



Catapult® C Synthesis

User's and Reference Manual

University Version

Calypto Design Systems, Inc
2933 Bunker Hill Lane, Suite 202
Santa Clara, CA 95054
Tel: +1-408-850-2300
Fax: +1-408-850-2301

The software described herein is copyright ©2002-2011 Calypto Design Systems, Inc. All rights reserved. The software described herein, which contains confidential information and trade secrets, is property of Calypto Design Systems, Inc.

This manual is copyright ©2005-2011 Calypto Design Systems, Inc. Printed in U.S.A. All rights reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, transmitted, or reduced to any electronic medium or machine-readable form without prior written consent from Calypto Design Systems, Inc.

This manual and its contents are confidential information of Calypto Design Systems, Inc., and should be treated as confidential information by the user under the terms of the nondisclosure agreement and software license agreement, as applicable, between Calypto Design Systems, Inc. and user.

Last Updated: October 2011

Product Version: Release 2011a

Table of Contents

Chapter 1	
Catapult Introduction.....	23
Catapult Features	23
Catapult and the Design Flow	24
Catapult User Interface	24
About this Manual	25
Chapter 2	
Designing with Catapult C Synthesis	27
The Catapult Design Process	27
Step 1: Writing and Testing the Source Code.....	27
Step 2: Analyzing the Algorithm	28
Step 3: Creating the Hardware Design	31
Step 4: Performing Timed Simulations	33
Step 5: Synthesizing the RTL Design.....	33
Step 6: Analyzing Power Consumption	34
Step 7: Incorporating Post-Synthesis Design Changes.....	34
Writing Input Source Code	36
Hardware Technology Constraints	37
Libraries	37
Design Frequency	38
Algorithmic Synthesis	38
Set Design Elements using Catapult.....	38
Automatic Synthesis Features in Catapult	38
Interface Synthesis	39
Adding Timing in a Design	40
Frequently Asked Questions	40
What if my design doesn't meet timing after place and route?.....	40
Why doesn't Catapult support unions?	41
Chapter 3	
Getting Started	43
Understanding Catapult Projects and Solutions.....	43
Branching Solutions	44
Saving and Restoring a Project.....	44
Migrating Design Solutions to Newer Catapult Versions	44
Creating a Working Directory	45
The Catapult Work Flow	46
Format of Catapult Generated Object Names.....	49
Invoking the Graphical User Interface.....	59
The Catapult Session Window.....	60
The Start Page.....	64

The Tool Bar	65
The Task Bar	67
The Details Window	68
The Constraint Editor Window	69
The Project Files Window	70
The Command Input and Transcript Window	72
The Flow Manager Window	77
The Table Window	79
The Bar Chart and XY Plot Windows	80
The Toolkits Window	82
Changing the Window Layout	89
Setting the Working Directory	92
Adding and Removing Input Files	93
Preparing the Source Files	95
Setting Up the Design	96
Specifying the Technology and Clock Frequency	97
Setting Up the Interface Control Signals	98
Setting the Top-Level Function	100
Design Analysis in Catapult	100
Performing Design Analysis	100
Design Analysis Checklist	104
Specifying Architectural Constraints	104
Architectural Constraints on the Design	106
Architectural Constraints on Processes	112
Architectural Constraints on Resources	114
Architectural Constraints on Variables Assigned to Resources	118
Architectural Constraints on Loops	120
Multiple Selection Capability	122
Block Diagram Constraint Editor	124
Scheduling the Design	127
About the Schedule Window	127
Interpreting Data in the Gantt Chart	131
Exploring the Design	134
Specifying Cycle Constraints	139
Interpreting Results When Scheduling Fails	139
Generating the Output Files	141
Catapult Report Files	141
Catapult HDL Output Files	142
Memory Map Exported for IP-XACT	142
Catapult Schematics	143
Catapult Generated Synthesis Scripts	144
Customizing and Packaging Netlist Files	145
Configurable Basenames for Generated Files	146
Using the Schematic Viewer Window	148
Schematic Hierarchy	148
View States in the Schematic Viewer	149
Querying for Point-to-Point Path Delay	153
Zoom and Page Navigation Controls	154
Integrated Code Analysis Flow	155

Table of Contents

Enabling the CXXAnalysis Flow and Setting Options	156
Running the CXXAnalysis Flow	156
Understanding Messages in the Transcript	159
Runtime Information	159
Severity Classification of Messages	161
Message Identifiers	161
ASSERT-1 Messages	163
File Versioning System	163
The Process for Handling Input Files	163
DesignPad Text Editor	165
Browsing Code	165
Bookmarks	167
Outline Mode	167
Understanding Catapult Project File Types	168
Input File Types	168
Project Control Files Types	169
Output Files	169
SIF Files	171
Incremental Update Files	171
Chapter 4	
Setting Up Catapult Default Options	173
The Options Window	173
General Options	175
Message Options	176
Project Initialization Options	178
Version Control Options	179
Component Libraries Options	180
Interface Options	181
Architectural Options	184
Hardware Options	186
Input Options	188
View Compiler Settings	189
Output Options	189
Constraints Editor Options	195
Text Editor Options	195
Schedule Viewer Options	196
Schematic Viewer Options	197
Toolkits Options	198
Flows Options	198
ModelSim Flow Options	199
Precision RTL Flow Options	201
CXXAnalysis Flow Options	202
DCPower Flow Options	203
Design Compiler Flow Options	204
MATLAB Flow Options	207
NCSim Flow Options	207
Novas Flow Options	208

OSCI Flow Options	209
PowerPlay Flow Options	210
RTLCompiler Flow Options.....	210
SCVerify Flow Options	212
SLEC Formal Verification Flow Options.....	215
SOPC_Builder flow Options	217
SpyGlassPower Flow Options	218
TalusDesign Flow Options	219
Valgrind Flow Options	221
VCS Flow Options	221
Vista Flow Options	222
XPower Flow Options	222
Saving and Restoring Session Options	223
Chapter 5	
Design Optimization	225
Loop Constraints	225
Loops and Timing	226
Timing and Loop Pipelining.....	227
Loop Iteration Interval Delay	228
Consecutive Loops	228
Loops and I/O Access Placement	229
Constraining Loop Iterations	229
Loop Unrolling.....	231
Loop Merging	234
Loop Pipelining.....	239
Distributed Pipeline Synthesis	245
I/O and Cycle Flow Analysis	248
Memories and Arrays.....	248
Array Naming	249
Resources and Memory Arrays.....	250
Mapping Arrays in Physical Memory.....	251
Packing Mode Algorithms	253
Word Width Constraint.....	255
Automatic Memory Optimizations	258
Splitting Memory Resources	260
Merged Read/Write Memory Operations	264
Bytewise Write Enable Memories	265
DirectInput Resource Type	270
Advanced I/O Control for Memories.....	271
Memory Streaming	271
General Design Constraints	273
Clocks	273
Scheduler Optimization	276
Register Fanout Optimization.....	279
Process-Level Handshake.....	280

Table of Contents

Chapter 6		
Interface Synthesis		283
Interface Synthesis Protocols.....		283
Rules for I/O Signals		284
Port Naming Conventions.....		285
Input Parameters		286
Output Parameters.....		287
Basic I/O Components		288
Handshake.....		292
Timing Constraints on Input/Output Ports.....		294
Setting I/O Delay Constraints.....		295
I/O Delay Constraints Transferred to RTL Synthesis Tools.....		297
Chapter 7		
Directives		299
Introduction		299
Directives.....		299
How to Set Directives.....		299
Valid Object Types.....		300
Alphabetical List of Directives		300
Directives Set from the Graphical User Interface		305
Architectural Constraints Editor.....		306
Setup Design Dialog Box		307
Pre-Defined Directives		307
ARRAY_SIZE		307
AREA_GOAL		308
ASSIGN_OVERHEAD		309
BASE_ADDR.....		310
BASE_BIT		310
BLOCK_SIZE		310
CHARACTERIZE_OPERATOR.....		311
CLOCK_EDGE		311
CLOCK_NAME.....		312
CLOCK_OVERHEAD		312
CLOCK_PERIOD		312
CLOCK_UNCERTAINTY		313
CLOCKS.....		313
COMPGRADE		314
CREATE_COMBINATIONAL.....		314
CSTEPS_FROM.....		314
DECOPLING_STAGES		316
DEFAULT_VALUE_OPT.....		316
DESIGN_GOAL.....		317
DISTRIBUTED_PIPELINING		317
DONE_FLAG		317
EFFORT_LEVEL		318
ENABLE_ACTIVE		318
ENABLE_NAME.....		318

EXTERNAL_MEMORY	319
FSM_ENCODING	319
GATE EFFORT	319
GATE_EXPAND_MIN_WIDTH	320
GATE_MIN_WIDTH	320
GATE_REGISTERS	321
GEN_EXTERNAL_ENABLE	321
IDLE_SIGNAL	321
IGNORE_DEPENDENCY_FROM	322
IGNORE_PROCESS	323
INCR_IO_CYCLES	323
INCR_OP_CYCLES	323
INCR_REG_SHARING	323
INCR_RES_SHARING	324
INPUT_DELAY	324
INPUT_REGISTERS	324
INTERLEAVE	325
ITERATIONS	325
MAP_TO_MODULE	325
MAXLEN	326
MAX_LATENCY	326
MEM_MAP_THRESHOLD	327
MERGEABLE	327
MIN_CSTEPS_FROM	328
NO_X_ASSIGNMENTS	328
OLD_SCHED	329
OPT_CONST_MULTS	329
OUTPUT_DELAY	330
OUTPUT_REGISTERS	330
PACKING_MODE	330
PIPELINE_INIT_INTERVAL	331
PIPELINE_RAMP_UP	331
PRESERVE_STRUCTS	331
READY_FLAG	332
REDUNDANT_MUX_OPT	332
REG_MAX_FANOUT	333
REGISTER_IDLE_SIGNAL	333
REGISTER_INPUT	333
REGISTER_NAME	334
REGISTER_THRESHOLD	334
RESET_ASYNC_ACTIVE	335
RESET_ASYNC_NAME	335
RESET_CLEAR_ALL_REGS	336
RESET_KIND	336
RESET_SYNC_ACTIVE	336
RESET_SYNC_NAME	337
RESOURCE_NAME	337
SAFE_FSM	338
SPECULATE	338

Table of Contents

STAGE_IO_CNS	338
STAGE_REPLICATION	339
START_FLAG	339
STREAM	339
TECHLIBS	340
TRANSACTION_DONE_SIGNAL	340
UNROLL	340
WORD_WIDTH	340
Pragmas	341
hls_builtin	342
hls_design	342
hls_direct_input	342
hls_fixed	344
hls_map_to_operator	344
hls_noglobals	345
hls_pipeline_init_interval	345
hls_preserve_struct	345
hls_remove_out_reg	345
hls_resource	347
hls_stream	347
hls_unroll	347
Chapter 8 Commands	349
General Command Syntax	349
Documentation Conventions for Catapult Commands	350
Command Interface to the SIF Database	352
Common Command Switches	356
Using Tcl Commands in Scripts	358
Interactive Command Line Shell	359
GUI	359
Command Line Invocation Argument	359
Tcl Startup Script	360
Command Reference	360
Command Summary	360
Commands Mapped to Flow Tasks	364
application	366
application get	367
application exit	369
application report	370
catapult	371
cycle	374
cycle add	375
cycle find_op	379
cycle get	380
cycle remove	383
cycle set	385
directive	388

directive get	389
directive remove	392
directive set	394
dofile	396
flow	397
flow get	398
flow package forget	401
flow package names	402
flow package option add	403
flow package option get	405
flow package option remove	406
flow package option set	407
flow package present	408
flow package provide	410
flow package require	412
flow package script	414
flow package vcompare	415
flow package versions	417
flow package vsatisfies	418
flow provide	420
flow run	422
go	423
help command	424
help message	427
ignore_memory_precedences	429
logfile	431
logfile close	432
logfile isopen	433
logfile message	434
logfile move	439
logfile name	440
logfile open	441
logfile save_commands	442
options	443
options defaults	444
options exists	446
options get	447
options load	450
options save	452
options set	455
project	457
project close	458
project get	459
project load	461
project new	462
project report	463
project save	465
project set	466
quit	468

Table of Contents

resource	469
resource add	470
resource remove	473
set_working_dir	475
solution	476
solution get	477
solution file add	482
solution file remove	487
solution file restore	488
solution file set	489
solution new	492
solution netlist	494
solution remove	498
solution rename	499
solution report	500
solution restore	504
solution select	505
solution timing	506
view schedule	509
view schematic	510
view file	512
view source	513
Chapter 9	
Design Verification	515
SCVerify Flow	515
C/C++ Designs and SCVerify	516
SystemC Designs and SCVerify	517
Setting Up SCVerify	518
Verifying Input Source Code	519
Verifying RTL Model	520
Linking External Libraries into SCVerify	522
External Libraries and Multiple Platforms/Compilers	523
Accelerated FPGA Devices and Verification	523
Makefiles Generated by SCVerify	526
Infrastructure Files Generated by SCVerify (C/C++)	528
Design Synchronization (C/C++)	529
Preparing a Testbench for SCVerify (C/C++)	530
Design and Testbench Example (C/C++)	531
SCVerify TestBench Transaction Controls (C/C++)	534
Generating a Random Vector Testbench (C/C++)	541
Random Vector Testbench Structure (C/C++)	542
Non-Blocking Reads and Verification (C/C++)	543
Known Issues with SCVerify Replay Verification (C/C++)	544
Trace/Replay Verification Example (C/C++)	545
SCVerify Frequently Asked Questions	551
SCVerify Flow Troubleshooting	552
Formal Verification Flow	565

A Sample Formal Verification Flow Session	568
Supported Interface Synthesis Components	571
Flop Mappings for the SLEC Flow	572
Generating Proofs for Hierarchical Blocks	572
MATLAB/Simulink Flow (C/C++)	574
Enabling the MATLAB/Simulink Flow (C/C++)	580
Chapter 10	
Power Analysis and Optimization	583
Analyzing a Design for Power Usage	583
Design Flow	584
Preparation Before Running Catapult	584
Running a Power Flow in Catapult	586
Power Analysis Flow Limitations	588
Low Power Optimization	589
Clock Gate Insertion	589
FSM State Encoding	591
Idle Signal Insertion	592
Estimating Clock Gate Power Savings	594
Chapter 11	
CCORE Design Flow	597
CCORE Restrictions	599
CCORE Variables	600
Top-Down CCORE Methodology	600
Bottom-Up CCORE Methodology	602
CCOREs in the Gantt Chart	603
Creating CCORE Component Libraries	603
Creating the Library	604
Running Downstream Synthesis	604
Netlisting the Component and Setting Library Component Properties	605
Examples	605
Chapter 12	
Flow Customization	609
Flow Package File Outline	609
Loading Flows	610
Simple Flow Example	611
Safe Interpreters	613
Package Versioning	614
Version Format	614
Accessing Versions	615
FLOWS_API Database	615
Chapter 13	
Accelerated Component Libraries	625
Using the Altera Accelerated Libraries	626
Altera Custom Libraries and Header Files	626

Table of Contents

Use Model for Altera	626
Using or Compiling the Altera Simulation Libraries	629
Verifying the Design.	629
Supported Operators and Functions	630
Using Extra Input and Output Registers	635
Using the Xilinx Accelerated Libraries	637
Adding the Xilinx Custom Libraries and Header Files	637
Use Model for Xilinx	637
Using or Compiling the Xilinx Simulation Libraries	640
Supported Operators/Functions	640
Using Extra Input and Output Registers	642
Appendix A Catapult Library Components	645
Base Components	645
I/O Components	647
Memory Components	649
Channel Components	652
Altera Accelerated Components	654
Xilinx Accelerated Components	656
Appendix B GUI Reference	657
SCVerify Makefile Popup Menu	658
Flow Package: SCVerify Dialog Box	660

Glossary

Index

Third-Party Information

List of Examples

Example 3-1. XML Code to Add Custom Toolkits.....	86
Example 5-1. Code Timing Example	227
Example 5-2. Consecutive Loop	229
Example 5-3. I/O Access Pointer.....	229
Example 5-4. Unknown Loop Iteration Count Shown in Cycle Report	231
Example 5-5. Loop Unrolling Code Example.....	233
Example 9-1. <i>cubed</i> Function	531
Example 9-2. Testbench for <i>cubed</i> Function.....	531
Example 9-3. Output of the <i>cubed</i> Testbench	532
Example 9-4. Testbench for <i>cubed</i> Function extended for SCVerify	532
Example 9-5. <i>cubed</i> Testbench Waveform with 5 Iterations in ModelSim.....	533
Example 9-6. Arbiter Example Using Non-Blocking size() Method	545
Example 9-7. Arbiter Example Header File	547
Example 9-8. Testbench for Arbiter Example Design.....	548
Example 9-9. Setup Script for Arbiter Example Design	548
Example 11-1. CCORE Example Design	597
Example 13-1. Basic Design to be Modified to Use Accelerated Libraries	627

List of Tables

Table 3-1. Definitions of Net Name Suffixes in Wait Controller	57
Table 3-2. Editable Constraints Corresponding to Task Bar State	70
Table 3-3. Drop Points for Rearranging Windows Inside the Session	91
Table 3-4. Data Dependency Path Color Key	130
Table 3-5. Interpreting Slack in the Gantt Chart	132
Table 3-6. Backward/Forward Tracing Options	134
Table 3-7. Sorting and Filtering Options in the Schedule Window	136
Table 3-8. Setting Cycle Constraints	139
Table 3-9. Default Basenames	146
Table 3-10. Effect of Changing Factory Default Basenames	147
Table 3-11. System Message Identifier Tags	162
Table 3-12. Input File Extensions	169
Table 3-13. Catapult Control Files	169
Table 3-14. Output Files	170
Table 3-15. Incremental Solution Tcl Scripts	171
Table 4-1. Buttons in the Catapult Options Window	174
Table 4-2. Replacement Strings for Illegal Characters in HDL Identifiers	193
Table 4-3. Default Flow Settings	199
Table 5-1. Compatible Resource and Component Types	258
Table 6-1. Input Port Naming Conventions	285
Table 6-2. Output Port Naming Conventions	285
Table 6-3. Inout Port Naming Conventions	286
Table 7-1. Valid Object Types for Directives	300
Table 7-2. Alphabetical Summary of Directive	300
Table 7-3. Keyword Values for MAP_TO_MODULE Directive	325
Table 7-4. Alphabetical Summary of Pragmas	341
Table 8-1. Basic Tcl Syntax	349
Table 8-2. Typical Command Form Examples	350
Table 8-3. Other Command Form Examples	350
Table 8-4. Documentation Conventions for Command Syntax	351
Table 8-5. Commands That Take Database Path Arguments	353
Table 8-6. Alphabetical Command Summary	360
Table 8-7. Commands Mapped to Flow Tasks	364
Table 8-8. <i>flow package vcompare</i> return states	415
Table 8-9. Log Message Severity Formatting Options	434
Table 8-10. Arguments for the -xref Option	435
Table 8-11. File Type Names Known to Catapult	482
Table 8-12. Netlist Properties	497
Table 8-13. Valid Values for bool Arguments	500
Table 9-1. Makefile Variables	523

Table 9-2. SCVerify Testbench Transaction Controls	535
Table 9-3. C++ and STL Headers	561
Table 9-4. SLEC Support for Catapult Interface Synthesis Components	571
Table 9-5. Limited Support for Data Types on the Interface	576
Table 10-1. Supported Power Analysis Tools	583
Table 11-1. <i>solution netlist</i> Netlisting Properties	605
Table 12-1. Data Elements in the Top Level Node and HIER_BLOCK Nodes	618
Table 12-2. Data Elements in the RESOURCE_BLOCKS Node	619
Table 12-3. Data Elements in the PORTS Node	620
Table 12-4. Data Elements in the BINDINGS Node	622
Table 12-5. Data Elements in the PROPERTIES Node	622
Table 12-6. Data Elements in the VARIABLES Node	622
Table 12-7. Data Elements in the TRIOSY Node	623
Table 12-8. Data Elements in the TECH_INFOS Node	623
Table 13-1. Altera Accelerated Library Performance Benefits	626
Table 13-2. Template Parameters for Altera Accelerated Libraries	627
Table 13-3. Xilinx Accelerated Library Performance Benefits	637
Table 13-4. Template Parameters for Xilinx Accelerated Libraries	638
Table A-1. Base Library Component Parameters	645
Table A-2. I/O Library Component Parameters	647
Table A-3. Memory Library Component Parameters	649
Table A-4. Channel Library Component Parameters	652
Table A-5. Altera Accelerated Component Parameters	654
Table A-6. Xilinx Accelerated Component Parameters	656
Table B-1. Makefile Popup Menu Options	658
Table B-2. Flow Package: SCVerify Options	661

List of Figures

Figure 2-1. Gantt Chart with Data Objects Highlighted	29
Figure 2-2. Output Reports for Design Analysis	30
Figure 2-3. Interface Controls for Setup Design	32
Figure 2-4. Incremental Update GUI	35
Figure 2-5. C Code Design for Synthesis	37
Figure 2-6. Interface Control in the Design	39
Figure 2-7. Catapult Adds Virtual Wait to the Design	40
Figure 3-1. Migrating Solutions	45
Figure 3-2. File Structure of the Working Directory	46
Figure 3-3. Synthesis Flow	48
Figure 3-4. Data Transition States in the Synthesis Flow	49
Figure 3-5. Architectural Element Names in Source Code	55
Figure 3-6. Architectural Element Names in GUI	56
Figure 3-7. Setting Up the Catapult C Synthesis Shortcut	60
Figure 3-8. Initial Catapult Window	62
Figure 3-9. Start Page Window	65
Figure 3-10. Tool Bar Button for Creating a New Project or Document	66
Figure 3-11. Cross-Probe Buttons on the Tool Bar	66
Figure 3-12. Stop Button on the Tool Bar	66
Figure 3-13. Solution Selection and View Controls on the Tool Bar	67
Figure 3-14. Window Activation Buttons on the Tool Bar	67
Figure 3-15. Context Help Button	67
Figure 3-16. Progressive Disclosure of Tasks on the Task Bar	68
Figure 3-17. Design Object Information Displayed in the Details Window	69
Figure 3-18. Project Files Window	71
Figure 3-19. Editing the Solution Properties	72
Figure 3-20. Command Input and Transcript Window	73
Figure 3-21. Transcript Window Messages	74
Figure 3-22. Viewing the Long Description of a Message	75
Figure 3-23. Filtering Comments Out of the Transcript (before and after)	76
Figure 3-24. Flow Manager Window	78
Figure 3-25. Quality of Results Summary Table	79
Figure 3-26. Timing Report Table	80
Figure 3-27. Opening a Bar Chart or XY Plot Window	80
Figure 3-28. Bar Chart of Area Score	81
Figure 3-29. XY Plot of Area Versus Time/Throughput	82
Figure 3-30. The Toolkits Window	83
Figure 3-31. Toolkit Interface	84
Figure 3-32. Custom Toolkits Added to Toolkits Window	85
Figure 3-33. Window Display Mode Menu	90

Figure 3-34. Window Maximize and Restore Buttons	90
Figure 3-35. Moving a Window to a Drop Point.....	92
Figure 3-36. Manually Setting the Working Directory	93
Figure 3-37. Adding an Input File	94
Figure 3-38. Preparing the Source Code File	96
Figure 3-39. Design Technology ptions	97
Figure 3-40. Hardware Interface Controls.....	98
Figure 3-41. Setting Top-Level Function in Setup Design Window.....	100
Figure 3-42. Catapult Output Files	101
Figure 3-43. Cross-Probing from Source to Output	103
Figure 3-44. Architectural Constraints Graphical View of Design	105
Figure 3-45. Global Architectural Constraints	107
Figure 3-46.	112
Figure 3-47. Mapping a Resource to Specific Hardware.....	115
Figure 3-48. Viewing Map of Physical Memory.....	118
Figure 3-49. Re-assigning a Variable to Another Resource	120
Figure 3-50. Editing Loop Constraints	122
Figure 3-51. Multiple Selection in Architectural Constraints Window.....	123
Figure 3-52. Types of Input Fields on Architectural Constraints Dialog Box	124
Figure 3-53. Block Diagram Views and Tree View in Constraint Editor	125
Figure 3-54. Accessing Constraint Options in the Block Diagram	126
Figure 3-55. Block Diagram Tool Bar Buttons	127
Figure 3-56. Schedule Window	128
Figure 3-57. Schedule Window Toolbar	129
Figure 3-58. Zoom Command Strokes	129
Figure 3-59. Gantt Chart Data Dependencies	131
Figure 3-60. Gantt Chart C-Steps	131
Figure 3-61. Gantt Chart Default Display	133
Figure 3-62. Gantt Chart Compact Mode	133
Figure 3-63. Schedule Window Data Sorted by C-Steps.....	137
Figure 3-64. Schedule Window Data Sorted by Component Utilization	138
Figure 3-65. Schedule Window Data Sorted by Type, All Operations	138
Figure 3-66. Gantt Chart with Clock Overhead Column Highlighted.....	140
Figure 3-67. Gantt Chart with Failed Paths Highlighted.....	140
Figure 3-68. HDL Output File Hierarchy in GUI	142
Figure 3-69. Schematic Output Files	144
Figure 3-70. Launching Precision RTL Synthesis Scripts.....	144
Figure 3-71. Launching Design Compiler Synthesis Scripts.....	145
Figure 3-72. Schematic Viewer Window	149
Figure 3-73. Schematic View State “RTL Schematic”	150
Figure 3-74. Schematic View State “Data Path”.....	150
Figure 3-75. Schematic View State “Critical Path”.....	151
Figure 3-76. Secondary Net Branches Highlighted	152
Figure 3-77. Schematic View State “Critical Map”	153
Figure 3-78. Obtaining Point-to-Point Delay Data from the Schematic	154

List of Figures

Figure 3-79. Stroke Command Patterns	155
Figure 3-80. CXXAnalysis Flow Output Files	157
Figure 3-81. Static Report	158
Figure 3-82. Runtime Report	158
Figure 3-83. Coverage Report	159
Figure 3-84. DesignPad Window	165
Figure 3-85. Hierarchy and Parser Levels	166
Figure 3-86. Setting DesignPad Bookmarks	167
Figure 3-87. DesignPad Outline Mode	168
Figure 4-1. General Options Dialog Box	173
Figure 4-2. Messages Options Dialog Box	177
Figure 4-3. Excerpt from catapult.ini File	224
Figure 5-1. Source Code with Loop	226
Figure 5-2. Default Loop Iteration	226
Figure 5-3. Timing using Loop Pipelining	227
Figure 5-4. Iteration Loop Delay	228
Figure 5-5. Loop Iteration Count in Architectural Constraints Editor	230
Figure 5-6. Loop Icon for Unknown Iteration Count	231
Figure 5-7. Loop Unrolling	232
Figure 5-8. Viewing Data Dependencies in the Gantt Chart	233
Figure 5-9. Loop Unrolling Example	234
Figure 5-10. Loop Unrolling in Architectural Constraints Editor	234
Figure 5-11. Schedule View of Example 1 Code	237
Figure 5-12. Schedule View of Example 2 Code	238
Figure 5-13. Loop Pipelining Overview	240
Figure 5-14. Schedule Optimized for Conditional Data Dependency	241
Figure 5-15. Three-Level Nested Loops with No Pipelining	242
Figure 5-16. Three-Level Nested Loops with Middle Loops Pipelined	243
Figure 5-17. Three-Level Nested Loops with Outer Loop Pipelined	243
Figure 5-18. Loop Pipelining in Architectural Constraints Editor	244
Figure 5-19. Nested Loop Pipelining Symbols	245
Figure 5-20. Non-Distributed Pipelining Illustration	246
Figure 5-21. Distributed Pipelining Illustration	246
Figure 5-22. Decoupling Pipe Between Pipeline Stages	248
Figure 5-23. Variable and Resources	250
Figure 5-24. Setting a Variable Inside Memory	251
Figure 5-25. Default Addresses of Arrays Sharing a Resource	252
Figure 5-26. Bit and Word Location Mapping Example	253
Figure 5-27. Packing Modes for Arrays Mapped to Memories	255
Figure 5-28. Array Word Width	256
Figure 5-29. Using Word Width Constraint To Split I/O Ports	257
Figure 5-30. Catapult Thresholds	259
Figure 5-31. Byte Write Enabled Memory	266
Figure 5-32. Byte Enable Option in Architectural Constraints Window	269
Figure 5-33. DirectInput Resource Type	271

Figure 5-34. Streaming Example	272
Figure 5-35. Percent Sharing Allocation	274
Figure 5-36. Clock, Reset and Enable Information	275
Figure 5-37. Affect of DESIGN_GOAL on Scheduling	277
Figure 5-38. AREA_GOAL Limits Scheduler Optimizations.	278
Figure 5-39. Affect of EFFORT_LEVEL on AREA_GOAL	278
Figure 5-40. Affect of MAX_LATENCY on Scheduling	279
Figure 5-41. Process Level Handshake	281
Figure 6-1. Process-Level Handshake Protocol	283
Figure 6-2. I/O Signals and Timing	284
Figure 6-3. Input Component - wire	289
Figure 6-4. Input Component - register	289
Figure 6-5. Output Component O1	291
Figure 6-6. Output Component O2	291
Figure 6-7. Output Component O3	291
Figure 6-8. InOut Component I/O1	292
Figure 6-9. InOut Component I/O2	292
Figure 6-10. Input + Handshake	293
Figure 6-11. Output + Handshake	294
Figure 6-12. Input and Output Delay Illustration	294
Figure 6-13. Input Delay Wave Form Diagram	295
Figure 6-14. Output Delay Wave Form Diagram	295
Figure 6-15. Timing Diagram Showing Input and Output Delay	296
Figure 8-1. Hierarchy of Objects in the SIF Database	353
Figure 8-2. Catapult Default Schedule without Constraints	377
Figure 8-3. Catapult Default Schedule without Constraints	381
Figure 8-4. Schedule with Cycle Constraints	381
Figure 8-5. Example Cross-Probe Link from Logfile Command	438
Figure 8-6. Nodes of Interest in the Solution Database Hierarchy	479
Figure 8-7. Adding Input Files Example	484
Figure 8-8. Adding Output Files and Folder Hierarchy to the Solution	486
Figure 8-9. Changing File Options with the Solution File Set Command.	491
Figure 9-1. SCVerify: C/C++ and RTL Verification.	516
Figure 9-2. SCVerify: SystemC and RTL Verification	517
Figure 9-3. SCVerify: Makefile for C/C++ Simulation.	520
Figure 9-4. SCVerify: Makefile for VHDL, C/C++, ModelSim	521
Figure 9-5. SCVerify Makefiles	526
Figure 9-6. No Input Stalling	540
Figure 9-7. Stalls Read of the First Element by 2 Cycles	540
Figure 9-8. Stalls All Elements by 2 Cycles	541
Figure 9-9. Stalls Read of Every Other Element by 2 Cycles	541
Figure 9-10. Design Features	545
Figure 9-11. Trace/Replay Two Pass Verification	551
Figure 9-12. Diagram of Integrated Formal Verification Flow	567
Figure 9-13. Formal Verification Flow Output Files	569

List of Figures

Figure 9-14. SLEC Verification Results Output Files	570
Figure 9-15. Hierarchical Block Verification	573
Figure 9-16. Integration of Simulink and Catapult	575
Figure 9-17. MATLAB Flow Interface	577
Figure 9-18. Catapult Blockset Library in Simulink	578
Figure 9-19. Figure 9. Help text for Catapult generated blocks	579
Figure 10-1. Power Flow Folder and Makefiles	587
Figure 10-2. Explicit Clock Enable Signals.	589
Figure 10-3. Clock Gate Implementation in RTL Synthesis	590
Figure 10-4. Global Idle Signal in Hierarchical Design	592
Figure 10-5. Global and Block-Specific Idle Signals in Hierarchical Design.....	592
Figure 10-6. Global and Block-Specific Idle Signals and Multiple Clock Domains	593
Figure 10-7. Idle Signal Synthesis.....	593
Figure 10-8. Atrenta SpyGlassPower Flow	595
Figure 10-9. Atrenta SpyGlassPower Flow Output Files	596
Figure 10-10. Clock Gating Power Savings Table Column	596
Figure 11-1. CCOREs in the Architectural Constraints Window	599
Figure 11-2. Top-Down CCORE Architectural Constraints	601
Figure 11-3. Gantt Chart with CCOREs	603
Figure 12-1. Nodes of Interest in the FLOWS_API Database Hierarchy	616
Figure 12-2. Database Paths and Corresponding Design Blocks	617
Figure 13-1. Accelerated Library Flow	625
Figure 13-2. Altera DSP Block Alt_4mult_add Structure.....	628
Figure 13-3. Controlling Add/Sub for DSP Block Products	630
Figure 13-4. StratixII High-level Architecture	635
Figure 13-5. Using Extra Input and Output Registers to Reduce Routing Delays	636
Figure 13-6. Xilinx DSP48E Block	639
Figure 13-7. Virtex-5 High-level Architecture	642
Figure 13-8. Using Extra Input and Output Registers to Reduce Routing Delays	643
Figure B-1. Makefile Popup Menu	658
Figure B-2. Flow Package: SCVerify Dialog Box	660

Chapter 1

Catapult Introduction

Catapult® C Synthesis (Catapult) is a high-level synthesis (HLS) tool that creates RTL implementations from compatible C, C++, and SystemC design specifications.

The C, C++, and SystemC design specification describes the structure and behavior of the design in such away that Catapult can synthesize the interfaces, data structures, and loops to a specified ASIC or FPGA technology and produce an optimized RTL implementation.

Catapult produces an RTL implementation that is ready for simulation and gate-level synthesis and includes the following files:

- **HDL files** — For each target output language (VHDL, Verilog, and SystemC), Catapult creates a two types of HDL output: *cycle* and *rtl*.
- **RTL simulation and synthesis scripts** — Simulation and synthesis scripts for specified downstream tools such as Mentor Graphics Questa/ModelSim or Synopsys DesignCompiler.
- **Analysis reports** — A cycle report and an RTL report.

Supported RTL synthesis tools include Mentor Graphics Precision RTL Synthesis, Synopsys Design Compiler, Cadence RTL Compiler, and Magma Talus Design.

Supported HDL Simulators include Mentor Graphics Questa/ModelSim, OSCI, NCSim, VCS, and Vista.

Note



From this point forward, this manual uses the term *source code* as a generic term to mean all three of the supported programming languages including C, C++, and SystemC. Unless otherwise specified, all features support all variations of the supported source code.

Catapult Features

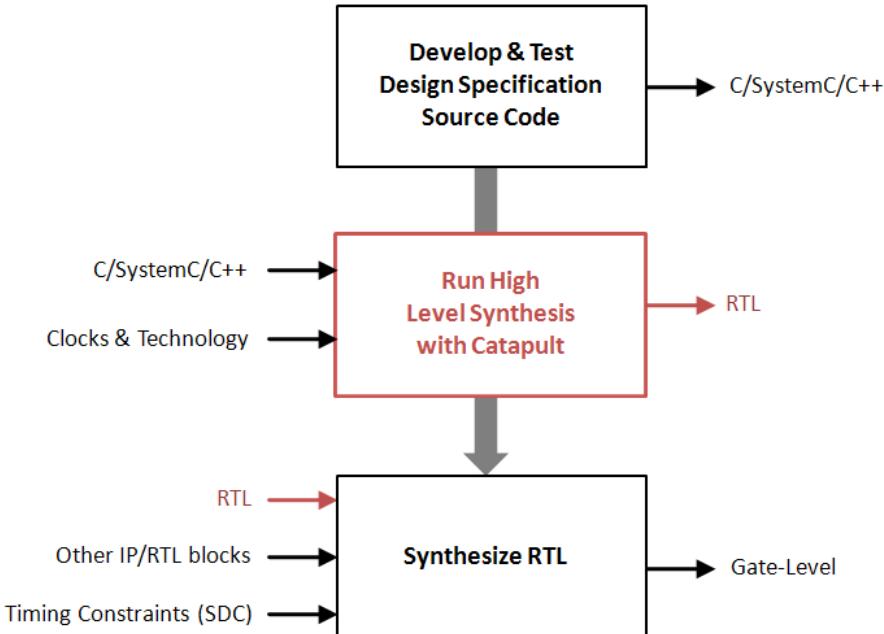
Catapult provides the following features to support the HLS process:

- **Bottom-up and Top-down synthesis strategies** — Supports synthesizing multi-block, hierarchical designs. Blocks can either be mapped to presynthesized units in a bottom-up manner or synthesized from the source code using the standard flow in a top-down manner. Hierarchical design is available in Catapult SL only.

- **Reusable CCORE logic** — Creates and reuses CCOREs (Catapult C Optimized Reusable Entities) to optimize the synthesis process, improve the processing of complex designs, and increase synthesis capacity. A CCORE is a custom operator synthesized from a single source code function that is reused throughout the design without being resynthesized. CCOREs are available in Catapult SL only.
- **Integrated verification flows** — Provides several integrated verification flows that generate makefiles and the necessary infrastructure files to verify your input source code and to verify the RTL output against the original input source code.
- **Customizable design flows** — Allows you to define custom flows that launch other programs, generate specialized output files, or post-process output files.

Catapult and the Design Flow

Catapult produces the initial RTL implementation in the IC design flow as shown in the following figure highlighted in red:



Design integration takes place at the RTL level, and the Catapult output is designed to merge into your existing RTL design flow.

Catapult User Interface

There are several methods for interfacing with Catapult to set up and control the synthesis process including:

- **Graphical User Interface (GUI)** — guides you interactively through the synthesis process including setting up and running synthesis, debugging your input source code, and visually analyzing and optimizing your design. For more information, see “[Getting Started](#)” on page 43.
- **Command-line/Scripts** — allows you to run the entire synthesis process interactively from the command line or through a Tcl script in batch mode with a comprehensive set of directives and Tcl commands. For more information, see “[Directives](#)” on page 299 and “[Commands](#)” on page 349.
- **Input Source Code** — allows you to control the synthesis process via programming styles, pragmas, and directives imbedded in your design source code. For information on programming styles and creating source code for high-level synthesis with Catapult, see the following manuals:
 - [Catapult C Synthesis C++ to Hardware Concepts](#)
 - [Catapult C Synthesis SystemC User’s Guide](#)
 - [High-Level Synthesis Blue Book](#)

About this Manual

This manual describes concepts and instructions for HLS using Catapult for the following product versions:

- **Catapult SL** — Supports all features.
- **Catapult BL and UV** — Supports all features except: SystemC, hierarchical designs, and CCOREs.

Content applies to all versions unless specifically stated otherwise.

This manual is intended to help logic designers and engineers use Catapult to perform high-level synthesis for ASIC and FPGA designs. An understanding of the high-level synthesis process, hardware description language such as Verilog or VHDL, and the syntax and semantics of the source code language used in your design specification is required.

Chapter 2

Designing with Catapult C Synthesis

This chapter provides an overview of the Catapult approach to high-level synthesis including the process for creating optimal source code and Catapult synthesis, analysis, and verification tool usage and capabilities. Catapult is designed to integrate in to your process for developing and testing IC designs and hardware.

The Catapult Design Process

The basic design process is broken into the following steps:

Step 1: Writing and Testing the Source Code.....	27
Step 2: Analyzing the Algorithm	28
Step 3: Creating the Hardware Design	31
Step 4: Performing Timed Simulations	33
Step 5: Synthesizing the RTL Design.....	33
Step 6: Analyzing Power Consumption	34
Step 7: Incorporating Post-Synthesis Design Changes.....	34

Step 1: Writing and Testing the Source Code

The first step in the process of building hardware is to develop the algorithm behind the hardware. You can use drawings, MATLAB, C++ or SystemC code or some other abstract design language to model the design. The algorithm impacts the final hardware more than any other step in this process, so an effective algorithm is essential.

Catapult C Synthesis requires either untimed ANSI C++ or SystemC as an input. The code allowed for synthesis is a subset of standard C++ including all the operators, classes, templates and other structures in the language. Catapult doesn't support dynamic memory allocation, so "malloc," "free" and function recursion are not allowed. Catapult also has some minor restrictions about how pointers are used, but any algorithm can be written within these restrictions.

In order to synthesize designs written in standard C/C++ (non-SystemC), the top-level function of the design must be identified. That can be done in the source code by inserting the pragma `"#pragma hls_design top"` immediately before the top-level function declaration. If the pragma is not used, the top-level function can be specified interactively from within Catapult after the design is loaded. All sub-functions called by top-level function are synthesized into hardware

and everything outside of that function is considered to be software or a test bench and is ignored during the synthesis process.

Catapult provides an integrated text editor and includes more in-depth C++ checking than standard C++ compilers, so the best approach is to use Catapult while developing your algorithm.

The C++ code can be tested using any standard C++ simulation framework. Catapult also provides the SCVerify flow, an internal verification flow that can simulate your C++ design and validate the cycle-accurate (or RTL) netlist output from Catapult against the original C++ input using the user-supplied C++ testbench.

Step 2: Analyzing the Algorithm

Catapult provides a suite of algorithm and design analysis tools. The algorithm is analyzed with respect to the target hardware and clock speed because these constraints play an important role in determining how an algorithm is structured.

Setting Loop Constraints

Click **Architectural Constraints** on the Task Bar to display the architectural constraint options in the Constraint Editor window. The left side of the Constraint Editor window is a hierarchical representation of the design that provides information about the hardware inferred from the C++ including interfaces, data structures, and loops. Selecting an item in the hierarchical view displays the editable options and constraints for that item on the right side of the window.

Catapult runs a *preliminary* analysis of the design to determine the number of iterations in each loop in your design. A number that is larger or smaller than expected can point to an error or inefficiency in your algorithm. If Catapult can not determine the number of iterations, you should modify the algorithm or add an iteration constraint.

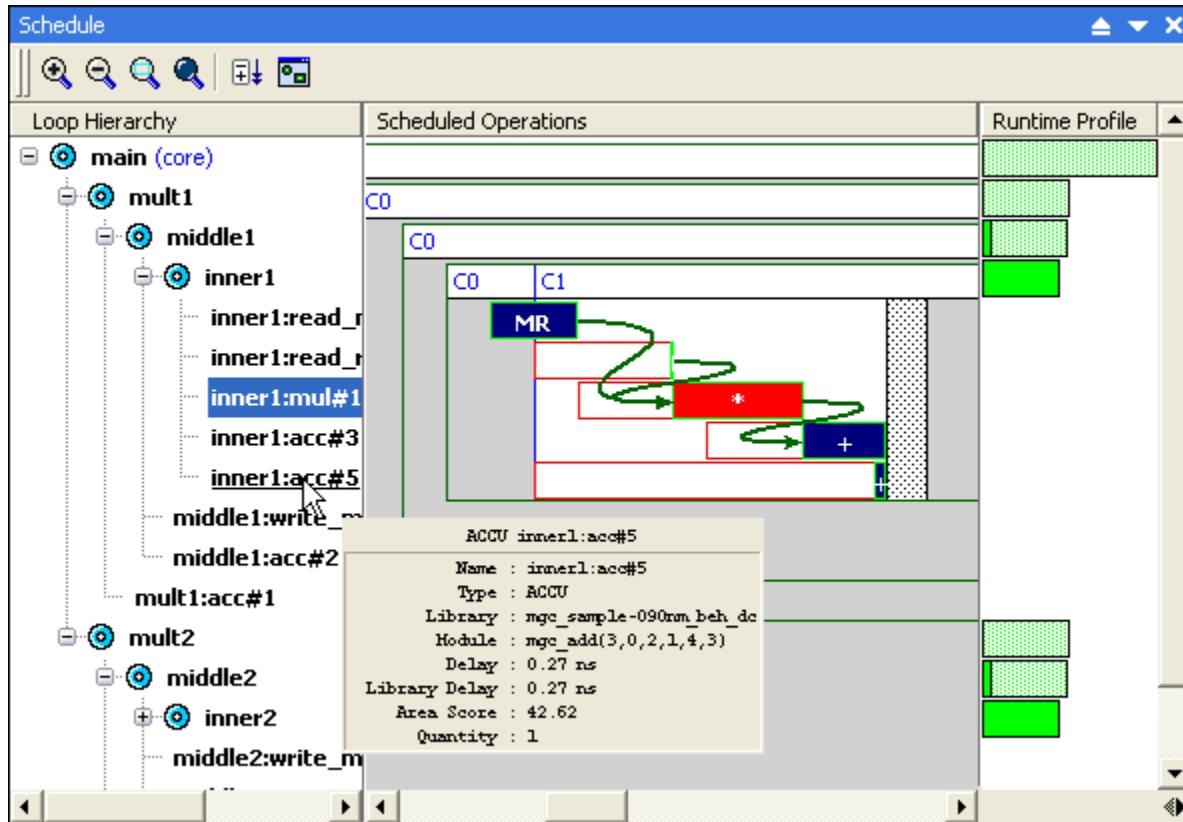
Viewing the Algorithm

Click **Schedule** on the Task Bar to display a Gantt chart in the Schedule window as shown in [Figure 2-1](#). The Gantt chart is like a schematic viewer for the algorithm in your design and provides information about how long your design will take to process information and where you might want to work on your algorithm.

In addition to scheduled operations, you get a full view of the functional units in your design. This is a view of the data path without the muxes used for sharing. You can look at these operations to verify the bit widths. Catapult optimizes the bit widths of all the variables in your design based on the preliminary analysis of the design. Some styles of C++ code prevent bit width optimization. In this case, you can change your loop constraints or your source code to quickly see the effect of these changes.

Also you can hover the cursor over an object to display database information about that object as shown in [Figure 2-1](#). For more information, see “[Scheduling the Design](#)” on page 127.

Figure 2-1. Gantt Chart with Data Objects Highlighted



Generating RTL Files

Click **Generate RTL** on the Task Bar to generate one or more RTL netlists (VHDL, Verilog or SystemC), report files, and control files for running downstream tools. As shown in [Figure 2-2](#), some of these reports are more hardware-centric, while others provide a quick summary of the algorithm characteristics. The reports also cross-probe back to the C source for easier analysis.

Figure 2-2. Output Reports for Design Analysis

The screenshot shows the 'cycle.rpt' output report window. The report is structured as follows:

- Solution Settings: dct.v1**
 - Current state: schedule
 - Project: Catapult
- Design Input Files Specified**
- Processes/Blocks in Design**
- Clock Information**

Clock Signal Edge	Period	Sharing Alloc (%)	Used by Processes/
rising	10.000	20.00	/dct/core
- I/O Data Ranges**

Port	Mode	DeclType	DeclWidth	DeclR
clk	IN	Unsigned	1	
rst	IN	Unsigned	1	
input:rsc:REGISTER_FILE.data_out	IN	Unsigned	9	
output:rsc:REGISTER_FILE.data_out	IN	Unsigned	11	
input:rsc:REGISTER_FILE.data_in	OUT	Unsigned	9	
input:rsc:REGISTER_FILE.addr_rd	OUT	Unsigned	6	
input:rsc:REGISTER_FILE.addr_wr	OUT	Unsigned	6	
input:rsc:REGISTER_FILE.re	OUT	Unsigned	1	
input:rsc:REGISTER_FILE.we	OUT	Unsigned	1	
output:rsc:REGISTER_FILE.data_in	OUT	Unsigned	11	
output:rsc:REGISTER_FILE.addr_rd	OUT	Unsigned	6	
output:rsc:REGISTER_FILE.addr_wr	OUT	Unsigned	6	
output:rsc:REGISTER_FILE.re	OUT	Unsigned	1	
output:rsc:REGISTER_FILE.we	OUT	Unsigned	1	
- Memory Resources**
 - Resource Name: /dct/input:rsc
 - Resource Name: /dct/output:rsc
 - Resource Name: /dct/coeff.rom:rsc
 - Resource Name: /dct/core/temp:rsc
- Multi-Cycle (Combinational) Component Usage**
- Loops**
- Loop Execution Profile**
- End of Report**

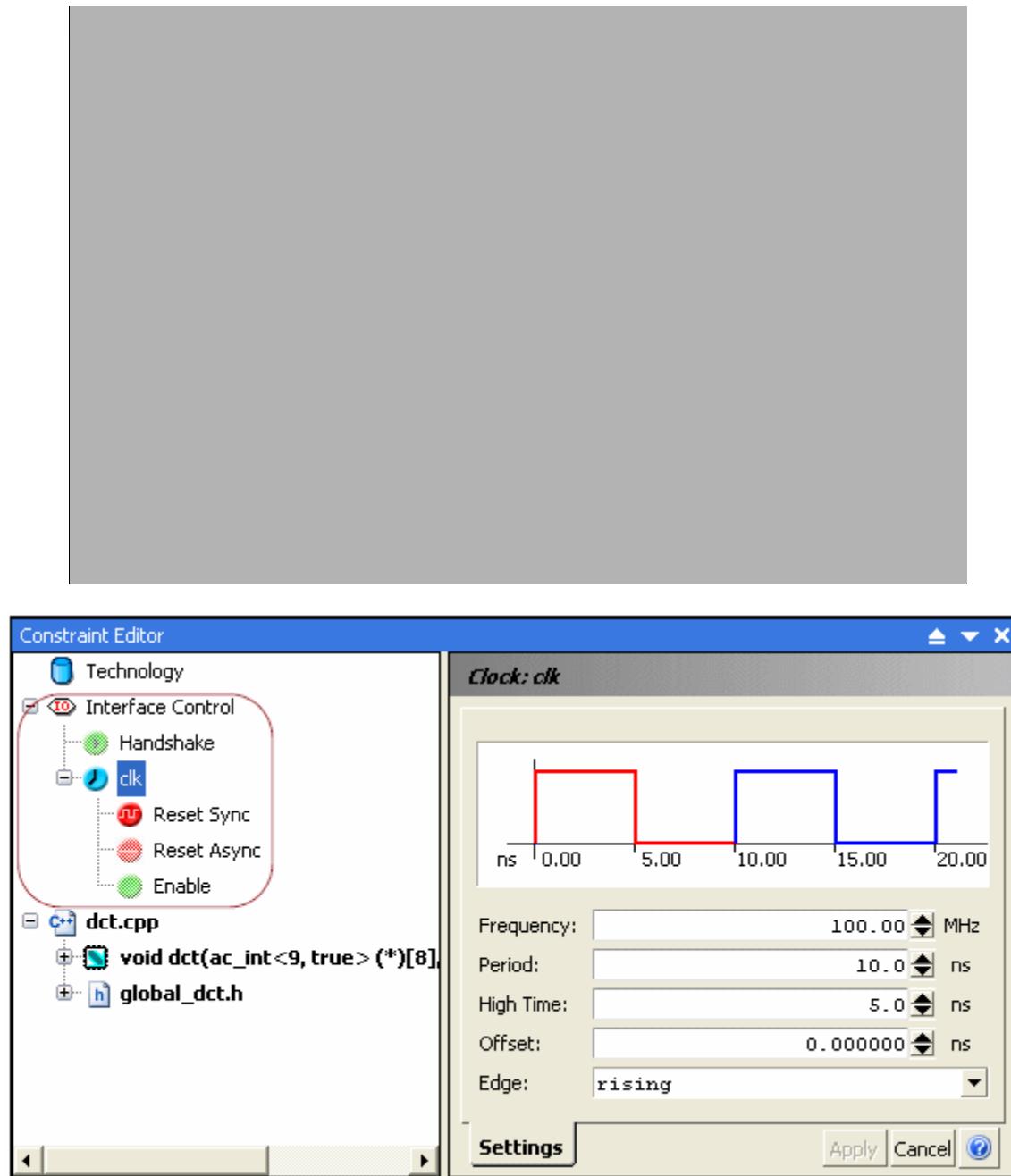
Step 3: Creating the Hardware Design

Once the algorithm is correct, you need to set up the hardware constraints. This entire process only takes a few minutes and can be changed any number of times for the same design. Catapult saves all work during a session, and you can save the project and return for a subsequent session.

Setting Global Hardware Constraints

Click **Setup Design** on the Task Bar to display the Interface Control settings from the Constraint Editor window as shown in [Figure 2-3](#). This is where you specify the clock frequency, reset and enable behavior and enable process level handshake signals. Select an interface element in the **Interface Control** section, and the editable constraints for that element display.

Figure 2-3. Interface Controls for Setup Design



Setting I/O, Loop, and Memory Architecture Constraints

Click **Architectural Constraints** on the Task Bar to edit the I/O, loop and memory architectures. Select a design element in the hierarchical view on the left and the associated constraints appear on the right.

Catapult infers I/O ports from the formal parameters of the top level function. For each I/O parameter, a port *resource* is created to associate data variables with their respective hardware components. For example, an I/O resource can be mapped to any one of a number of wire protocols or memories (if the variable is an array). This process is referred to as *Interface Synthesis*. For more information, see “[Interface Synthesis](#)” on page 39.

Other types of resources are channels, constant arrays (ROMs), and local arrays inside subprocesses. You can use the Constraint Editor window to change the component mapped to any resource, and to reassign variables to different resources. For more information about mapping memory arrays, see “[Memories and Arrays](#)” on page 248.

Finally, by selecting loops in the hierarchical view, you can unroll or pipeline them to add parallelism and make the design faster. This is also where you set up the design for a throughput constraint, if needed. For more information, see “[Loop Constraints](#)” on page 225.

Reviewing Design

Click **Schedule** on the Task Bar to schedule the design. Then use the Gantt chart to see how the operations in your design are scheduled. The Gantt chart displays a state by state account of where all the operations in your design will end up. For more information, see “[Scheduling the Design](#)” on page 127.

Generating Reports and RTL Files

Click **Generate RTL** on the Task Bar to generate reports, netlist files, and other output files for RTL synthesis. For more information, see “[Generating the Output Files](#)” on page 141.

Step 4: Performing Timed Simulations

Catapult provides both a behavioral and RTL level output. Both these outputs simulate exactly the same at their interfaces. The behavioral output is for simulation only and simulates at about 30 times faster than RTL. Since the output of Catapult is standard VHDL, Verilog, or SystemC, your normal testing flow can be used to verify the output is correct.

In addition, Catapult provides an integrated SystemC verification flow that automates the process of validating the cycle-accurate (or RTL) netlist output from Catapult against the original source code input. See “[C/C++ Designs and SCVerify](#)” on page 516.

Step 5: Synthesizing the RTL Design

Catapult integrates with many synthesis products. All of the required constraints are written to a file that can be used with these products to automate the synthesis of your design.

Step 6: Analyzing Power Consumption

Catapult integrates third party power analysis tools to generate all the necessary data files and driver scripts to run your design through the target power analysis tool and capture the results. For more information, refer to “[Power Analysis and Optimization](#)” on page 583.

Step 7: Incorporating Post-Synthesis Design Changes

Catapult provides an *incremental update* flow to help you implement ECO or other design changes after synthesis. The incremental update flow allows you to make small functional design changes and resynthesize only the affected design elements via an incremental solution.

An incremental solution is created from an existing (reference) solution, and Catapult applies the data from the reference solution for each transformation of the incremental solution.



Note

An incremental solution can be created from a top solution regardless of the last stage successfully completed.

Design Changes and Incremental Update

The incremental update flow minimizes RTL changes in cases of small functional design changes including:

- Changing a constant operand
- Changing a condition
- Adding or removing an operation

And smaller non-functional changes including:

- Changing a resource assignment
- Breaking a critical path by manipulating the schedule to move critical components

Incremental effectiveness decreases as the size of the corresponding RTL changes become larger. Larger functional changes ranging from bit-width changes to adding entire functions would result in large RTL changes and would benefit less from incremental mode. Similarly, larger non-functional changes such as pipelining, unrolling, and clock constraints are not feasible.

Incrementally Updating Your RTL Design

Use this procedure to incrementally update an RTL design for a small functional change after RTL synthesis.

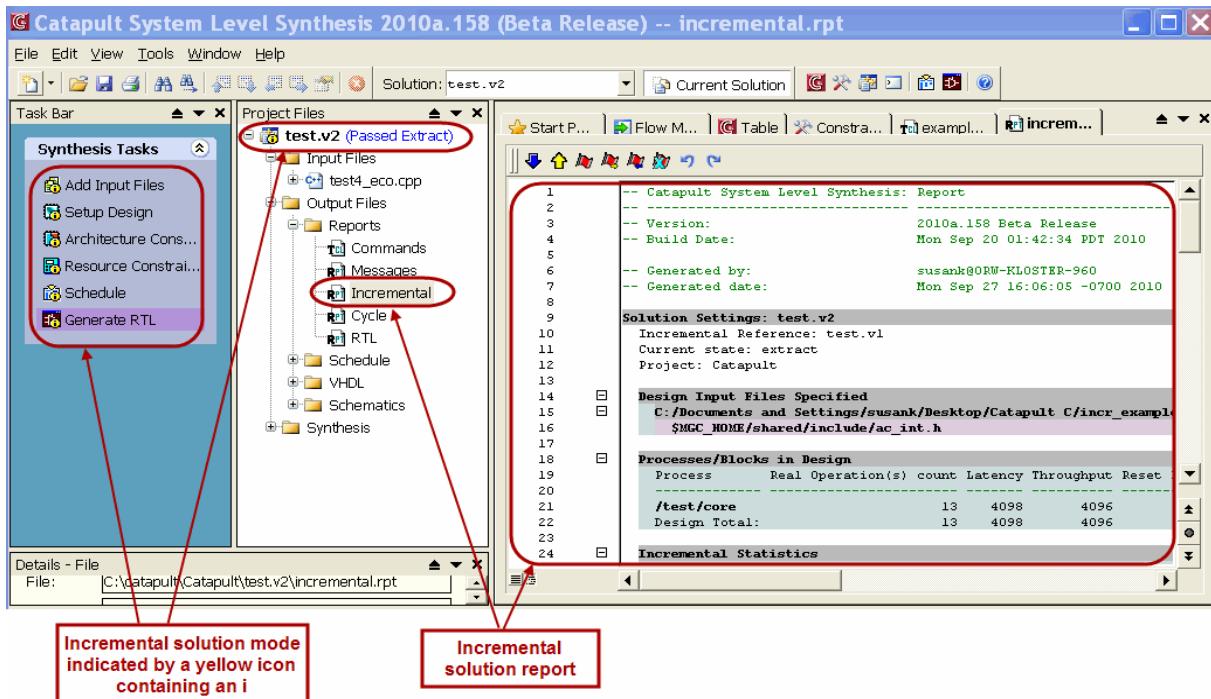
Prerequisites

- **Reference solution** — this is the original solution that produced the golden RTL design before the design change.
- **New version of design** — this is the revised, verified, and new golden C++ design.

Procedure

1. Invoke Catapult and open the project containing the reference solution.
2. In the Project Files window, right-click on the reference solution and select **Branch**. The Branch Solution dialog box displays.
3. Select the Incremental option, enter a name if desired, and click **OK**. An incremental solution is created and branched from the reference solution. The incremental solution file and icons listed in the Task Bar should display a small yellow circle containing an **i** to indicate incremental mode as shown in [Figure 2-4](#).

Figure 2-4. Incremental Update GUI



4. In the Project Files window, right-click on the original source design file and select **Remove**. The original source design file is deleted from the incremental solution.
5. Click **Task Bar: Input Files**. The Add Input Files dialog box displays.
6. Browse to and select the new version of the design source file and click **Open**. The new design file is added to the incremental solution.

7. Click **Task Bar: Generate RTL**. A new version of the RTL design with minimal changes is created.
8. Select **Output Files > Reports > Incremental** to display a report on the differences between the reference solution and incremental solution as shown in [Figure 2-4](#). Names of added objects are appended with *new* and objects whose data type changed are appended with *rev*.

Three Tcl scripts are generated and saved to the solution directory (not visible from GUI). These files can be used for advanced troubleshooting. For more information, see “[Incremental Update Files](#)” on page 171.

