This is used internally in bus and memory blocks. The Bus sets this flag for the first word when it sends it to the Cache/DRAM blocks. It is used in the HW_DRAM to determine whether the word is the first or not.
A_Instruction	array	{"ADD","ADD"}	Instructions. This is the list of instructions for the Processor to execute. It is also the instruction used to identify the matching task in the DMA Database.
A_Instruction_Reorder	array	{1,1}	(Optional Field) Enable or disable out-of-order execution. This determines how many newer instructions can execute before the current one must

Field Name	Type	Value	Comment
			execute. The possible values are 0 and 1 where 1 means everything must execute in order and 0 means that anyone can go ahead. There must be a one-to-one correspondence between this and the A_Instruction. Additional values of 23... are possible. 2 indicates that next two instructions can execute before the Processor has to wait for the current instruction to execute.
A_Interrupt	Boolean	false	Interrupt flag for DMA requests and Prefetch. Set by Processor or cache block; Decision made on what to return at the cache and DRAM blocks; Sets it to true when we need to get only one word back.
A_Prefetch	boolean	false	Prefetch Flag. Same as A_Interrupt but the request is for a whole line as opposed to a word. Used between the processor and cache; and cache and DRAM.
A_I_Addr	Long	0L	Used for Instruction cache address
A_IDX	int	0	Index number of the current Instruction in the A_Instruction array.
A_IDY	int	0	Last index of a multi-instruction access.

Field Name	Type	Value	Comment
A_Priority	int	0	Transaction Priority
A_Proc_Return	int	-1	Processor Return ID and used internally for decisions
A_Return	int	-1	Return ID
A_Protocol_State	String	MyState	Protocol State -- FSM Map. Not used.
A_Task_Flag	Boolean	false	Indicates a required return for a Write transaction
A_Task_Name	String	Name	Unique Task Name
A_Task_ID	long	0	Unique Task ID
A_Task_Address	int	1	Unique Task Address
A_Task_Source	String	Src	Transaction Source
A_Source	String	Src	Transaction Source
A_Hop	String	Hop	Next Intermediate Device name
A_Status	String	Status	Routing Status
A_Destination	String	Dest	Transaction Destination
A_Time	Double	0.0	Internal Timestamp
A_Variables	Int	16	Number of Software Variables

16.2 Usage of Data Structure fields in the Model

A_Address_Min – It holds the starting bus or memory address.

A_Address_Max – It holds the ending bus or memory address.

A_Address_Control_Flag - Used internally by the Linear, PCI, and AHB buses.

A_Address – Instruction, Data or Device Address, used by the hardware devices such as processor, cache and memory.

A_Branch - Instruction Branch, used internally by the processor when an instruction arrives and the branch needs to be taken. If "true", the processor pipeline is flushed.

A_Bytes - Total Number of Bytes in a transaction. This is typically the number of bytes requested from the Slave.

A_Bytes_Remaining - When partial transfers are made, the bytes remaining to be transferred are maintained in this field. This is used by the Bus to know if all the transfers are complete for a Write/Read. The Slave looks for the **A_Bytes_Remaining=0** to know that the transfer is complete.

A_Bytes_Sent - Actual number of bytes transferred in the current transaction. This is used for partial data transfer. For a Bus, this is equal to the burst size.

A_Command – It holds the Request type. The standard ones are Read, Write, Prefetch, and Erase. Others can be added by the user as required. Read, Write, and Prefetch are special keywords that are understood by the Bus and Memory for processing.

A_D_Addr – Data cache address, used internally by the processor when an instruction arrives.

A_Data - Contains the user data.

A_First_Word - This is enabled or true for the first transaction in a multi-transfer sequence. All other transfers in this sequence have false.

A_Instruction - An array of instructions that need to be executed on the Processor. The instruction names in this field must match the list in the Instruction Set of the target processor.

The Load/Store instruction does not contain the “#” in this field. The # is annotated on the Instruction_Set lookup.

A_Instruction_Reorder – (Optional field) An array corresponding to the list of instructions. The default of all "1"s means that all the instructions are executed sequentially. All "0" allows complete out-of-order execution. When an instruction completes, it sets the corresponding value to "0". If the value is "2", it means that two instructions before must have completed before this current one can execute. You can have any combinations of 0, 1,2,.....

A_Interrupt - This is linked to the A_Prefetch. If A_Prefetch is set to "true" and A_Interrupt" is set to false, this is the very first prefetch for this task. If A_Prefetch and A_Interrupt are true, it is an ongoing prefetch. For the DMA, this operates without the A_Prefetch flag. It is a good practice not to change these fields.

A_Prefetch - If A_Prefetch is set to "true" and A_Interrupt" is set to false, then it is the very first prefetch for this task. If A_Prefetch and A_Interrupt are true, it is an ongoing prefetch. It is a good practice not to change these fields. This is set to false for cache misses and retrieving one word.

A_I_Addr – Instruction cache address, used internally by the processor when an instruction arrives.

A_IDX - Current instruction index in the A_Instruction array.

A_IDY – This is used for managing multiple instruction dispatch per cycle. Specifies the last instruction index is the set that has been dispatched for the current cycle.

A_Priority - Priority of the current transaction. It is used for queue reordering and bus preemption.

A_Proc_Return - This specifies a return ID when the processor sends the request to a bus, DMA, slave or any other device. When the user goes to a “task”, this specifies where to return it. This is used by the Processor and DMA block.

A_Return - This is an index to the Pipeline on return. This tells which stage of the pipeline this task is returning too. The user should not modify this. A positive number indicates it is waiting and a negative number means it is ignored.

A_Protocol_State - This is used for the pipeline and is used internally by the processor and Linear Bus.

A_Task_Flag - This is used for the pipeline and is used internally by the processor and Linear Bus.

A_Task_Address - This is used for the pipeline and is used internally by the processor and Linear Bus.

A_Task_Source - This is used by the Bus to keep track of intermediate information. Do not change.

A_Task_Name - Unique Processor Task Name. This is an identifier at the Processor on which task is executing. This is used by the DMA to determine the sequence of operations to be performed based on a match of the Task Name and the instruction Name.

A_Task_ID - Unique Processor Task ID and each task coming into the processor should have a unique number for multiple task execution on the processor.

A_Source - Transmitting block name. If the processing is sending a read request to the Slave, the Processor name is the Source. On the return, the Slave name is the Source. Used for routing on the Bus

A_Hop - Used for routing across the Bus. This is used to determine what the intermediate node is in the transfer between devices.

A_Status - Routing Status and is maintained with the Bus Arbiter. This is not a user-editable field

A_Destination – This is used to determine where to send this data structure next. It could be Processor, bus, dma, memory or any other hardware deviceUsed for Routing, destination where the request has to be sent.

A_Time - Internal Timestamp

A_Variables - Number of Software Variables. This determines the hit or miss at the Registers and D_Cache.

16.3 Processor_DS Example

```

OUTPUT AT TIME      ----- 0.10 ns -----

{A_Addr_Ctrl_Flag    = true,
A_Address_Max        = 100,
A_Address_Min        = 1,
A_Address            = 0L,
A_Branch             = false,
A_Bytes              = 8,
A_Bytes_Remaining    = 4,
A_Bytes_Sent         = 4,
A_Command            = "Read",
A_D_Addr             = 0L,
A_Data               = "MyData",
A_Destination        = "Dest",
A_First_Word         = true,
A_Hop                = "Hop",
A_IDX                = 0,
A_IDY                = 0,
A_I_Addr             = 0L,
A_Instruction         = {"ADD", "ADD"},      // Optional field
A_Instruction_Reorder= {1},


```

```
A_Interrupt      = false,
A_Prefetch       = false,
A_Priority       = 0,
A_Proc_Return    = -1,
A_Protocol_State = "MyState",
A_Return         = -1,
A_Source         = "Src",
A_Status         = "Status",
A_Task_Address   = 1,
A_Task_Flag      = false,
A_Task_ID        = 0L,
A_Task_Name      = "Name",
A_Task_Source    = "Src",
A_Time           = 0.0,
A_Variables      = 16,
BLOCK            = "Trans_Src",
DELTA            = 0.0,
DS_NAME          = "Processor_DS",
ID               = 1,
INDEX            = 0,
TIME             = 1.0E-10}
```

17 Parameters


17.1 Introduction

Parameters are constants defined in the model window or a block. Parameters are constants and do not change during a simulation. Parameters are evaluated at the start of an execution and the value is maintained until the end of the simulation. Each block has a defined set of parameters that are associated with the operation of the block. The users can add parameters to both model windows and blocks. Parameters define attributes such as the processor speed, queue depth or simulation time, or run-time conditions such as Traffic_ON.

 Seed	Parameters are used to specify attributes such as the processor speed, queue depth or simulation time, or run-time conditions such Traffic_ON.
--	--

17.2 Parameter Values

The values of the parameters can be scalar, strings, file names, colors, expressions, and functions. All data types are supported in the parameters.

 Types	Strings must be enclosed in "Quote" marks. Integers should not have decimal points. Doubles should have decimal points. Longs should have "L" appended at end of the number. Booleans should be lowercase 'true' or 'false'.
---	--

Expression in parameters can be an arithmetic or logical operation. The parameter expressions can contain values from other parameters.

Type	Example
String	"Queue1" "file://C:/VisualSim/filename.txt"

Type	Example
Integer	1
Double	1.0
Long	123L
Boolean	true
Array	{1,2,3}
Matrix	[1,2;3,4] Note: Can contain any data type
Expression	(Parameter1==4)?Parameter2:Parameter4
Data Structure	{first=1,second="name"}

17.3 Creation

To add a new model parameter:

1. Drag-n-Drop the block from the Library Folder **Model Setup >Parameter** ('parameter=') into an open Block Diagram Editor window.
2. Right-click to select **Customize Name** of the parameter and enter a name. A new parameter name must be unique, else the BDE will generate an exception.
3. Double click the new parameter name to set the value of the parameter.

For a block, double click on the block and select **Add**. Enter the parameter name and value.

17.4 Using Parameters

Parameters can be on the Right-Hand Side of an expression and as a block parameter value.

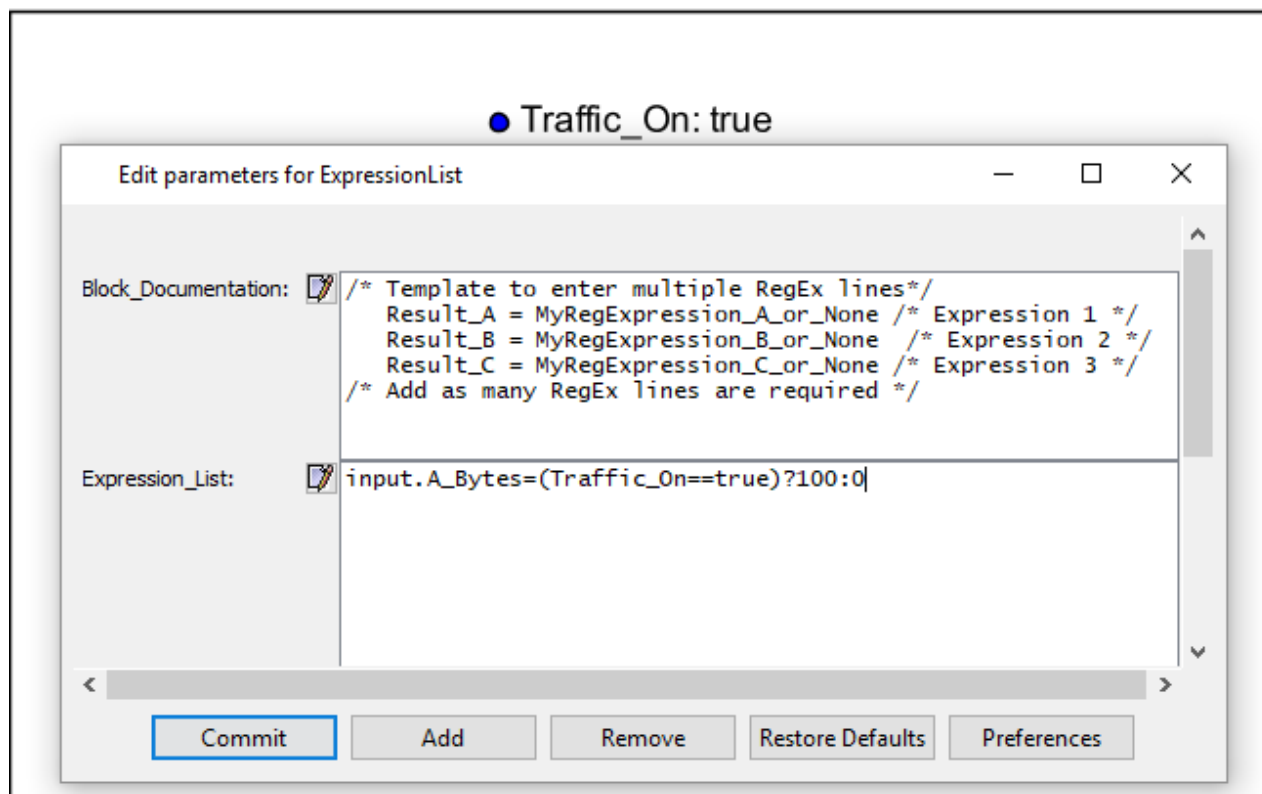


Figure 66.

Using Parameters in Expressions

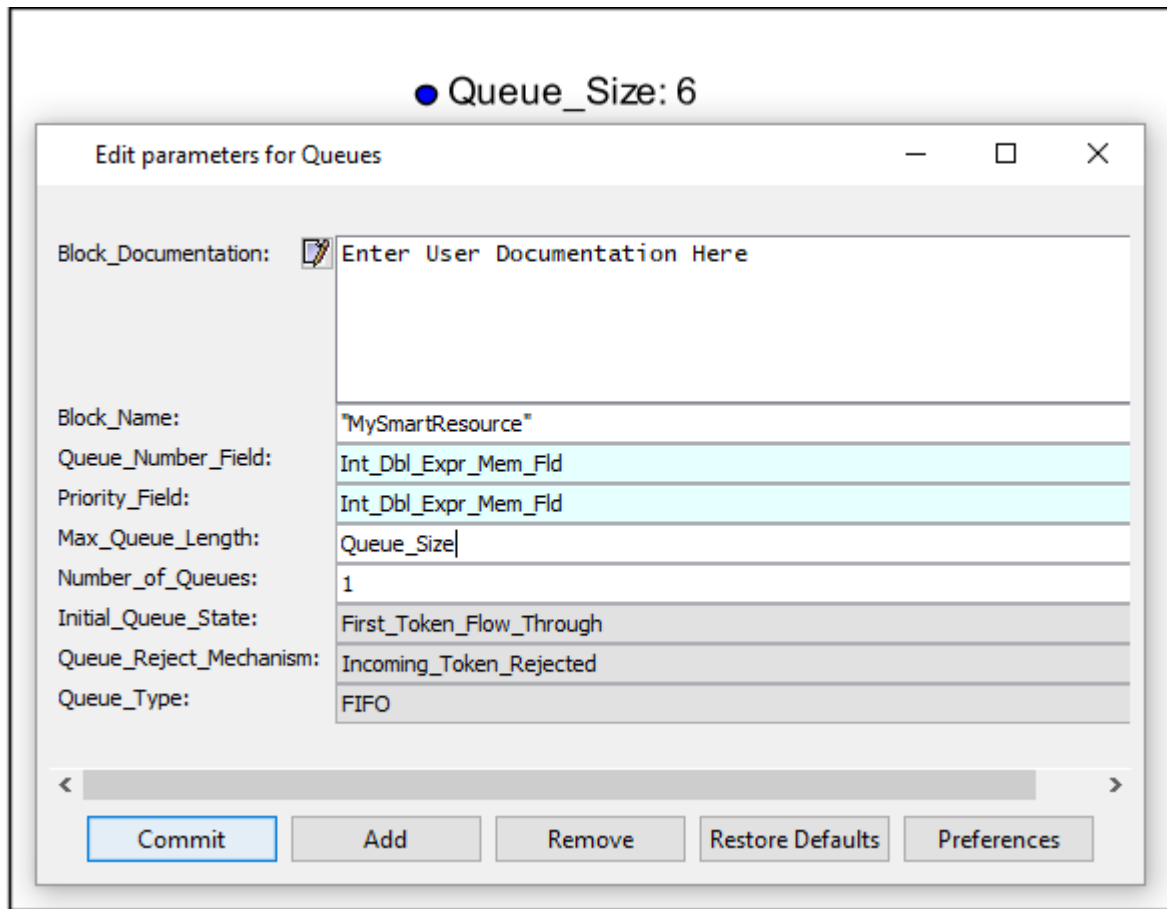


Figure 67.

Using Parameters to set Block attributes

Parameters can be concatenated with strings to form the Destination Name for Virtual Connections and the file names for file access blocks.

Parameters of a block at the top-level of the model, parameters of a hierarchical block or parameters of a block within a hierarchical block can be accessed from any other block or parameter. The format is as follows:

MyBlock::MyBlock_Param

Example: Server::Number_of_Queue

Where Server is the block name and Number_of_Queue is the parameter name. Now, if you have to access a parameter from a Hierarchical block, you provide the complete path containing all the intermediate Hierarchical block starting from the top-level.

My_HBlock::My_HBlock2::BlockName::Low_Level_Param

The BlockName will be the name appearing above the block. Parameters in a hierarchical block that are not linked to top-level parameters and parameters of a block cannot be accessed from another hierarchical level. The value of the parameter in hierarchy or parameter of a block can be accessed from the batch-mode simulation using the following format:

Hierarchical block: <window1_name.window2_name.Parameter_name>

Block: <Window1_name.block_name.Parameter_name>



Batch-Mode

When defining parameter values for the batch-mode simulation, the “ of a string value must be followed by a \”. For example, a value of “MyName” must be defined as “\”MyName\”.

17.5 Linking Parameters Up/ Down the Model Hierarchy

The value of a parameter can be set in the parameter or can reference a parameter name in the block that this parameter is contained. This can be block parameter that refers to a parameter in the window or a parameter window reference a parameter in the window that contains this block.



Value Flow

Values of parameters flow top-down and never bottom-up.

Blocks can link to top level system model parameters by entering the name of the top level parameter in the Block parameters. In the case below, the top level parameter Processor_Speed' is passed to 'Hierarchical_Block' ARM_Speed' (Value 133.0).

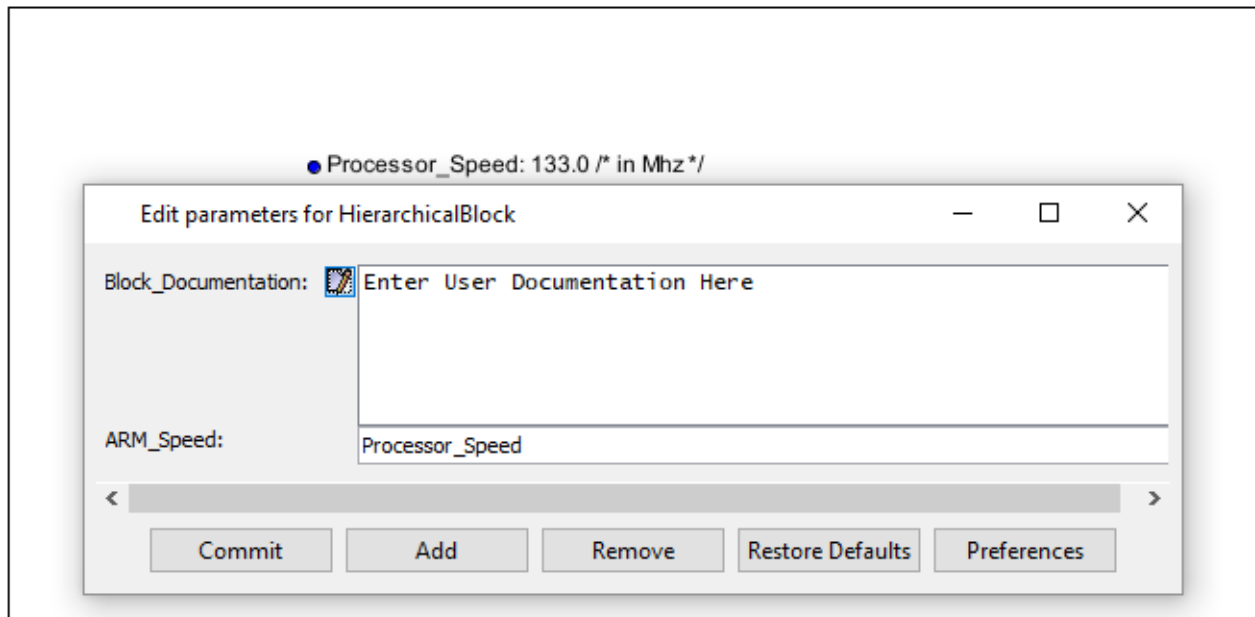


Figure 68.

Linking Parameters

17.6 Parameters as Variables

Parameters are constants and, as such, are evaluated at the start of the execution and remain the same until the end of the simulation. SetVariable block can modify the value of a parameter during the simulation. The modified value is recognized by the algorithmic blocks, Expression and, interactive display block. Traffic, Resources, Hardware, and Behavior blocks do not recognize the modified value.

17.7 Shared Parameters

Parameters can be defined at one-level of the hierarchy and reused many levels below in the hierarchy. A parameter at a lower-hierarchy can link to the top-level parameter and get the top-level value, provided there is no other parameter that matches the name in the hierarchy between the two-levels. A special case of the shared parameter is the stopTime of the Digital simulator. The user can set value at the top-level and all Digital simulators in the model adopt this value. There is also a standard parameter in the folder called SharedParameter.

17.8 Special Parameters

The "Parameter=" is the most-commonly used parameter. There are other parameter types such as FileParameter (with a browse button), StringParameter (no double-quotes required), and ColorParameter.

17.9 Parameter Value on the Icon

The Appearance > Edit Custom Icon has ParameterValue to display the value of a parameter on the block icon. The ParameterExpression can be used to combine values of multiple Parameters in an expression and the resulting value is displayed.

It is also possible to modify the color of the icon dynamically during the simulation. The fillColor parameter of the icon building attributes can be a transitional-if statement ($x==y?$ red: $(x==z)?$ blue:black). When the SetVariable block modifies the parameter value of x, the icon color is modified.

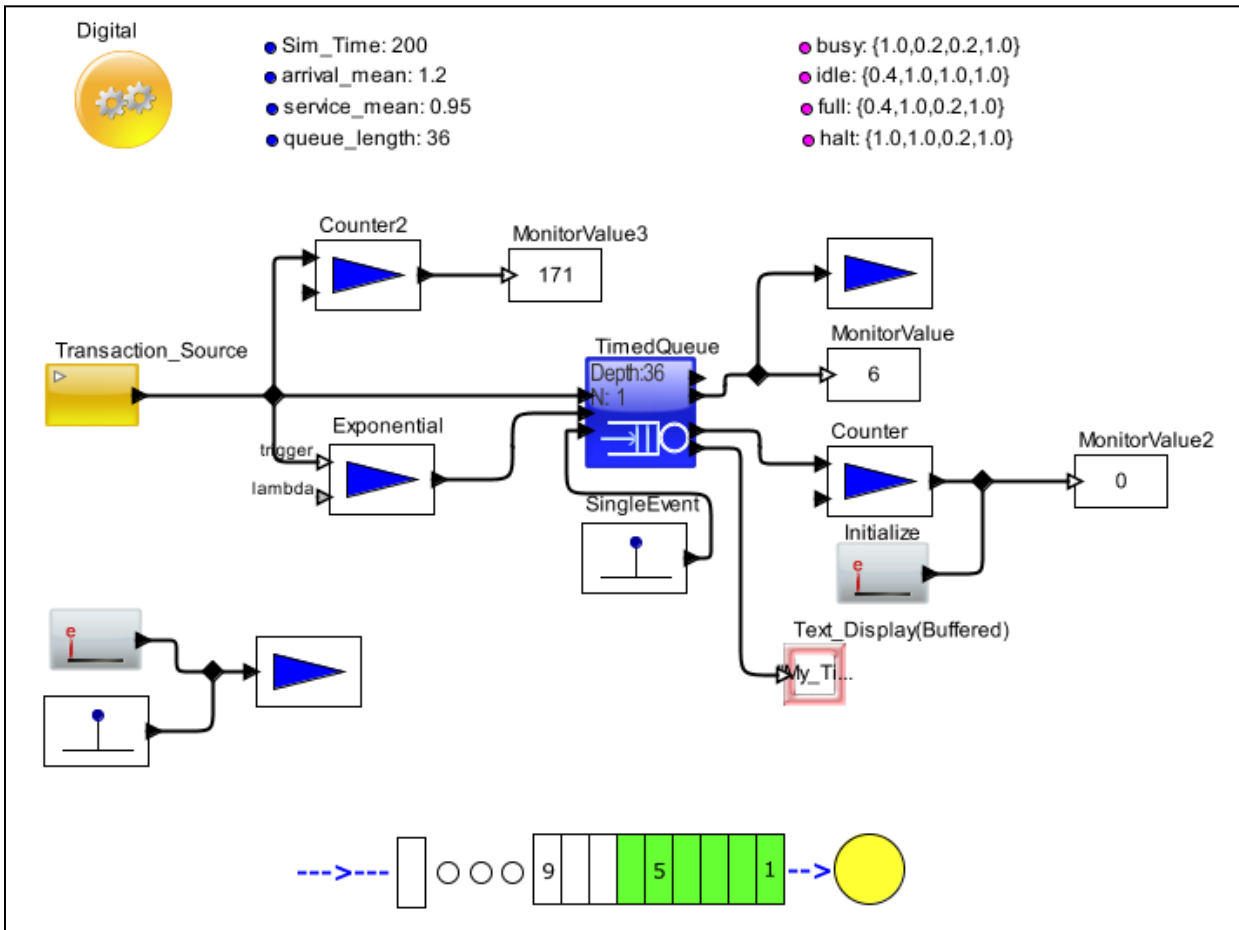


Figure 69.

Using Parameters to set the Icon color

In the example, the colors are identified by a type - idle, busy, full or halt. As the queue size changes, the icons on the lower part of the screen change color based on the new value of the Queue length.

18 Configuring Ports and Making Connections

18.1 Introduction

The Configure Ports dialog supports adding, removing, and changing key properties of ports of an entity. It displays the ports of an actor in tabular form with each row representing a port and each column a particular property of a port. An example is shown below:

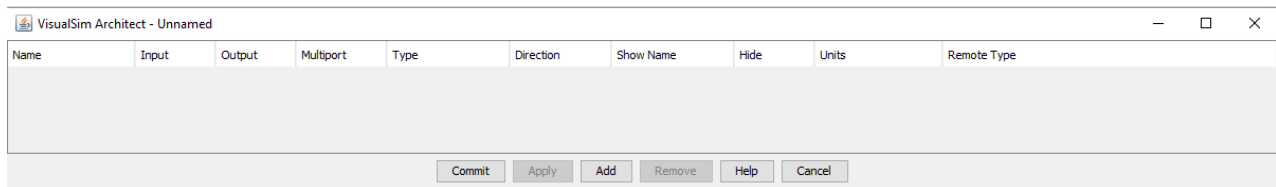


Figure 70.

Configure ports for the block

To configure the ports, right click the block and select **Customize > Ports**. Editing a cell does not immediately cause the corresponding port property to be changed. Click the **Apply** button to modify the port properties. Click **Commit** to dismiss after any table modifications are applied. Clicking **Cancel** causes any pending table modifications to be discarded and the dialog dismissed.

18.2 Editing

For the most part, editing the port properties is straight forward. Just click in the cell to be edited and an obvious interaction ensures. Some properties may not be editable because they are part of a class definition, so immutable. These cells have a red background and ignore any attempts at editing as shown below.

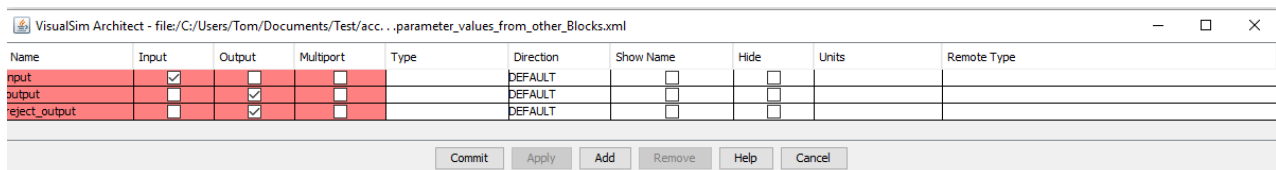


Figure 71.

Ports that cannot be edited

Name - Names must not contain a period, must not be blank, and must be unique.

Input, Output, Multiport - Checkboxes that determines if a port is input, output, or both.

Type - The type can be given by any expression that can be evaluated (this expression is called a "prototype"). For example, you can specify the type to be double by giving the string "0.0" (or any other number with a decimal point). However, this does not look very good as a designator for a type. The VisualSim expression language defines the constant "double" to equal "0.0". Thus, the preferred way to specify that a port has type double is to give its type as "double".

The primitive types include:

- Data Structure
- boolean
- complex
- double
- fixedpoint
- float
- general
- int
- long
- matrix
- object
- scalar
- string
- unknown
- unsignedByte
- xmltoken
- {int}
- [double]
- {x = double, y = double}

The constant "unknown" has a rather special behavior, in that it sets the type of the port to be unknown, allowing type resolution to infer it. The constant "matrix" designates a matrix without specifying an element type, in contrast to, for example, "[double]", which specifies a double matrix (see below). Similarly, the constant "scalar" designates a scalar of any type (double, int, long, and so on.). The constant "general" designates any type.

As the type is given by a "prototype", any data type that can be given in an expression can be specified as a type. For structured types, follow the same syntax as in expressions. For example:

```
{double} - double array
[int]     - int matrix
{field1 = string, field2 = int} - For Data Structure
```

Direction - The Cardinal Direction determines which side of the icon the port is located on. Clicking in this cell brings up a menu that provides the means to select a specific side, that is NORTH, EAST, SOUTH, and WEST. DEFAULT can also be selected. Below are the defaults:

- Input ports are usually on the West side.
- Output ports are usually on the East side.
- Ports that are both an input and an output are located on the South side.

Show Name – Enabling this option causes the name of the port to be displayed. Disabling removes the name from the display.

Hide - Enabling this option causes the port to be hidden.

Units - The units specification for the port. This column is not currently being used.

Remote Types- This column is used to define the data types on the remote system or application. This is used to define the data types for the Verilog and SystemC_Cosim only. This is a string field and must be fully contained within "". The content in this column text depends on the type of block and the interface.

18.3 Adding a port

To add a port, click **Add** in Figure 63. This adds a new row with either default, or unspecified cells. Edit the cells as required for the new port.

18.4 Removing ports

Select the row that contains the port to be removed. On a Windows platform, selecting a row is accomplished with a double-mouse click in the row. On a Mac platform, selecting a row is accomplished with a command-click in the row. When a row is selected, the name of the port is displayed in the Remove button. Click Remove to remove the port from the table.