Writing Input Source Code

Most logic synthesis tools require an RTL source description of the overall architecture in hardware description language (HDL). This architecture description may contain thousands of lines of handcrafted code. The process of writing and testing the code is time consuming and error prone.

The Catapult C Synthesis tool can read designs created in C++ code. Just like RTL designs, the style of the C++ code has the biggest impact on the resulting hardware. C++ code is typically written for simulation and doesn’t include all of the information needed for synthesis. For example, many C++ models include extra code for fast C++ simulation, do not fully specify bitwidths, or have constructs that can’t be optimized.

Catapult requires some C++ code modification to get the best synthesis results. See the [Catapult C Synthesis C++ to Hardware Concepts](#) book for information on modifying C++ code for synthesis. Some basic concepts are:

- The tool requires that the top-level function be identified. One way to do that is to place a “#pragma [hls_design top](#)” immediately before the top-level function definition in the source code. Alternatively, the top-level function can be identified from within the Catapult tool when setting up the design. Refer to “[Getting Started](#)” on page 43 for more information.
- Dynamic memory allocation is not allowed (the tool can’t dynamically create memory resources).
- Hardware constraints can be specified in a *catapult.ini* file, or as pragmas in the design or interactively in the tool. Refer to the sections “[The Catapult Initialization File](#)” on page 224 and “[Pragmas](#)” on page 341 for more information.
- The *catapult.ini* file can be written out locally and modified for design constraints that don’t change often.
- Pragmas can be used for design specific things that don’t change much, such as loop pipelining and unrolling.

- Directives are used for constraints that change from solution to solution.

Figure 2-5 shows an example of C++ code that is suitable for running in Catapult.

Figure 2-5. C Code Design for Synthesis

```
#define num_taps 8

#pragma hls_design top
void fir_filter ( int *input,
                  int coeffs[num_taps],
                  int *output ) {
    static int regs[num_taps] = {0,};
    short temp = 0;

    for ( int i = num_taps-1; i >= 0; i-- ) {
        if ( i == 0 )
            regs[0] = *input;
        else
            regs[i] = regs[i-1];
        temp += coeffs[i] * regs[i];
    }
    *output = temp;
}
```

Catapult is an interactive tool which provides many features to help you understand issues with the hardware implementation and how the C++ code needs to be modified. Interactive constraints can be entered using scripts or entered directly from the Catapult command line.

For more information on what is required by Catapult and how to modify C code for synthesis, see the following sections in this chapter: “[Hardware Technology Constraints](#)”, “[Algorithmic Synthesis](#)”, “[Libraries](#)” and “[Interface Synthesis](#).”

Hardware Technology Constraints

The technology is a set of building blocks needed for synthesis. Within Catapult, technology defines the building blocks in the design including:

- **Basic blocks:** adders, subtractors, bitwise operators
- **IP blocks:** RAMs, pipelined multipliers

Libraries

You can also integrate IP by creating and adding various libraries using the Catapult C Library Builder. Library Builder is a standalone product that allows you to create, alter, and view libraries used by Catapult. For more information, see the [Catapult C Library Builder User's and Reference Manual](#).

Design Frequency

The design frequency defines the base timing constraint for the design that is used to calculate all other timing constraints. Coverage points include Reg-Reg, Input-Reg, and Reg-Output. I/O Pads are not considered for delay.

Catapult targets the frequency using the technology. Datapath delay is easy for Catapult to estimate, so datapath dominated designs are recommended. Control dominated designs are difficult to estimate and requires advanced Catapult skills.

To set the design frequency, click **Setup Design** on the Task Bar, and select **Technology** in the Constraint Editor window. For more information, see “[Setting Up the Design](#)” on page 96.

Algorithmic Synthesis

Most of the design interface and the performance of the hardware created is controlled by how the source C++ code is written, but some changes can also be made from Catapult.

Within the basic constraints discussed in this section, Catapult is able to build high-quality hardware to help automate synthesis. The tool contains many second generation synthesis algorithms designed to automate many of the manual tasks associated with hardware design. Catapult reduces bitwidth, generates datapath and FSM, facilitates sharing, and writes scripts for downstream synthesis tools.

Set Design Elements using Catapult

Catapult allows you to control how data is communicated to and from the hardware in relation to design area, performance, interfaces, and time. From the Catapult graphical user interface, you can control:

- Micro-architecture including unrolling, merging, or pipelining loops and mapping arrays to memory.
- Design latency and how many resources are used for the design.
- Data communication to and from hardware (interface synthesis).
- Design timing including latency, throughput, and I/O timing.

Automatic Synthesis Features in Catapult

Catapult takes care of the rest of the architectural synthesis including:

- Optimizes to minimize bitwidths and the number of hardware operators.
- Optimizes to minimize RAM and I/O accesses.

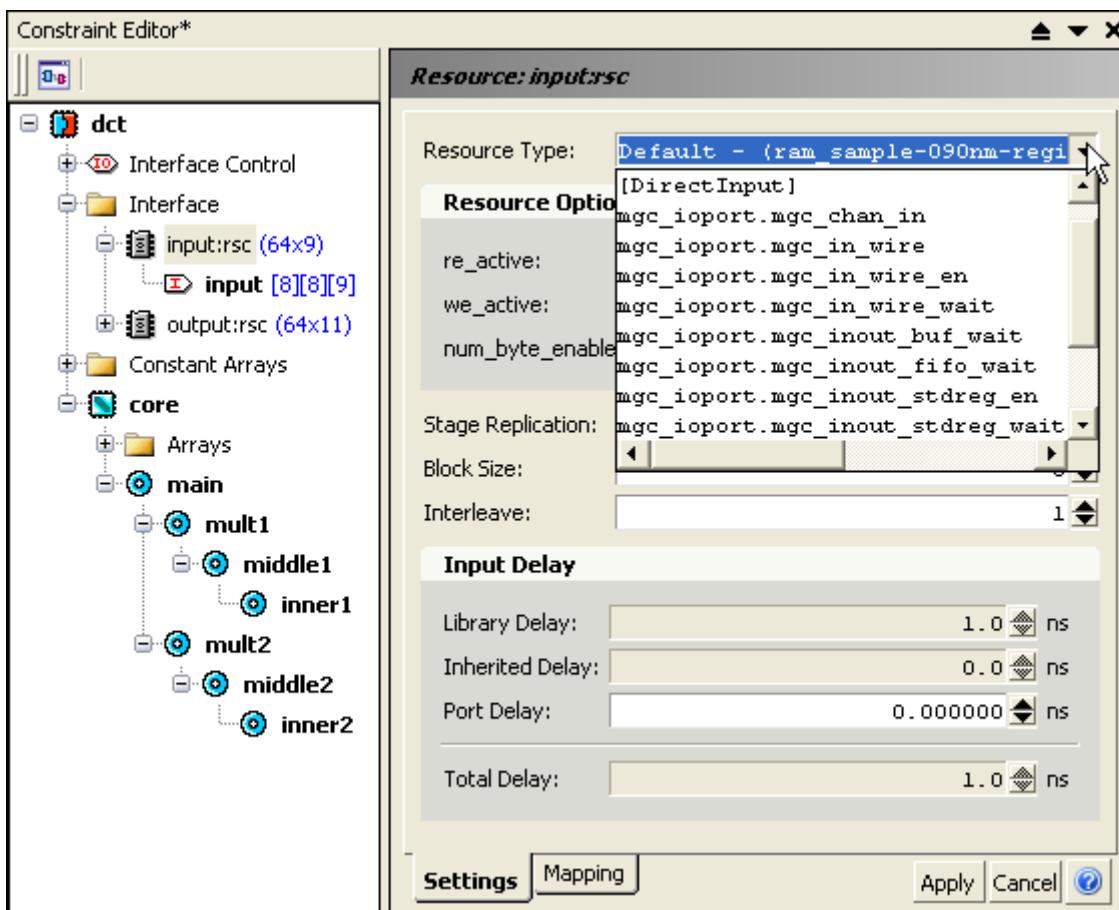
- Uses the constraints to add timing information to the design.
- Provides automatic RTL generation with scripts for a bottom-up RTL flow.

Interface Synthesis

Catapult allows you to control how data is communicated to and from the hardware with *interface synthesis*. The C++ code doesn't provide any timing information about the design interfaces. Using the Constraint Editor to set architectural constraints, you can control how the I/O is communicated by applying different I/O components (from a library of I/O components) to the interface resources as shown in [Figure 2-6](#). Select a resource and view the drop-down list in the **Resource Type** field.

You can also create custom I/O components using Catapult. For more information, see Chapter 6, [Interface Synthesis](#).

Figure 2-6. Interface Control in the Design



Adding Timing in a Design

The C++ code contains no information about time. Timing information is added during the synthesis process. Catapult automatically adds a *virtual wait* at the start of the top-level function call. The virtual wait is important for two reasons:

- All outputs are registered by default. Use Interface synthesis to change this setting.
- All storage elements have at least one clock cycle for read and write, which means no latches are inferred from the design style.

[Figure 2-7](#) illustrates where the virtual wait appears in a design.

Figure 2-7. Catapult Adds Virtual Wait to the Design

```
#pragma hls_design top
void fir_filter ( int *input,
                  int coeffs[num_taps],
                  int *output ) {

    <Virtual Wait>

    static int regs[num_taps] = { 0, };
    ...
}
```

For more information on how Catapult handles loops and timing, see “[Loop Constraints](#)” on page 225.

Frequently Asked Questions

What if my design doesn't meet timing after place and route?

The tools in the Catapult C Synthesis product line are designed to work together. Catapult C Synthesis creates an RTL design with information about the target technology and, unless an estimate is incorrect, the design will meet timing without any effort from RTL Synthesis or Physical Synthesis. If the Catapult C Synthesis estimates are incorrect, the down stream synthesis tools are very good at rectifying the problem.

If, after trying to fix a problem in RTL Synthesis and/or Physical Synthesis, you still can't meet timing, you can go back to Catapult and use the “[Percent Sharing Allocation](#)” constraint (discussed on page 274) to make the Catapult estimates more conservative. This will give RTL synthesis more slack to work with.

Why doesn't Catapult support unions?

Catapult does not support unions because:

- The packing of the different types into the union is not defined by the C standard. This means that unions function differently on different platforms and would require you to define the underlying memory architecture during compile.
- The packing of unions cannot easily be controlled on some platforms, leading to the need for complex pragmas to define how to pack unions. Catapult would need to support all of these pragmas and would generate hardware that functions differently in the case that the pragmas are not detected.
- Unions are not needed when designing hardware. The register sharing algorithms, for example, can automatically re-use registers in the same way that memory is re-used when a C program defines a union.

Chapter 3

Getting Started

This section describes how to manage projects from the Catapult graphical user interface (GUI) and begins with a description of the fundamental features of the Catapult tool. It covers the underlying concepts of how design data is managed (“[Understanding Catapult Projects and Solutions](#)” on page 43) and the intrinsic design flow built into Catapult (“[The Catapult Work Flow](#)” on page 46).

The following sections describe the significant features of the GUI in the order of the basic work flow:

- “[Invoking the Graphical User Interface](#)” on page 59
- “[The Catapult Session Window](#)” on page 60
- “[Setting the Working Directory](#)” on page 92
- “[Adding and Removing Input Files](#)” on page 93
- “[Preparing the Source Files](#)” on page 95
- “[Setting Up the Design](#)” on page 96
- “[Design Analysis in Catapult](#)” on page 100
- “[Specifying Architectural Constraints](#)” on page 104
- “[Scheduling the Design](#)” on page 127
- “[Generating the Output Files](#)” on page 141
- “[Using the Schematic Viewer Window](#)” on page 148

The remainder of the section covers other key features of Catapult that are not directly related to the work flow, such as “[Understanding Messages in the Transcript](#)” on page 159 and “[Understanding Catapult Project File Types](#)” on page 168.

Understanding Catapult Projects and Solutions

A Catapult *project* organizes and manages all of the files associated with a particular design. Projects can be saved and re-opened by other Catapult sessions. Upon invocation, Catapult provides a default project for you to begin working on a new design. You can also open a saved project from a previous Catapult session.

Catapult projects contain one or more *Solutions*. Each solution is a unique implementation of your design based on the set input files you provide, the target technology and the constraint settings you apply. Each solution contains a SIF (synthesis internal format) database representation of your design and all output files generated from by the solution. The **Project Files** window in the Catapult GUI provides a hierarchical view of the solutions.

Pathnames embedded in solution files are relative to the parent project directory. A solution directory cannot be copied to another location without also copying the project directory.

Branching Solutions

Catapult will automatically *branch* new solutions as needed while you work on your design and iterate through the Catapult work flow (see “[The Catapult Work Flow](#)“). In this way you can experiment with several implementations, each one isolated in it’s own solution.

Because the work flow is iterative in nature, whenever you go back to an earlier stage in the design flow and change any key settings (such as constraints or input files) that alter the implementation, Catapult branches a new solution for that implementation. More precisely, the new solution is branched from the currently active solution. That means the new solution is a clone the old solution, but includes the change(s) that caused the branching. The old solution is preserved, but deactivated (only one solution can be active at a time). Any inactive solution can be reactivated by using the solution selector on the tool bar in the GUI, or by issuing the [solution timing](#) command.

Saving and Restoring a Project

A Catapult C Synthesis Project File (.ccs) is created in the *working directory* (see “[Creating a Working Directory](#)“) when you select the pull-down menu **File > Save project**. You do not have to explicitly save the generated output files inside the project directory. Catapult automatically saves them when you initiate a command that acts on the current design, such as the “[go compile](#)” command. At that point, Catapult will create/update the appropriate files in the project directory. Catapult also creates/updates the session log file, “catapult.log” automatically.

To open a saved project, select the pull-down menu **File > Open Project...**, then select a project file and click the **OK** button. The project is restored to the state it was in when last saved.

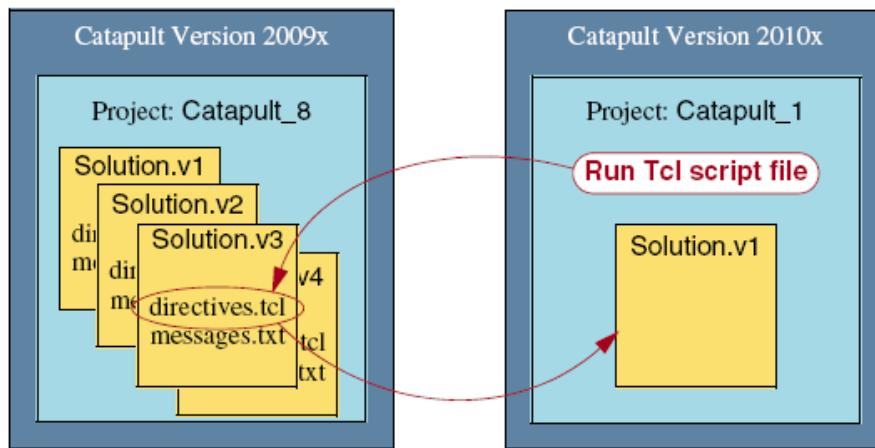
Migrating Design Solutions to Newer Catapult Versions

As Catapult modifies its functionality from one release to the next, projects that were created with older versions of Catapult may not be compatible with newer versions. However, Catapult provides an automated method for recreating your designs in newer versions of the product. Catapult automatically creates and updates a Tcl script (directives.tcl) in each solution directory that contains all of the directive commands, “[go](#)” commands, and input file paths needed to regenerate the solution. In addition, Catapult creates a text file (messages.txt) containing all of

the Catapult system messages generated by the solution. Both of these files are viewable from within the Catapult GUI in the **Output Files/Reports** folder.

The directives.tcl file is essentially a record of the design's evolution from its first solution through the current one. Each time a new solution is branched, the child solution inherits the directives.tcl from the parent. To recreate a solution in a newer version of Catapult, simply open the new version of Catapult and run the directives.tcl script in a new (empty) project. From the GUI, use the **File > Run Script...** menu item. From the command line, use the **dofile** command.

Figure 3-1. Migrating Solutions



Because the directives.tcl file is the accumulated set of directives for a series of solutions, it is possible that some directive commands in the file may become invalid as the design evolves. Such invalid commands will cause errors when the script is run. If after inspecting the error messages you decide the invalid commands can be ignored, run the dofile command again with the “-noerrors” switch to suppress the error messages.

If the pathnames for input files in the directives.tcl script are specified as relative paths, you may need to adjust them before running the script. Also, note that regenerated design output may differ from the original due to enhancements and new features available in the newer version of Catapult.

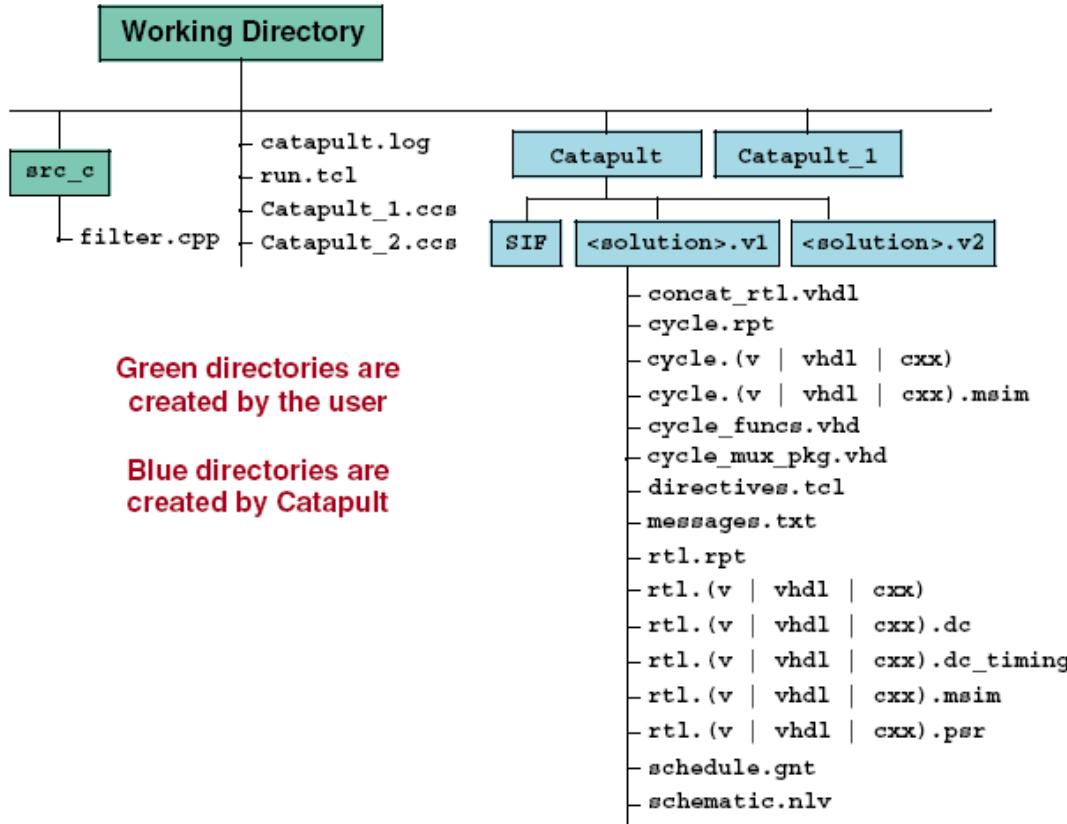
Creating a Working Directory

Create a directory in the file system for your Catapult project(s). This *working directory* is where Catapult sends all generated output files. For information about the output files generated by Catapult, refer to “[Generating the Output Files](#)” on page 141, and “[Understanding Catapult Project File Types](#)” on page 168.

Typically the working directory also contains your design source files. It is a good idea to create a different sub-directory for your source files to keep them separated from all of the generated files. Figure 3-2 illustrates the file structure that is created as Catapult progresses through the

design flow. The green directories are created by you, and the blue directories are created by Catapult.

Figure 3-2. File Structure of the Working Directory



In Figure 3-2, the sub-directory “src_c” on the left is for source files. All of the other files and sub-directories shown here are generated by Catapult. The files at the top-level of the working directory are Catapult control files, including a session log file (.log), project files (.ccs) and Tcl command files (.tcl). The directories “Catapult” and “Catapult_<n>” are project directories, each of which contains all generated files for that project. Within each project directory are *Solution* directories whose names have the form <name>.v<n>, where the default value for <name> is the name of the top-level function in the design, and <n> is an integer that is automatically incremented by Catapult. Project directories also contain a SIF (Synthesis Internal Format) directory.

The Catapult Work Flow

Catapult is designed around the basic work flow illustrated in Figure 3-3. The tool helps guide you through the flow and automatically ensures the integrity of the design data through each stage.

The Catapult GUI provides the Task Bar, shown in [Figure 3-3](#), steps you through the work flow. The *command icons* on the Task Bar help you in two ways. First they give you a visual cue about where you are in the work flow, and second they launch commonly used commands with a single click. Each command icon corresponds to a particular stage in the flow. Clicking on a command icon invokes the necessary command(s) to advance the design to the corresponding stage. For more information, refer to “[The Task Bar](#)” on page 67.

Data integrity is managed by the tool’s underlying *state model*, which consists of eight states as illustrated in [Figure 3-4](#) on page 49. As you move your design through the flow, the internal design database transitions through the corresponding states. The Task Bar and/or the “[go](#)” command can be used to move between states.

For example, if the current state is “Passed Analyze”, then clicking on **Architectural Constraints** on the Task Bar (or entering the “go architect” command) moves the flow forward to the “Architect” state. To go backward in the flow, simply click on an earlier task on the Task Bar (or enter the “go <state_name>” command). If you move to a previous state and change a design constraint, Catapult will automatically branch a new solution. The GUI displays the current state of the active solution in the **Project Files** window (see [Figure 3-8](#) on page 62).

Refer to the following sections for detailed descriptions about each stage of the design flow:

- “[Preparing the Source Files](#)” on page 95
- “[Setting Up the Design](#)” on page 96
- “[Specifying Architectural Constraints](#)” on page 104
- “[Scheduling the Design](#)” on page 127
- “[Generating the Output Files](#)” on page 141

Figure 3-3. Synthesis Flow

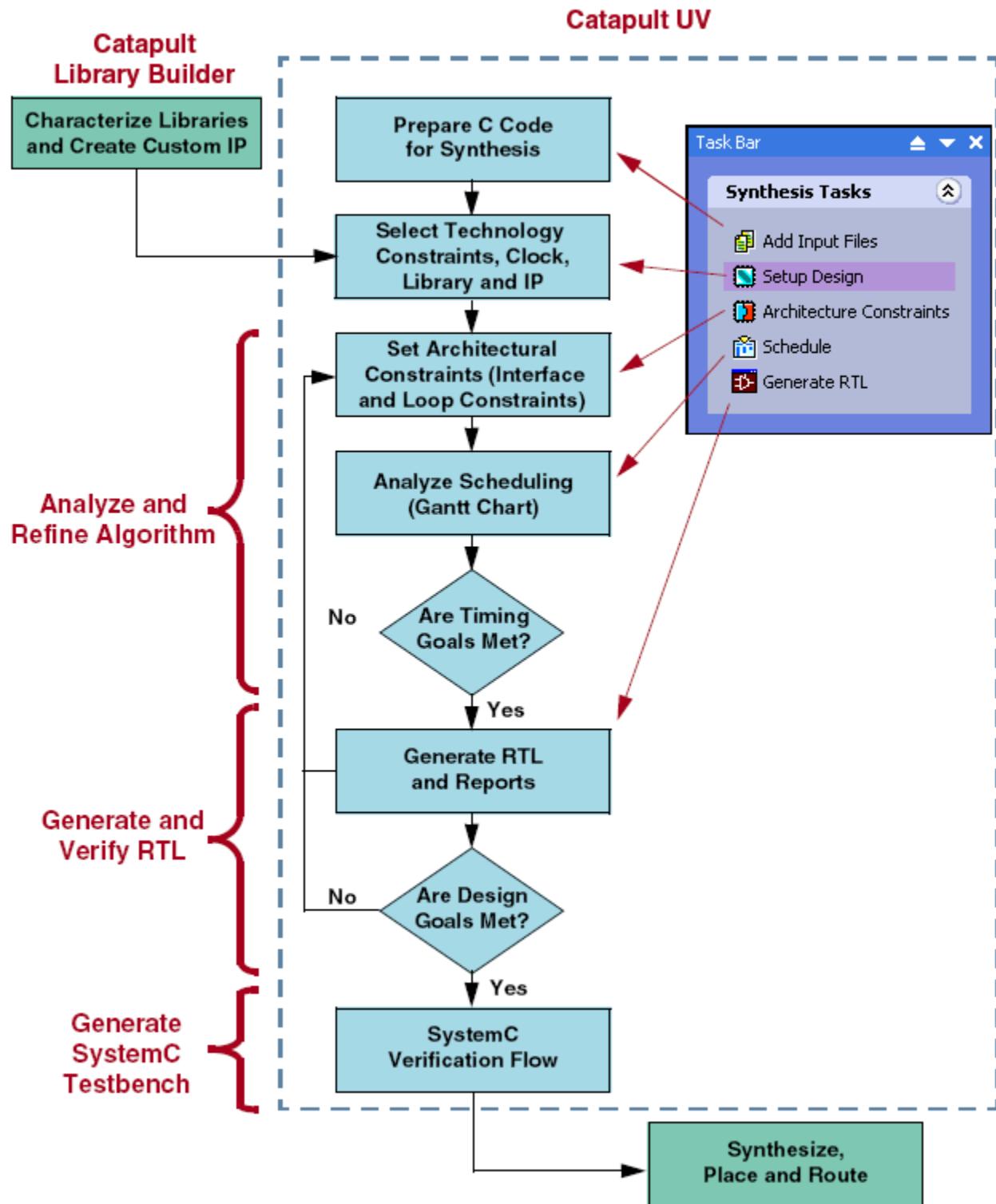
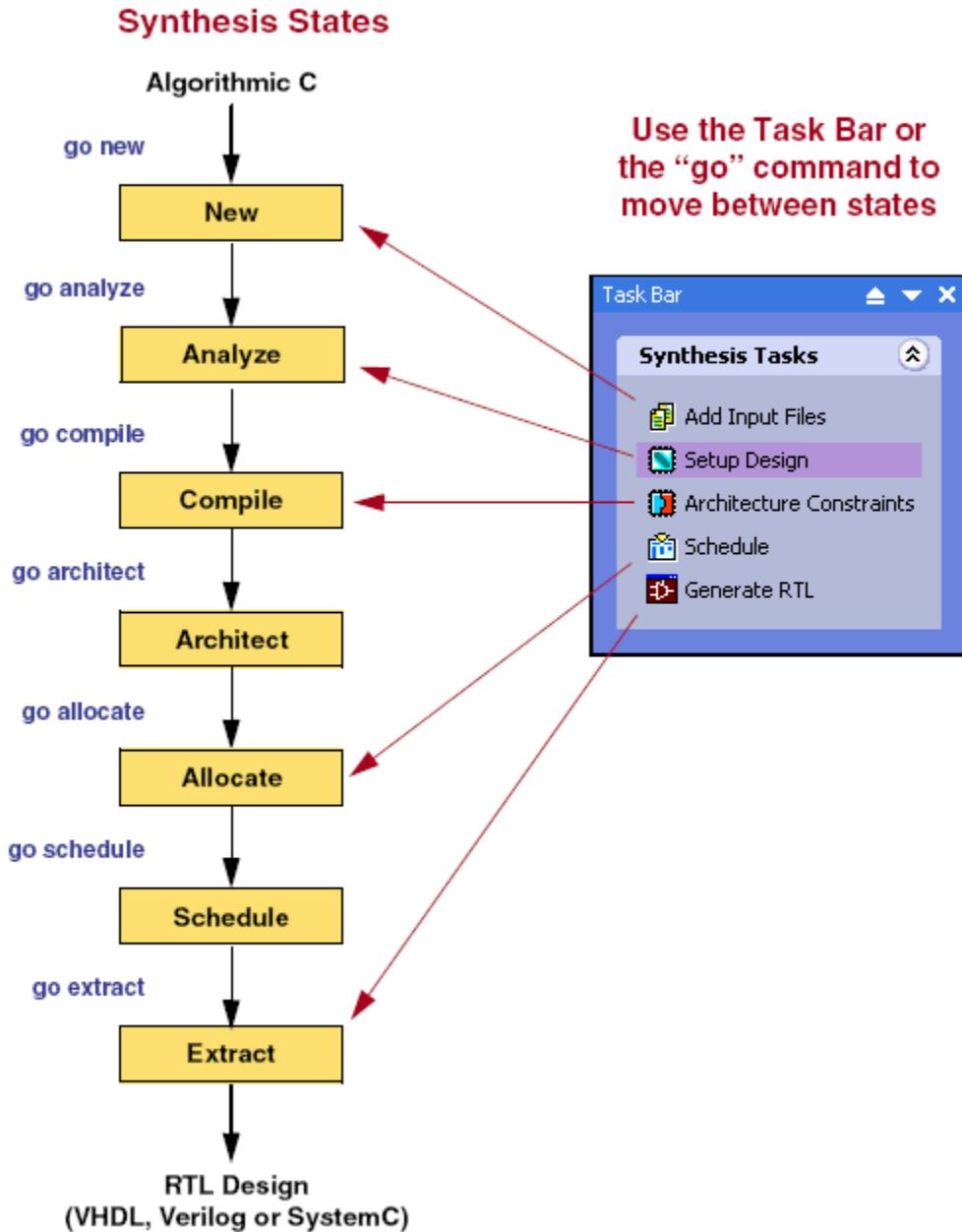


Figure 3-4. Data Transition States in the Synthesis Flow



Format of Catapult Generated Object Names

Translating a C++ algorithm into a synthesizable hardware description involves a series of transformations of the design elements (functions, loops, variables, operations, conditions and so on) into uniquely named instances of components in the schematics, schedules, netlists and other files output by Catapult. In addition to being unique, the names generated by Catapult are also intuitively correlated with the original C++ design elements. The correlation makes it easy

to locate all occurrences of an instance in any of the output files as well as its source in the C++ code.

Catapult's rules for generating names are described in the following sections. Familiarity with Catapult's naming scheme will improve your comprehension of Catapult-generated design files, and make it easier for you to construct Tcl commands to drive the Catapult tool interface.

General Naming Policies	50
Names for Architectural Elements	51
Names for Operations	56
Names in Schematics and Netlists	57

General Naming Policies

- **Uniqueness**

If there are multiple objects of the same type in the same scope level, the first occurrence gets the default name, and subsequent occurrences are made unique by adding the “#<int>” suffix. This applies to scopes such as loops and conditional branches and to operations.

For example, multiple occurrences of a for loop within a function appear as:

```
for, for#1, for#2, ...
```

Multiple accumulate operations appear as:

```
acc, acc#1, acc#2, ...
```

Multiple registers in a schematic appear as:

```
reg, reg#1, reg#2, ...
```

- **Struct Elements**

Struct names are similar to the dereference format: variable.element0, variable.element1, variable.element2, and so on. Nested structs follow a similar scheme. Example: if variable.element3 is also a struct, the objects are named variable.element3.element0, variable.element3.element1, and so on.

- **Class Methods**

When a class method is inlined or created as a sub-block, the generated name has the form “<object>.<method>”. For example:

```
template <typename T>
class wait_in {
    T d_data;

public:
    T get() const { return d_data; }
    ...
}
```

```

};

SC_MODULE(top) {
    wait_in<int> in1;
    wait_in<char> in2;
    ...
}
void process() {
    int val = in1.get();
    ...
}

```

In the example above, the modular I/O block for the “get()” method will be named in1.get().

Names for Architectural Elements

The general algorithm for constructing unique names for architectural elements (blocks, ports, RAM/ROM memories, data channels and loops) is to start with the C++ names (functions, variables, scope labels and loop key words) and then append a *type* suffix and *scope* prefix.

Note

 While reading the following descriptions, refer to [Figure 3-5](#) on page 55 and [Figure 3-6](#) on page 56 to see how the architectural element names in the dct_hier.cpp example design will appear in the Catapult Architectural Constraints window.

- **Design top**

Takes the name of the top-level function. Example:

```
#pragma hls_design top
void my_top_function(....)
{
    ...
}
```

The generated design name is “my_top_function.”

- **Process blocks**

The name “core” is given to the logic process block. Cross-probing the core block will resolve to the top-level function in the source code. Example:

```
#pragma design top
void my_top_function (....)
{
    ...
    subprocess1();
    subprocess2();
}
```

The process block is named “core.”

- **Resources**

The suffix “:rsc” is added to all resource objects (I/O ports, internal channels, and arrays). In the following descriptions, <variable> represents the C++ variable name for which the resource was inferred, and <process> is the process block name as described above.

- **Interface:** Port resources are named <variable>:rsc. Port variables are simply named <variable>.
- **Interconnect:** Inferred Channel resources are named <variable>.chn:cns, and the Channel variables are named <variable>.chn.
- **Ac_channel:** Ac_channel resources are named <variable>:cns, and the variables are simply named <variable>.
- **Constant Arrays:** Const array resources are named <variable>:<process>.rom:rsc, and the array variables are named <variable>:<process>.rom.
- **Arrays:** Array resources are named <variable>:<process>:rsc, and the array variables are named <variable>:<process>.

- **Loops**

- **Implicit “Main” Process Loops**

Every process block has an implicit “main” loop at the top-level. The “main” loop represents the active state of the process. In hardware when the process is started (a function call in C++), it runs indefinitely like an infinite loop.

- **Loops with C++ Scope Labels**

Since C++ labels must be unique within their scope in the program, Catapult simply uses the label name as the name of the loop.

Example:

```
void subfunction(...)  
{  
    if (...)  
    {  
        LOOP1:for (x = 0; x < X; x++)  
        {  
            LOOP2:for (y = 0; y < Y; y++)  
            {  
                LOOP3:for (z = 0; z < Z; z++)  
                {  
                    ...  
                }  
            }  
        }  
    }  
}
```

The resulting loop names are:

```
LOOP1
LOOP2
LOOP3
```

Labelling loops in the source code is highly recommended for complex designs. Custom label names can express more meaningful information about the purpose of the loop and are inherently unique.

- **Loops without C++ Scope Labels**

Loop names have the form [*<scope>*:]*<type>*, where *<type>* is one of the following: *for*, *while* or *do*. The *<scope>* qualifier is used for loops that occur in conditional statement branches. The scope qualifier describes the C++ scope of the loop as illustrated in the following example.

```
if (...)
{
    for (int x = 0; x < 10; x++)
        ...
}
else if (...)
{
    for (int x = 0; x < 10; x++)
        ...
}
else if (...)
{
    for (int x = 0; x < 10; x++)
        ...
}
```

The respective names of the three for loops in the example will be:

```
if:for
else:if:for
else:else:if:for
```

- **Loops Within Switch Statement Cases**

The case branches become the scope qualifiers for the loop names. Example:

```
switch (s)
{
    case 132:
    case 153:
        for (int x = 0; x < 10; x++)
            ...
        break;
    case 171:
        for (int x = 0; x < 10; x++)
            ...
        break;
}
```

The for loops are named:

```
case-132:for
case-171:for
```

Note in the example that two case branches resolve to the first for loop. In that situation, only the first branch name is used as the scope qualifier.

- **Loops Within Sub-Functions**

Loops that are in nested sub-functions get the name of the innermost sub-function as a scope qualifier. Sub-function loops that are inside a conditional branch get both the sub-function name and the conditional scope, as illustrated in the following example.

```
void bottom(...)
{
    for (x = 0; x < X; x++)
        ...
}

void middle(...)
{
    if (...)
        bottom(...);
}

#pragma hls_design top
void top(...)
{
    if (...)

        middle(...);
}
```

The loop is named:

```
bottom:for
```

- **Nested Loops**

Each level of nesting adds another prefix to the scope qualifier of the loop name.
Example:

```
for (x = 0; x < X; x++)
{
    for (y = 0; y < Y; y++)
    {
        for (z = 0; z < Z; z++)
        {
            ...
        }
    }
}
```

The loops are named:

```
for
for:for
for:for:for
```

- **Reset Loops**

Reset loops appear in the Gantt chart (Schedule window). Their names have the format <process>:rlp, where rlp is an abbreviation for *reset loop*.

Example Design

Figure 3-5 shows the source code for the dct_hier.cpp dct.cpp example design and highlights the key architectural element names in red. Figure 3-6 shows how those element names display in the Catapult Architectural Constraints window.

Figure 3-5. Architectural Element Names in Source Code

```
## dct.cpp

#include "global_dct.h"

#pragma design top
void dct(ac_int<9> input[XYSIZE] [XYSIZE], ac_int<11>
output[XYSIZE] [XYSIZE]) {

const ac_int<10> coeff[XYSIZE] [XYSIZE] = {
    362, 362, 362, 362, 362, 362, 362,
    502, 425, 284, 99, -99, -284, -425, -502,
    473, 195, -195, -473, -473, -195, 195, 473,
    425, -99, -502, -284, 284, 502, 99, -425,
    362, -362, -362, 362, 362, -362, -362, 362,
    284, -502, 99, 425, -425, -99, 502, -284,
    195, -473, 473, -195, -195, 473, -473, 195,
    99, -284, 425, -502, 502, -425, 284, -99
};

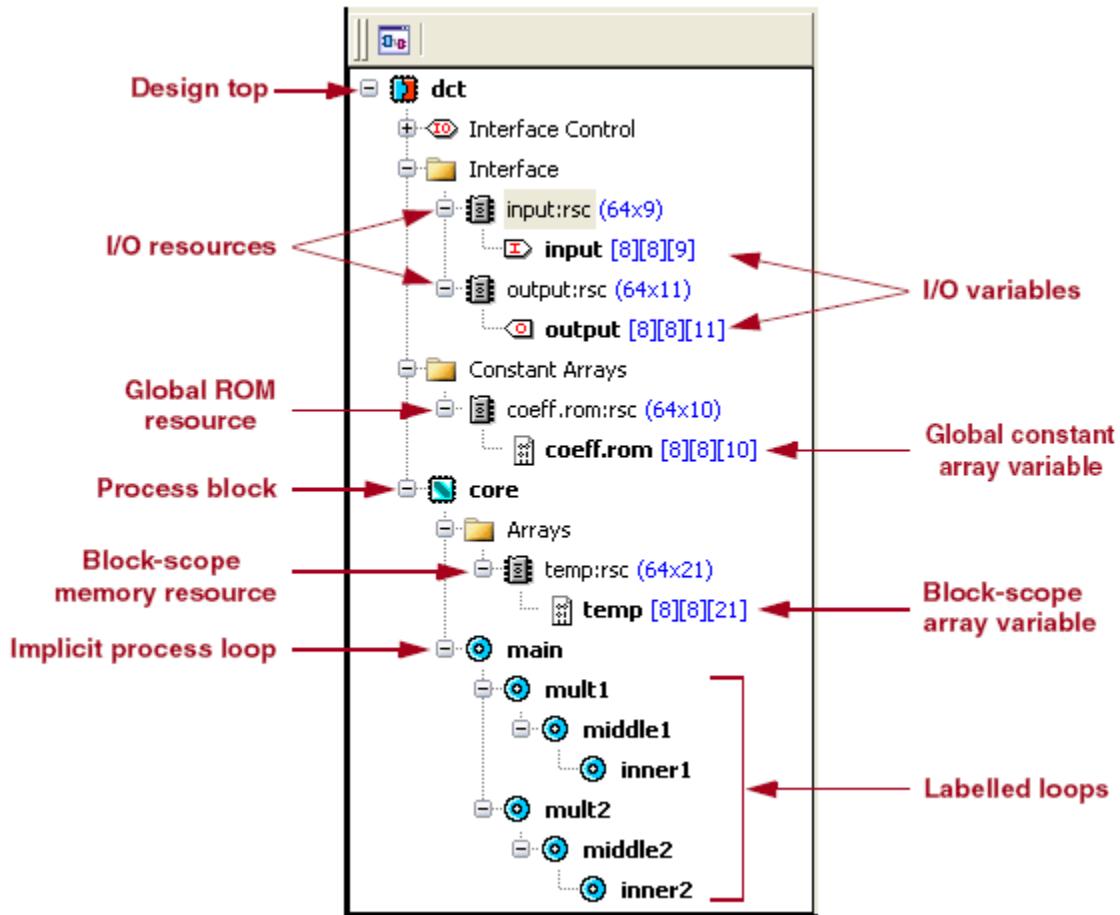
ac_int<21> temp[XYSIZE] [XYSIZE];
ac_int<21> tmp;
ac_int<31> dct_value;

mult1:for (int i=0; i < XYSIZE; ++i )
    middle1:for (int j=0; j < XYSIZE; ++j ) {
        tmp = 0;
        inner1:for (int k=0; k < XYSIZE; ++k )
            tmp = tmp + input[i][k] * coeff[j][k];
            temp[j][i] = tmp;
    }

mult2:for (int ii=0 ; ii < XYSIZE; ++ii )
    middle2:for (int j=0; j < XYSIZE; ++j ) {
        dct_value = 0;
        inner2:for (int k=0 ; k < XYSIZE ; ++k )
            dct_value = dct_value + coeff[ii][k] * temp[j][k];
        output[ii][j] = dct_value >> 20;
    }
}
```

}

Figure 3-6. Architectural Element Names in GUI



Names for Operations

Logical operations, such as accumulate, add, multiply, read and write have names of the form `[<scope>:]<oper_type>`, where `<oper_type>` is an abbreviated name for the operation. Some examples of `<oper_type>` names are: add, acc, mul, mux, not, io_read and io_write. The `<scope>` prefix is added to the name if necessary.

Examples:

```
LOOP1:acc#4
LOOP2:if:acc#3
for:mul#1
```

Read and write Operation, io_read, io_write, read_mem and write_mem, are named using the related resource. Examples:

- I/O read of port A inside a nested loop:

```
for:for:io_read(A:rsc.d)
```

In this case the “.d” suffix is a placeholder variable for all variables mapped to that resource.

- Write to the “temp” variable, which is a ping-pong RAM in a loop labeled inner2:

```
inner2:write_mem(temp.chn.pp0:rsc.d)
```

Note



In the Catapult graphical user interface, logical operations appear in the Gantt chart (Schedule window). Only the operation name, not the full path, is displayed. You will need the full path to construct commands like `cycle` and `ignore_memory_precedences`. The full path is found by right-clicking on an operation and choosing **Copy Operation Path to Clipboard**.

Names in Schematics and Netlists

Catapult derives variable names from operation names for the RTL netlist. Instance names have the form [`<scope>:<cell_type>[:#<int>]`] where `<cell_type>` is the logical operation (acc, and, or, mux, reg and so on), and `<scope>` and `#<int>` are added if necessary. Examples:

```
mux#4
MAC:mux#2
SHIFT:if:and#11
```

Nets are named similarly, usually with a format

[`<scope>:<cell_type>[:#<int>].<var_type>[:#<int>]`], where `<var_type>` is one of the strings in Table 3-2 or Table 3-3. Examples:

```
mux#27.item
SHIFT:acc#1.tmp
i#1.sva#2
```

The netlist variables match the schematic net names except underscores ‘_’ replace illegal RTL characters. Example:

```
mux_27_item
SHIFT_acc_1_tmp
i_1_sva_2
```

Table 3-1. Definitions of Net Name Suffixes in Wait Controller

Control from Process	
Suffix	Description
OSWT	Output select
ISWT	Input select

Table 3-1. Definitions of Net Name Suffixes in Wait Controller

Input Control	
Suffix	Description
ISWT#	Input select (seq delay #)
DSWT#	Defer input select (seq delay #)
DCWT	Input control disable (no inpreg)
DSWT	Input control disable (with inpreg) (should be PDCWT)
PDSWT#	Preregister defer input select
ICWT	Input count
PICWT	Preregister input count
OGWT	Output get
Output Control	
Suffix	Description
BIWT	Buffer increment
BDWT	Buffer decrement
BCWT	Buffer count
PBCWT	Preregister buffer count
BAWT	Buffer available
BFWT	Buffer
MXWT	Buffer multiplexor
Distributed Pipelines	
Suffix	Description
WNEN	Enable signal from default wait controller
WPEN	Enable signal from pipe controller
WWEN	Stage ready to advance (STGWT & WNEN)
STGWT	Stage in first state
RTR	Ready to receive (should be LDPRV)
RTS	Ready to send (should be VDPRV)
PYWT	Pipe idle (negated)
VDPRVEFF	Effective RTS (VDPRV)
WTPRV	Next stage ready (with buffering)
WTNXT	Previous stage ready (with buffer)

Table 3-1. Definitions of Net Name Suffixes in Wait Controller

Decouple Stages	
Suffix	Description
DCCWT	Decouple buffer count (1 bit)
DCPL	Decouple buffer
PDCPL	Preregister decouple buffer

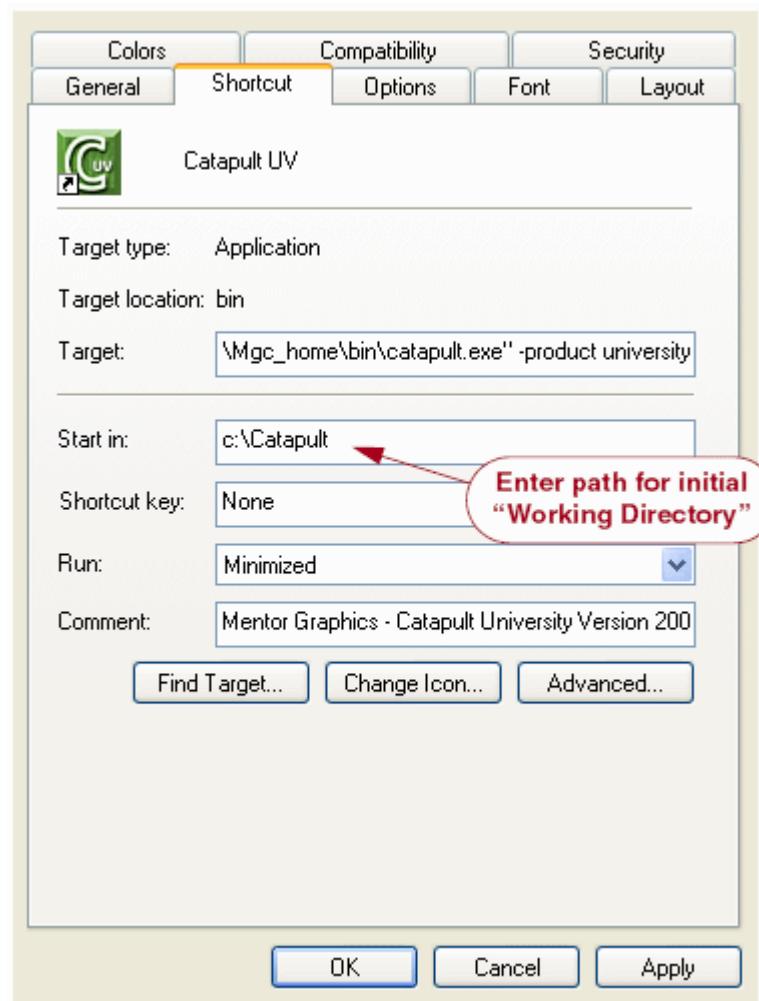
Invoking the Graphical User Interface

You can run Catapult C Synthesis from the GUI or from a shell command line. The shell interface requires Tcl command syntax and scripting techniques. The section “[General Command Syntax](#)” on page 349 provides information about how Tcl commands are used in the Catapult tool. Reference help files about the Tcl language are available in the Catapult software tree at “\$MGC_HOME/pkg/tcl_msg/man” (UNIX man pages) or “\$MGC_HOME\pkgs\tcl_msg\doc” (Windows help files).

The GUI is invoked with the “[catapult](#)” command from a Windows shell or a UNIX/Linux shell. By default, the invocation directory becomes the Catapult working directory. To change the working directory, see “[Setting the Working Directory](#)” on page 92.

You can also configure Catapult to start in a different working directory by default. Refer to “[General Options](#)” on page 175. In a Windows environment, you can optionally create a Shortcut on your Desktop and set the Shortcut Properties similar to that shown in Figure 3-7. The Catapult setup program can create an invocation icon on your desktop for you.

Figure 3-7. Setting Up the Catapult C Synthesis Shortcut



The Catapult Session Window

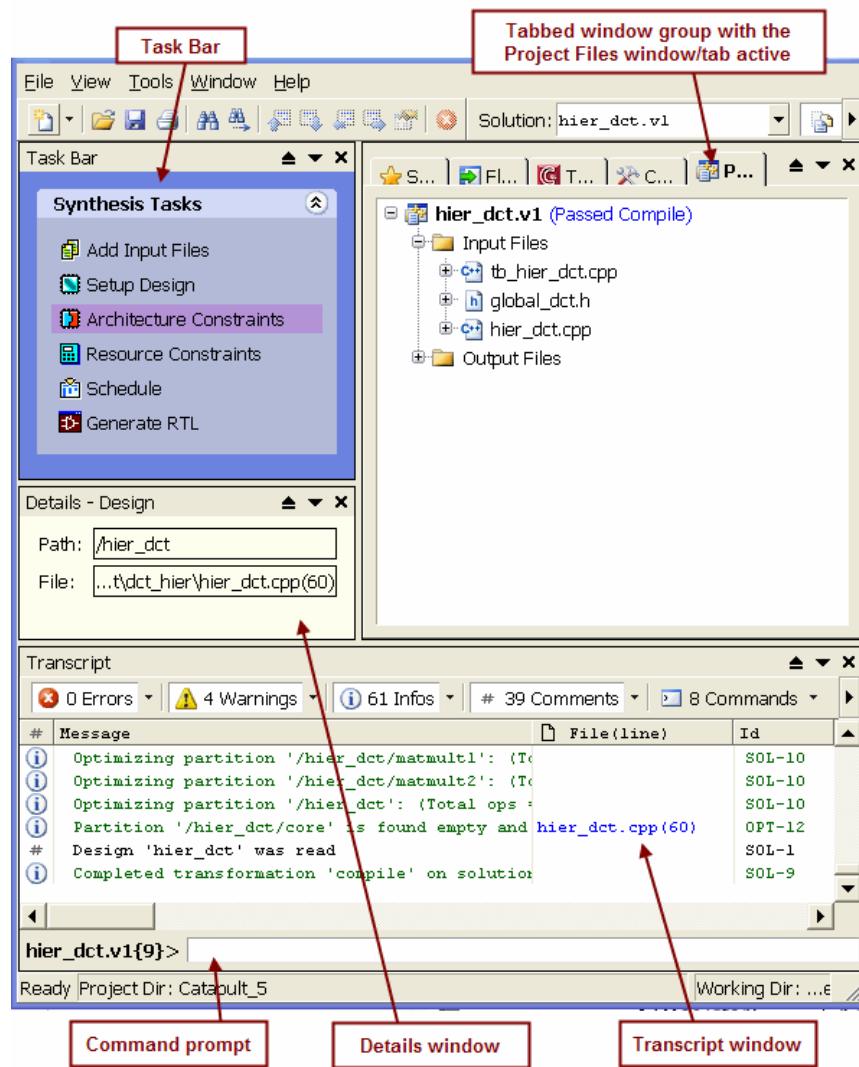
The default layout of the Catapult session window displays all of the project management windows, identified in Figure 3-8. The project management windows are your interface for entering commands, controlling the work flow, constraining the design, managing the project files and tracking area/timing results data. In addition, these windows also provide graphical views of the design throughout the work flow.

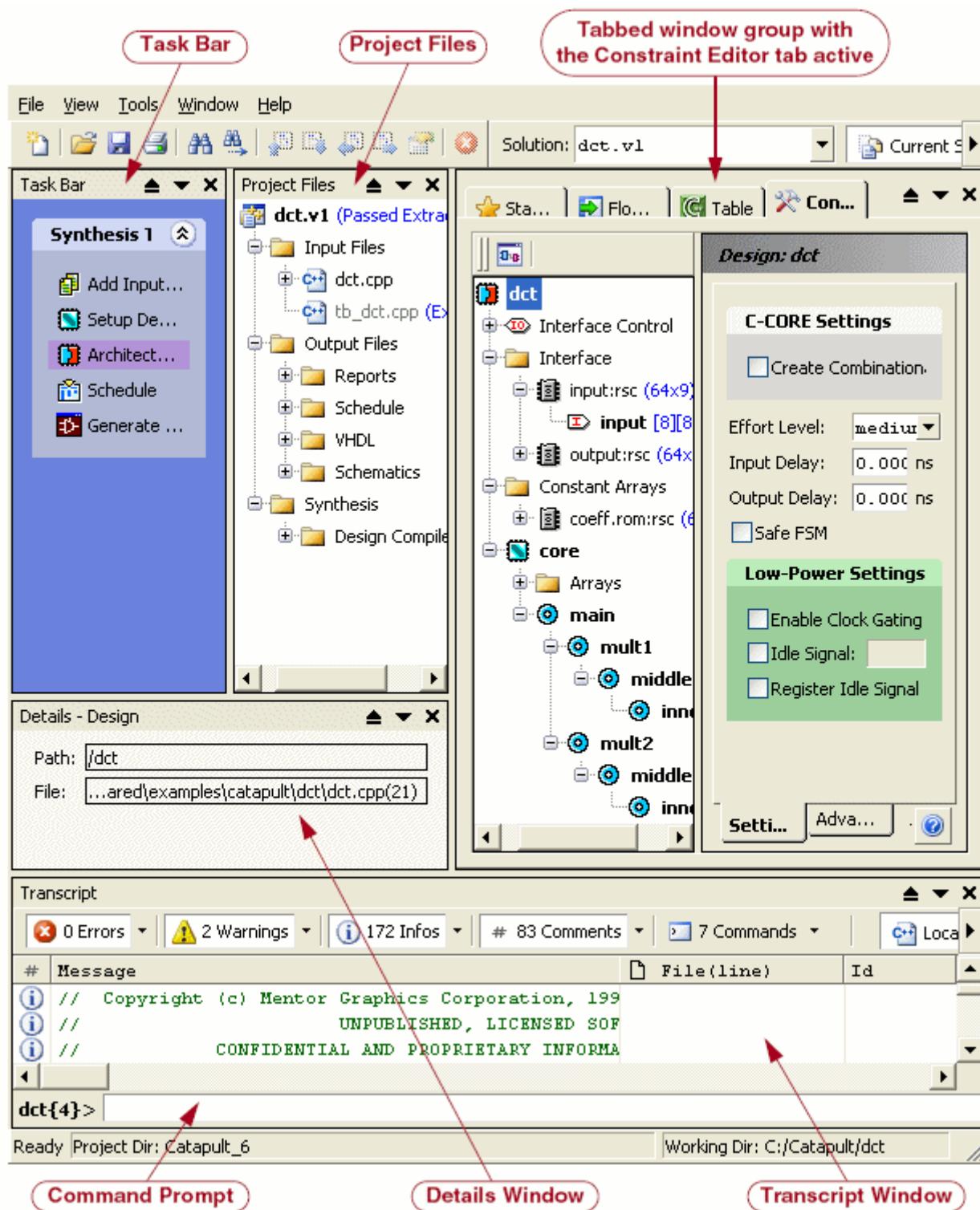
In the default configuration, the project management windows are visible at all times. Other windows, such as the Constraints Editor, the Schematic viewer and so on, that are opened as needed, are stacked in a tabbed window arrangement. Select a tab of a hidden window to bring it to the top of the stack.

At the top of the session window is a set of pulldown menus and a tool bar for quick access to commonly used commands. All window can be moved, resized, undocked from the session

window, or changed to/from a tabbed window. Refer to “[Changing the Window Layout](#)” on page 89 for more information.

Figure 3-8. Initial Catapult Window





The basic features of the tool bar and the project management windows are described in the following sections:

- “[The Start Page](#)” on page 64

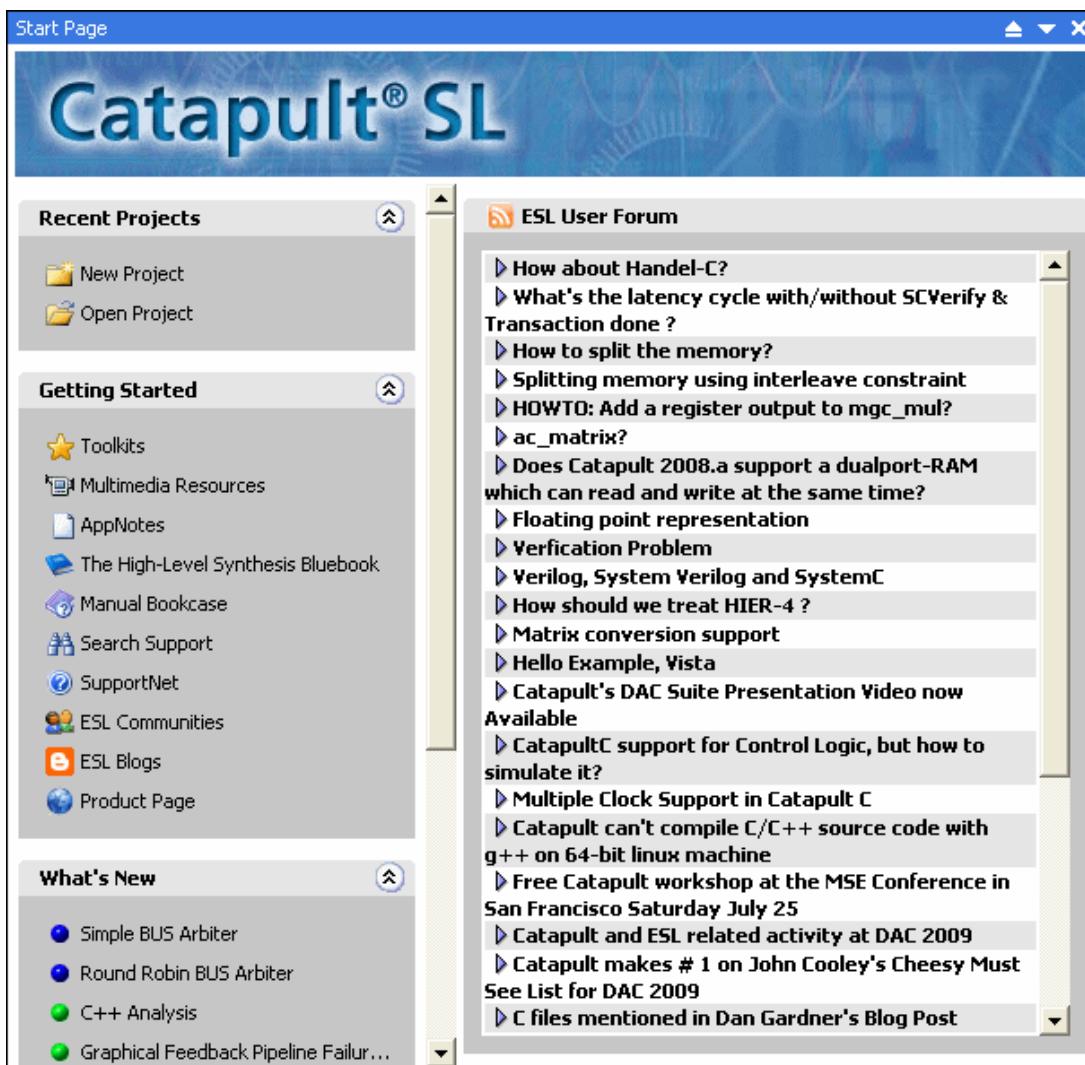
- “[The Tool Bar](#)” on page 65
- “[The Task Bar](#)” on page 67
- “[The Details Window](#)” on page 68
- “[The Constraint Editor Window](#)” on page 69
- “[The Project Files Window](#)” on page 70
- “[The Command Input and Transcript Window](#)” on page 72
- “[The Flow Manager Window](#)” on page 77
- “[The Table Window](#)” on page 79
- “[The Bar Chart and XY Plot Windows](#)” on page 80
- “[The Toolkits Window](#)” on page 82

The Start Page

The first window you see when the Catapult tool opens is the Start Page, shown in Figure 3-9. The Start Page gives you one click access to a variety of information about the Catapult tool as well as shortcut links to some useful commands. The links are grouped into four fields:

- **Recent Projects:** Commands for opening saved project files or creating a new project.
- **Getting Started:** Links to resource materials about how to use the Catapult tool, including product documentation and examples, as well as online resources like SupportNet, the ESL Communities and Blogs.
- **What’s New:** Links to toolkit examples that describe and demonstrate the new features available in the current release.
- **ESL User Forum:** Links to the current discussion threads in the forum. The list of topic is dynamically updated every time the window is opened.

Figure 3-9. Start Page Window



The Tool Bar

In addition to the standard buttons for creating, opening, saving, printing and searching files, the tool bar contains a number of specialized buttons described in this section.

New Project/Document Button

The “**New**” button, shown in Figure 3-10, is context sensitive and opens either a new project or a new text document. If a text document window is active, the button opens a new document window. Otherwise it issues the “[project new](#)” command. You are prompted to save the current project before it is closed.

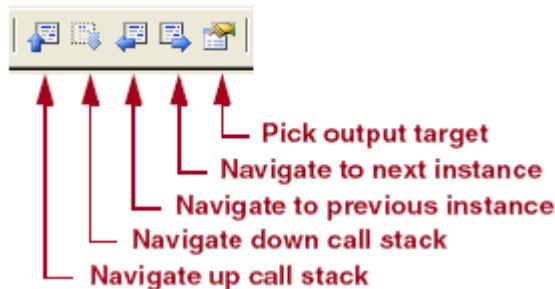
Figure 3-10. Tool Bar Button for Creating a New Project or Document



Cross-Probe Navigation Buttons

The cross-probe buttons shown in Figure 3-11 are used to trace design objects between source code and generated output files. Hovering the cursor over a button for a moment will display a description of the button. The buttons are dynamically enabled or disabled (greyed) based on the selected object. If the object has a cross-probe link, the appropriate button(s) are enabled. For a detailed discussion about how to use the cross-probing feature, refer to “[Cross-Probing Between Source Code and Generated Output](#)” on page 102.

Figure 3-11. Cross-Probe Buttons on the Tool Bar



Stop Button

The Stop button initiates an interrupt of whatever process Catapult is doing. The interrupted process will not stop immediately, but will wait for the next opportunity to safely terminate the process and restore the design database to its previous state. For example, if the Stop button is clicked while the scheduler is running, then the scheduler will stop its current optimization iteration, regenerate the last good schedule and then terminate.

Figure 3-12. Stop Button on the Tool Bar



Solution Selection and View Controls

The solution selector gadget, shown in Figure 3-13, provides a drop-down menu containing the names of all of the solutions in the project. Select a solution from the list to make it the active solution. The “**Current Solution**”/“**All Solutions**” button toggles the visibility of inactive solutions in the Project Files window and optionally in the transcript window.

When the “Allow Solution Filtering in the Transcript” option (See “[Message Options](#)” on page 176) is enabled, the “**Current Solution**”/“**All Solutions**” setting will toggle the visibility of messages in the transcript window. In “Current Solution” mode, messages generated by

solutions other than the current solution are hidden. Regardless of the display mode, all messages are captured in the session log file.

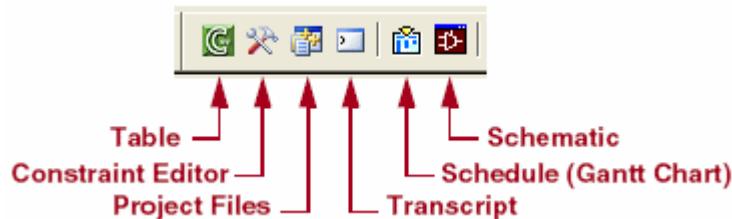
Figure 3-13. Solution Selection and View Controls on the Tool Bar



Window Activation Buttons

Each button in this group corresponds to one of the project management windows in the session. Clicking a window button will open the corresponding window and bring it to the front if it is hidden. Hovering the cursor over a button for a moment will display a description of the button. The same functionality is available from the “View” pulldown menu.

Figure 3-14. Window Activation Buttons on the Tool Bar



Help Button

Click the Help button to get information about the active window.

Figure 3-15. Context Help Button



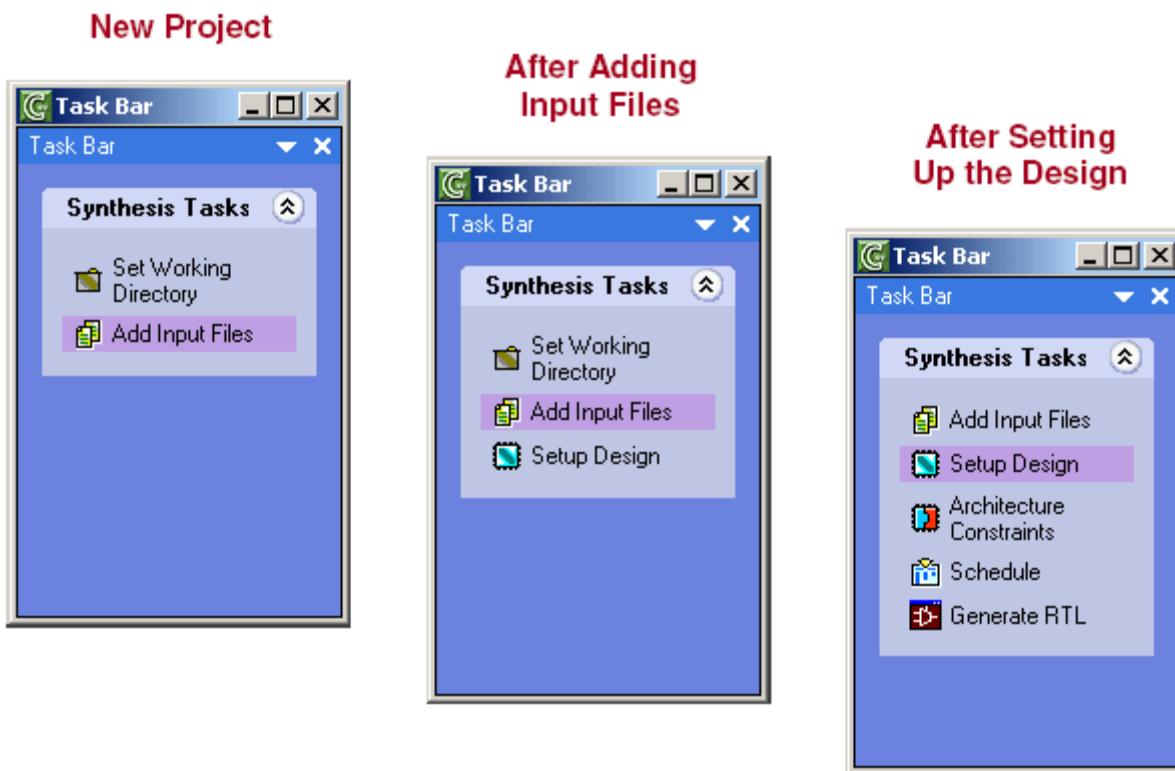
The Task Bar

The Task Bar controls the Catapult work flow. It contains a set of task icons, each one corresponding to a particular stage in the Catapult work flow. See “[The Catapult Work Flow](#)” on page 46 for a flow diagram of the work flow. Clicking on a task invokes the necessary Catapult command(s) to advance the design to the corresponding stage. The selected task on the Task Bar remains highlighted so that you will always know the stage of the process in which you are working.

Since Catapult stores the state of the design at each stage in the flow, clicking on a earlier task in the flow allows you to access to the design as it was in that stage. Any changes made to the design at an earlier stage will cause Catapult to branch a new solution. You can return to the old solution at any time by using the [Solution Selection and View Controls](#) (see page 66).

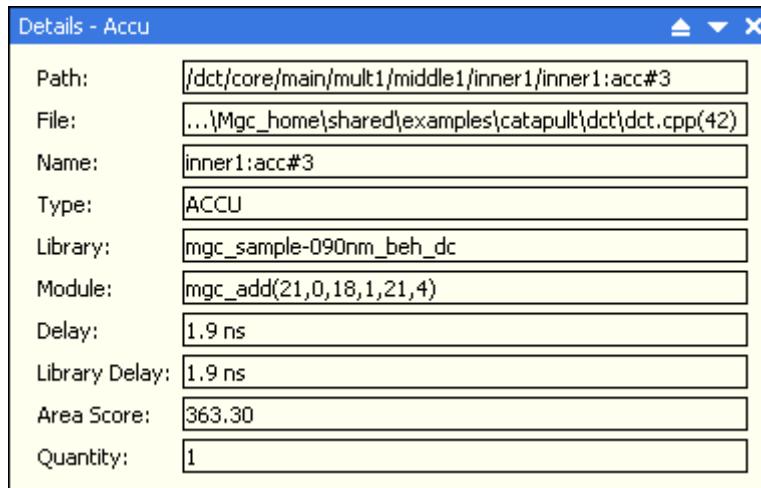
When starting a new project, the initial Task Bar contains only two tasks: **Set Working Directory** and **Add Input Files**. Other tasks appear on the Task Bar after these initial tasks are completed. Figure 3-16 shows the order in which the tasks are progressively disclosed. Note that the **Set Working Directory** task is removed from the Task Bar after the **Setup Design** task is complete. To reset the working directory after that point, use the **File > Set Working Directory...** pull-down menu item.

Figure 3-16. Progressive Disclosure of Tasks on the Task Bar



The Details Window

The Details window displays information about the currently selected design object in the **Constraint Editor**, **Project Files** window, **Schedule** (Gantt chart) window or **Schematic** window. The type of information provided depends on the type of object that is selected. The example in Figure 3-17 shows the type of data displayed for an accumulator operator selected in a Gantt chart. For information about the format of generated names, refer to "[Format of Catapult Generated Object Names](#)" on page 49.

Figure 3-17. Design Object Information Displayed in the Details Window

The Constraint Editor Window

The **Constraint Editor** window is the graphical interface for setting design constraints. It is context sensitive with respect to the current stage of the Catapult work flow. As you progress through the work flow by using the Task Bar, the Constraint Editor changes context accordingly and displays only the constraints that are appropriate for that stage. (The “**Generate RTL**” stage does not allow constraints to be edited, but provides a graphical view of the design.) For detailed information about each context of the Constraint Editor, refer to the following sections:

- “[Setting Up the Design](#)” on page 96
- “[Specifying Architectural Constraints](#)” on page 104
- “[Scheduling the Design](#)” on page 127
- “[Generating the Output Files](#)” on page 141

The Constraint Editor window is divided into two panes. In the left pane of the window is a graphical view of the design objects. Selecting a design object causes the constraints for that

object to be displayed/edited in the pane on the right side. Table 3-2 shows the sets of constraints that are accessible in each state of the Task Bar.

Table 3-2. Editable Constraints Corresponding to Task Bar State

Task Bar State	Selected Object	Applicable Constraints
Setup Design	Technology	Synthesis tool, target hardware technology, compatible libraries, design frequency, and global optimization options.
	Interface Control	Settings for the clock, reset, enable and handshake signals.
	Source code function	Designate the top-level function.
Architectural Constraints	Design or Process	Constraints and optimizations that are global to the design or selected process.
	Resource	Resource type, memory configuration and other optimizations depending on the type of resource.
	Resource Variable	Word width and, depending on the type of component the resource is mapped to, memory address mapping and/or data streaming.
	Loop	Iteration count, unrolling, pipelining and merging.
Schedule	Loop	View only. Displays a Gantt chart of the selected loop.
Generate RTL	Anything	View only. Displays a hierarchical representation of the RTL design.

The Project Files Window

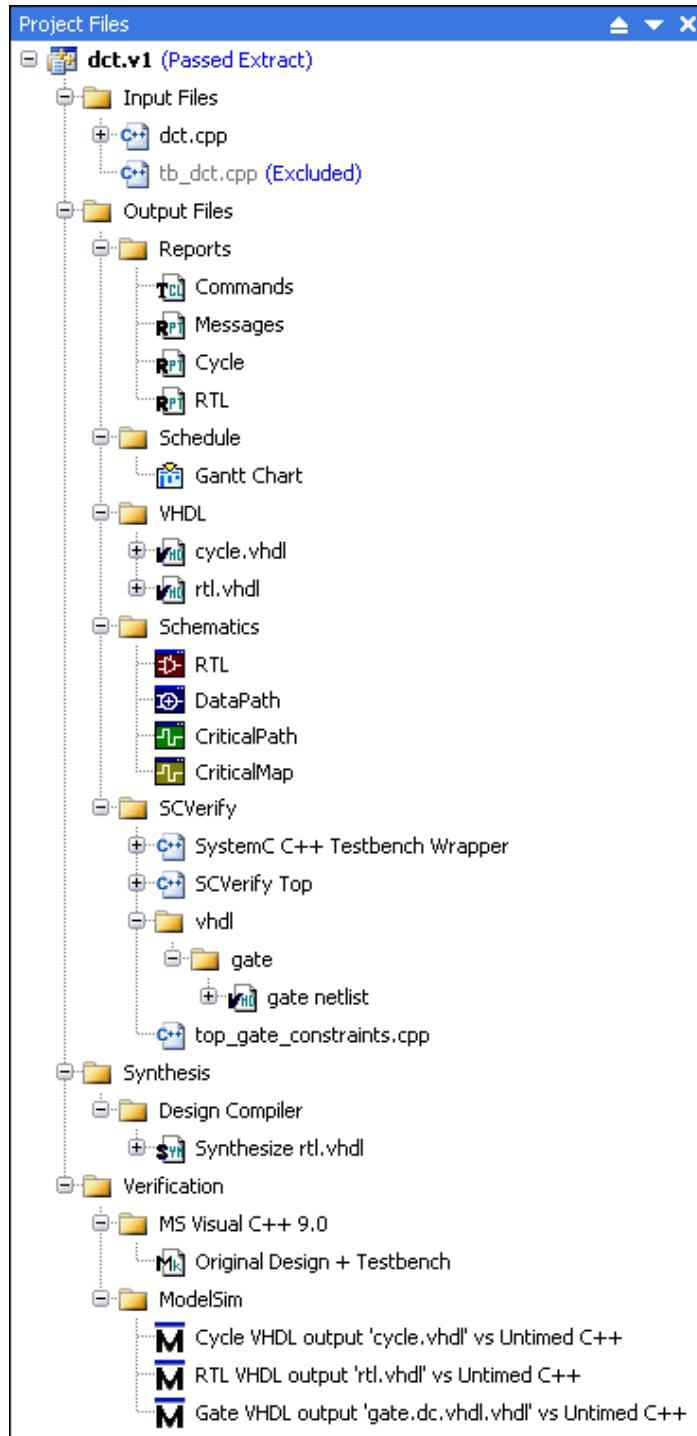
This window shows you all of the files in each solution in the project. The files are grouped by type into hierarchical folders, as shown in Figure 3-18. Folders are created as needed when files are added to a solution.

Note  The folder hierarchy displayed in the Project Files window does not correspond to the actual file system structure in the solution directory. To get the actual file system path, select the file in the Project Files window and look at the “File” path in the Details window.

Selecting a file or any of its dependents will display information about the item in the Details window. Right-clicking on a file will open a popup menu for the file. The file type determines the set of commands that appear on the popup menu.

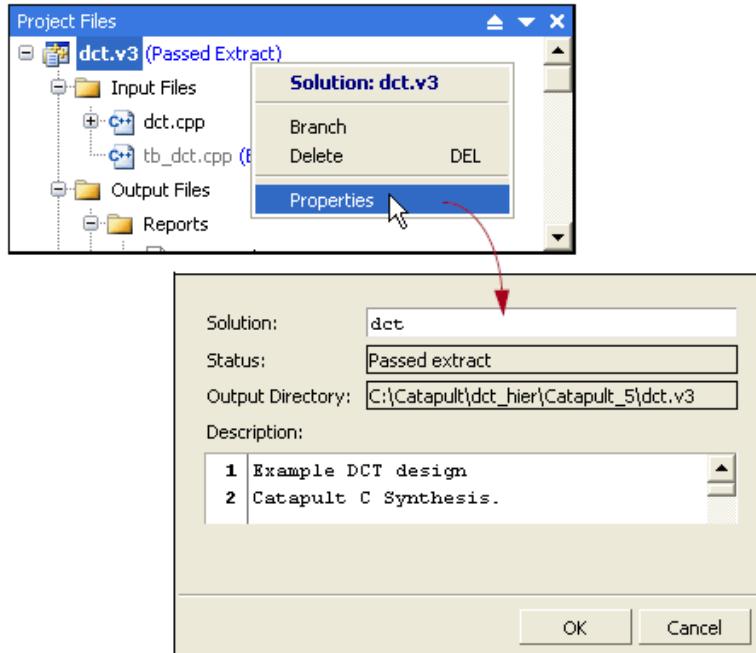
By default, only the contents of the current solution is displayed. To see all solutions, click on “**Current Solution**” button on the tool bar at the top of the session window. The button name changes to “**All Solutions**” in that mode. To toggle back, click on “**All Solutions**”. When all solutions are visible, double-clicking on a solution makes it the active solution.

Figure 3-18. Project Files Window



Right-click on a solution name in the Project Files window to display a popup menu of commands. Use the branch command to force the creation of a new solution. Use the Properties command to modify the solution name and/or description text. Refer to Figure 3-19.

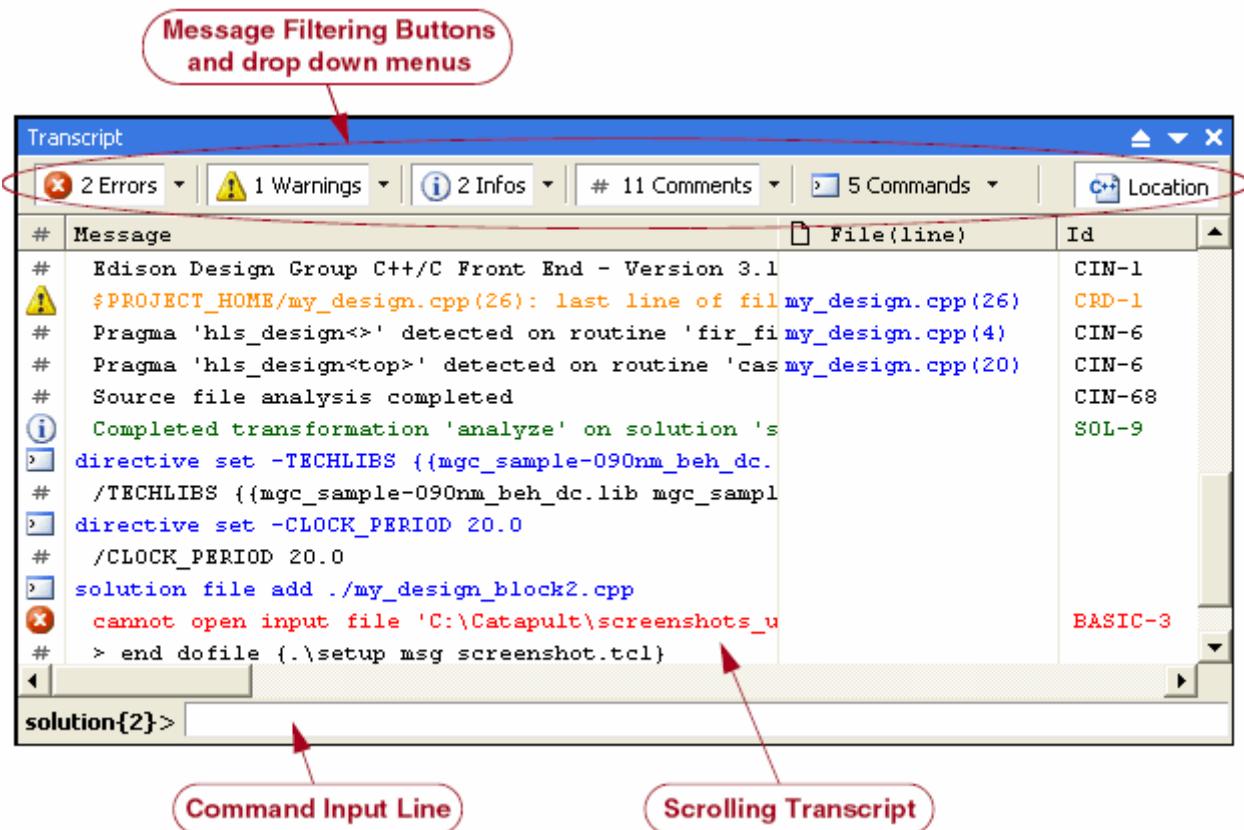
Figure 3-19. Editing the Solution Properties



The Command Input and Transcript Window

This window, shown in Figure 3-20, accepts Tcl command input and displays a scrolling transcript of all activity performed during the Catapult session. The message filtering buttons and drop down menus allow you to change the visibility of messages and search for messages by their severity classification.

Figure 3-20. Command Input and Transcript Window



Command Input Line

The command input line is a Tcl command interpreter that can access the host operating system shell commands as well as Tcl commands for Catapult. (For information about all Catapult Tcl commands and the Tcl interface, refer to “[Commands](#)” on page 349.) As commands are entered, a transcript of the commands and any associated system messages display in the transcript area of the window.

The label next to the command input area shows the name of the active solution and a count of the number of commands entered. The command input area provides context sensitive command completion. To use command completion, place the cursor in the command input area, begin typing a command string, then press the TAB key to automatically complete the *word* under the cursor, whether the word is a command name, option, switch, directive or path argument. All text to the left of the cursor must match the beginning portion of a valid command string in order for Catapult to supply a result. If the partial string is viable, one of the following outcomes will occur:

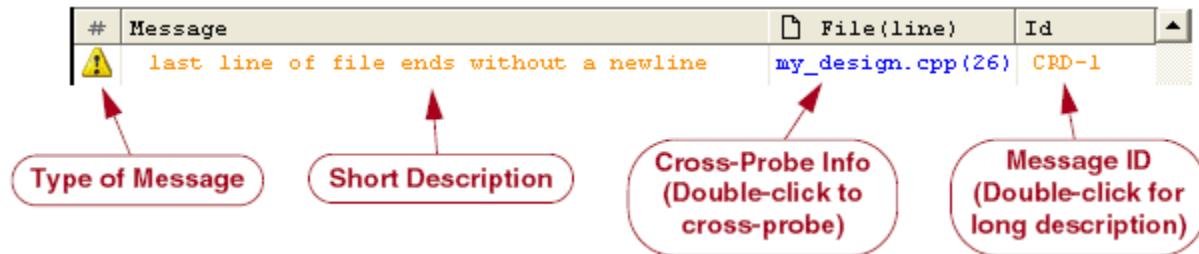
- If a unique match for the partial word under the cursor can be found, that string replaces the partial word.

- If multiple matches are found, a list of all possible matches is displayed in the transcript. Scan the list to find the string you want and type it or copy it to the command line.
- If no match is possible, no change is made to the command line.

Transcript Area

Transcript messages are color coded by type. Error messages are red, warning messages are orange, informational messages are green, commands are blue and comments are black. Each message appears on a single line and has four parts, as shown in [Figure 3-21](#). First is an icon that indicates the message type. Second is the message text in a concise form. Third is cross-probe information consisting of the name and line number of the source code file related to the message. Forth is the message identifier. For a detailed discussion about message identifiers and how to configure their severity levels, refer to “[Understanding Messages in the Transcript](#)” on page 159.

Figure 3-21. Transcript Window Messages



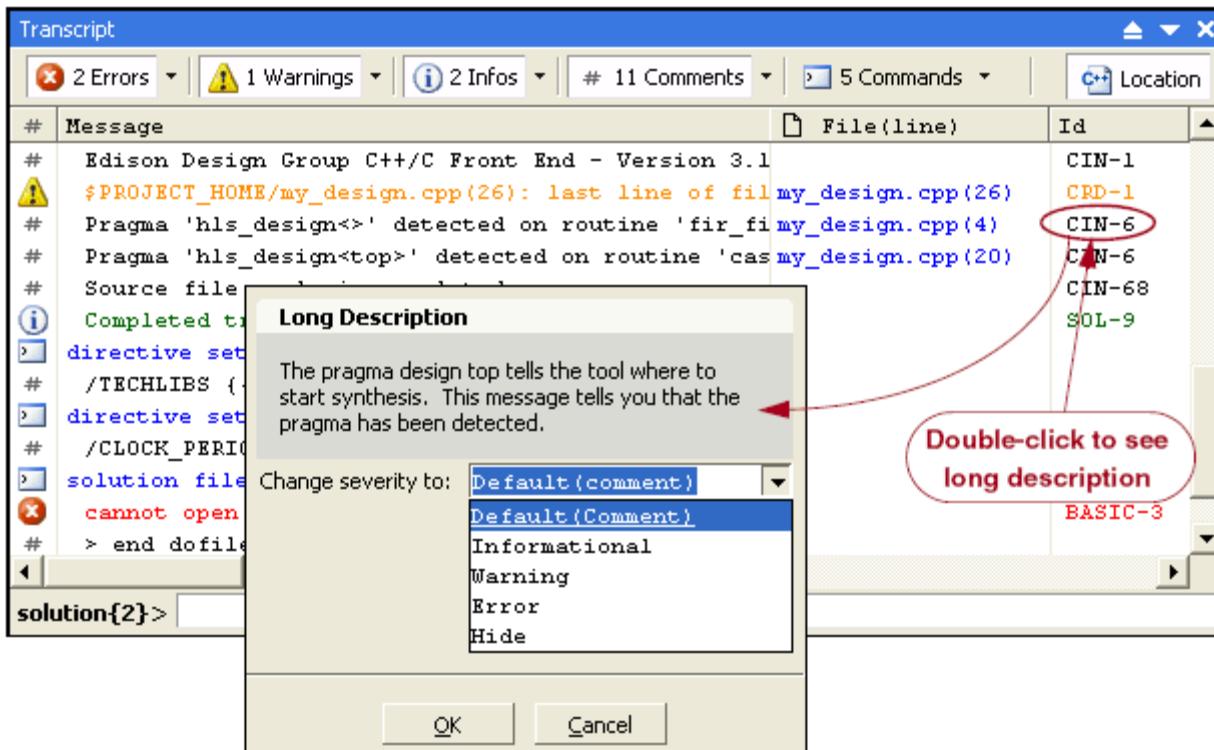
Viewing the Long Description

Message descriptions in the transcript are short and concise. Additional information is available for some messages by double-clicking on the message ID at the end of the line. As shown in Figure 3-22, that action opens the “Long Description” window. In addition to displaying the long description, the window also allows you to change the severity level of that message ID.

You can also use the “[help message](#)” command to display the long description of a message in the transcript window. For example, the following command will display the long description of the “CIN-6” comment:

```
help message CIN-6
# The pragma design top tells the tool where to start synthesis. This
# message tells you that the pragma has been detected.
```

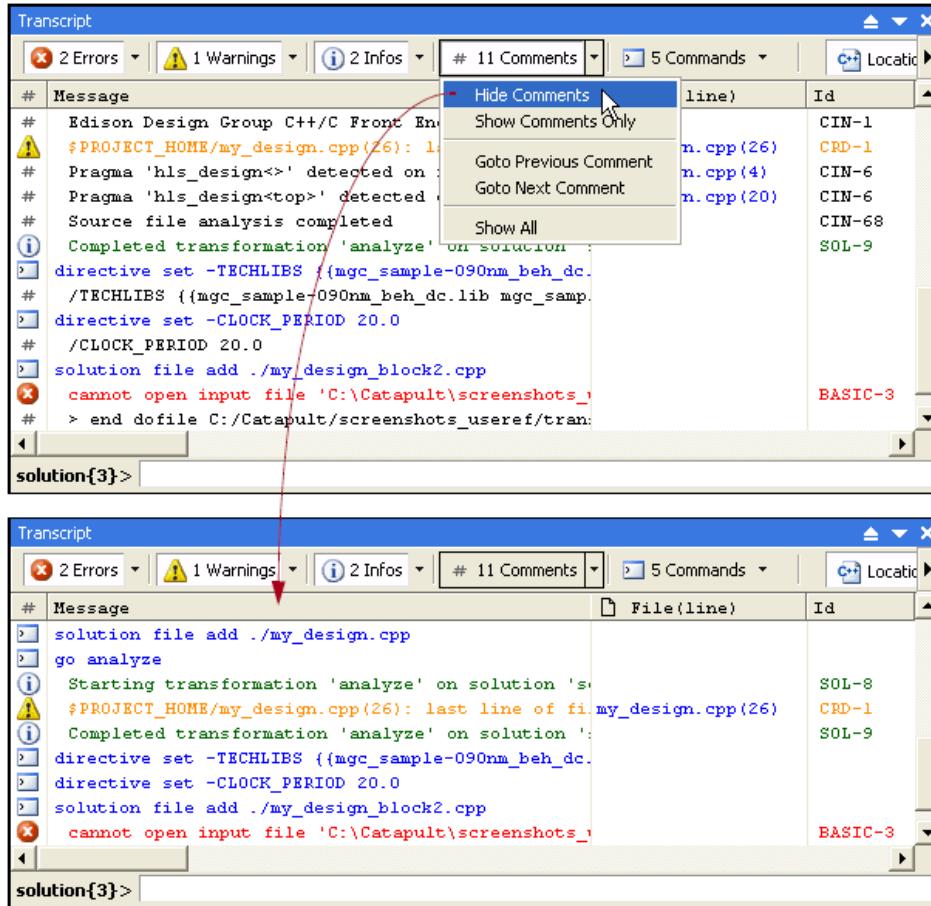
Figure 3-22. Viewing the Long Description of a Message



Filtering Buttons and Drop Down Menus

The message filtering buttons at the top of the window allow you to toggle the visibility of each message type. Each button also has a drop down menu that provides addition filtering and search options.

Figure 3-23. Filtering Comments Out of the Transcript (before and after)



The menu items are as follows (<msg_type> is either Errors, Warnings, Infos or Comments):

- Hide <msg_type> / Show <msg_type>
- Show <msg_type> Only
- Goto Previous <msg_type>
- Goto Next <msg_type>
- Show All
(Makes all messages visible regardless of <msg_type>)

The “Commands” button drop-down menu has the following items:

- Show Only Commands / Show All
- Show Hierarchy / Hide Hierarchy
- Goto Previous Command

- Goto Next Command

In “Show Hierarchy” display mode, all messages generated by a command are subordinate to that command. Each command can be individually expanded or collapsed to show or hide its subordinate messages.

The “Location” button toggles the format of the cross-probe column to display a file icon instead of the file name. The icon takes up less space on the line and allows more of the message description to be seen.

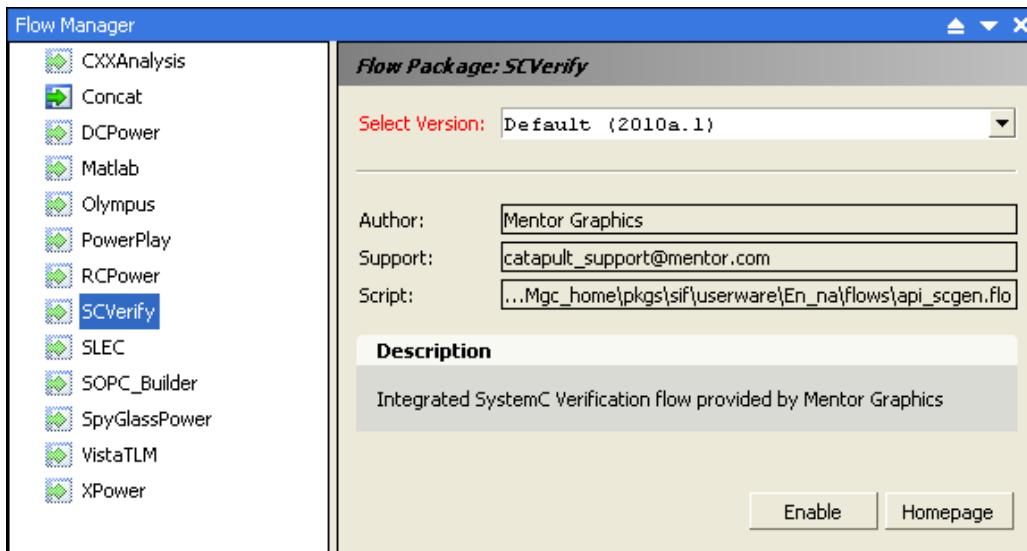
By default, only messages generated by the current solution are displayed. However, messages from all other solutions can be displayed by toggling the “**Current Solution**”/“**All Solutions**” button on the session window tool bar.

The Flow Manager Window

The Flow Manager window, shown in Figure 3-24, is one of the tabbed document windows. It is the interface for enabling/disabling optional flow packages and configuring their settings. Some flows are available in a basic Catapult installation, such as the SystemC Verification flow (SCVerify). Other flows, such as SpyGlassPower, must be licensed separately. If a valid license exists, the optional flow will appear in the window. In addition, custom flow packages can also be accessed from the Flow Manager. For more information about creating custom flow packages, refer to “[Flow Customization](#)” on page 609.

When a flow is selected in the pane on the left, details about the flow will appear in pane on the right. (Note that the same detail information is also displayed in the Details window anytime the flow is selected.) The drop-down menu in the “Version” field lists the compatible flow package versions that are available. Choosing “**Default**” will load the latest package version. The “**Homepage**” button is a link to a web page about the flow tool. It opens a web browser in a separate window.

Figure 3-24. Flow Manager Window



Enabling and Running a Flow

Enabling a flow is a two step process. The first step activates the flow and the second step allows you to set optional flow properties. Begin by selecting a flow and clicking the “**Enable**” button. When the flow is enabled it immediately runs (“[flow run](#)” command) using its default flow options. When it is finished, the flow options are displayed in the flow window. If you modify the option settings, click the “**Apply**” button to re-run the flow.

You can modify the flow options at anytime after the flow has been enabled by selecting the flow in the Flow Manager window. The changes are saved in the current Solution and will be inherited by Solutions branched from it. The default settings are not affected.

Use the Catapult Options window to override the default property settings for a flow package. Open the window by choosing the “**Tools > Set Options...**” pulldown menu item. Then expand the **Flows** section and select the flow package you wish to modify. Only active flows are listed, so you must have already enabled the flow at this point. When the flow is selected, its properties are displayed on the right side of the window. Applying your changes preserves them as part of the current project. To save changes for future Catapult sessions, choose the “**Tools > Save Options**” or “**Tools > Save Options As...**” pulldown menu item. Refer to “[Setting Up Catapult Default Options](#)” on page 173.

Disabling a Flow

Select an active flow in the Flow Manager window and click the “**Disable**” button.

The Table Window

This window displays a variety of summary report tables covering the latency, area, throughput, run time, memory usage and power estimates for each solution in the project. Report tables are added to the window, and existing reports are updated, as new data becomes available. As new solutions are created, they are automatically added to the window. The “Bar Chart” and “XY Plot” buttons on the tool bar to dynamically generate charts and graphs of the table data. Refer to “[The Bar Chart and XY Plot Windows](#)” on page 80 for more information.

Figure 3-25. Quality of Results Summary Table

The screenshot shows the 'Table' window with the following data:

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
dct.v1 (extract)	3783	18915.00	3793	18965.00	195492.58	-0.24
dct.v2 (extract)	4090	81800.00	4092	81840.00	189611.21	0.45
dct.v3 (extract)	8182	40910.00	8184	40920.00	213019.17	-0.24

Double-clicking a solution row in the table makes it the active solution. Left-click on a column heading to sort the table by the values in that column. Each click toggles the sort order between ascending and descending. Select and drag a column heading to move the column.

The initial view displays the “General” report. To view other reports, choose one from the drop down menu in the “Report” field. Power analysis reports are also available in the Table window. They are added to the Reports drop down list after the power tool flow has run and the data has been collected.

The “Timing” and “Area Score” report tables are hierarchical in format. A triangle icon appears to the left of each solution name in the window. Click on the triangle icons to expand or collapse the hierarchy as shown in [Figure 3-26](#).

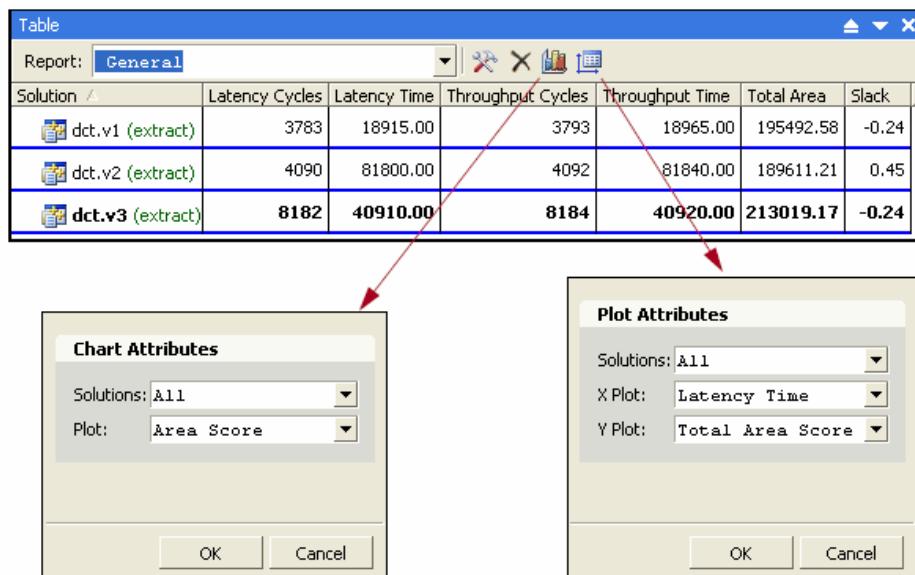
Figure 3-26. Timing Report Table

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Iterations	Slack
dct.v1 (extract)	3783	18915.00	3793	18965.00		
dct						
core	3783	18915.00	3793	18965.00		
main	3783	18915.00	3793	18965.00	Infinite	
mult1			1608	8040.00	8	
middle1			1600	8000.00	8	
inner1			1536	7680.00	8	
mult2			2184	10920.00	8	
middle2			2176	10880.00	8	
dct.v2 (extract)	4090	81800.00	4092	81840.00		
dct.v3 (extract)	8182	40910.00	8184	40920.00		

The Bar Chart and XY Plot Windows

You can display the data from the Table window in a bar chart or xy-plot format as shown in [Figure 3-27](#). Use the drop-down menus to select a data set to be graphed. The “Solutions” field allows you to select a single solution to graph, or all solutions (default). Click **Ok** to display the chart/plot.

Figure 3-27. Opening a Bar Chart or XY Plot Window

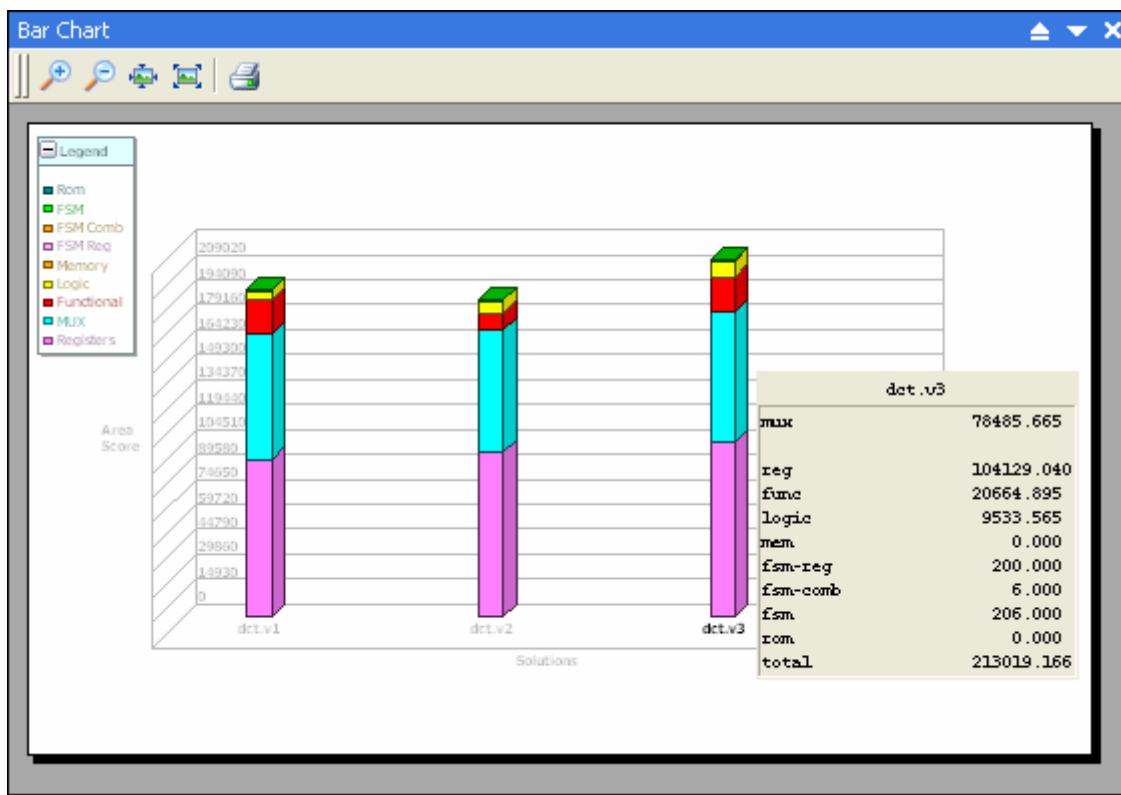


Bar Charts

Four different data sets can be graphed: Area Score, Timing, Memory Usage and Run Time. Each bar in the chart shows the proportional distribution of data elements for one solution. Expand the chart legend to see the names and color codes of each data element in the chart. As shown in Figure 3-28, hovering the cursor over a bar displays a table of the values depicted in the bar graph.

To zoom the view, use the tool bar buttons or mouse strokes. This window uses the same zoom mouse strokes as the Schematic window. Refer to “[Zoom and Page Navigation Controls](#)” on page 154 for a description of the zoom mouse strokes.

Figure 3-28. Bar Chart of Area Score

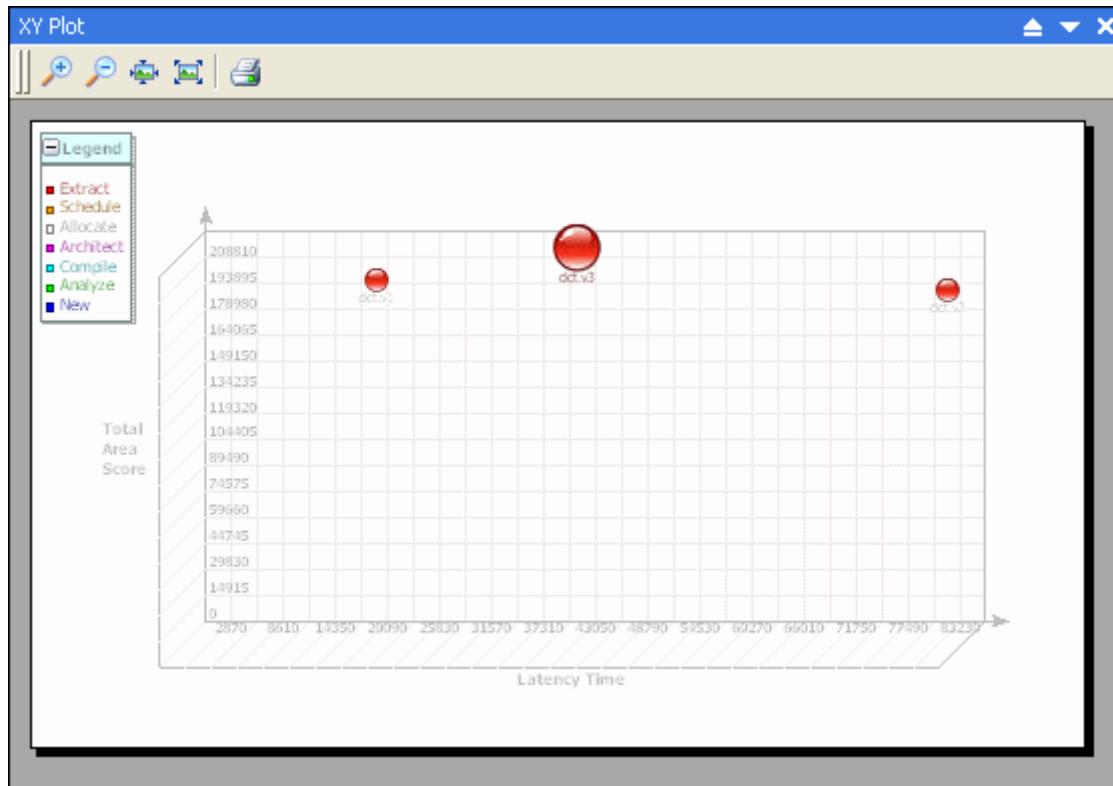


XY Plot

The XY Plot can display any one of several different graphs of the design area (Y axis) versus either latency or throughput (X axis). Each plot point on the graph is a solution and the active solution is the large point. The color of the points indicates the state of the solution in terms of the Catapult work flow. To see the color codes, expand the graph legend. Refer to Figure 3-29.

To zoom the view, use the tool bar buttons or mouse strokes. This window uses the same zoom mouse strokes as the Schematic window. Refer to “[Zoom and Page Navigation Controls](#)” on page 154 for a description of the zoom mouse strokes.

Figure 3-29. XY Plot of Area Versus Time/Throughput



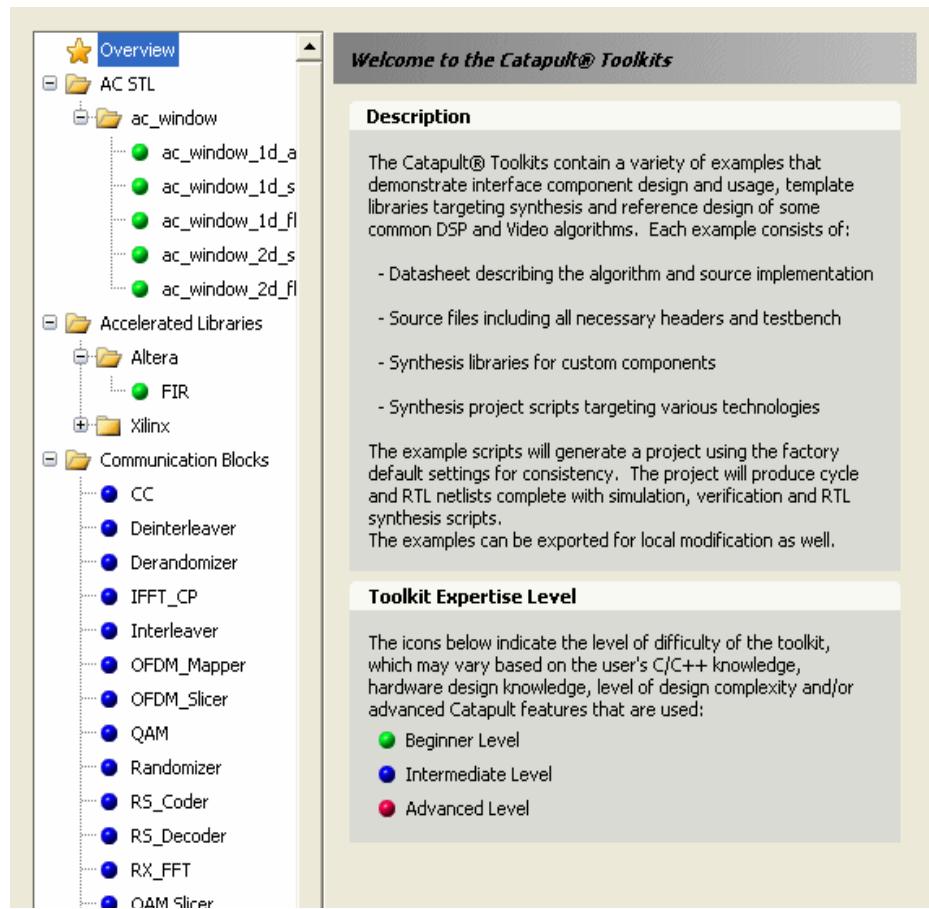
The Toolkits Window

The Catapult Toolkits provide a variety of examples that demonstrate many features of the Catapult products, such as interface component design and usage, template libraries targeting synthesis and reference designs of some common DSP and video algorithms. To access the Toolkits window, shown in Figure 3-30, select the **Help > Toolkits...** pulldown menu item in the Catapult session window.

Note  The “Reference Design Toolkits” option must be enabled in the Catapult Setup Wizard in order to make the toolkits available in the product. Refer to “[Using the Catapult Setup Wizard](#)” in *Catapult C Synthesis Installation Guide* for information about how to install the Toolkits option.

The left side of the window lists the available toolkits grouped into folders. Expand the folders to access the individual toolkits. The colored icons indicate the level of difficulty of the toolkit. Green is beginner level, blue is intermediate and red is advanced. The actual level of difficulty will vary based on your C/C++ and hardware design knowledge, as well as the level of design complexity and/or advanced Catapult features used in each toolkit. Note: Advanced toolkits are only available in the Catapult SL product.

Figure 3-30. The Toolkits Window

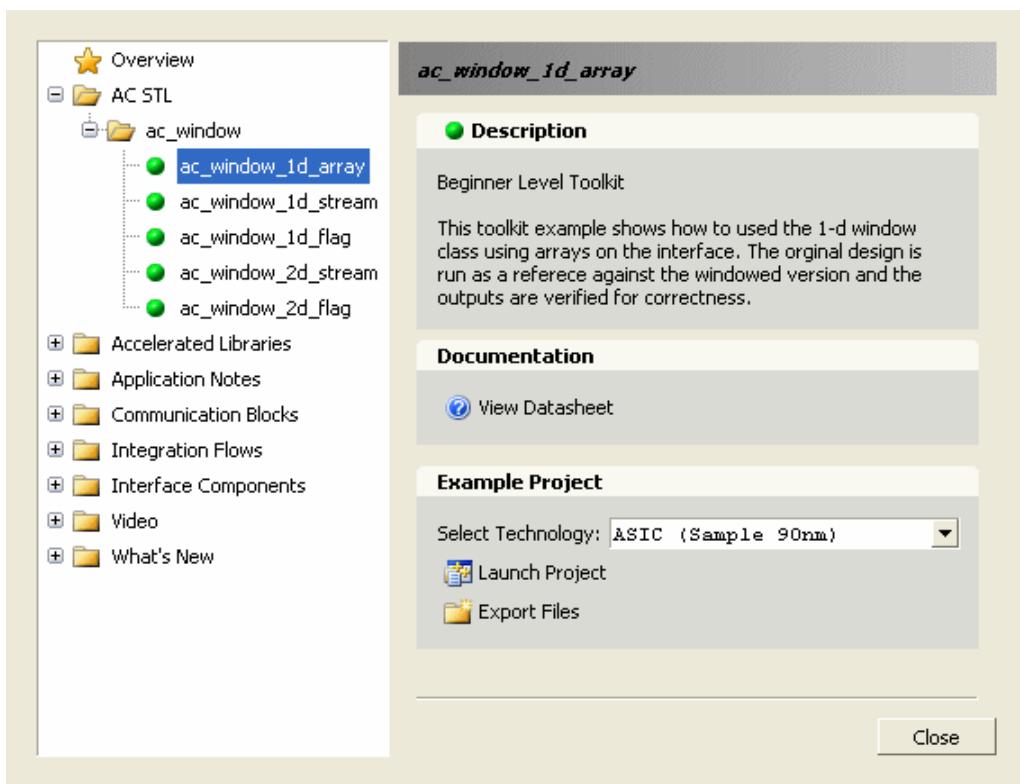


When a toolkit is selected, as shown in Figure 3-31, the Toolkits window gives you access to all of the files for that example, including documentation, source code and Tcl scripts to run the design in Catapult. In the “**Example Project**” group, select a target technology and then click the “**Launch Project**” to run the design through the Catapult work flow to the “Generate RTL” state. The “**Export Files**” feature allows you to save local copies of all the example files. The exported directory includes a “ReadMe.txt” file that lists the toolkit version number (if defined) and all of the files contained in the toolkit.

Each toolkit example consists of:

- Datasheet describing the algorithm and source implementation.
- Design source files including all necessary headers and testbench.
- Synthesis libraries for custom components.
- Synthesis project scripts targeting various technologies. These scripts will generate a Catapult project using the factory default settings. The project will produce cycle and RTL netlists complete with simulation, verification and RTL synthesis scripts.

Figure 3-31. Toolkit Interface



Adding Custom Toolkits to the Toolkits Window

Catapult provides an XML language interface for adding custom toolkits to the Toolkits window. A complete toolkit consists of a set of design files, Tcl scripts to run the design in Catapult, documentation files about the toolkit, and an XML file that defines the user interface and optional version number for the toolkit. A detailed description of the supported XML schema is defined in [“The XML schema for the Catapult Toolkits”](#) on page 87.

An example toolkit that demonstrates all of the various features and flexibility of the XML interface is installed in the Catapult software tree at:

```
<install_dir>/shared/examples/catapult/appnotes/AN_UserToolkit
```

The XML code for the example toolkit is shown in Example 3-1. It illustrates how a hierarchy of folders containing multiple projects and example designs can be coded. The resulting hierarchy is shown in Figure 3-32.

To load the example toolkit into the Toolkits window, do the following:

1. Select the "Tools > Set Options..." menu item to open the "Catapult Options" window.

2. Select "Toolkits" in the list on the left side of the window to display the Toolkits options.
3. In the "Configuration Files" field, click the Add button.
4. Enter the following path or use the file system navigation tool (icon to the right side of the field) to locate and select the following file:

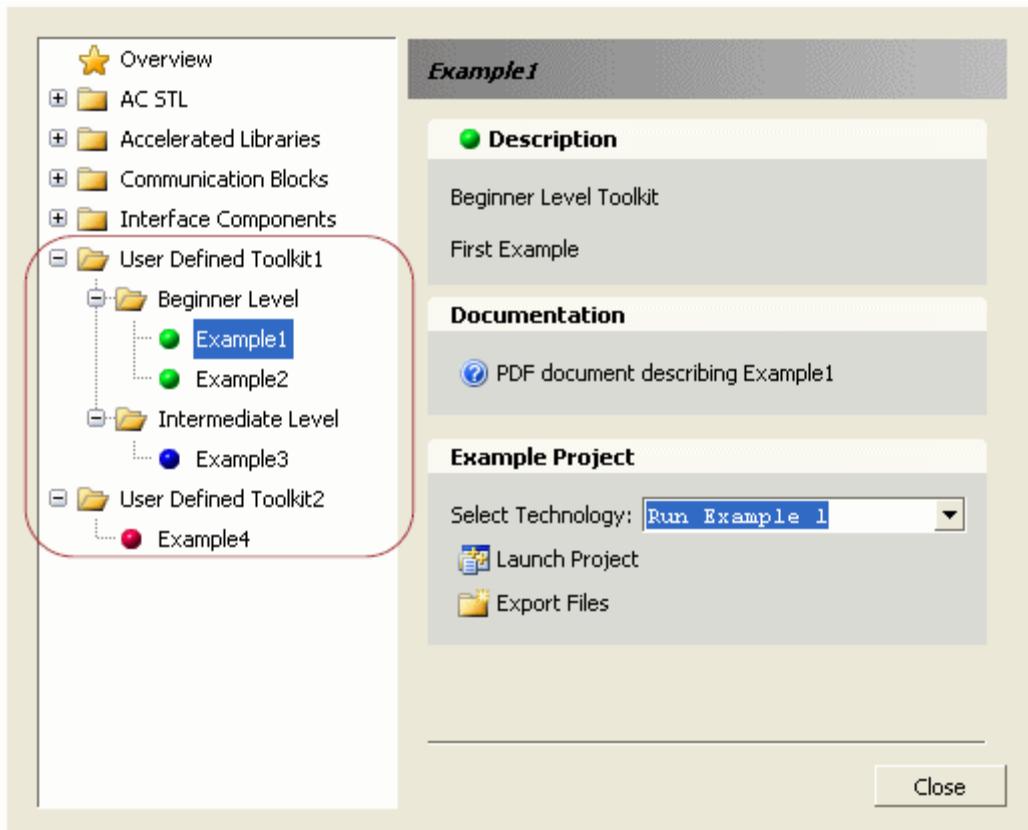
```
<install_dir>/shared/examples/catapult/appnotes/AN_UserToolkit/toolkit.xml
```
5. Click OK on all of the windows.
6. Select the "Help > Toolkits..." menu item to open the Toolkits window.

The example toolkits, named "User Defined Toolkit1" and "User Defined Toolkit2" appear in the window along with the standard toolkits. These examples are fully operational, although they do not contain actual designs. Clicking the "Launch Project" button runs a Tcl script in Catapult that simply prints a message to the transcript.

Note

 When loading XML files, the Toolkits window does not check for the existence or files and folders specified in the file.