18.5 Polymorphic Port Types

Most VisualSim blocks can accept several token types on an input port to trigger normal operation. Blocks have polymorphic input and output ports to allow the user more flexibility in implementing their design. While most of the time the queue data tokens are Data Structures, there may be instances where Int Token(s), or Double Token(s) are more appropriate. For example, the Boolean_Trigger Block can accept any data token on the input, output, or 'control' port.

The 'control' port does not care what data token enters. It is just used to control the flow of tokens from the input to output, based on the Boolean_Trigger menu attribute setting (See Full Library -> Delay and Utilities -> Switching folder):

Input_First_Next_Control

Control_First_Next_Input

Input_or_Control_First

Polymorphic port types reduce the number of library blocks needed by approximately fifty to sixty percent, while maintaining the same functionality, and a high level of simulation performance.

To understand the polymorphism system, run the model under

VS_AR\doc\Training_Material\Teaching\Type\

18.6 Ports

There are six different types of ports. For more details on Ports, refer to the section on Configuring Ports later in this Chapter.


- ⇒ The six ports are Input, Output, Multiport Input, Multiport Output, Input/Output, and Multiport Input/Output.
- ⇒ Single Input and Output ports can be selected from the toolbar. The remaining must be selected from the Menu.
- ⇒ Single ports are solid and Multiports are hollow.
- ⇒ View details below on Configuring Ports.

18.7 Relation and Link

Links are the wires between various blocks in the diagram. The relation is the Black Diamond that enables broadcast from a single port. The links simply transfer data structures and objects from one port to another. There are no attributes and parameters associated with links and relations.

- ⇒ External ports and relations can be created from the toolbar. View details below on Configuring Ports.
- ⇒ Control-click to create new relations. The relation provides the ability to connect multiple links to a single input. Alternatively you can click the Black Diamond in the standard toolbar.
- ⇒ Drag connections from one port to another. Control-drag to create a connection starting at a relation or an external port.
- ⇒ To broadcast a value from a single port, connect to a diamond relation.

18.8 Point-to-Point Block Connections

After drag-n-dropping several Blocks into the Block Diagram Editor, including hierarchical Blocks, it is easy to inter-connect the Blocks together. For a simple output port to single input port connections, one just drags from either the input, or output port, to the other port. If an output port must go to more than one input port, then one must add a "Relation" connection into the Block Diagram Editor by pushing the black diamond ICON  in the top menu bar, and the BDE will add a new relation in the middle of the open window. You then connect each input and output port(s) to the added "Relation". You can also make a connection from a "Relation" to an input or output port, simply by holding-down the "CTRL" key before dragging to the input or output port.

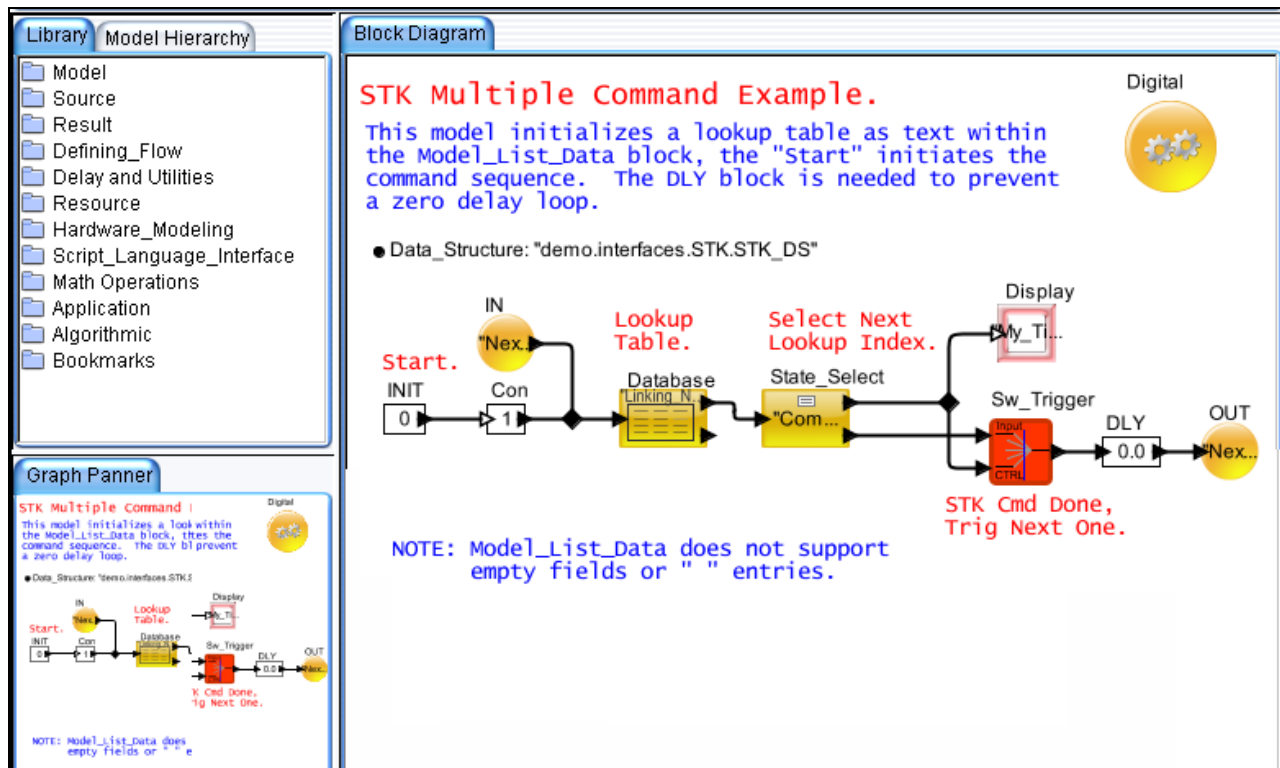


Figure 72.

Block-to-block Connections

In terms of Hierarchical blocks, one must first add the input and output ports by right-mouse click **"Open Block"** and then return to the top level to connect the ports. Finally, one can leave non-mandatory input and any output ports "open" and the simulation runs fine.

18.9 Making Connections

To connect the input (black) port to an output (black) port, select the input, hold down the left-button and drag over the output port.

To connect multiple wires, between multiple sets of input and output that meet at a point, a relation (black diamond) must be used. Connect each input and output port to the relation. It is not sufficient to have the wire go over the Relation. The wires must be connected to the relation.

In a Hierarchical block, you can connect a BDE port to a block port by holding the CTRL button and drawing the wire. Alternately, you can use the normal approach to connect the input/output port of a block to the BDE port.

When you connect a multi-port (white) of a Hierarchical block to another block, you must always connect the output port to the multi-port and then the input port to the multi-port. If this process is not followed, data may not go across the interface.

18.10 Virtual Connections

A unique connection in VisualSim is the ability to communicate between two points in the model without the use of wires. The communication can be between a Script and a IN block or an OUT to an IN block. This communication is done using destination name matching. The destination types can be local (same window) and global (full model).

18.11 Adding Parameters to Links and Ports

An annotated display and tooltip of the ports can be defined by adding a parameter called `_showInfo` and `_explanation` respectively to the port. The color and tooltip of the link can be modified by using `_color` and `_explanation` respectively. To add a parameter, right-click to "Configure". Click Add to create a parameter.

An explanation of the three parameters is given below.

- ⇒ (Block, Port and Link) The tooltip is done by adding a parameter name `_explanation`, value as a string without "" and the class `VisualSim.kernel.util.StringAttribute`.

- ⇒ (Link only) For the color, you add `_color` giving a color name for the value (string without `"`) and then making the class to be `VisualSim.kernel.util.StringAttribute`. The following is the list of color values: black, blue, cyan, darkgray, darkgrey, gray, grey, green, lightgray, lightgrey, magenta, orange, pink, red, white and yellow.
- ⇒ (Port only) A annotation can be displayed on a port by adding a parameter `_showInfo`, value as any data type and the default Class `VisualSim.data.expr.Parameter`.

18.12 Creating Generic Port Icon

Port icons are dynamically created in VisualSim during the block instantiation on the Block Diagram Editor. The icon and color for Input, Output, Input/Output, and none port types are fixed and cannot be modified by the user. For a number of reasons, the user might want to keep the look of the port different from the standard view. A generic icon of rectangular shape and green fill is provided. The port icon looks the same for input, Output, and input/output. Users can modify their port icon to this generic type by doing the following:

1. Drag a block into the BDE.
2. Right-click on the port and select "Configure".
3. Now click Add.
4. Enter the following information
 - a. Name: `_icon` ; Value: `true`; Class: `VisualSim.kernel.util.StringAttribute`
5. Click OK.
6. Click OK.
7. Now, you see that the port icon has changed to a green rectangle.

You can do the above graphical function by updating the xml file:

```
<port name="input" class="VisualSim.actor.TypedIOPort">
  <property name="input"/>
  <property name="_icon" class="VisualSim.kernel.util.StringAttribute" value="true">
  </property>
</port>
```

You need to add the property name `_icon` with the associated class and value to each port that needs to be converted to this generic icon.

19 Model Variable

19.1 Introduction

Memories in VisualSim are programming variables or hardware registers or lookup tables. They can contain any data type. Memories can be accessed using Variable blocks, Expression_List, , and Script/Smart_Controller library blocks. Variable are defined as global, local and block in VisualSim. The initialization, access, and data types are identical in all cases.

19.2 Variable Types

- **Global** - Global variable can be defined anywhere in a model using the VariableList block. These Variable locations are visible throughout the model.
- **Local** - Local variable are defined and used in the same block diagram window. They are defined in the VariableList. The main advantage of Local Variable is that multiple instances of a Hierarchical Block can be added to a model without being concerned that the different Hierarchical Block variables interact with each other. Local variables can be accessed from other model locations by using concatenating the model name and all the levels of hierarchy leading to the Variable location. The model name, level, and Variable name are separated by ".".
- **Block** - This is a special form of Variable that is defined and used entirely within a block. The Script, ExpressionList and Smart_Controller allow the creation of a Block Variable. This can be accessed within a single instance of the respective block. The DS Language-based Statement blocks have a Block Variable called **DSNow** that holds the incoming Data Structure.

19.3 Variable Data Types

Data Type	Example
Int	123
Long	123L
Double	12.3
String	"string"
Boolean	true/false
Binary string	4'b101 (Verilog format)
Data Structures	{field1=1,field2=1.0}
Arrays	newArray(10,1)

Array values can contain any of the above data types.

19.4 Variable Names

Memories can have any name. Local Variable names in different BDE can have the same names. Local and Global Variable name cannot be the same. Also, Variable names cannot be the same as the Block names for the Hardware Script, Smart_Controller, Queue or SystemResource blocks. You can define a Variable name as a concatenation of three strings or less. These strings can have parameters in them. Once defined, the Variable can be accessed on the LHS and RHS from any block by using the name. If the name is a concatenation of strings, you can create a reference to these memories by using the readVariable or writeVariable RegEx function.

19.5 Initialize Variable

The Variable must be initialized before being used. Global and Local Variable are defined in the VariableList block using the following format.

Format: `Variable_Name Variable_Type Initial_Value`

Examples are:

`Var_name local 1` (Var_name is a local Variable of type int)

`Var_name global 1.0` (Var_name is global Variable of type double)

Initial_Value formats are:

<u>Data Type</u>	<u>Example</u>
Integer	1
Long	1L
Double	1.0
String	Any string
Boolean	true or false
Binary String	4'b101 (This is being deprecated and should not be used.)
Array	{1.0, 2.0}
Array	{10:Processor_DS} (Creates an array of 10 indices with initial value Processor_DS. Processor_DS is a data structure definition in \$VS/VisualSim/data. If the file is not

<u>Data Type</u>	<u>Example</u>
	in this location, this defaults to a string "Processor_DS". If the Data Structure definition is in a different location, access it as User_Library.Processor_DS or C:.VS_Lib.Proc_DS.) Processor_DS here can also be replaced with any other data type.
Data Structure	Processor_DS or User_Library.Processor_DS or C:.VS_Lib.Proc_DS
Others	Parameters or Variable, which references a value.

Block variables must be defined in the ExpressionList or Script. It is best to define the variable in the top-section of the script for read-ability. In these blocks, the memories can be dynamically created in the code or assignment. If the variable is not initialized but used, it is essential to assign a value that can be statically mapped to a data type. Otherwise an error message is generated.

19.6 Checkers

There are two blocks and a RegEx function to check the content of the variable at any time. The two blocks are the Variable_Monitor and the Variable_Dump. The RegEx is called readAllVariable().

The Variable_Monitor block monitors the local and global variable names listed in the Trace_Name parameter. The block lists the accessing block name, current value, and variable name. The fields called Read_Variable_Name and Write_Variable_Name indicate whether a read or Write has occurred.

The Variable_Dump block outputs the current value in all the global and local variables in the model. For every input trigger of the block, the name and content are output. The output is a data structure with each field representing one of the memories. Global memories are identified by the name while the local memories have the complete hierarchical path plus name. The field names are the memory names while the value is the current content.

The readAllVariable is simply assigned as followings: Value = readAllvariable().

19.7 Accessing Local Variables

Local memories can be accessed within the same hierarchical windows. If you need to access a local variable from another hierarchical block or anywhere else in the model, the following RegEx approach must be utilized.

```
StringName = local_variable + "My_Local_Variable_Name" or
```

```
StringName = (Block_Path + "My_Local_Variable_Name").StringName.read()
```

You can do a similar operation to overwrite the content of this variable by doing the following:

```
StringName ="My_Variable_Name"
```

```
MyVariableToken = StringName.write(Token)
```

Where Token is the new value.

If you would like to access a Block Variable in a Script or Smart Controller, use the following format:

```
StringName = "My_Block_Variable_Name"
```

```
MyVariableToken = StringName.read(Block_Name)
```

Where Block_Name is the name of the Script block. It is not recommended to overwrite the value of the Block Variable from outside the block, as it is very difficult to debug.

19.8 Things to Remember

- Variables must be initialized before being accessed for the first time.
- If the variable has not been initialized before use, the RHS reference to the name is treated as a String while a LHS usage generates an error saying "Cannot write to a string on LHS".
- Variable names are Case-Sensitive.
- Variable names must be Strings and cannot be Parameters.

- A parameter on the LHS generates an exception, as a parameter cannot be changed during a simulation.
- Names of Global and Local variables cannot be the same. Local names can be the same provided they are not in the same BDE.
- The current content in all variable locations can be accessed using the **Variable_Dump** blocks available in Full Library /Model/ Utility/ Checkers library folder. You can also use the RegEx function, `readAllVariables()` to access the list of all the local and global variables in the model with the full path for the local variable. . In addition, users can use RegEx functions to monitor variable, or Script execution for specific simulation times.

19.9 Pointer / Reference to a Variable

In the Script, Smart_Controller and ExpressionList blocks, you can create reference name to a variable (local and global). This capability is important where the name of the variable is composed of multiple strings concatenated together. This is a convenience feature that enables a concatenated name to be used in an expression without using `readVariable` every time. In these blocks,

```
Ref_Var = readVariable("GlobalVariableName") or
```

```
Ref_Var = ("GlobalVariableName").read()
```

is a reference to the `GlobalVariableName` location. If you would like to access the variable in a new variable, use

```
Ref_Var = newToken(readVariable("GlobalVariableName"))
```

Note: If you place the `GlobalVariableName` or the `Ref_Mem` on the LHS of an equation and have an assignment on the RHS, the reference is lost and the two behave like independent variables.

For example, this is correct:

```
Ref_Var = readVariable("GlobalVariableName")
```

```
Ref_Var.incr()
```

```
Other_Variable = Ref_Var + 2
```

This is incorrect and delinks the variable.

```
Ref_Var = readVariable("GlobalVariableName")
```

```
Ref_Var = Ref_Var.incr() /* Creates new variable and reference  
is lost */
```

```
GlobalVariableName = Ref_Var + 2 /* Delinks original reference  
*/
```

20 File Paths

There are a few file path methods in VisualSim. The following blocks support the URI standard of the Web. For more information on URI, click [here](#).

<u>Plotters:</u>	<u>Text</u>	<u>Content Management</u>
TimeDataPlotter	FileWriter	Database
XYPlotter	File Reader	Transaction_Sequence
Histogram	Traffic_Reader	
DS_TimeDataPlotter	TextDisplay	
DS_XYPlotter	TextDisplay	

These blocks follow the following formats:

- File name: The file is in the same directory as the model XML.
- ./File Name: The file is in the same directory as the model XML. This is the Unix format.
- ../File_Location/File name: The file is located in a different directory but in the same disk. This is the Unix format.
- VS (or \$VS) + File Path + File Name: This uses the VisualSim constant string (VS) that represents the install location of VS_AR.
- Absolute Path: This includes the file path + file name from the base of the disk.
- File:/(Absolute Path): This is the URI format. This includes the file path + file name from the base of the disk. An alternate format can also contain ///.
- \$CLASSPATH + File Path + File Name: The CLASSPATH is defined in the batch script (VisualSim.bat for Windows and VisualSim.sh for UNIX). The system searches in all the directories in the CLASSPATH list. For any block that needs to write, the file must exist.
- [http://URL](#): This format is used to find a file that is on a Web Server. This format is to access data files located on remote Web servers. This is format is used when the model must be used when the model is embedded in the html page and is viewed using VisualSim Explorer. This format is only available for reading. Write is not permitted on Web Servers.

21 RegEx Language

The RegEx is a collection of Mathematical, Logical, Statistical, and Algorithm-Specific Functions. These can be used in scripts and expressions to accelerate model construction and for fast simulation execution. The RegEx functions work with most data types and have a specific format.

21.1 Computation using RegEx

The RegEx can be used in parameter fields to compute the value at the start of the simulation based on other parameters. The RegEx can also be used in the FSM for the actions. In the Script, Smart Controller, Processor, and ExpressionList blocks, RegEx can be used in methods and on the Right Hand Side (RHS) of an equation. The equation can have any number of operators and variables and combine Data Structure Fields, Memories, Values and Parameters. A complex expression can combine the built-in functions in any order, as far as they confirm to the data types. The **Left Hand Side (LHS)** of the equation can only be a **Field** or **Variable**. There are two types of formats:

Scalar and String Data types- Integer, Double and Long

$$X = Y + Z$$

Boolean

$$X = (\text{Boolean function})?Value1:Value2$$

where X is assigned the value1 or value 2 depending on the Boolean function resolving to true and false respectively. Value1 and Value2 can be any expression containing variables, Fields, Parameters, and Values.

Arrays

1. $\text{Array}(i) = \text{Array}(j) - Y + Z$ where Y and Z can reference an Array location, parameters, variables, and values. The data types can be scalar, Boolean or string data types. The indices i and j, must be an integer type and can be a value, variable, parameter or field.

2. `Val = Array(i)(j) + Y + Z`. This is a definition for a multi-dimension array. The indices `i` and `j`, must be integer and can be a value, variable, parameter or field. The assignment of value to a particular index in a multi dimension array requires temporary variables to update the sub array. For example `A = {{1,2,1},{4,5,6}}`, to update index `(0,2)`, please follow the following flow.

- a. `Temp = A(0)`
- b. `Temp(2) = Temp(2) + Y + Z`
- c. `A(0) = Temp`

21.2 Casting and Conversions

When an operator involves two distinct types, the expression language has to make a decision about which type to use to implement the operation. If one of the two types can be converted without loss into the other, then it will be. For instance, `int` can be converted losslessly to `double`, so `1.0/2` results in `2` being first converted to `2.0`, so the result is `0.5`.



Note

Among the scalar types, ***unsignedByte*** can be converted to anything else, ***int*** can be converted to ***double***, and ***double*** can be converted to ***complex***. Note that ***long*** cannot be converted to ***double*** without loss, nor vice versa. So, an expression like `2.0/2L` yields the following error message:

Error evaluating expression "`2.0/2L`"

It is because the divide method is not supported between Double '`2.0`' and Long '`2L`' as the operation between these types cannot be resolved losslessly.

All scalar types have *limited precision* and *magnitude*. As a result of this, **Arithmetic Operations** are subject to **underflow** and **overflow**. For **Double Numbers**, **overflow** results in the corresponding **positive** or **negative infinity**. **Underflow** (that is, the precision does not suffice to represent the result) yields **zero**.

For *integer types* and *fixedpoint*, **overflow** results in **wraparound**. For instance, while the value of **MaxInt** is **2147483647**, the expression is “MaxInt + 1 yields –2147483648”. Similarly, while **MaxUnsignedByte** has value **255ub**, “MaxUnsignedByte + 1ub” has value **0ub**.



Note

However the “MaxUnsignedByte + 1” yields 256, which is an **int** and not an **unsignedByte**. This is because **MaxUnsignedByte** can be losslessly converted to an **int**, so the addition is **int addition**, not **unsignedByte addition**. Note that these relational operators check the values when possible, irrespective of type.

So, for example,

```
>> 1 == 1.0
returns true
```

To check for equality of both type and value, use the equals() method, as in

```
>> 1.equals(1.0)
false
```

21.3 RegEx Functions- Queue and SystemResource; Methods

21.3.1 Mapping Functions to Blocks for getBlockStatus

VisualSim provides a number of special functions for accessing the content of Queues. The following functions are used:

- **getBlockStatus**: This is used to get a variety of information from the block. Below are some usage examples. There are three variations of these functions - 2, 3, and 4 variables. The function returns the value to the LHS after performing certain operations on the selected Queue. The queue number starts from 1 and the positions in each queue also start from 1. Blocks Supported: SystemResource_Extend, SystemResource, Layer_Protocol, Node_Master, Queue, Server, Channel_N, Script, Smart_Controller, DRAM, Processor, Linear Bus, Cache, and DMA.

- **QUEUE:** This is used to define, put, pop, and do processing in a Queue defined entirely with a Script. The queue number starts from 1 and the positions in each queue also start from 1.
Blocks Supported: Script
- **TIMEQ:** This is used to define, put, pop time, and do processing in a Timed Queue defined entirely with a Script. The queue number starts from 1 and the positions in each queue also start from 1.
Blocks Supported: Script
- **getDeviceStatus:** This function returns the status (either free or currently active) for a specific device. The free means that there is space available in the input Queue.
Blocks Supported: DRAM, Processor, Linear Bus, Cache and DMA
- **getResourceLength:** This functions gets the length of the queue. The queue number start from 1 and the positions in each queue also start from 1.
Blocks Supported: Queue

21.3.2 getBlockStatus Functions

<u>Operation</u>	<u>Explanation</u>
copy	Returns the copy of the data structure at the head of queue .
take	Removes data structure at the position without updating the statistics or placing it on the output port. Note: In the Script block, the QUEUE and TIMEQ methods make these data structures available to the next statement line.
pop	Removes the head of the Queue and places it on the output port. Updates statistics Note: In the Script block, the QUEUE and TIMEQ methods make these data structures available to the next statement line.
stats	Output the statistics for the queue or set of queues. ➔-(Queue Number) set the statistics and (Queue Number) reset the statistics ➔For all the Queues, Queue Number= Number Of Queues +1 -(Queue Number) set the stastistics and (QueueNumber) reset the statistics.
length	Current length of the queue.
array	If current length > 0, then true, else false. Position 0 is always false as the Queue number start from 1.

<u>Operation</u>	<u>Explanation</u>
copy.fieldname	Returns the value in the field (fieldname) of all the data structure in the selected Queue

21.3.3 **getBlockStatus format and example**

x= getBlockStatus("Block_Name", "Queue Name", Type, Queue Number, Position in Queue)

Type is copy, take, pop, stats, length, array and copy.fieldname where fieldname is a name of the field in the Data Structure.

Return values for each type are:

- copy: data structure
- take: data structure. Statistics are not updated for this data structure.
- pop: data structure
- Stats: data structure or array of data structure, if all Queue requested
- Length: integer value
- Array: Boolean array
- copy.fieldname: Array with any data type

Queue Name is only applicable for queues in the Script and Smart Controller.

Queue Number and Position in Queue are only applicable for Queue blocks, Server and Channel_N. First Queue is 1 and the First position in the Queue is 1.

Block/ Function	Function Details
SystemResource_Extend SystemResource	getBlockStatus ("SystemResource Name", "Any value", Type, Any value, Any value) Type can be "length and "stats". Does not require Queue number or Position
Queue	getBlockStatus("Queue_Name", "Any value", Type, Queue Number, Position in Queue) Type: Copy - Requires Queue Number, Position in Queue

Block/ Function	Function Details
	<p>take- Requires Queue Number, Position in Queue</p> <p>pop- Requires Queue Number, Position in Queue</p> <p>stats- Requires Queue Number, Position in Queue is ignored</p> <p>length- Requires Queue Number, Position in Queue is ignored</p> <p>array- Does not require Queue number or Position</p> <p>Copy.fieldName- Require Queue number, Position in Queue is ignored</p>
Server	<p>getBlockStatus("Server_Name", "Any value", Type, Queue Number, Position in Queue)</p> <p>Type:</p> <p>copy: Requires Queue Number, Position in Queue</p> <p>stats: Requires Queue Number, Position in Queue is ignored</p> <p>length: Requires Queue Number, Position in Queue is ignored</p> <p>array: Does not require Queue number or Position</p> <p>Copy.fieldName: Require Queue number, Position in Queue is ignored</p>
Channel_N	<p>getBlockStatus("Channel_Name", "Any value", Type, Channel Number, Position in Channel)</p> <p>Type:</p> <p>copy: Requires Channel Number, Position in Channel</p> <p>stats: Requires Channel Number, Position in Channel is ignored</p> <p>length: Requires Channel Number, Position in Channel is ignored</p> <p>array: Does not require Channel Number or Position</p>
Layer_Protocol	<p>getBlockStatus("Layer_Protocol_Name", "Any value", Type, Index, Position in Queue)</p> <p>Type:</p> <p>stats: To get statistics, use 1 and for reset use -1 in Index. Position in Queue is ignored</p> <p>length: To get length, Up Queue use 1 and for Down Queue use 2 in Index field. Position in Channel is ignored</p>

Block/ Function	Function Details
Script, Smart_Controller (Input_Queue)	getBlockStatus("Script_Name", "Input_Queue", Type, Any value, Any value) Type: length: Does not require Queue Number or Position in Queue

Any value means that the user enters a value but the value is ignored.

21.4 Math Functions

VisualSim provides inbuilt mathematical functions for the users to perform various calculations on the algorithm design of a system. It helps the user to implement the algorithm quickly by avoiding the logic realization using script coding.

Function/RegEx	Explanation	Example
Abs double/int/long/complex Return Type: double/ int/long (complex returns double)	Absolute value complex case:	$x = \text{abs}(-2.0)$ $= 2.0$
Angle complex Return Type: double in the range $[-\pi, \pi]$	angle / argument of the complex number:	$x = \text{angle}(2.0+1.0i)$ $= 0.46$
ceil double Return Type: double	Ceiling function, which returns the smallest (closest to negative infinity) double value that is not less than the argument and is an integer.	$x = \text{ceil}(1.7)$ $= 2.0$
compare double, double Return Type: int	Compare two numbers, returning -1 if the first argument is less than; 0, if the first argument is equal to; 1 if the first argument is greater than the second;	$x = \text{compare}(1.5, 2.0)$ $= -1$

conjugate complex Return Type: complex	Complex conjugate	$x = \text{conjugate}(2.0+1.0i)$ $= 2.0 - 1.0i$
Token.equivalent (Token Argument)	Return true if the argument is equivalent value of the Token, else false. The Token and Argument can be of different types. Supported types are int, long, double, or string.	$x = \text{2.equivalent}(2.0)$ true $x = \text{2.equivalent}(2L)$ true $x = \text{2.equivalent}("2")$ true
exp double / complex Return Type: double in the range[0.0, infinity] / complex	Exponential function (e^{argument}) complex case:	$x = \text{exp}(-1)$ $= 0.367$
floor double Return Type: double	Floor function, which is the largest (closest to positive infinity) value not greater than the argument that is an integer.	$x = \text{floor}(2.4)$ $= 2.0$
gaussian double, double / double, double, int, / double, double, int, int Return Type: double / {double} / [double]	One or more Gaussian random variables with the specified mean and standard deviation.	$x = \text{gaussian}(4.0,2.5)$ $= 7.6282997399571$
imag complex Return Type: double	Imaginary part	$x = \text{imag}(1.0+2.0i)$ $= 2.0$
isInfinite double Return Type: boolean	Return true if the argument is infinite	$x = \text{isInfinite}(2.0)$ $= \text{false}$
isNaN double Return Type: boolean	Return true if the argument is "not a number"	$x = \text{isNaN}(1.0)$ $= \text{false}$

log double / complex Return Type: double / complex	Natural logarithm complex case:	$x = \log(1.5)$ $= 0.405$
log10 double Return Type: double	Log base 10	$x = \log_{10}(1.5)$ $= 0.176$
log2 double Return Type: double	Log base 2	$x = \log_2(1.5)$ $= 0.584$
max double, double/int, int/long, long/unsignedByte, unsignedByte/{double}/ {int}/{long}/{unsignedByte} Return Type: double / int / long / unsignedByte	Maximum	$a = \{2L, 3L, 1L, 5L, 2L, 4L\}$ $x = \max(a)$ $= 5L$
min double, double/int, int/ long, long/unsignedByte, unsignedByte/{double}/ {int}/{long}/ {unsignedByte} Return Type: double/int/ long/unsignedByte	Minimum	$a = \{2.0, 3.0, 1.0, 5.0, 4.0\}$ $x = \min(a)$ $= 1.0$
pow double, double/complex, complex Return Type: double/complex	First argument to the power of the second	$x = \text{pow}(3, 2)$ $= 9.0$
random no arguments/int/int, int Return Type: double/ {double}/[double]	One or more random numbers between 0.0 and 1.0	$x = \text{random}()$ $= 0.4649$ $x = \text{random}(1)$ $= \{0.646\}$
real complex Return Type: double	Real part	$x = \text{real}(3.0 + 5.0i)$ $= 3.0$

remainder double, double Return Type: double	Remainder after division, according to the IEEE 754 floating-point standard	$x = \text{remainder}(9,2)$ $= 1.0$
round double Return Type: long	Round to the nearest long, choosing the next greater integer when exactly in between, and throwing an exception if out of range. If the argument is NaN, the result is 0L. If the argument is out of range, the result is either MaxLong / MinLong, depending on the sign.	$x = \text{round}(1.6)$ $= 2L$
roundToInt double Return Type: int	Round to the nearest int, choosing the next greater integer when exactly in between, and throwing an exception if out of range. If the argument is NaN, the result is 0. If the argument is out of range, the result is either MaxInt / MinInt, depending on the sign.	$x = \text{roundToInt}(1.4)$ $= 1$
sgn double Return Type: int	-1 if the argument is negative, 1 otherwise	$x = \text{sgn}(-9)$ $= -1$ $x = \text{sgn}(9)$ $= 1$
sqrt double/complex Return Type: double/complex	Square root. If the argument is double with value less than zero, then	$x = \text{sqrt}(9)$ $= 3$

	the result is NaN. complex case:	
toDegrees double Return Type: double	Convert radians to degrees	$x = \text{toDegrees}(3.0)$ $= 171.88$
toRadians double Return Type: double	Convert degrees to radians	$x = \text{toRadians}(1)$ $= 0.0174$

22 Accelerating Model Simulation Performance

Models in VisualSim can be developed in three ways:

1. You can make the model very verbose. This means that you build the model with many blocks. This allows the user to understand the flows, debugging based on functionality and also for transferring a specification to other team members. This model can have many real-time viewers. This model is not expected to be very fast and speed is not a consideration here. This model is also very quick to develop because all the library blocks are available.
2. Simulation depends greatly on the number of blocks in a model. You can reduce the number of blocks by creating the functionality using Script. These models take much longer to develop because every line has to be analyzed with the profiler to get maximum performance out of the design.
3. Another way to build the model is to use Java or C++ code. Java code is preferred because all the same functions from the GUI are available from the code. C++ is preferred for areas where there is no timing, just functionality.

Below are recommendations to accelerate an existing model.

1. Minimize the number of simulation time events, which can take the form of scheduled events in schedulers or Script block WAIT() or TIMEQ() functions. This has the biggest impact on performance.
2. Minimize the number of blocks, which minimizes the number of port to port events. These events are approximately 60% faster than simulation time events.
3. Reduce the size of the Data Structure by eliminating non-required fields. The number of fields have a direct correlation to simulation. When the data structure is modified or routed to multiple destinations they need to be modified. This has to be done for every transaction and will slowdown the simulation. This problem is very specific to the "ExpressionList", and Script blocks.
4. In the case of processor designs, storing the large instruction array (data structure field 500 to 5000) in a global variable reduces duplication of large arrays. This is faster than appending to the main data structure.

5. Minimize the number of ports added to key blocks. Internally, the simulator must search each port multiple times to determine which ports have tokens.
6. If one knows that the fields have not changed in the Data Structure being passed through the model, then one can add a parameter to the Script called `Duplicate_Input`, and set it to false. This does not duplicate the incoming Data Structure and improves performance.
7. If one knows the fields have not changed in the Data Structure being passed through the model, then one can add an argument to the Script `SEND` (`output`, `MyToken`, `no_dup`) function, and it does not duplicate the outgoing Data Structure.
8. In the Script, minimizing the number of statements improves overall performance. Also, the type of statement has an impact. For example, Boolean decisions are faster than statements.
9. In the Script, use of events to trigger the next activity or to initiate an acknowledgment. This is faster than input port processing of a block. If the event arrives via `EVENT()` before executing `WAIT("MyEvent")` or `TIMEQ("MyEvent")` executes, this results in 10X faster performance for this function alone.
10. User can add `Profile_File` parameter to the Script to see exactly how long a statement takes to execute to further refine the logic, flow. One can see the average time for each instruction, and the number of times it was executed. This can also be used for code coverage investigations.
11. In the Script, consider use of the `CLOCK()` event, which is like an `EVENT()`, except the event is only issued if the referenced `WAIT()` or `TIMEQ()` has a data structure waiting. Think of this as issuing a clock only if the register equivalent `WAIT()` or `TIMEQ()` has some data to be processed. It is a selective event, and improves performance as it does not process non-existing data.

23 Pause and Resume function with Debugging facility

VisualSim Architect has a debugging and step simulation facility that allows users to specify time stamps to pause simulation and enables the user to save the simulation data and restart from that time stamp even after closing the tool. This feature enables the user to save the simulation data every day or every time they close the software when they have to run large simulation models for days. This eliminates the requirement of running simulation continuously till the simulation end point is reached.

The pause and resume functions can also be used for debugging facility as well. User can pause for a given timestamp and analyze the system response and continue simulation step by step. The system can be analyzed for required functionality and also helps the designer to identify if the crucial tasks are being executed in the given timing deadlines.

23.1 How to Use Pause and Resume Function

Current version supports models with SystemResource, Servers, Queues, and Script.

Make sure you have the file PauseSimulationAt.txt in the same directory, this file must have the timestamps for pausing simulation, example 10e-6.

23.2 Steps:

1. Run Simulation by clicking the green arrow button in tool bar or press CTRL+R.
2. Pause Simulation by clicking the Pause button in tool bar
3. Click on Save As.
4. Select the check box "Save Simulation Data".
5. Give Name for the model with .xml extension.
6. Click on Ok.
7. .bin file is created in the same directory and this file has the Simulation data.
8. Close VisualSim Architect.
9. Start VisualSim Architect.
10. Open the .xml file saved in the previous step.
11. Model starts running from the time stamp where it was stopped. The Model runs till it reaches the next time stamp for pause.

24 Data Types

24.1 Scalar and Numerical Values

- Numerical values without decimal points, such as "10" or "-3" are integers of type *int*.
- Numerical values with decimal points, such as "10.0" or "3.14159" are of type *double*.
- Numerical values without decimal points followed by the character "l" (el) or "L" are of type *long*.
- Unsigned integers followed by "µb" or "UB" are of type *unsignedByte*, as in "5µb". An *unsignedByte* has a value between 0 and 255.
- Numbers of type *int*, *long*, or *unsignedByte* can be specified in decimal, octal or hexadecimal. Numbers beginning with a leading "0" are octal numbers. Numbers beginning with a leading "0x" is hexadecimal numbers. For example, "012" and "0xA" are both equal to the integer 10.
- A *complex* is defined by appending an "i" or a "j" to a double for the imaginary part. This gives a purely imaginary complex number which can then leverage the polymorphic operations in the Token classes to create a general complex number. Thus "2 + 3i" results in the expected complex number. You can optionally write this as "2 + 3*i".

Scalar values use a 64 bit with 54 bits mantissa, 1 bit mantissa sign, 8 bits exponent and 1 bit exp sign. All numerical values are stored as tokens which contain the values, in the above format, and the type. All math operations are done with full resolution. Basic math functions are built into the token for simulation speed. For all other basic functions, Java math and Java extended math are used. All other functions are custom built in the RegEx language.

24.2 String Constants

Anything between double quotes, "...", is interpreted as a string constant. The built-in string-valued constants are shown in the RegEx section of this document.

24.3 Arrays

Arrays are specified with curly brackets, for example, "{1, 2, 3}" is an array of *integer*, while "{x, y, z}" is an array of *string*. The types are denoted "{int}" and "{string}" respectively. "{1, 2, 3}" is a single-dimension array while "{{1, 2, 3},{4,5,6}}" is a multi-dimension array. An array is an ordered list of tokens of any type, with the only constraint being that the elements all have the same type. If an array is given with mixed types, the expression evaluator attempts to losslessly convert the elements to a common type. Thus, for example,

```
{1, 2.3}
```

has value

```
{1.0, 2.3}
```

Its type is {double}. The elements of the array can be given by expressions, as in the example "{2*pi, 3*pi}". The elements of an array can be accessed as follows:

```
>> {1.0, 2.3} (1)
```

```
2.3
```

which yields 2.3.



Note

Note that indexing begins at 0. All array operations are listed in the RegEx section.

An array can be accessed by the name and index as follows:

Single-dimension:

```
>> x = {1.0, 2.3}
```

```
{1.0, 2.3}
```

```
>> x(0)
```

```
1.0
```

Multi-dimension:

```
>> a = {{1,2,3},{4,5,6}}
```

```
{{1, 2, 3}
```

```
, {4, 5, 6}
```

```
}
```

```
>> a(0)(0)
```

```
1
```

Arithmetic operations on arrays are carried out element-by-element, as shown by the following examples:

```
>> {1, 2}*{2, 2}
```

```
{2, 4}
```

```
>> {1, 2}+{2, 2}
```

```
{3, 4}
```

```
>> {1, 2}-{2, 2}
```

```
{-1, 0}
```

```
>> {1, 2}^2
```

```
{1, 4}
```

```
>> {1, 2}%{2, 2}
```

```
{1, 0}
```

An array can be checked for equality with another array as follows:

```
>> {1, 2}=={2, 2}
```

```
false
```

```
>> {1, 2}!={2, 2}
```

```
true
```

For other comparisons of arrays, use `compare()` function. As with scalars, testing for equality using the `==` or `!=` operators tests the values, independent of type. For example,

```
>> {1, 2}=={1.0, 2.0}

true
```

You can extract a subarray by invoking `subarray()` method as follows:

```
>> {1, 2, 3, 4}.subarray(2, 2)

{3, 4}
```

The first argument is the starting index of the subarray and the second argument is the length.

You can also extract non-contiguous elements from an array using `extract()` method. This method has two forms:

1. The first form takes a boolean array of the same length as the original array which indicates which elements to extract, as in the following example:

```
>> {"red","green","blue"}.extract({true,false,true})

{"red", "blue"}
```

2. The second form takes an array of integers giving the indices to extract, as in the following example:

```
>> {"red","green","blue"}.extract({2,0,1,1})

{"blue", "red", "green", "green"}
```

To add an additional item to the array, you use the `append` function.

There are a few variations also available.

Single Array

```
>> a = {1,2,3}

{1, 2, 3}

>> a = a.append(4)

{1, 2, 3, 4}
```

Multi-dimension Array

```
>> a = {{1,2,3},{4,5,6}}

{{1, 2, 3}
, {4, 5, 6}
}

>> a = append(a,7)

{{1, 2, 3, 7}
, {4, 5, 6, 7}
}

>> a = a.append({7,8,9,10})

{{1, 2, 3, 7}
, {4, 5, 6, 7}
, {7, 8, 9, 10}
}
```

To search the array, you can use the `find` (returns an index) or `search` (list of values).

```
>> a = {true,false,false}

{true, false, false}

>> b = find(a)

{0}

>> a = {1,2,3,4}

{1, 2, 3, 4}

>> x = find(a,2)

{1}
```

```
>> b = {{1,2,3},{4,5,6}}

{{1, 2, 3}
, {4, 5, 6}
}

>> x = find(b(1),4)

{0}

>> a

{1, 2, 3, 4}

>> X=search(a, 3,1)

2
```

24.4 Matrices

In VisualSim, arrays are ordered as sets of tokens. VisualSim also supports matrices, which are more specialized than arrays. They contain only certain primitive types, currently Boolean, complex, double, fixedpoint, int, and long. UnsignedByte matrices are not supported. Matrices cannot contain arbitrary tokens, so they cannot, for example, contain matrices. They are intended for data intensive computations.

Matrices are specified with square brackets, using commas to separate row elements and semicolons to separate rows. For example, "[1, 2, 3; 4, 5, 5+1]" gives a two by three integer matrix (2 rows and 3 columns). Note that an array or matrix element can be given by an expression. A row vector can be given as "[1, 2, 3]" and a column vector as "[1; 2; 3]". Some MATLAB-style array constructors are supported. For example, "[1:2:9]" gives an array of odd numbers from 1 to 9, and is equivalent to "[1, 3, 5, 7, 9]". Similarly, "[1:2:9; 2:2:10]" is equivalent to "[1, 3, 5, 7, 9; 2, 4, 6, 8, 10]". In the syntax "[p:q:r]", p is the first element, q is the step between elements, and r is an upper bound on the last element. That is, the matrix does not contain an element larger than r. If a matrix with mixed types is specified, then the elements are converted to a common type, if possible. Thus, for example, "[1.0, 1]" is equivalent to "[1.0, 1.0]", but "[1.0,

1L]" is illegal (because there is no common type to which both elements can be converted losslessly).

Reference to elements of matrices have the form "matrix(n, m)" or "name(n, m)" where name is the name of a matrix variable in scope, n is the row index, and m is the column index. Index numbers start with zero, as in Java, not 1, as in MATLAB. For example,

```
>> [1, 2; 3, 4](0,0)

1

>> a = [1, 2; 3, 4]

[1, 2; 3, 4]

>> a(1,1)

4
```

Matrix multiplication works as expected. For example, (as seen in the expression evaluator)

```
>> [1, 2; 3, 4]*[2, 2; 2, 2]

[6, 6; 14, 14]
```

Of course, if the dimensions of the matrix do not match, then you get an error message. To do element wise multiplication, use the `multiplyElements()` function. Matrix addition and subtraction are element wise, as expected, but the division operator is not supported. Element wise division can be accomplished with the `divideElements()` function, and multiplication by a matrix inverse can be accomplished using the `inverse()` function. A matrix can be raised to an int or unsignedByte power, which is equivalent to multiplying it by itself some number of times. For instance,

```
>> [3, 0; 0, 3]^3

[27, 0; 0, 27]
```

A matrix can also be multiplied or divided by a scalar, as follows:

```
>> [3, 0; 0, 3]*3

[9, 0; 0, 9]
```


A matrix can be added to a scalar. It can also be subtracted from a scalar, or have a scalar subtracted from it. For instance,

```
>> 1-[3, 0; 0, 3]
[-2, 1; 1, -2]
```

A matrix can be checked for equality with another matrix as follows:

```
>> [3, 0; 0, 3]!=[3, 0; 0, 6]
true
>> [3, 0; 0, 3]==[3, 0; 0, 3]
true
```

For other comparisons of matrices, use the `compare()` function. As with scalars, testing for equality using the `==` or `!=` operators tests the values, independent of type. For example,

```
>> [1, 2]==[1.0, 2.0]
true
```

To get type-specific equality tests, use the `equals()` method, as in the following examples:

```
>> [1, 2].equals([1.0, 2.0])
false
>> [1.0, 2.0].equals([1.0, 2.0])
true
>>
```



Caution

ExpressionList and Script blocks do not support the assignment of values to a matrix. You cannot have an existing matrix on the LHS of an expression.

24.5 Data Structure or Transaction or Record Token

A Data Structure is a class type containing named fields and associated value. The value of each field can have a distinct type. Data Structures are delimited by curly braces, with each field given a name. For example, "{a=1, b="foo"}" is a data structure with two fields, named "a" and "b", with values 1 (an integer) and "foo" (a string), respectively. The value of a field can be an arbitrary expression, and data structures can be nested (a field of A data structure may be A data structure).

Fields may be accessed using the period operator. For example,

```
{a=1,b=2}.a
```

yields 1. You can optionally write this as if it were a method call:

```
{a=1,b=2}.a()
```

The arithmetic operators +, -, *, /, and % can be applied to data structures. If the data structures do not have identical fields, then the operator is applied only to the fields that match, and the result contains only the fields that match. Thus, for example,

```
{foodCost=40, hotelCost=100} + {foodCost=20, taxiCost=20}
```

yields the result

```
{foodCost=60}
```

You can think of an operation as a set intersection, where the operation specifies how to merge the values of the intersecting fields. You can also form an intersection without applying an operation. In this case, using the intersect() function, you form a data structure that has only the common fields of two specified data structures, with the values taken from the first data structure. For example,

```
>> intersect({a=1, c=2}, {a=3, b=4})
```

```
{a=1}
```

Data structures can be joined (think of a set union) without any operation being applied by using the merge() function. This function takes two arguments, both of which are data structures. If

the two data structures have common fields, then the field value from the first data structure is used. For example,

```
merge({a=1, b=2}, {a=3, c=3})
```

yields the result {a=1, b=2, c=3}.

Data structures can be compared, as in the following examples:

```
>> {a=1, b=2}!={a=1, b=2}
```

```
false
```

```
>> {a=1, b=2}!={a=1, c=2}
```

```
true
```

Note that two data structures are equal only if they have the same field labels and the values match. As with scalars, the values match irrespective of type. For example:

```
>> {a=1, b=2}=={a=1.0, b=2.0+0.0i}
```

```
true
```

The order of the fields is irrelevant. Hence

```
>> {a=1, b=2}=={b=2, a=1}
```

```
true
```

Moreover, data structure fields are reported in alphabetical order, irrespective of the order in which they are defined. For example,

```
>> {b=2, a=1}
```

```
{a=1, b=2}
```

To get type-specific equality tests, use the `equals()` method, as in the following examples:

```
>> {a=1, b=2}.equals({a=1.0, b=2.0+0.0i})
```

```
false
```

```
>> {a=1, b=2}.equals({b=2, a=1})
```

```
true
```

```
>>
```

24.6 Invoking Methods

Every element and sub-expression in an expression represents an instance of the **Token** class in VisualSim (or more likely, a class derived from **Token**). The expression language supports invocation of any method of a given token, as long as the arguments of the method are of type **Token** and the return type is **Token** (or a class derived from **Token**, or something that the expression parser can easily convert to a token, such as a string, double, int, and so on.). The syntax for this is `(token).methodName(args)`, where `methodName` is the name of the method and `args` is a comma-separated set of arguments. Each argument can itself be an expression. Note that the parentheses around the token are not required, but might be useful for clarity. As an example, the **Array Token** and **Data Structure** classes have a `length()` method, illustrated by the following examples:

```
{1, 2, 3}.length()
```

```
{a=1, b=2, c=3}.length()
```

each of which returns the integer 3.

The **MatrixToken** classes have three particularly useful methods, illustrated in the following examples:

```
[1, 2; 3, 4; 5, 6].getRowCount()
```

which returns 3, and

```
[1, 2; 3, 4; 5, 6].getColumnCount()
```

which returns 2, and

```
[1, 2; 3, 4; 5, 6].toArray()
```

which returns {1, 2, 3, 4, 5, 6}. The latter function can be particularly useful for creating arrays using MATLAB-style syntax. For example, to obtain an array with the integers from 1 to 100, you can enter:

```
[1:1:5].toArray()
>>{1,2,3,4,5}

[1:2:5].toArray()
>> {1, 3, 5}
```

24.7 Defining Functions

The expression language supports definition of functions. The syntax is:

```
function(arg1:Type, arg2:Type...)
    function body
```

where "function" is the keyword for defining a function. The type of an argument can be left unspecified, in which case the expression language attempts to infer it. The function body gives an expression that defines the return value of the function. The return type is always inferred based on the argument type and the expression. For example:

```
function(x:double) x*5.0
```

defines a function that takes a double argument, multiplies it by 5.0, and returns a double. The return value of the above expression is the function itself. Thus, for example, the expression evaluator yields:

```
>> function(x:double) x*5.0
    (function(x:double) (x*5.0))
>>
```

To apply the function to an argument, simply do

```
>> (function(x:double) x*5.0) (10.0)
50.0
>>
```

Alternatively, in the expression evaluator, you can assign the function to a variable, and then use the variable name to apply the function. For example,

```
>> f = function(x:double) x*5.0
      (function(x:double) (x*5.0))
>> f(10)
50.0
>>
```

Functions can be passed as arguments to certain "higher-order functions" that have been defined. For example, the `iterate()` function takes three arguments, a function, an integer, and an initial value to apply the function. It applies the function first to the initial value, then to the result of the application, and finally to that result, collecting the results into an array whose length is given by the second argument. For example, to get an array whose values are multiples of 3, try

```
>> iterate(function(x:int) x+3, 5, 0)
{0, 3, 6, 9, 12}
```

The function given as an argument simply adds three to its argument. The result is the specified initial value (0) followed by the result of applying the function once to that initial value, then twice, then three times, and so on.

Another useful higher-order function is the `map()` function. This one takes a function and an array as arguments, and simply applies the function to each element of the array to construct a result array. For example,

```
>> map(function(x:int) x+3, {0, 2, 3})
{3, 5, 6}
```

A typical use of functions in a VisualSim model is to define a parameter in a model whose value is a function. Suppose that the parameter named "f" has value "function(x:double) x*5.0". Then within the scope of that parameter, the expression "f(10.0)" yields the result 50.0.

Functions can also be passed along connections in a VisualSim model. Consider the model shown in figure.

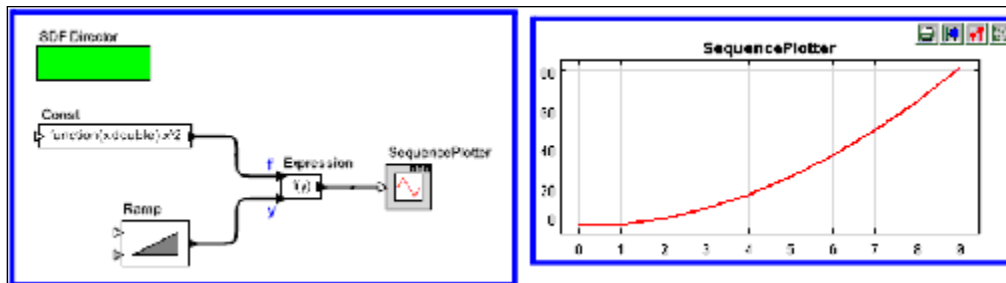


Figure 73. Example of a function being passed from one actor to another

The Const actor defines a function that simply squares the argument. Its output, therefore, is a token with type function. That token is fed to the "f" input of the Expression actor. The expression uses this function by applying it to the token provided on the "y" input. That token, in turn, is supplied by the Ramp block, so the result is the curve shown in the plot on the right.

A more elaborate use is shown in the following figure:

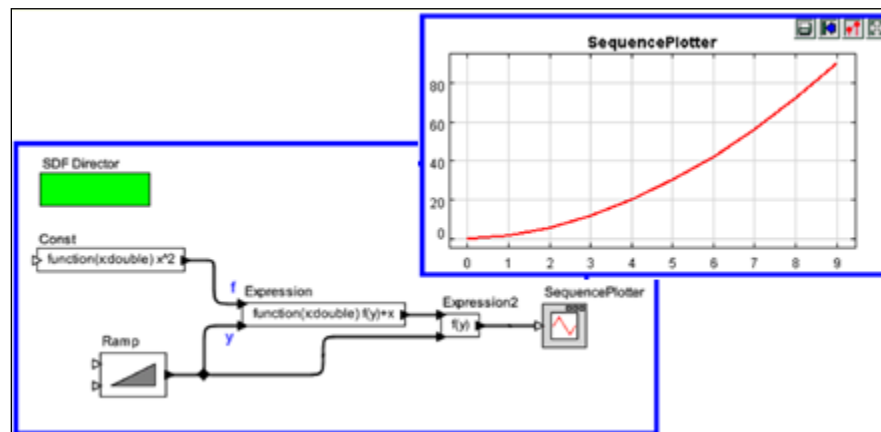


Figure 74. Elaborate use

The Const actor produces a function, which is then used by the Expression actor to create new function, which is finally used by Expression2 to perform a calculation. The calculation performed here adds the output of the Ramp to the square of the output of the Ramp.

Functions can be recursive, as illustrated by the following (rather arcane) example:

```
>> fact = function(x:int,f:(function(x,f) int)) (x<1?1:x*f(x-1,f))

(function(x:int, f:function(a0:general, a1:general) int)
(x<1)?1:(x*f((x-1), f)))

>> factorial = function(x:int) fact(x,fact)

(function(x:int) (function(x:int, f:function(a0:general,
a1:general) int) (x<1)?1:(x*f((x-1), f)))(x, (function(x:int,
f:function(a0:general, a1:general) int) (x<1)?1:(x*f((x-1), f)))))

>> map(factorial, [1:1:5].toArray())

{1, 2, 6, 24, 120}

>>
```

The first expression defines a function named "fact" that takes a function as an argument, and if the argument is greater than or equal to 1, uses that function recursively. The second expression defines a new function "factorial" using "fact." The final command applies the factorial function to an array to compute factorials.

24.8 Fixed Point Numbers

VisualSim includes a preliminary fixed point data type. We represent a fixed point value in the expression language using the following format:

```
fix(value, totalBits, integerBits)
```

Thus, a fixed point value of 5.375 that uses 8 bit precision of which 4 bits are used to represent the (signed) integer part can be represented as:

```
fix(5.375, 8, 4)
```

The value can also be a matrix of doubles. The values are rounded, yielding the nearest value representable with the specified precision. If the value to represent is out of range, then it is saturated, meaning that the maximum or minimum fixed point value is returned, depending on the sign of the specified value. For example,

```
fix(5.375, 8, 3)
```


yields 3.968758, the maximum value possible with $(8/3)$ precision.

In addition to the `fix()` function, the expression language offers a `quantize()` function. The arguments are the same as those of the `fix()` function, but the return type is a `DoubleToken` or `DoubleMatrixToken` instead of a `FixToken` or `FixMatrixToken`. This function can therefore be used to quantize double-precision values without ever explicitly working with the fixed-point representation.

To make the `FixToken` accessible within the expression language, the following functions are available:

To create a single `FixPoint Token` using the expression language:

```
fix(5.34, 10, 4)
```

This creates a `FixToken`. In this case, we try to fit the number 5.34 into a 10 bit representation with 4 bits used in the integer part. This may lead to quantization errors. By default the round quantizer is used.

To create a `Matrix with FixPoint values` using the expression language:

```
fix([ -.040609, -.001628, .17853 ], 10, 2)
```

This creates a `FixMatrixToken` with 1 row and 3 columns, in which each element is a `FixPoint` value with precision $(10/2)$. The resulting `FixMatrixToken` tries to fit each element of the given double matrix into a 10 bit representation with 2 bits used for the integer part. By default, the round quantizer is used.

To create a single `DoubleToken`, which is the quantized version of the double value given, use the expression language:

```
quantize(5.34, 10, 4)
```

This creates a `DoubleToken`. The resulting `DoubleToken` contains the double value obtained by fitting the number 5.34 into a 10 bit representation with 4 bits used in the integer part. This may lead to quantization errors. By default the round quantizer is used.

To create a `Matrix with doubles quantized to a particular precision` using the expression language:

```
quantize([ -.040609, -.001628, .17853 ], 10, 2)
```

This creates a DoubleMatrixToken with 1 row and 3 columns. The elements of the token are obtained by fitting the given matrix elements into a 10 bit representation with 2 bits used for the integer part. Instead of being a fixed point value, the values are converted back to their double representation and by default the round quantizer is used.

24.9 Special Functions

Few functions have sufficiently subtle properties that require further explanation. That explanation is here.

```
eval()
```

The built-in function eval() evaluates a string as an expression in the expression language. For example,

```
eval("[1.0, 2.0; 3.0, 4.0]")
```

returns a matrix of doubles. The following combination can be used to read parameters from a file:

```
eval(readFile("filename"))
```

where the filename can be relative to the current working directory (where VisualSim was started, as reported by the property user.dir). The user's home directory (as reported by the property user.home), or the classpath, includes the directory tree in which VisualSim is installed.



Note

If eval() is used in an expression, it is impossible for the type system to infer any more specific output type than general. If you need the output type to be more specific, then you need to cast the result of eval().

Note that if eval() is used in an expression, it is impossible for the type system to infer any more specific output type than general. If you need the output type to be more specific, then you need to cast the result of eval(). For example, to force it to type double:

```
>> cast(double, eval("pi/2"))


1.5707963267949

traceEvaluation()
```

The `traceEvaluation()` function evaluates an expression given as a string, much like `eval()`, but instead of reporting the result, reports exactly how the expression was evaluated. This can be used to debug expressions, particularly when the expression language is extended by users.

24.10 Distributions

VisualSim provides a large number of distributions as blocks and in the RegEx. The random numbers generated can be arrays or single values. There are two types of RegEx random distributions - Random and Gaussian.

 Seed	<p>To see the distribution seed, the user must have a top-level parameter called <code>model_seed</code> with value of <code>seed(integer value)</code>. The Traffic block and the Basic Processing blocks have the model seed as the parameter of each block. These blocks do not use the <code>model_seed</code> parameter.</p>
--	---

The RegEx distribution functions use a different seed for each run. To make each simulation run have the same seed, the user must have a top-level parameter called `model_seed`. The parameter value is `seed (integer value)`. The Traffic and the Basic Processing blocks have the model seed as the parameter of each block. These blocks do not use the `model_seed` parameter.

The functions `random()` and `gaussian()` return one or more random numbers. With minimum number of arguments (zero or two, respectively), they return a single number. With one additional argument, they return an array of the specified length. With a second additional argument, they return a matrix with the specified number of rows and columns.

The `irand` and `rand` have two arguments and return an integer and double value respectively. There is also a random distribution associated with the integer token which can be used to speed up the random number generation.

There is a key subtlety when using these functions in VisualSim. In particular; they are evaluated only when the expression within which they appear is evaluated. For example, if the value parameter of the Const block is set to "`random()`", then its output is a random constant, that is, it does not change on each firing. The same is true for parameter also. If the model seed is not set, the output changes on successive runs of the model. In contrast, if this is used in an ExpressionList or Script block, then each firing triggers an evaluation of the expression, and consequently results in a new random number.

24.11 `property()`

The `property()` function accesses system properties by name. Some possibly useful system properties are:

- **VisualSim.VS.dir**: The directory in which VisualSim is installed.
- **VisualSim.VS.dirAsURL**: The directory in which VisualSim is installed, but represented as an URL.
- **user.dir**: The current working directory, which is usually the directory in which the current executable was started.

24.12 `remainder()`

This function computes the remainder operation on two arguments as prescribed by the IEEE 754 standard, which is not the same as the modulo operation computed by the `%` operator. The result of `remainder(x, y)` is $(x - yn)$, where n is the integer closest to the exact value of x/y . If two integers are equally close, then n is the integer that is even. This yields results that may be surprising, as indicated by the following examples:

```
>> remainder(1,2)
1.0
```

```
>> remainder(3,2)
-1.0
Compare this to
>> 3%2
1
```

which is different in two ways. The result numerically different and is of type `int`, whereas `remainder()` always yields a result of type `double`.

If either argument is `NaN`, or the first argument is infinite, or the second argument is positive zero or negative zero, then the result is `NaN`.


If the first argument is finite and the second argument is infinite, then the result is the same as the first argument.

24.13 Power and Modulo

The `^` operator computes "to the power of" or exponentiation where the exponent can only be an *int* or an *unsignedByte*.

The *unsignedByte*, *int* and *long* types can only represent integer numbers. Operations on these types are integer operations, which can sometimes lead to unexpected results. For instance, `1/2` yields 0 if 1 and 2 are integers, whereas `1.0/2.0` yields 0.5. The exponentiation operator `'^'` when used with negative exponents can similarly yield unexpected results. For example, `2^-1` is 0 because the result is computed as `1/(2^1)`.

The `%` operation is a **Modulo** or **Remainder Operation**. The result is the remainder after division. The sign of the result is the same as that of the dividend (the left argument).

 <p>Example</p>	<pre>>> 3.0 % 2.0 1.0 >> -3.0 % 2.0 -1.0 >> -3.0 % -2.0</pre>
---	---

	-1.0
	>> 3.0 % -2.0
	1.0

The magnitude of the result is always less than the magnitude of the divisor (the right argument). Note that when this operator is used on doubles, the result is not the same as that produced by the remainder() function. For instance,

```
>> remainder(-3.0, 2.0)
```

```
1.0
```

The remainder() function calculates the IEEE 754 standard remainder operation. It uses a rounding division rather than a truncating division, and hence the sign can be **positive** or **negative**, depending on complicated rules. For example, counter intuitively,

```
>> remainder(3.0, 2.0)
```

```
-1.0
```

25 Modeling Languages

VisualSim offers a number of methods to construct the model. The fastest way is to use the blocks. Blocks are the components, probes and simulators used to assemble a model and instrument the model for generating analysis output. The blocks provide the required modeling concepts required to describe the system, generate traffic, generate statistics, and make decisions. The Resources, Traffic and Statistics Generation are best built using blocks. The behavior, algorithms and arbitration schemes can be described using ExpressionList blocks, Script, or a commercial programming language such as Python, C, C++, SystemC and Verilog. For the ExpressionList blocks and the Script, VisualSim provides an extensive collection of mathematical, logical and custom functions called RegEx.