Figure 3-32. Custom Toolkits Added to Toolkits Window



Example 3-1. XML Code to Add Custom Toolkits

```
<toolkits xmlns="http://www.mentor.com/catapult/toolkit/1.0">

    <folder>
        <version>
            <release>Version 1.12</release>
            <date>March 2008</date>
            <type>User Defined Toolkit1 Example</type>
        </version>
        <name>User Defined Toolkit1</name>
        <description>Beginner and intermediate level examples</description>

        <folder>
            <path>Beginner</path>
            <name>Beginner Level</name>
            <description>Skill Level: Beginner</description>
            <project>
                <name>Example1</name>
                <path>Example1</path>
                <description>First Example</description>
                <expertlevel>beginner</expertlevel>
                <document>
                    <description>PDF document describing Example1</description>
                    <file>Example1.pdf</file>
                </document>
                <script>
                    <description>Run Example 1</description>
                    <file>example1.tcl</file>
                </script>
                <file>example1.cpp</file>
            </project>
            <project>
                <name>Example2</name>
                <path>Example2</path>
                <description>Second Example</description>
                <expertlevel>beginner</expertlevel>
                <document>
                    <description>PDF document describing Example2</description>
                    <file>Example2.pdf</file>
                </document>
                <script>
                    <description>Run Example 2</description>
                    <file>example2.tcl</file>
                </script>
                <file>example2.cpp</file>
            </project>
        </folder>

        <folder>
            <path>Intermediate</path>
            <name>Intermediate Level</name>
            <description>Skill Level: Intermediate</description>
            <project>
                <name>Example3</name>
                <path>Example3</path>
                <description>Third Example</description>
            </project>
        </folder>
    </folder>
</toolkits>
```

```

<expertlevel>intermediate</expertlevel>
<document>
    <description>PDF document describing Example3</description>
    <file>Example3.pdf</file>
</document>
<script>
    <description>Run Example 3</description>
    <file>example3.tcl</file>
</script>
    <file>example3.cpp</file>
</project>
</folder>
</folder>

<folder>
    <version>
        <release>Version 1.12</release>
        <date>March 2008</date>
        <type>User Defined Toolkit2 Example</type>
    </version>
    <name>User Defined Toolkit2</name>
    <description>Advanced level examples</description>
    <project>
        <path>Example4</path>
        <name>Example4</name>
        <description>Fourth Example</description>
        <expertlevel>advanced</expertlevel>
            <document>
                <description>PDF document describing Example4a</description>
                <file>Example4a.pdf</file>
            </document>
            <document>
                <description>PDF document describing Example4b</description>
                <file>Example4b.pdf</file>
            </document>
            <script>
                <description>Run Example 4a</description>
                <file>example4a.tcl</file>
            </script>
            <script>
                <description>Run Example 4b</description>
                <file>example4b.tcl</file>
            </script>
            <file>source/example4a.cpp</file>
            <file>source/example4b.cpp</file>
        </project>
    </folder>
</toolkits>

```

The XML schema for the Catapult Toolkits

URI: <http://www.mentor.com/catapult/toolkit/1.0>

```

Root element: <toolkits>
  Child element: <folder> **
    Child element: <folder>

```

```
Child element: <path> <value> *
Child element: <name> <value> *
Child element: <description> <value> *
Child element: <version> *
    Child element: <release> *
    Child element: <date> *
    Child element: <type> *
Child element: <project> **
    Child element: <name> <value> *
    Child element: <path> <value> *
    Child element: <expertlevel> <value> *
    Child element: <description> <value> *
    Child element: <version> *
        Child element: <release> *
        Child element: <date> *
        Child element: <type> *
    Child element: <document> **
        Child element: <name> <value> *
        Child element: <description> <value> *
        Child element: <file> <value> *
    Child element: <script> **
        Child element: <name> <value> *
        Child element: <description> <value> *
        Child element: <file> <value> *
    Child element: <file> <value> **
```

KEY: ** -- Element can be specified multiple times,
 * -- Last element value is used

<folder>	Defines the hierarchy tree displayed in the left hand side of the Toolkits dialog. **
<folder>	This element may be repeated to define several levels of hierarchy **
<path>	Specifies the file system location for all child projects. The initial path is relative to the source XML file location. An absolute path may be specified. *
<name>	Defines the text displayed as the node in the hierarchy tree *
<description>	Shown in the right hand side of the dialog when the node is clicked *
<version>	Defines a version label for this folder and all child objects it contains. *
<release>	Arbitrary string that specifies a release label for the version. *
<date>	A string that specifies the version date. *
<type>	Arbitrary string that specifies a version type. *
<project>	Defines a group of files, documents, and scripts. **
<name>	Defines the text displayed as the node in the hierarchy tree *
<description>	Shown in the right hand side of the dialog when the node is clicked *
<path>	Specifies the file system location relative (or absolute) to the folder path *
<version>	Defines a version label for this project. This version overrides the parent folder version, if defined. *

```

<release>      Arbitrary string that specifies a release label
                  for the version. *
<date>          A string that specifies the version date. *
<type>          Arbitrary string that specifies a version type. *
<expertlevel>  Displays a color coded icon next to the project name
                  in the left hand side of Toolkit dialog. Valid
                  values: "beginner" (green), "intermediate" (blue),
                  "advanced" (red). *
<document>     Defines a documentation file for the project **
<description>  Shown in the right hand side of the dialog in
                  Documentation section *
<file>          File name to view as a document. Relative path
                  from the project or absolute path *
<script>         Defines a script that to be sourced when the "Launch
                  Project" field is clicked **
<description>  Shown in the right hand side in the example
                  project pull down *
<file>          File name to source. Relative path from the
                  project or absolute path. *
<file>          Specifies a file name relative to the project (or
                  absolute) to be exported when export project is
                  clicked. **

```

XML Namespaces may be used as well as other normal XML syntax and features. To use a namespace such that multiple files can be combined you can define a namespace and associate it to the URI.

```

<MyNameSpace:toolkits
  xmlns:MyNameSpace="http://www.mentor.com/catapult/toolkit/1.0">
<MyNameSpace:folder> ...

```

The red syntax is a namespace and the string is arbitrary, but must be consistent. The blue string associates the namespace string with the Uniform Resource Identifier (URI).

Changing the Window Layout

The default layout of the windows in the Catapult session (see [Figure 3-8](#) on page 62) has all the primary project windows displayed and keeps them unobstructed as you work. All windows in the session are independent and can be resized, rearranged and undocked from the session window. To return the session to the factory default layout, choose the **Window > Reset Window Layout** menu item. When the Catapult session is closed, the current window layout is saved for the next session.

Floating, Dockable, Tabbed and Hidden Windows

All of the primary project windows can be switched between the following four display modes. The display modes for all other windows do not include **Hide** mode.

Floating:

Detached from session window.

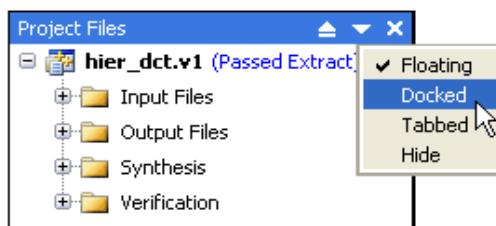
Dockable:

Contained within the session window and always visible.

- Tabbed Document:** Contained within the session window with other tabbed document windows. May be obscured by other tabbed document windows.
- Hide:** Window remains open but is not visible. To make the window visible again, use the **View** pulldown menu or the window activation buttons on the tool bar (see “[Window Activation Buttons](#)” on page 67).

To change the display mode use the drop-down menu in the upper right corner of the window, as shown in Figure 3-33.

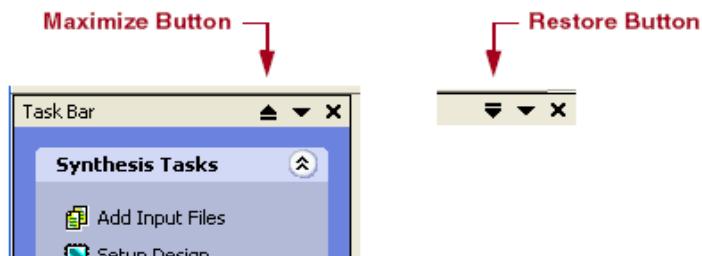
Figure 3-33. Window Display Mode Menu



Maximize Button for Sub-Windows

Every window inside the session window has a button to maximize/restore the window within the session window. When maximized, the window fills the entire Catapult session window. The button, shown in Figure 3-34, is in the upper right corner of each window.

Figure 3-34. Window Maximize and Restore Buttons



Resizing Docked Windows

To resize a window that is docked in the session window, use the mouse to grab an edge of the window, drag it to the desired size and release it. All adjacent windows are resized to accommodate the change.

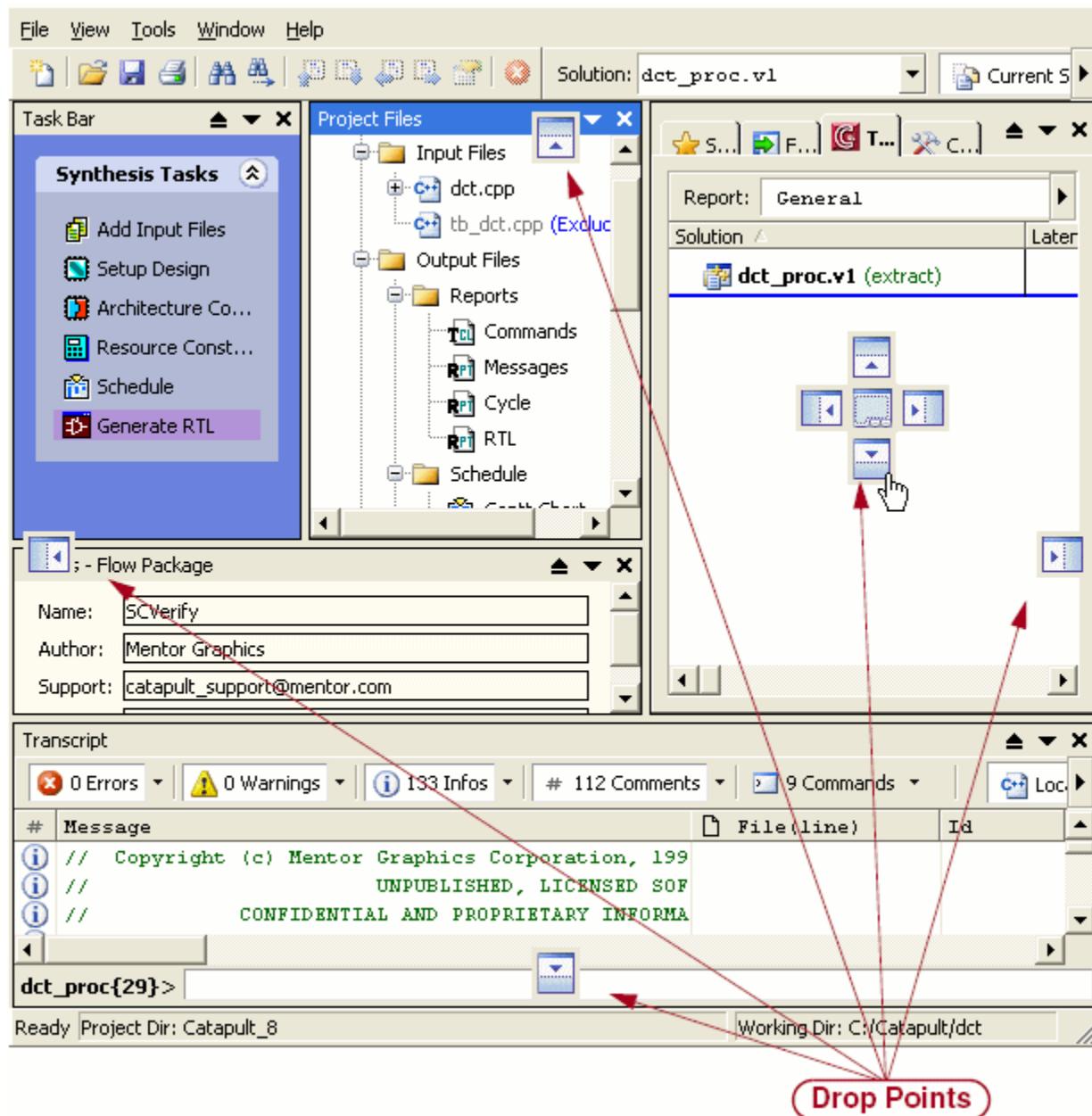
Rearranging Docked Windows

To move a window to a new location in the session window, use the mouse to grab the title bar of the window and drag it to a *drop point*. Drop points appear as soon as you start to drag the window. Figure 3-35 shows how the drop points appear in the session window and Table 3-3 describes each drop point.

Table 3-3. Drop Points for Rearranging Windows Inside the Session

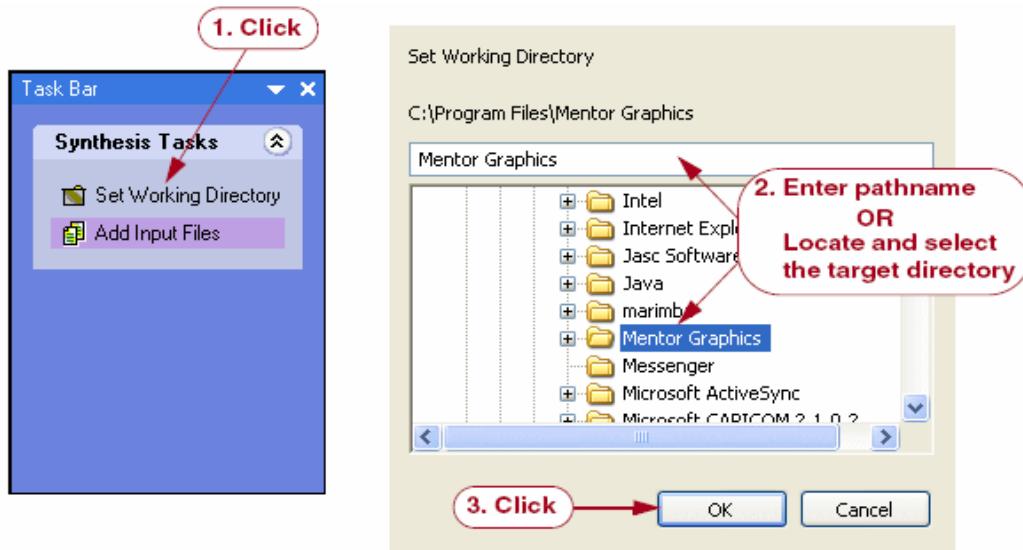
Window Drop Points	Descriptions
	Place window on right edge of the session window.
	Place window on left edge of the session window.
	Place window on top edge of the session window.
	Place window on bottom edge of the session window.
	This four drop point icon appears when the selected window is dragged over another window. Drop the window on one of the four points to place it adjacent to the corresponding side of the window below.
	This five drop point icon appears when the selected window is dragged over a tabbed window. Drop the window on one of the four side points to place it adjacent to the corresponding side of the tabbed window below. Drop the window on the center point to make it a tabbed window.

Figure 3-35. Moving a Window to a Drop Point



Setting the Working Directory

The default working directory is the current working directory of the shell from which Catapult was invoked. If that is not your target Catapult working directory (see “[Creating a Working Directory](#)” on page 45), the first step to take after invoking Catapult is to set the working directory. Figure 3-36 illustrates how to set the working directory by using the Task Bar. You can also set the working directory by using the `set_working_dir` command.

Figure 3-36. Manually Setting the Working Directory

You may set the working directory manually each time you invoke Catapult, or you can configure Catapult to automatically set it for you upon invocation by changing the General options settings. Refer to “[General Options](#)” on page 175. The options give the choice to have Catapult start in the same working directory every time, or to have it start in the working directory in which the last Catapult session ended.

Note

If you change the working directory setting, the log file for the current session is not automatically moved to the new working directory. Use the command “`logfile move <path>/catapult.log`” to relocate the log file.

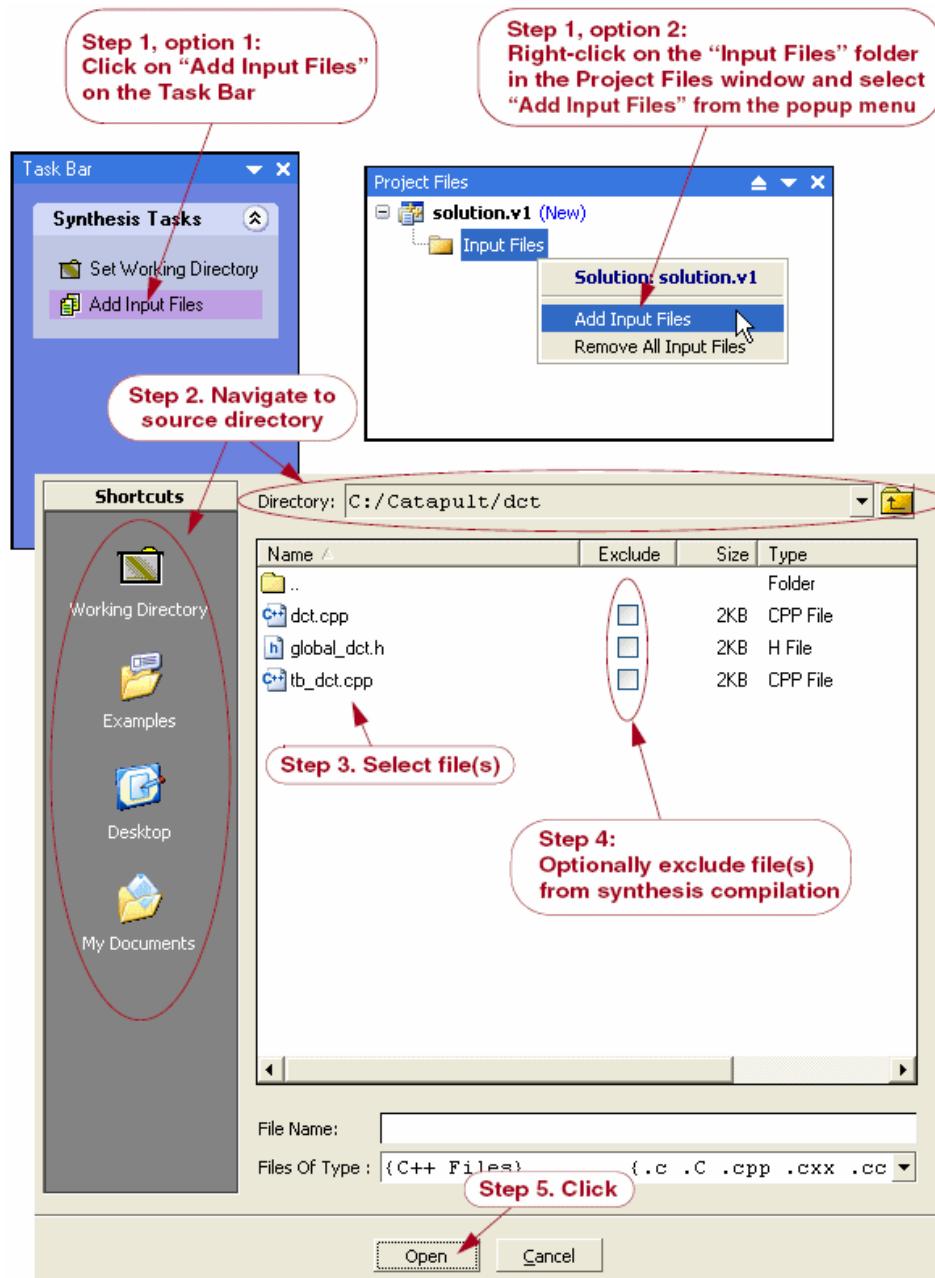
Note

If you change the working directory setting, the log file for the current session is not automatically moved to the new working directory. Use the command “`logfile move <path>/catapult.log`” to relocate the log file.

Adding and Removing Input Files

Catapult operates on the source code files that you add to the Input Files folder of the current solution. Figure 3-37 illustrates how to add input files. When a C++ source file is added, all of its included files, such as header files, are implicitly included.

Figure 3-37. Adding an Input File



By default, the Add Input Files dialog box displays the source code files available in the working directory. Click on the **Examples** icon on the shortcut panel (refer to Figure 3-37) to access the example designs shipped with the Catapult software.

To remove files from the Input Files folder, either right-click on the Input Files folder and choose the **Remove All Input Files** popup menu item, or right-click on a file in the Input Files folder and choose the **Remove** popup menu item. You can also add or remove input files by using the **solution file add** and **solution file remove** commands.

Preparing the Source Files

Catapult supports a number of pragmas that can be inserted into the source code to direct the synthesis process. Refer to “[Pragmas](#)” on page 341 for descriptions of all Catapult pragmas. The use of these pragmas is optional and Catapult provides equivalent commands in the user interface to achieve the same results. The pragmas are most useful for specifying settings that are not expected to change.

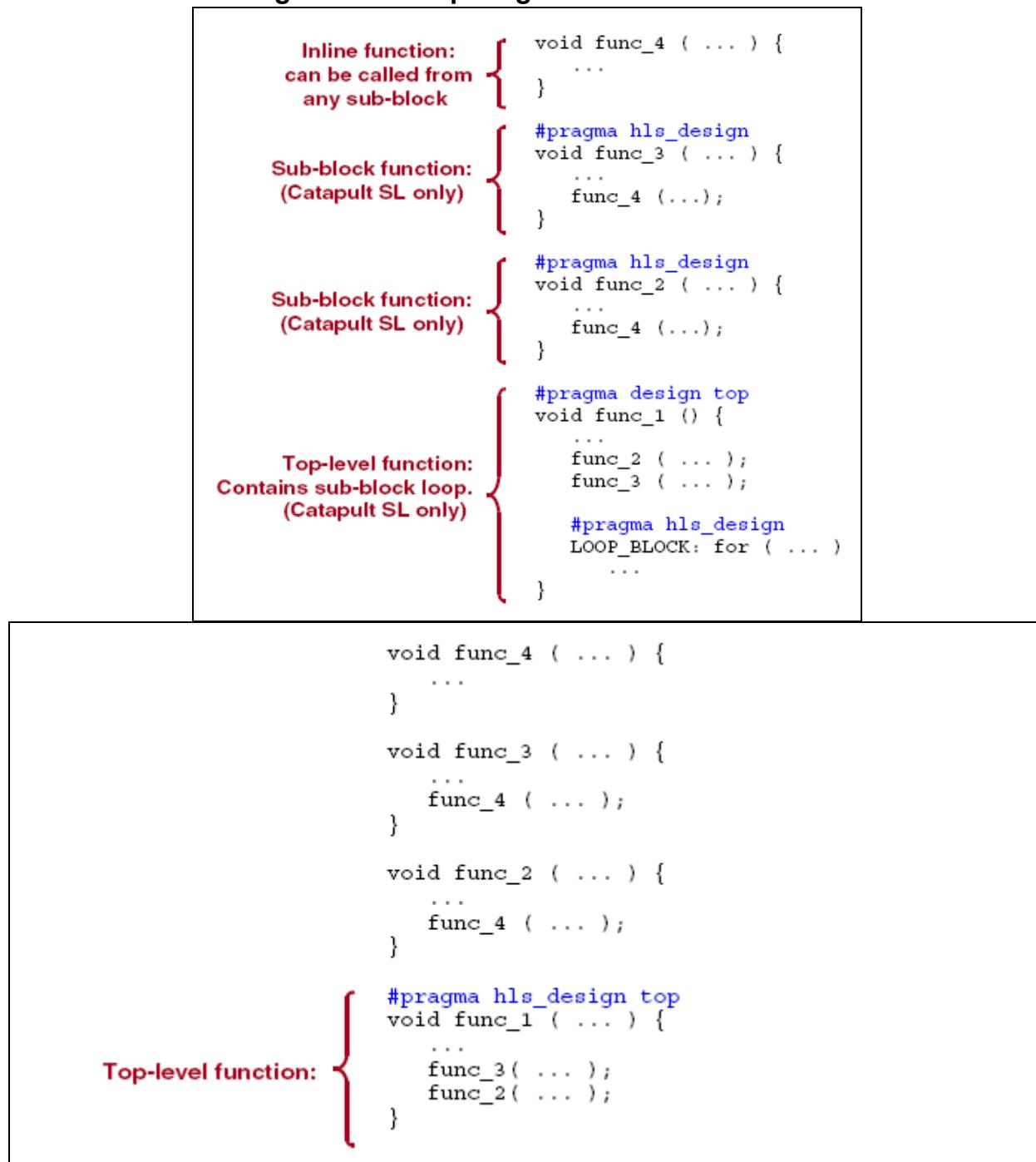
One good example is the `hls_design` pragma. It is used to designate which function is the top-level function in a design. Because Catapult synthesizes only one function, the top-level function of the design must be identified. Everything called by top-level function will be hardware and everything outside of that function is considered to be software or a test bench and is ignored during the synthesis process.

To identify the top-level functionin the source code, insert the “`#pragma hls_design top`” statement immediately before the top-level function definition. Figure 3-38 shows an example.

If a design contains multiple “`#pragma hls_design top`” statements, Catapult will issue a warning and select the last top detected.

If you do not want to add the `hls_design` pragma to your source code, Catapult allows you to interactively specify the top-level function after the design has been loaded. Refer to “[Setting the Top-Level Function](#)” on page 100.

Figure 3-38. Preparing the Source Code File



Setting Up the Design

After the C++/SystemC design is loaded into Catapult, you must specify some basic information about the target hardware and the hierarchical structure of the code from the *Setup Design* stage. For more information, see the following topics in this section:

- “Specifying the Technology and Clock Frequency” on page 97
- “Setting Up the Interface Control Signals” on page 98

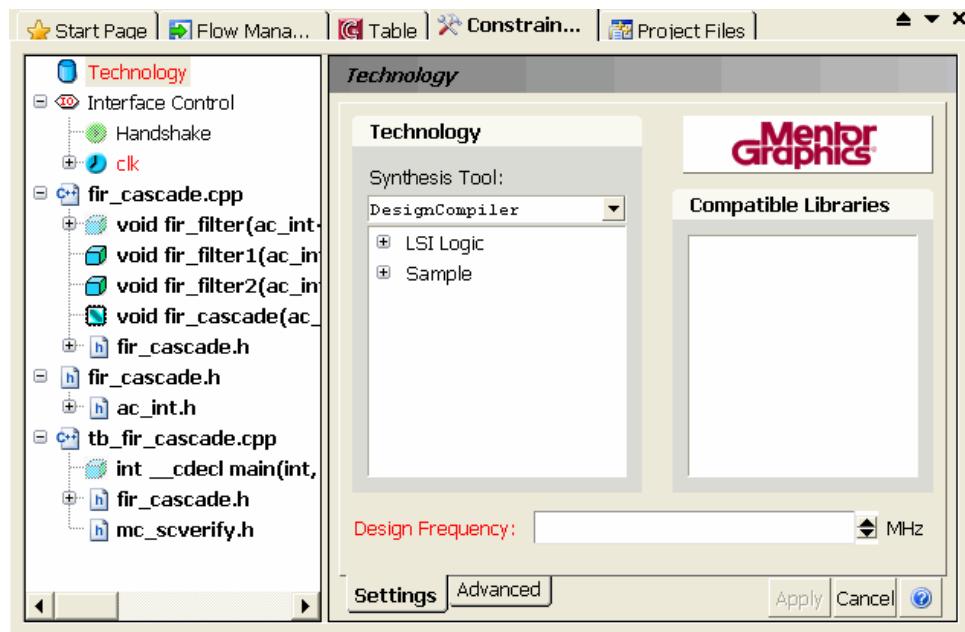
Specifying the Technology and Clock Frequency

Use the following procedure to specify the technology settings and clock frequency for your design.

Prerequisites

- Source design input files are loaded.
- Setup Task is active and the Constraint Editor window is open as shown in [Figure 3-39](#).

Figure 3-39. Design Technology Options



Procedure

1. Select the target RTL synthesis tool from the dropdown menu. Hardware technology and compatible library options associated with the compiler display.
2. Select the appropriate hardware technology and libraries for your design. The base library is checked by default because it is always needed.
3. Enter the design frequency and click **Apply**.

If necessary, click the **Advanced** tab and modify the default setting for the **Constant Multipliers** optimization. Refer to “[OPT_CONST_MULTS](#)” on page 329 for more information about this optimization. You can configure the system default value for this

field by changing its setting as described in the section “[Component Libraries Options](#)” on page 180.

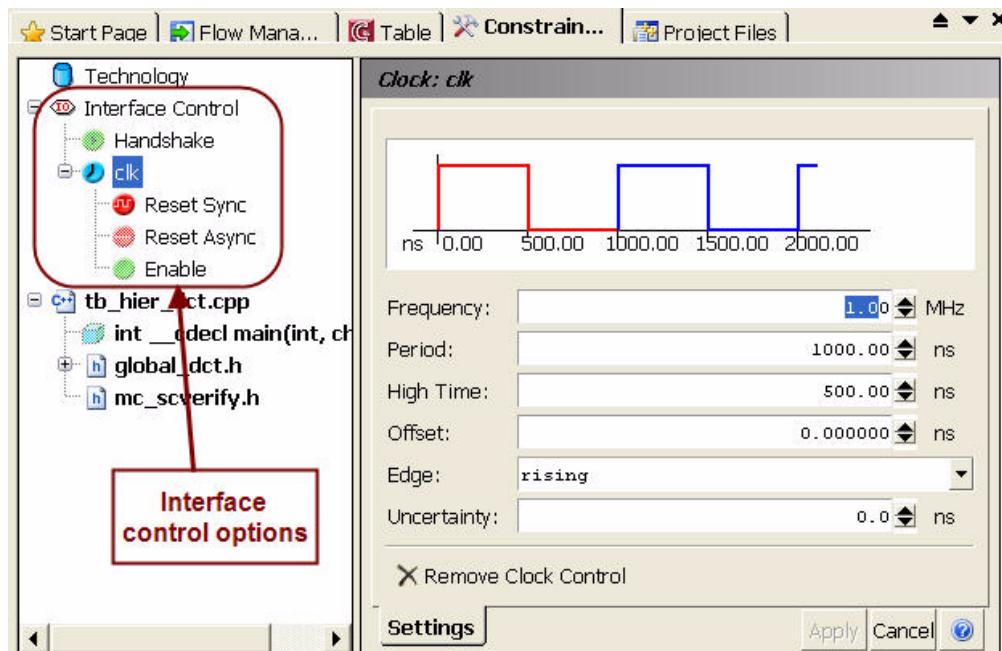
Related Topics

[Setting Up the Interface Control Signals](#)
[Setting Up Catapult Default Options](#)

Setting Up the Interface Control Signals

The signals for the global clock, reset, enable and process level handshake are configured from the Interface Control group as shown in Figure 3-40. Expand the “**Interface Control**” group and select one of the interface controls (**Handshake**, **Clock**, **Enable**, etc.) to access associated constraint settings. Except for the clock frequency setting, each setting is initialized to a system default value. You must supply the clock frequency. You can configure the system default values for these fields by changing the Interface option settings as described in “[Interface Options](#)” on page 181.

Figure 3-40. Hardware Interface Controls



Setting Handshake Signals

Selecting the **Handshake** interface allows you to enable/disable four types of handshake signals including a **Ready Flag**, **Start Flag**, **Done Flag** and **Transaction Done**. You can also edit the default signal names of the Ready, Start, and Done flags.

Note



These handshake signals are only valid for C++ designs. SystemC contains all the necessary synchronization signals for a design.

A design with a Start handshake waits for the start signal and then reads all the non-pointer inputs. A design with a Done handshake sets the done signal to high in the same cycle that the return value is written to the outputs. Both the Start and the Done handshake can be active in the same design. Optionally change the default names of the signals. The names must be string values supported by the target RTL language.

Enabling the **Transaction Done Signal** option adds transaction signals to all processes. Each process in a design will have one or more signals that go high for one cycle at the completion of an I/O transaction. More complex handshake protocols are supported by [Interface Synthesis](#) constraints set in the Architectural Constraints Editor window.

Note



When the **Transaction Done Signal** option is enabled, the **Done** flag is ignored.

Setting the Clock, Reset and Enable Signals

As shown in [Figure 3-40](#), selecting the **clk** interface allows you to modify the clock frequency (in MHz), period, high-time, offset, active edge, and clock uncertainty. The frequency and period values are tied together. When you set one, the other is computed automatically.

Selecting the **Reset Sync**, **Reset Async** or **Enable** interface allows you to add or omit (enable/disable) the signal in the design, edit the signal name, and specify whether it is active high or low.

- Input files are loaded.
- Setup Design phase is active and the Constraint Editor window is open.
 1. Click on the new clock. The clock settings display.
 2. Specify the clock frequency and other options as needed and click **Apply**.
 3. Repeat Steps 1- 4 to define each additional clock.

Related Topics

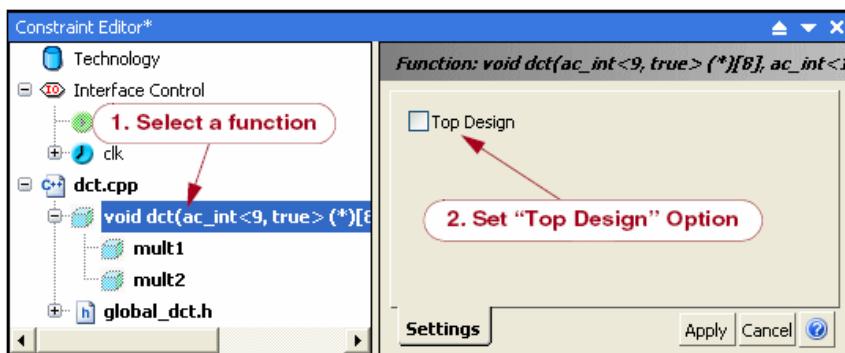
[Setting Up the Interface Control Signals](#)

[Setting Global Hardware Constraints](#)

Setting the Top-Level Function

For designs that have multiple functions within the C++ input file, you must specify one function to be the top-level function. As shown in Figure 3-41, the **Setup Design** window displays all of the functions in the design file and allows you to select one to be the top-level function.

Figure 3-41. Setting Top-Level Function in Setup Design Window



If a top-level function is specified in the source code with a “#pragma hls_design top”, then that function will be displayed as the top-level function in the **Setup Design** window. Setting a different top-level function in the **Setup Design** window will override the pragma in the source code.

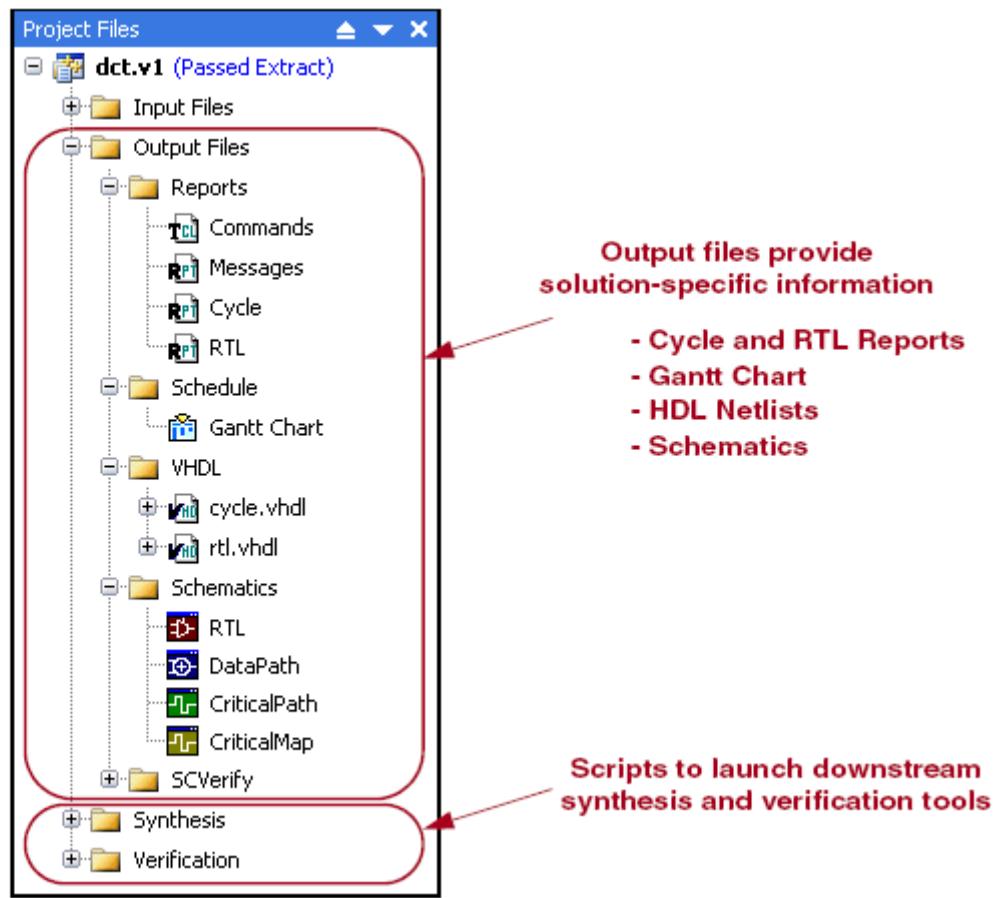
Design Analysis in Catapult

Catapult analyzes your design and provides detailed feedback to help identify problems in the algorithm and provide guidance about constraints or code changes necessary to meet design constraints. This section describes the design analysis features provided by Catapult.

Performing Design Analysis

Most of the information about your design is contained in the output files generated for each solution. These files include cycle and RTL reports, Gantt charts, HDL netlists and schematics. Comparative data about each solution’s results is provided in Bar Chart, XY Plot and Table windows. Each chart or table view compares the results of all solutions in the project. Cross-probing is another valuable feature to help you analyze your design. Figure 3-42 shows how the output files are organized in the Project Files window.

Figure 3-42. Catapult Output Files



Solution Output Files

All output files are specific to the solution and are saved in the solution directory. Double-click on a file to view its contents. The following list describes each type of output file.

- **Schedule** - Catapult creates a Gantt chart that provides a graphical view of the algorithm and architecture. See “[Scheduling the Design](#)” on page 127 for more information on how to use this tool.
- **Reports** - Catapult generates a Cycle report (*cycle.rpt*) and a RTL report (*rtl.rpt*) that provide information on the synthesis run. See “[Catapult Report Files](#)” on page 141 for more information on Catapult reports. The Directives file (*directives.tcl*) contains all of the directive commands, go commands, and input file paths needed to regenerate the solution. The Messages file (*messages.txt*) files contains all of the Catapult system messages generated by the solution.
- **Schematics** - Catapult generates a schematic and provides four different views of it: *RTL*, *Datapath*, *CriticalPath* and *CriticalMap*. See “[Catapult Schematics](#)” on page 143 for more information.

- **HDL Output** - Catapult can generate cycle-accurate and RTL netlists in VHDL, Verilog and/or SystemC. See “[Catapult HDL Output Files](#)” on page 142 for more information on Catapult HDL output.

Charts and Table Comparison of Solution Results

As the design progresses through the Catapult flow and new solutions are created, the project stores the area and latency results of each solution. The comparative results of all of the solutions is viewable in three different formats, an **XY Plot**, a **Bar Chart** and a **Table**. Use the XY-Plot and the Bar Chart views to get a graphical view of how close each solution is to meeting the design constraints. The Table view provides numerical data for area, latency and throughput cycles. See “[The Table Window](#)” on page 79 and “[The Bar Chart and XY Plot Windows](#)” on page 80 for more information.

Cross-Probing Between Source Code and Generated Output

The Catapult cross-probe feature allows you to trace items in your design across all input and output files in the solution. You can cross-probe from an output file to a source code file, or from an input file to any output file (Gantt chart, reports, HDL and schematics). Similarly, you cross-probe from an output file to any other output file. In addition, the cross-probe feature allows you to traverse the call stack.

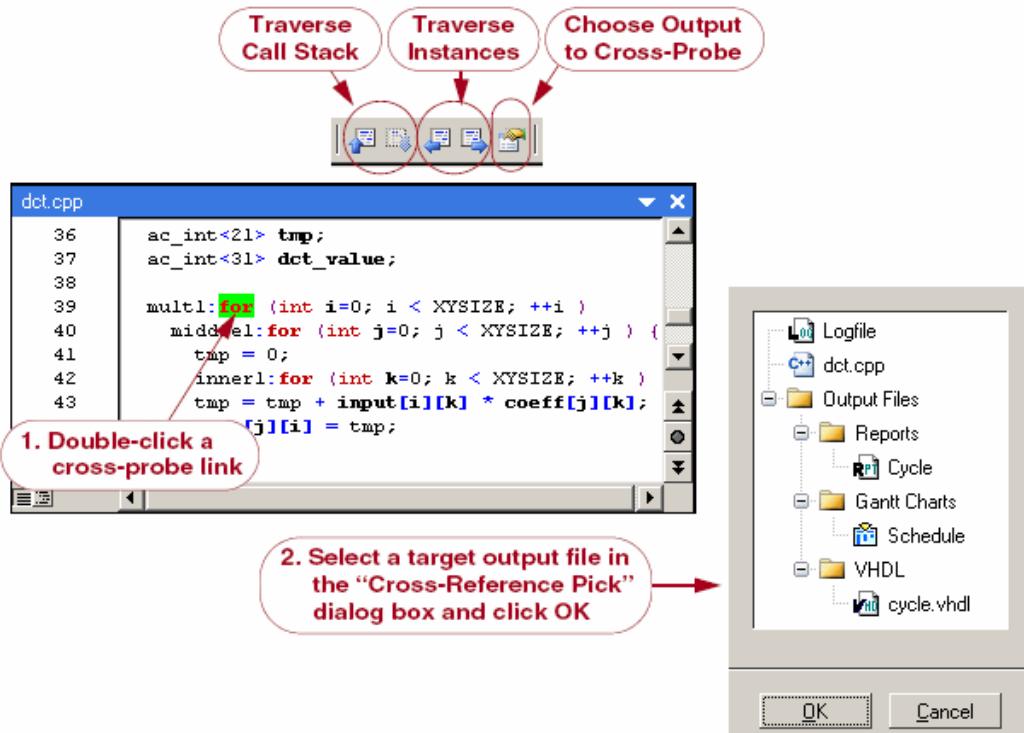
Cross-probe links are automatically included in all views of source and output files. Links in text are highlighted in a bold face font, and when the cursor hovers over them the text changes color and becomes underlined. Graphical links in schematics are highlighted in blue, although the highlighting can be toggled on or off. See “[Schematic Viewer Options](#)” on page 197 for information about how to change the highlight color (the “objecthighlight15” color is used for cross-probe links).

 **Note** Not all items can be cross-probed. For instance, operations that are optimized away do not appear in the generated output. Also, the synthesis process adds items that do not directly correspond to any particular item in the source code.

To use a cross-probe link, double-click on it, or right-click on it and choose **Cross-probe** on the popup menu. When you cross-probe from an output view, the corresponding source code file is displayed and the target item is highlighted.

Cross-probing from a source file will open the **Cross-Reference Pick** dialog box, shown in Figure 3-43. Select an output view and click OK to open the view.

Figure 3-43. Cross-Probing from Source to Output



Often items you cross-probe in the source code correspond to multiple items in the output files. The cross-probe initially shows you the first instance found in the output. Use the **Next Instance** and **Previous Instance** buttons on the tool bar to traverse the instances in the output.

Traversing the Call Stack

To traverse the call stack, use the **Up Call Stack** and **Down Call Stack** buttons on the tool bar (shown in Figure 3-43). When you select a cross-probe link, the buttons allow you to cross-probe up and down the call stack between that link and the top-level function call. Clicking the **Down Call Stack** button to traverse past the selected link (top of stack) is the same as cross-probing to an output file. Catapult will open the Cross-Reference Pick dialog box or cross-probe back to the initial output file which was first traced.

Cross-Probing from Precision RTL Schematics to Catapult HDL

The schematic viewer in the Precision RTL tool (v2008a and later), when launched from the Catapult GUI, allows you to cross-probe (trace back) objects to the HDL source code in the Catapult session window. The procedure is as follows:

1. When setting up the design in Catapult, select Precision in the “Synthesis Tool” field.
2. Enable the Precision RTL flow option “Run Next Generation User Interface” to ensure that the new Precision GUI is active when Catapult launches the tool. Only the new GUI

supports cross-probing to Catapult. Refer to “[Precision RTL Flow Options](#)” on page 201 for more information about the option.

3. Generate the synthesis script by selecting the “Generate RTL” task, or calling the “go extract” command.

4. Run the Precision script generated by Catapult to launch the Precision RTL tool:

```
flow run /Precision/precision -file <solution_dir>/rtl.vhdl.psr
```

5. In the Precision tool, open the schematic viewer, right-click on an object in the schematic and select “Trace to Catapult HDL Source” on the popup menu. That command will cause the source HDL file to appear in the Catapult session window and the line in the file corresponding to the object in the schematic will be highlighted.

Design Analysis Checklist

The following is a list of things you should check when running a design within Catapult:

- Are all of the I/O bitwidths correct?
- Are the bitwidths and the number of elements correct?
- Are the loop iterations correct?
- Are the bitwidths of all the internal operations correct?
- Are the timing and area goals met?
- Does the design simulate as expected?
- What kind of hardware architecture was built?
- How does this solution compare to others?

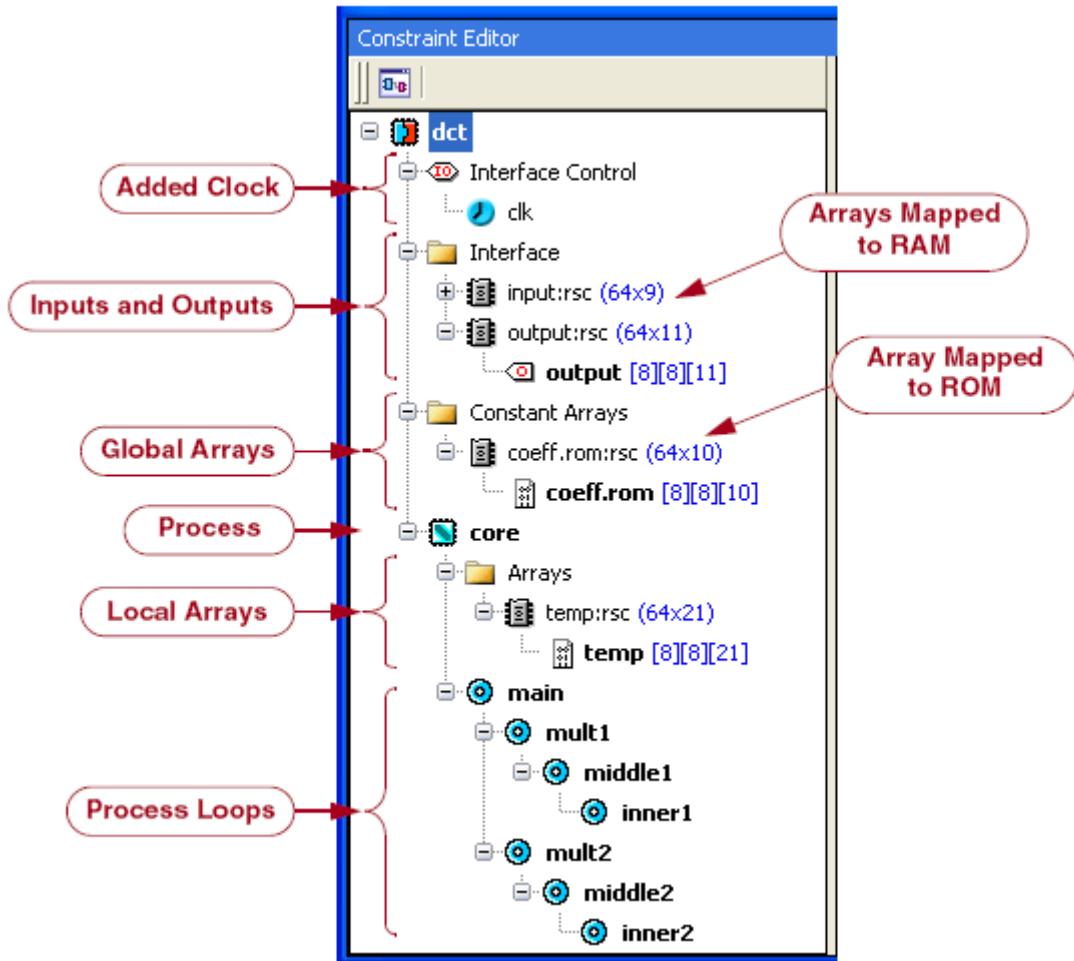
Specifying Architectural Constraints

The first time a design enters the *Architectural Constraints* stage of the work flow, Catapult automatically initializes all of the constraints. The initial settings are either inferred from the design or set to predefined default values. You should review these initial settings in the Constraint Editor window and modify them if necessary. The design browser portion of the Constraint Editor window provides a graphical view of the ports, arrays, processes and loops in the design and can be displayed in hierarchical tree format or as a block diagram. The tree view is the default. For information about the block diagram view, refer to “[Block Diagram Constraint Editor](#)” on page 124.

Figure 3-44 identifies all of the objects that can be constrained in this state of the work flow. The **Interface** folder contains all of the I/O ports on the design. Catapult automatically maps the I/O data variables in the source code to *input*, *output* or *inout* resources. Notice in Figure 3-44 that all port variables and all arrays are automatically mapped to separate resources. The

resource objects are automatically created when the design is compiled. All arrays in the design are listed and can be optionally mapped to registers, RAMs, or ROMs. For each array, the number of elements and the bit width of those elements is listed.

Figure 3-44. Architectural Constraints Graphical View of Design



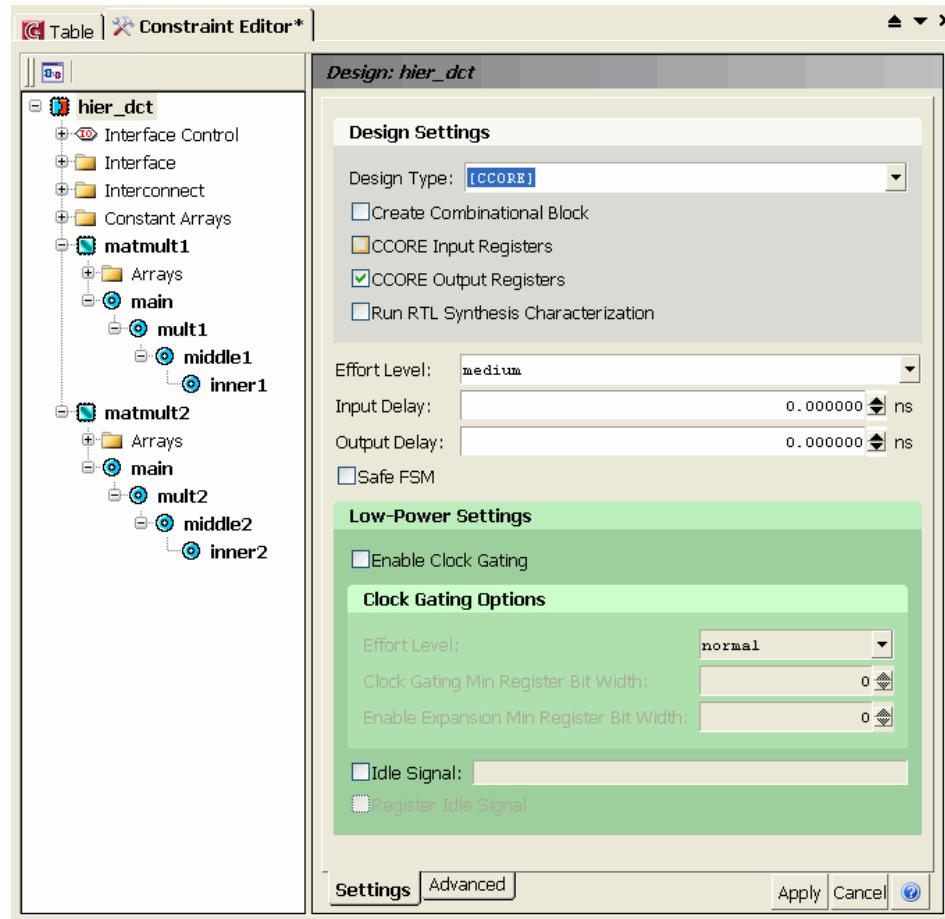
The constraint options vary depending on the type of object selected in the design browser. The following sections describe the architectural constraint options available for each type of design object.

- “Architectural Constraints on the Design” on page 106
- “Architectural Constraints on Processes” on page 112
- “Architectural Constraints on Resources” on page 114
- “Architectural Constraints on Variables Assigned to Resources” on page 118
- “Architectural Constraints on Loops” on page 120

Architectural Constraints on the Design

A set of global architectural constraints display when the top *design* object is selected as shown in [Figure 3-45](#). These constraints consist of design optimization and configuration options that apply to the entire design during hardware generation.

Figure 3-45. Global Architectural Constraints



Design Settings

- Design Type: [CCORE]
 Create Combinational Block
 CCORE Input Registers
 CCORE Output Registers
 Run RTL Synthesis Characterization

Effort Level: medium

Input Delay: 0.000000 ns

Output Delay: 0.000000 ns

Safe FSM

Low-Power Settings

- Enable Clock Gating

Clock Gating Options

Effort Level: normal

Clock Gating Min Register Bit Width: 0

Enable Expansion Min Register Bit Width: 0

Idle Signal:

Register Idle Signal

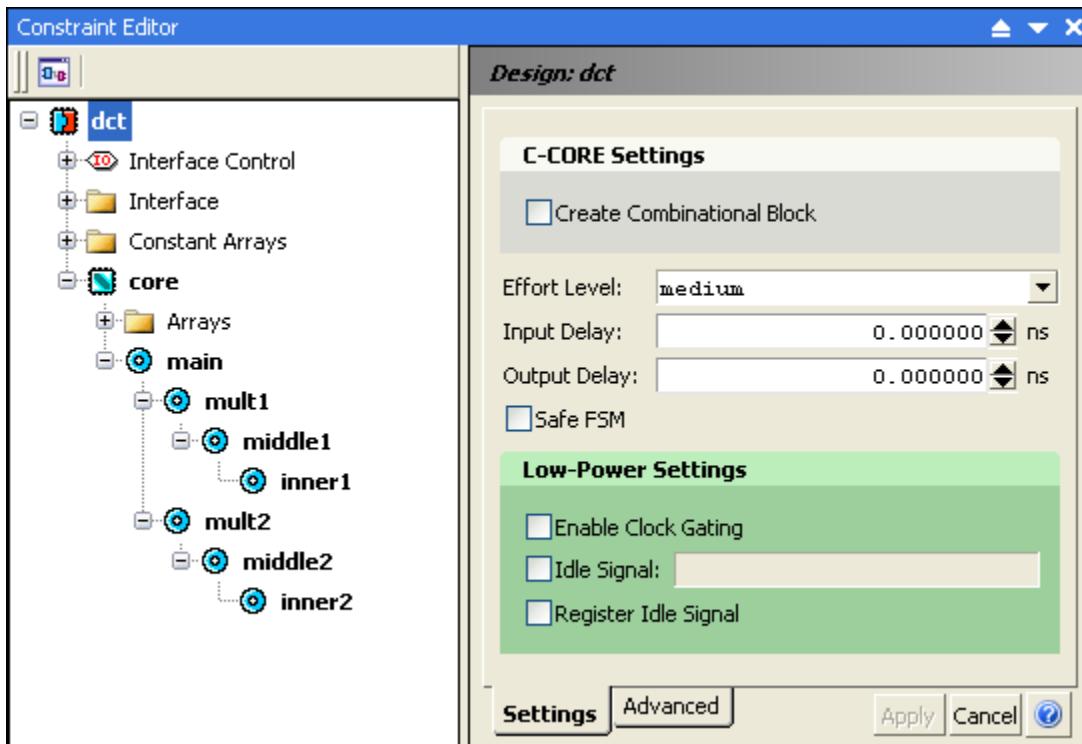
Settings

Advanced

Apply

Cancel





Settings Tab:

- **Design Type**

Specifies the design architecture. Select CCORE to specify CCORE options or leave blank for top-level logic. Depending on the specified design type, the following Design Setting options display:

 - **Create Combinational Block** — Specifies the design/CCORE is combinational logic. By default, the design type is sequential logic. The [CREATE_COMBINATIONAL](#) directive controls this option.
 - **CCORE Input Registers** — Determines if registers are created on the CCORE inputs. By default, no input registers are created. The [INPUT_REGISTERS](#) directive controls this option.
 - **CCORE Output Registers** — Determines if registers are created on the CCORE outputs. By default, output registers are created. The [OUTPUT_REGISTERS](#) directive controls this option.
 - **Run RTL Synthesis Characterization** — Synthesizes the CCORE with the downstream RTL synthesis tool and uses the characterization data instead of estimated data. The [CHARACTERIZE_OPERATOR](#) directive controls this option.

For more information, see “[CCORE Design Flow](#)” on page 597.

- **Effort Level**

Specifies the amount of effort used to optimize a design during scheduling. Options include:

- **high** — Applies maximum effort to optimize the area and may take up to ten times longer than the medium effort.
- **medium** — Applies the minimum effort required to produce a viable schedule based on specified design goals. Default.

The [EFFORT_LEVEL](#) directive controls this option. For more information, see “[Scheduler Optimization](#)” on page 276

- **Input Delay and Output Delay**

These options set the global input/output delay values for all I/O ports in the design. To modify the global delay on an individual port, edit the resource constraints for that port. These options set the [INPUT_DELAY](#) and [OUTPUT_DELAY](#) directives. For more information, see “[Setting I/O Delay Constraints](#)” on page 295.

- **Safe FSM**

Enables safe FSM (Finite State Machine) flows. When enabled, Catapult allows a default state that is isolated from other states during RTL simulation. By default, the reset state is used. If no reset state exists, the first state in the FSM is used. The [SAFE_FSM](#) directive controls this option. For more information, see “[FSM State Encoding](#)” on page 591.

- **Enable Clock Gating:**

Enables clock gating. When enabled, explicit clock enable signals are added to registers for clock gate insertion by downstream RTL synthesis tools. For more information, refer to “[Clock Gate Insertion](#)” on page 589. The [GATE_REGISTERS](#) directive sets this option.

- **Effort Level:**

Specifies the amount of effort used to implement low-power clock gating. Options include:

- **normal** — Main data storage elements of FIFO/Register file cores are gated, where explicit enable signals are placed on all design sequential nodes.
- **high** — In addition to the main data storage elements, control registers holding the gating information for the main data storage elements are sequentially clock gated based on groups of eight, 1-bit registers.

The [GATE EFFORT](#) directive sets this option. For more information, see “[Clock Gate Insertion](#)” on page 589.

- **Clock Gating Min Register Bit Width:**

Specifies a minimum register bit width for the *normal* clock gate effort. By default, 0 (no constraint) is selected.

The [GATE_MIN_WIDTH](#) directive sets this option. For more information, see “[Clock Gate Insertion](#)” on page 589.

- **Enable Expansion Min Register Bit Width:**

Specifies a minimum register bit width for the *high* clock gate effort level. By default, 0 (4-bit) is selected.

The [GATE_EXPAND_MIN_WIDTH](#) directive sets this option. For more information, see “[Clock Gate Insertion](#)” on page 589.

- **Idle Signal:**

This option adds idle indicator signals to process blocks in the design. An idle signal is driven HIGH if all three of the following conditions are true:

- The Catapult design is waiting for input data.
- No inputs are supplied to the design.
- All the output data has been read from the output ports of the design (non-pipelined designs and designs with distributed pipeline).

Enabling the Idle Signal option on the top block creates a *global* idle signal for the design by adding individual signals to all sub-blocks that feed into (ANDed together) the signal port at the top level. For more information about how idle signals are implemented in Catapult, refer to “[Idle Signal Insertion](#)” on page 592. This option sets the [IDLE_SIGNAL](#) directive.

- **Register Idle Signal:**

This option adds a register to an idle indicator signal. This option sets the [REGISTER_IDLE_SIGNAL](#) directive.

Advanced Tab:

- **Speculative Execution**

Enable/disable the Speculative Execution optimization. This optimization analyzes conditional logic (such as if-else) in the design in order to reduce latency. It analyzes all conditional branches in order to find operations that are not dependent on the condition, and therefore can be scheduled in a different clock cycle. When enabled, Speculative Execution is not employed unless it is needed. Catapult will attempt to schedule the design without Speculative Execution. Only if scheduling fails due to pipelining or cycle constraints will Catapult attempt to optimize with Speculative Execution and reschedule. This option sets the [SPECULATE](#) directive.

- **% Assignment Overhead**

Specify the percent of the clock period that will not be used by sharing and re-allocation. This constraint can be used to reserve some of the clock period for routing delay in the back-end flow. The value can be a negative number. Note that “**% Assignment Overhead**” is a different constraint than the “Percent Sharing Allocation” constraint. Refer to “[Percent Sharing Allocation](#)” on page 274 for more information. This option sets the [ASSIGN_OVERHEAD](#) directive.

- **FSM State Encoding**

This option sets the low-power state encoding method for FSMs. FSMs can potentially improve their power consumption numbers by employing one of the following low-power state encoding methods: none, binary, grey, onehot. Sub-processes inherit the setting on the top process unless individually overridden. For more information, refer to “[FSM State Encoding](#)” on page 591. This option sets the [FSM_ENCODING](#) directive.

- **Reset Clears All Registers**

This option sets the [RESET_CLEAR_ALL_REGS](#) directive. When enabled, the reset signal will reset all registers in the design. Otherwise reset only those registers that need to be reset for correct simulation results.

- **Use Old Scheduling and Allocation Algorithms**

Use this option to switch between the new combined allocation and scheduling algorithm (new scheduler) introduced in the 2007a release and the old scheduler. The new scheduler can reduce area on many designs because it is able to change the type and number of allocated components while the design is being scheduled. This option sets the [OLD_SCHED](#) directive, described on page 329. The new scheduler is enabled by default.

This release is part of a transition period during which both schedulers are available in Catapult. At the end of the transition period (a future release), the old scheduler will be removed. When running the old scheduler, the following warning message appears in the transcript informing you that the old scheduler is being used:

Warning: This scheduling mode will not be supported in future releases of Catapult. Please set directive "OLD_SCHED" to false.

Note



If you use a “directives.tcl” file generated by Catapult 2007a or 2007a_update1, the old scheduler will be enabled because that was the default behavior in those versions. You can change the file to use the new scheduler by changing the OLD_SCHED setting to false.

- **Add an Enable Signal for External Memories**

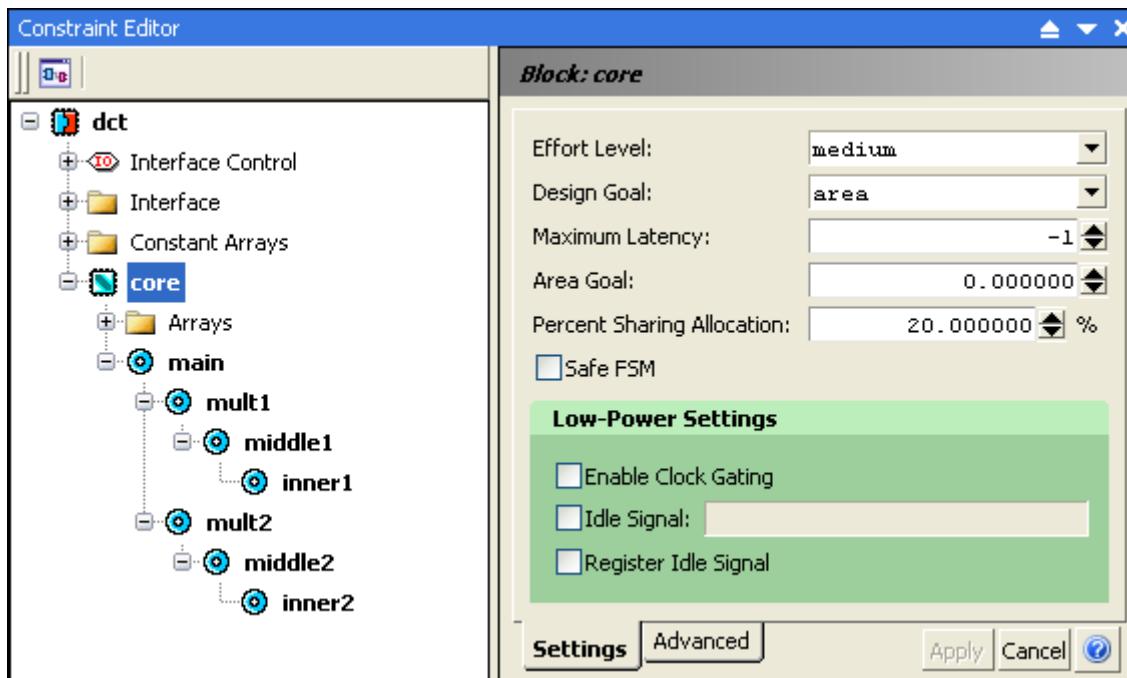
Adds an enable signal for external memory. Beginning with the 2006a release, Catapult no longer netlists an enable port driven to a constant for external components. If this is enabled, Catapult will match the behavior in the pre-2006a releases and add an enable port that is driven to a constant if the external component has an enable port and the

design doesn't have an enable. This option sets the [GEN_EXTERNAL_ENABLE](#) directive.

Architectural Constraints on Processes

The architectural constraints shown in [Figure](#) display when a process is selected.

Figure 3-46.



Settings Tab:

- **Effort Level**

Specifies the amount of effort to use for design optimization. Options include: “high” or “medium” (default). High effort level will generally take up to 10 times longer and give a better area result. This option sets the [EFFORT_LEVEL](#) directive. It applies only when the new scheduler is enabled (see [“Use Old Scheduling and Allocation Algorithms”](#) on page 111).

- **Design Goal**

Use this constraint to specify the optimization goal for the new scheduler. Valid values are either “area” or “latency.” This option sets the [DESIGN_GOAL](#) directive. The Design Goal setting is similar to, but mutually exclusive to the Component Grade setting for the old scheduling algorithm. This option is available only when the new scheduler is enabled (see [“Use Old Scheduling and Allocation Algorithms”](#) on page 111).

- **Maximum Latency**

Use this constraint to specify the maximum number of cycles allowed for the design. A setting of “-1” (default) mean no maximum is set (unlimited). This option sets the **MAX_LATENCY** directive. It applies only when the new scheduler is enabled (see “[Use Old Scheduling and Allocation Algorithms](#)” on page 111).

- **Area Goal**

Use this constraint to specify a size goal for area optimization. A setting of “0.0” (default) mean no area goal is set. This option sets the **AREA_GOAL** directive. It applies only when the new scheduler is enabled (see “[Use Old Scheduling and Allocation Algorithms](#)” on page 111).

- **Percent Sharing Allocation**

This setting corresponds to the Clock Overhead directive (see “[CLOCK_OVERHEAD](#)” on page 312). Different processes can have different percentages. Once this value is changed, all new solutions will use this value to override the default the “Percent Sharing Allocation” constraint for that process. See “[Percent Sharing Allocation](#)” on page 274 for more information.

- **Safe FSM**

See “[Safe FSM](#)” on page 109.

- **Enable Clock Gating**

See “[Enable Clock Gating:](#)” on page 109.

- **Effort Level:**

See “[Effort Level:](#)” on page 109

- **Clock Gating Min Register Bit Width:**

See “[Clock Gating Min Register Bit Width:](#)” on page 109

- Enable Expansion Min Register Bit Width:

See “[Enable Expansion Min Register Bit Width:](#)” on page 110

- **Idle Signal**

See “[Idle Signal:](#)” on page 110.

- **Register Idle Signal**

See “[Register Idle Signal:](#)” on page 110.

Advanced Tab:

- **Speculative Execution**

See “[Speculative Execution](#)” on page 110.

- **FSM State Encoding**

See “[FSM State Encoding](#)” on page 111.

- **Max Cycles**

This setting corresponds to the **MAXLEN** directive. It sets the maximum latency is for the process. Catapult will use this setting to reduce the area of the design. This setting can only be used if the iterations are set for every loop in the process.

- **Component Grade**

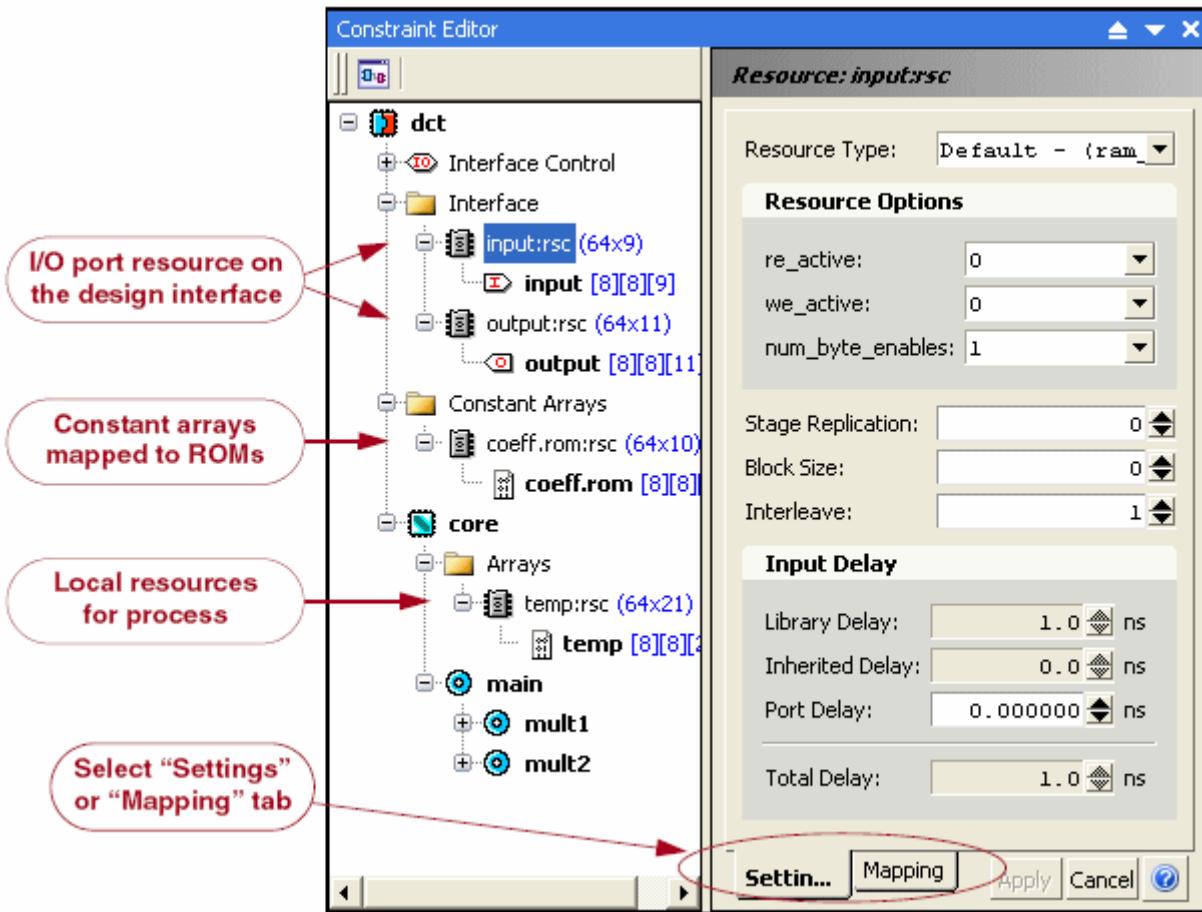
Sets the **COMPGRADE** directive which specifies the component selection criteria (speed vs. size) used during allocation. Valid settings are: “**fast**”, which selects the fastest component; “**small**”, which selects the smallest component; “**product**” which selects the component with the lowest “product” value, where product = (Area x Delay). This option is similar to, but mutually exclusive to the Design Goal setting for the new scheduling algorithm. This option is available only when the old scheduler is enabled (see “[Use Old Scheduling and Allocation Algorithms](#)” on page 111).

Architectural Constraints on Resources

An important concept in Catapult C Synthesis is the creation and utilization of *resources*. A resource, in this context, is an internal data object created in the in-memory database. A resource serves as an intermediate mapping object. Once a resource is declared (either automatically or manually), you can use the GUI to map one or more variables to the resource, map the resource to a particular type of component, set constraints on the resource, and view a graphical layout of the resource in physical memory.

As shown in Figure 3-47, select a resource in the design browser portion of the Constraint Editor to access its constraint options. The icon next to a resource indicates the type of component that is mapped to it. The blue numbers in parenthesis next to array resources show the size of the object expressed as <num_of_words>x<word_width>. For example, in Figure 3-47 the I/O resource “input:rsc” is composed of 64 words that are 9 bits wide. That size is derived from its array variable (“input”) that is an 8x8 two dimensional array of 9-bit elements ($8 \times 8 = 64$ words).

Figure 3-47. Mapping a Resource to Specific Hardware



The Settings Tab

Use the Settings tab to assign a library component type to the selected resource, and configure other constraints related to timing control, memory array splitting and physical memory mapping. The set of constraints available on the Settings tab vary based on the following factors:

- The kind of resource that is selected (I/O port, constant array or local array).
- The type of component that is mapped to the resource (set in the Resource Type field).
- The number of variables sharing the selected resource.

All possible fields are described in the following list. For a detailed discussion about how to use these constraints refer to “[Memories and Arrays](#)” on page 248.

- **Resource Type**

This option sets the [MAP_TO_MODULE](#) directive on the selected resource. A default component is automatically selected by Catapult based on several factors. For more information, see “[Predictive Resource and Memory Mapping](#)” on page 258.

You can select other component types from the drop-down list in the “Resource Type” field. The set of components that appear in the list depends on the type of resource that is selected, the data type of the variable assigned to the resource, and the set of IP libraries you selected in the “Compatible Libraries” field of the Setup Design window. For more information about the various component types, refer to “[Basic I/O Components](#)” on page 288 and “[DirectInput Resource Type](#)” on page 270.

- **Resource Options**

This field allows you to modify user-configurable settings defined in the component library. The options contained in this field are specific to the component currently mapped to the resource. If the component does not have any user-configurable options defined, then the Resource Options field does not display. For an example, refer to “[Bytewise Write Enable Memories](#)” on page 265.

- **Packing Mode**

If multiple variables are sharing the selected resource, Catapult aligns the variables in physical memory according to the specified Packing Mode algorithm.

- Choose **absolute** to preserve, if possible, the Base Bit and Base Address offsets, and the Word Width setting specified for each variable. If an address overlap is encountered, compact mode will be used by default.
- Choose **sidebyside** to align the variables along word width boundaries.
- Choose **compact** to compress the variables as close as possible in vertical alignment.
- Choose **base_aligned** to align the variables on power of 2 indices.

For more information, refer to “[Packing Mode Algorithms](#)” on page 253. This option sets the [PACKING_MODE](#) directive. **Packing Mode** is not displayed for resource types such as Register and DirectInput.

- **Stage Replication**

This option is only available on I/O ports. It specifies the maximum number of times the resource can be replicated. I/O ports can be replicated to increase parallelism for pipelining. This option sets the [STAGE_REPLICATION](#) directive on the selected resource.

This option takes an integer values from 0 to N, where N is any positive integer. The value of N is the number of RAMs to use on the interface. The values 0 - 2 have special meaning:

0 = No replication for I/O ports (default)

1 = No replication

2 = Ping-pong replication

3 and greater = Round-robin replication

- **Block Size**

Use this option to split a memory resource (array) into multiple blocks. The integer setting specifies the size of each block. Set it to zero to disable Block Size splitting. When an array is mapped to a memory (RAM or ROM), the block size must be either zero or greater than 1. If it is mapped to an I/O component, such as a wire, the block size value must be either zero or one. Invalid settings will generate an error later in the flow.

This option works in conjunction with the Interleave option (below) to allow you to split a memory resource into an arbitrary number of new memory resources of the same type as the original memory resource. It sets the **BLOCK_SIZE** directive on the selected resource. For a detailed discussion of how memory splitting works, refer to “[Splitting Memory Resources](#)” on page 260.

- **Interleave**

Set this option on RAM or ROM resources to split the memory resource into several new resources and distributes elements among new resources in interleaving fashion. The integer setting specifies the number of blocks to be created. Set it to zero to disable Interleave splitting.

This option works in conjunction with the Block Size option (above) to allow you to split a memory resource into an arbitrary number of new memory resources of the same type as the original memory resource. It sets the **INTERLEAVE** directive on the selected resource. For a detailed discussion of how memory splitting works, refer to “[Splitting Memory Resources](#)” on page 260.

- **Input Delay/Output Delay**

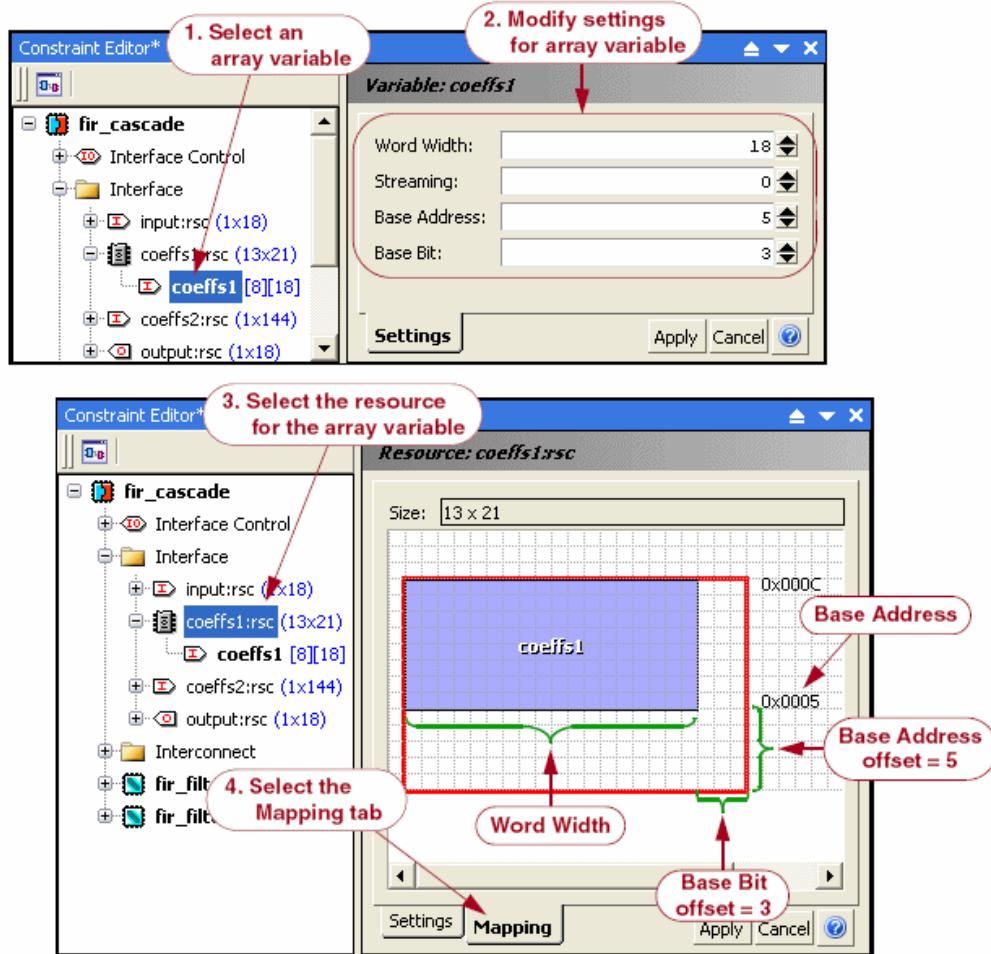
These options modify the total delay on the port resource. They set the **INPUT_DELAY** or **OUTPUT_DELAY** directives depending on the type of I/O port. Refer to “[Setting I/O Delay Constraints](#)” on page 295 for more information about how I/O delays are managed by Catapult.

The Mapping Tab

Select the **Mapping** tab to see a graphical representation of how variables assigned to the resource are mapped to physical memory. The mapping view shows the size of the resource and the packing mode. The map grid shows how all of the bits of each variable are laid out.

Figure 3-48 illustrates how the Mapping tab represents the settings on a resource variable. The settings on the resource variable control the Word Width, Base Address offset and Base Bit offset. For more information, refer to “[Mapping Arrays in Physical Memory](#)” on page 251 and “[Packing Mode Algorithms](#)” on page 253.

Figure 3-48. Viewing Map of Physical Memory



Architectural Constraints on Variables Assigned to Resources

The architectural constraints accessible when a resource variable is selected are described below and shown in Figure 3-48. You can change the word width of the variable, its physical address in memory, and enable streaming.

- **Word Width**

The word width constraint modifies the default word width of the selected resource variable. For memory resources, use this constraint to configure how the array variable is mapped in memory. For I/O resources, use the constraint to split the variable into multiple I/O ports. This option sets the **WORD_WIDTH** directive. For more information, refer to “[Word Width Constraint](#)” on page 255.

- **Streaming**

Specify the number of dimensions to stream. See the “[Memory Streaming](#)” on page 271 for more information. This option sets the **STREAM** directive.

- **Base Address**

The memory address where the variable will start. See “[The Mapping Tab](#)” on page 117. This option sets the **BASE_ADDR** directive.

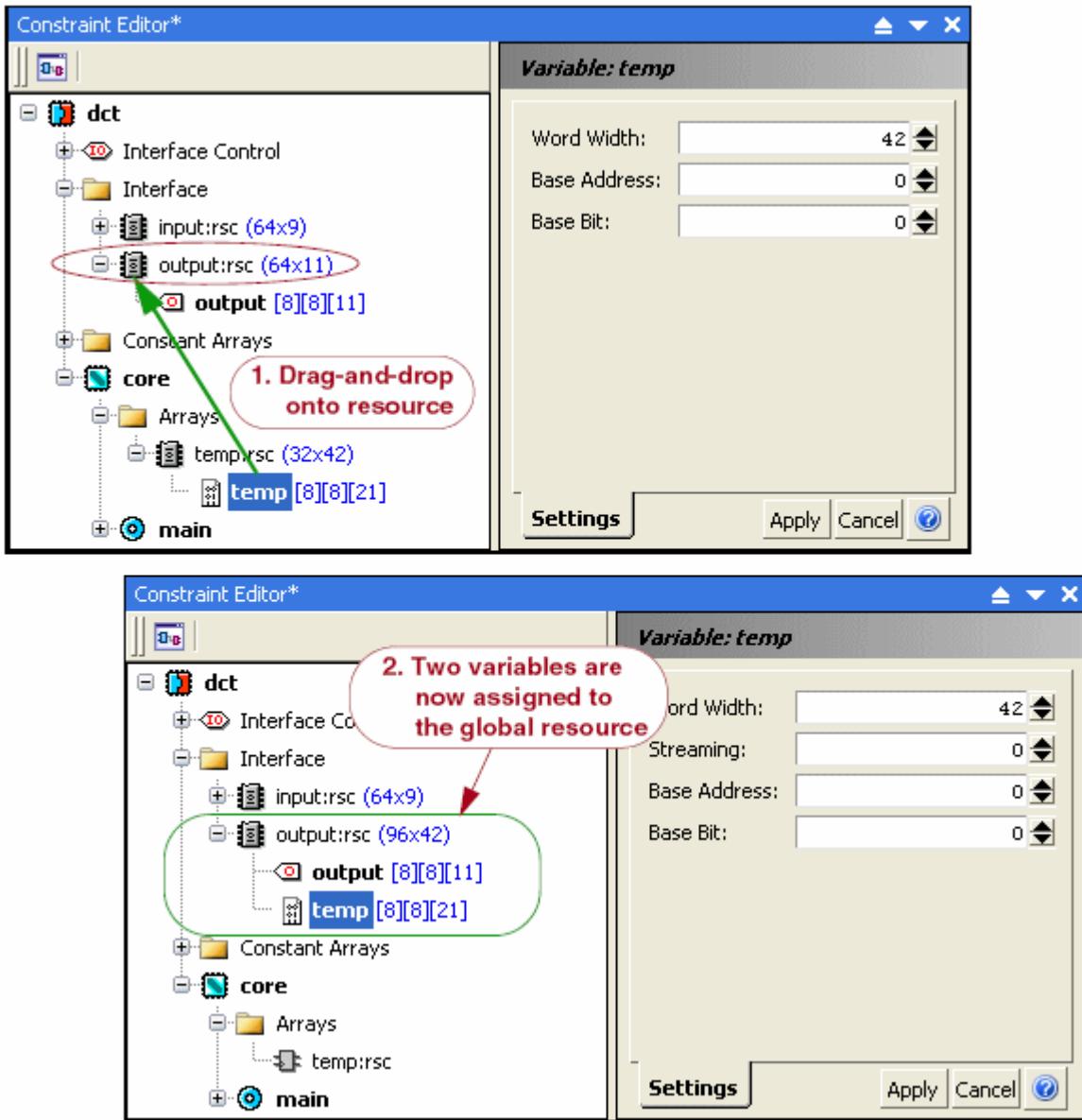
- **Base Bit**

The base bit where the variable will start. See “[The Mapping Tab](#)” on page 117. This option sets the **BASE_BIT** directive.

Assigning Multiple Variables to One Resource

The Constraint Editor allows you to reassign one or more variables to a resource. Figure 3-49 illustrates how to re-assign an internal variable to a global resource. In the example, the “**temp**” variable is reassigned to the “**output:rsc**” resource. Notice that because the word width of the “**temp**” variable was set to 42, the resource size increases from **(64x11)** to **(96x42)**.

Figure 3-49. Re-assigning a Variable to Another Resource



Architectural Constraints on Loops

The architectural constraints accessible when a loop is selected are shown in Figure 3-50. The figure also points out the different loop icons Catapult uses to indicate which constraints are currently applied to each loop. For a detailed discussion about these settings, refer to “[Loop Constraints](#)” on page 225.

- **Iteration Count**

This option specifies the number of times the loop runs before it exits. It sets the [ITERATIONS](#) directive. Catapult analyzes every loop to determine if the loop has a

maximum number of iterations. Conditional exits, which may cause the loop to exit early are not included in this calculation. If the maximum number of loop iterations can be determined, Catapult will display it here. If the maximum iteration count can't be determined, this field will be blank.

You may edit this field if you know the maximum number of iteration for a loop. Setting this value too low, however, will break your design. Iteration count is used by the optimizations in Catapult, so a C++/SystemC design that has a loop without an iteration count must have C++/SystemCthe bitwidths defined for the internal variables. The only exception for this rule is the “main” loop.

Refer to “[Constraining Loop Iterations](#)” on page 229 for more information.

- **Unroll and Partial**

Enable the **Unroll** field and optionally set the **Partial** field to specify the number of times to copy the loop body. This option sets the **UNROLL** directive. A loop with an iteration count can be unrolled completely, but a loop that contains other loops that aren’t unrolled should not be unrolled.

Typically, a loop with a small number of iterations and a small number of operations should be unrolled if the design is not meeting latency constraints. Partial unrolling should only be used if complete unrolling and pipelining cannot be used to achieve latency goals. Refer to “[Loop Unrolling](#)” on page 231 for more information.

- **Pipelining and Initiation Interval**

Enable the **Pipelining** field and set the **Initiation Interval** to specify the number of cycles to wait between each loop iteration. This option sets the **PIPELINE_INIT_INTERVAL** directive. After Catapult applies the loop unrolling constraints and several other transformations, the scheduler uses the pipelining constraints to build the pipelined loops. Refer to “[Loop Pipelining](#)” on page 239 for a detailed discussion about how pipelining is handled.

- **Generate Distributed Pipeline and Decoupling Stages**

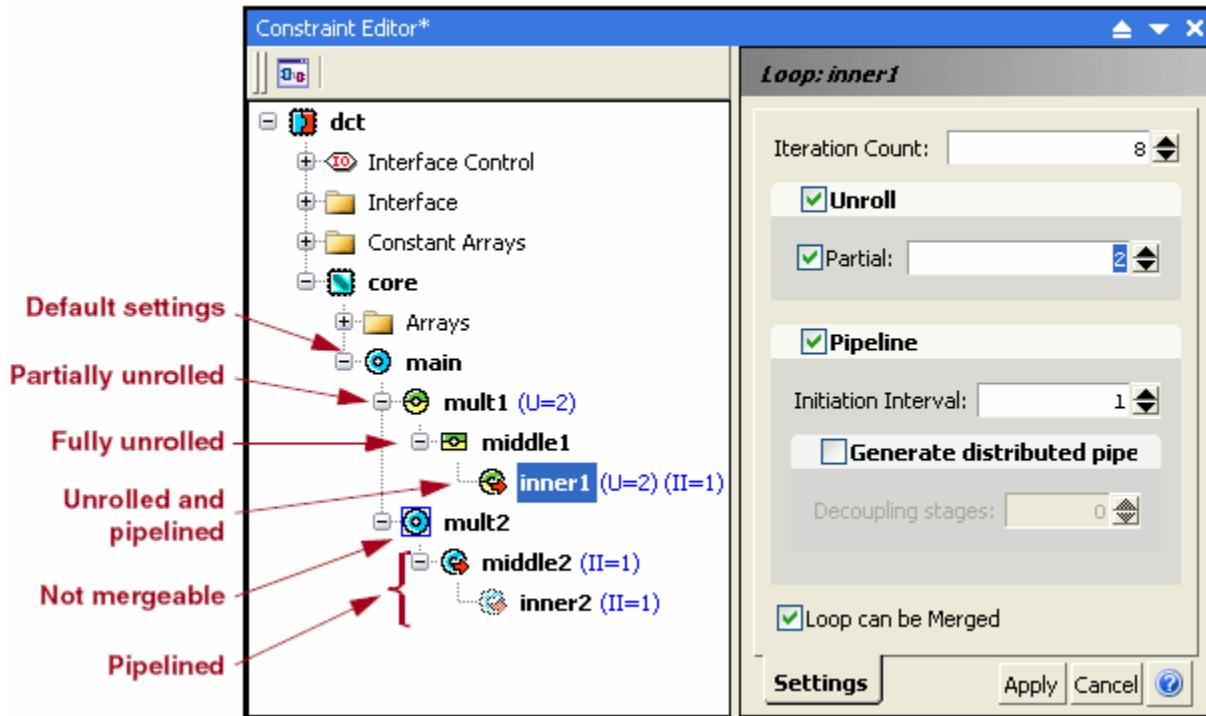
Enable the **Generate Distributed Pipeline** field and set the **Decoupling Stages** to have Catapult construct pipelined loops in such a way that they provide the benefits of pipeline “flushing” and “bubble compression”. For a detailed discussion about how distributed pipelines work, refer to “[Distributed Pipeline Synthesis](#)” on page 245.

These options set the **DISTRIBUTED_PIPELINING** and **DECOUPLING_STAGES** directives. The Decoupling Stages field specifies the maximum allowable number of back-to-back stages in the distributed pipeline. Zero means no limit.

- **Loop can be merged**

Allows loops that would normally execute in series to merged, if possible, in order to achieve parallel execution of the loops. This option sets the **MERGEABLE** directive. Refer to “[Loop Merging](#)” on page 234 for more information.

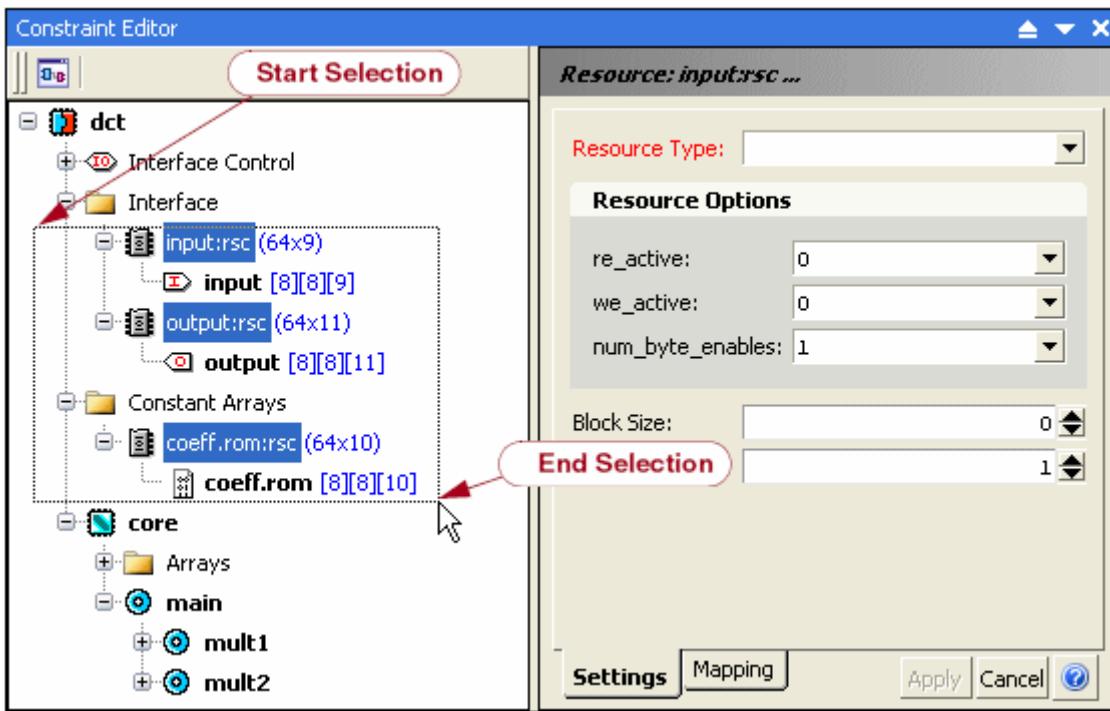
Figure 3-50. Editing Loop Constraints



Multiple Selection Capability

The Constraint Editor window allows you to select and apply constraints to a group of items at the same time. Multiple selection is limited to items of the same type (processes, resources, variables or loops). The easiest way to select a group of items is by using the mouse to select an area in the Constraint Editor, as illustrated in Figure 3-51. Notice that only arrays are selected because the first item selected was an array.

Figure 3-51. Multiple Selection in Architectural Constraints Window



Another way to select a group of adjacent items is as follows: First select an item at one end of the group by clicking on it with the left mouse button (LMB). Then move the cursor to the other end of the group and hold down the Shift key while clicking the LMB. To add/remove individual items to/from the selected set, put the cursor on the item and hold down the Control key while clicking the LMB.

The behavior of input fields (types are identified in Figure 3-52) is different when multiple items are selected. Drop-down lists and integer increment/decrement fields behave as follows:

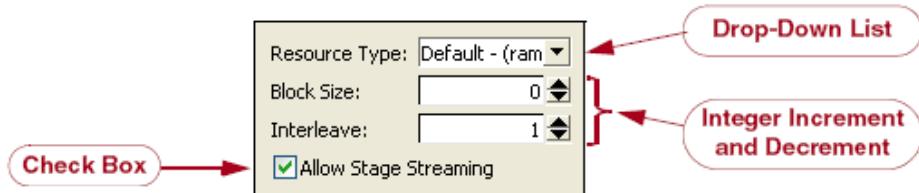
- If all selected items have the same setting, that setting is displayed in the field.
- If one or more selected item has a different setting, the field is blank.
- If you enter a new setting in the field, that setting will be applied to all selected items.

Check boxes behave as follows:

- If the check box is enabled for all selected items, the box is checked for the group.
- If the check box is disabled for all selected items, the box is unchecked for the group.
- If some are enabled and others disabled, a square appears in the check box.
- To change the setting, click on the check box. Each time you click, it increments through the three states: enable all (check mark), disable all (empty), keep each as is (square).

Changes you make to the settings are displayed in the Constraint Editor window, but are not applied to the design until you click the Apply button. The Cancel button will discard the changes.

Figure 3-52. Types of Input Fields on Architectural Constraints Dialog Box

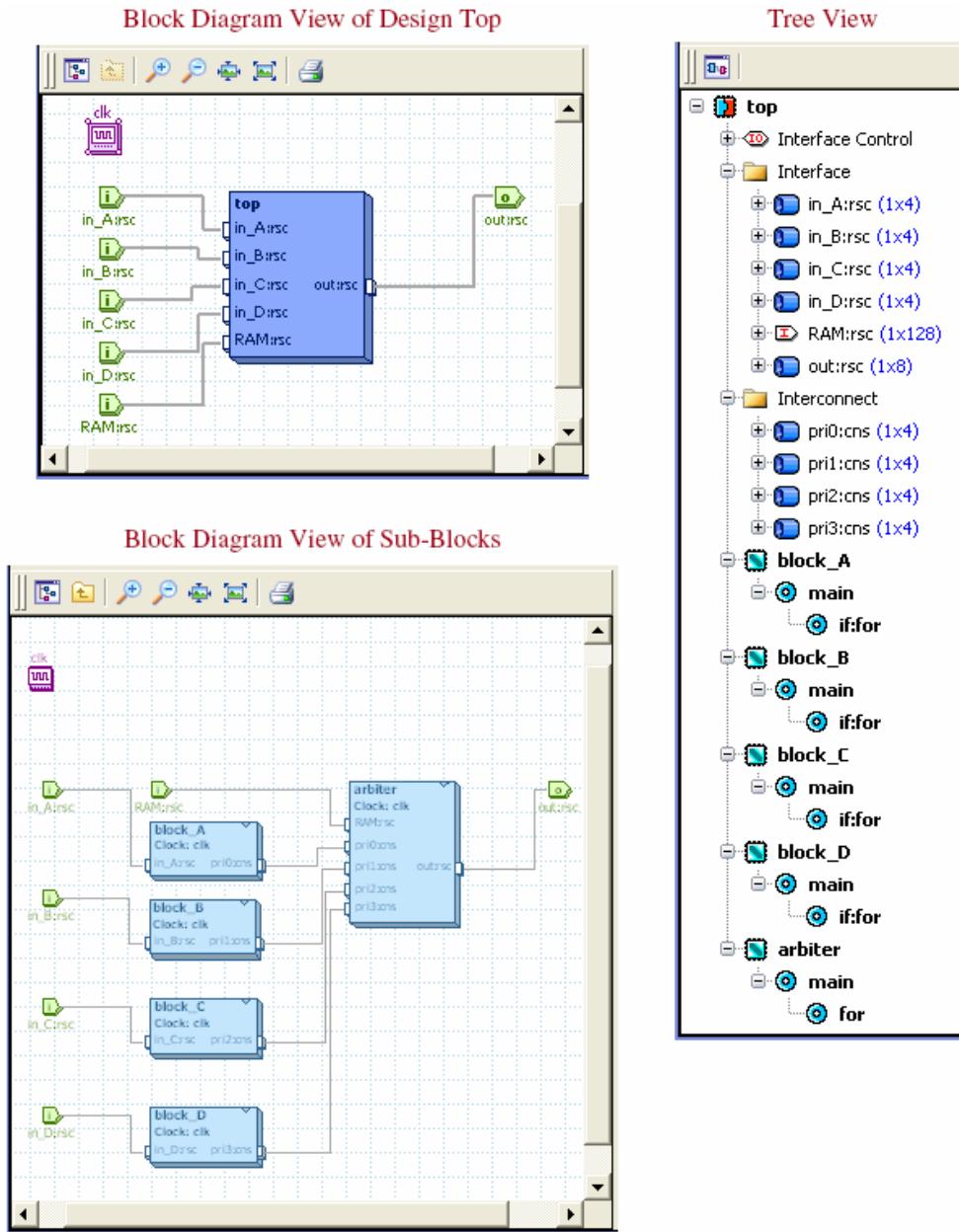


Block Diagram Constraint Editor

The graphical user interface for editing architectural constraints includes block diagram views of the design in addition to the traditional tree view. The interactive block diagram shows all of the blocks in the design and the interconnections between them. The primary block diagram view shows the contents of the top-level block. From there you can switch to a view of the top-level block showing the all of the I/O ports of the design. Figure 3-53 shows all three views of the same design.

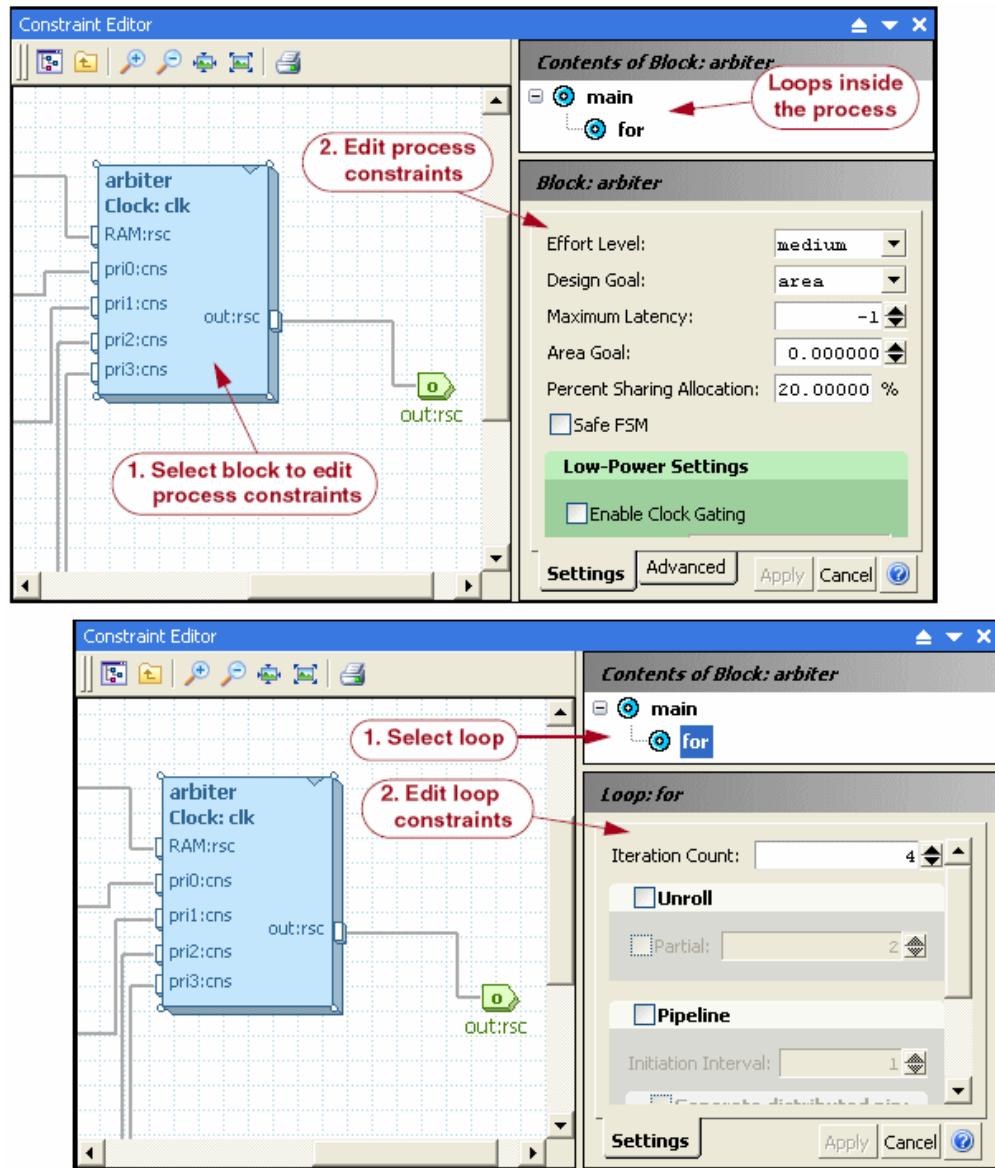
The Architectural Constraints editor displays only one view at a time. The tree view is the factory default view. To switch to a block diagram view, click the “Show Block Diagram” button in the toolbar above the tree view. Refer to [Figure 3-55](#) on page 127 for information about toolbar buttons. For instructions about how change the default view, refer to [“Constraints Editor Options”](#) on page 195.

Figure 3-53. Block Diagram Views and Tree View in Constraint Editor

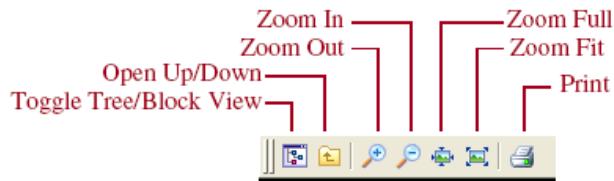


Just as with the traditional tree view, selecting an item in the block diagram displays the constraint options for that object on the right side of the Constraint Editor window. In addition, the objects contained in the selected object are shown in tree-view format in the top-right portion of the window. For example, when a process block is selected, as shown in Figure 3-54, the process constraints are displayed on the lower right, and the process loops are displayed at the top. When a resource is selected, the resource variable is displayed in the top. Select a loop (or resource variable) to access its constraint options.

Figure 3-54. Accessing Constraint Options in the Block Diagram



To switch between the top level view and the sub-block view, click on the “Open Up/Down” button on the tool bar. Refer to Figure 3-55. Double-clicking on the top block will also open the sub-block view.

Figure 3-55. Block Diagram Tool Bar Buttons

When an item is selected, information about the item is displayed in the Details window. Multiple items can be selected by holding down the Shift key while dragging the mouse over an area of the diagram. Individual items can be added to the selected set by holding down the Control key while clicking the object. Cross-probe an object by double-clicking it.

Scheduling the Design

When you click **Schedule** on the taskbar, Catapult schedules the in-memory design and displays the resulting data in the Schedule window. The Schedule window provides a graphical view of the design data and architecture for analysis.

The following topics describe how to use the Schedule window to analyze the design data:

- “[About the Schedule Window](#)” on page 127
- “[Interpreting Data in the Gantt Chart](#)” on page 131
- “[Exploring the Design](#)” on page 134
- “[Specifying Cycle Constraints](#)” on page 139
- “[Interpreting Results When Scheduling Fails](#)” on page 139

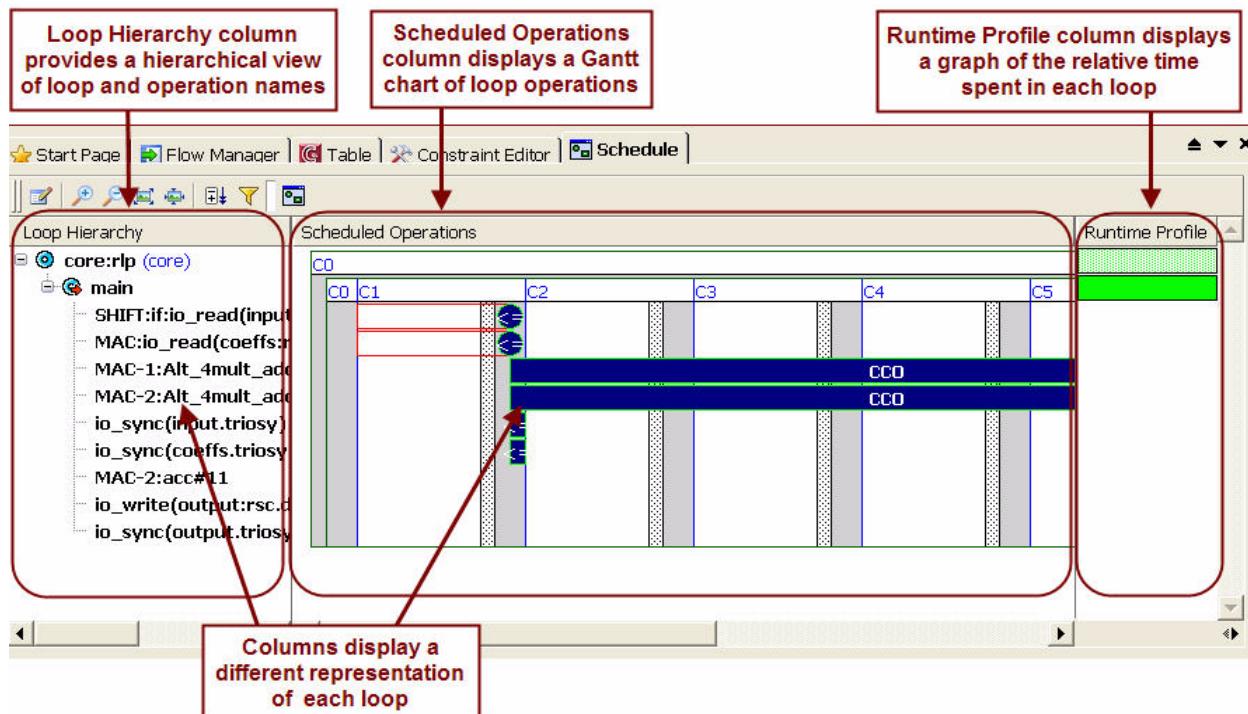
Note

 The number of operations in the design is reported in the Catapult transcript. 2000 operations is a sign of a complex design. Designs with 10,000 or more operations should be partitioned for synthesis.

About the Schedule Window

The Schedule window, shown in [Figure 3-56](#), is the main tool for analyzing the algorithm. It is like a schematic viewer for the algorithm. The Schedule window graphically shows the hierarchical nesting of loops, relative runtimes of each loop, types and sequence of scheduled operations, and data paths and dependencies between operations. It also textually shows the names of all loops, operations and components, area, delay and other database information about individual items.

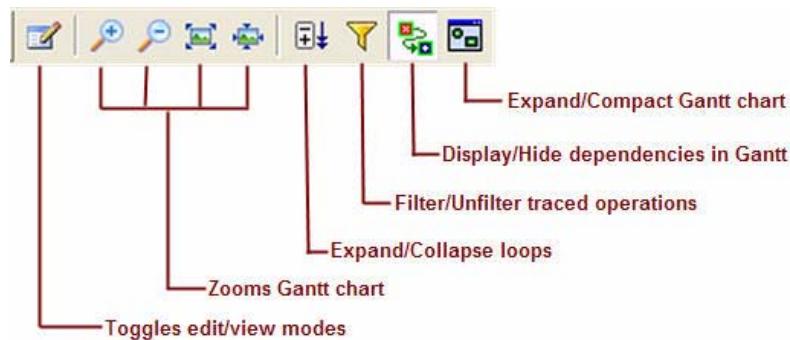
Figure 3-56. Schedule Window



- **Loop Hierarchy Column** — provides a nested list of all loops and operations in the design. Clicking the + icon next to a loop name expands the loop and displays all of the operations within it. The same loop simultaneously expands in the Scheduled Operations column. By default, the operation names display. To display component names instead, right-click and choose **View sorted > by Component Utilization**. See “[Sorting and Filtering Data](#)” on page 135.
- **Scheduled Operations Column** — displays a Gantt chart of the loop operations in the scheduled sequence. For more information, see “[Interpreting Data in the Gantt Chart](#)” on page 131.
- **Runtime Profile Column** — displays the loop runtime. Light green indicates that a loop contains sub-loops. The size of the light green bar is the total runtime of the loop, including sub-loops. The dark green is the runtime of the selected loop. When added together, the dark green sections of the bars represent 100% of the latency of the design.

Toolbar

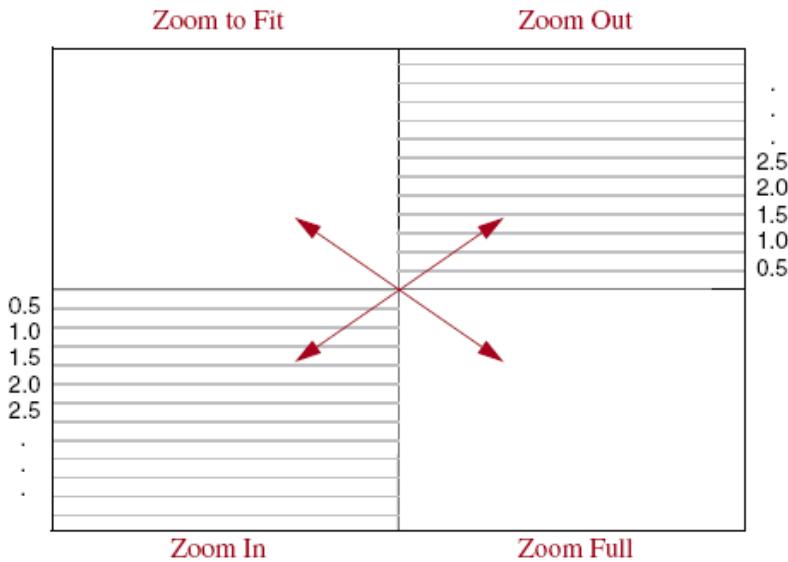
The Schedule window toolbar operates on data in the Loop Hierarchy and/or Scheduled Operations columns as described in [Figure 3-57](#)

Figure 3-57. Schedule Window Toolbar

Mouse Strokes

You can also use mouse strokes in the Scheduled Operations column to perform zoom operations on the Gantt chart. Begin a stroke by pressing and holding the left mouse button while dragging the mouse. The direction of the stroke determines the zoom command applied. A line appears from the point of origin to the current position of the cursor, and the zoom function displays at the cursor position. Complete the stroke by releasing the left mouse button.

Figure 3-58 shows the stroke command mapping. Strokes to the upper left quadrant zoom the Gantt chart to fit the width of the column. To the lower right quadrant zooms to the default, full-size view regardless of the column width. The upper right or lower left zooms out or in respectively. The zoom factor is determined by the vertical distance from the point of origin.

Figure 3-58. Zoom Command Strokes

Query Information

Query information refers to the information in the Catapult design database. Place the cursor on any data object in the Schedule window to display a text box containing the query information for that object. To copy and save the query information, right-click on a data object and choose **Copy Query Info to Clipboard**. To disable the display of query information, right-click anywhere in the Schedule window and choose **Query Mode**.

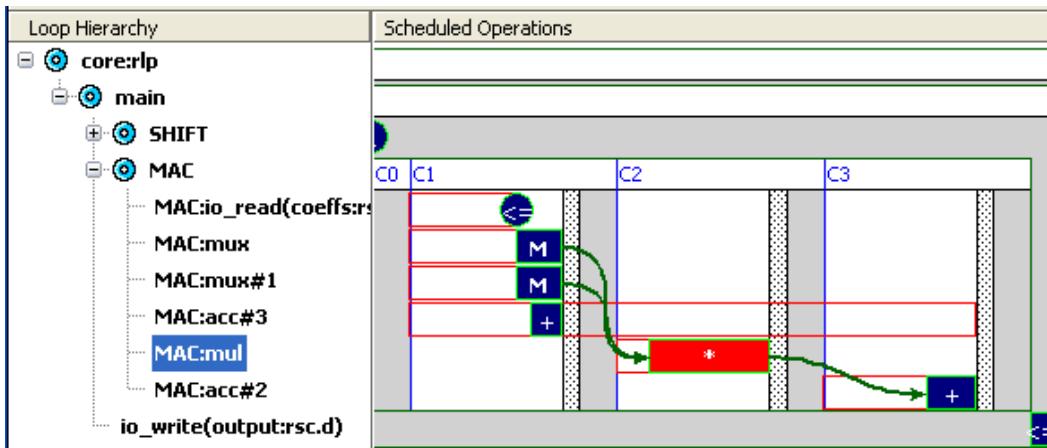
Data Dependencies

Selecting a data object in the Schedule window highlights the object in all columns and displays arrows in the Gantt chart to show dependencies between the selected data object and other operations. Different colored arrows indicate different types of dependency paths as described in [Table 3-4](#)

Table 3-4. Data Dependency Path Color Key

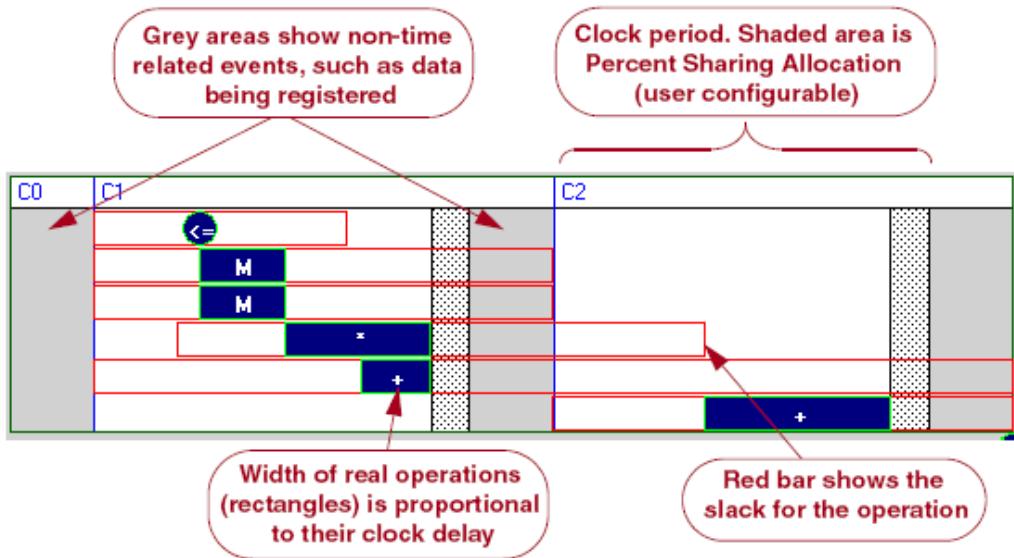
Arrow Color	Data Dependency Type
maroon/burgundy	user
blue	io mode
black	control
green	data
orange	feedback
red	failed (dependency arrows also turn red when the cursor is placed over them)

By default, all types of dependencies display. To disable the display of any or all dependency types, right-click anywhere in the Schedule window and choose **Dependencies**. (See “[Sorting and Filtering Data](#)” on page 135). To configure the default visibility and color of each type of dependency, refer to “[Schedule Viewer Options](#)” on page 196. [Figure 3-59](#) shows the predecessor and successor data dependencies (green arrows) for the MAC:mul operation.

Figure 3-59. Gantt Chart Data Dependencies


Interpreting Data in the Gantt Chart

The Gantt chart graphs the number of control steps (C-steps) in each loop and the sequence of the operations scheduled within the C-steps. Operations are shown in blue in a box proportional to the operation delay. [Figure 3-60](#) describes the main features of the C-step diagram.

Figure 3-60. Gantt Chart C-Steps


C-steps (C_0, C_1, \dots, C_n) are roughly equivalent to states in a finite state machine (FSM). If the design has complex conditional statements, several FSM states may map to the same C-step. Each C-step represents one clock cycle and has three sections, white, shaded, and grey.

The white and shaded areas comprise the actual clock period. The shaded area is the percentage of the clock period held in reserve for logic needing to share components and ports. See “Percent Sharing Allocation” on page 274.

The grey area indicates events that have no effect on timing. For example, memory read/write operations via pipes or wires. A red box extends into the grey area when data is registered between the C-steps. For example, pseudo-operations such as assignments.

The red box around each operation represents the slack for that operation in the algorithm. This is useful for finding the most critical operations in the algorithm. For example, the operations in Figure 3-60 have the following slack:

Table 3-5. Interpreting Slack in the Gantt Chart

Operation	Slack
Muxes (M)	Both can be scheduled anywhere in C1
Multiplier (*)	Can be scheduled in either the latter part of C1 or first part of C2.
Accumulators (+)	First one can be scheduled anywhere in C1 or C2. Second one can be scheduled anywhere in C2

C-steps that display in a lighter blue indicate they are affected by a cycle constraint. Place your cursor on the C-step to display information about the cycle constraint. You can modify the constraint from the Schedule window. For more information, see “Sorting and Filtering Data” on page 135.

About Compact Mode

Compact mode reduces the overall size of the Gantt chart by hiding non-essential information and presenting the chart data in a more efficient format. The compact mode displays all loops in the algorithm and hides the Loop Hierarchy and Runtime Profile columns. Sub-loops are nested inside their parent loops and can be expanded or collapsed independently of the parent.

In compact mode, sequential loops are positioned next to each other from left to right. Similarly, dependent operations are also aligned horizontally from left to right. If an operator is dependent on more than one upstream operator, it is aligned with the last of its upstream dependencies. Figures 3-61 and 3-62 show the same design in both modes to illustrate how loops and operations are aligned differently in the default display versus compact mode.

To enable compact mode, right-click in the Schedule window and select **Compact Mode**. To set compact mode as the default, see “Schedule Viewer Options” on page 196.