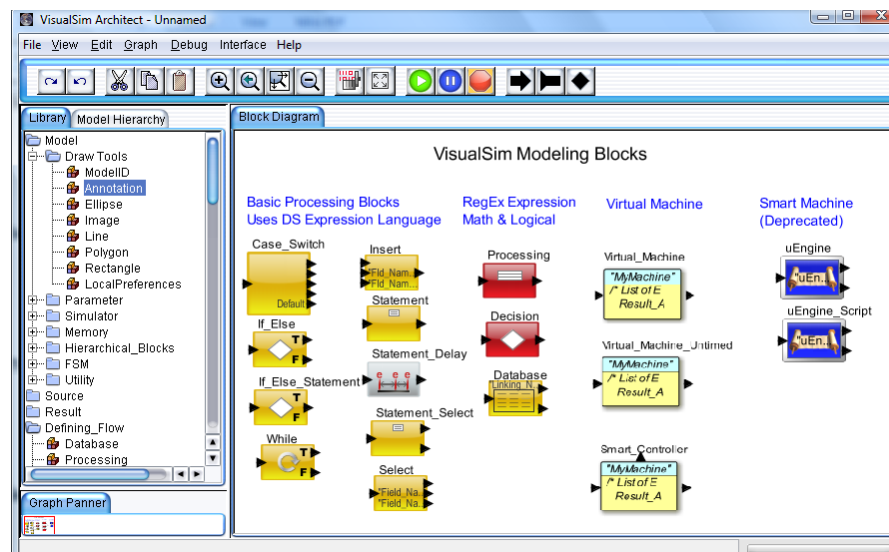


Figure 75. VisualSim Modeling Languages

1. **RegEx Expression:** This is the underlying expression description language in VisualSim engine. The Expression is used in Parameter definitions of all blocks and for the Actions in the FSM. This language contains a large number of functions and methods that are optimized for generating model information and for operating at

extremely high performance. The RegEx can use parameters, variables, and data structure fields. All data types are supported. There are four types of RegEx:

- a. **Operators:** Mathematical, logical, DSP, trigonometric, and imaging functions.
 - b. **Transaction if statement:** $(a > b)? \text{value1: value2}$.
 - c. **Methods:** `getBlockStatus(Queue,"copy",1)`
 - d. **Function:** `array.length()`
2. **Expression Blocks:** The ExpressionList blocks use the RegEx in an Expression containing a RHS and LHS. In this block, the variables and parameters are identified by their name. The data structure is identified by the port on which it arrived. So, the field of the data structure arriving on input1 is `input1.field`. These blocks can have any number of input and output ports. All inputs have to arrive for the block to start executing the lines of expression. All lines of expressions are executed in sequence listed in the block window. The block can send a value, data structure, field, variable or the result of an expression on each port. The ExpressionList block can control whether a value must be sent on an output using the condition. There are no restrictions on the number of values and operands on each expression line.
3. **DS Expression Language:** This language is used in the Basic Processing blocks and are distinguished as the Yellow shaded. This is an easy-to-use subset of the RegEx. There are fixed number of items in an expression line. The block can use the parameters, memories, and data structure fields. The number of input and output ports are fixed. This block is best suited for a casual user. The expression describes Standard, Mathematical, and Logical operations. The operations can be conducted on Scalar Values, that is Integer, Doubles, and Long. There are a number of pre-defined casting provided (no user intervention required) and certain specific function provided (like getting time value, generating distributions and so on). The language does not allow for any extensions, no adding input/output ports and no Array processing (Add, subtract etc.) are allowed in these blocks. The DS Expression has a fixed format namely,

`(X=Y op Z) or ((X logic_op Z) && or || (A logic_op Z)`

4. **Scripting (Script block):** This is a C-like language that is fully integrated with the RegEx. There are three variations of the block - Script can handle timing, Virtual_Machine_Untimed is a high performance untimed block and the Smart_Controller is a subset used in conjunction with the Queue blocks.

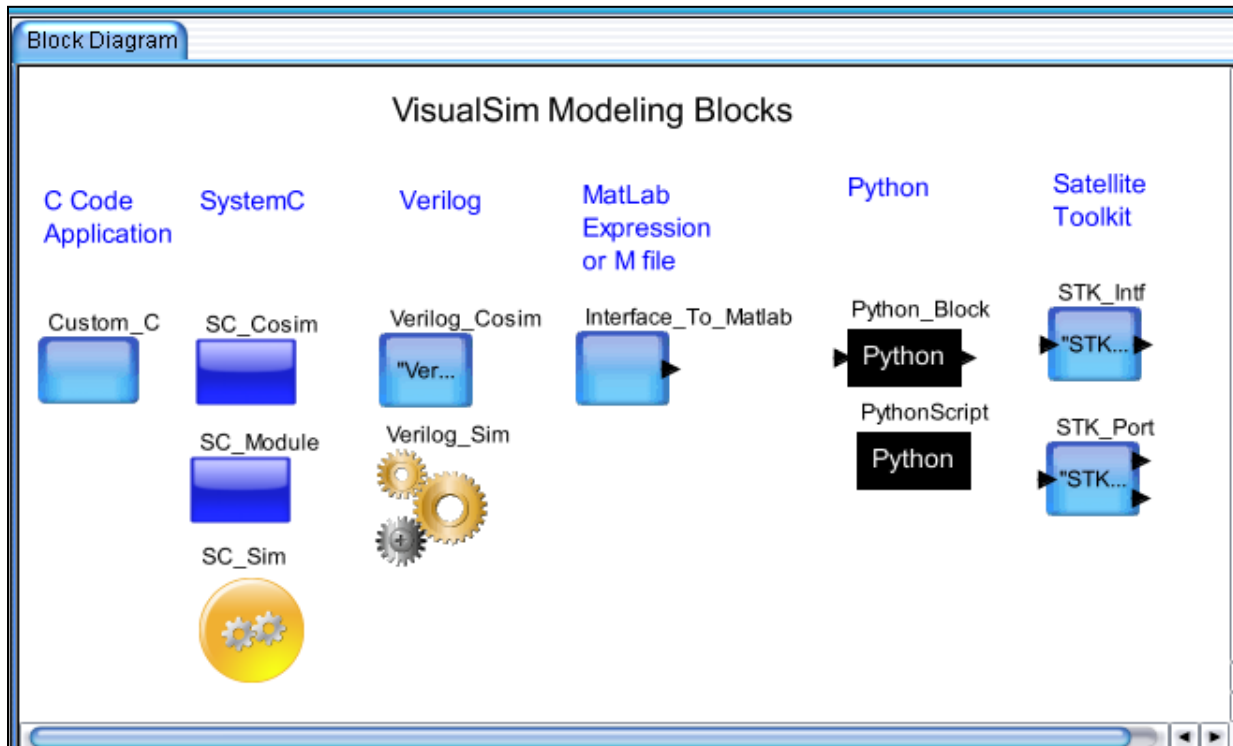


Figure 76.

List of Application Interface Blocks

5. **Custom C-Code:** This block allows the user to import C/C++ code that has been constructed to describe a behavior. The block can maintain state information during runs, count cycles, and have multiple input and output ports. The block defines the code source file and the expected input and return values. A wrapper code is generated to integrate this code without modifying the source.
6. **SystemC:** This block allows the integration of SystemC, TLM 1.0, and TLM 2.0 blocks into VisualSim. The block provides a graphical entry to integrate SystemC models and run it in VisualSim. The SystemC simulator is shipped with VisualSim.

7. **Verilog:** This provides a co-simulation with the commercial simulators such as ModelSim. This block generates a VPI2.0 wrapper to integrate the two simulators together. The block synchronizes time across the two simulator.
8. **MatLab:** This block allows the user to dynamically send functions and values to the external simulator and return with a result.
9. **Python:** This block allows the user to create a script and run it as part of a model.

26 Data Structures Expression Language

The Data Structure expression is a simplified expression mechanism and is used by the Statement blocks. These include the:

Statement
If_Else
If_Else_Statement
Statement_Delay,
Statement_Switch
Statement_Select
While
Case_Switch
Insert
Select

The Expression can be of two types - One/Two variable or three variables. The spacing between all variables and between operators is required. If the spacing is not correct between the variable, errors are generated. The two options are just different ways of expressing the same operations. This is along the lines of action statements in C programming.

The format can either be

X = Y op Z

or

X op = Y

The Statement blocks can accept Data Structure Fields, Memories, Parameters, Numerical Values, and String Constants.

The Expression Language provides Arithmetic, Logical/Binary, and Statistical Operators. The use of these different operators is shown in the Example Section below. The following table summarizes them:

Arithmetic	+ , - , * , /
	sqrt (square-root)
	^ (Power)
	% (Modulo)
	! (invert)
Logical	&& (AND)
	 (OR)
	== (equals)
	!= (Not equal)
	> (Greater than)
	< (Less than)
Binary	& (AND)
	 (OR)
	# (XOR)

	~ (NOT)
	>> (Shift right)
	<< (Shift left)

Statistical	irand (integer random)
	rand (double random)
	Norm (normal distribution)
	exp (exponential distribution)
Combined op + =	++ (Increment)
	--(Decrement)
	+= (add RHS to LHS)
	-= (Sub RHS from LHS)
	= (LHS = LHS RHS)
	/= (LHS=LHS/RHS)
	%= (LHS= LHS Module RHS)
	&= (AND)
	= (OR)
	~= (XOR)

Time Values Can be used on the RHS only	TNow is the current simulation time.
	TReal() is the current wall clock in msec
	TNano() is the last wall clock in nano sec.
	TNanoDelta() is the difference of current TNano and specific time in TNano format. Eg: a = TNano(); b = TNanoDelta(a)
	TRealDelta() is the difference of current TReal and specific time in TReal format. Eg: a = TReal(); b = TRealDelta(a)

26.1 Format for the Statement Blocks

The format for the Statement blocks is as follows:

Option 1				
LHS: X Field	Assignment	RHS: Y	RHS: Operation	RHS: Z
Can contain a field name of the incoming data structure or variable.	=	Can contain, values, Time, parameter, field name of the incoming data structure or variable.	Arithmetic: +, -, *, /, 1/, %, ^ Logical: & (AND), (OR), ~ (NOT), ! (invert), 'xor' (Exclusive OR) Logical Shift: <<, '<=<', '>>', '>=>'.	Can contain, values, Time, parameter, field name of the incoming data structure or variable.
Data Types supported: Integer, double, Boolean, binary, long.	Space is required on either sides of the "=".	Data Types supported: Integer, double, Boolean, binary, long.	Space is required on either side of the "operator".	Data Types supported: Integer, double, Boolean, binary, long.
Notes: 1. Data type casting is from right to left. The "Z" data type is converted to "Y" data type before processing.				

LHS: X Field	Operation + Assignment	RHS: Y
Can contain a field name of the incoming data structure or variable.	=	Distributions: irand(lower, upper), rand(lower, upper), exp(mean), norm(mean, std dev).
Can contain a field name of the incoming data structure or variable.	Arithmetic: =, +=, -=, *=, /=, 1/=, %= !, ^, sqrt, ~, &, , xor Logical: '&=', ' =', '~=', 'xor='	Can contain, values, Time, parameter, field name of the incoming data structure or variable.
Data Types supported: Integer, double, Boolean, binary and long.	Space is required on either sides of the "=".	Data Types supported: Integer, double, Boolean, binary and long.

27 Differences between Script and C Programming

The Script works like a function block in C programming or script. The simple logic expressions use a C/C++ syntax. Following are few differences:

1. The ';' is optional at the end of each line.
2. There must be only one operation per line. This can be an expression, LABEL(), or curly brackets{ }.
3. Declaration are done by Name = value. The data type is polymorphic, meaning that the data type is identified with the first incoming value.
4. Declarations can be anywhere in the code but must be before it is used in an expression on the Left Hand Side (LHS).
5. SWITCH-CASE is all capital.
6. CASE has CASE: LABEL.
7. End of each CASE requires a GTO(LABEL).
8. The last CASE must be CASE: DEFAULT.
9. No pointers are supported and only a reference name for a variable is supported.
10. There must be at least one statement line between { and }.
11. Two { in sequence are not allowed. There must be at least one statement in between.

28 Selecting the right block for your model

VisualSim has a number of library blocks. After you understand the operation of a block, you can apply it in many ways. We have created a table that provides guidance on the use of VisualSim.

28.1 Traffic Generation

Action	Blocks	Example Model
Fixed Rate with fixed size For example, Sensor, I/O	Traffic (Set Distribution to Fixed)	
Clock	Traffic (or) Clock	
Packet-based network traffic Variable data size	Transaction_Sequence (or) Single_Event + TriggeredTraffic + ExpressionList (generate size and delay to next) + Delay in the loop-back to start next sequence	
Multiple instance of any traffic generation	Dynamic Instantiation + (Any combination of blocks inside)	
Bursty traffic + periodic interval	Transaction_Sequence (Time_Field) + Delay in the loop-back to start new sequence	
Sequence of operations. For example, Read, Write, read, Read For example, 32, 64, 128, 256 Bytes	Transaction_Sequence (Time_Field) + Delay in the loop-back to start new sequence	
Custom Distribution	Transaction_Sequence (Time_ and Probability_Field)	

Action	Blocks	Example Model
	+ Delay in the loop-back to start new sequence	
Custom Distribution with variable delay	Transaction_Sequence (Probability_Field) + Delay in the loop-back to start next row	
Trace- Network and hardware	Traffic_Reader	
Use traffic generated from another model	File Reader	
Read from file and use the content for the delay between transactions	Transaction_Sequence	
Start a model execution	Single_Event	
Read from csv/Excel file	Database, Transaction_Sequence and Excel_to_DS	

28.2 Variables

Action	Blocks	Example Model
Constants or fixed values. Value needs to be changed for each run and stays constant in a single simulation	Parameter	
Register or variable	Variable	
Flags for setting system condition For example, Resource availability.	Variable	

Action	Blocks	Example Model
Storing same states of multiple devices For example, Queue length of multi-core.	Variable (array type)	
Storing intermediate values associated with a specific transaction For example, Device trace. For example, Time arrived at a port or processor.	Data Structure field	
Attributes of a transaction For example, Data size, priority, source, destination	Data Structure field	
Fragment number, last fragment indicator	Data Structure field	
Communication information from one part of the system to another instantaneously For example, Queue length of an ingress port.	Variable	
Send notification from one device to the next one	Event (not a block)	
Trigger the sending of the next transaction based on completion of prior transaction	Event (not a block)	

28.3 Plotting

Action	Blocks	Example Model
Latency	ExpessionList + TimeDataPlotter For example, TNow – input.TIME as output value	
Throughput	Variable- Increment each completed size into a variable location Display: ExpresionList + TextDisplay (end of sim) Plotting: Trigger (Traffic) + ExpessionList + TimeDataPlotter	
Read_Latency	ExpessionList + TimeDataPlotter Probe at Source in return path.	
Write_Latency	ExpessionList + TimeDataPlotter If acknowledge required (A_Task_Flag==true), then probe at source, else probe at destination (like DRAM)	
Custom calculation plot (separate based on packet size)	Variable (store results, can be a array) ExpessionList + TimeDataPlotter (might need trigger for regular output)	
Timing Diagram	Timing_Diagram	

Action	Blocks	Example Model
Processor, Cache, DRAM, Linear_Bus, AHB, APB, PCI, PCIx, CA_DRAM, Memory_Controller, AXI		
Extract fields to plot	ExpressionList block (output_values to specific field like input.ID) + TimeDataPlotter	
Two data sets against each other (Throughput vs. packet size)	ExpressionList block (two output ports) + XYPlotter	
Any dynamic variable	Variable + ExpressionList (update) Display: ExpressionList + TextDisplay (or) TimeDataPlotter	
Histogram	Statistics + Histogram (or) Variable + ExpressionList + Histogram	
Multiple dataset from different blocks	TimeDataPlotter or XYPlotter (each dataset comes from a different block or connected to a relation)	
Multiple dataset from a single block output	DS_XYPlotter (for different x and y values) (or) DS_TimeDataPlotter (for y value vs. time)	
Use data structure input but plot specific field values	DS_XYPlotter (for different x and y values) (or)	

Action	Blocks	Example Model
arriving from the same block output.	DS_TimeData (for y value vs. time)	
Set specific color for each dataset arriving from the same block output.	DS_XYPlotter (for different x and y values) (or) DS_TimeDataPlotter (for y value vs. time)	

28.4 Statistics

Action	Block	Example Model
Generate statistics for custom values	Statistics + TextDisplay (or) Statistics + Histogram	
Statistics for Server, Queue and SystemResource	ResourceStatistics + TextDisplay	
Hardware Blocks: Processor, Cache, DRAM, Linear_bus, AHB, APB, PCI, PCIx, DMA_Controller, Req_Ack_Node, CoreConnect	ArchitectureSetup (internal_stats_out, util_stats_out) (or) ArchitectureSetup (Statistics_to_plot parameter + plots_out)- Must set the width of output link to the number of items to be plotted. Use a Relation block (black diamond) and double click on the relation to set the width.	
Other Hardware: Rapid IO, Switch Ethernet, VME, 1553B	Connect blocks' status port to TextDisplay Set Sim_Time parameter to end time	

28.5 Power Statistics

Action	Blocks	Example Model
Average Power	PowerTable (middle port) + TimeDataPlotter	
Instant Power	PowerTable (top port) + TimeDataPlotter	
State Change of each device	PowerTable (lower port) + TextDisplay	
Power/device	Use RegEx powerManager and Cumulative + TimeDataPlotter Eg: A = powerManager("Manager_1") P = (A.Device).Cummulative	
Bar graph of Power/device	RegEx powerManager and Cumulative + TimeDataPlotter	
Bar graph of power/Device as a percentage	RegEx powerManager, and (Cumulative/total Cumulative)*100 for Percentage + TimeDataPlotter	
Check Deadline for task completion	For scheduler, (Time Finish – Time Activation) > Task_Deadline	
VCD output	VCD_Writer	

Action	Blocks	Example Model
Write data structure for use later or in another model	FileWriter	
Write to Excel	CSVWriter	
Write to XML	DS_to_XML	
Write to file any text display or plot	Use the Save setting of the plotters	

28.6 Resources and Hardware

Action	Blocks	Example Model
DMA	Channel (statistical) (or) DMA + DMADatabase (or) DMA + ExpressionList	
Switch	Switch (or) SystemResource (Blocking) (or) Non_Blocking_Switch (or) Server (Non-blocking)	
Multi-core with a pool of cores for first come-first serve	Server_N_Resource	
Multi-core with assignment to specific cores	SystemResource (or) Server	
Multi-core with selection of first available	Group of SystemResource (or) Server + RegEx Queue Name + “_Length” to find the first 0	
Wireless/wired channels with error detection and retransmission	Channel + Channel_Complete + Database (error probability)	

Action	Blocks	Example Model
Wireless channels with errors but no retransmission	Channel + Channel_Complete (all data sent to accept port and rejected after channel_complete) (or) Server (rejected after block)	
RTOS	SystemResource (or) Queue + Smart_Controller (or) Server (Slot Mode)	
Scheduling- First come-First Serve or Round Robin	SystemResource or SuystemResource_Extend	
Scheduling- Any algorithm	Queue + Smart_Controller	
Scheduling- Preemption	SystemResource	
Memory allocation & deallocation	DRAM (or) CycleAccurateDRAM + Memory_Controller (or) Counter (Memory) + ExpressionList (to check for available memory) + Queue	
Cache	Cache (or) IntegratedCache (or) Memory + ExpressionList (generate hit-ratio and see if it is below a threshold)	
Processor	Processor (See all Processor examples in Block Documentation for different configurations) (or) Traffic (or) Traffic_Reader (Trace file input) (or) SystemResource (or) Server (or) Computer Model	

Action	Blocks	Example Model
Thread Management	Queue + Smart_Controller (for coordinating the threads)	
Timer	Server	
I/O and any form of interface	Traffic (or) Transaction_Sequence (or) Traffic_Reader (or) SystemResource	
Multichannel I/O	Server_N_Resource (or) Server (or) Channel	
Fiber Channel	Channel (or) FC_Switch	
Disk	DRAM (or) use Disk Template	
Flash	DRAM + Script (custom controller) (or) Flash template	
Non-blocking switch	Non_Blocking_Switch (or) Server	
Ingress/Egress Ports	Server (for known delay) (or) Queue + Smart_Controller (for custom arbitration, with traffic management) or SystemResource (for single port)	
SATA	Queue + Smart_Controller	
Arbiter/Scheduler	SystemResource (or) Queue + Smart_Controller	
Manage resource quantity with intelligent data organization on physical media	Quantity_Based_Resource	
Memory (or) Storage (or) Archive	Quantity_Based_Resource	

28.7 Utilities and Functions

Action	Block	Example Model
Fragmenting	While (Requires field to identify each fragment and another field to denote last fragment)	
Defragment	ExpressionList (Look for last fragment field)	
Routing or path selection	Database (lookup or routing)	
Redirect or select a flow	Virtual Connection (IN, OUT, Mux and Demux)	
Goto	Virtual Connection (IN, OUT, Mux and Demux)	
Wait for control to send output	Boolean_Switch	
Manage clock boundary transition	ClockAlign	
Manage clock boundary transition and do a calculation before output	ClockAlign	

29 Error messages

There are three types of errors in VisualSim.

1. The first is a **Syntax Error** which means that information in the block fields has been entered incorrectly. VisualSim does not allow the user to continue until the error is corrected.
2. The second is a **Run-Time Error** which is identified when the model starts executing. These are the errors that require some basic understanding to detect.
 - a. **Data Type Error:** All blocks have ports that identify the data type at runtime. Some block ports might be assigned a specific data type, typically **Double** or **Boolean**. If a block is pre-assigned a fixed data type, this port type cannot be modified. The data type for the port attached to it must be modified. The port type is modified by right-clicking the block and then selecting "**Configure Port**". The Type can be modified by selecting the appropriate data type from the list.
 - b. **Channel not in range:** This indicates that the input port listed requires to be connected but is currently open. This port is a required port and must have an input for the block to fire.
 - c. **Null:** This is typically the most difficult error to detect. If this occurs for the Display_Fast (Buffered), then simply delete the block, and replace it with a new one. If this is for any other block, this indicates that a port is not connected in the data flow. Look at all the wires and identify the ones that are thin as opposed to fat. This wire will not be connected.
3. The third type of error is an activity in the model that causes an error. For example, the queue may overflow due to the arrival rate being too high. This requires the queue depth to be increased or the input rate to be reduced. These are design consideration errors.

30 Model-level Debugging

VisualSim has a number of graphical and textual debugging built into the environment. In addition, the user can attach third-party code-level debuggers to the environment debugging block level details. You can easily debug your models in VisualSim by using a combination of techniques. The easiest and fastest way to "see" what the model is doing in terms of block execution is to use the visual animation through Model Animation. Next, one can find the Data Structures being sent through the model, using the "**Listen to Port**" feature on any output port in the model.

The key capability available includes:

- **Listen to Port:** All blocks including Class instances, except Hierarchical blocks, Right-click on the port to select "**Listen to Port**". You have to click the inner-most point of the port to get the Menu.
- **Listen to Block:** All blocks, except Hierarchical blocks and Class instance, Right-click on the block to select "**Listen to Block**". If you need to see the details of operation within a Hierarchical block, you need to select Open Block and select the block to view the execution details. If you need to view the detailed operation of a Instantiated class, you need to select Open Instance and select the block to view the execution details.
- **Listen to Simulator:** This feature provides a sequence of execution for the selected simulation. This is also integrated with the Digital Debugging utility in the Digital simulator. This window displays the usage statistics, current block execution, and model summary information. From the Debug Menu, select "**Listen to Simulator**" and select the debugging option in Digital simulator parameter "digitalDebugger".
- **Animated Debugging:** Animate the model by selecting **Debug > "Animate Execution"** and specifying a time in ms. Select "**Stop Animation**" to bring the model execution back to the regular mode.
- **Breakpoints:** The model can be stopped by setting a Breakpoint on a block by right-clicking the block and select "**Set Breakpoint**". Refer [40.12 Setting Breakpoints](#).

- **ShowTypes:** Drag and drop this block to see the individual port types. If the port type mismatches, error is generated. This block helps in indentifying the non native connection of ports.

30.1 Model Animation

You can animate block diagrams to debug your simulation or simply to view the dynamic operation of the system model. VisualSim animates a model by highlighting the executing block, based on the 'Debug/Animate Execution' time setting (msec). One can also turn off animation by using the main menu bar 'Debug/Stop Animating'. The figure below illustrates Statement_Select Block is currently firing with animation enabled.

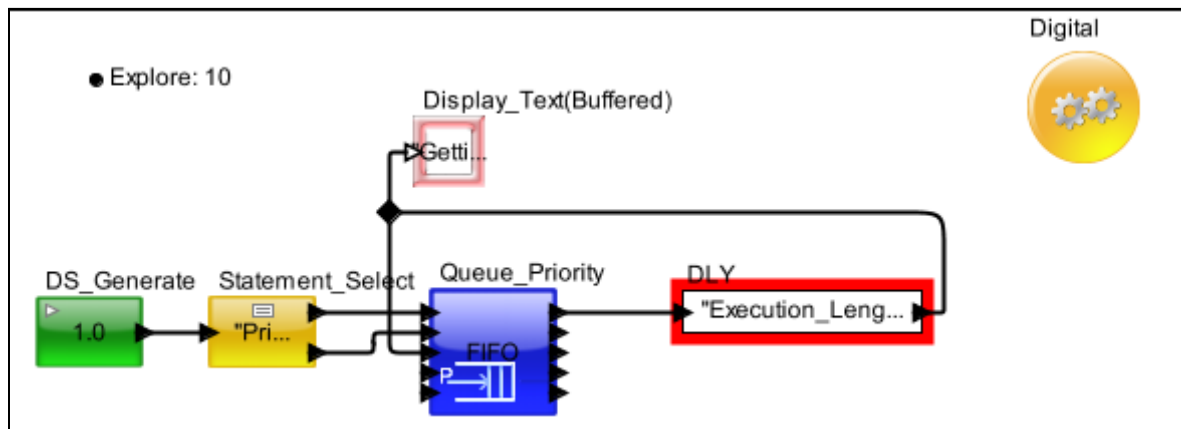


Figure 77.

Model Animation

The animation can be paused. At that point, either you can start the simulation, stop the animation or start other debugging operations.

30.2 Display and Statistics

Operation: Select the display block from the Library palette. Drag the block onto the canvas. Connect the block to the input and output as required in the model.

Explanation: Display, and statistics are provided as extract information from the model during simulation. These blocks can be embedded in the simulation model during construction. These entities will extract the appropriate fields in the data structure or the entire object and display

them. The change in these values can be viewed while the simulation is being executed. Moreover, all the SmartBlocks have built-in statistics gathering.

30.3 Listen to Simulator

Operation: In the menu of the Simulation Cockpit, select the **Listen to Manager** and **Listen to Simulator** under Debug menu.

Explanation: The internal operations of each block and the model, itself, can be viewed by turning on the "Listen to Simulator". The **Listen to Manager** records the activities at the simulation kernel level. This records the specific models that are fired and kernel activities for total debugging.

30.4 Listen to Port

"**Listen to Port**" can be enabled in the Block Diagram Editor, simply by right-mouse clicking over the desired port. **Listen to Port** then opens a separate window, identifying the output port location in the top bar of each window. You can open as many "Listen to Port" windows as needed to monitor a simulation. After the simulation is started, each Token passing through a port is displayed in "Listen to Port" window. If the Token type is IntToken, DoubleToken or BooleanToken, then one sees the value. If the Token type is RecordToken or StringToken, then one sees either a simple string, or the Data Structure in an Array. If there is no output on the "Listen to Port" window, make sure the right port was selected, or alternatively this indicates that the model did not generate any output from this block. If there is no output, then one needs to check the ports, or virtual connections, driving this block to see that they are being activated correctly. If the ports driving the block seem to be correct, then experiment with "Listen to Block" on that block.

30.5 Listen to Block

"**Listen to Block**" can be enabled in the Block Diagram Editor, simply by right-mouse clicking the desired block. "Listen to Block" provides more insight into the internal block operation, and may explain why an expected output port did not fire as anticipated. For example, if one is using

If_Else Block, and a Data Structure did not appear on the "true" output, then one can view the "If-else" decisions being made internally, simply by performing "Listen to Block". In addition to block level information, "Listen to Block" provides simulator level information relative to methods being fired, and so on. Simulator information is probably most useful for evaluating your custom blocks in the simulation environment.

This feature is not available for Hierarchical and instantiated hierarchical class blocks. If you need to see the details of operation within a Hierarchical block, you need to Open Block and select the block to view the execution details. If you need to view the detailed operation of a Instantiated class, you need to select Open Instance and select the block to view the execution details.

30.6 Script Profiler for the Script block and Smart Controller

The Script Profiler is embedded in the block execution and keeps track of the number of times a statement executes and also the average time the statement took to execute down to the nano-second level. This feature provides all users with the detailed execution information about their script. In addition, it provides the script address, and the script statement associated with that address. If the number of times executed equals 0, then this provides feedback on whether a certain function is even executed. This may be useful information, sometimes referred to as code coverage in lower level verification testing.

Profile_File: The Script Profiler can be turned on by adding a parameter to the Script and Smart_Controller block called "Profile_File". This identifies the name of the file to be generated by the profiler. If the parameter does not exist, or is set to "none", then the Script Profiler is turned off. A typical Profile_File, Block_Name + "_Profiler.txt".

Path:One can place the entire path in "Profile_File" variable, or add the "Path" variable. A typical path, "C:/VisualSim/Profiles". If just a file name is used then the file will created under VS_AR.

30.7 Listen to File for Script block and Smart Controller

The Listen to File is a text based version of listen to block.

Listen_to_File: The Listen to File can be turned on by adding a parameter to the Script and Smart_Controller block called “Listen_to_File”. This identifies the name of the file to be generated. If the parameter does not exist, or is set to “none”, then Listen to File is turned off. A typical Listen_to_File value is Block_Name + “_Listen_to_File.txt”.

Path: One can place the entire path in “Listen_to_File” variable, or add the “Path” variable. A typical path is “C:/VisualSim/Listen_to_Files”.

30.8 Debugging Variable

There are two blocks - **Variable_Dump** and **Variable_Monitor** Block for viewing variable content. **Variable_Dump** can generate the value in a variable location when triggered on the input port. The variable name is specified in the parameter and can be a location or the complete list of global, local or block variable. If the default is retained, the complete list of all variables is printed out. The **Variable_Monitor** Block can monitor individual variables by name or by triggering the input port of this block. One can list "all" variable names, to get values on an output port, each time it is triggered. The Variable_Monitor Block has a parameter entry that one can enter a comma separated list of variable names to monitor as the simulation executes. Each time the list of internal variable names are accessed, read or write, then it will generate a trace Data Structure on its output. The primary difference between the blocks is that the Variable_Dump requires a trigger to generate the variable values while Variable_Monitor will generate the values every time variable is accessed.