Figure 3-61. Gantt Chart Default Display

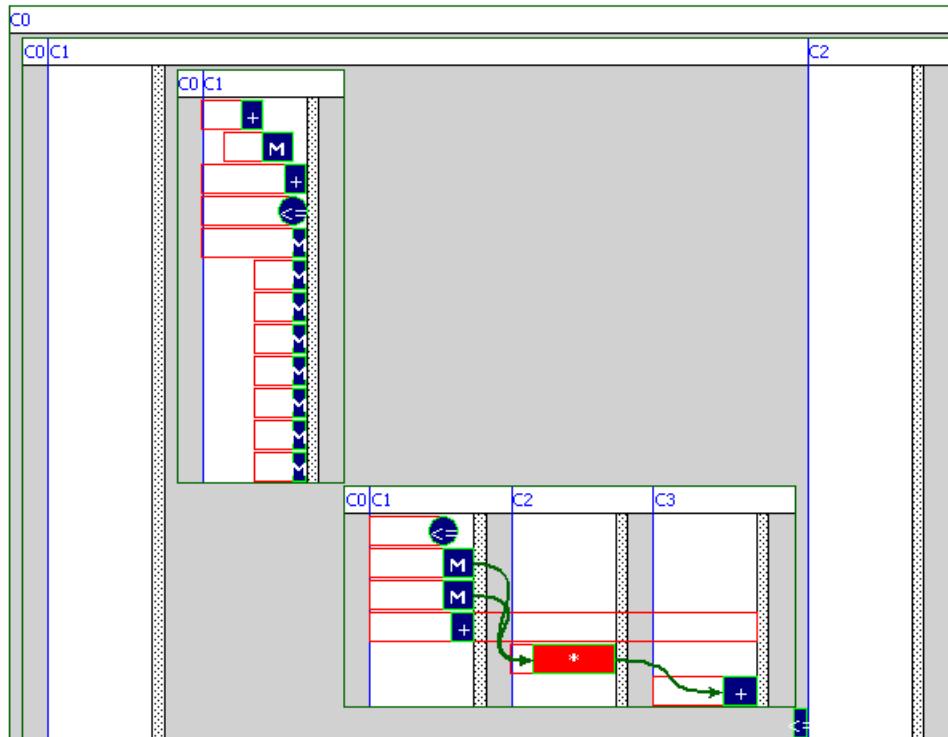
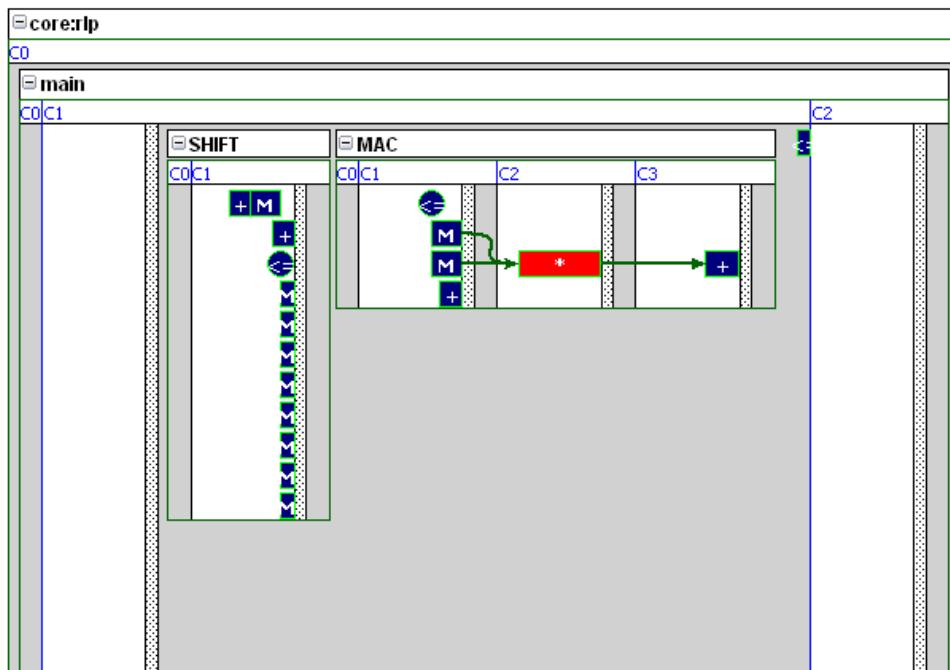


Figure 3-62. Gantt Chart Compact Mode



Exploring the Design

There are several ways to sort, filter and explore data once your design algorithm is scheduled. Options apply to operations/components within each loop but do not effect the ordering of the loops. By default, the Schedule window displays loops, C-steps, real operations, and data dependencies. Depending on your analysis needs, use the following methods to analyze your design data:

- “[Tracing Operations](#)” on page 134
- “[Sorting and Filtering Data](#)” on page 135
- “[Cross-Probing Algorithm Data](#)” on page 139

Tracing Operations

You can trace selected operations through multiple levels of logic in the algorithm data as follows:

1. Select the operation(s) to trace in either the Loop Hierarchy or the Scheduled Operations columns.
2. Right-click in the Schedule window, select **Forward Trace** or **Backward Trace** and one of the options described in [Table 3-6](#).
3. Click  on the toolbar to restore the default display.

Table 3-6. Backward/Forward Tracing Options

Popup Menu Option	Sub Option	Description
Backward Trace Forward Trace	Level 1 Level 2 Level 3 Level 4 Level 5	Highlights/displays the chain dependencies for the selected operation(s) through the specified logic level and hides all other operations.
Backward Trace	From Inputs	Highlights/displays the chain dependencies from the beginning of the schedule to the selected operation and hides all other operations.
	From C-Step	Highlights/displays the chain dependencies from the beginning of the C-step to the selected operation and hides all other operations.

Table 3-6. Backward/Forward Tracing Options (cont.)

Forward Trace	To Outputs	Highlights/displays the chain dependencies from the selected operation through the end of the schedule and hides all other operations.
	To C-Step	Highlights/displays the chain dependencies from the selected operation to the end of the C-step and hides all other operations.

Sorting and Filtering Data

Depending on your needs, you can use either of the following methods to sort and filter data in the Schedule window.

Note

 These filter/sort options cannot be directly applied to the Gantt chart in compact mode. Apply them in default display mode and then switch to compact mode to display the sorted data.

Method 1

This option only works when the window contents are not already filtered.

1. If necessary, click  on the toolbar to remove any filters and restore the default display.
2. Select one or more operations in either the Loop Hierarchy or Scheduled Operations columns.
3. Click  to hide all unselected objects from the display.
4. Click  to restore the default display.

Method 2

This method sorts/filters all data of a specified type in the Schedule window.

1. Right-click in the Schedule window and select the desired sort/filter options as described [Table 3-7](#). The data immediately displays sorted/filtered as specified.

2. Click  on the toolbar to restore the default display.

Table 3-7. Sorting and Filtering Options in the Schedule Window

Popup Menu Option	Sub-Menu Option	Description
View sorted	by C-Step	Displays operations in the scheduled order as shown in Figure 3-63 . Default.
	by Component Utilization (N/A in compact mode)	Displays components instead of operations. Components are listed alphabetically in the Loop Hierarchy column, and the Scheduled Operations column shows C-steps that use each component as shown in Figure 3-64 .
	by Type (N/A in compact mode)	Displays operation types and names in the Loop Hierarchy column, sorted alphabetically by type. Refer to Figure 3-65 .
View filtered	Show All Ops	Displays all “real” and “pseudo” operations. Refer to the example in Figure 3-65 . Real operations are those that effect delay and area, and are shown as blue rectangles in the Scheduled Operations column. Pseudo operations have no delay or area impact and are shown as blue circles.
	Show Real Ops	Displays only real operations. Default.
	Show Pseudo Ops	Displays only pseudo operations.
	Show Loops Only (N/A in compact mode)	Displays only loops. Hides all operations.
Dependencies	Successors	Enables/disables visibility of dependency arrows flowing into the selected item.
	Predecessors	Enables/disables visibility of dependency arrows flowing out from the selected item.
	User	Enables/disables visibility of <i>user</i> type dependency arrows. User dependencies only exist if cycle constraints are set.
	IO Mode	Enables/disables visibility of <i>IO mode</i> type dependency arrows.
	Control	Enables/disables visibility of <i>control</i> type dependency arrows.
	Data	Enables/disables visibility of <i>data</i> type dependency arrows.

Table 3-7. Sorting and Filtering Options in the Schedule Window

Popup Menu Option	Sub-Menu Option	Description
Show Paths	All	Displays all data paths. Default.
	Non-feedback	Displays only non-feedback data paths.
	Feedback	Displays only feedback data paths.

Figure 3-63. Schedule Window Data Sorted by C-Steps

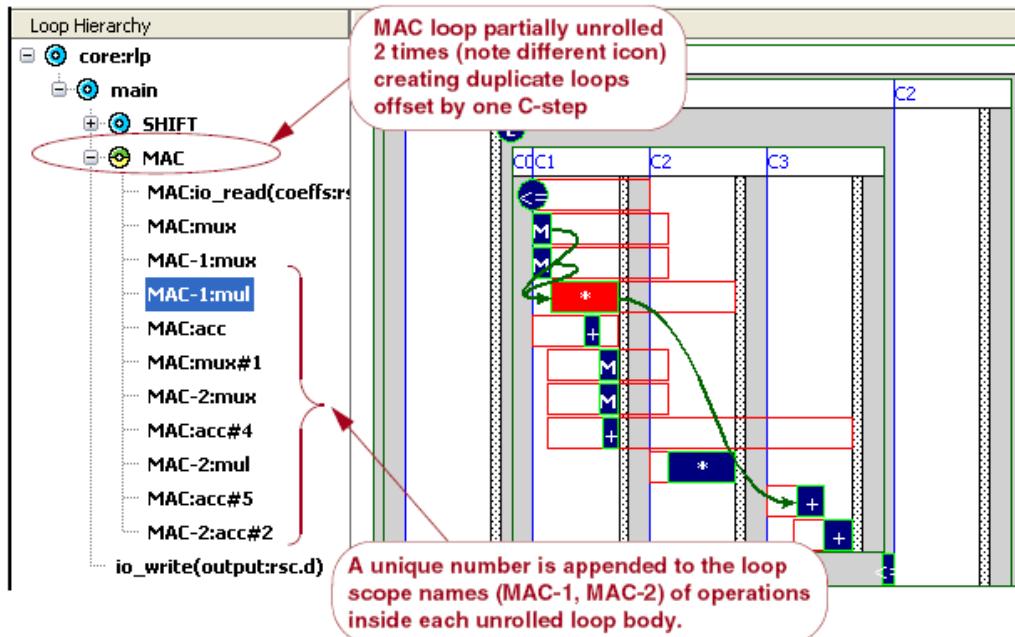


Figure 3-64. Schedule Window Data Sorted by Component Utilization

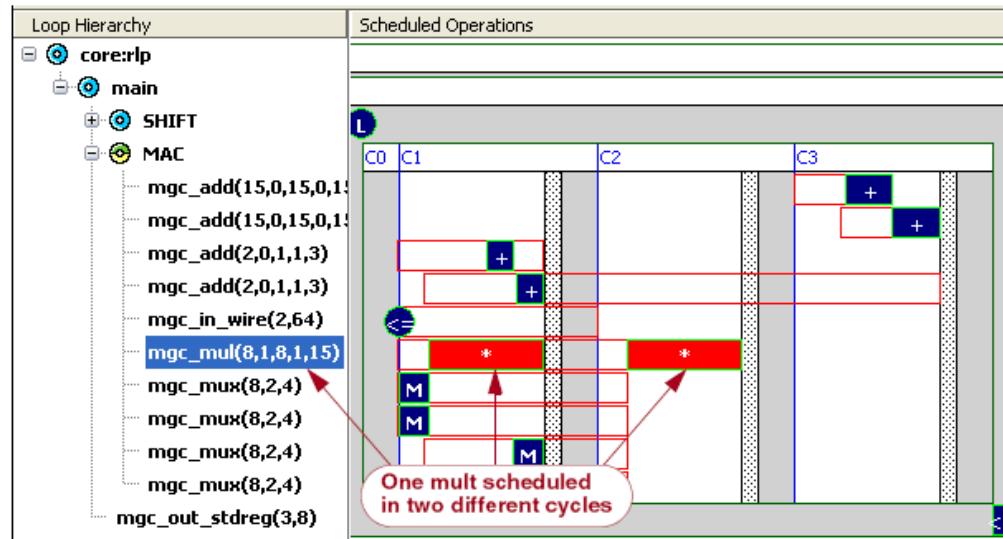
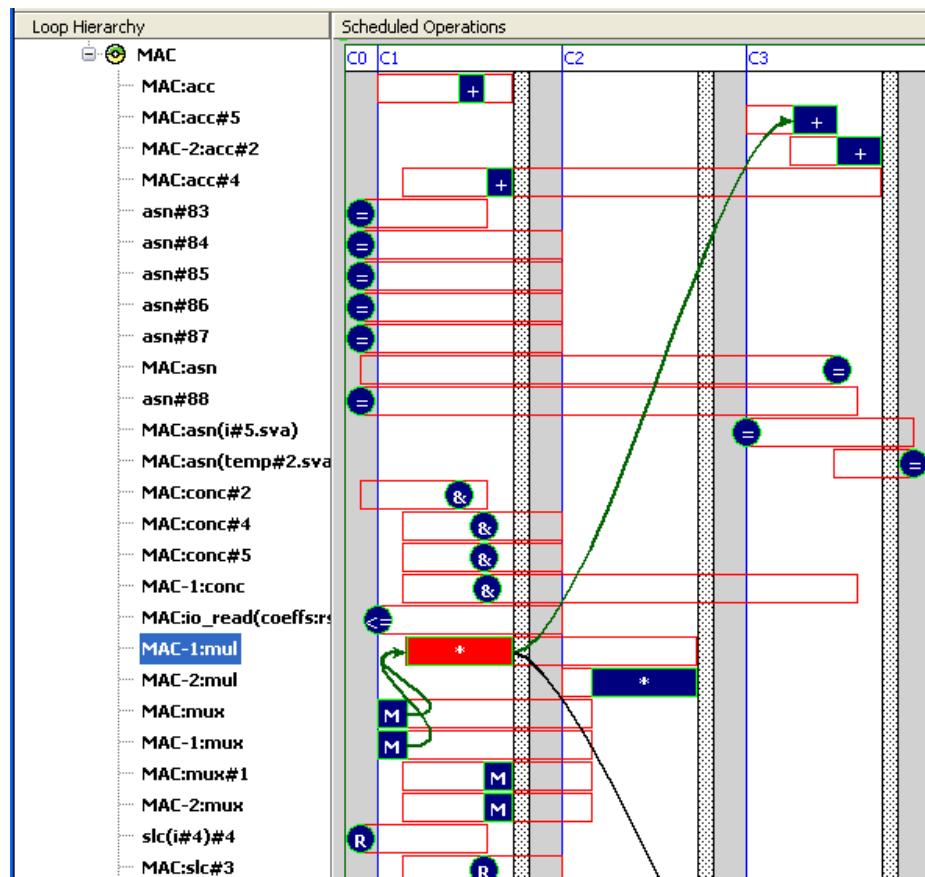


Figure 3-65. Schedule Window Data Sorted by Type, All Operations



Cross-Probing Algorithm Data

You can cross-probe data from the source code or the architectural constraints to the Schedule window as follows:

- **Crossprobing from the source window** — Highlight the desired code, right-click, and choose **View in Schedule**. The Schedule window displays the Gantt chart with the associated operation highlighted.
- **Crossprobing from the Constraints Editor window** — Select a loop from the tree or block diagram view, right-click and choose **View in Schedule**. The Schedule window opens and displays the selected loop highlighted.

Specifying Cycle Constraints

You can select operations in the Schedule window to apply cycle constraints.

1. Click  on the tool bar to enable Edit mode, and use one of the methods described in [Table 3-8](#) to specify cycle constraints

Table 3-8. Setting Cycle Constraints

Method 1	<ol style="list-style-type: none"> 1. Select one or two operations in the Schedule window. 2. Right-click and select Set Cycle Constraint. The Set Cycle dialog box displays with the selected operation name(s) populated. 3. Select a Type, enter Value, and click Ok.
Method 2	<ol style="list-style-type: none"> 1. Drag and drop an operation(s) from one C-step to another C-step. The previous location of the operation displays shaded.

2. Click **Apply**. The cycle constraints are applied, the algorithm is rescheduled, and a new Schedule window opens and displays the revised algorithm.

Interpreting Results When Scheduling Fails

If scheduling fails due to timing violations in pipelined designs with feedback loops, negative slack and failed paths display highlighted in the Gantt chart.

If timing violations occur in combinational logic, the combinational path that cannot be scheduled displays highlighted in the Gantt chart as shown in Figure 3-66.

- The clock overhead column in the C-Step where the violation occurs is resized to accommodate the failed operation, and it is highlighted in red to indicate that the operation did not meet the timing constraints.
- The problem operations can extend beyond the boundary of the C-Step column.
- The names of failed operations are highlighted in the Loop Hierarchy column.

- The Failed Paths field in the tool bar contains a drop-down list of failed paths (maximum of ten). Selecting a path highlights all of the operations in that path as shown in Figure 3-67.

Figure 3-66. Gantt Chart with Clock Overhead Column Highlighted

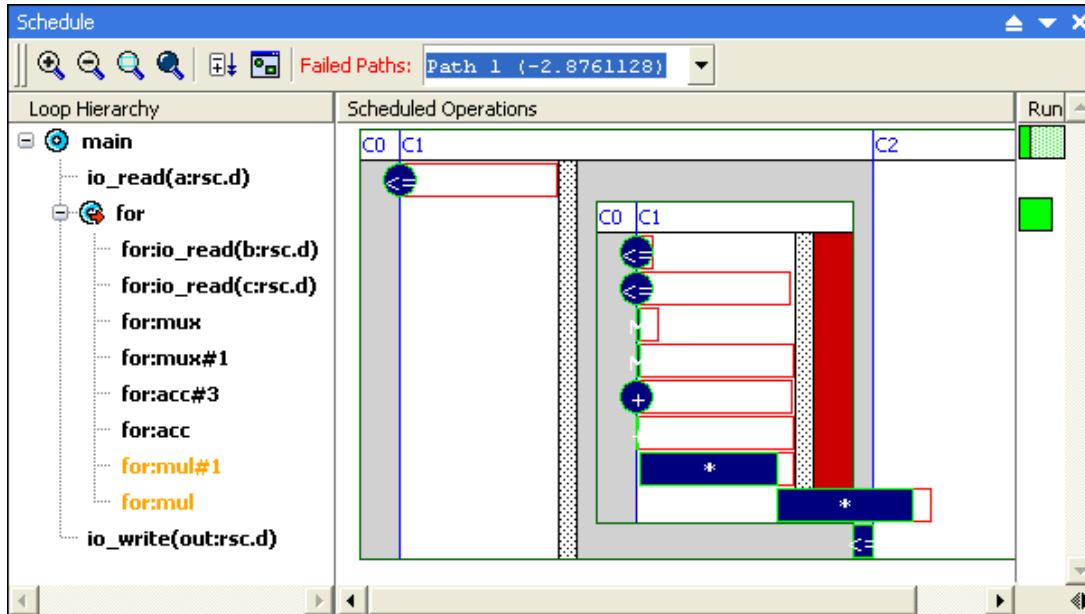
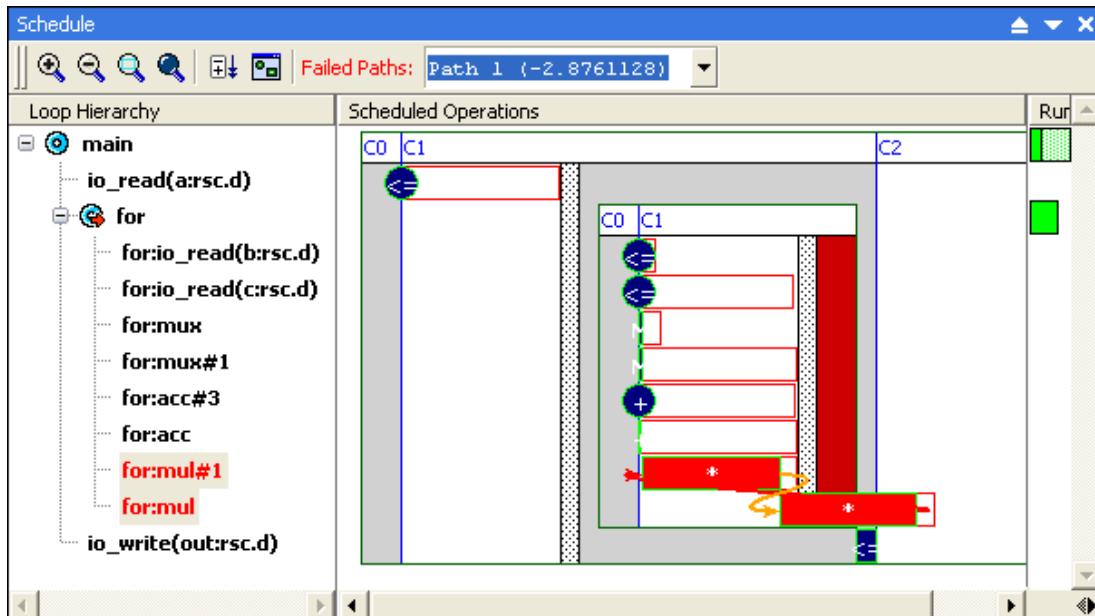


Figure 3-67. Gantt Chart with Failed Paths Highlighted



If timing violations occur in sequential components, the scheduler stops, no Gantt chart is generated, and a list of sequential operations display in the transcript to show the path that cannot be scheduled. The accumulated sequential delay of these operations violates the

pipelining initiation interval. To resolve the issue, investigate the chain of operations or relax cycle constraints so that scheduling can pass.

Example transcript messages:

```
# Error: Schedule failed, sequential delay violated. List of sequential
         operations and dependencies: (SCHD-20)
# Error: pipeline.cpp(5):    IOWRITE "LOOP0:io_write(a:rsc.d)"
         pipeline.cpp(5,9,1) (BASIC-25)
# Error: pipeline.cpp(5):    IOREAD "LOOP0:io_read(a:rsc.d)"
         pipeline.cpp(5,9,1) (BASIC-25)
# Error: Feedback path is too long to schedule design with current
         pipeline and clock constraints. (SCHD-3)
# Error: Design 'fn' could not schedule partition '/fn/core' - could not
         schedule even with unlimited resources
```

Generating the Output Files

Click on the **Generate RTL** task (or enter the “go extract” command) to generate all output files (except for the Gantt chart which is generated by the **Schedule** task). All output files are saved in the active solution directory in the file system. In the Project Files window the output files are organized hierarchically in folders, as shown in [Figure 3-42](#) on page 101. The organization reflects the basic work flow.

The Output Files folder contains generated data files used during the design process. These include text files such as reports and HDL files, and graphical data files such as schematics and the Gantt chart. You can see in [Figure 3-42](#) that the different types of output files are sorted into separate subfolders. Double-click on an output file to view its contents. Catapult will open the file in the appropriate viewing tool.

 **Note** You can configure the basenames of design files generated by Catapult. For more information, see [“Configurable Basenames for Generated Files”](#) on page 146.

Catapult Report Files

The Catapult C Synthesis tool generates two reports automatically for every solution.

- The Cycle report (*cycle.rpt*) is a high-level architecture/algorithm report including information about the clock(s), I/O ports, memory resources, loops, latency, and throughput.
- The RTL report (*rtl.rpt*) is similar to reports seen from RTL synthesis tools. The estimates for clock period and area are typically conservative and within about 20% of the area reported by RTL synthesis.

Use the cycle and RTL reports to check if the design has the correct latency, throughput, clock speed and area. It's unlikely that Catapult won't be able to meet timing, but it can happen in designs with moderate to heavy amounts of control logic or if the design constraints aren't correct.

Catapult HDL Output Files

For each target output language (VHDL, Verilog and SystemC), Catapult creates a separate folder and automatically generates two types of HDL output: *cycle* and *rtl*.

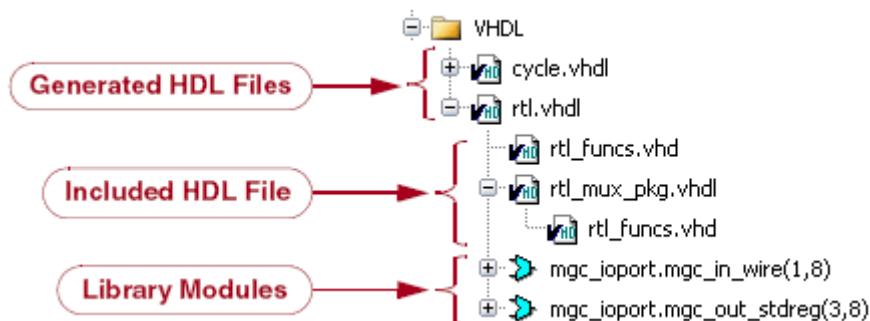
- The cycle output is designed to give the fastest simulation where the I/O of the design is cycle accurate (i.e. simulates exactly the same as the RTL). The main reason to use cycle output is for fast simulation (about 20-30 times faster than RTL). This type of output is also much closer to C++/SystemC format, so it might be easier to understand.

The files *cycle.v*, *cycle.vhdl* and/or *cycle.cxx* are automatically generated and put in the solution directory. Also included is a compile script of the same name and a *.msim* filename extension for the Mentor Graphics ModelSim tool. The *.msim* file can also be used to list the order of files to compile.

- The RTL output is the output for synthesis and RTL level simulation. The files *rtl.v*, *rtl.vhdl* and/or *rtl.cxx* are automatically generated and put in the solution directory. It includes a compile script of the same name and a “*.msim*” filename extension. A synthesis script is also automatically generated for VHDL and Verilog. RTL output supports std_logic_vector format.

HDL files are displayed as hierarchical objects, as shown in Figure 3-68. Click on the ‘+’ icon next to a file name to expand it and view its dependency files. The first sublevel shows included HDL files and library modules. Expanding a library module reveals its underlying HDL file.

Figure 3-68. HDL Output File Hierarchy in GUI



Memory Map Exported for IP-XACT

This IP-XACT flow allows the creation and use of an IP-XACT file to describe the memory/register map of the design. Generation of the IP-XACT file to describe interfaces is based on the constraints specified in the Architecture Constraints dialog. Constraints that are

captured in the IP-XACT description include RESOURCE, BASE_BIT, BASE_ADDR and MAP_TO_MODULE. Likewise, when an IP-XACT file is added to the input file list along with the design, the memory/register map in the IP-XACT description is turned into the Catapult constraints.

You can use [The Flow Manager Window](#) to enable/disable the flow, or use the following commands to enable and run the flow from the command line:

```
flow package require /IP_XACT
flow run /IP_XACT
```

When the flow is enabled, it automatically generates an IP-XACT file during the “schedule” stage of the work flow (“go schedule” command). The file is in XML format. The file name has the form <design_name>.mmr.xml.

To load an IP-XACT memory map file into a Catapult project, simply add it to the project’s input file list.

```
solution file add <path_to_file> -type XML
```

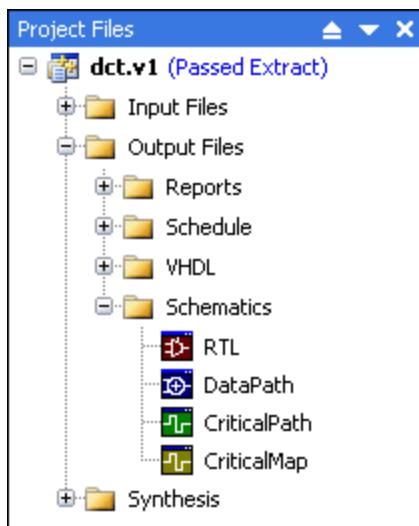
For Example:

```
solution file add ./hier_dct_proc_mmr.xml -type XML
```

Catapult Schematics

Catapult generates one RTL netlist file, schematic.nlv, that can be filtered to show only data paths or critical paths. When the netlist is generated four schematic icons appear in the Output Files > Schematics folder: **RTL**, **DataPath**, **CriticalPath** and **CriticalMap**, as shown in Figure 3-69. Double-clicking on one of these objects opens the schematic file in the schematic viewer and sets the appropriate *view state*. Refer to [“Using the Schematic Viewer Window”](#) on page 148.

Figure 3-69. Schematic Output Files



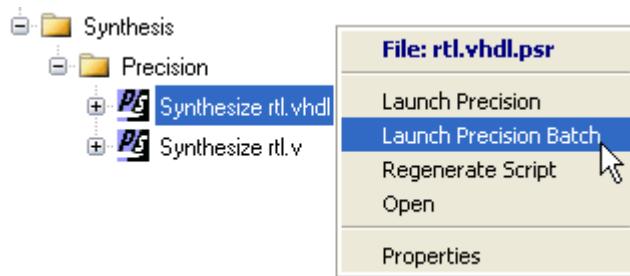
Catapult Generated Synthesis Scripts

Catapult interfaces to downstream synthesis tools by generating scripts that launch and drive the target synthesis tool. You select one of the supported synthesis tools, such as Precision RTL Synthesis or Design Compiler. See [“Setting Up the Design” on page 96](#). Double-clicking on a synthesis script launches the target tool.

Precision RTL Synthesis Script

The Precision RTL Synthesis script is placed in a subfolder named **Precision** and the script has a “.psr” filename extension. If you double-click on the **Synthesis rtl.<hdl>.psr** file, a Precision RTL session is started in GUI mode. Use the popup menu, shown in Figure 3-70, to launch the tool in batch mode or to open the script file in a text editor.

Figure 3-70. Launching Precision RTL Synthesis Scripts

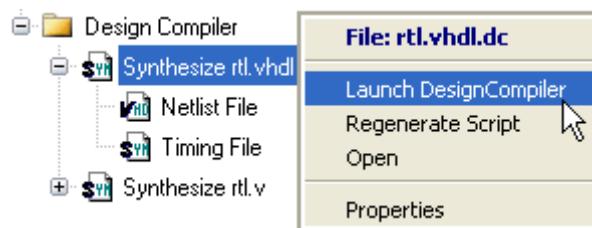


When Precision RTL is launched in GUI mode, the script loads the input files and sets up the design technology for you. When launched in batch mode, the script also compiles and synthesizes your design, then adds the session transcript file to the Precision folder.

Design Compiler Synthesis Scripts

The Design Compiler script is placed in a folder named **Design Compiler** and the script has a “.dc” filename extension. Click on the ‘+’ icon next to the script file to expand it and view its dependency files. These include input files for DC and the DC session transcript file. Either double-click on the script file or use the popup menu, shown in Figure 3-71, to launch the script.

Figure 3-71. Launching Design Compiler Synthesis Scripts



Customizing and Packaging Netlist Files

Catapult allows you to customize the certain aspects of the HDL in the generated netlist, such as stripping out pragmas, truncating variable names, and prefixing user-defined strings to sub-block names. The netlist filenames are also configurable. Refer to “[Configurable Basenames for Generated Files](#)” on page 146.

In addition, you can have Catapult gather all referenced netlist files together in the solution directory, making the netlist easily relocatable. All of these options are described in “[Output Options](#)” on page 189. The Concat flow, described below, is also used for packaging the netlist.

Concat Flow

When this flow runs, Catapult concatenates all netlist files for the design into a single HDL file. The output file includes Catapult-generated HDL and referenced IP files. Dependency files are added at the top of the output file prior to the files that depend on them. The output file is saved in the solution directory, and its name has the form:

```
concat_<basename>.<hdl_type>
```

where <basename> is the value of the Basename output option, and <hdl_type> is the either “h” or “vhdl.” SystemC output is not supported.

For example, if both Verilog and VHDL output is generated, and the default Basename setting is used, the two concatenated output files would be:

```
concat_rlt.v
concat_rlt.vhd
```

Use the following commands to enable and run the Concat flow:

```
flow package require /Concat
```

```
flow run /Concat/generate
```

To enable (disable) the flow from the Catapult GUI, select “Concat” in the Flow Manager window, then click the Enable (Disable) button.

Configurable Basenames for Generated Files

The basenames of design files generated by Catapult are configurable. Use the “Basename” option to change the default basenames of files generated during any particular stage of the Catapult work flow. Table 3-9 shows default basenames for the stages during which Catapult generates output files.

Table 3-9. Default Basenames

Stage	Default Basename	Output Files
allocate	schedule	Gantt chart
schedule	cycle	Cycle accurate netlist files
extract	rtl	RTL netlist files and schematic

To change basenames from the GUI, open the Catapult Options window (**Tools > Set Options...**) to access the “Output” options, and modify the “Netlist Basenames” field. Refer to “[Output Options](#)” on page 189 for more information. Alternatively, you can use the “[options set](#)” command.

Specify one or more stage-basename pairs. If a pair is omitted, the factory default basename for that stage will be used. The following command example changes the default basenames for the allocation, schedule and extract stages. Table 3-10 shows how the settings in the example command below will affect the generated file names.

```
options set Output Basename {allocate my_allocate_3 schedule my_cycle_3
extract my_rlt_3}
```

Table 3-10. Effect of Changing Factory Default Basenames

Factory Default Basenames	User Defined Basenames
directives.tcl	directives.tcl
messages.txt	messages.txt
cycle.rpt	my_cycle_3.rpt
cycle.vhdl	my_cycle_3.vhdl
cycle.vhdl.msim	my_cycle_3.vhdl.msim
cycle_funcs.vhd	my_cycle_3_funcs.vhd
cycle_mgc_ioport.vhd	my_cycle_3_mgc_ioport.vhd
cycle_mux_pkg.vhdl	my_cycle_3_mux_pkg.vhdl
rtl.rpt	my_rtl_3.rpt
rtl.vhdl	my_rtl_3.vhdl
rtl.vhdl.dc	my_rtl_3.vhdl.dc
rtl.vhdl.dc_timing	my_rtl_3.vhdl.dc_timing
rtl.vhdl.msim	my_rtl_3.vhdl.msim
rtl_funcs.vhd	my_rtl_3_funcs.vhd
rtl_mgc_ioport.vhd	my_rtl_3_mgc_ioport.vhd
rtl_mux_pkg.vhdl	my_rtl_3_mux_pkg.vhdl
schedule.gnt	my_allocate_3.gnt
schematic.nlv	my_rtl_3.nlv

Dynamically Insert Entity Name in Basename

The syntax for the "Basename" output option provides the macro “\${ENTITY}” that dynamically resolves to the name of the top-level entity in the design. Optionally use this macro to insert the entity name in the base filenames generated during netlisting:

```
 ${ENTITY}
```

For example, suppose the top-level entity name is hier_dct. The following setting combines the entity name with different literal strings to form the base filenames for the files generated at the “schedule” and “extract” stages.

```
options set Output Basename {schedule ${ENTITY}_cycle
extract ${ENTITY}_rtl}
```

The resulting base filenames would be hier_dct_cycle and hier_dct rtl. For comparison, below are two solution directory listings. The first contains modified basenames and the second shows the default basenames.

Solution directory with modified basenames:

```
concat_hier_dct_rtl.vhdl
directives.tcl
gate_dc.vhdl.vhdl
hier_dct_cycle.rpt
hier_dct_cycle.vhdl
hier_dct_cycle.vhdl.msim
hier_dct_cycle_funcs.vhd
```

```
hier_dct_cycle_mgc_ioport.vhd
hier_dct_cycle_mux_pkg.vhdl
hier_dct_rtl.rpt
hier_dct_rtl.vhdl
hier_dct_rtl.vhdl.dc
hier_dct_rtl.vhdl.dc_timing
hier_dct_rtl.vhdl.msim
hier_dct_rtl.vhdl_order.txt
hier_dct_rtl_funcs.vhd
hier_dct_rtl_hier_dct_rtlmgc_rom_23_64_10.dat
hier_dct_rtl_hier_dct_rtlmgc_rom_23_64_10.vhdl
hier_dct_rtl_mgc_ioport.vhd
hier_dct_rtl_mux_pkg.vhdl
messages.txt
schedule.gnt
schematic.nlv
```

Note



The \${ENTITY} variable will become the default setting for the operator flow to avoid naming conflicts in the file import flow.

Using the Schematic Viewer Window

The Schematic viewer is an interactive window that not only allows you to browse schematics and cross-probe design objects, but also visualize critical paths and query for data path timing information in order to perform timing correlation with downstream tools.

Double-clicking on a design object will cross-probe to the origin of that object in the source code. Alternatively, selecting the object and choosing the “**Cross-probe**” menu item on the popup menu will open the **Cross-Reference Pick** window with which you can choose the file to be cross-probed. Not all objects in the schematic can be cross-probed. Those that can be cross-probed are highlighted in blue. The default color is a bright blue, but is configurable. Refer to “[Schematic Viewer Options](#)” on page 197.

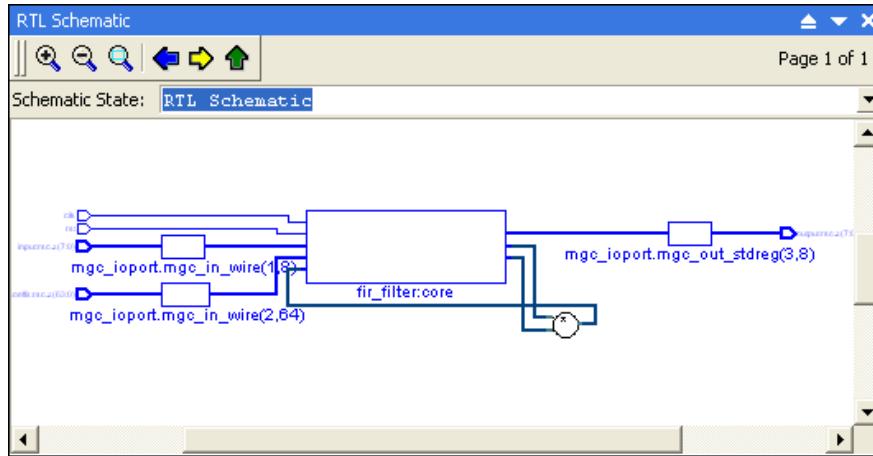
Hovering the cursor over an object in the schematic will display database information about the object, such its instance name, the qualified module name, delay and area data, and so on. The same information will be displayed in the **Details** window if the object is selected. Once selected, the information can be copied to the paste buffer by using the popup menu items “**Copy Query Info to Clipboard**” and “**Copy Name to Clipboard**.”

Schematic Hierarchy

The Schematic Viewer provides a hierarchical view of the generated RTL. The top-level view, such as the one shown in Figure 3-72, shows the top process, port interfaces and memories. Double-click a design block to view its internal logic. Lower-level blocks may contain FSM sub-blocks.

By default, large schematics are distributed across multiple *pages*. Use the Next/Previous Page commands to view other pages. To turn off multi-page format and view the entire schematic on a single page, choose the “**Multipage Schematics**” menu item on the popup menu.

Figure 3-72. Schematic Viewer Window

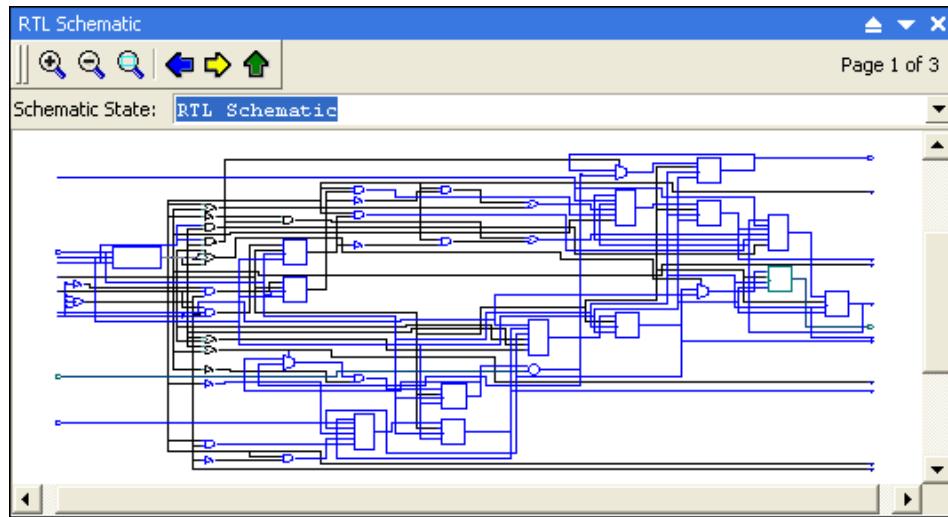


View States in the Schematic Viewer

The Catapult schematic viewer can display schematics in four different *view states*: RTL, Data Path, Critical Paths and Critical Map. Each state is a filtered view of the schematic, as described below. To switch states, use the Schematic State drop-down selector at the top of the window, or the **Schematic State** pop-up menu item.

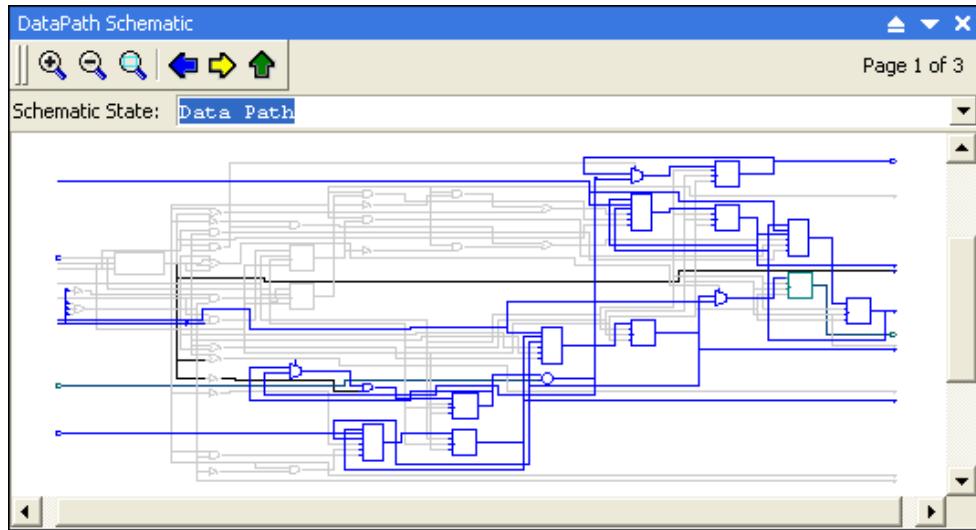
- The *RTL* view state is the default view. Different colors are used to indicate certain properties about the design objects. For information about the default color coding and how to change it, refer to “[Schematic Viewer Options](#)” on page 197.

Figure 3-73. Schematic View State “RTL Schematic”



- The *Data Path* view state highlights data path elements and dims all non-datapath elements in grey. This state shows you the architecture that was generated by the tool.

Figure 3-74. Schematic View State “Data Path”

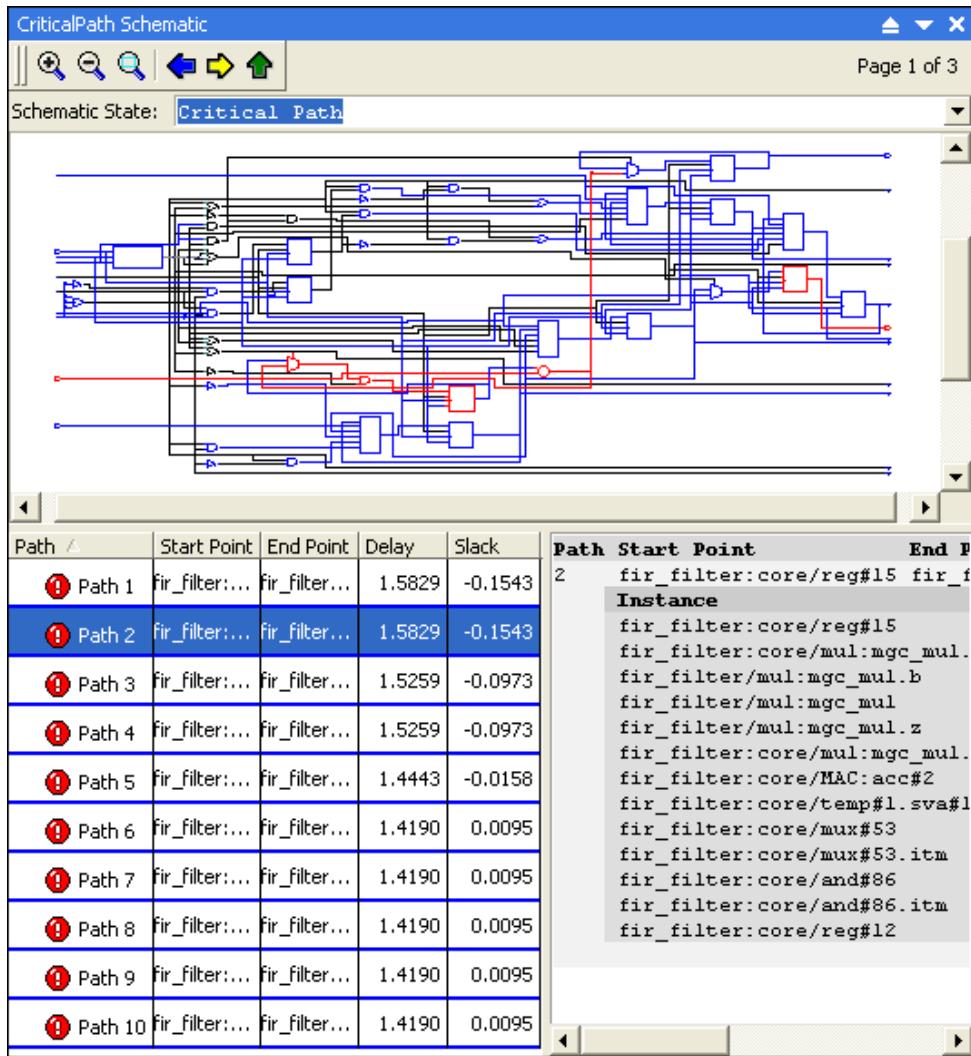


- The *CriticalPath* view state highlights in red user-selected critical paths. The window is divided into three sections. The top section contains the schematic, the lower left corner contains the Path Table, and the lower right corner contains the Path Quick View. The Path Table lists the ten most critical paths. Select a table row to highlight its path in the schematic. In the Path Quick View is detailed timing data about each instance in the selected path.



Tip: The same timing data can also be obtained in non-GUI mode by using the “solution timing” command.

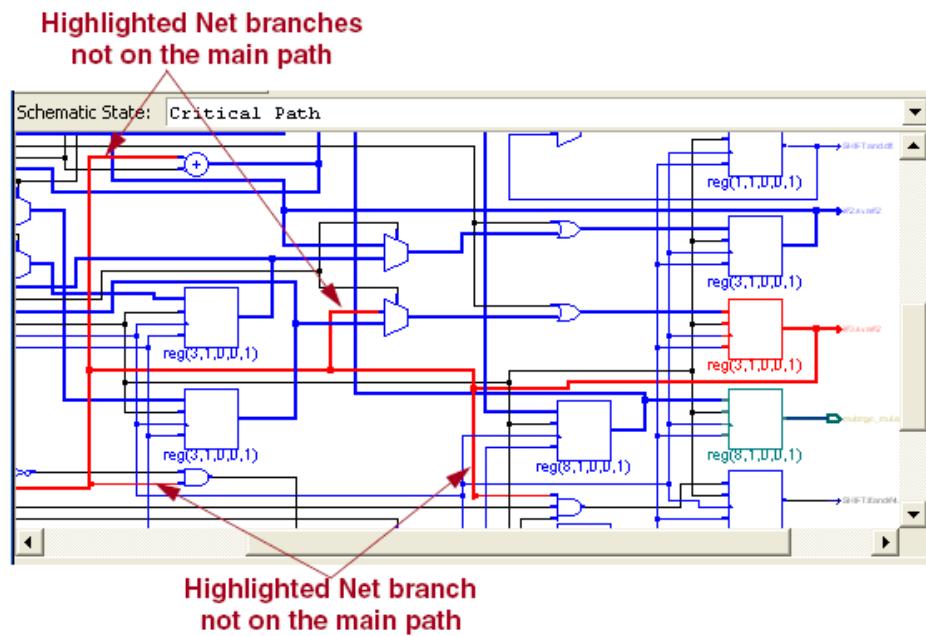
Figure 3-75. Schematic View State “Critical Path”



When one or more paths are selected in the table, all of the nets and instances in those paths are highlighted and the schematic view is automatically repositioned so that the starting instance of the most critical selected path is in the center of the window.

Note that for each net or interconnection in a critical path, all of the net branches (only the net and not the output operator) are highlighted also. Figure 3-76 shows an example of highlighted net branches.

Figure 3-76. Secondary Net Branches Highlighted



To change the number of critical paths listed in the table, right-click in the background area of the schematic and choose “Edit Number of Paths” from the popup menu. The setting applies only to the current window. The equivalent command line interface is “[view schematic critical -count <num>](#).”

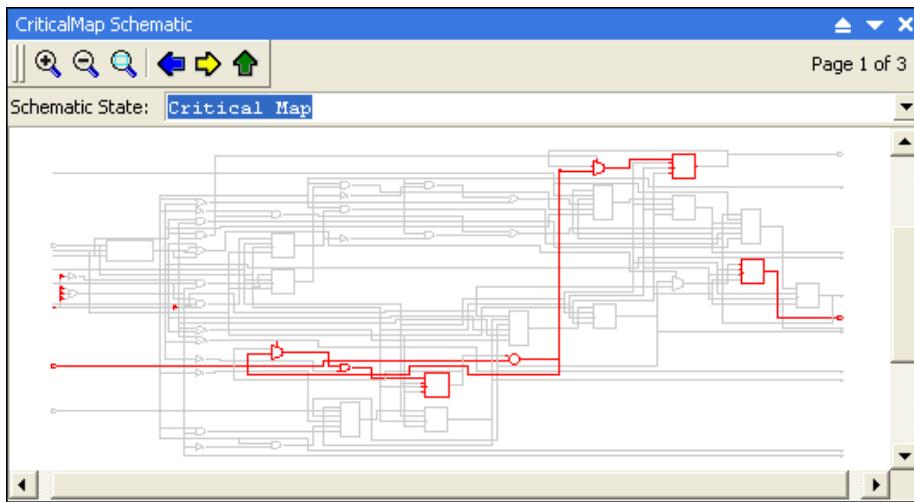
- The *Critical Map* view state highlights in red all paths having negative slack that exceeds a user-specified threshold value. Valid threshold values are integers greater than or equal to 100. The default threshold is 100 (100% of the clock period). That means by default all paths whose delay is greater than the clock period will be highlighted on the Critical Map schematic. Setting a higher threshold (greater negative slack) will filter out paths whose delay is less than the threshold. If no paths have negative slack, or if none is greater than the threshold, then no paths will be highlighted in the Critical Map schematic.

To set the threshold from within the schematic window, right-click in the background area of the schematic and choose “Edit Map Threshold” from the popup menu. The equivalent command line interface is “[view schematic criticalmap -map <num>](#).”



Tip: The same timing data can also be obtained in non-GUI mode by using the “[solution timing -map <num>](#)” command. The command returns a detailed report of all the critical paths that exceed the threshold value.

Figure 3-77. Schematic View State “Critical Map”



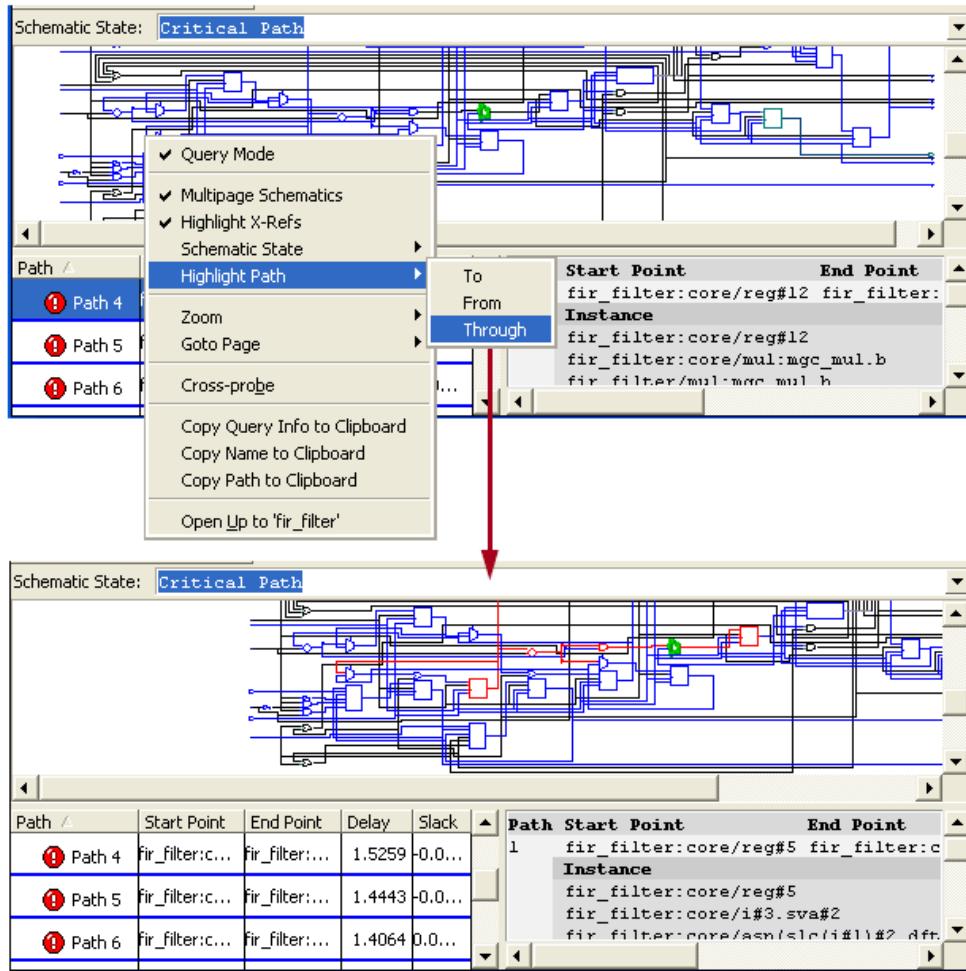
Querying for Point-to-Point Path Delay

Obtaining information about the path delay between specific objects in the design enables you to perform point-to-point timing comparison with downstream tools. Path delay reports are obtained interactively in the Critical Path view of the Schematic window, or from the “solution timing” command.

Reports generated by the Critical Path schematic are displayed in the Path Quick View area of the window. The reports list the individual delay for all instances in the path and the cumulative delay and slack of the path. In the schematic window, the specified path becomes highlighted.

To get a path delay report by using the Schematic window, first right-click on an instance in the schematic and select “Highlight Path” from the popup. Then choose “To,” “From” or “Through” on its submenu. Figure 3-78 illustrates the procedure. Note that the “Through” option is not applicable to registers. Registers are treated as path boundaries because they can have more than one path flowing through them.

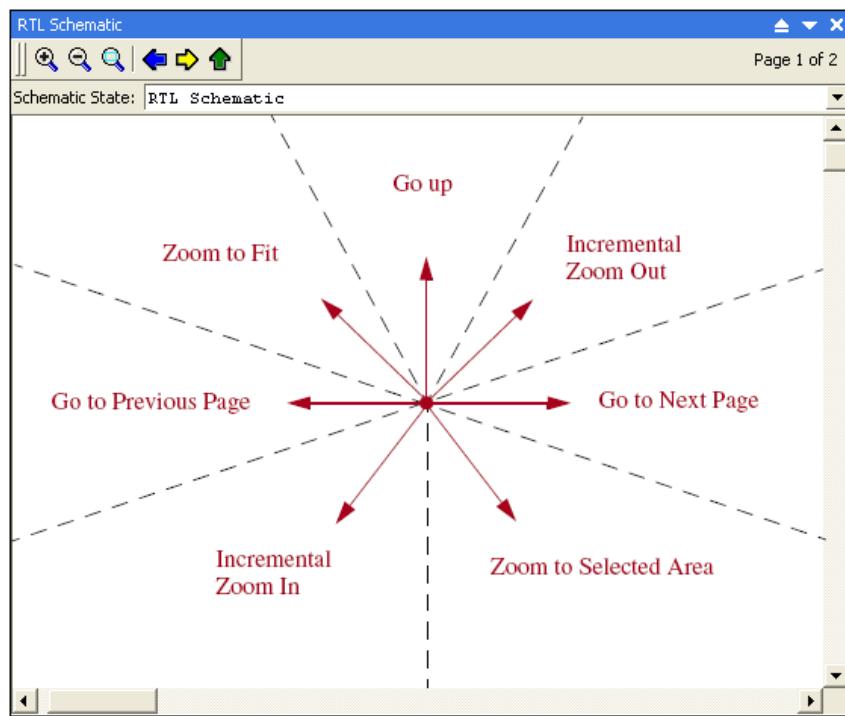
Figure 3-78. Obtaining Point-to-Point Delay Data from the Schematic



Zoom and Page Navigation Controls

The zoom and page navigation commands are accessible from the tool bar, the popup menu, and using mouse strokes. Stroke commands are initiated by pressing and holding the left mouse button while dragging the cursor. When the button is released, the command is executed. As illustrated in Figure 3-79, the direction of the stroke determines the command that is executed.

Figure 3-79. Stroke Command Patterns



Integrated Code Analysis Flow

Catapult provides the “CXXAnalysis” flow that performs static and runtime checks, as well as code coverage analysis, of your design and testbench code. The respective reports can help you achieve the most optimal coding style for synthesis by Catapult.

The following three types of analysis can be performed:

- Static Analysis reports un-initialized variables.
- Runtime Analysis reports out-of-bounds array accesses.
- Coverage Analysis reports how well each branch of the design code is exercised by the data set in the test bench.

The flow obtains the report data from the compilation tools installed on your system, then reformats the data into report files that are saved in the Catapult solution directory. The supported compilation tools are:

- For Linux/Unix systems, GNU GCC is used. Mudflap, Gcov, and Purify will be used to carry out runtime and coverage analyses.
- For Microsoft Windows systems, Microsoft Visual Studio is the only supported compiler. Purify will be used to carry out runtime and coverage analysis.

Note



Evaluation versions of the Purify software is not supported by the CXXAnalysis flow. The evaluation version does not allow file output, resulting in unusual error messages.

Enabling the CXXAnalysis Flow and Setting Options

In the Flow Manager window, select “CXXAnalysis” to display its description information. Optionally select a flow package version from the “Select Version” drop-down menu, then click the Enable button. Once enabled, the CXXAnalysis flow options appear in the Flow Manager window.

Use the following command to enable the flow from the command line:

```
flow package require /CXXAnalysis
```

The CXXAnalysis flow options can be modified at anytime from Catapult Options window. Refer to “[CXXAnalysis Flow Options](#)” on page 202.

Running the CXXAnalysis Flow

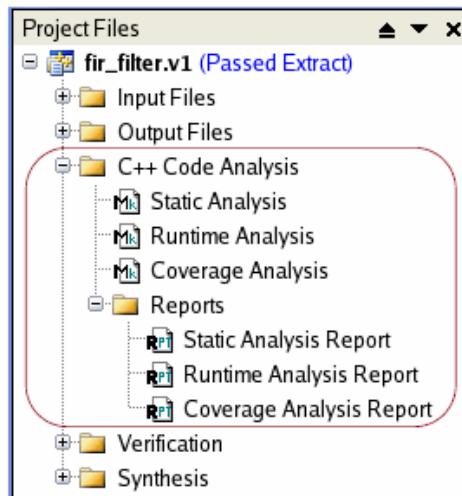
Once the flow is enabled, Catapult will automatically run it as soon as the solution reaches the “compile” stage. If the solution has past the “compile” stage, the flow runs immediately after it is enabled. To run the flow manually, execute the following command:

```
flow run /CXXAnalysis
```

When the flow runs, it first generates a makefile for each of the three types of analyses, and then executes each makefile whose corresponding “autorun” option is enabled in the CXXAnalysis flow options. The makefiles are created in the solution directory and are used to invoke the underlying tools that generate the analysis data. The CXXAnalysis flow extracts relevant data from the output of the underlying tools and generates corresponding report files in the solution directory. The report file names are:

```
static.rpt  
coverage.rpt  
runtime.rpt
```

Each time an analysis is performed, its generated report is automatically opened for viewing in the Catapult GUI. The makefiles and report files are accessible from the Project Files window. Figure 3-80 shows the Project Files window after all report files have been generated.

Figure 3-80. CXXAnalysis Flow Output Files

Individual analysis reports can be generated at any time by executing the appropriate makefile interactively. From the Project Files window, right-click on a makefile and choose “Perform Analysis” on the popup menu. To execute the makefile from command line, use the flow run command as follows:

```
flow run /CXXAnalysis/launch_make ./<makefile> <analysis_type>
```

where:

<makefile>	= cxxanalysis_<analysis_type>_<tool>.mk
<analysis_type>	= one of "static," "runtime" or "coverage."
<tool>	= one of "gcc" or "msvc"

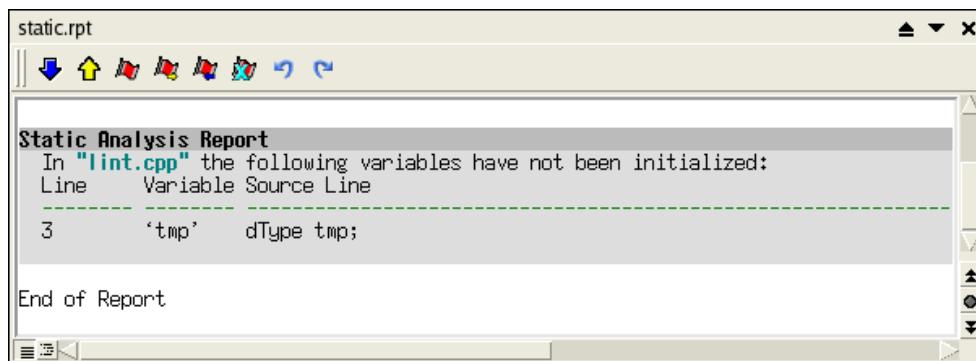
Examples:

```
flow run /CXXAnalysis/launch_make ./cxxanalysis_static_msvc.mk static
flow run /CXXAnalysis/launch_make ./cxxanalysis_runtime_msvc.mk runtime
flow run /CXXAnalysis/launch_make ./cxxanalysis_coverage_msvc.mk coverage
```

Temporary files generated by each type of analysis are saved in separate subdirectories in the solution directory. The subdirectory names are cxxanalysis_<analysis_type>.

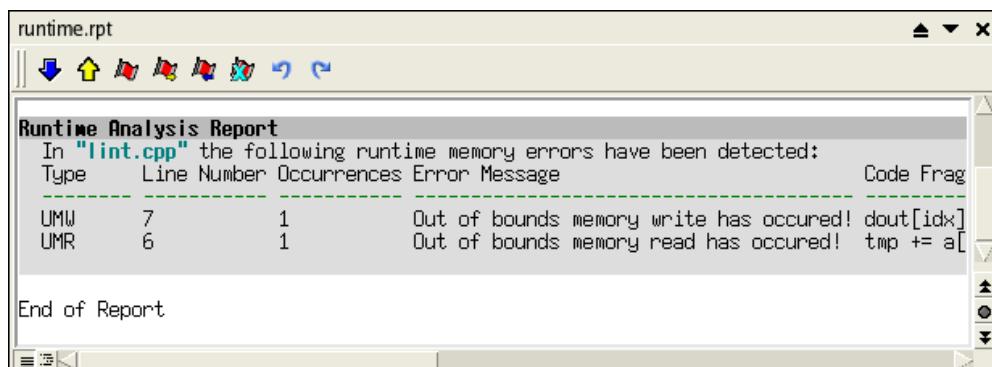
Figure 3-81 shows the format of the static report. If errors are found, it lists the source code line on which the error occurs and the type of error.

Figure 3-81. Static Report



The runtime report, shown in Figure 3-82, tells you what type of runtime errors are found, and lists the source code line on which the errors occur.

Figure 3-82. Runtime Report



As shown in Figure 3-83, the coverage report contains two sections. The first lists the lines in your code that not being executed by the testbench. The second section lists each line of the source code and shows the number of times that line executes in the “Coverage” column.

Figure 3-83. Coverage Report

```

coverage.rpt
|| D Y R B C M N E

Coverage Analysis Report
Your test bench is not executing the following lines of your design in: "test.cpp"
Line Number Coverage Source Line
9      0      dout[0] = a[0] - b[0];
10     0      dout[1] = a[1] - b[1];
11     0      dout[2] = a[2] + b[2];

Coverage Overview
Complete coverage information for "test.cpp"
Line Number Coverage Source Line
1      -      #include "test.h"
2      1      void test(dType a[3], dType b[3], dType dout[3], bool sel)
3      -
4      1      if(sel){
5      1      dout[0] = a[0] + b[0];
6      1      dout[1] = a[1] + b[1];
7      1      dout[2] = a[2] + b[2];
8      -
9      0      }else{
10     0      dout[0] = a[0] - b[0];
11     0      dout[1] = a[1] - b[1];
12     -
13     3      dout[2] = a[2] + b[2];
14     1      }

End of Report

```

Understanding Messages in the Transcript

Catapult displays various types of messages in the transcript window to keep you informed about the state of the design, as well as problems and potential problems it detects. This section describes the meaning of certain elements of transcribed messages.

Runtime Information

Three types of runtime informational messages are described here:

1. Catapult marks the beginning and end of each design transformation (i.e. analyze, compile, architect, allocate, schedule, dpfsm and extract) with “start” and “complete” messages. The completion messages report the elapsed time and memory usage data about the transformation. For example:

```
# Info: Starting transformation 'architect' on solution 'dct.v1'
(SOI-8)
...
```

```

# Info: Completed transformation 'architect' on solution 'dct.v1':
elapsed time 3.91 seconds, memory usage 95564kB, peak memory usage
121676kB (SOL-9)

# Info: Starting transformation 'allocate' on solution 'dct.v1'
(SOL-8)
...
# Info: Completed transformation 'allocate' on solution 'dct.v1':
elapsed time 0.61 seconds, memory usage 97204kB, peak memory usage
121676kB (SOL-9)

# Info: Starting transformation 'schedule' on solution 'dct.v1'
(SOL-8)
...
# Info: Completed transformation 'schedule' on solution 'dct.v1':
elapsed time 5.14 seconds, memory usage 103312kB, peak memory usage
129260kB (SOL-9)

```

2. Optimization messages also provide runtime information in the format shown below. The messages display the number of operations and variables in the design after the optimization has been performed.

```

# Info: Optimizing partition '/dct': (Total ops = 233, Real ops =
67, Vars = 196) (SOL-10)
# Info: Optimizing partition '/dct/dct:core': (Total ops = 199, Real
ops = 67, Vars = 171) (SOL-10)

```

3. As the scheduler iteratively optimizes the schedule to find the best results, it prints latency and area information at the conclusion of each optimization iteration, but only if the results are better than the previous iteration. The latency reported for any loop includes the total number of iterations through the loop and the latency of sub-loops. The final results for the entire design are printed after the scheduler is finished.

The information is reported in a series of “Info” messages in the transcript window. The first message in the series provides the name of the function block and the results of the initial schedule. The last message contains the results of the final schedule for the block. All intermediate messages provide the name of the loop being optimized and the its results. Each type of message is shown in the following example:

```

# Info: Initial schedule of SEQUENTIAL 'core': Latency = 1158, Area
(Datapath, Register, Total) = 18572.45, 1000.61, 19573.05 (CRAAS-11)

# Info: Optimized LOOP 'inner2': Latency = 1158, Area (Datapath,
Register, Total) = 14625.18, 1000.61, 15625.78 (CRAAS-10)

# Info: Optimized LOOP 'inner2': Latency = 1158, Area (Datapath,
Register, Total) = 13825.40, 1000.61, 14826.01 (CRAAS-10)

# Info: Optimized LOOP 'middle2': Latency = 1158, Area (Datapath,
Register, Total) = 8001.84, 1000.61, 9002.45 (CRAAS-10)

...
# Info: Final schedule of SEQUENTIAL 'core': Latency = 1158, Area
(Datapath, Register, Total) = 5761.84, 1000.61, 6762.45 (CRAAS-12)

```

```
# Resource allocation and scheduling done. (CRAAS-2)
# Netlist written to file 'schedule.gnt' (NET-4)
```

Severity Classification of Messages

Messages fall into the four classifications listed below in priority order (highest to lowest):

- Error – Something has happened so that Catapult can't or shouldn't continue operation.
- Warning – Something that should be examined has happened.
- Informational – General statement describing the actions Catapult is performing.
- Comment – Informational statements that are not part of the system message database, and have no message identifier. These messages can originate from Tcl commands and custom scripts as well as from the Catapult system.

Every message has a factory default severity level. You can raise the severity level of messages in a few different ways. One way is to use the “**Change severity to:**” field on the **Long Description** dialog box as shown in [Figure 3-22](#) on page 75. Another way is to change the default Message settings as described in the section “[Message Options](#)” on page 176. Or you can use the [options set](#) command. The severity level of a message cannot be lower than its factory default level.

Message Identifiers

Each system message has a unique identifier that displays at the end of the short description (see [Figure 3-21](#)). The identifiers are composed of two elements, a text label that indicates type of operation that was being performed when the message was generated, and an integer to

uniquely identify each message associated with a label. Table 3-11 lists all of the identifier tags and their meanings.

Table 3-11. System Message Identifier Tags

General Category	Message Tag	Description
Design Flow Messages	ALOC	Allocation
	ASG	Assignment - Component binding and Sharing
	ASM	Assemble CCOREs
	CRAAS	Concurrent resource allocation and scheduling
	CIN	C SIFgen
	CRD	C reading
	FSM	FSM Extraction + Reg Sharing
	HIER	Hierarchical
	LOOP	Loop
	MEM	Memory and Interface Mapping
Optimization Messages	NET	Netlisting
	SCHD	Scheduling
Other Messages	OPT	Sequential Design Analysis + Other optimizations
	ASSERT-1	See the section “ ASSERT-1 Messages ” below
	BASIC	Low level
	CNS	Constraint Management
	LIB	Library
	LIC	License
	NL	General Netlisting
	PRJ	Project
	READ	SIF Reading
	SEQ	Sequential Component Analysis
	SIFG	SIF Gen (VHDL and Verilog)
	SOL	Solution
	VHDL	VHDL Netlisting
	VLOG	Verilog Netlisting
	WRITE	SIF Writing

ASSERT-1 Messages

A separate class of messages generated by Catapult are called SIF-Asserts. These are extremely rare occurrences because they can only occur if the Catapult internal database (know as SIF - Synthesis Internal Format) gets into an invalid state. The following example shows how an ASSERT-1 message appears in the transcript:

```
# Error: internal assertion failed (sif_assert_fail) Isrc/sif_ci_data.cxx
line 1141 (ASSERT-1)
```

The ASSERT-1 messages are Error-level severity by default. However, they are actually implemented as Warning-level severity that have been promoted to Error-level severity by using the “[Message Options](#)” on page 176 to override the system default. This type of implementation provides a mechanism for temporarily downgrading ASSERT-1 to Warning-level in the unlikely situation that you want Catapult to continue running in spite of the SIF-Assert.

File Versioning System

Catapult contains an optional file versioning system that maintains a backup of code revisions used for synthesis. The versioning system lets you modify your source code without worrying about losing the older versions of the code.

The file versioning system is turned off by default. To enable file versioning, select **Tools > Set Options > Project Initialization > Version Control** in the Catapult GUI and check the **Enable** option. See “[Version Control Options](#)” on page 179.

The Process for Handling Input Files

Catapult has a list of default directories that will not be backed up. The list will be editable from **Tools> Set Options > Project Initialization > Version Control** and includes the following defaults:

- System and Compiler include paths
- The “\$MGC_HOME” directory

Environment variables are allowed in directory paths. Any file that is not in an *excluded* directory will be backed up. Files are only backed up during a successful “[go compile](#)”.

Excluding Directories

You can identify a list of directories that you want excluded from the version control system. All files and subdirectories in an excluded directory will not be backed up. To edit the list of excluded directories, select **Tools > Set Options > Project Initialization > Version Control** and use the **Add**, **Edit** and **Delete** buttons to modify the list in the **Exclude Paths** field.

How Catapult Backs Up Files

All files in tagged directories will be backed up when compile completes. Backup does NOT occur if compile fails. These backup files are not immediately used.

If you edit a file after you have compiled, then the original file is saved and Catapult creates a new solution. When a new solution is created because a source file has changed, all Solutions are updated to point to the backed up version of that file that was used during compile. Cross probing from that solution will cross probe to the backed up file.

Note: All backed up files can still be accessed in read only mode.

Deleting Solutions

If a solution is deleted and it is the only solution that uses a version of a file, then that version of the file is no longer accessible. To avoid this situation, we suggest that you do not delete solutions. However, this can lead to large projects.

Checking Out Backed Up Files

You can select a solution and check out that version of all the source code. If you make edits to that version of the code and compile the code, the version of the newly saved file will reflect its origin. This lets you track the different solutions with different code changes.

You can enable the versioning system by using the following command or selecting it from the GUI as shown in the previous figure:

```
options set VersionControl Enabled true
```

This setting will take effect on the next new project. The Versioning system cannot be enabled or disabled once a project is created. This command will check out files that are associated with the current solution and replace the files on disk. The “-replace” switch is required to overwrite files that Catapult has not already backed up. If any files that will be overwritten are not backed up and the “-replace” switch is not used, then no files will be overwritten.

The backup and version control will be internal to Catapult. This means that it will not interact with an external version control system. However, files can be “checked out” of Catapult and checked in to your own version control system.

Only solution level access will be given to backed up files outside of the GUI. This means that you will not be able to check out a file without checking out all of the files associated with a solution.

Some of the features available with many version control systems, such as comments during check-in or file merging when checking out a file will not be supported.

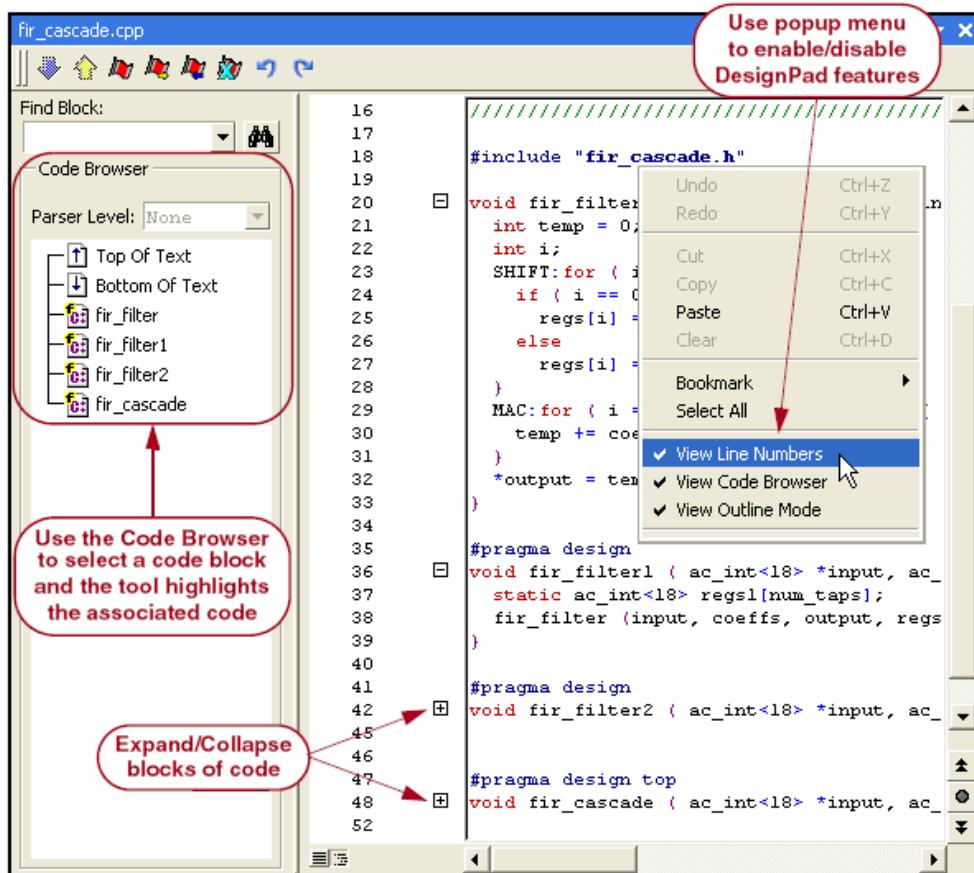
Note

Merging is not supported, so any changes on disk will be overwritten.

DesignPad Text Editor

The DesignPad editor provides productivity features such as a Code Browser, an Outline Mode, and line numbering of your code. Figure 3-84 shows an example of the DesignPad Editor window.

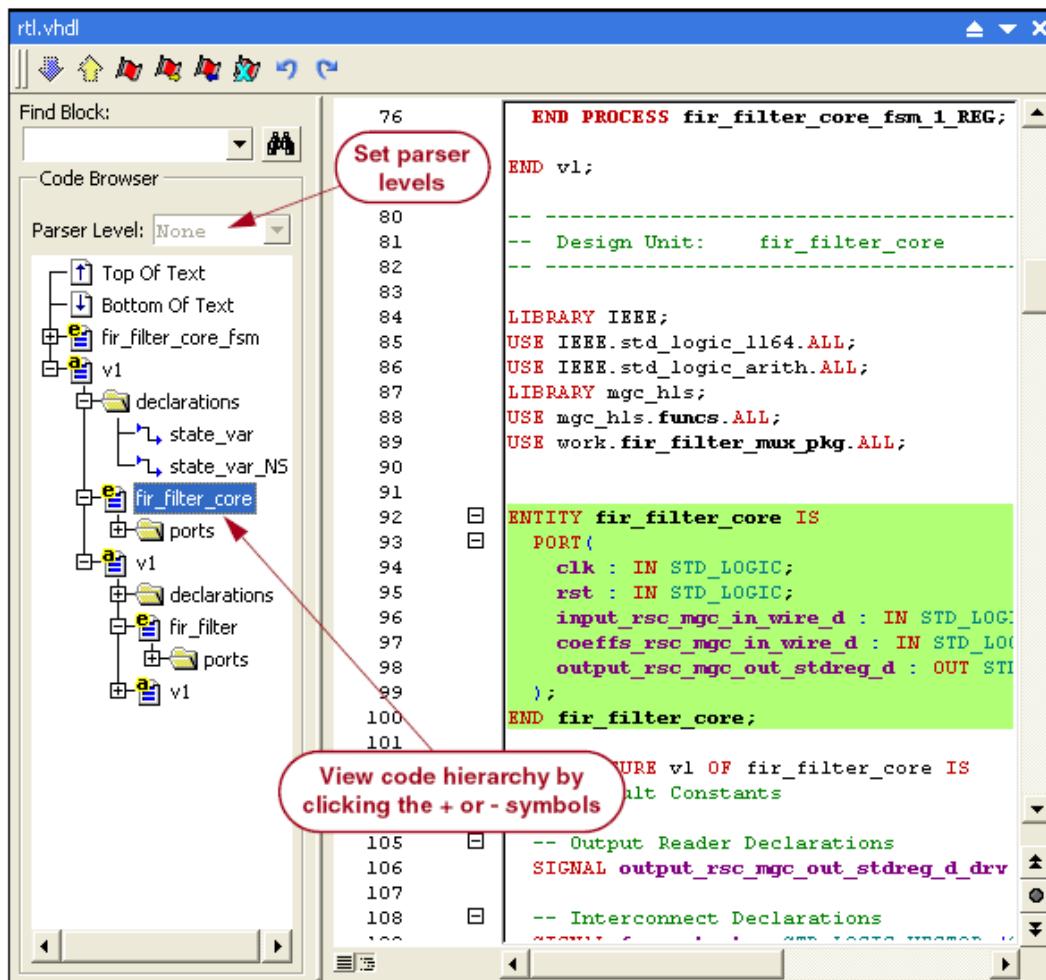
Figure 3-84. DesignPad Window



Browsing Code

The code browser represents the HDL code structure as a hierarchical tree and allows you to navigate through the code blocks or objects which correspond to lines or code blocks in the HDL text file. Figure 3-85 shows the structure of the hierarchical tree and describes how to browse the code and set parser levels.

Figure 3-85. Hierarchy and Parser Levels



Using the DesignPad editor, you can display an outline of the code structure and gain immediate access to any code block. You can also use bi-directional cross-highlighting to move between the code browser and the actual HDL text. You can set the parser level to *Full*, *Fast* or *None*: Full parsing locates all ports and declarations; Fast parsing displays instantiations only; None locates the top and bottom of the text file only.

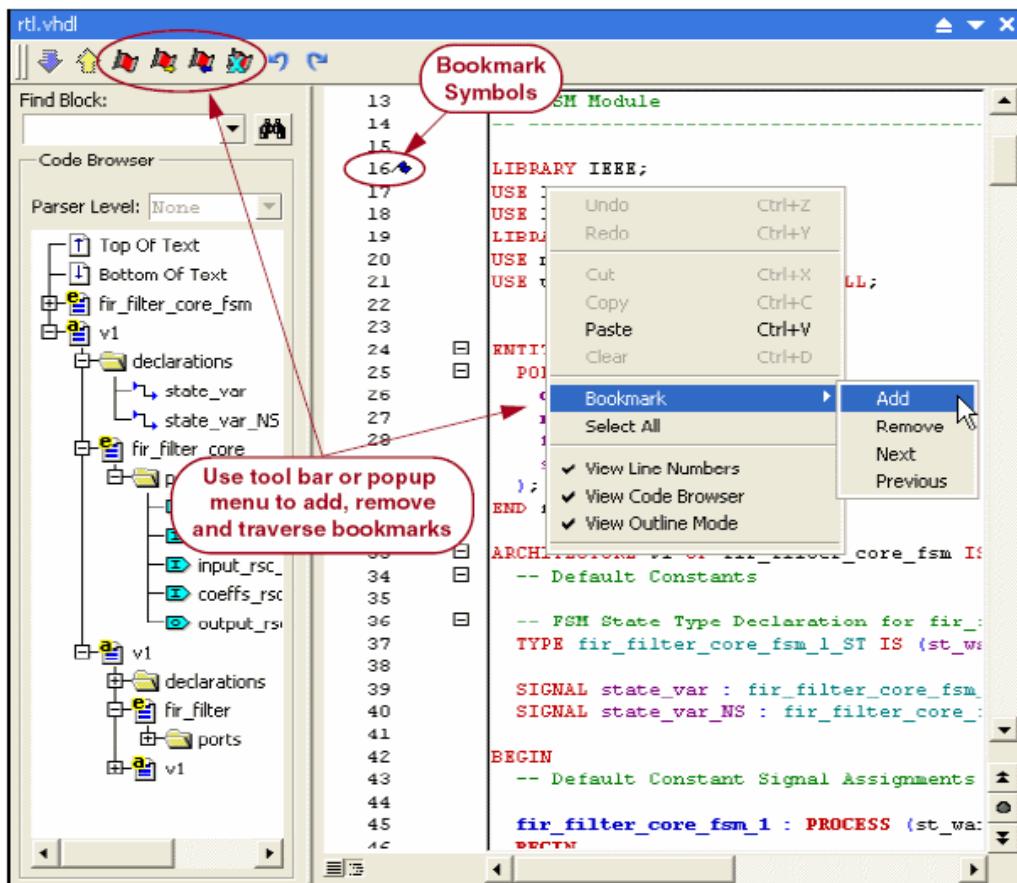
You can also type text in the “Find Block” field or use the Browse icon to specify a block of code. The tool will automatically move to this code block within the code browser window. Use the Find (binoculars) icon in the left pane of the Design window to search for text in the transcript window.

Bookmarks

You can set “Bookmarks” next to lines of code in the DesignPad window. You can then search for the bookmarks or make changes to them. As shown in Figure 3-86, bookmarks are indicated by a flag icon on the current line in the file.

You set and modify bookmarks by placing the cursor on the target line, then use either the tool bar buttons or the popup menu to access the bookmark commands. You can add, remove, or move to the next or previous bookmark from this window.

Figure 3-86. Setting DesignPad Bookmarks



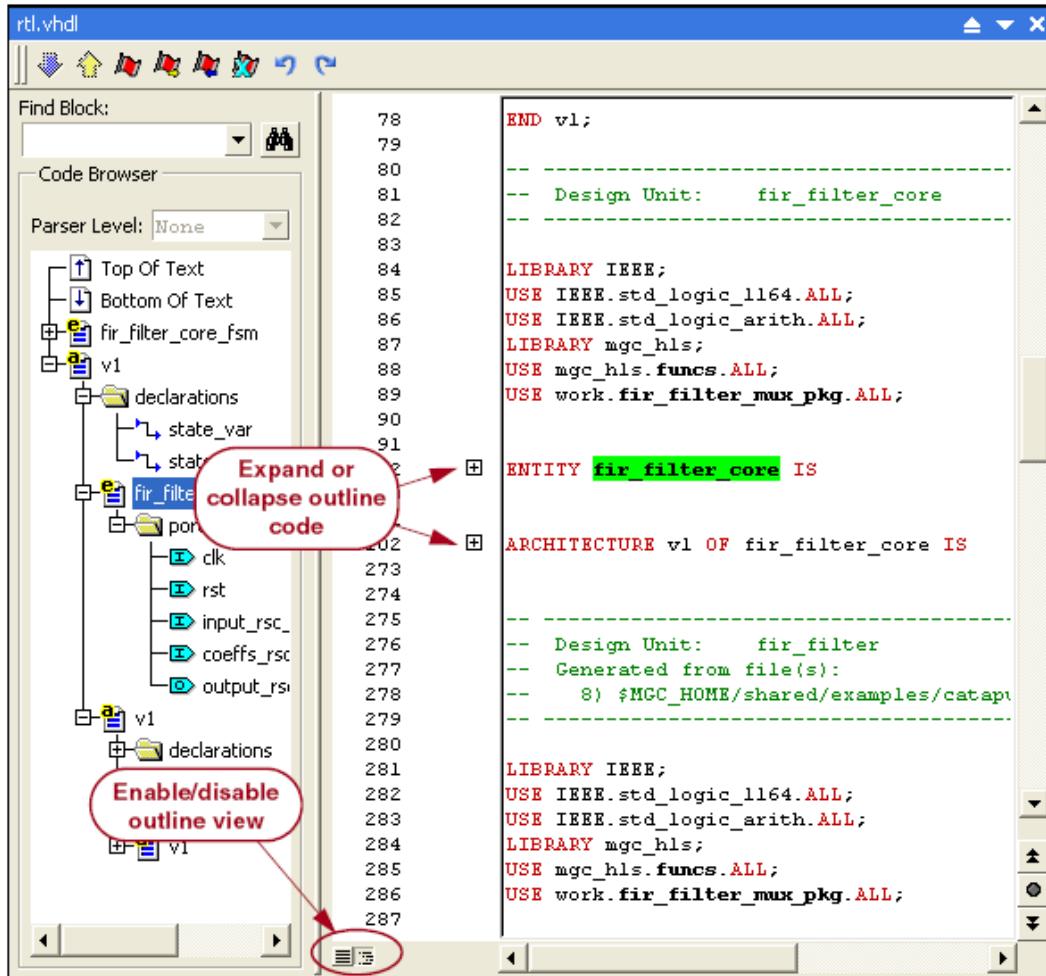
Outline Mode

Outline mode can be used when you do not want to view all of the contents of a long text file. Outline mode is indicated by plus or minus icons to the left of the code. Click a plus/minus icon to show/hide a block of code.

For example, you might want to collapse a VHDL entity and architecture file in outline mode to show only the entity and architecture declaration lines. You can expand the entity or architecture and then expand lower level code blocks such as the port declarations.

You select Outline mode by placing the cursor in the transcript window and doing a right-mouse click to display the window shown in the following figure. Click next to the View Line Numbers, View Code Browser, or View Outline Mode fields to select or deselect them.

Figure 3-87. DesignPad Outline Mode



Understanding Catapult Project File Types

Input File Types

Table 3-12 identifies the set of valid input file types for Catapult. All input files are read as C++/SystemC files. It is common to place input files in a separate sub-directory to keep them isolated from tool-generated files.

Table 3-12. Input File Extensions

File Extension	File Description
.c	Design source file in C format.
.cpp .cxx	Design source file in C++ format.
.h .hh	Design header file. Can contain SystemC design.

Project Control Files Types

Table 3-13 identifies the control file types that are generated by Catapult for managing a project. Catapult creates and maintains these files in the working directory. These file types are common to the project during every synthesis run.

Table 3-13. Catapult Control Files

File Extension	File Description
catapult.log	A transcript of the most recent Catapult C Synthesis run. Can be used as a command file to repeat the run. Always located in the project working directory.
<project_name>.ccs	A Catapult C Synthesis Project File. Can execute all the previous project steps to restore the saved project state. Use the File > Open Project pull-down menu to execute.
<name>.tcl	A command file created with a common text editor or generated by from the “catapult.log” file. This file can be executed with the “ catapult -file ” command or you can use the File > Run Script pull-down menu.
messages.txt	A text file containing all of the Catapult system messages generated by the solution. This file exists in the solution directory.
directives.tcl	A Tcl script containing all of the directive commands, go commands, and input file paths needed to regenerate the solution. This file provides an automated method for recreating design solutions in newer versions of the Catapult C Synthesis.

Output Files

Table 3-13 identifies the various types of output files generated by Catapult and placed in a Solution sub-directory.

The basenames of the files in the table are the factory default names. For information about how to change the default basenames, refer to “[Configurable Basenames for Generated Files](#)” on page 146.

Table 3-14. Output Files

rtl.<lang>.<tool>	Script to launch downstream RTL synthesis tool. <lang>=[v vhdl], <tool> = [dc psr rc]
rtl.<lang>.<tool>_timing	Timing data for Design Compiler script. <lang>=[v vhdl], <tool> = [dc psr rc]
rtl.<lang>.msim	ModelSim control file for the RTL design. <lang>=[v vhdl cxx]
rtl.<lang>.psr	Precision RTL Synthesis project file for the RTL design. <lang>=[v vhdl cxx]
rtl_funcs.vhd	Package that declares synthesizable functions needed for RTL output.
rtl_mgc_ioport.(v or vhdl) rtl_mgc_ioport_v2001.v	Calypto Design Systems I/O port components (Verilog or VHDL).
schedule.gnt	Gantt chart view of scheduling.
schematic.nlv	Schematic Viewer file of the entire RTL design.
*_mmr.xml	Memory map settings exported to XML file in IP-XACT format.
*.xrf	Files for cross-probing.

SIF Files

SIF (Synthesis Internal Format) files comprise the Catapult design database. The current state of your project and all of its solutions are captured in these files, which are used to restore a saved project when you select **Files > Open Project**.

Incremental Update Files

Table 3-15 describes three Tcl scripts generated after key transformations of the incremental solution and saved to the solution directory (not visible from the GUI). Each Tcl script reflects the solution cycle and sharing behavior constraints for a particular transformation and can be used for advanced analysis of the incremental solution.

Table 3-15. Incremental Solution Tcl Scripts

Tcl Script Filename	Constraints	Generated After...
cycle_set.tcl	Operations cycle (CSTEPS_FROM)	schedule
reg_sharing.tcl	Register sharing (REGISTER_NAME)	dpm

Table 3-15. Incremental Solution Tcl Scripts

Tcl Script Filename	Constraints	Generated After...
res_sharing.tcl	Resource sharing (RESOURCE_NAME)	extract

Chapter 4

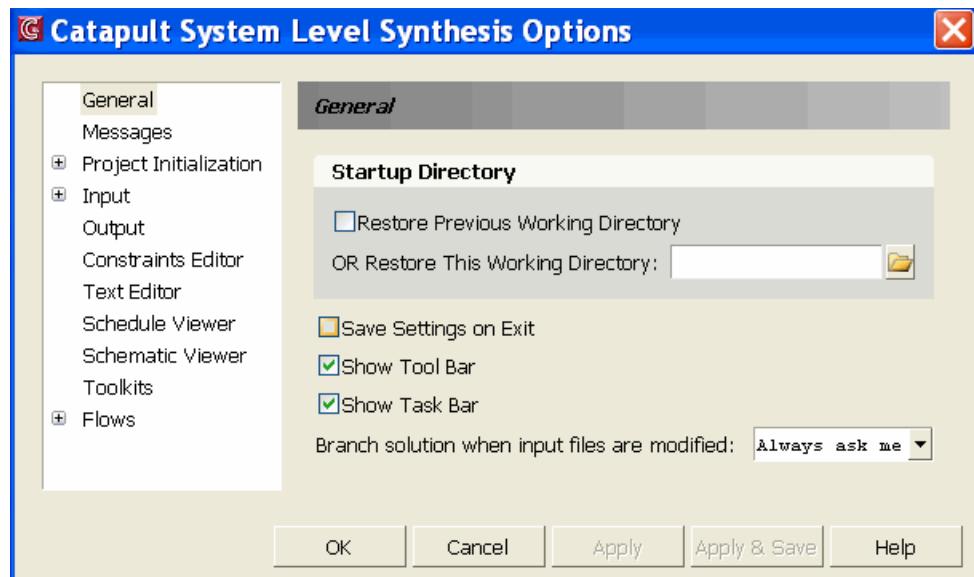
Setting Up Catapult Default Options

This chapter describes the session-level options available in Catapult and the user interface for setting default values for these options. When you change the option settings, the changes take effect immediately and remain in effect for the duration of the current Catapult session. To save your settings for use in subsequent sessions refer to “[Saving and Restoring Session Options](#)” on page 223. See also “[The Catapult Initialization File](#)” on page 224.

The Options Window

The Catapult Synthesis Options window shown in Figure 4-1 allows you to set default values for GUI options throughout the design environment. To open the Catapult Synthesis Options window, choose **Tools > Set Options....**

Figure 4-1. General Options Dialog Box



The left side of the window lists the categories of options. The right side of the window contains a dialog box of options for the currently selected category. The function of each window button is described in Table 4-1.

Table 4-1. Buttons in the Catapult Options Window

Button	Function
OK	Apply the option settings in the design database and close the Options window.
Cancel	Discard changes to the option settings and close the Option window
Apply	Apply the option settings in the design database.
Apply & Save	Apply the option settings in the design database and save the new settings in the Catapult Registry .
Help	Open online help for the active options page.

The Catapult Synthesis Options window contains the following categories of options, each of which is described in this section:

General Options	175
Message Options	176
Project Initialization Options	178
Version Control Options	179
Component Libraries Options	180
Interface Options	181
Architectural Options	184
Hardware Options	186
Input Options	188
View Compiler Settings	189
Output Options	189
Constraints Editor Options	195
Text Editor Options	195
Schedule Viewer Options	196
Schematic Viewer Options	197
Toolkits Options	198
Flows Options	198
ModelSim Flow Options	199
Precision RTL Flow Options	201
CXXAnalysis Flow Options	202
CXXAnalysis Flow Options	202
DCPower Flow Options	203

Design Compiler Flow Options	204
MATLAB Flow Options.....	207
NCSim Flow Options	207
Novas Flow Options	208
OSCI Flow Options	209
PowerPlay Flow Options	210
RTLCompiler Flow Options.....	210
SCVerify Flow Options	212
SLEC Formal Verification Flow Options.....	215
SOPC_Builder flow Options	217
SpyGlassPower Flow Options	218
TalusDesign Flow Options	219
Valgrind Flow Options	221
VCS Flow Options	221
Vista Flow Options.....	222
XPower Flow Options	222

General Options

The General options dialog lets you control tasks such as setting the default startup directory, how tool bars are displayed, and whether option settings are automatically saved. All of the General options are described in the following list:

- **Startup Directory**

These options are used to set the default current working directory. If not set, then the Catapult working directory is the directory from which it was launched.

If Catapult is running on a Windows operating system and you invoke the tool from the Start menu, you will probably want to enable this option. This will cause Catapult to start up in the last directory in which it was started.

- **Restore Previous Working Directory**

Check this box to restore the working directory from the previous session. If this option is checked, the “Or Restore This Working Directory” setting is ignored. You can also set this option using the “*options set General RestoreCWD <true_or_false>*” command.

- **Or Restore This Working Directory**

Use this option to set the default working directory to one specific directory, causing Catapult to always start in that directory. Click the **Browse** icon to navigate to the directory that you want to use on startup. You can also set this option using the “*options set General StartInWD <path>*” command.

- **Save Settings on Exit**

Check this box to save all option settings when the Catapult session closes. The options are saved into the [Catapult Registry](#). This ensures that the options settings will appear each time the tool is invoked, regardless of the invocation directory. You can also set this option using the “[*options set General SaveSettings <true_or_false>*](#)” command. For more information about saving option settings and about the Catapult registry, refer to the section [“Saving and Restoring Session Options”](#) on page 223.

- **Show Tool Bar**

Uncheck the box to hide the Tool Bar on startup (see [“The Tool Bar”](#) on page 65). The view menu can be used to display the Tool Bar when it’s hidden. You can also set this option using the “[*options set General ShowToolBar <true_or_false>*](#)” command.

- **Show Task Bar**

Uncheck the box to hide the Task Bar window on startup. The view menu can be used to display the Task Bar when it’s hidden. You can also set this option using the “[*options set General ShowTaskBar <true_or_false>*](#)” command.

- **Branch solution when the input files are modified**

Set the policy for branching a new solution when input files are modified. By default Catapult prompts you before branching. The other options are to always branch without asking, or to never branch. When the “never branch” policy is selected, the user might need to manually branch a new solution, depending on the type of changes made to the input file(s). You can also set this option using the “[*options set General BranchOnChange <string>*](#)” command, where <string> can be one of the following: {Always ask me}, {Always branch} or {Never branch}.

Message Options

Catapult provides many informational, warning and error messages. These messages are displayed in the transcript to give you feedback about the design process. The Messages options allow you to control the visibility and severity-level of messages in the transcript. For more information, refer to [“Understanding Messages in the Transcript”](#) on page 159.

Use the Messages dialog box, shown in the Figure 4-2, to change the message options.

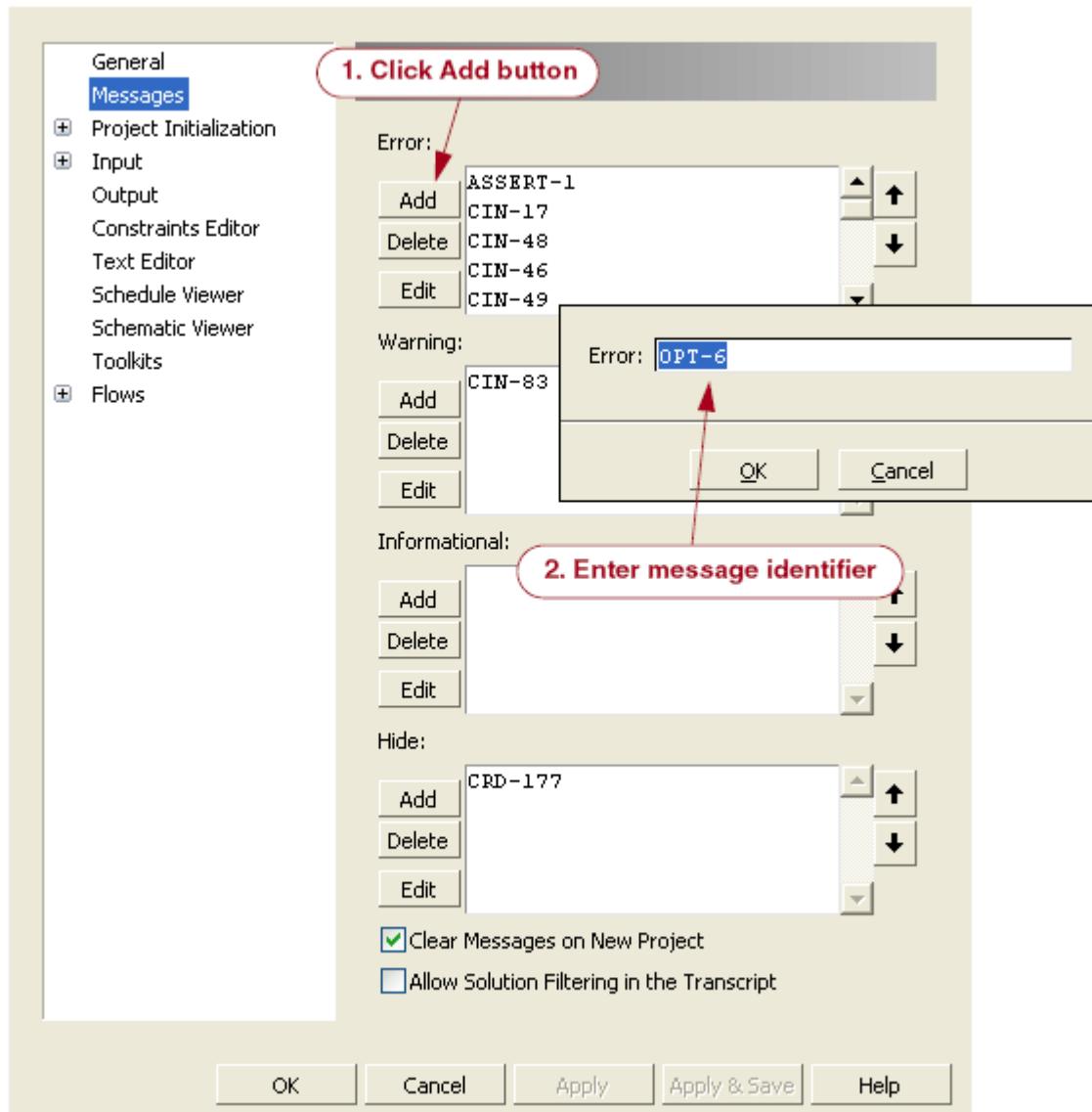
- **Error, Warning, Informational, and Hide boxes**

These fields allow you to control the severity level and visibility of individual messages. You can “hide” individual messages so that they will not be displayed. For example, if you want fewer messages in the transcript, you can hide some of the less important informational messages. You can raise the severity-level of messages. For example, you can have Catapult handle a warning message as if it were an error message. You cannot add, delete or customize the actual system messages.

The four option fields of this dialog box, Error, Warning, Informational and Hide, are each lists to which you can add or delete message identifiers. A message identifier is a

unique name given to a particular message. Refer to “[Message Identifiers](#)” on page 161 for detailed information about the format and categories of message identifiers.

Figure 4-2. Messages Options Dialog Box



All messages have a default severity level (see “[Severity Classification of Messages](#)” on page 161). This dialog allows you to promote messages to higher levels of severity (Error, Warning, or Informational) by adding the messages to the desired lists. If a message belongs to multiple lists, then the higher priority list is used. You cannot downgrade the severity level of a message below its default severity level. When you click OK, this dialog box will check if the options are valid and report an error if they are not.

- **Adding a Message to a Category**

Click the **Add** button next to the desired message category (Error, Warning, Informational, or Hide) and the Add dialog box displays. Enter the message identifier and click OK to accept the entry.

In the example in Figure 4-2, the severity level of the “OPT-6” message is being increased from an Informational message (its default) to an Error message. If the tool encounters the condition associated with this message during the next Compile operation, it will report the message as an error in the transcript window as shown in the following example.

```
# Error:  
$MGC_HOME/shared/examples/catapult/fir_filter/fir_filter.c(15):  
Inout port 'input' is only used as an input. (OPT-6)  
# Error:  
$MGC_HOME/shared/examples/catapult/fir_filter/fir_filter.c(15):  
Inout port 'coeffs' is only used as an input. (OPT-6)
```

- **Deleting a Message from a Category**

Select the message identifier you want to delete from the Error, Warning, Informational, or Hide category and click the **Delete** button. The message number is removed from the list. This does not delete the message from the Catapult database, but simply reverts that message to its default severity level.

- **Editing a Message in a Category**

Select the message identifier you want to edit from the Error, Warning, Informational, or Hide categories and click the **Edit** button. The selected message number appears in the edit dialog box. Click OK to save your changes.

- **Clear Messages on New Project**

Specifies whether messages in the transcript window will be preserved or cleared when a new project is created. The messages in the session log file are not affected by this option. You can also set this option using the “*options set Message NewProjectMessagesClear <true_or_false>*” command.

- **Allow Solution Filtering in the Transcript**

Specifies whether messages in the transcript window will be filtered by the “Current Solution/All Solutions” filter. Refer to “[Solution Selection and View Controls](#)” on page 66 for more information about setting this filter. You can also set this option using the “*options set Message AllowTranscriptSolutionFiltering <true_or_false>*” command.

Project Initialization Options

The Project Initialization dialog box lets you set the default project and solution names.

- **Default Project Name**

Defaults to “Catapult.” This is the prefix of the project directory name for the current session. A new project directory is created in the specified working directory each time the tool is invoked or by creating a new project (**File > New Project** menu). A suffix of the form “_<number>” is appended to insure a unique project name (for example: Catapult_12).

The default project name can also be set using the “*options set ProjectInit ProjectNamePrefix <string>*” command.

- **Solution Name and Solution Version**

These two options combine to form the names of solutions for the current project. A new solution directory is created in the current project directory each time a new solution is created. Solution names have the format <name>.<version_string>, where <name> is defined by the “Solution Name” option, and <version_string> is defined by the “Solution Version” option. The factory default settings use the following internal variables:

Option	Default Value	Description
Solution Name	%design%	Variable that resolves to the base name of the top-level function of the design.
Solution Version	v%version%	Literal character ‘v’ followed by the integer value of the %version% variable. Catapult increments %version% based on the existing solution names in the project directory.

The default solution name can also be set using the following commands:

```
options set ProjectInit SolutionName <string>
options set ProjectInit SolutionVersion <string>
```

Version Control Options

Catapult contains an optional versioning system that maintains a backup of C code revisions that are being used for synthesis. The Version Control dialog box lets you enable and configure this feature.

- **Options**

- **Enable**

This option is disabled by default and no files are backed up. Check this box to turn on version control and save files. You can also set this option using the “*options set ProjectInit/VersionControl Enabled <true_or_false>*” command.

- **Transcript Verbose Messages**

Check this box if you want the tool to show verbose messages in the transcript window. This option only affects messages related to version control operations.

You can also set this option using the “`options set ProjectInit/VersionControl Verbose <true_or_false>`” command.

- **Query When Overwriting on Disk**

By default, Catapult will ask you for confirmation before overwriting files. Uncheck this box to allow Catapult to overwrite files without asking. You can also set this option using the “`options set ProjectInit/VersionControl QueryOverwrite <true_or_false>`” command.

- **Exclude Paths From Backup**

- **Compiler Include Paths**

Compiler include paths are excluded from the backup by default. Uncheck this box if you want the tool to backup compiler include paths. You can also set this option using the “`options set ProjectInit/VersionControl ExcludeCompilerIncludes <true_or_false>`” command.

- **Paths**

The paths in this list will be excluded from version control and will not be backed up. On UNIX systems, the paths “\$MGC_HOME” and “/usr/include” are excluded from version control by default. Use the **Add**, **Delete** and **Edit** buttons to modify the list of paths. You can also set this option using the “`options set ProjectInit/VersionControl ExcludePaths <list_of_paths>`” command.

Component Libraries Options

The Component Libraries dialog box lets you set library search path, default synthesis tool and default libraries options.

- **Library Search Path**

Use this field to add other search locations for Catapult libraries. Designs targeting the supported FPGA technologies should leave this as the default. Designs using an ASIC flow need to add the path to the location of the ASIC library created with the Library Builder. Use the **Add**, **Edit** and **Delete** buttons to modify the list of paths. You can also set this option using the “`options set ProjectInit/ComponentLibs SearchPath <list_of_paths>`” command.

- **Default Synthesis Tool**

Sets the default for the Synthesis Tool field of the Setup Design view in the Constraint Editor window. Precision RTL Synthesis, Synopsys Design Compiler, Magma Talus Design, and Cadence RTL Compiler are currently supported. You can also set this option using the “`options set ProjectInit/ComponentLibs DefaultSynthesisTool`” command.

`<tool_name>`” command, where `<tool_name>` can be one of the following strings: Precision, DesignCompiler, TalusDesign, or RTLCompiler.

- **Default Libraries**

Specify a default set of technology libraries for when a new project is created. The value in this field will be used to set the **TECHLIBS** directive. A simple way to enter a value in this field is to first use the Setup Design window to set the target technology, then copy the resulting TECHLIBS setting from the transcript window and paste it here.

The following example entry specifies four libraries for VIRTEX-II: the base FPGA library, the specific target technology (2V40cs144), and the dual-port and single-port RAM component libraries.

```
{ {mgc_Xilinx-VIRTEX-II-4_beh_psr.lib {
    {mgc_Xilinx-VIRTEX-II-4_beh_psr part 2V40cs144} }
    {ram_Xilinx-VIRTEX-II-4_RAMSB.lib ram_Xilinx-VIRTEX-II-4_RAMSB}
    {ram_Xilinx-VIRTEX-II-4_RAMDB.lib ram_Xilinx-VIRTEX-II-4_RAMDB}
}}
```

You can also use the “*options set ProjectInit/ComponentLibs DefaultLibs <list_of_libs>*” command.

- **Constant Multipliers**

This optimization decomposes a constant multiplier into shifts and adders, leading to a higher performance design. This optimization is turned on by default for ASIC and disabled for FPGA technologies with hardware multipliers. The optimization is controlled using the **OPT_CONST_MULTS** directive.

You can also use the “*options set ProjectInit/ComponentLibs ConstantMults <value>*” command, where `<value>` can be one of the following: **on**, **off** or **default**.

Interface Options

The Interface dialog box lets you set default clock, and handshaking settings on the interface.

- **Default Clock Settings**

- **Name:** Specify a default clock name. The default clock name should be a string that is allowed in the target RTL language. You can also set this option using the “*options set ProjectInit/Interface DefaultClockName <string>*” command. This option corresponds to the **CLOCK_NAME** property of **CLOCKS** directive.
- **Edge:** Select the default active clock edge to indicate whether it is rising or falling. You can also set this option using the “*options set ProjectInit/Interface DefaultClockEdge <rising_or_falling>*” command. This option corresponds to the **CLOCK_EDGE** property of **CLOCKS** directive.
- **Period:** Specify the default clock period. You can also set this option using the “*options set ProjectInit/Interface DefaultClockPeriod <real_number>*” command.

This option corresponds to the **CLOCK_PERIOD** property of **CLOCKS** directive and the Design Frequency option of the Setup Design view in the Constraint Editor window.

- **High Time:** Specify the default clock high time in nano-seconds. When the clock high time is set, Catapult automatically computes and sets the clock Frequency and Period fields. The default high time is 50% of the clock period. You can also set this option using the “*options set ProjectInit/Interface DefaultClockHighTime <real_number>*” command. This option corresponds to the **CLOCK_HIGH_TIME** property of the **CLOCKS** directive.
- **Offset:** Specifies the offset of the rising edge of the clock in nano-seconds relative to time zero. Default value is zero. You can also set this option using the “*options set ProjectInit/Interface DefaultClockOffset <real_number>*” command. This option corresponds to the **CLOCK_OFFSET** property of the **CLOCKS** directive.
- **Percent Sharing Allocation:** Specify the default percentage by clicking the down arrow to select a value. This option corresponds to the **CLOCK_OVERHEAD** directive. You can also set this option using the “*options set ProjectInit/Interface DefaultClockOverhead <real_number>*” command. Default value is 20%.

Percent sharing overhead is the percentage of clock period reserved for the logic needed to share components. This can affect the latency and area of the design. Typical percentage is 10-20% for ASICs and 20-30% for FPGAs. Increasing the percentage typically produces a smaller design with a longer latency.

- **Uncertainty:** Specifies a global margin of error to compensate for the variance in the clock pulse due to jitter, skew, and so on. This value is subtracted from the total clock period before other clock constraints are applied. Specify an uncertainty value as a floating number in nanoseconds. You can also set this option using the *options set ProjectInit/Interface DefaultClockUncertainty <real_number>* command. Default value is 0. This option corresponds to the **CLOCK_UNCERTAINTY** property of the **CLOCKS** directive.

- **Default Reset Settings**

- **Synchronous Reset Name:** Specify a default name for the synchronous reset signal. The name should be a string that is allowed in the target RTL language. You can also set this option using the “*options set ProjectInit/Interface DefaultSyncResetName <string>*” command. Default value is “rst.” This option corresponds to the **RESET_SYNC_NAME** property of the **CLOCKS** directive.
- **Asynchronous Reset Name:** Specify a default name for the asynchronous reset signal. The name should be a string that is allowed in the target RTL language. You can also set this option using the “*options set ProjectInit/Interface DefaultAsyncResetName <string>*” command. This option corresponds to the **RESET_ASYNC_NAME** property of the **CLOCKS** directive.

- **Kind:** Select the type(s) of resets: synchronous, asynchronous or both. You can also set this option using the “*options set ProjectInit/Interface DefaultResetKind <sync_or_async_or_both>*” command. This option corresponds to the **RESET_KIND** property of the **CLOCKS** directive.
 - **Synchronous Reset Active:** Select the signal mode (“high” or “low”) that will trigger a synchronous reset. You can also set this option using the “*options set ProjectInit/Interface DefaultSyncResetActive <high_or_low>*” command. This option corresponds to the **RESET_SYNC_ACTIVE** property of the **CLOCKS** directive.
 - **Asynchronous Reset Active:** Select the signal mode (“high” or “low”) that will trigger an asynchronous reset. You can also set this option using the “*options set ProjectInit/Interface DefaultAsyncResetActive <high_or_low>*” command. This option corresponds to the **RESET_ASYNC_ACTIVE** property of the **CLOCKS** directive.
- **Default Enable Settings**
 - **Name:** Specify a default enable name. The name should be consistent with the target language. This feature adds an enable to all registers. An empty field turns this feature off by default. You can also set this option using the “*options set ProjectInit/Interface DefaultEnableName <string>*” command. This option corresponds to the **ENABLE_NAME** property of the **CLOCKS** directive.
 - **Active:** Select the signal mode (“high” or “low”) that will enable the design to run. You can also set this option using the “*options set ProjectInit/Interface DefaultEnableActive <high_or_low>*” command. This option corresponds to the **ENABLE_ACTIVE** property of the **CLOCKS** directive.
 - **Default Handshake Settings**
 - **Start Flag:** The process waits for the start flag to go high before beginning computation. Set this value to blank to disable the start flag by default. The default start flag name should be a string that is allowed in the target RTL language. You can also set this option using the “*options set ProjectInit/Interface DefaultStartFlag <string>*” command. This option corresponds to the **START_FLAG** directive.
 - **Ready Flag:** The process sets the ready flag high when it is ready to start computations. Set this value to blank to disable the ready flag by default. The ready flag name should be a string that is allowed in the target RTL language.
- T
- You can also set this option using the “*options set ProjectInit/Interface DefaultReadyFlag <string>*” command. This option corresponds to the **READY_FLAG** directive.
- **Done Flag:** The process sets the done flag high when the computation is complete, and the output signals are valid. Set this value to blank to disable the done flag by

default. The done flag name should be a string that is allowed in the target RTL language.

T

You can also set this option using the “[*options set ProjectInit/Interface DefaultDoneFlag <string>*](#)” command. This option corresponds to the [**DONE_FLAG**](#) directive.

- **Transaction Done Signal:** Adds transaction signals to all processes. When this setting is enabled, each process in a design has one or more signals that go high for one cycle at the completion of an I/O transaction. You can also set this option using the “[*options set ProjectInit/Interface DefaultTransactionDoneSignal <true_or_false>*](#)” command. This option corresponds to the [**TRANSACTION_DONE_SIGNAL**](#) directive.
- **Idle Signal:** Adds idle indicator signals to process blocks in the design and appends a string to the signal name. An idle signal is driven HIGH if all the following conditions are true:
 - The Catapult design is waiting for input data.
 - No inputs are supplied to the design.
 - All the output data has been read from the output ports of the design (non-pipelined designs and designs with distributed pipeline).

You can also set this option using the “[*options set ProjectInit/Interface DefaultIdleSignal <string>*](#)” command. This option corresponds to the [**IDLE_SIGNAL**](#) directive.

For more information see “[**Idle Signal Insertion**](#)” on page 592.

- **Register Idle Signal:** Inserts a register on idle signals to stabilize the output by ensuring a transition only occurs on a clock edge. Only available when idle signals are specified.

You can also set this option using the “[*options set ProjectInit/Interface DefaultRegisterIdleSignal <string>*](#)” command. This option corresponds to the [**REGISTER_IDLE_SIGNAL**](#) directive.

Architectural Options

The Architectural dialog box lets you set automatic memory mapping threshold, the register threshold, loop merging and speculative execution.

- **Automatic Memory Mapping Threshold**

Any array with a number of elements equal to or greater than this threshold value is automatically seeded with a default constraint to map the array to a RAM. If no RAM

library is selected in setup design, or if the array is smaller than the threshold value, then it is mapped to the default non-RAM component. For internal arrays, the default non-RAM component is a set of registers, and for external arrays the default non-RAM component is the default I/O component. Select the value using the up and down arrows. See “[Predictive Resource and Memory Mapping](#)” on page 258.

You can also set this option using the “[*options set ProjectInit/Architectural DefaultMemMapThreshold <integer>*](#)” command. See also [**MEM_MAP_THRESHOLD**](#) directive.

- **Register Threshold**

Any array with a number of elements equal to or greater than this threshold value and which is also mapped to a register bank, will produce an error in order to prevent long runtime. Select the value using the up and down arrows. See “[Predictive Resource and Memory Mapping](#)” on page 258.

You can also set this option using the “[*options set ProjectInit/Architectural DefaultRegisterThreshold <integer>*](#)” command. See also [**REGISTER_THRESHOLD**](#) directive.

- **Loop Merging**

Check to enable loop merging by default for all loops. Loop merging is a technique used to reduce latency and area consumption in a design by allowing parallel execution, where possible, of loops that would normally run in series. See “[Loop Merging](#)” on page 234.

You can also set this option using the “[*options set ProjectInit/Architectural DefaultLoopMerging <true_or_false>*](#)” command. See also [**MERGEABLE**](#) directive.

- **Speculative Execution**

Check to enable Speculative Execution optimization by default. This optimization analyzes conditional logic (such as if-else) in the design in order to reduce latency. It analyzes all conditional branches in order to find operations that are not dependent on the condition, and therefore can be scheduled in a different clock cycle.

This feature is enabled by default, however it is not employed unless it is needed. Catapult will attempt to schedule the design without using Speculative Execution. Only if scheduling fails due to pipelining or cycle constraints will Catapult attempt to optimize with Speculative Execution and reschedule.

You can also set this option using the “[*options set ProjectInit/Architectural DefaultSpeculativeExec <true_or_false>*](#)” command. See also [**SPECULATE**](#) directive.

- **Reset Clears All Registers**

Specifies whether all registers, or only the necessary subset of registers will be reset. You can also set this option using the “[*options set ProjectInit/Architectural*](#)

DefaultResetClearsAllRegs <true_or_false>” command. This option corresponds to the [RESET_CLEAR_ALL_REGS](#) directive.

- **Default Scheduler Settings**

Use the following options to enable and configure the new, combined scheduling and Allocation algorithm that was introduced in the 2007a release.

- **Use Old Scheduling and Allocation Algorithms**

Use this option to switch between the new combined scheduling and Allocation algorithm and the old scheduler. This option corresponds to the [OLD_SCHED](#) directive. You can also set this option using the “[*options set ProjectInit/Architectural OldSchedule <true_or_false>*](#)” command.

The following message will appear in the transcript during the scheduling transformation if the new scheduler is enabled:

```
# Performing concurrent resource allocation and scheduling ...
```

The old scheduler displays the following warning message in the transcript when the old scheduler is being used:

```
Warning: This scheduling mode will not be supported in future
releases of Catapult. Please set directive "OLD_SCHED" to false.
```

Note



If you use a “directives.tcl” file generated by Catapult 2007a or 2007a_update1, the old scheduler will be enabled because that was the default behavior in those versions. You can change the file to use the new scheduler by changing the OLD_SCHED setting to false.

- **Design Goal for New Scheduling and Allocation Algorithms**

This option sets the optimization goal for the scheduler. See also [DESIGN_GOAL](#) directive.

- **area**: Optimize to make the design small.
- **latency**: Optimize to make the design fast.

You can also set this option using the “[*options set ProjectInit/Architectural DesignGoal <area_or_latency>*](#)” command.

Hardware Options

The Hardware dialog box lets you set hardware optimization defaults for the assignment overhead and default safe FSM settings.

- **Default Assignment Overhead**

Assignment Overhead is the percent of the clock period that will not be used by sharing and re-allocation. This can be used to reserve some of the clock period for routing delay in the back-end flow. Assignment Overhead can be negative, which has the result of adding time to the clock period. Assignment overhead corresponds to the [ASSIGN_OVERHEAD](#) directive, which is different than the [CLOCK_OVERHEAD](#) constraint.

You can also set this option using the “[*options set ProjectInit/Hardware DefaultAssignOverhead <integer>*](#)” command.

- **Default Safe FSM**

Check this option to enable support for safe FSM flows. When enabled, Catapult will allow for a default state that is not reachable from other states during RTL simulation. The default state will unconditionally set the state to the first state in the FSM or the reset state, if such a state exists.

You can also set this option using the “[*options set ProjectInit/Hardware DefaultSafeFSM <true_or_false>*](#)” command. See also [SAFE_FSM](#) directive.