One might also use the Read_Variable Block. Each Data Structure triggered on the input gives the current value of the named-variable on its output. You can also have multiple Read_Variable Blocks in the model to read the same variable location. One could "Listen to Port" on the output of Read_Variable to see the variable values as the simulation runs.



Read_Variable block is in **Model >Variables**.

Finally, one might use the IN Block, then you can trigger from any number of OUT blocks within the model, which triggers the IN Block (assumes OUT and IN names match and of same type

(local or global)). When the IN Block is triggered it reads the variable parameter name entered (local or global) and sends to its output. Though this block was created for easier plotting, it works equally well for variable monitoring. IN Block can perform conversions, such as integer to double.



IN is in **Defining Flow** -> **IN**.

30.9 Debugging a New Model

If you are debugging a new model, then there are some things you can do while still constructing the model:

- **Workload Generator** -- After adding the workload generator, or trace file that generates composite Data Structures, it might be helpful to test that this portion of the model is generating the necessary workload, using "Listen to Port", or adding a Text_Display Block. This save time later, when more behavioral and architectural blocks are in the model.
- **Hierarchical Block** -- For each hierarchical block used in the model, it might be useful to debug each hierarchical block before adding it to the top level model. To debug a hierarchical block, one needs to add a Digital Simulator to the block, possibly disconnect input and output ports of the block, and add a simple traffic generator to test the functionality of the blocks. After debug is completed on the hierarchical block, then one needs to remove the Digital Simulator, reconnect the external input and output ports, to use it in the top level system model.
After the top-level system model is constructed that is items (1) and (2) are completed, then one can begin to look at the following items:
- **Data Structure Flow** – The Traffic blocks generate the data structure. To check the correctness of the field values, use "Listen to Port", "Listen to Block", or add Display Block at the output of a block.
- **Behavioral, Architectural Processing** -- Now begin to focus on the functionality of the behavioral aspects of the model, the state machines (if any), algorithm validity, and so on. A behavioral portion of a model typically has pure function associated with it, and no delays. To check out the behavioral portion, one can use "Listen to Port", "Listen to Block",

or TextDisplay Block to check the Data Structure flow within state machines, protocol, or algorithms. The architectural aspects of the model can be debugged in terms of Data Structure flow, similar to the behavioral debugging. An architectural portion of a model typically has architectural delay.

VisualSim has a number of debugger features and methodology that simplifies model development and refinement.

30.10 Digital Debugging

This is a graphical editor version of the popular textual debuggers. In the Digital Simulation mode, users can get model summary, the firing order of the blocks and perform step-by-step evaluation of the model. To enable this feature, select **Pause**, **Summary_Only** or **Run** in the **Digital Simulator** icon. If the top-level simulator is enabled, all lower-level simulators must be set to **Off**.

The debug mode can be triggered when the "**DigitalDebuggingExpr**" parameter evaluates to **true**. This parameter is a condition and can contain any **RegEx** function. You can use keywords such as **TNow**, RegEx functions such as "**getBlockStatus**", **global** and **local** (requires the Window name.variable_name) memories and number accesses to a block name.

To view the debugging activity, select **Listen to Simulator** in Debug menu at the current or top-level. The debugging can be turned on at the top-level to view activity for the entire model or can be done at any Hierarchical window.

When the expression evaluates to false, the model continues running as before.

Note: To stop the simulation, while debugging, you need to press Stop and Play/Go twice.

The debugger parameters are:

- **Off**: This disables the debugging mode and is recommended when the model is running in batch mode or when running interactive simulation of a complete model.
- **Pause**: This turns on the debugger to stop at every block in the model flow. The GO button must be selected to continue the simulation after the model stops at each block.

Use Listen to Simulator to determine which block is currently being fired. This provides a model summary at the end of the simulation (see Summary_Only).

- **Summary_Only:** At the end of the simulation, this generates a list of all the blocks in this Hierarchy. For each block, the output shows the number of times the block was fired, the average execution time for each firing, and the total time spent in that block. It also lists all blocks that are not used in the model. Some blocks do not get fired by a port. These will show in the inactive list. Example of these blocks are IN, OUT, SystemResource, and Init_variable.
- **Run:** This mode outputs the order in which each block is fired in the model. This provides a model summary at the end of the simulation (see Summary_Only). This does not stop the simulation at every block.

30.11 Setting Breakpoints

30.11.1 Set Breakpoints context menu choice

The context menu for actors includes a "Set Breakpoints" menu choice.

"Set Breakpoints" is currently only implemented for the Digital and SDF domain. "before iterate" and "after iterate" work with SDF. In the case of the Digital Simulator, you can "Stop on Entry to Block" or "Stop before Executing Timed Activity". After the simulation stops at a breakpoint, you can unselect the breakpoint choice.

The **Set Breakpoints** facility stops the model at a specific point in a specific actor. This is helpful if there are multiple instances of an actor within a model.

- ⇒ Start ModelBuilder from within a debugger
- ⇒ Open an SDF model

Warning: Save your model before running with breakpoints set because saving in the middle of a debugging session may not work.

- ⇒ Right click an actor in an SDF model, select Set Breakpoints > before iterate.

- ⇒ Run the model.
- ⇒ The model stops when it hits the breakpoint.
 - A message appears in the status bar at the bottom of the ModelBuilder graph display.
 - The actor with the breakpoint is highlighted in magenta.
- ⇒ Click **Run** again to continue execution.

30.12 Pause and Resume function with Debugging facility

Pause and Resume function is a debugging and step simulation facility that allows users to specify time stamps to pause simulation and save the simulation data, events and status in a file. The saved model can be restarted from that simulation time. This feature is handy to debug simulations that run for a large period of time. Moreover the user can analyze system behavior at various points in the simulation.

The pause and resume function can also be used for debugging. User can pause at a timestamp and analyze the system response and continue simulation step by step from that point onwards. The system can be analyzed for required functionality and also helps the designer to identify if the crucial tasks are being executed within the deadlines.

30.12.1 How to Use Pause and Resume Function

The latest version of Pause and Restart supports the following library blocks - SystemResource, SystemResource_Extend, Server, Queue, Script, ExpressionList, Database, Delay, SingleEvent, Smart Controller, Traffic and File Reader.

The PauseSimulationAt.txt file must be located in the same directory. This file contains the timestamps at which the simulation activity needs to be saved.

Note: The simulation pauses at the closest event after the time value in the PauseSimulationAt.txt file. This means that the simulation may not stop at the exact timestamp.

30.12.2 Steps:

Manual Mode

12. Save your current model.
13. Run Simulation by clicking the green arrow button in the tool bar or press CTRL+R.
14. Pause Simulation by clicking Pause button in tool bar.
15. Click Save As.
16. Select the check box "Save Simulation Data".
17. Give Name for the model with .xml extension
18. Click Ok.
19. .bin file is created in the same directory and this file has the Simulation data.
20. Continue the simulation and repeat the above process to stop the simulation at additional times.

To start the simulation at a specific point,

1. Open the saved file for that simulation point.
2. Model starts running from the time stamp where it was stopped.
3. The model runs to completion, if the PauseSimulationAt.txt file does not exist or the user does not Pause the simulation.

30.13 Testing the RTL

Hardware designs are built to a specification. Due to Clock-based accuracy requirements and the need to migrate the design components from specification to implementation, a VCD writer is provided as a standard block in the library. This block can be used to generate vcd files and the resulting values can be used as the golden reference to validate the implementation. An example is provided in VisualSim to show the application of this block in a real example. The generated vcd file can be viewed by any standard waveform viewer. The generated file can support integer, wire, register, and double.

30.14 How to Debug- Using the methodology provided

30.14.1 INTRODUCTION

Models constructed in VisualSim can help you make better design specification decisions. Evaluations of system specification using VisualSim performance, power and architectural models

can help eliminate clearly inferior choices, point out major problem areas, and evaluate a variety of cost, performance and partitioning trade-offs. Simulation is cheaper and faster than building hardware prototypes and can also help with software development, debugging, testing, documentation, and maintenance. This document will provide the information on how the models are debugged. It also explains the various methods of debugging a model.

METHODS TO DEBUG

- Animate
- Listen to Simulator
- Listen to Port
- Listen to Block
- Tracer

30.14.2 Animate

When there is an error in the model we should identify where the error has occurred, so Animate will help you to trace the path and to capture the block where the error has occurred.

To enable this feature

- Go to Debug >> Animate Execution from the menu
- Click OK
- Then Run the model

Now you can notice the animation over the blocks which helps you to capture the error.

30.14.3 Listen To Simulator

Listen to Simulator will help you to get the detailed analysis on the model. It summarizes the firing order of the block and performs step by step evaluation of the model.

To enable this feature

- Go to Configure >> Digital Debugger >> Summary_only in the Digital Simulator
- To view the debugging information, select Debug >> Listen to Simulator from the menu.

Then run the model.

This will help you to view the summarized result of the model.

30.14.4 Listen To Port

Listen to Port will help you to find out the incoming/outgoing data of that particular port of the block along with the time, which will be useful for identifying the missing data.

To enable this feature

Right click on the port and select Listen to Port

Then Run the model

The tokens that are sent/received are displayed.

30.14.5 Listen To Block

Listen to Block will help you to get the detailed analysis of the data present in the block at a particular time. It also displays the input and output of that block along with the time. It is basically used for ExpressionList, Script, Smart Controller where it displays the result for each of the statements.

To enable this feature

Right click on the block and select Listen to Block

Then Run the model

The tokens that are present in the block are displayed along with the input and output.

30.14.6 Tracer

The Tracer Block is similar to the Listen To Block feature. When the Tracer Block is instantiated in the model, it captures the information of multiple scripts when they are executed and writes them into a text file. It is used to check the communication between different scripts in the model and

also the sequence of execution. It writes into a text file called VisualSimTraceLog.txt in .Visualsim folder under user directory (for windows C:\Users typical user directory). In the Log file we could see what happens when a data structure comes into each of the script blocks and how the execution flows within. This makes debugging easy.

30.14.7 Difficulties in debugging any system and how it is different from debugging a regular piece Java or C/C++ code.

Software systems are complex and debugging is hard. This is especially true for embedded systems where there may be real-time constraints on when data is received or sent, the timing of interrupts, and so on. This means that traditional techniques, such as breakpoints or changing to code to print out state information, cannot be used because they will drastically change the timing and therefore the behavior of the program. Program flow is often controlled by asynchronous and external events, for example inputs from touch sensors and other peripherals whereas while debugging a java code it will be either a compile time error or a run time error.

Generally, these two scenarios are encountered:

Syntax error: Consists of grammatical errors or incorrect use of syntax in code.

Logical error: More difficult to rectify as it involves correcting logic in the code itself. The program may still run with a logical error, it will not run with a syntax error though.

This can be fixed by means of traditional techniques such as breakpoints or changing to code to print out state information.

30.14.8 Sequence for debugging

With the help of Display Stack Trace find out the error block.

The animate can also be used to track down the bug in the model so that we can find which block has to be modified.

Perform Listen to port/block at the particular block.

Check for the correct values in the fields

Check the expected order of execution.

Debugging feature used

SCENARIO 1

Click on the Display Stack Trace and check where the error has occurred. Use the explanation given in the error and update the changes. In some cases the error message will have suggestions to try on.

SCENARIO 2

Even if the model runs we will get some incorrect output so to rectify that we need to first animate the model while running and then check whether it is traversing through all the blocks. If not, we need to stop the process and make changes in the block by using Listen to Port/Block.

31 Software Modeling

Software modeling for a target hardware platform is a complex process, one has to identify and represent the software component and their relationship with the each hardware devices. VisualSim provides four ways to model the software without compromising the accuracy.

31.1 Task Graph Approach

The task graph method allows the user to define the sequence of operation with or without dependent flows, where each operation assign the task to specific hardware resource in the design. This make sure the tasks are processed according to the software sequence that are implemented using the mapper and system resource combination. The design can contain hierarchical order of task graph to replicate the software flow and each task can be assigned to particular resource or a resource in pool of hardwares. The designer can determine the complete flow, number of hardware component, and scheduling or dispatch of task to appropriate hardware unit. The following image shows an implementation of taskgraph based processing where each operation sends request to the dispatcher and the tasks are dispatched to predefined hardware components according to the functions.

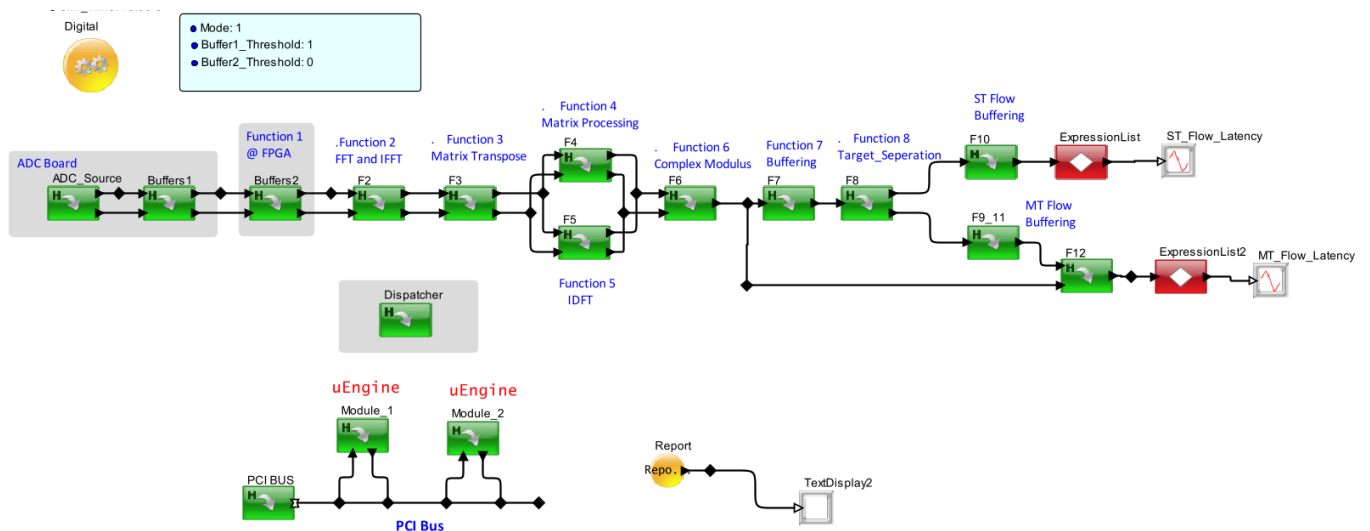


Figure 78. Task Graph Based Software modeling.

31.2 Task Generator Approach

Task Generator method allows user to define the application programs as a combination of instructions. A Instruction Mix table will contain a task or list of tasks, instruction types, and the corresponding instructions in each type. A task will be defined with percentage of each type of instructions. This helps the user to recreate the instruction profile for an application to analyse the performance of the entire design by feeding the tasks to appropriate hardware unit such as Processor and DSP.

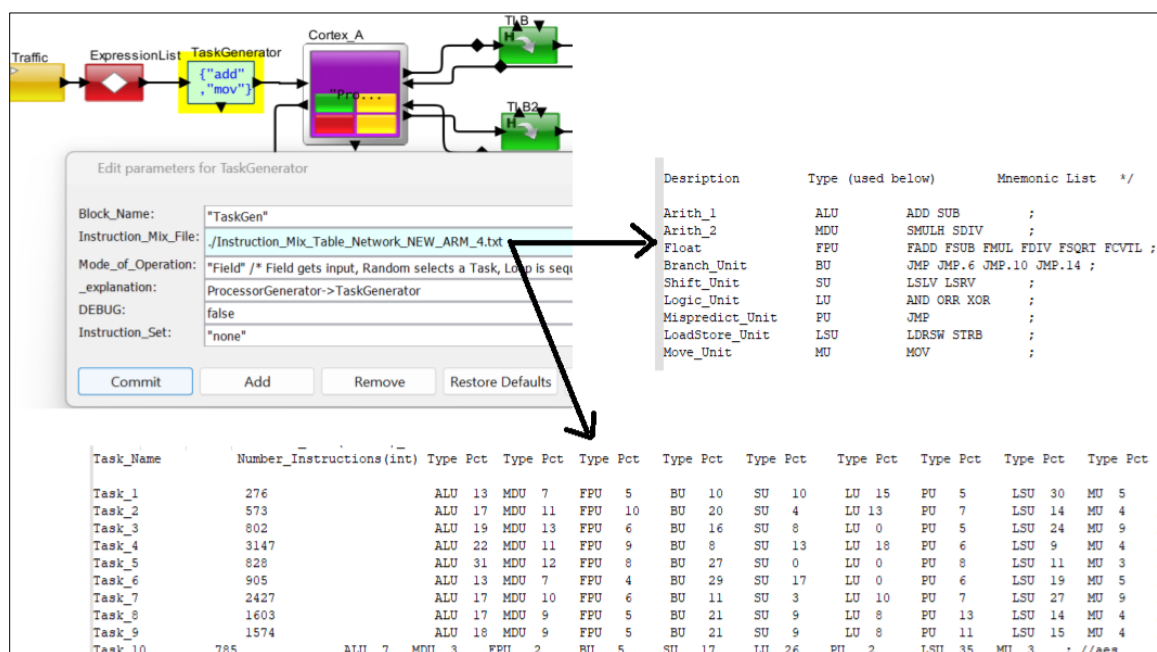


Figure 79. Task Generator Based Software modeling

31.3 Trace Approach

The trace based modeling helps the user to optimize the specific device in the entire design. The trace can be obtained from the real hardware or cycle accurate design model. The trace file will contain the instruction sequence of an application program that ran on a specific hardware. By feeding this sequence into the hardware design the user can observe the impact of each hardware components and optimize them to get improved performance. The trace will contain the I cache address, Instruction and D cache address for each instruction that are executed.

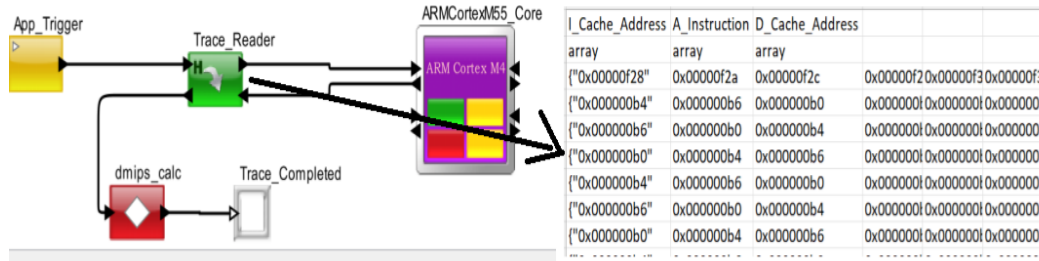


Figure 80. Trace Based Software modeling

31.4 Microarchitecture Approach

The micorachitecture method generates the instructions for a giver application program. The disassembly code is generated through the gcc compiler and the the instructions are fetched, decoded, issued, executed based on the functions defined in the application program. The control flow of instructions are handled by the computed result and system handler. The helps the user to model and anlyase the entire system in low level design.

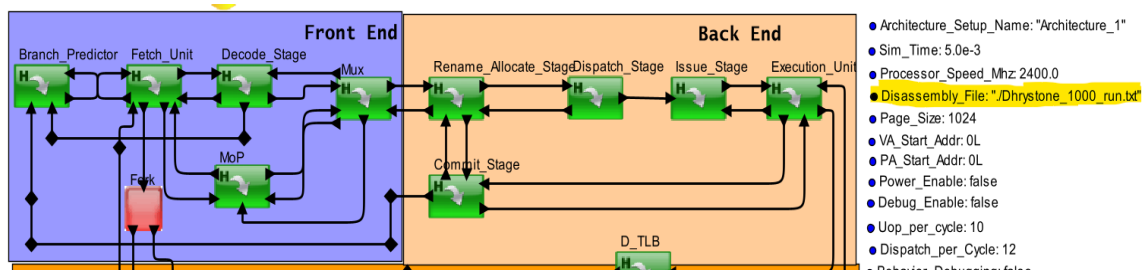


Figure 81. Low level software modeling

32 Introduction to Finite State Machine

Finite State Machines (FSM's) are used extensively in designing sequential control logic. FSM is a very intuitive way to capture control logic and easier to communicate a design. The VisualSim philosophy of hierarchical composition of heterogeneous models of computation allows embedding hierarchical FSMs within a variety of concurrency models. FSM model is contained by an instance of FSM Block. The FSM model reacts to inputs of the FSM block by making state transitions. Actions such as sending tokens to the output ports of the FSM block can be associated with state transitions.

Expressions give the guards for state transitions, as well as the values used in actions that produce outputs and actions. This sets the values of parameters in the refinements of destination states. The exceptions are parameters that are strictly string parameters, in which case the value of the parameter is the literal string, not the string interpreted as an expression, as for example the *function* parameter of the *TrigFunction* actor, which can take on only "sin," "cos," "tan", "asin", "acos", and "atan" as values.

32.1 Finite State Machine

32.1.1 FSM-Controller

The FSM Block contains states and transitions. The State has two ports: incomingPort, which links to incoming transitions to the state, and outgoingPort, which links to transitions going out from the state. The Transition links to two ports - the outgoing port of its source state and the incoming port of its destination state. The two ports of the States are not displayed on the circle.

32.1.2 Guard Expressions

The guard of a transition is specified by its guardExpression string attribute. Guard expressions are parsed and evaluated using the RegEx expression language. Guard expressions must evaluate to a Boolean value. A transition is enabled if its guard expression evaluates to true. Parameters of the FSM block and input variables (defined below) can be used in guard

expressions. Input variables represent the status and input value for each input port of the FSM block.

32.1.3 Actions

A transition can have a set of actions that produce output tokens or set parameters of the FSM block. A transition has an `outputActions` attribute and allows the user to specify a list of semicolon separated output actions of the form `destination = expression`. The expression can use parameters of the FSM block and input variables.

32.2 Execution

The methods that define the execution of an FSM block are implemented as follows:

Preinitialize: Create receivers and input variables for each input port; set current state to the initial state as specified by the `initialStateName` attribute.

Initialize: perform simulator-specific initialization by calling the `initialize (Block)` method of the Simulator. Note that in the example given in AMI Coder of the Simulation Domain documentation under FSM, the Simulator is the Untimed Digital Simulator.

Prefire: always return true. An FSM block is always ready to fire.

Fire: set the values of input variables; choose the enabled transition among the outgoing transitions of the current state; execute the choice actions of the chosen transition.

Postfire: execute the commit actions of the last chosen transition; change state to the destination state of that transition.

Non-deterministic FSMs are currently not allowed. The `fire()` method checks whether there is more than one enabled transition from the current state. An exception is thrown if there are more than one transition from any state. In the case when there is no enabled transition, the FSM will stay in its current state.

32.3 FiniteStateMachine

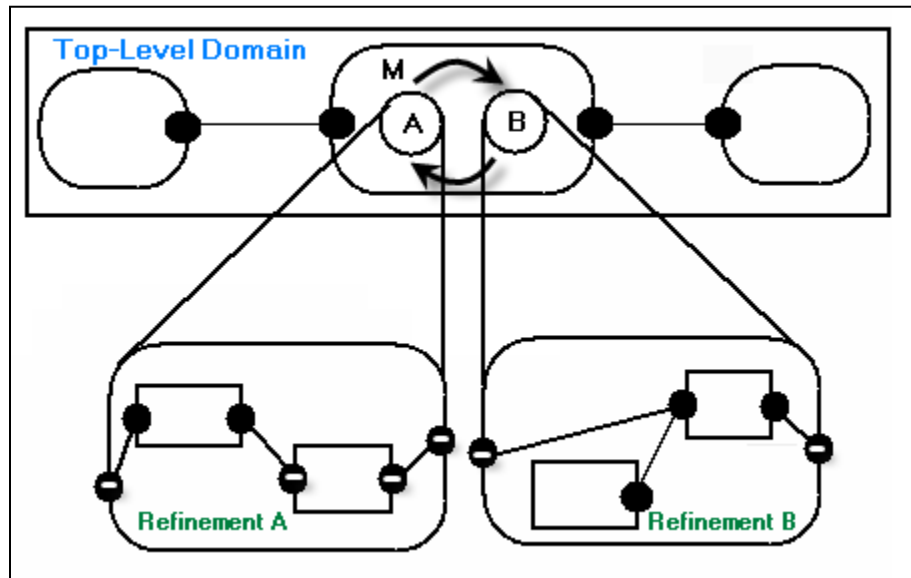


Figure 82. FiniteStateMachine Illustration

The FSM simulator supports the *charts formalism with modal models. The concept of modal model is illustrated in Figure 76. M is a modal model with two operation modes. The modes are represented by states of FSM that controls mode switching. Each mode has a refinement that specifies the behavior of the mode. In VisualSim, a FiniteStateMachine is an entity in the Base Blocks and uses the FSM simulator internally. The refinement in the States and the Transitions can either be another FSM or a Block Diagram.

33 VisualSim Cloud and File Export to Web

VisualSim Cloud support allows the user to launch the full featured VisualSim in Online Web version. This makes the user to access the tool, build the model and run simulation without installing the tool in the local system.

33.1 How to access VisualSim Cloud

The user have to get login credentials by registering for the VisualSim cloud [here](#). Once the credentials are received through the mail, please use that in the same page to login.

Once you login, the site will give you clear instructions on what to do. The prerequisite for VisualSim cloud are

OpenWebStart - <https://openwebstart.com/download/>

Java 17 - <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

Once you have these and you launched the tool from the web page the Startermodel.jnlp will be downloaded. Go to the downloaded location and double-click on Startermodel.jnlp . Click yes on java security warnings.

Now the VisualSim Cloud version is available for the user.

33.2 Export model to web. (Disabled in the current version. It will be available in the future release)

User can share the model to someone else in the team or company, where they do not want to install the tool in the local system. The user have to export the model to web and share the html page, by this way the can share the model access them using VisualSim cloud.

To Export a model to web, open the model in VisualSim Tool and select File-> Export->Export to Web.

Then user will get the following window and update them as mentioned below.

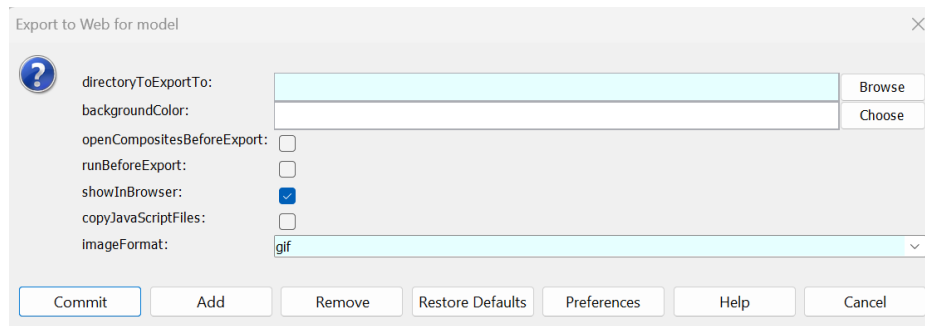


Figure 83. Model export to web configuration.

Click Browse on directoryToExportTo option and select the model directory. Enable openCompositeBeforeExport, runBeforeExport, showInBrowser and CopyJavaScriptFiles in the window. Click Commit.

Now the Index file for this model is created in the specified directory.

By sharing this index.html file the user can launch the model suing VisualSim cloud.

Once they click launch, the jnlp file can be downloaded in the local system, and then by double clicking them the VisualSim cloud will open the model.

34 VisualSim® Interface with FPGA

We describe the integration of the VisualSim modeling environment with FPGA-in-the-loop. Here we have considered Xilinx Zynq 7000 platform board as the example device that communicates with VisualSim®. However the source code and the setup is completely generic. This interface has been demonstrated on Xilinx Zynq 7000 platform board (Zybo board), using the Vivado tool suite from Xilinx and the accompanying Xilinx SDK. The version of Xilinx tool used is Vivado 10.2. Using this interface the designer can communicate in both direction between VisualSim and the FPGA boards.

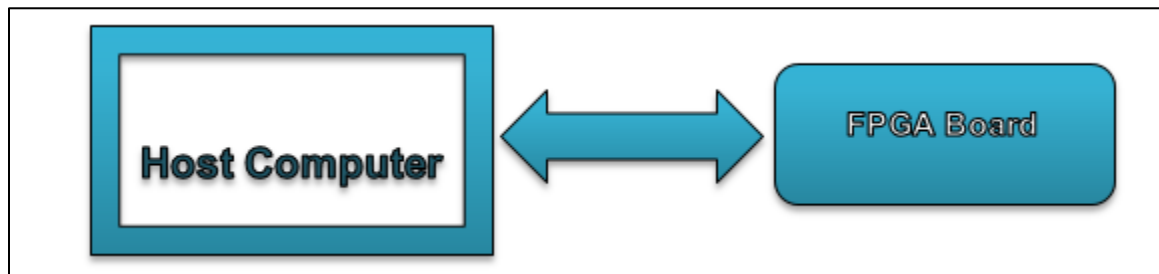


Figure 84.

VisualSim Interface with FPGA

34.1 VisualSim FPGA Interface applications

VisualSim FPGA Interface can be used for System level analysis of Embedded Systems including performance, protocol checking, power, and for generating test scenarios for the target FPGA. VisualSim can generate verities of test scenarios very quickly and enables the user to validate application on FPGA covering all corner cases.

VisualSim FPGA interface allows users to adopt below mentioned applications but it is not restricted.

- Generating and running system level scenarios
 - VisualSim can be used to generate system level scenarios from the system definition.
 - Transferring the data to the FPGA to run actual data through the system.
 - Transfer the data back to VisualSim to perform analysis and display the information in user readable format.

- Run performance simulations on the actual hardware system
 - VisualSim to generate complex use case scenarios that define performance bottle necks.
 - Run these scenarios on the actual hardware system.
 - Collect process and display the results, so system level optimizations can be performed.
- Run power scenarios on the actual hardware
 - Generate complex power scenarios for the system.
 - Run these scenarios on the actual hardware.
 - Collect process and display the results, so system issues can be identified and fixed.

34.2 Hardware and Software Requirements

The FPGA must have a microprocessor, Ethernet Port and a Ethernet MAC core on the device. Example Hardware and Softwares are as follows; Xilinx Zynq 7000, SmartFusion2, Xilinx Vivado.

34.3 VisualSim FPGA Interface Kit

VisualSim FPGA Interface Kit provides all the interface files and demo examples. As VisualSim FPGA Interface Kit is provided as completely configurable, users can compile the interface based on their requirements.

34.4 How Interface Works

The Zynq platform has an ARM processor and an FPGA as part of its architecture. This interface uses the ARM processor to enable the interface, so the users can use all of the FPGA for their logic, except for a small layer to communicate with the interface.