- **Default No X Assignments**

When this option is enabled, HDL components generated by Catapult will not include assignments of “X” values (“don’t care” values). Typically this option is used in conjunction with the “[NoPragmas in netlists](#)” option. Refer to “[Output Options](#)” on page 189.

You can also set this option using the “[*options set ProjectInit/Hardware DefaultNoXAssignments <true_or_false>*](#)” command. See also [NO_X_ASSIGNMENTS](#) directive.

- **Default Max Fanout For Register Replication**

Limits fanout of register outputs to the specified threshold value. Specify an integer value that reflects a fanout count, not an actual fanout load. This option sets the default value for the [REG_MAX_FANOUT](#) directive. For more information, refer to “[Register Fanout Optimization](#)” on page 279.

You can also set this option using the “[*options set ProjectInit/Hardware DefaultRegMaxFanOut <integer>*](#)” command.

- **FSM State Encoding**

This option sets the low-power state encoding method for FSMs. FSMs can potentially improve their power consumption numbers by employing one of the following low-power state encoding methods: none, binary, grey, onehot. Sub-processes inherit the setting on the top process unless individually overridden. For more information, refer to “[“FSM State Encoding”](#) on page 591. This option sets the [FSM_ENCODING](#) directive.

You can also set this option using the “[*options set ProjectInit/Hardware DefaultFSMEncoding <value>*](#)” command, where value is one of the following: **none**, **binary**, **grey** or **onehot**.

Input Options

Use the Input dialog box to specify default compiler settings. You can specify a compiler for Catapult to use, compiler command flags, and search paths for header files and library files. Note that changes to these option will not take effect until a new project is created.

- **Compiler Home**

The Compiler Home options specify the location of the compiler Catapult will use.

- **Type:** This field provides a list of the compilers Catapult finds installed on your system. On UNIX/Linux systems, the compiler installed in “\$MGC_HOME/bin” is listed. On Windows systems, Microsoft compilers found in the Windows registry are listed, followed by the compiler supplied with ModelSim, if installed.

You can either select a compiler from the list, or you can select the **Custom** option to specify a different compiler. You can also set this option using the “[*options set Input Compiler <name_or_custom>*](#)” command.

This field works in conjunction with the [View Compiler Settings](#) page described in the next section.

- **Custom Path:** This field is used only when the “Custom” choice is selected in the Type field. Enter the path to top-level directory where the compiler is installed. You can either type the path in the text box or use the file system browser (click on the icon to the right of the text box). You can also set this option using the “[*options set Input CompilerHome <path>*](#)” command.

- **Compiler Flags**

Command line flags passed to the compiler during C compilation. Compiler options can also be set by right-clicking on the input file. You can also set this option using the “[*options set Input CompilerFlags <list_of_flags>*](#)” command.

- **Header File Search Path**

This search path will be added to the default search path for the Catapult compiler. Use the **Add**, **Edit** and **Delete** buttons to modify the list of directories in the search path. You can also set this option using the “[*options set Input SearchPath <list_of_paths>*](#)” command.

- **Library File Search Path**

The Library File Search Path is only used by the SCVerify flow. It is not used for synthesis. This search path will be added to the default search path for the Catapult compiler. Use the **Add**, **Edit** and **Delete** buttons to modify the list of directories in the

search path. You can also set this option using the “*options set Input LibPaths <list_of_paths>*” command.

View Compiler Settings

The **View Compiler Settings** page works in conjunction with the “Compiler Home” field on the [Input Options](#) dialog box. This read-only page shows information about the compiler selected in the Compiler Home field. If the “Custom” option is selected and the “Custom Path” value is not valid, the fields on this page will be blank.

Output Options

The Output dialog box lets you set the HDL and schematic output file formats.

- **Output Format**

Select any combination of VHDL, Verilog and SystemC output formats. These output formats can be generated in the same flow. If you change the settings, the new settings take effect when the next HDL output is generated. You can also set these options using the “*options set Output <lang_name> <true_or_false>*” command, where <lang_name> is one of the following: *OutputVHDL*, *OutputVerilog*, *OutputSystemC*, *SystemCUseBigint* and *SystemCUseLogicForClk*.

Refer to “[Catapult HDL Output Files](#)” on page 142 for more information about the set of files generated for each output format.

Note

 Using uninitialized variables may cause simulation differences between SystemC netlists and VHDL/Verilog netlists. See “[Limitation of SystemC Netlists During Simulation](#)” on page 194 for more information.

- **SystemC Use sc_big(u)int**

This option controls which SystemC datatypes Catapult uses when handling integers. When this option is disabled (default), integers that are 64 bits or smaller are cast to *sc_int*/*sc_uint*, and those greater than 64 bits are cast to *sc_bigint*/*sc_bignum*. When this option is enabled, all integers are cast to *sc_bigint*/*sc_bignum*.

You should enable this option if you get compilation or simulation errors relating to data widths exceeding 64 bits.

Note

 This option requires a SystemC 2.1 (or higher) compiler. ModelSim version 6.1 (and later) provides SystemC 2.1 support.

- **SystemC Use sc_logic as Clock Type**

Enabling this option will generate clocks of type sc_logic in the SystemC netlists. This option is for mixed language simulations in which multi-value clock ports are used. Because the SystemC clock type, sc_clock, is a boolean type, it is incompatible with multi-value clock types in VHDL/Verilog, such as std_logic.

- **NoPragmas in netlists**

Strips all “pragma translate_off (on)” blocks from generated netlist files. Setting NoPragmas to true will eliminate the code lines between the translate_off/translate_on pragmas:

```
// pragma translate_off
    <code ...>
// pragma translate_on
```

Typically this option is used in conjunction with the “[Default No X Assignments](#)” option. Refer to “[Hardware Options](#)” on page 186.

You can also set this option using the “*options set Output NoPragmas <true_or_false>*” command.

- **Package Output in Solution dir**

When this option is enabled, copies of all HDL files used by the solution, whether generated by Catapult or imported IP, are placed in the current solution directory. You can also set this option using the “*options set Output PackageOutput <true_or_false>*” command.

The types of files controlled by this option are all generated RTL, IP RTL, SystemC transactors and transactor resources. Catapult will issue an error if duplicate file names are encountered during the netlisting process. If multiple languages are output, all files for each language are saved in the solution directory. When this option is enabled, all Catapult synthesis and verification scripts are generated so that the dependencies are based on the files in the solution directory.

- **SystemC Netlist Namespace:**

Specifies a namespace string that will be applied to all SystemC netlist files generated by Catapult. The factory default string is “HDL.” The SystemC code in generated netlist files is encapsulated in a C++ “namespace <string>” block in order to avoid potential ambiguity between object names in the netlist and names in the SystemC files generated by the SCVerify flow. The SCVerify flow uses scope syntax (<namespace_string>::<object_name>) to reference objects in the SystemC netlist.

You can also set this option using the “*options set Output SysC_Namespace <string>*” command.

- **ROM Format**

This option specifies whether generated ROM models have initialization data internally or externally. It is applicable for all ROMs in the design, including both synchronous and asynchronous ROMs. You can also set this option using the “[options set Output ROMFormat <inline_or_fileio>](#)” command.

- **infile:** Include rom-initialization data in the generated ROM model. The default models are synthesizable by both ASIC and FPGA synthesis tools. They also simulate with all our partner industry-standard simulation tools like ModelSim and NCSim.
- **fileio:** This setting directs Catapult to write out ROM models that read the rom-initialization data from a file. This is applicable for both Verilog and VHDL. For Verilog, the model is implemented using the \$readmemb() call. For VHDL, the model is implemented using TEXTIO routines.

The File I/O based models are black-boxed for ASIC because Design Compiler does not support them for synthesis. These could be helpful if you use custom ROM flows as you can easily plug in their custom model. These models will still simulate correctly.

For FPGA, these models have been tested with the Precision RTL synthesis tool. These models generally tend to improve synthesis runtime and decrease the memory footprint compared to our default models.

The following example shows the different output generated by Catapult based on the **ROM Format** setting. The example uses the “dct” example design in the Catapult software tree (\$MGC_HOME/shared/examples/catapult/dct) and the output language is Verilog.

When **ROM Format** is set to “inline,” the ROM file will contain initialization code similar to the following:

```
// pragma attribute mem MEM_INIT_BLOCK rom_init_blk
always@(addr_f) begin: rom_init_blk
    mem[0] = 10'b0101101010;
    mem[1] = 10'b0101101010;
    mem[2] = 10'b0101101010;
    .
    .
    .
    mem[62] = 10'b0100011100;
    mem[63] = 10'b1110011101;
end
```

When **ROM Format** is set to “fileio,” the ROM file calls \$readmemb() as follows:

```
localparam datafile_name =
"rtl_dct_proc_mgc_rom_sync_regin_4_64_10_1_0_0_1_1_0_0_0_0.dat";
initial $readmemb(datafile_name, mem);
```

- **Netlist Names Truncating**

The “Netlist Names Truncating” feature truncates variable names in RTL output files to the specified length. The factory default length is 255 characters. Truncated variable names are kept unique by including a unique 6 digit number at the end of the name. The algorithm preserves the first [Max Name Length - 6] characters of the original name and replaces the rest with the 6 digit number. For example, if the maximum name length is set to 20, then the variable name “st_fir_filter2_proc_loop_1” will become “st_fir_filter2xxxxxx, where “xxxxxx” is a unique 6 digit number.

- **Do Not Modify Names**

Enable/disable the “Netlist Names Truncating” feature. You can also set this option using the “*options set Output DoNotModifyNames <true_or_false>*” command.

- **Or Set Max Name Length**

Specify the maximum length of variables names. The value must be an integer greater than or equal to sixteen. You can also set this option using the “*options set Output MaxNameLength <integer>*” command.

- **Netlist Basenames (<state> <basename> ...)**

The “Netlist Basenames” option sets the default basenames of files generated during the specified stage of the Catapult work flow. Specify one or more stage-basename pairs in the “Basename” field. If a pair is omitted, the factory default basename for that stage will be used.

You can also set this option using the “*options set*” command as follows:

```
options set Output Basename { <stage> <basename> ... }
```

<stage> is one of [compile | architect | allocate | schedule | dpfsm | extract]

<basename> is a user-defined string. This string can include the macro “\${ENTITY}” to dynamically insert the name of the top-level entity in the basename.

For more information, refer to “[Configurable Basenames for Generated Files](#)” on page 146.

- **Sub-Block Name Prefix**

The SubBlockNamePrefix option allows you to specify a string that will be prefixed to every block name in the generated HDL, except the top-level entity for the design. For example, Verilog modules, VHDL entities (except the top-level entity), packages, and other objects.

If the PackageOutput option ([Package Output in Solution dir](#)) is also enabled, then block names in the files copied into the directory by that option are also prefixed. You can also set this option using the “*options set Output SubBlockNamePrefix <string>*” command.

The option can be set at any point in the flow and any netlist file generated after that point will use this option.

The following example sets the SubBlockNamePrefix string to “version_2__”:

```
options set Output SubBlockNamePrefix version_2__
```

The prefix would appear in the generated rtl.vhdl file as follows (blue highlight):

```
ENTITY version_2__matmult2_proc_fsm IS
  ...
END version_2__matmult2_proc_fsm;
```

Any prefix string that results in the creation of an illegal VHDL name will automatically be changed to a legal name. This applies to both VHDL and Verilog files. Table 4-2 lists the set of illegal characters and the literal string that are substituted for each one.

Table 4-2. Replacement Strings for Illegal Characters in HDL Identifiers

Illegal Character	Replacement String	Illegal Character	Replacement String	Illegal Character	Replacement String
+	plus	>	greater	,	comma
-	dash	%	percent	(lprn
*	star	^	carrot)	rprn
/	slash		or	{	lcrl
=	equal	&	and	}	rcrl
!	not	~	tilde	[lsqr
<	less	.	dot]	rsqr

Additional rules for legalizing identifiers are:

- If the last character of an identifier is an underscore (‘_’), the underscore is deleted.
- If the first character of an identifier is an underscore (‘_’), the string “mgc” will be prefixed to the identifier.
- If the first character of an identifier is a numeric character (0-9), the string “mgc_” will *replace* the numeric character.
- If an identifier is a keyword, the string “_1” is appended to the identifier.

Note



The SubBlockNamePrefix option does not apply to SystemC files.

• **Schematic Output**

When this option is enabled Catapult will generate a schematic file as part of the standard set of output files. Disabling this option may speed up the flow for large designs. You can also set this option using the “*options set Output RTL*Schem

`<true_or_false>`

” command. For information about the schematics generated by Catapult and the features of the schematic window, refer to “[Using the Schematic Viewer Window](#)” on page 148.

- **VHDL Settings**
 - **Architecture Name:** Enter an architecture name for the VHDL output file. The default is an empty string, which causes Catapult to use the solution name as the architecture name. You can also set this option using the “*options set Output Architecture <string>*” command.

Limitation of SystemC Netlists During Simulation

The SystemC data types sc_int/sc_uint and sc_bigint/sc_bignum are initialized to 0 by default. The same applies to their base class sc_int_base, sc_uint_base, sc_signed and sc_unsigned as well. On the other hand, VHDL SIGNED and UNSIGNED data types can have unknown or tri-state values. Because of this, anytime an uninitialized variable is used in a design, the generated SystemC netlist may not have the same simulation behavior as the VHDL and Verilog netlists. We recommend that variables not be read before they are initialized. And if the design requires this behavior, the SystemC netlist is not expected to give consistent simulation results with VHDL and Verilog till the design stabilizes.

Consider the following design. The C++ source has two uninitialized variables oa1 and oa2 that are read in statement 1.

```
#pragma hls_design top
void sinit(int a_in, int b_in, int NT *out1, int *out2)
{
    int oa1;
    int oa2;

    for ( int i = 0; i < 2; i++ ) {
        oa1 += oa2;----- (1)
        oa2 = oa1 + ++a_in + b_in;----- (2)
        *out1 = oa1; ----- (3)
        *out2 = oa2; ----- (4)
    }
}
```

In the VHDL netlist, these two variables are of type STD_LOGIC_VECTOR(31 downto 0) and are both initialized to “---...---”;

```
oa1 := STD_LOGIC_VECTOR' ( "-----" );
oa2 := STD_LOGIC_VECTOR' ( "-----" );
```

The statement (1) translates to the following where the CONV_SIGNED function and the SIGNED cast preserves unknown values:

```
oa1_1 := STD_LOGIC_VECTOR( CONV_SIGNED(SIGNED(oa1) + SIGNED(oa2), 32) );
```

In the SystemC netlist these two variables are of type sc_lv<32> and initialized to “xx..xx” - ensuring a behavior consistent with VHDL. However, the inconsistency comes in with the addition operation. The CONV_SIGNED and SIGNED function calls (implemented in funcs.h) returns the SystemC data types sc_(un)signed/sc(u)int_base that have a default value of 0. So while the VHDL variables oa1 and oa2 have unknown values when they are added in the first iteration, SystemC variables oa1 and oa2 are 0.

Constraints Editor Options

The Constraints Editor dialog box lets you set the default view type.

- **Default Window Layout**

Specify the default type of window to be used for Constraints Editor. Select either tabbed document window or a free floating window. You can also set this option using the “*options set ConstraintsEditor WindowLayout <Floating_or_Tabbed>*” command.

- **Default View**

The Constraints Editor window displays a graphical view of the design in either of two formats, a tree view or a block diagram. Use the Default View option to set the default view format. For more information about each view type, refer to “[Specifying Architectural Constraints](#)” on page 104 and “[Block Diagram Constraint Editor](#)” on page 124. You can also set this option using the “*options set ConstraintsEditor DefaultView <TreeView or BlockDiagram>*” command.

Text Editor Options

The Text Editor dialog box lets you set options for the DesignPad editor. You can override the default settings by using the popup menu in the DesignPad editor window.

- **Default Window Layout**

Specify the default type of window to be used for Text Editor documents. Select either tabbed document window or a free floating window. You can also set this option using the “*options set TextEditor WindowLayout <Floating_or_Tabbed>*” command.

- **Show Code Browser**

Specify whether or not the Code Browser feature of the DesignPad editor is visible by default. The code browser allows you to move to selected sections in the code. You can also set this option using the “*options set TextEditor CodeBrowser <true_or_false>*” command.

- **Show Line Numbers**

Specify whether or not the DesignPad editor displays line numbers by default. You can also set this option using the “*options set TextEditor LineNumbers <true_or_false>*” command.

- **Enable Outline Mode**

Specify whether or not the Outline mode is enabled by default in the DesignPad editor. Outline mode provides a hierarchical view of the text in the editor. It allows you collapse/expand sections of the document independently. Outline mode is indicated by plus or minus icons to the left of a text block. Click a plus/minus icon to expand/collapse a block. You can also set this option using the “*options set TextEditor OutlineMode <true_or_false>*” command.

Schedule Viewer Options

The Schedule Viewer dialog box lets you set the default window layout, view type and color options for the schedule viewer.

- **Default Window Layout**

Select the default type of window to be used for the Schedule viewer. Select either tabbed document window or a free floating window. You can also set this option using the “*options set ScheduleViewer WindowLayout <Floating_or_Tabbed>*” command.

- **Default Schedule View**

Select the default format of the Gantt chart, either **Compact** or **Expanded**. For more information about these formats, refer to “[Scheduling the Design](#)” on page 127. You can also set this option using the “*options set ScheduleViewer DefaultScheduleView <Compact_or_Expanded>*” command.

- **Default visibility settings for dependency types**

When an operation is selected in the Gantt chart, its dependency relationships with other operations are displayed using arrows. The visibility of individual dependency types can be toggled interactively from the popup menu in the Schedule window. Unchecking a dependency type in this field will make invisible by default.

- **Default color settings**

Click on a listed object, such as ClockColor, select a new color, then click **OK**. You can also set this option using the “*options set*” command followed by these arguments:

```
BackgroundColor | ForegroundColor | PreClockColor | ClockColor  
LoopColor | LoopOutlineColor | LoopHeaderColor | LoopCollapseColor  
OverheadColor | OpTextColor | RealOpColor | RealOpOutlineColor  
PseudoOpColor | PseudoOpOutlineColor | MobilityOutlineColor  
SelectColor | SelectOutlineColor | ConstraintColor  
DataDependencyColor | IOModeDependencyColor  
ControlDependencyColor | UserConstraintColor
```

Schematic Viewer Options

The Schematic Viewer dialog box lets you set the default window layout and display options for the schematic viewer. For a detailed discussion of the Schematic window features, refer to “[Using the Schematic Viewer Window](#)” on page 148.

- **Default Window Layout**

Select the default type of window to be used for the Schematic viewer. Select either tabbed document window or a free floating window. You can also set this option using the “[*options set SchematicViewer WindowLayout <Floating_or_Tabbed>*](#)” command.

- **Multi-Page Schematics**

Uncheck the box to display the entire schematic on a single page. This may result in long routines for large schematics. You can also set this option using the “[*options set SchematicViewer MultiPage <true_or_false>*](#)” command. Multi-page view can be toggled on and off from the popup menu in the schematic viewer window.

- **Show Instance Name**

Check the box to show instance names on the schematic. When this option is unchecked and instance names are hidden, hover the cursor over an instance to see its name. You can also set this option using “[*options set SchematicViewer showinstname <true_or_false>*](#)”.

- **Show Pin Name**

Check the box to show pin names on the schematic. When this option is unchecked and pin names are hidden, hover the cursor over a pin to see its name. You can also set this option using the “[*options set SchematicViewer showpinname <true_or_false>*](#)” command.

- **Highlight Cross-Reference Objects**

Uncheck the box to disable highlighting of cross-references. You can also set this option using the “[*options set SchematicViewer HighlightXRefs <true_or_false>*](#)” command. Cross-reference highlighting can be toggled on and off from the popup menu in the schematic viewer window.

- **Change a Color**

Click on a listed object, such as Netcolor, select a new color, then click **OK**. You can also set this option using the “[*options set*](#)” command followed by these arguments:

```

backgroundcolor rubberbandcolor | netcolor | buscolor | overlapcolor
boxcolor0 boxcolor1 | boxcolor2 | boxcolor3 | boxpincolor
boxinstcolor portcolor | rippercolor | attrcolor | framecolor
objectgrey objecthighlight0 | objecthighlight1 | objecthighlight2
objecthighlight3 | objecthighlight4 | objecthighlight5
objecthighlight6 | objecthighlight7 | objecthighlight8
objecthighlight9 | objecthighlight10 | objecthighlight11
objecthighlight12 | objecthighlight13 | objecthighlight14
objecthighlight15 | objecthighlight16 | objecthighlight17
objecthighlight18 | objecthighlight19

```

Toolkits Options

Use the Toolkits dialog box to add custom toolkits to the Toolkits window. Refer to “[Adding Custom Toolkits to the Toolkits Window](#)” on page 84 for more information.

- **Configuration Files**

This list of XML files will be loaded into the Toolkits window when it is invoked. These configuration files define custom toolkits that will be added to the default set of toolkits. Use the **Add**, **Edit** and **Delete** buttons to modify the list. You can also set this option using the “*options set Toolkit ConfigurationFiles <list_of_files>*” command.

Flows Options

The Flows dialog box allows you to configure the search path for flow files, and the initial enable/disable state for integrated tool flows. Refer to the section “[Flow Customization](#)” on page 609 for information about user-defined flows.

- **Flow Search Path**

Add and delete directory pathnames to user-defined flow package files. When a new project is created, Catapult scans all of the flow package files (.flo) in the Flows Search Path and constructs an index of the packages it finds. The paths are searched in the order in which they appear. If duplicate filenames are found, only the first one is indexed. The default Catapult flow packages are scanned prior to user-defined flow packages.

Use the **Add**, **Edit** and **Delete** buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order. You can also set this option using the “*options set Flows FlowSearchPath <list_of_paths>*” command.

- **Enable Flow on New Project**

Set the default enable/disable state for integrated flows. For each flow, select one of the following settings:

Table 4-3. Default Flow Settings

Setting	Description
<blank>	Flow is enabled <i>on demand</i> .
“yes”	Flow is always enabled.
“no”	Flow is always disabled.

ModelSim Flow Options

The ModelSim dialog box lets you set invocation options for the ModelSim simulation tool. ModelSim is the default simulation tool.

- **Path**
Specifies the path checked first for a ModelSim executable. If none is found, then on Unix, the standard \$PATH variable is checked and on Windows, the registry is checked followed by the standard path. You can also set this option using the “[options set Flows/ModelSim Path](#)” command.
You must specify the MODEL_TECH variable or an absolute pathname.
- **Command-Line Flags**
Specifies optional flags used to invoke ModelSim. You can also set this option using the “[options set Flows/ModelSim Flags](#)” command.
- **Default Radix for WAVE window**
Specifies the radix used when adding signals to the wave view while setting up simulation. You can also specify the radix from the ModelSim GUI or the “[options set Flows/ModelSim RADIX <string>](#)” command.
- **Native AC datatypes display**
Displays Algorithmic C (AC) datatypes as native datatypes in the waveform viewer. Requires ModelSim version 6.3c or newer. This feature is enabled by the ModelSim compiler directive named SC_INCLUDE_MTI_AC. You can also set this option using the “[options set Flows/ModelSim MSIM_AC_TYPES <true_or_false>](#)” command.
- **VHDL compilation options**
Specifies VHDL compiler options. VHDL compiler options can also be set with the “[options set Flows/ModelSim VCOM_OPTS <list_of_opts>](#)” command.
- **Verilog compilation options**

Specifies Verilog compiler options. Verilog compiler options can also be set with the “*options set Flows/ModelSim VLOG_OPTS <list_of_opts>*” command.

- **SystemC compilation options**

Specifies SystemC compiler options. SystemC compiler options can also be set with the “*options set Flows/ModelSim SCCOM_OPTS <list_of_opts>*” command.

- **Simulation options**

Specifies simulation options. Simulation options can also be set with the “*options set Flows/ModelSim VSIM_OPTS <list_of_opts>*” command.

- **Additional options for gate simulation**

Specifies additional options for gate simulation only. These options are added to the “Simulation options” specified above. Gate simulation options can also be set with the “*options set Flows/ModelSim GATE_VSIM_OPTS <list_of_opts>*” command.

- **Default modelsim ini file path**

Specifies the path to a custom modelsim.ini file. The modelsim.ini path can also be specified with the “*options set Flows/ModelSim DEF_MODELSIM_INI <path>*” command.

- **Show ModelSm LIST window**

Automatically populates the ModelSim List window and Waveform window. This may be useful for identifying delta-cycle simulation issues in your design. The List window can also be set up with the “*options set Flows/ModelSim SHOW_LIST <true_or_false>*” command.

- **Optional dofile name to run on GUI startup**

Specifies a Tcl script to use when invoking ModelSim in GUI mode. The script path can be absolute or relative to the project directory. This script can also be specified with the “*options set Flows/ModelSim MSIM_DOFILE <path>*” command.

- **Enable the old Simulation flow**

Enables the previous simulation flow in Catapult. The old simulation flow behavior loads the compiled RTL netlist into ModelSim for verification. The supported simulation flow is SCVerify which verifies the compiled RTL netlist against the input C/C++. For more information, see “[C/C++ Designs and SCVerify](#)” on page 516. The old simulation flow can also be enabled with the “[*options set Flows/ModelSim ENABLE_OLD_MSIM_FLOW <true_or_false>*](#)” command.

Precision RTL Flow Options

This dialog box lets you set invocation options for the Precision RTL Synthesis tool.

- **Path**

Specifies the path to the Precision RTL Synthesis executable. If none is found, then on Unix, the standard \$PATH variable is checked. On Windows, the registry is checked followed by the standard path. You can also set this option using the “[*options set Flows/Precision Path <path>*](#)” command.

You must specify the PRECISION_HOME variable or an absolute pathname.

- **Command-line Flags**

Specifies command line switches for the Precision RTL Synthesis executable. You can also set this option using the “[*options set Flows/Precision Flags <list_of_flags>*](#)” command.

- **Add IO Pads**

Specifies Precision RTL Synthesis option: Optimization - Add IO Pads. You can also set this option using the “[*options set Flows/Precision addio <true_or_false>*](#)” command.

- **Run Retiming**

Specifies Precision RTL Synthesis option: Retiming. You can also set this option using the “[*options set Flows/Precision retiming <true_or_false>*](#)” command.

- **Run Integrated Place and Route in Precision**

Enables the Precision RTL Synthesis option to automatically place and route the design. You can also set this option using the “[*options set Flows/Precision run_pnr <true_or_false>*](#)” command.

- **Run Next Generation User Interface**

Enable this option to run Precision RTL Synthesis with its new graphical user interface. When this option is disabled, the old GUI is used. You can also set this option using the “[*options set Flows/Precision newgui <true_or_false>*](#)” command.

- **RTL Plus**

Enable this option to run the RTL Plus version of Precision Synthesis. Precision RTL Plus performs physically aware synthesis. You can also set this option using the “[*options set Flows/Precision rtlplus <true_or_false>*](#)” command.

- **Generate EDIF output**

Outputs an EDIF netlist in addition to the netlist output in the format (VHDL/Verilog) of the source design.

- **Place and Route Install Path**

Specifies the path to the default place and route tool. You can also set this option using the “[*options set Flows/Precision PlaceAndRouteInstallPath <path>*](#)” command.

- **Gather detailed timing information**

Includes the following types of timing data in the Precision timing report:

- input to register
- register to register
- register to output

You can also set this option using the “[*options set Flows/Precision GatherDetailedTimingInformation <true_or_false>*](#)” command.

CXXAnalysis Flow Options

Sets the default values for the CXXAnalysis flow that performs static and runtime checks, as well as code coverage analysis, of your C++ design and testbench code. Refer to “[Integrated Code Analysis Flow](#)” on page 155 for more information about this flow.

- **Additional Include Directory Paths**

Specifies one or more paths to directories containing include files. Multiple paths should be separated by spaces. You can also set this option using the “[*options set /CXXAnalysis/INCL_DIRS <list_of_paths>*](#)” command.

- **Additional Link Library Paths**

Specifies one or more paths to search for libraries during compilation. Multiple paths should be separated by spaces. You can also set this option using the “[*options set /CXXAnalysis/LINK_LIBPATHS <list_of_paths>*](#)” command.

- **Additional Link Libraries**

Specifies one or more libraries for compilation. Multiple files should be separated by spaces. You can also set this option using the “[*options set /CXXAnalysis/LINK_LIBNAMES <list_of_names>*](#)” command.

- **Testbench Invocation Args**

Specifies command-line arguments to the testbench (i.e. the traditional "int argc, char *argv[]" parameters of "main"). You can also set this option using the "[*options set /CXXAnalysis\(INVOKE_ARGS <args_string>\)*](#)" command.

- **Autorun Static Analysis**

Enables automatic execution of the static analysis checks and reports generation when the CXXAnalysis flow runs. Default setting is enabled. If this option is disabled, static analysis does run unless you explicitly launch it. You can also set this option using the "[*options set /CXXAnalysis/SET_STATIC_ANALYSIS <true_or_false>*](#)" command.

- **Autorun Runtime Analysis**

Enables automatic execution of the runtime analysis checks and reports generation when the CXXAnalysis flow runs. Default setting is disabled. If this option is disabled, runtime analysis does run unless you explicitly launch it. You can also set this option using the "[*options set /CXXAnalysis/SET_RUNTIME_ANALYSIS <true_or_false>*](#)" command.

- **Autorun Coverage Analysis**

Enables automatic execution of the coverage analysis checks and reports generation when the CXXAnalysis flow runs. Default setting is disabled. If this option is disabled, coverage analysis does run unless you explicitly launch it. You can also set this option using the "[*options set /CXXAnalysis/SET_COVERAGE_ANALYSIS <true_or_false>*](#)" command.

- **STATIC_ANALYSIS_TOOL**

Specifies the tool used for static analysis.

- **RUNTIME_ANALYSIS_TOOL**

Specifies the tool used for runtime analysis.

- **COVERAGE_ANALYSIS_TOOL**

Specifies the tool used for coverage analysis.

DCPower Flow Options

Power analysis flow using the Synopsys Design Compiler. Refer to "["Power Analysis and Optimization"](#)" on page 583 for information about how to run the DCPower flow.

 **Note**
This flow requires a separate Synopsys license in order to Design Compiler to run the power optimization features.

- **Technology Libraries**

Specify one or more “.db” library files. This option is only needed if the Synopsys Design Compiler was not used to synthesize the RTL. You can also set this option using the “*options set Flows/DCPower dcpower_techlibs <list_of_paths>*” command.

- **SA Propagation Effort**

Specifies Design Compiler’s default effort level when propagating switching activity. Valid settings are low, medium (default) and high. The specified value is assigned to the DC variable named power_sa_propagation_effort. Refer to Synopsys documentation for a complete description of the variable.

You can also set this option using the “*options set Flows/DCPower sa_propagation_effort <low_medium_high>*” command.

Design Compiler Flow Options

This dialog box lets you set invocation options for the Synopsys Design Compiler tool. These options affect the DC scripts generated by Catapult.

- **Library Search Path**

This search path will be used to locate DC libraries. When DC is invoked, it searches the specified paths in addition to its default library search paths. Use the **Add**, **Edit** and **Delete** buttons to modify the paths in the list. You can also set this option using the “*options set Flows/DesignCompiler SearchPath <list_of_paths>*” command.

- **Executable Path**

The path to the bin directory containing the Design Compiler executable. This search path will be checked first, and if the executable is not found, the standard path will be checked. Use the Browse key to locate the desired path. You can also set this option using the “*options set Flows/DesignCompiler Path <path>*” command.

- **Design Compiler Executable**

This option specifies the name of the Design Compiler tool to invoke. The default tool is dc_shell. You can also set this option using the “*options set Flows/DesignCompiler ShellExe <string>*” command.

- **Additional compile options**

Command line switches for the Design Compiler executable. You can also set this option using the “*options set Flows/DesignCompiler CompileOpts <list_of_options>*” command.

- **Command-Line Mode**

Enter the environmental mode in which the Design Compiler executable will run. The supported modes are **dctcl** and **dcsh**. You can also set this option using the “*options set Flows/DesignCompiler ShellType <mode>*” command.

- **Command-Line Flags**

Enter the command line switches for the Design Compiler executable. You can also set this option using the “*options set Flows/DesignCompiler Flags <list_of_flags>*” command.

- **License Server**

List of license servers for Design Compiler licenses. Catapult assigns the specified list to the FLEXnet environment variable SNPSLMD_LICENSE_FILE prior to launching DC. Refer to the *FLEXnet Licensing End User Guide* for information about the syntax of FLEXnet environment variables. You can also set this option using “*options set Flows/DesignCompiler LicenseServer <list_of_servers>*” command.

- **Design License to check out**

Specifies a list of Synopsys DesignWare license features to be obtained. The list items are space separated. If the value is set, it is passed directly to the DC command “get_license” in the generated DC script so the specified license features are checked out at the beginning of synthesis. (They are held until the remove_license command is used or until the program is exited or until the DC shell is closed.) Refer to the license key file at your site to determine which licensed features are available.

You can also set this option using “*options set Flows/DesignCompiler GetSynLicense {<list_of_licenses>}*” command.

This option overrides any default settings defined in Catapult libraries. For more information about DesignWare settings in Catapult libraries, refer to “[The Library Options Tab](#)” in the *Catapult C Library Builder User's and Reference Manual*.

- **Design License not to use**

Specifies a list of DesignWare licenses that the DC tool is not allowed to use. The list items are space separated. The list is assigned to the DC variable “synlib_dont_get_license” in the generated DC script.

You can also set this option using “*options set Flows/DesignCompiler DontGetSynLicense {<list_of_licenses>}*” command.

This option overrides any default settings defined in Catapult libraries. For more information about DesignWare settings in Catapult libraries, refer to “[The Library Options Tab](#)” in the *Catapult C Library Builder User's and Reference Manual*.

- **Design License to wait for**

Specifies a list of DesignWare licenses that the DC tool should wait for if they are temporarily unavailable. The list items are space separated. The value of this variable is assigned to the DC variable “synlib_wait_for_design_license” in the generated DC script.

You can also set this option using “[*options set Flows/DesignCompiler WaitForSynLicense {<list_of_licenses>}*](#)” command.

This option overrides any default settings defined in Catapult libraries. For more information about DesignWare settings in Catapult libraries, refer to “[The Library Options Tab](#)” in the *Catapult C Library Builder User’s and Reference Manual*.

- **Output netlist name**

This option specifies the basename of the netlist generated by the Design Compiler. The filename extension is determined by the format of the netlist (.ddc, .vhdl or .v). See “[Output netlist format](#)” below for more information about setting the netlist format type. You can also set this option using the “[*options set Flows/DesignCompiler OutNetlistName <string>*](#)” command.

- **Output netlist format**

This option directs the Design Compiler to generate its netlist in the specified format. Use the drop-down list to select a format. The format choices are “ddc,” “verilog” or “vhdl.” You can also set this option using the “[*options set Flows/DesignCompiler OutNetlistFormat <string>*](#)” command.

- **Max Loop Iterations**

The value specified for this option will override the Design Compiler default maximum loop iteration count. Use this option if you see the following error message from Design Compiler:

```
"Loop exceeded maximum iteration limit (ELAB-900)"
```

The Design Complier script generated by Catapult sets the Design Compiler variable “hdlin_while_loop_iterations” to the specified value. The default is value is 1000. You can also set this option using the “[*options set Flows/DesignCompiler MaxLoopIterations <integer>*](#)” command.

- **Enable Power Reporting**

Directs the Design Compiler to generate a power analysis report. You can also set this option using the “[*options set Flows/DesignCompiler EnablePowerReporting <true_or_false>*](#)” command.

- **Enable Clock Gating**

Directs the Design Compiler to perform clock gating algorithm. For more in formation about clock gating, refer to “[Clock Gate Insertion](#)” on page 589. You can also set this option using the “[*options set Flows/DesignCompiler EnableClockGating <true_or_false>*](#)” command.

- **Output File Folder Name**

Specifies the default name of the output file folder that appears in the Catapult GUI. The hierarchical path to the folder is “<solution_name>/Synthesis/<folder_name>”. You can

also set this option using the “*options set Flows/DesignCompiler FOLDERNAME <string>*” command. The factory default value for this field is “Design Compiler”.

This field will not appear unless the Design Compiler flow is enabled. The Design Compiler flow is enabled automatically when ASIC technology is selected during the Setup Design task.

MATLAB Flow Options

Refer to “MATLAB/Simulink Flow (C/C++)” on page 574 for information about how to use the MATLAB/Simulink flow.

- **Matlab Installation Directory:**

Path to the top-level directory containing the Mathworks tools. The default setting relies on the value of the MATLABROOT environment variable. That environment variable must be set in order for the default option setting to work. To set the “Matlab Installation Directory” option from the command line interface, use the “*options set Flows/Matlab MATLAB_ROOT <path>*” command.

- **Simulink Model Directory:**

Optional output path for the compiled S-block/M-block models. By default, Catapult saves these files in the Solution directory. You can also set this option using the “*options set Flows/Matlab SIMULINK_DIR <path>*” command.

NCSim Flow Options

The NCSim dialog specifies invocation options for the NCSim simulation tool.

- **Path to NCSim installation directory**

Specifies the path to the NCSim executable. If none is found, the standard path is checked. The NCSim path can also be specified with the “*options set Flows/NCSim NC_ROOT <path>*” command.

- **VHDL compilation options**

Specifies default VHDL compiler options. VHDL compiler options can also be specified with the “*options set Flows/NCSim NCVHDL_OPTS <list_of_opts>*” command.

- **Verilog compilation options**

Specifies default Verilog compiler options. Verilog compiler options can also be specified with the “*options set Flows/NCSim NCVLOG_OPTS <list_of_opts>*” command.

- **SystemC compilation options**

Specifies default SystemC compiler options. SystemC compiler options can also be specified with the “*options set Flows/NCSim NCSC_OPTS <list_of_opts>*” command.

- **Elaboration options**

Specifies default elaboration options. Elaboration options can also be specified with the “*options set Flows/NCSim NCELAB_OPTS <list_of_opts>*” command.

- **Simulation options**

Specifies default simulator options. Simulator options can also be set with the “*options set Flows/NCSim NCSIM_OPTS <list_of_opts>*” command.

- **Simulation -timescale**

Specifies the simulation resolution time. The timescale is set to a reasonable default value that allows NCSim to handle conflicting and/or missing timescale settings that might be embedded in Verilog source files. Change the timescale if your RTL IP models, such as memory models, require a different setting. The timescale can also be specified with the “*options set Flows/NCSim NCSIM_TIMESCALE <string>*” command.

- **GCC Version to use with NCSIM**

Specifies the version of the GCC compiler to use with NCSIM.

Novas Flow Options

These options configure the version and platform settings for the Novas FsdbWriter tool. The Novas tool flow can be used for generating FSDB files for the SpyGlassPower flow. For more information about how the following options are used in the Novas tool, refer to Novas document “*Linking Novas Files with Simulators to Enable FSDB Waveform Dumping*” (\$NOVAS_INST_DIR/doc/linking.pdf).

- **Path to Novas installation directory**

The path to the top-level directory of the Novas installation tree. By default, the flow reads the value of the NOVAS_INST_DIR environment variable. Enter a valid path if the environment variable is not set or to override the variable. You can also set this option using the “*options set Flows/Novas NOVAS_INST_DIR <path>*” command.

- **Novas platform name**

The name of the platform on which Catapult is running. Use the Novas naming convention as found in the “\$NOVAS_INST_DIR/platform” directory. For example, “LINUX” or “SOL7_32bit”. You can also set this option using the “*options set Flows/Novas NOVAS_PLATFORM <string>*” command.

- **Version of the Novas PLI/FLI for Modelsim**

Specify one of the “modelsim_*” directory names in the “\$NOVAS_INST_DIR/share/PLI” directory. You can also set this option using the “*options set Flows/Novas NOVAS_MSIM_PLI <string>*” command.

- **NCSim version of Novas libraries**

Specify one of the “nc*” directory names in the “\$NOVAS_INST_DIR/share/PLI” directory. You can also set this option using the “*options set Flows/Novas NOVAS_NCSIM_VER <string>*” command.

- **Version of the Novas PLI for NCSim**

Specify one of the directory names in the “\$NOVAS_INST_DIR/share/PLI/systemc” directory. You can also set this option using the “*options set Flows/Novas NOVAS_NCSIM_PLI <string>*” command.

- **Version of the Novas VHPI for NCSim**

Specify one of the directory names in the “\$NOVAS_INST_DIR/share/PLI” directory. You can also set this option using the “*options set Flows/Novas NOVAS_NCSIM_LDV <string>*” command.

- **Version of the Novas FSDB writer for NCSim**

Specify one of the directory names in the “\$NOVAS_INST_DIR/share/FsdbWriter” directory. You can also set this option using the “*options set Flows/Novas NOVAS_NCSIM_FSDB <string>*” command.

OSCI Flow Options

The OSCI dialog box specifies options for the OSCI simulation tool.

- **Path to SystemC include**

Specifies the path to the SystemC include directory used by the OSCI simulator. By default, the SystemC include directory installed in the Catapult software tree is used. The include path can also be set with the “*options set Flows/OSCI SYSTEMC_INCLUDE <path>*” command.

- **Path to SystemC lib**

Specifies the path to the SystemC “lib” directory used by the OSCI simulator. The OSCI lib directory name has the form “lib-<architecture>” (such as “lib-linux” or “lib-solaris”). By default, the SystemC lib directory installed in the Catapult software tree is used. The SystemC lib path can also be set with the “*options set Flows/OSCI SYSTEMC_LIB <path>*” command.

- **OSCI Compiler Flags**

Specifies default OSCI compiler flags. You can also set this option with the “*options set Flows/OSCI COMP_FLAGS <list_of_flags>*” command.

- **Enables the use of a 32 bit compiler**

Forces the use of the 32-bit compiler even when a 64-bit compiler is available. The 32-bit compiler can also be set with the “[*options set Flows/OSCI USE_32BIT_COMPILER*](#)” command.

- **GUI executable for GDB debugger**

Specifies the pathname to the interactive GUI for the GDB debugger. This GUI (typically DDD) can be used to perform interactive simulation of the design. The pathname to the GUI can also be set with the “[*options set Flows/OSCI GDBGUI*](#)” command.

PowerPlay Flow Options

Power analysis flow using the Altera PowerPlay tool. Refer to “[Power Analysis and Optimization](#)” on page 583 for information about how to run the PowerPlay flow.

- **QUARTUS_ROOTDIR**

The path to the directory containing the PowerPlay executable. By default, the flow reads the value of the QUARTUS_ROOTDIR environment variable. Enter a valid path if the environment variable is not set or to override the variable. You can also set this option using the “[*options set Flows/PowerPlay QUARTUS_ROOTDIR <path>*](#)” command.

RTLCompiler Flow Options

Set the default values for the RTL Compiler flow to be used in the Catapult C Synthesis tool.

- **Library Search Path**

Add and delete directory pathnames to directories containing RTL Compiler libraries. Use the **Add**, **Edit** and **Delete** buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order.

You can also set this option using the “[*options set Flows/RTLCompiler SearchPath <list_of_paths>*](#)” command.

- **Executable Name**

This option specifies the name of the RTL Compiler tool to invoke. The default tool is rc. You can also set this option using the “[*options set Flows/RTLCompiler ShellExe <string>*](#)” command.

- **Executable path**

Full path to the directory containing the RTL Compiler executable. Click the Folder icon to browse to the directory where software is installed. You can also set this option using the “[*options set Flows/RTLCompiler Path <path>*](#)” command.

- **Command-Line Flags**

Specify any additional command line flags to be passed to RTL Compiler. You can also set this option using the “[*options set Flows/RTLCompiler Flags <list_of_flags>*](#)” command.

- **License Server**

List of license servers for the RTL Compiler licenses. List items are separated by spaces. Catapult assigns the specified list to the FLEXnet environment variable CDS_LIC_FILE prior to launching the RTL Compiler. Refer to the [*FLEXnet Licensing End User Guide*](#) for information about the syntax of FLEXnet environment variables. You can also set this option using “[*options set Flows/RTLCompiler LicenseServer <list_of_servers>*](#)” command.

- **Output netlist name**

This option specifies the basename of the netlist generated by the RTL Compiler. The filename extension is determined by the format of the netlist. See “Output netlist format” below for information about setting the netlist format type. You can also set this option using the “[*options set Flows/RTLCompiler OutNetlistName <string>*](#)” command.

- **Output netlist format**

This option directs the RTL Compiler to generate its netlist in the specified format. Use the drop-down list to select a format. Currently, Verilog is the only format supported. You can also set this option using the “[*options set Flows/RTLCompiler OutNetlistFormat <string>*](#)” command.

- **Max Loop Iterations**

The value specified for this option will override the RTL Compiler default maximum loop iteration count. Use this option if you see the following error message from Design Compiler:

```
"Loop exceeded maximum iteration limit (ELAB-900)"
```

The RTL Complier script generated by Catapult sets the RTL Compiler variable “hdl_max_loop_limit” to the specified value. The default is value is 1000.

You can also set this option using the “[*options set Flows/RTLCompiler MaxLoopIterations <integer>*](#)” command.

- **Enable Clock Gating**

Directs the RTL Compiler to perform clock gating algorithm. For more in formation about clock gating, refer to [**“Clock Gate Insertion”**](#) on page 589. You can also set this

option using the “*options set Flows/RTLCompiler EnableClockGating <true_or_false>*” command.

- **Enable Power Reporting**

Directs the RTL Compiler to generate a power analysis report. You can also set this option using the “*options set Flows/RTLCompiler EnablePowerReporting <true_or_false>*” command.

SCVerify Flow Options

For more information, see “[SCVerify Flow](#)” on page 515.

- **Reset Duration (cycles)**

Specifies the number of clock cycles to assert the reset signal. This is a floating point value so that you can create reset events that are asynchronous to the clock edges. You can also set this option using the “*options set Flows/SCVerify RESET_CYCLES <number>*” command.

- **Synchronize all resets**

This option is useful only in the context of a multi-clock design, and is therefore applicable only to the Catapult SL product. In multi-clock designs, each clock domain has a reset signal. The “Reset Duration” option above (RESET_CYCLES) specifies how many clock cycles the reset should be asserted. Since each clock has a different clock period, the resets will be deasserted at different times, though all at the same relative number of (their) clock cycles. This option overrides the individual reset durations and forces all resets to be deasserted at the same time (at RESET_CYCLES * slowest-clock-period). You can also set this option using the “*options set Flows/SCVerify SYNC_ALL_RESETS <true_of_false>*” command.

- **Default Stack Size**

If your design involves many large arrays of data, you may need to increase the default stack size for the SystemC modules by specifying a larger value in this option. You can also set this option using the “*options set Flows/SCVerify TB_STACKSIZE <number>*” command.

- **Testbench Invocation Args**

If you need to pass command-line arguments to the testbench (i.e. the traditional “int argc, char *argv[]” parameters of “main”), then specify them in this option. They will be passed through ModelSim and be available as local data in “testbench::main()”. You can also set this option using the “*options set Flows/SCVerify INVOKE_ARGS <args_string>*” command.

- **Additional args for Replay invocation**

For user-defined testbenches, this field allows you to specify input arguments for the testbench “main” function. The specified arguments do affect any of the Catapult generated files.

- **Enable C Debug Environment In ModelSim**

When this option is enabled, ModelSim will run with its debug environment active. In addition, the SCVerify flow automatically sets an initial breakpoint in the reset function and provides a GUI dialog box that allows you to dynamically specify which set of variables to report on. You can also set this option using the “*options set Flows/SCVerify MSIM_DEBUG <true_or_false>*” command.

- **Abort Simulation When Error Count Reaches**

Automatically terminate a simulation if the specified number of simulation errors occur. When a simulation is aborted, a corresponding message is displayed in the Catapult transcript. The default setting of zero means simulation will not abort regardless of the number of errors.

Simulation errors may be caused by incorrect synchronization between the testbench and the design. To ensure accurate input and output synchronization, enable the “**Transaction Done Signal**” option in the Setup Design dialog (or issue the command “*directive set -TRANSACTION_DONE_SIGNAL true*”).

You can also set this option using the “*options set Flows/SCVerify MAX_ERROR_CNT <integer>*” command.

- **Enable Automatic Deadlock Detection**

When this option is enabled, Catapult will abort the simulation run if a deadlock condition is detected. An error message is reported in the transcript. You can also set this option using the “*options set Flows/SCVerify DEADLOCK_DETECTION <true_or_false>*” command.

- **Additional Include Directory Paths**

Specify one or more paths to directories containing include files. List the paths separated by spaces. You can also set this option using the “*options set Flows/SCVerify INCL_DIRS*” command. For more information, refer to “[Linking External Libraries into SCVerify](#)” on page 522.

- **Additional Link Library Paths**

Specify one or more paths to library directories to be searched during compilation. List the paths separated by spaces. You can also set this option using the “*options set Flows/SCVerify LINK_LIBPATHS*” command. For more information, refer to “[Linking External Libraries into SCVerify](#)” on page 522.

- **Additional Link Libraries**

Specify one or more libraries to be used during compilation. List the files separated by spaces. You can also set this option using the “[*options set Flows/SCVerify LINK_LIBNAMES*](#)” command. For more information, refer to “[Linking External Libraries into SCVerify](#)” on page 522.

- **Use ModelSim/QuestaSim for Simulation**

When this option is enabled the SCVerify flow will generate makefiles to drive the ModelSim/QuestaSim simulation tools. You can also set this option using the “[*options set Flows/SCVerify USE_MSIM <true_or_false>*](#)” command.

- **Use OSCI for Simulation**

When this option is enabled the SCVerify flow will generate makefiles to drive the OSCI simulation tool. This option requires that Catapult is setup to generate SystemC output (see “[Output Options](#)” on page 189). You can also set this option using the “[*options set Flows/SCVerify USE_OSCI <true_or_false>*](#)” command.

- **Use NCSim for Simulation**

When this option is enabled the SCVerify flow will generate makefiles to drive the NCSim simulation tool. You can also set this option using the “[*options set Flows/SCVerify USE_NCSIM <true_or_false>*](#)” command.

Limitations of the NCSim flow:

- Supported platforms are Linux and Windows operating systems at this time
- Generated output includes cycle VHDL, RTL VHDL and RTL Verilog
- Precision RTL Synthesis is not supported
- TLM and SystemC are not supported

- **Use Vista for Simulation**

When this option is enabled the SCVerify flow will generate makefiles to drive the Vista™ simulation tool. You can also set this option using the “[*options set Flows/SCVerify USE_VISTA <true_or_false>*](#)” command.

- **Use VCS For Simulation**

When this option is enabled the SCVerify flow will generate makefiles to drive the VCS simulation tool. You can also set this option using the “[*options set Flows/SCVerify USE_VCS <true_or_false>*](#)” command. (See also “[VCS Flow Options](#)” on page 221.)

- **Turn off inputs when empty**

Enable this option if Catapult is generating distributed pipelines in order to provide “pipeline flushing” behavior. For more information, refer to “[Distributed Pipeline Synthesis](#)” on page 245 and “[Architectural Constraints on Loops](#)” on page 120.

When this option is enabled, the SCVerify environment will shut off each input resource (assuming it has full handshaking) as soon as the last input is read. The result is that as soon as all inputs have been read from the testbench, all of the handshake signals (the 'vz' signals) should then be inactive which should stall Catapult from reading those inputs. However, the remaining data in the pipeline should still come out even though the inputs have been stopped.

If distributed pipelining (pipeline flushing) is not enabled, then the design will stop (because the inputs are stopped) and no additional outputs will appear. When the “**Turn off inputs when empty**” option is disabled, the inputs will not be stalled, the design will continue to read whatever value was last placed on the input (with a warning from SCVerify indicating that the input FIFO has run empty) and the outputs will presumably come out assuming that the testbench iterates enough times.

You can also set this option using the “*options set Flows/SCVerify DISABLE_EMPTY_INPUTS <true_or_false>*” command.

- **Use Trace/Replay verification**

Enable this option for designs that model concurrency by using non-blocking ac_channel read methods. Refer to “[Non-Blocking Reads and Verification \(C/C++\)](#)” on page 543 for more information. You can also set this option using the “*options set Flows/SCVerify ENABLE_REPLAY_VERIFICATION <true_or_false>*” command.

- **Use design IDLE to synchronize transactions**

If the **IDLE_SIGNAL** directive is set on the design, enable this option so that SCVerify can utilize the IDLE signals to synchronize transactions between blocks. For more information about how idle signals are implemented in Catapult, refer to “[Idle Signal Insertion](#)” on page 592. You can also set this option using the “*options set Flows/SCVerify IDLE_SYNCHRONIZATION_MODE <true_or_false>*” command.

- **On output mismatch (golden vs DUT), print only the array elements that mismatch**

SLEC Formal Verification Flow Options

Refer to “[Formal Verification Flow](#)” on page 565 for information about the SLEC flow.

- **Path to SLEC bin directory**

This field is required. Specify the path to the “bin” directory in which the Calypto SLEC tool is installed. Include “bin” as the leaf of the path. You can also set this option using the “*options set Flows/SLEC SLEC_HOME <path>*” command.

- **Enable using FLOP mappings**

Enable this option to utilize the Flop mappings generated by Catapult. Refer to “[Flop Mappings for the SLEC Flow](#)” on page 572 for more information. You can also set this

option using the “`options set Flows/SLEC ENABLE_FLOP_MAPS <true_or_false>`” command.

- **Enable full proof verification mode**

This option toggles the SLEC verification mode. When checked, “Full Proof” mode is enabled. Otherwise “Find Error” mode is used by default. You can also set this option using the “`options set Flows/SLEC ENABLE_FULL_PROOF <true_or_false>`” command.

- **Max number of sim transactions**

The value specified in this field is assigned to the global variable “sim_max_transactions” in the Tcl script generated by Catapult. You can also set this option using the “`options set Flows/SLEC SIM_MAX_TRANSACTIONS <integer>`” command.

- **Specification design latency**

Specifies the latency of the specification design. Valid setting is any non-negative integer. Default setting is 0. The value specified in this field is assigned to the global variable “Spec_Latency” in the Tcl script generated by Catapult. You can also set this option using the “`options set Flows/SLEC SPEC_LATENCY <integer>`” command.

- **Specification design throughput**

Specifies the throughput of the specification design. Valid setting is any positive integer. Default setting is 1. The value specified in this field is assigned to the global variable “Spec_Throughput” in the Tcl script generated by Catapult. You can also set this option using the “`options set Flows/SLEC SPEC_TP <integer>`” command.

- **Override IMPLementation design latency**

Specifies the latency of the implementation design. Valid setting is any non-negative integer. Default setting is 0. The value specified in this field is assigned to the global variable “Impl_Latency” in the Tcl script generated by Catapult. You can also set this option using the “`options set Flows/SLEC OVERRIDE_IMPL_LATENCY <integer>`” command.

- **Override IMPLementation design throughput**

Specifies the throughput of the implementation design. Valid setting is any positive integer. Default setting is 0. The value specified in this field is assigned to the global variable “Impl_Throughput” in the Tcl script generated by Catapult. You can also set this option using the “`options set Flows/SLEC OVERRIDE_IMPL_TP <integer>`” command.

- **Override IMPLementation reset latency**

Specifies the reset latency of the implementation design. Valid setting is any positive integer. Default setting is 0. The value specified in this field is assigned to the global

variable “Impl_Rst_Length” in the Tcl script generated by Catapult. You can also set this option using the “*options set Flows/SLEC OVERRIDE_RESET_LATENCY <integer>*” command.

- **Generate hierarchical proofs**

When this option is enabled, the SLEC flow will automatically generate SLEC wrapper scripts for each hierarchical sub-block in the design and its corresponding RTL sub-module (in addition to the regular top-level wrapper/script generation). The original C source and the generated RTL for the full block are used in the sub-verification runs. Each sub-block/sub-module pair can be verified separately. The appropriate sub-function and sub-module are set as the top module in the wrappers. Refer to “[Generating Proofs for Hierarchical Blocks](#)” on page 572 for more information.

You can also set this option using the “*options set Flows/SLEC GENERATE_HIERARCHICAL_PROOFS <true_or_false>*” command.

- **Load ‘SPEC’ DB file (if exists)**

When this option is enabled, SLEC is directed to reuse an existing spec.db file (C portion). If the file is found, SLEC will generate a new one. If the option is disabled, SLEC will always regenerate the file. You can also set this option using the “*options set Flows/SLEC ENABLE_DB_FLOW <true_or_false>*” command.

- **Extended reset length**

This option allows you to artificially extend the duration of the reset state in SLEC to ensure that all blocks are well into their normal operating mode before SLEC starts to identify transaction intervals using simulation. You can also set this option using the “*options set Flows/SLEC EXTENDED_RESET_LENGTH <integer>*” command.

- **Maximum ac_channel size**

This option sets the size for all ac_channel objects in the C++ design. This must be set large enough to allow the C++ simulation to run without running into a deadlock. If overflow or underflow occur during the SLEC simulation phase, a message is issued.

You can also set this option using the “*options set Flows/SLEC MAXIMUX_AC_CHANNEL_SIZE <integer>*” command.

SOPC_Builder flow Options

For information about how to run the SOPC_Builder flow, refer to the examples provided in the Catapult toolkits (Help > Toolkits > Integration Flows > Altera SOPC_Builder Flow).

- **Catapult SOPC_Builder Component Repository**

Use this option to specify a directory where the SOPC_Builder flow can place copies of all of the files required to import the design into the Altera SOPC Builder. The specified path must resolve to an existing directory. The flow will not create the directory. You

can also set this option using the “[*options set Flows/SOPC_Builder SOPC_COMP_LIB <path>*](#)” command.

- **Use Precision RTL Synthesis**

When this option is enabled, the Precision RTL tool is used to synthesis the design and generate an EDIF netlist. Disable this option if you want to use the Altera synthesis tool. You can also set this option using the “[*options set Flows/SOPC_Builder PSR_RTL_SYN <true_or_false>*](#)” command.

Note

 The “Use Precision RTL Synthesis” option is enabled by default because the Altera synthesis tool does not properly infer the dual-port memories provided by Catapult.

SpyGlassPower Flow Options

Power analysis flow using the Atrenta SpyGlass-Power tool. Refer to “[Power Analysis and Optimization](#)” on page 583 for information about how to run the SpyGlassPower flow.

- **Installation Path**

The path to the top-level directory of the SpyGlass-Power installation tree. By default, the flow reads the value of the SPYGLASS_HOME environment variable. Enter a valid path if the environment variable is not set or to override the variable. You can also set this option using the “[*options set Flows/SpyGlassPower Path <path>*](#)” command.

- **Format for simulation value change dump (vcd or fsdb)**

The SpyGlassPower flow can use either VCD or FSDB input files. The default is “vcd” generated by the ModelSim tool. Alternatively, specify “fsdb” to use FSDB input generated by the Novas FsdbWriter tool. Refer to “[Novas Flow Options](#)” on page 208.

You can also set this option using the “[*options set Flows/SpyGlassPower VCD_FORMAT <name>*](#)” command.

- **Technology library compilation options**

Specify compiler options for SpyGlass-Power to use when compiling technology libraries. Refer to the Atrenta SpyGlass documentation for information about valid command options. You can also set this option using the “[*options set Flows/SpyGlassPower SG_TECHLIB_OPTS <list_of_options>*](#)” command.

- **Design Compilation Options**

Specify compiler options for SpyGlass-Power to use when compiling the design. Refer to the Atrenta SpyGlass documentation for information about valid command options.

You can also set this option using the “[*options set Flows/SpyGlassPower SG_COMP_OPTS <list_of_options>*](#)” command.

- **Power Analysis Policies**

Specify compiler options for SpyGlass-Power to use when performing power analysis. Refer to the Atrenta SpyGlass documentation for information about valid command options. You can also set this option using the “*options set