The software running on the ARM processor is a lwIP Echo Server. The lwIP Echo server application starts an echo server at port 7. Any data sent to this port is accepted by the FPGA, processed by the FPGA and sent back through this interface.

By default, the program assigns the following settings to the board:

IP Address: 192.168.1.10
Netmask : 255.255.255.0
Gateway : 192.168.1.1
MAC address: 00:0a:35:00:01:02

These settings can be changed in the file main.c.

platform.c implements processor and platform dependent functions.

The file platform_config.h is generated based on the hardware design. It makes two assumptions:

The timer has its interrupt line connected to the interrupt controller, and all the Ethernet peripherals (xps_ethernetlite or xps_ll_temac) accessible from the processor can be used with lwIP.

34.4.1 Procedure to Run Example

- Import the Vivado project into the Vivado tool suite.
- Make the required changes to the project to suit the hardware being used for the demo.
- Compile the project in Vivado.
- Launch the SDK from Vivado, so all the required board support packages are exported to the SDK.
- Import the SDK project into the SDK.
- Make the required changes in the SDK to match the paths of the new packages that were generated for the board being used.
- Compile the software.
- Program the FPGA with the generated bit stream.
- Then launch the software to run on the FPGA and monitor using a telnet from the host PC.

To communicate with FPGA, VisualSim uses UDP socket ports and the user can send packet data from VisualSim to FPGA using VisualSim's "Datagram Writer" library block and receives packet from FPGA using "Datagram Reader" library block. Sample VisualSim model and configuration of Datagram Writer and Datagram Reader is shown in below figure.

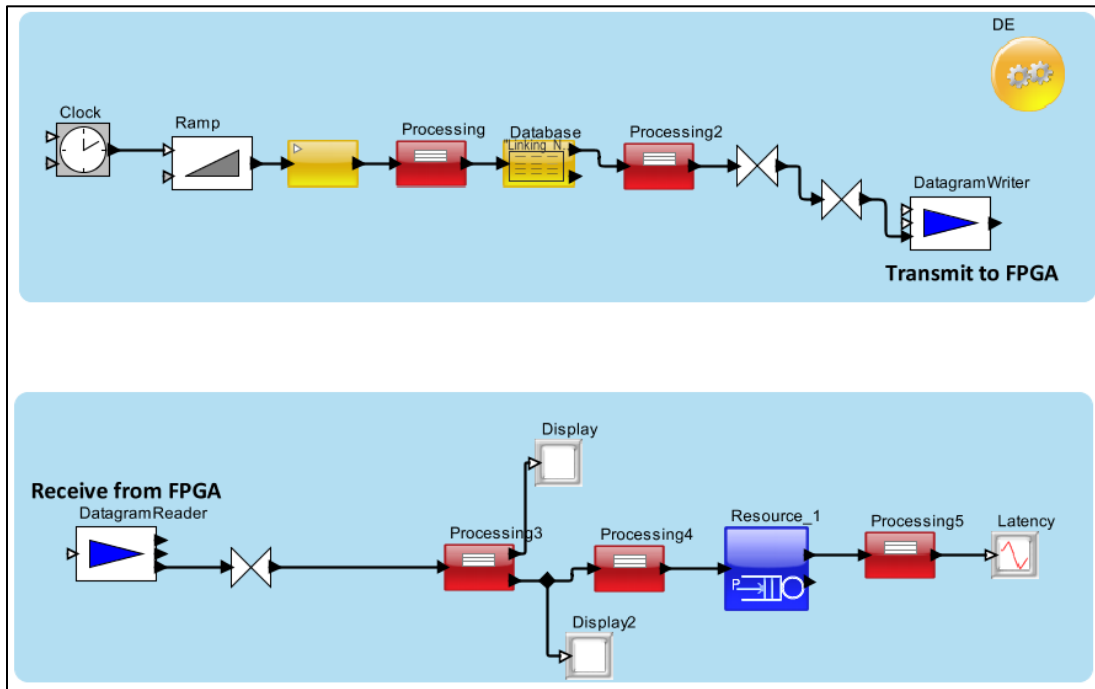


Figure 85.

VisualSim Model

VisualSim Datagram Writer and Datagram Reader play a very important role in VisualSim FPGA interface. Datagram Writer and Datagram Reader should be configured with right Socket number and Remote/Local address.

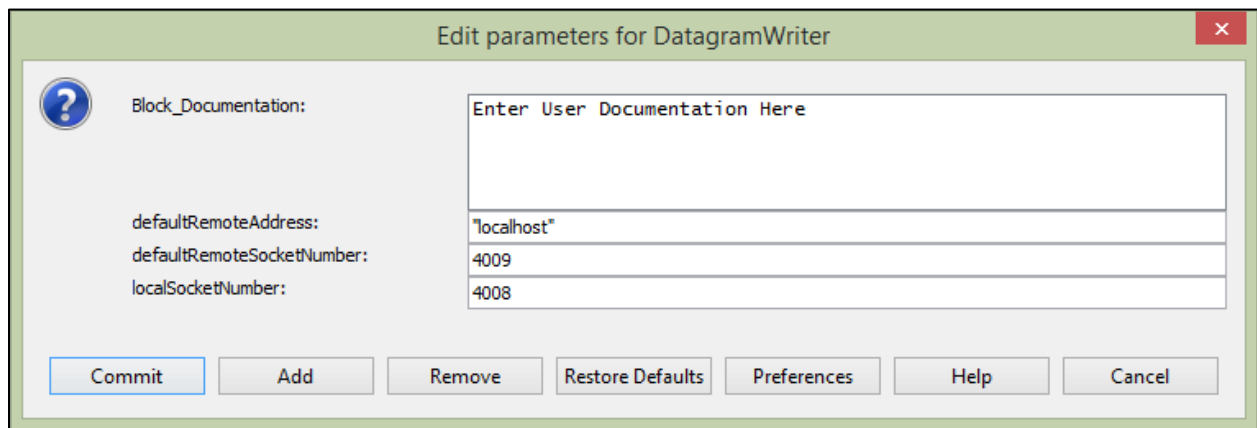


Figure 86.

Configuring Datagram Writer block

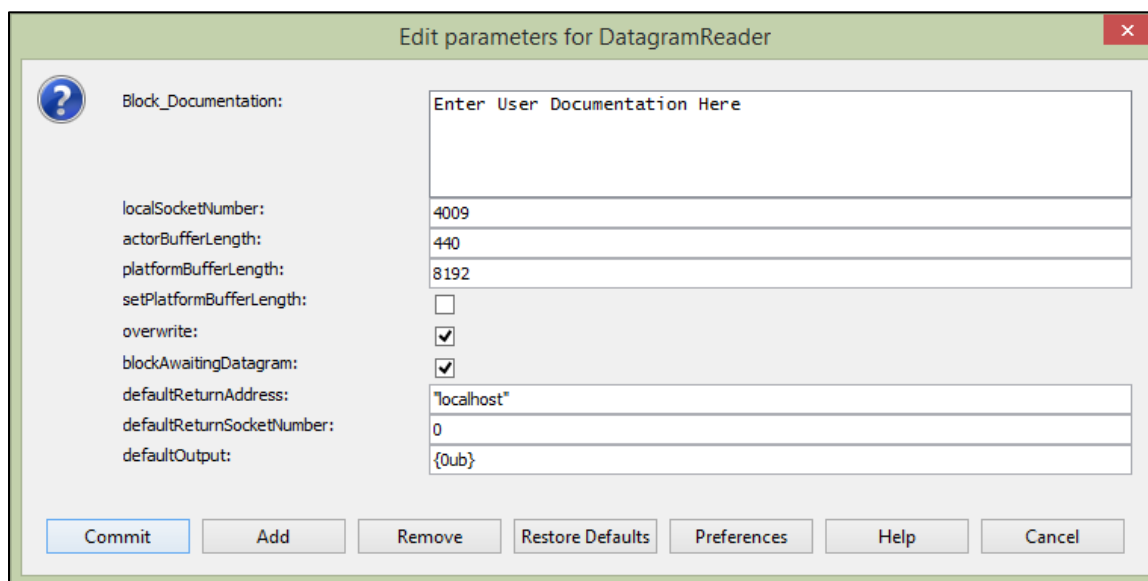


Figure 87. Configuring Datagram Reader

The blocks connected after Datagram Reader correspond to the subsystems that are connected to FPGA platform and does processing based on the data from FPGA. In this example, we have connected a Server that models a simple buffer and it delays the incoming data based on the data size and buffer speed MHz.

34.5 Configuring FPGA and Source Files

We have created a sample Vivado project that works with VisualSim. Import the Vivado project into Xilinx Vivado tool suite. To Import the project, please click on **File > Open Project** and select **mirab_eth.xpr** and click **Ok**. Now the project has been loaded into Vivado design suite.

Now we need to generate bit stream for programming FPGA. Click on Run Implementation from the menu bar **Flow > Run Implementation** or click on the green arrow button on the tool bar.

After successful Implementation Run, generate bit stream by selecting Generate Bitstream Option as shown below.

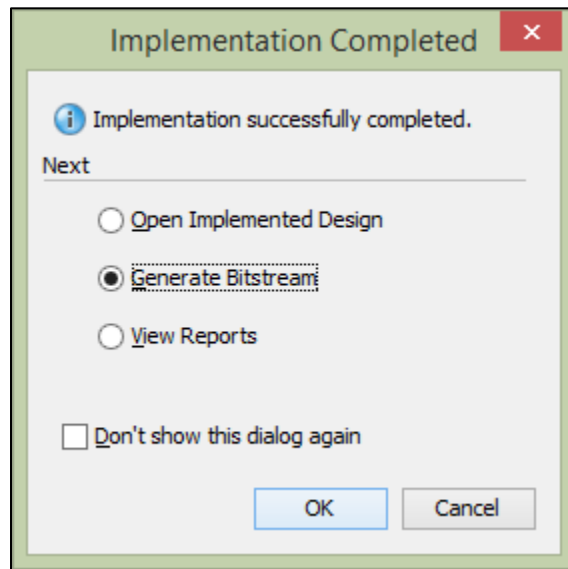


Figure 88.

Generate Bitstream

After successful Bitstream generation, we need to export hardware description file for SDK. To perform this, click on **File → Export Hardware**. As the original project was compiled using Vivado 10.2, users with latest version of Vivado need to upgrade IP and generate a new block diagram. Refer appendix 1 for more details on updating IP and block diagram generation.

When you click on **Export Hardware**, Vivado Suite will open as if you want to generate output products or to Skip generation of output products.

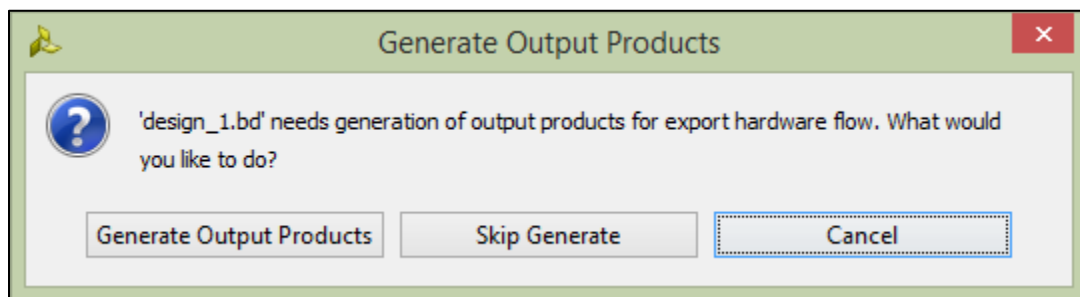
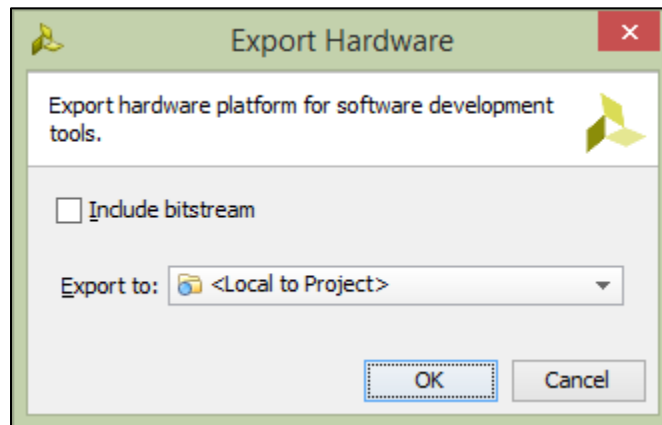


Figure 89.

Generate Bitstream

Click Generate output Products.

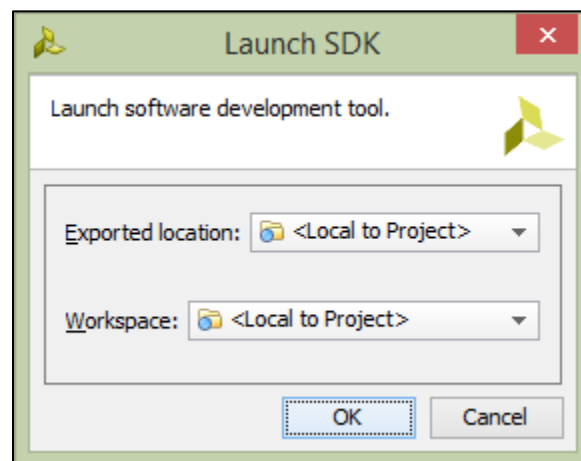
After generating output products for hardware, select the export location as local to project.

**Figure 90.****Export Hardware**

Click OK.

Now we need to launch Xilinx SDK. Make sure that you have installed Xilinx SDK package during installation procedure.

As we need all the board support packages in SDK, launch Xilinx SDK from Vivado Design suite. Click on **File > Launch SDK**. This opens Xilinx SDK, make the required changes in the SDK to match the paths of the new packages that were generated for the board being used. By default you can select the exported location as Local to project as shown below.

**Figure 91.****Launch SDK**

Click on Ok.

The source files are listed under TestApp1, select TestApp1 in Project Explorer as shown below.

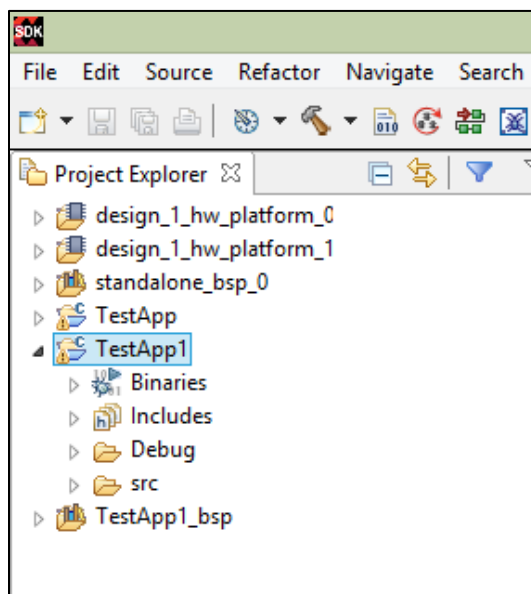


Figure 92.

Project Explorer

To compile source file, click the Run menu from the Menu bar or Run button on the Tool Bar.
Now program the FPGA with the generated bitstream by clicking on **Tools > Program Device**.
Launch the software to run on the FPGA and monitor using a telnet from the host PC.

34.6 Possible Errors

1. Failed to create a new socket on port {Port Number}.
Solution: This error appears if the Datagram Writer local socket number and Datagram Reader local socket number are same. Make sure that both are different.
2. Blank Report display.
Solution: This is due to incorrect remote address/IP address or Remote socket number defined in Datagram Writer.

34.7 Appendix

1. For the users with Vivado design suite above 10.2, follow the steps mentioned below to upgrade IP and to generate new block diagram.

If you are running Vivado design suite above 10.2 you may receive a message as mentioned below while performing **Export Hardware**.



Figure 93.

Cannot Export Hardware

At this point, click on Open Block Design from the flow navigator as shown below.

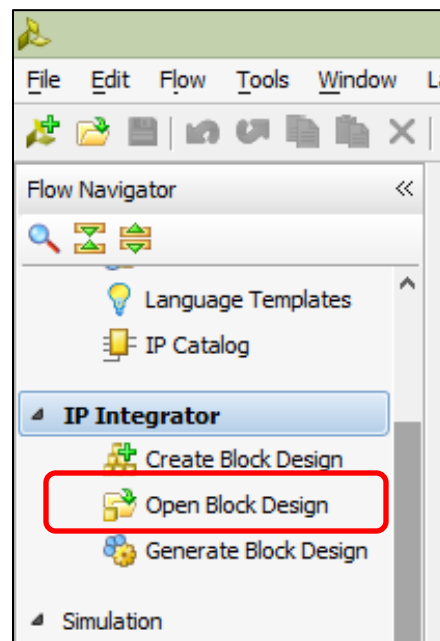


Figure 94.

Open Block Design

Click on **Show IP Status**

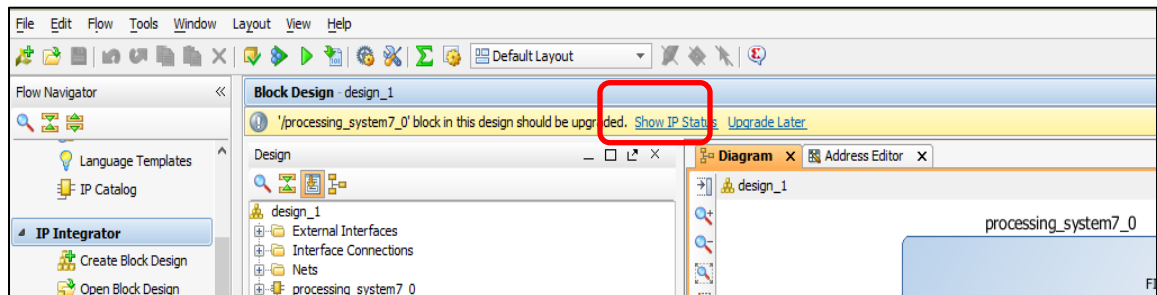


Figure 95.

Show IP Status

Click on **Update Selected**

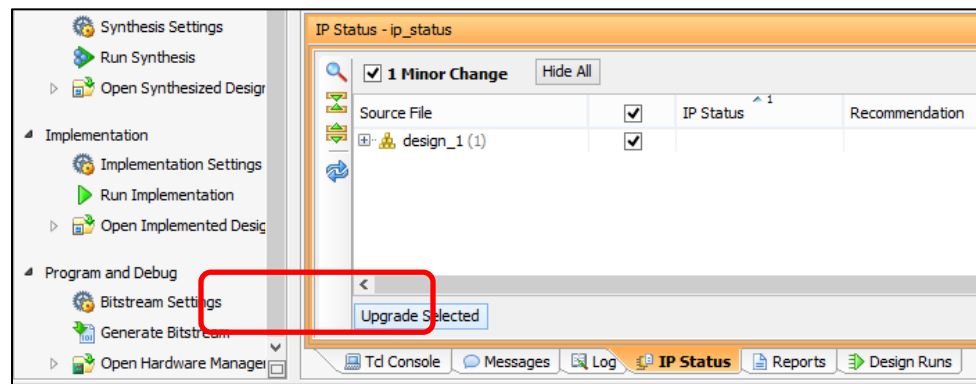


Figure 96.

Update Selected

This updates the existing IPs.

35 Qemu Guide

For Ubuntu 18.04

35.1 Steps to install qemu on Ubuntu

- 1) Download and install QEMU from source code.
- 2) Site referred - [https://wiki.qemu.org/Hosts/Linux#Getting the source code](https://wiki.qemu.org/Hosts/Linux#Getting_the_source_code).

```
sudo apt-get install git libglib2.0-dev libfdt-dev libpixman-1-dev zlib1g-dev
sudo apt-get install git-email
sudo apt-get install libaio-dev libbluetooth-dev libbrlapi-dev libbz2-dev
sudo apt-get install libcap-dev libcap-ng-dev libcurl4-gnutls-dev libgtk-3-dev
sudo apt-get install libibverbs-dev libjpeg8-dev libncurses5-dev libnuma-dev
sudo apt-get install librbd-dev librdmacm-dev
sudo apt-get install libsasl2-dev libsdl1.2-dev libseccomp-dev libsnappy-dev libssh2-1-dev
sudo apt-get install libvde-dev libvdeplug-dev libvte-2.91-dev libxen-dev libz02-dev
sudo apt-get install valgrind xfslibs-dev
sudo apt-get install libnfs-dev libiscsi-dev
```

- 3) git clone [git://git.qemu-project.org/qemu.git](https://git.qemu-project.org/qemu.git)
- 4) # Switch to the QEMU root directory.
- 5) cd qemu
- 6) # Prepare a native debug build.
- 7) mkdir -p bin/debug/native
- 8) cd bin/debug/native
- 9) # Configure QEMU and start the build. This will build for all the target list – ARM, x86-64, RISC-V, PowerPC, SPARC etc.
- 10) ../../configure --enable-debug
- 11) make
- 12) # Return to the QEMU root directory.
- 13) cd ../../..

-----*****-----

35.2 Download benchmark to run the program

- 1) Download the STREAM benchmarking suite:
 - a. Download stream.c from here: <https://www.cs.virginia.edu/stream/FTP/Code/stream.c>
 - b. Download mysecond.c from here (not required, but just to be safe): <https://www.cs.virginia.edu/stream/FTP/Code/mysecond.c>
- 2) Or create your own C program and run against the different architectures in the next steps.

-----*****-----

35.3 Steps to compile and Run C program for x86 and ARM

- *Test Code used – **stream.c** -download from the above or create your own test code*
- *Connect to the test code directory*
- *Create log-file for either of the architectures –x86 or ARM*

1) x86

a) Compile the program

```
elc1@ubuntu:~/QEMU/tests$ gcc stream.c -o stream-x86
```

user defined test code: gcc filename.c -o filename

b) Run the program

```
elc1@ubuntu:~/QEMU/tests$ ./stream -x86
```

```
-----
STREAM version $Revision: 5.10 $
-----
```

```
This system uses 8 bytes per array element.
-----
```

```
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
```

The *best* time for each kernel (excluding the first iteration) will be used to compute the reported bandwidth.

Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 197978 microseconds.
(= 197978 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.

WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	5549.6	0.029829	0.028831	0.032965
Scale:	5217.2	0.031839	0.030668	0.033178
Add:	7839.0	0.032011	0.030616	0.034741
Triad:	7401.3	0.033327	0.032427	0.033997

Solution Validates: avg error less than 1.000000e-13 on all three arrays

c) Check the type of the binary thus generated (output is shown below)

elc1@ubuntu:~/QEMU/tests\$ **file stream-x86**

stream-x86: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux
3.2.0, BuildID[sha1]=7ff9b39569aef2b4fea79b0913eb1674335d11f3, not
stripped

**d) Generate the instruction trace along with running the executable on
emulated x86 architecture.**

**-d flag specifies the type of logs required (it will help to generate
the log). The entire list can be obtained by specifying -d help. We
will use -d strace.**

For assembly code – in_asm

**-D flag specifies the location where the logs should be written.
Use the following command (It will generate the logfile-x86-64)**

The entire list can be obtained by specifying -d help

```
elc1@ubuntu:~/QEMU/tests$  
/home/elc1/QEMU/qemu/bin/debug/native/x86_64-linux-user/qemu-x86_64  
-d help  
Log items (comma separated):  
out_asm      show generated host assembly code for each compiled TB  
in_asm       show target assembly code for each compiled TB  
op           show micro ops for each compiled TB  
op_opt       show micro ops after optimization  
op_ind       show micro ops before indirect lowering  
int          show interrupts/exceptions in short format  
exec         show trace before each executed TB (lots of logs)  
cpu          show CPU registers before entering a TB (lots of logs)  
fpu          include FPU registers in the 'cpu' logging  
mmu          log MMU-related activities  
pcall        x86 only: show protected mode far calls/returns/exceptions  
cpu_reset    show CPU state before CPU resets  
unimp        log unimplemented functionality  
guest_errors log when the guest OS does something invalid (eg  
accessing a  
non-existent register)  
page         dump pages at beginning of user mode emulation  
nochain      do not chain compiled TBs so that "exec" and "cpu" show  
complete traces  
strace       log every user-mode syscall, its input, and its result  
trace:PATTERN enable trace events
```

Use "-d trace:help" to get a list of trace events.

```
elc1@ubuntu:~/QEMU/tests$  
/home/elc1/QEMU/qemu/bin/debug/native/x86_64-linux-user/qemu-  
x86_64 -d strace -D /home/elc1/QEMU/tests/logfile-stream-x86_64  
./stream-x86
```

Explanation on the above command

Path to qemu-x86_64: /home/elc1/QEMU/qemu/bin/debug/native/x86_64-linux-user/qemu-x86_64

Type of Log: -d strace

Path to Output file: -D /home/elc1/QEMU/tests/logfile-stream-x86_64

*This will create the logfile in the given location. For example here -
/home/elc1/QEMU/tests/*

Output file name: logfile-stream-x86_64

Executable file name: ./stream-x86

2) ARM -32bit

Install arm compiler: `sudo apt-get install gcc-arm-none-eabi`

a) Compile the program

```
elc1@ubuntu:~/QEMU/tests$ arm-none-eabi-gcc --specs=rdimon.specs  
filename.c -o filename-arm -static
```

b) Check the type of the binary thus generated (output is shown below)

```
elc1@ubuntu:~/QEMU/tests$ file filename-arm  
filename -arm: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV),  
statically linked, with debug_info, not stripped
```

c) (not necessary – if want to see the output)Execute the binary on emulated ARM architecture using qemu

```
/home/elc1/QEMU/qemu/bin/debug/native/arm-linux-user/qemu-arm ./  
filename -arm
```

d) Generate the instruction trace along with running the executable on emulated x86 architecture.

-d flag specifies the type of logs required (it will help to generate the log). The entire list can be obtained by specifying -d help. We will use -d in_asm.

-D flag specifies the location where the logs should be written.

Use the following command(It will generate the logfile-arm)

```
elc1@ubuntu:~/QEMU/tests$  
/home/elc1/QEMU/qemu/bin/debug/native/arm-linux-user/qemu-arm -d  
in_asm -D /home/elc1/QEMU/tests/logfilename ./filename-arm
```

Explanation on the above command

Path to qemu-arm: /home/elc1/QEMU/qemu/bin/debug/native/arm-linux-user/qemu-arm

Type of Log: -d in_asm

Path to Output file: /home/elc1/QEMU/tests/

this will create the logfile in the given location. For example here - /home/elc1/QEMU/tests/

Output file name: logfilename

Executable file name: ./filename-arm

3) ARM -64bit

Install arm-64bit compiler: `sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu`

I. compile the program

`elc1@ubuntu:~/QEMU/tests$ aarch64-linux-gnu-gcc hello.c -o hello`

II. Check the type of the binary thus generated (output is shown below)

a. compile the program *dynamically* – (already done in step I.)

`elc1@ubuntu:~/QEMU/tests$ file hello`

64-bit LSB executable, ARM aarch64, version 1 (SYSV),
dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, for
GNU/Linux 3.7.0,
BuildID[sha1]=0a161541c986a6a5ea1cea24eb19579ce9f64fdb, not
stripped

b. compile the program *statically*

`elc1@ubuntu:~/QEMU/tests$ aarch64-linux-gnu-gcc hello.c -o
hello -static`

`elc1@ubuntu:~/QEMU/tests$ file hello`

hello: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV),
statically linked, for GNU/Linux 3.7.0,
BuildID[sha1]=dc9c2aac0c174d49749a1b3b47175f7a759547da, not
stripped

III. (not necessary – if want to see the output)Execute the binary on emulated ARM architecture using qemu

`$ /home/elc1/QEMU/qemu/bin/debug/native/aarch64-linux-user /qemu-aarch64./hello`

IV. Generate the instruction trace along with running the executable on emulated x86 architecture.

-d flag specifies the type of logs required (it will help to generate the log). The entire list can be obtained by specifying -d help. We will use -d in_asm.

-D flag specifies the location where the logs should be written.

Use the following command (It will generate the logfile-arm)

```
elc1@ubuntu:~/QEMU/tests$
/home/elc1/QEMU/qemu/bin/debug/native/aarch64-linux-user /qemu-
aarch64 -d in_asm -D /home/elc1/QEMU/tests/logfile-arm64 ./hello
Hello world! - this will create the logfile in the given location. For example
here - /home/elc1/QEMU/tests/
```

4) PowerPc -32bit

Install arm compiler: `sudo apt-get install gcc-4.8-powerpc-linux-gnu`

I. compile the program

```
elc1@ubuntu:~/QEMU/tests$ powerpc-linux-gnu-gcc-4.8 hello.c -o hello
```

II. Check the type of the binary thus generated (output is shown below)

a. compile the program *dynamically* – already done in above step

```
elc1@ubuntu:~/QEMU/tests$ file hello
hello: ELF 32-bit MSB executable, PowerPC or cisco 4500, version
1 (SYSV), dynamically linked, interpreter /lib/ld.so.1, for GNU/Linux
3.2.0,
BuildID[sha1]=2d8db2bb6dc97b29db5527e284bb0d56cf71f3b6, not
stripped
```

b. compile the program *statically*

```
elc1@ubuntu:~/QEMU/tests$ powerpc-linux-gnu-gcc-4.8 hello.c -
o hello -static
elc1@ubuntu:~/QEMU/tests$ file hello
```

```
hello: ELF 32-bit MSB executable, PowerPC or cisco 4500, version
1 (SYSV), statically linked, for GNU/Linux 3.2.0,
BuildID[sha1]=081f2652a0604602668cecf1f1011f0bd17b5756, not
stripped
```

- III. **(not necessary – if want to see the output)Execute the binary on emulated architecture using qemu**

```
$ /home/elc1/QEMU/qemu/bin/debug/native/ppc-linux-user/qemu-ppc
./hello
```

- IV. **Generate the instruction trace along with running the executable on emulated x86 architecture.**

-d flag specifies the type of logs required (it will help to generate the log). The entire list can be obtained by specifying -d help. We will use -d in_asm.

-D flag specifies the location where the logs should be written.

Use the following command(It will generate the logfile-arm)

```
elc1@ubuntu:~/QEMU/tests$
/home/elc1/QEMU/qemu/bin/debug/native/ppc-linux-user/qemu-ppc -d
in_asm -D /home/elc1/QEMU/tests/logfile-pp32 ./hello
Hello world! - this will create the logfile in the given location.For example
here - /home/elc1/QEMU/tests/
```

5) PowerPc -64bit

Install arm compiler: `sudo apt-get install gcc-4.8-powerpc64le-linux-gnu`

- I. **compile the program**

```
elc1@ubuntu:~/QEMU/tests$ powerpc64le-linux-gnu-gcc-4.8 hello.c -o
hello-ppc64
```

- II. **Check the type of the binary thus generated (output is shown below)**

a. compile the program dynamically- (already done in above step)

```
elc1@ubuntu:~/QEMU/tests$ file hello-ppc64
hello-ppc64: ELF 64-bit LSB executable, 64-bit PowerPC or cisco
7500, version 1 (GNU/Linux), dynamically linked for GNU/Linux
3.2.0,
BuildID[sha1]=1a89598dc9eccc38b264d9a0e22741e422866dbd,
not stripped
```

b. compile the program *statically*

```
elc1@ubuntu:~/QEMU/tests$ powerpc64le-linux-gnu-gcc-4.8
hello.c -o hello-ppc64 -static
```

```
elc1@ubuntu:~/QEMU/tests$ file hello-ppc64
hello-ppc64: ELF 64-bit LSB executable, 64-bit PowerPC or cisco
7500, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0,
BuildID[sha1]=1a89598dc9eccc38b264d9a0e22741e422866dbd,
not stripped
```

III. (not necessary – if want to see the output)Execute the binary on emulated ARM architecture using qemu

```
$ /home/elc1/QEMU/qemu/bin/debug/native/ ppc64le-linux-user/qemu-
ppc64le . / hello-ppc64
```

IV. Generate the instruction trace along with running the executable on emulated x86 architecture.

-d flag specifies the type of logs required (it will help to generate the log). The entire list can be obtained by specifying -d help. We will use -d in_asm.

-D flag specifies the location where the logs should be written.

Use the following command(It will generate the logfile-arm)

```
elc1@ubuntu:~/QEMU/tests$
/home/elc1/QEMU/qemu/bin/debug/native/ ppc64le-linux-
user/qemu-ppc64le -d in_asm -D
/home/elc1/QEMU/tests/logfile-ppc64 ./hello-ppc64
Hello world! - this will create the logfile in the given location.For
example here - /home/elc1/QEMU/tests/
```

36 VisualSim Gem5-Arm Integration

Note: To build gem5, you will need the following software:

- `git`
- `gcc/g++` (version ≥ 5)
- `Python` (3.6+),
- `SCons`(3.0 or greater),
- `SWIG`,
- `zlib`,
- `m4`, and lastly
- `protobuf` (2.1+) if you want trace capture and playback support.

Please see http://www.gem5.org/documentation/general_docs/building for more details concerning the minimum versions of the aforementioned tools.

The below information specify the gem5 version and the gem5 build on OS

```
-----
Gem5 - :
Branches: stable - v21.0.0.0
package - git clone https://gem5.googlesource.com/public/gem5
OS - Centos-8.3
python version - python 3.6
gcc version - gcc 8.3
```

36.1 Installation steps for Ubuntu OS

Assumption: The user has installed Java version 14 or later and VisualSim Architect.

1. `$ sudo apt install git`
2. `$ sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python3-dev python python3-six python-is-python3 libboost-all-dev pkg-config`
3. `$ sudo apt-get install swig`
4. Connect to gem5 directory and type
 - a. `$ scons build/ARM/gem5.opt -j2`

36.2 Installation steps for CentOS 8.3

Assumption: The user has installed Java version 14 or later and VisualSim Architect.

1. `$ sudo yum groupinstall "Development Tools"`
2. `$ sudo dnf update`

3. \$ sudo dnf config-manager --set-enabled powertools
4. \$ sudo yum install swig
5. \$ sudo yum install zlib-devel
6. \$ python --version
7. Python 3.6.8
8. Note: (optional) If python --version does not points to python3.6 or more, then create a symbolic links as follow -:
 - a. \$ which python
/usr/bin/python
 - b. \$ which python3.6
/usr/bin/python3.6
 - c. \$ sudo ln -s /usr/bin/python3.6 /usr/bin/python
9. \$ sudo yum install python3-devel
10. \$ sudo python -m pip install scon
11. \$ sudo dnf -y install protobuf-devel

On cmdline: - Installing:

a.	protobuf-devel	x86_64	3.5.0-13.el8	powertools	357 k
Installing dependencies:					
b.	protobuf	x86_64	3.5.0-13.el8	appstream	892 k
c.	protobuf-compiler	x86_64	3.5.0-13.el8	powertools	789 k
12. \$ sudo dnf install https://extras.getpagespeed.com/release-el8-latest.rpm
13. \$ sudo dnf install gperftools-devel
14. \$ sudo yum install libpng-devel
15. \$ sudo yum -y install hdf5-devel
16. Connect to gem5 directory and type
 - a. \$ scon build/ARM/gem5.opt -j2

-----*****-----

36.3 Run the Gem5 with Visualsim

- 1) Connect to gem5 directory on terminal and type
\$./interactive_sim.py

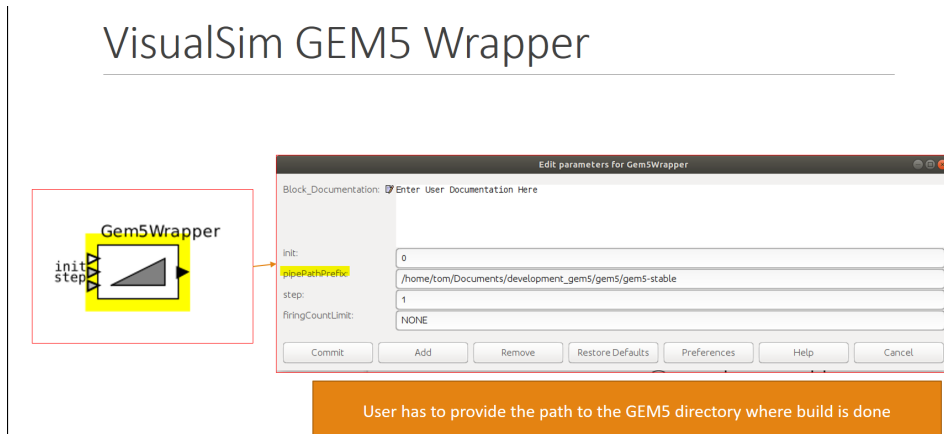
```
[elc@localhost gem5]$ ./interactive_sim.py
starting the simulation!!!
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 21.0.0.0
gem5 compiled Apr 23 2021 11:17:02
gem5 started Apr 30 2021 03:44:49
gem5 executing on localhost.localdomain, pid 3124
command line: ./build/ARM/gem5.opt --debug-start=1 --debug-flags=DRAM --interactive ./configs/example/se_vs.py --c
pu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1l size=16kB --l1d size=16kB --mem-type=DDR3
_1600 8x8 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfile ../MiBench/consumer/jp
eg/output_large_encode.jpeg ../MiBench/consumer/jpeg/input_large.ppm'

warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
Global frequency set at 1000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
Redirecting stdout
```

- 2) Now open the model from VisualSim - [demo/interfaces/gem5/gem5_demo.xml](#)

- 3) Specify the gem5 folder path in **GEM5Wrapper** parameter **pipePathPrefix** in the model.

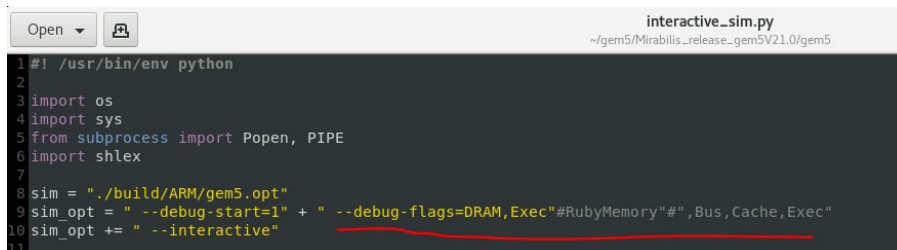


- 4) Run the model.

-----*****-----

36.4 Trace Generation from Gem5:

- 1) Generate trace using debug flags – In gem5 these are the flags used
 - a. **DRAM** debug flag – gives memory details
 - b. **Exec** debug flag - based on the exact instruction the CPU is executing, shows details of how each instruction is executed by the simulated CPU.
- 2) Edit interactive_sim.py. Enable the **Exec** flag as shown below (**DRAM** flag is enabled by default).



- 3) Now run the interactive_sim.py again (same steps as described in section 49.3)
- 4) The output is generated in file **temp_pipe**, in gem5 directory.

-----*****-----

36.5 Debug Gem5 using gdb:

- 1) \$ gdb --args ./build/ARM/gem5.opt
- 2) (gdb) run --debug-break=1000 --debug-flags=DRAM --interactive ./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-

```
type=DDR3_1600_8x8 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfile
../MiBench/consumer/jpeg/output_large_encode.jpeg ../MiBench/consumer/jpeg/input_large.ppm'
```

- 3) Run the visualsim model
- 4) Debug the code using gdb commands such as (n, p, f, bt, step, b main, c)
- 5) After debugging type **continue** or **c** on the terminal, for rest of iteration to finished up
- 6) Plots will be displayed.
- 7) For more information refer:

https://www.gem5.org/documentation/general_docs/debugging_and_testing/debugging/debugger_based_debugging

design

command line

```
elc1@ubuntu:~/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03$ gdb --args
./build/ARM/gem5.opt
Reading symbols from ./build/ARM/gem5.opt...done.
(gdb) run --debug-break=1000 --debug-flags=DRAM --interactive
./configs/example/interactive_se.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --
sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_x64 -c
../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfile
../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
Starting program: /home/elc1/gem5/gem5-ptolemy-master/gem5-
stable_2015_09_03/build/ARM/gem5.opt --debug-break=1000 --debug-flags=DRAM --
interactive ./configs/example/interactive_se.py --cpu-type=TimingSimpleCPU --cpu-
clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-
type=DDR3_1600_x64 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -
progressive -opt -outfile ../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
```

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
gem5 Simulator System. <http://gem5.org>
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Sep 22 2020 19:37:46
gem5 started Sep 23 2020 00:46:20

```
gem5 compiled Sep 22 2020 19:37:46
gem5 started Sep 23 2020 00:46:20
gem5 executing on ubuntu
command line: /home/elc1/gem5/gem5-ptolemy-master/gem5-
stable_2015_09_03/build/ARM/gem5.opt --debug-break=1000 --debug-flags=DRAM --
interactive ./configs/example/interactive_se.py --cpu-type=TimingSimpleCPU --cpu-
clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-
type=DDR3_1600_x64 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -
progressive -opt -outfile ../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
```

warn: need to stop all queues
Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)

O: system.mem_ctrls: Memory capacity 536870912 (536870912) bytes
O: system.mem_ctrls: Row buffer size 8192 bytes with 128 columns per row buffer
O: system.remote_gdb.listener: listening for remote gdb #0 on port 7000

Redirecting stdout

command line

warn: need to stop all queues
 Global frequency set at 1000000000000 ticks per second
 warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
 0: system.mem_ctrls: Memory capacity 536870912 (536870912) bytes
 0: system.mem_ctrls: Row buffer size 8192 bytes with 128 columns per row buffer
 0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
 Redirecting stdout
 ---*Now run the VisualSim here*

command line – give gdb commands

<p>Redirecting stdout Iteration: 0</p> <p>Program received signal SIGTRAP, Trace/breakpoint trap. 0x00007ffff5e5a237 in kill () at ../sysdeps/unix/syscall-template.S:78 78 ../sysdeps/unix/syscall-template.S: No such file or directory. (gdb) b main Breakpoint 1 at 0x555555de69e0: file build/ARM/sim/main.cc, line 42. (gdb) n kill () at ../sysdeps/unix/syscall-template.S:79 79 in ../sysdeps/unix/syscall-template.S (gdb) n GlobalEvent::BarrierEvent::process (this=0x5555584a5280) at build/ARM/sim/global_event.cc:136 136 globalBarrier(); (gdb) n 137 } (gdb) EventQueue::serviceOne (this=this@entry=0x555558498320) at build/ARM/sim/eventq.cc:229 229 if (event->isExitEvent()) { (gdb) p eventQueue No symbol "eventQueue" in current context. (gdb) n 238 if (event->flags.isSet(Event::AutoDelete) && !event- >scheduled())</p>	<p>238 if (event->flags.isSet(Event::AutoDelete) && !event->scheduled()) (gdb) print SimObject::find("system.cpu") \$1 = (SimObject *) 0x555558bedb80 (gdb) print (BaseCPU *) SimObject::find("system.cpu") \$2 = (BaseCPU *) 0x555558bedb80 (gdb) \$3p instCnt Undefined command: "\$3p". Try "help". (gdb) p \$3->instCnt History has not yet reached \$3. (gdb) print (BaseCPU *) SimObject::find("system.cpu") \$3 = (BaseCPU *) 0x555558bedb80 (gdb) p \$3->instCnt \$4 = 0 (gdb) (gdb) c Continuing.</p>	<p>(gdb) c Continuing. Iteration 0 finished Iteration: 1 Iteration 1 finished Iteration: 2 Iteration 2 finished Iteration: 3 Iteration 3 finished Iteration: 4 Iteration 4 finished Iteration: 5 Iteration 5 finished Iteration: 6 Iteration 6 finished Iteration: 7 Iteration 7 finished Iteration: 8 Iteration 8 finished Iteration: 9 Iteration 9 finished Iteration: 10 Iteration 10 finished</p>
--	---	---



To continue the
execution



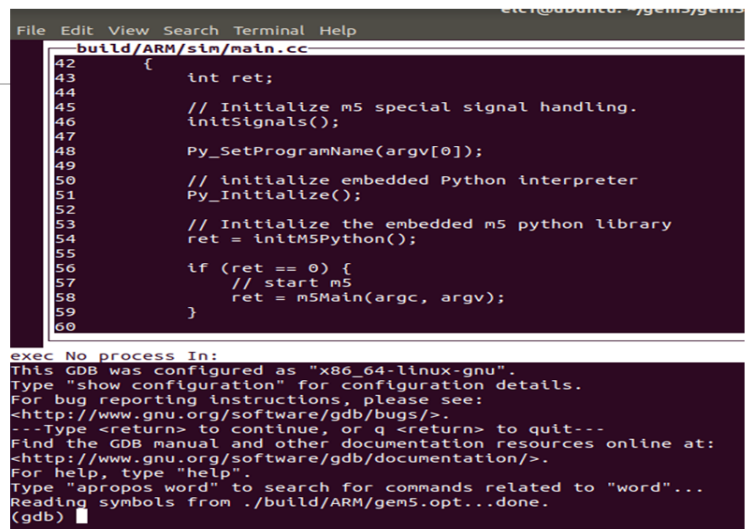
Show plots

36.6 Debug Gem5 using graphical gdb:

- 1) `$ gdb -tui --args ./build/ARM/gem5.opt`
- 2) `(gdb) run --debug-break=1000 --debug-flags=DRAM --interactive ./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_8x8 -c ./MiBench/consumer/jpeg/jpeg-6a/cjpeg -o 'dct int -progressive -opt -outfile ./MiBench/consumer/jpeg/output_large_encode.jpeg' ./MiBench/consumer/jpeg/input_large.ppm'`
- 3) Run the visualsim model
- 4) Debug the code using gdb commands such as (n, p, f, bt, step, b main, c)
- 5) After debugging type **continue** or **c** on the terminal, for rest of iteration to finished up
- 6) Plots will be displayed.
- 7) For more information refer the site : <http://beej.us/guide/bggdb/#qref>
- 8) **NOTE:** All the normal **gdb** commands will work in GUI mode, and additionally the arrow keys and pgup/pgdown keys will scroll the source window (when it has focus, which it does by default).

Graphical gdb

elc1@ubuntu:~/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03\$ `gdb -tui --args ./build/ARM/gem5.opt`

```

File Edit View Search Terminal Help
build/ARM/sim/main.cc
42 {
43     int ret;
44
45     // Initialize m5 special signal handling.
46     initSignals();
47
48     Py_SetProgramName(argv[0]);
49
50     // initialize embedded Python interpreter
51     Py_Initialize();
52
53     // Initialize the embedded m5 python library
54     ret = initM5Python();
55
56     if (ret == 0) {
57         // start m5
58         ret = m5Main(argc, argv);
59     }
60
exec No process in:
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
---Type <return> to continue, or q <return> to quit---
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./build/ARM/gem5.opt...done.
(gdb)

```

```

elc1@ubuntu: ~/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03
File Edit View Search Terminal Help
59 }
60

multi.thre Thread 0x7ffff7fdd4 In:
(gdb)
(gdb) run --debug-break=1000 --debug-flags=DRAM --interactive ./configs/example/interactive_se.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz
--sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_x64 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -pro
gressive -opt -outfile ../MiBench/consumer/jpeg/output_large_encode.jpeg ../MiBench/consumer/jpeg/input_large.ppm'
Starting program: /home/elc1
/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03/build/ARM/gem5.opt --debug-break=1000 --debug-flags=DRAM --interactive ./configs/example/inte
ractive_se.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_x64 -
c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfile ../MiBench/consumer/jpeg/output_large_encode.jpeg ../MiBench/
consumer/jpeg/input_large.ppm'
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Sep 22 2020 19:37:46
gem5 started Sep 23 2020 01:26:24
gem5 executing on ubuntu
command line: /home/elc1/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03/build/ARM/gem5.opt --debug-break=1000 --debug-flags=DRAM --interactiv
e ./configs/example/interactive_se.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --
mem-type=DDR3_1600_x64 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfile ../MiBench/consumer/jpeg/output_large
_encode.jpeg ../MiBench/consumer/jpeg/input_large.ppm'

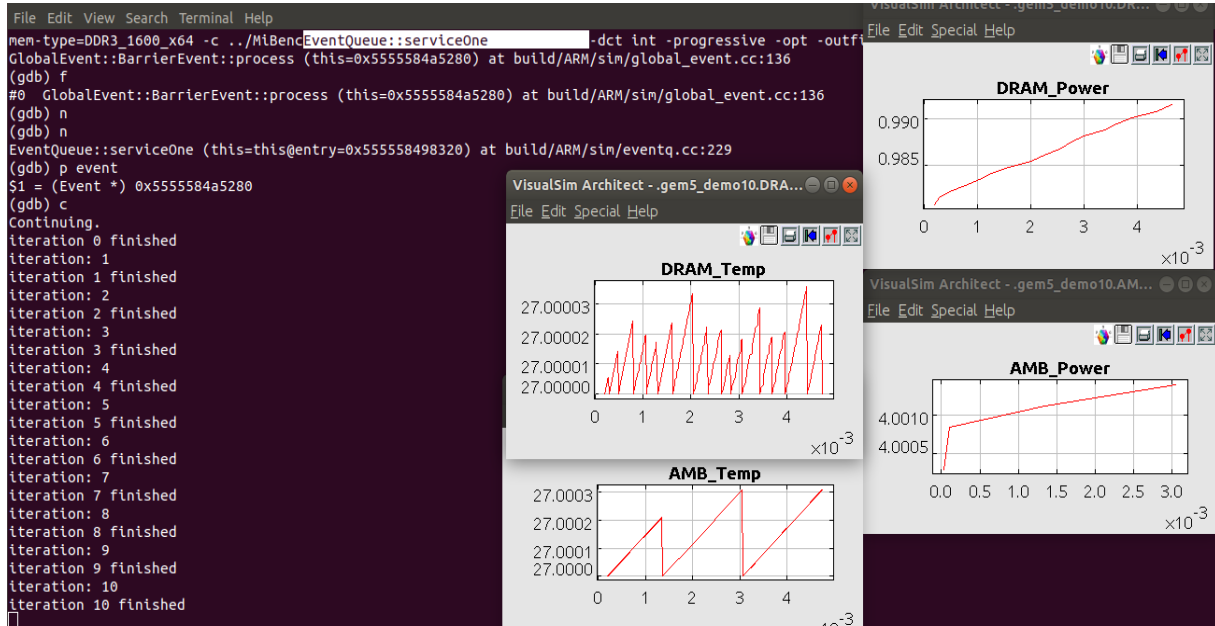
warn: need to stop all queues
Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.mem_ctrls: Memory capacity 536870912 (536870912) bytes
0: system.mem_ctrls: Row buffer size 8192 bytes with 128 columns per row buffer
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7001
Redirecting stdout

```

```

elc1@ubuntu: ~/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03
File Edit View Search Terminal Help
60 build/ARM/sim/eventq.cc
221 }
222
223 // handle action
224 if (!event->squashed()) {
225     // forward current cycle to the time when this event occurs.
226     setCurTick(event->when());
227
228     event->process();
229     if (event->isExitEvent()) {
230         assert(!event->flags.isSet(Event::AutoDelete) ||
231             !event->flags.isSet(Event::IsMainQueue)); // would be silly
232         return event;
233     }
234     } else {
235         event->flags.clear(Event::Squashed);
236     }
237
238     if (event->flags.isSet(Event::AutoDelete) && !event->scheduled())
239         delete event;
240
241 e ./configs/example/interactive_se.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --
mem-type=DDR3_1600_x64 -c ../MiBench/EventQueue:serviceOne -dct int -progressive -opt -outfile ../MiBench/c
kill () at ../sysdeps/unix/syscall-template.S:79
(gdb) n
GlobalEvent::BarrierEvent::process (this=0x5555584a5280) at build/ARM/sim/global_event.cc:136
(gdb) f
#0 GlobalEvent::BarrierEvent::process (this=0x5555584a5280) at build/ARM/sim/global_event.cc:136
(gdb) n
(gdb) n
EventQueue::serviceOne (this=this@entry=0x555558498320) at build/ARM/sim/eventq.cc:229
(gdb) p event
$1 = (Event *) 0x5555584a5280
(gdb)

```



-----*****-----

36.7 Running the Gem5 in System Call Emulation (SE) Mode

The SE mode simulation focuses on the CPU and memory system, and does not emulate the entire system. In this mode, one only needs to specify the binary file to be simulated. This binary file can be statically/dynamically linked. `configs/examples/se.py` is used for configuring and running simulations in this mode. What follows is probably the simplest example of how to use `se.py`. The binary file to simulate is specified with option `-c`.

- 1) `cd` to the `gem5` directory on terminal and use the below command to run the `hello arm` executable.

```
a) $ ./build/ARM/gem5.opt ./configs/example/se.py -c ./tests/test-progs/hello/bin/arm/linux/hello
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.
gem5 compiled Mar  2 2014 00:06:39
gem5 started Mar  4 2014 10:52:10
gem5 executing on $
command line: ./build/ARM/gem5.opt ./configs/example/se.py -c ./tests/test-
progs/hello/bin/arm/linux/hello
Global frequency set at 1000000000000 ticks per second
O: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
info: Increasing stack size by one page.
Hello world!
Exiting @ tick 3233000 because target called exit()
```

- 2) For statistics refer to this location → `/home/elc1/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03/m5out/stats.txt`
- 3) Detailed Information - http://mail.gem5.org/Running_gem5
- 4) Create your own executable for arm (additional information for trying out more experiments)

Install ARM compiler \$ **`sudo apt-get install gcc-arm-linux-gnueabi gcc-arm-linux-gnueabihf gcc-aarch64-linux-gnu`**

- a. Compile the C program : \$ **`arm-linux-gnueabi-gcc filename.c -o filename-arm -static`**
- b. \$ **`file filename`**
- c. Run the gem5 \$ **`./build/ARM/gem5.opt ./configs/example/se.py -c path to the arm executables`**

-----*****-----

36.8 Run the Gem5 (SE mode) in Multi-Core

- 1) cd to the gem5 directory on terminal and use the below command to run the hello arm executable.

```
elc1@ubuntu:~/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03$ ./build/ARM/gem5.opt
configs/example/se.py --num-cpus=2 -c './tests/test-progs/hello/bin/arm/linux/hello;./tests/test-
progs/hello/bin/arm/linux/hello'
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Sep 22 2020 19:37:46
gem5 started Nov 6 2020 00:42:37
gem5 executing on ubuntu
command line: ./build/ARM/gem5.opt configs/example/se.py --num-cpus=2 -c './tests/test-
progs/hello/bin/arm/linux/hello;./tests/test-progs/hello/bin/arm/linux/hello'
Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
O: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
O: system.remote_gdb.listener: listening for remote gdb #1 on port 7001
*** REAL SIMULATION ***
info: Entering event queue @ 0. Starting simulation...
Hello world!
Hello world!
Exiting @ tick 2924500 because target called exit()
```

- 2) For statistics refer → m5_out/Stats.txt
- 3) Create your own executable for arm (additional information for trying out more experiments)

```
Install ARM compiler
$ sudo apt-get install gcc-arm-linux-gnueabi gcc-aarch64-linux-gnu
```

- a. Compile the C program : `$ arm-linux-gnueabi-gcc filename.c -o filename-arm -static`
- b. `$ file filename`
- c. Run the gem5

```
$ ./build/ARM/gem5.opt configs/example/se.py --num-cpus=2 -c 'path to the testcode; path to the
testcode '
```

-----*****-----

36.9.1.1.2 Steps to boot Linux on gem5

- 1) \$ cd /home/elc1/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03
- 2) \$./build/ARM/gem5.opt configs/example/fs.py --disk-image=/home/elc1/full_system_image/disks/aarch32-ubuntu-natty-headless.img

gem5 Simulator System

Copyright (c) 2001-2006
The Regents of The University of Michigan
All Rights Reserved

```
gem5 compiled Aug 16 2006 18:51:57
gem5 started Wed Aug 16 21:53:38 2006
gem5 executing on zeep
command line: ./build/ARM/gem5.debug configs/example/fs.py
0: system.tsunami.io.rtc: Real-time clock set to Sun Jan 1 00:00:00 2006
Listening for console connection on port 3456
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
warn: Entering event queue @ 0. Starting simulation...
<...simulation continues...>
```

Note: --disk-image=path to the full system image directory

*By default, the fs.py script boots Linux and starts a shell on the system console. To keep console traffic separate from simulator input and output, this simulated console is associated with a TCP port. To interact with the console, you must connect to the port using a program such as **telnet**, for example:*

Open new terminal and type

- 3) \$ telnet localhost 3456

Here 3456 is a port number on gem5 terminal

```
elc1@ubuntu:~$ telnet 127.0.0.1 3457
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
==== m5 slave terminal: Terminal 0 ====
Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.13.0-rc2 (tony@vamp) (gcc version 4.8.2 (Ubuntu/Linaro 4.8.2-16ubuntu4) ) #1 SMP PREEMPT Mon Oct 13 15:09:23 EDT 2014
Kernel was built at commit id "
CPU: ARMv7 Processor [410fc0f0] revision 0 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: V2P-CA15
bootconsole [earlycon0] enabled
```

Memory policy: Data cache writealloc
kdebugv2m: Following are test values to confirm proper working
kdebugv2m: Ranges 42000000 0
kdebugv2m: Regs 30000000 1000000
kdebugv2m: Virtual-Reg f0000000
kdebugv2m: pci node addr_cells 3
kdebugv2m: pci node size_cells 2
kdebugv2m: motherboard addr_cells 2
On node 0 totalpages: 131072
free_area_init_node: node 0, pgdat 8072dcc0, node_mem_map 8078f000
Normal zone: 1024 pages used for memmap
Normal zone: 0 pages reserved
Normal zone: 131072 pages, LIFO batch:31
sched_clock: 32 bits at 24MHz, resolution 41ns, wraps every 178956969942ns
PERCPU: Embedded 8 pages/cpu @80b97000 s11648 r8192 d12928 u32768
pcpu-alloc: s11648 r8192 d12928 u32768 alloc=8*4096
pcpu-alloc: [0] 0
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 130048
Kernel command line: earlyprintk=pl011,0x1c090000 console=ttyAMA0 lpj=19988480 norandmaps rw loglevel=8 mem=512MB root=/dev/sda1
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 495584K/524288K available (5248K kernel code, 249K rwddata, 1540K rodata, 295K init, 368K bss, 28704K reserved, 0K highmem)
Virtual kernel memory layout:
vector : 0xffff0000 - 0xffff1000 (4 kB)
fixmap : 0xffff0000 - 0xfffe0000 (896 kB)
vmalloc : 0xa0800000 - 0xff000000 (1512 MB)
lowmem : 0x80000000 - 0xa0000000 (512 MB)
pkmap : 0x7fe00000 - 0x80000000 (2 MB)
modules : 0x7f000000 - 0x7fe00000 (14 MB)
.text : 0x80008000 - 0x806a942c (6790 kB)
.init : 0x806aa000 - 0x806f3d80 (296 kB)
.data : 0x806f4000 - 0x80732754 (250 kB)
.bss : 0x80732754 - 0x8078e9d8 (369 kB)
SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
Preemptible hierarchical RCU implementation.
RCU restricting CPUs from NR_CPUS=8 to nr_cpu_ids=1.
NR_IRQS:16 nr_irqs:16 16
Architected cp15 timer(s) running at 25.16MHz (phys).
sched_clock: 56 bits at 25MHz, resolution 39ns, wraps every 2730666655744ns
Switching to timer-based delay loop
Console: colour dummy device 80x30
Calibrating delay loop (skipped) preset value.. 3997.69 BogoMIPS (lpj=19988480)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
CPU0: update cpu_power 1024
CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
Setting up static identity map for 0x804fee68 - 0x804fee9c
Brought up 1 CPUs
SMP: Total of 1 processors activated.
CPU: All CPU(s) started in SVC mode.
VFP support v0.3: implementor 41 architecture 4 part 30 variant a rev 0
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/sysctl@020000
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/aaci@040000
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/mmci@050000
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/uart@0a0000
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/uart@0b0000
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/uart@0c0000

```
of_amba_device_create(): amba_device_add() failed (-19) for /smb/motherboard/iofpga@3,00000000/wdt@0f0000
hw-breakpoint: Debug register access (0xee113e93) caused undefined instruction on CPU 0
hw-breakpoint: Debug register access (0xee013e90) caused undefined instruction on CPU 0
hw-breakpoint: Debug register access (0xee003e17) caused undefined instruction on CPU 0
hw-breakpoint: CPU 0 failed to disable vector catch
Serial: AMBA PL011 UART driver
1c090000.uart: ttyAMA0 at MMIO 0x1c090000 (irq = 37, base_baud = 0) is a PL011 rev3
console [ttyAMA0] enabled
console [ttyAMA0] enabled
bootconsole [earlycon0] disabled
bootconsole [earlycon0] disabled
PCI host bridge to bus 0000:00
pci_bus 0000:00: root bus resource [io 0x0000-0xffffffff]
pci_bus 0000:00: root bus resource [mem 0x00000000-0xffffffff]
pci_bus 0000:00: root bus resource [bus 00-ff]
pci 0000:00:00.0: [8086:1075] type 00 class 0x020000
pci 0000:00:00.0: reg 0x10: [mem 0x00000000-0x0001ffff]
pci 0000:00:00.0: reg 0x30: [mem 0x00000000-0x000007ff pref]
pci 0000:00:01.0: [8086:7111] type 00 class 0x010185
pci 0000:00:01.0: reg 0x10: [io 0x0000-0x0007]
pci 0000:00:01.0: reg 0x14: [io 0x0000-0x0003]
pci 0000:00:01.0: reg 0x18: [io 0x0000-0x0007]
pci 0000:00:01.0: reg 0x1c: [io 0x0000-0x0003]
pci 0000:00:01.0: reg 0x20: [io 0x0000-0x000f]
pci 0000:00:01.0: reg 0x30: [mem 0x00000000-0x000007ff pref]
PCI: bus0: Fast back to back transfers disabled
pci 0000:00:00.0: BAR 0: assigned [mem 0x40000000-0x4001ffff]
pci 0000:00:00.0: BAR 6: assigned [mem 0x40020000-0x400207ff pref]
pci 0000:00:01.0: BAR 6: assigned [mem 0x40020800-0x40020fff pref]
pci 0000:00:01.0: BAR 4: assigned [io 0x2f000000-0x2f00000f]
pci 0000:00:01.0: BAR 0: assigned [io 0x2f000010-0x2f000017]
pci 0000:00:01.0: BAR 2: assigned [io 0x2f000018-0x2f00001f]
pci 0000:00:01.0: BAR 1: assigned [io 0x2f000020-0x2f000023]
pci 0000:00:01.0: BAR 3: assigned [io 0x2f000024-0x2f000027]
pci_bus 0000:00: resource 4 [io 0x0000-0xffffffff]
pci_bus 0000:00: resource 5 [mem 0x00000000-0xffffffff]
PCI map irq: slot 0, pin 1, devslot 0, irq: 68
PCI map irq: slot 1, pin 2, devslot 1, irq: 69
bio: create slab <bio-0> at 0
vgaarb: loaded
SCSI subsystem initialized
libata version 3.00 loaded.
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
PTP clock support registered
Advanced Linux Sound Architecture Driver Initialized.
Switched to clocksource arch_sys_counter
NET: Registered protocol family 2
TCP established hash table entries: 4096 (order: 2, 16384 bytes)
TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP: reno registered
UDP hash table entries: 256 (order: 1, 8192 bytes)
UDP-Lite hash table entries: 256 (order: 1, 8192 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
```

```

RPC: Registered tcp NFSv4.1 backchannel transport module.
PCI: CLS 64 bytes, default 64
hw perfevents: enabled with ARMv7_Cortex_A15 PMU driver, 1 counters available
jffs2: version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
msgmni has been set to 967
io scheduler noop registered (default)
brd: module loaded
loop: module loaded
ata_piix 0000:00:01.0: version 2.13
PCI: enabling device 0000:00:01.0 (0040 -> 0041)
scsi0 : ata_piix
scsi1 : ata_piix
ata1: PATA max UDMA/33 cmd 0x2f000010 ctl 0x2f000020 bmdma 0x2f000000 irq 69
ata2: PATA max UDMA/33 cmd 0x2f000018 ctl 0x2f000024 bmdma 0x2f000008 irq 69
e100: Intel(R) PRO/100 Network Driver, 3.5.24-k2-NAPI
e100: Copyright(c) 1999-2006 Intel Corporation
e1000: Intel(R) PRO/1000 Network Driver - version 7.3.21-k8-NAPI
e1000: Copyright (c) 1999-2006 Intel Corporation.
PCI: enabling device 0000:00:00.0 (0040 -> 0042)
ata1.00: ATA-7: M5 IDE Disk, , max UDMA/66
ata1.00: 6291936 sectors, multi 0: LBA
ata1.00: configured for UDMA/33
scsi 0:0:0:0: Direct-Access ATA M5 IDE Disk n/a PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 6291936 512-byte logical blocks: (3.22 GB/3.00 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Mode Sense: 00 3a 00 00
sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA
sda: sda1
sd 0:0:0:0: Attached scsi generic sg0 type 0
sd 0:0:0:0: [sda] Attached SCSI disk
e1000 0000:00:00.0 eth0: (PCI:33MHz:32-bit) 00:90:00:00:00:01
e1000 0000:00:00.0 eth0: Intel(R) PRO/1000 Network Connection
e1000e: Intel(R) PRO/1000 Network Driver - 2.3.2-k
e1000e: Copyright(c) 1999 - 2013 Intel Corporation.
igb: Intel(R) Gigabit Ethernet Network Driver - version 5.0.5-k
igb: Copyright (c) 2007-2013 Intel Corporation.
igbvf: Intel(R) Gigabit Virtual Function Network Driver - version 2.0.2-k
igbvf: Copyright (c) 2009 - 2012 Intel Corporation.
ixgbe: Intel(R) 10 Gigabit PCI Express Network Driver - version 3.15.1-k
ixgbe: Copyright (c) 1999-2013 Intel Corporation.
ixgbev: Intel(R) 10 Gigabit PCI Express Virtual Function Network Driver - version 2.11.3-k
ixgbev: Copyright (c) 2009 - 2012 Intel Corporation.
ixgb: Intel(R) PRO/10GbE Network Driver - version 1.0.135-k2-NAPI
ixgb: Copyright (c) 1999-2008 Intel Corporation.
smsc911x: Driver version 2008-10-21
smsc911x 1a000000.ethernet (unregistered net_device): couldn't get clock -2
nxp-isp1760 1b000000.usb: NXP ISP1760 USB Host Controller
nxp-isp1760 1b000000.usb: new USB bus registered, assigned bus number 1
nxp-isp1760 1b000000.usb: Scratch test failed.
nxp-isp1760 1b000000.usb: can't setup: -19
nxp-isp1760 1b000000.usb: USB bus 1 deregistered
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
rtc-pl031 1c170000.rtc: rtc core: registered pl031 as rtc0
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
ashmem: initialized
logger: created 256K log 'log_main'
logger: created 256K log 'log_events'
logger: created 256K log 'log_radio'
logger: created 256K log 'log_system'

```

```
oprofile: using timer interrupt.
TCP: cubic registered
NET: Registered protocol family 10
NET: Registered protocol family 17
rtc-pl031 1c170000.rtc: setting system clock to 2009-01-01 00:00:00 UTC (1230768000)
ALSA device list:
  No soundcards found.
input: AT Raw Set 2 keyboard as /devices/smb.14/motherboard.15/iofpga.17/1c060000.kmi/serio0/input/input0
input: touchkitPS/2 eGalax Touchscreen as /devices/smb.14/motherboard.15/iofpga.17/1c070000.kmi/serio1/input/input2
kjournald starting. Commit interval 5 seconds
EXT3-fs (sda1): using internal journal
EXT3-fs (sda1): mounted filesystem with writeback data mode
VFS: Mounted root (ext3 filesystem) on device 8:1.
Freeing unused kernel memory: 292K (806aa000 - 806f3000)
random: init urandom read with 14 bits of entropy available

input: AT Raw Set 2 keyboard as /devices/smb.14/motherboard.15/iofpga.17/1c060000.kmi/serio0/input/input0
input: touchkitPS/2 eGalax Touchscreen as /devices/smb.14/motherboard.15/iofpga.17/1c070000.kmi/serio1/input/input2
kjournald starting. Commit interval 5 seconds
EXT3-fs (sda1): using internal journal
EXT3-fs (sda1): mounted filesystem with writeback data mode
VFS: Mounted root (ext3 filesystem) on device 8:1.
Freeing unused kernel memory: 292K (806aa000 - 806f3000)
random: init urandom read with 14 bits of entropy available

Ubuntu 11.04 gem5sim ttySA0
```

this will take 30 mins

4) gem5sim login: root root

Welcome to Ubuntu 11.04 (GNU/Linux 3.13.0-rc2 armv7l)

* Documentation: <https://help.ubuntu.com/>

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

```
root@gem5sim:~#
root@gem5sim:~#
```

-----*****-----

37 GEM5 with RISCv

Download the zip file from (details of ftp or github)

Two option for the user:

- 1) Directly run the command without the build – Extract the patch GEM5_Patch_Java_14.zip in the Ubuntu and run commands directly from section 50.2*
- 2) If the user experiences trouble while running directly without building the gem5 from scratch then follow the steps from the section 50.1*

37.1 For Ubuntu 18.04 installation steps

1. Assumption: The user has installed Java 14 and VisualSim 2030.
2. `mkdir gem5`
3. `cd gem5`
4. `sudo apt-get install gcc-riscv64-linux-gnu g++-riscv64-linux-gnu`
5. `sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python-dev python`
6. `sudo apt-get install libpython3.8-dev`
7. `sudo apt-get install swig` --- if swig is not installed
8. `scons build/RISCV/gem5.opt -jn -> (n = 1,2,3number of processor)`
9. if there is no error on the terminal then build is complete.

37.2 Run the Gem5 <-> VisualSim (normal run without debugger)

- 1) `cd` to the gem5 location on terminal and type

`$./interactive_sim.py`
(this start the gem5 simulator)

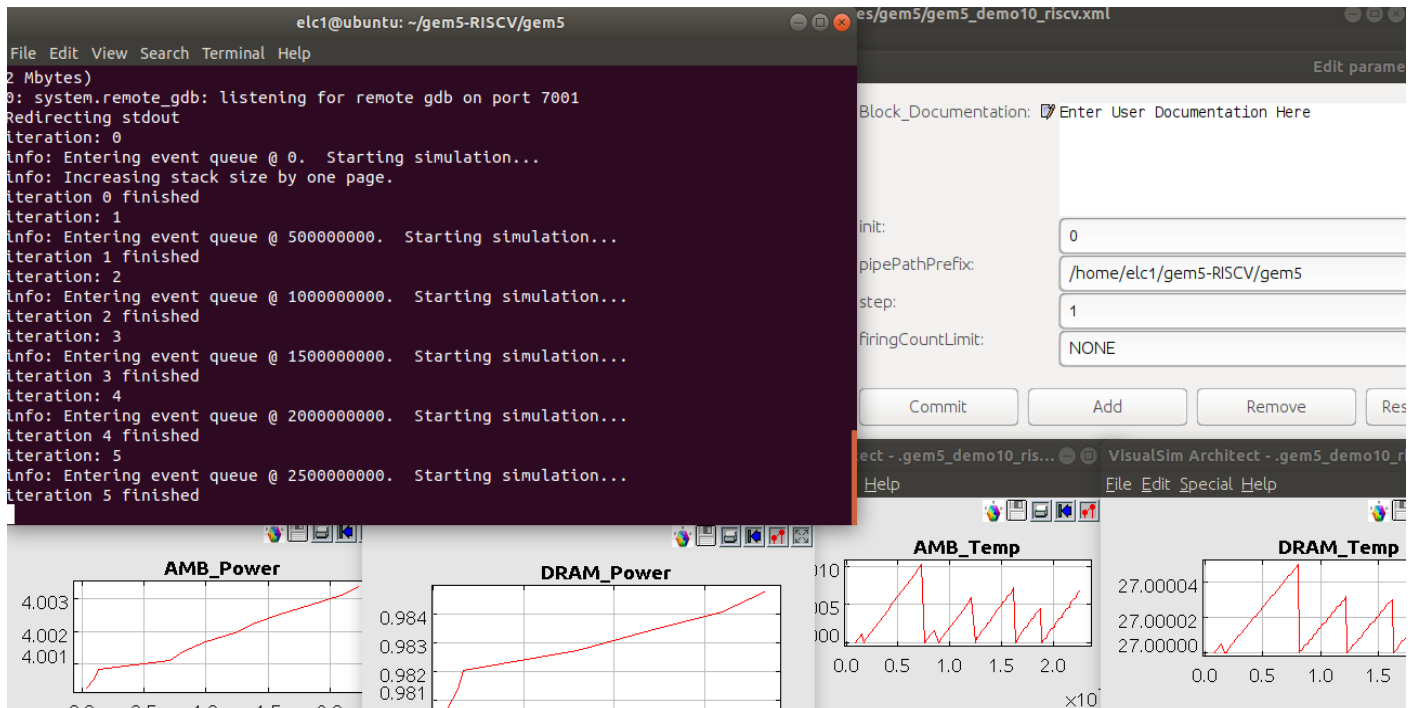
- 2) After **Redirecting Stdout** shows on terminal, run visualsims model by following below steps
- 3) Now open the model in the VisualSim under `demo/interfaces/gem5/gem5_demo.xml`
- 4) Give the correct path in gem5 wrapper block parameter where gem5 is located.
- 5) Provide parameter `gem5iteration = integer number (of your choice)`

- 6) Run the model
- 7) Check the plots display and below screenshots

```
e1c1@ubuntu:~/gem5-RISCV/gem5$ ./interactive_sim.py
starting the simulation!!!
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 20.1.0.0
gem5 compiled Nov  2 2020 17:05:29
gem5 started Nov  6 2020 14:07:36
gem5 executing on ubuntu, pid 4214
command line: ./build/RISCV/gem5.opt --debug-start=1 --debug-flags=DRAM --inter
active ./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --s
ys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_8x8
-c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfil
e ../MiBench/consumer/jpeg/output_large_encode.jpeg ../MiBench/consumer/jpeg/inp
ut_large.ppm'

warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.master is deprecated. `master` is now called `mem_side_ports`
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and p
df.
warn: DRAM device capacity (8192 Mbytes) does not match the address range assign
ed (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
Redirecting stdout
```

37.3 Run with debugger -command line:

- 1) cd to gem5 directory \$ **gdb --args ./build/RISCV/gem5.opt**
- 2) wait for gdb to open up , use run command (step 3) after gdb shows
- 3) (gdb) **run --debug-break=1000 --debug-flags=DRAM --interactive**
./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz -
-sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-
type=DDR3_1600_8x8 -c ./MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -
progressive -opt -outfile
./MiBench/consumer/jpeg/output_large_encode.jpeg
./MiBench/consumer/jpeg/input_large.ppm'
- 4) Wait for gdb() to show on terminal
- 5) After **Redirecting Stdout** shows on terminal, run visualsिम model by following below steps
- 6) Run the visualsिम model
- 7) Now debug the code using gdb commands such as (n, p, f, bt, step, b main, c)
- 8) After debugging type **continue** or **c** on the terminal, for rest of iteration to finished up

- 9) Plots will be displayed.
- 10) For more information refer:

command line

```
elc1@ubuntu:~/gem5-RISCV/gem5$ gdb --args build/RISCV/gem5.opt
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from build/RISCV/gem5.opt...done.
(gdb) run --debug-break=1000 --debug-flags=DRAM --interactive
./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-
clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_8x8
-c ./MiBench/consumer/jpeg/jpeg-6a/cjpeg -o 'dct int -progressive -opt -outfile
../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
```

```
Starting program: /home/elc1/gem5-RISCV/gem5/build/RISCV/gem5.opt --debug-
break=1000 --debug-flags=DRAM --interactive ./configs/example/se_vs.py --cpu-
type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --
l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_8x8 -c
../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o 'dct int -progressive -opt -outfile
../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.
gem5 version 20.1.0.0
gem5 compiled Nov  2 2020 17:05:29
gem5 started Nov  6 2020 17:54:33
gem5 executing on ubuntu, pid 2177
command line: /home/elc1/gem5-RISCV/gem5/build/RISCV/gem5.opt --debug-break=1000 --
debug-flags=DRAM --interactive ./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-
clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-
type=DDR3_1600_8x8 -c ./MiBench/consumer/jpeg/jpeg-6a/cjpeg -o 'dct int -progressive -opt
-outfile ../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
```

https://www.gem5.org/documentation/general_docs/debugging_and_testing/debugging/debugger_based_debugging

command line

```
warn: need to stop all queues
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.master is deprecated. 'master' is now called 'mem_side_ports'
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
O: system.mem_ctrls.dram: Setting up DRAM Interface
O: system.mem_ctrls.dram: Creating DRAM rank 0
O: system.mem_ctrls.dram: Creating DRAM rank 1
O: system.mem_ctrls.dram: Memory capacity 536870912 (536870912) bytes
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
O: system.mem_ctrls.dram: Row buffer size 8192 bytes with 128 bursts per row buffer
O: system.remote_gdb: listening for remote gdb on port 7001
O: system.mem_ctrls.port: Pushing DRAM ranges to port
```

Redirecting stdout

--- Now run the VisualSim model here

command line – give gdb commands

<pre>Redirecting stdout iteration: 0 info: Entering event queue @ 0. Starting simulation... Program received signal SIGTRAP, Trace/breakpoint trap. 0x00007ffff5a4a237 in kill () at ../sysdeps/unix/syscall-template.S:78 78 ../sysdeps/unix/syscall-template.S: No such file or directory. (gdb) n kill () at ../sysdeps/unix/syscall-template.S:79 79 in ../sysdeps/unix/syscall-template.S (gdb) b main Breakpoint 1 at 0x55555564c9d0: main. (4 locations) (gdb) bt #0 kill () at ../sysdeps/unix/syscall-template.S:79 #1 0x00005555556c165a2 in GlobalEvent::BarrierEvent::process (this=0x55555589a3f00) at build/RISCV/sim/global_event.cc:131 #2 0x00005555556c0e77d in EventQueue::serviceOne (this=this@entry=0x5555558981f20) at build/RISCV/sim/eventq.cc:222 #3 0x00005555556c33277 in doSimLoop (eventq=0x5555558981f20) at build/RISCV/sim/simulate.cc:216 #4 0x00005555556c342ca in simulate (num_cycles=<optimized out>) at build/RISCV/sim/simulate.cc:129 #5 0x000055555569db2e1 in pybind11::detail::argument_loader<unsigned long>::call_impl<GlobalSimLoopExitEvent*, GlobalSimLoopExitEvent* (*) (unsigned long), 0ul, pybind11::detail::void_type> (f=<optimized out>, this=<optimized out>) at ext/pybind11/include/pybind11/cast.h:1935 #6 pybind11::detail::argument_loader<unsigned long>::call<GlobalSimLoopExitEvent*, pybind11::detail::void_type, (gdb) n GlobalEvent::BarrierEvent::process (this=0x55555589a3f00) at build/RISCV/sim/global_event.cc:136</pre>	<pre>GlobalSimLoopExitEvent* (*) (unsigned long)> (GlobalSimLoopExitEvent* (*) (unsigned long)) && (f=<optimized out>, this=<optimized out>) at ext/pybind11/include/pybind11/cast.h:1912 #7 void pybind11::cpp_function::initialize<GlobalSimLoopExitEvent* (*) (unsigned long), GlobalSimLoopExitEvent*, unsigned long, 136 globalBarrier(); (gdb) n 137 } (gdb) n EventQueue::serviceOne (this=this@entry=0x5555558981f20) at build/RISCV/sim/eventq.cc:223 223 if (event->isExitEvent()) { (gdb) n 232 event->release(); (gdb) n 234 return NULL; (gdb) n 232 event->release(); (gdb) n 199 std::lock_guard<EventQueue> lock(*this); (gdb) n 235 }</pre>	<pre>(gdb) c Continuing. info: Increasing stack size by one page. iteration 0 finished iteration: 1 info: Entering event queue @ 500000000. Starting simulation... iteration 1 finished iteration: 2 info: Entering event queue @ 1000000000. Starting simulation... iteration 2 finished iteration: 3 info: Entering event queue @ 1500000000. Starting simulation... iteration 3 finished iteration: 4 info: Entering event queue @ 2000000000. Starting simulation... iteration 4 finished iteration: 5 info: Entering event queue @ 2500000000. Starting simulation... iteration 5 finished iteration: 6 info: Entering event queue @ 3000000000. Starting simulation... iteration 6 finished iteration: 7 info: Entering event queue @ 3500000000. Starting simulation... iteration 7 finished iteration: 8 info: Entering event queue @ 4000000000. Starting simulation... iteration 8 finished iteration: 9 info: Entering event queue @ 4500000000. Starting simulation... iteration 9 finished iteration: 10 info: Entering event queue @ 5000000000. Starting simulation... iteration 10 finished</pre>
	<p>(gdb) c Continuing.</p> <p>← To continue the execution</p>	<p>↓ Show plots</p>

37.4 Run with debugger - graphical gdb:

- 1) cd to gem directory\$ **gdb -tui --args ./build/RISCV/gem5.opt**
- 2) wait for gdb to open up , use run command (step 3) after gdb shows
- 3) (gdb) **run --debug-break=1000 --debug-flags=DRAM --interactive**
./configs/example/se_vs.py --cpu-type=TimingSimpleCPU --cpu-clock=1GHz -
-sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-
type=DDR3_1600_8x8 -c ../MiBench/consumer/jpeg/jpeg-6a/cjpeg -o '-dct int -
progressive -opt -outfile
../MiBench/consumer/jpeg/output_large_encode.jpeg
../MiBench/consumer/jpeg/input_large.ppm'
- 4) After **Redirecting Stdout** shows on terminal, run visualsimg model by following below steps
- 5) Run the visualsimg model
- 6) Now debug the code using gdb commands such as (n, p, f, bt, step, b main, c)
- 7) After debugging type **continue** or **c** on the terminal, for rest of iteration to finished up
- 8) Plots will be displayed.
- 9) For more information refer: <http://beej.us/guide/bggdb/#qref>
- 10) **NOTE:** All the normal **gdb** commands will work in GUI mode, and additionally the arrow keys and pgup/pgdown keys will scroll the source window (when it has focus, which it does by default).

Graphical gdb

```
elc1@ubuntu:~/gem5/gem5-ptolemy-master/gem5-stable_2015_09_03$ gdb -tui --args ./build/RISCV/gem5.opt
```



```

./build/RISCV/stn/main.cc
40 {
41     int ret;
42
43     // Initialize m5 special signal handling.
44     initSignals();
45
46     #if PY_MAJOR_VERSION >= 3
47     std::unique_ptr<char_t[], decltype(&PyMem_RawFree)> program(
48         Py_DecodeLocale(argv[0], NULL),
49         &PyMem_RawFree);
50     Py_SetProgramName(program.get());
51     #else
52     Py_SetProgramName(argv[0]);
53     #endif
54
55     // Register native modules with Python's init system before
56     // initializing the interpreter.
57     registerNativeModules();
58
exec No process in:
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
...Type <return> to continue, or q <return> to quit...
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./build/RISCV/gem5.opt...done.
(gdb)

```

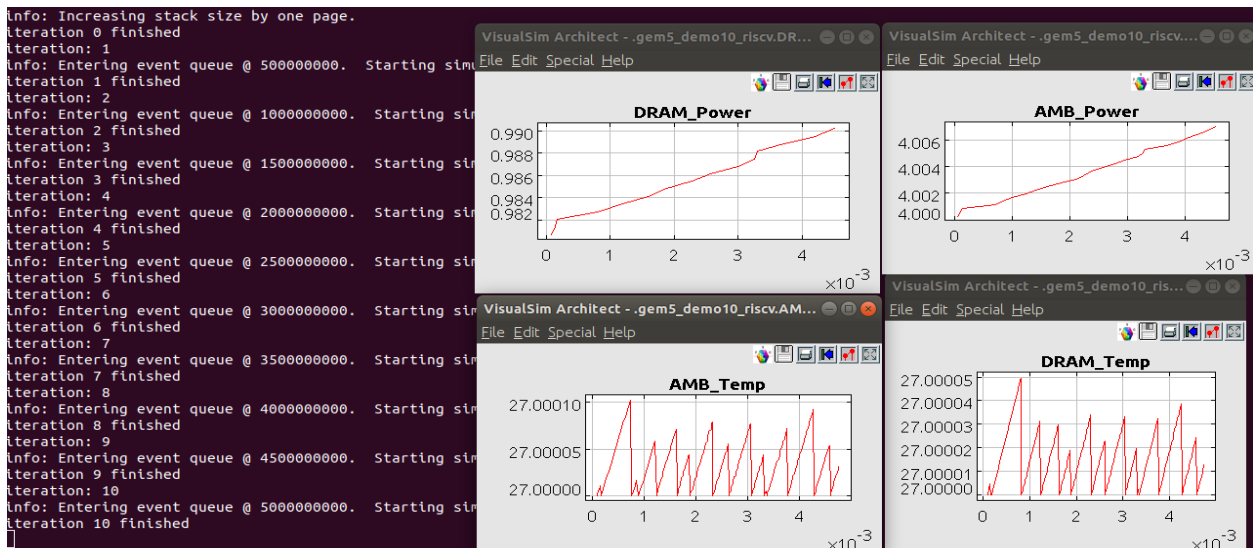
```

gem5 version 20.1.0.0
gem5 compiled Nov  2 2020 17:05:29
gem5 started Nov  6 2020 18:34:33
gem5 executing on ubuntu, pid 2457
command line: /home/elc1/gem5-RISCV/gem5/build/RISCV/gem5.opt --debug-break=1000 --debug-flags=DRAM --interactive ./configs/example/se_vs.py -
-cpu-type=TimingSimpleCPU --cpu-clock=1GHz --sys-clock=1GHz --caches --l1i_size=16kB --l1d_size=16kB --mem-type=DDR3_1600_8x8 -c ../MiBench/co
nsumer/jpeg/jpeg-6a/cjpeg -o '-dct int -progressive -opt -outfile ../MiBench/consumer/jpeg/output_large_encode.jpeg ../MiBench/consumer/jpeg/i
nput_large.ppm'

warn: need to stop all queues
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.slave is deprecated. 'slave' is now called 'cpu_side_ports'
warn: membus.master is deprecated. 'master' is now called 'mem_side_ports'
Global frequency set at 100000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
0: system.mem_ctrls.dram: Setting up DRAM Interface
0: system.mem_ctrls.dram: Creating DRAM rank 0
0: system.mem_ctrls.dram: Creating DRAM rank 1
0: system.mem_ctrls.dram: Memory capacity 536870912 (536870912) bytes
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.mem_ctrls.dram: Row buffer size 8192 bytes with 128 bursts per row buffer
0: system.remote_gdb: listening for remote gdb on port 7002
0: system.mem_ctrls.port: Pushing DRAM ranges to port
Redirecting stdout

```

```
gem5 $build/RISCV/sim/eventq.cc
215
216 // handle action
217 if (!event->squashed()) {
218 // forward current cycle to the time when this event occurs.
219 setCurTick(event->when());
220 if (DTRACE(Event))
221 event->trace("executed");
222 event->process();
223 if (event->isExitEvent()) {
224 assert(!event->flags.isSet(Event::Managed) ||
225 !event->flags.isSet(Event::IsMainQueue)); // would be silly
226 return event;
227 }
228 } else {
229 event->flags.clear(Event::Squashed);
230 }
231
232 event->release();
233
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
0: system.mem_ctrls.dram: SetEventQueue::serviceOne
(gdb) ../sysdeps/unix/syscall-template.S:1: No such file or directory.
(gdb) n
kill () at ../sysdeps/unix/syscall-template.S:79
(gdb) n
GlobalEvent::BarrierEvent::process (this=0x5555589a3f00) at build/RISCV/sim/global_event.cc:136
(gdb) n
EventQueue::serviceOne (this=this@entry=0x555558981f20) at build/RISCV/sim/eventq.cc:223
(gdb) p event
$1 = (Event *) 0x5555589a3f00
(gdb)
```



37.5 Running the Gem5 in System Call Emulation (SE) Mode

The SE mode simulation focuses on the CPU and memory system, and does not emulate the entire system. In this mode, one only needs to specify the binary file to be simulated. This binary file can be statically/dynamically linked. `configs/examples/se.py` is used for configuring and running simulations in this mode. What follows is probably the simplest example of how to use `se.py`. The binary file to simulated is specified with option `-c`.

- 1) cd to the gem5 location on terminal and use the below command to run the hello arm executable.

```
$ elc1@ubuntu:~/gem5-RISCV/gem5$ ./build/RISCV/gem5.opt
./configs/example/se.py -c ./tests/test-progs/hello/bin/riscv/linux/hello
```

- 2) For statistics refer → `/gem5/m5out /stats.txt`

37.6 Running the Gem5 (SE mode) in Multi-core

- 1) cd to the gem5 location on terminal and use the below command to run the hello arm executable.

```
$ elc1@ubuntu:~/gem5-RISCV/gem5$ ./build/RISCV/gem5.opt
configs/example/se.py --num-cpus=2 -c './tests/test-
progs/hello/bin/riscv/linux/hello;./tests/test-progs/hello/bin/riscv/linux/hello'
```

```
elc1@ubuntu:~/gem5-RISCV/gem5$ ./build/RISCV/gem5.opt configs/example/se.py --num-cpus=2 -c
'./tests/test-progs/hello/bin/riscv/linux/hello;./tests/test-progs/hello/bin/riscv/linux/hello'
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.
```

```
gem5 version 20.1.0.0
gem5 compiled Nov  2 2020 17:05:29
gem5 started Nov  6 2020 18:52:21
gem5 executing on ubuntu, pid 2526
command line: ./build/RISCV/gem5.opt configs/example/se.py --num-cpus=2 -c './tests/test-
progs/hello/bin/riscv/linux/hello;./tests/test-progs/hello/bin/riscv/linux/hello'
```

```
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.slave is deprecated. `slave` is now called `cpu_side_ports`
warn: membus.master is deprecated. `master` is now called `mem_side_ports`
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7003
0: system.remote_gdb: listening for remote gdb on port 7004
**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
info: Increasing stack size by one page.
info: Increasing stack size by one page.
Hello world!
Hello world!
```

Exiting @ tick 3339000 because exiting with last active thread context

When number of cores = 4

```
./build/RISCV/gem5.opt configs/example/se.py --num-cpus=4 -c './tests/test-  
progs/hello/bin/riscv/linux/hello;./tests/test-  
progs/hello/bin/riscv/linux/hello;./tests/test-  
progs/hello/bin/riscv/linux/hello;./tests/test-progs/hello/bin/riscv/linux/hello'
```

2) For statistics refer -> `/gem5/m5out/stats.txt`

37.7 Running the Gem5 in Full System (FS) Mode

As currently gem5 supports the RISC-V ISA only in SE mode,

38 Spike to VisualSim

Spike is a functional RISC-V ISA simulator with full system emulation. We provide the support to run the extracted trace from Spike in the VisualSim Processor model.

The user must run the source code in Spike (linux environment) and extract the executed trace using the following steps.

1. Compile the source code using the following command:

```
riscv64-unknown-elf-gcc ./Source_Code/main.c
```

2. Run the compiled binary using spike simulator and generate the log file using the following command. (a.out is the compiler output)

```
./riscv-isa-sim/build/spike -l --log-commits --isa=RV64IMAC ./riscv-pk/build/pk a.out  
2>gcc_out.log
```

3. Parse the log file using the Spike trace parser in \VS_AR\VisualSim\parser\
4. Use the command line with following format to generate the CSV file VisualSim model.

```
Spike_Trace_Parser.py <trace name> <trace path> <_start label starting instr  
address in hex (0x...)> <isa path>
```

<trace name> : This is the name of the trace. Pennant or Stream etc. It can be any name. This name will be used to create the output file.

<trace path> : This is the path to the spike output log trace. Example:

```
D:\Projects\Trace\rvv_test_vector\rvv_test_vector\stream\gcc_out4\gcc_out.log
```

<_start label starting instr address in hex (0x...)> : Open the objdump of the binary and look for _start label. The starting address in that label needs to be entered.

<isa path> : This is the path to the file riscv_isa_boom.txt

Example format:

```
Spike_Trace_Parser.py stream
```

```
D:\Projects\Trxace\rvv_test_vector\rvv_test_vector\stream\gcc_out4\gcc_out.log
```

```
0x10a2c D:\Projects\Samsung_SAIT\riscv_isa_boom.txt
```

5. The parser will provide the CSV file in the same folder where the log file is located.
6. Use this file name in dynamic mapper to assign each instruction to a target RISC-V core.

Add model image with dynamic mapper and processor coeres

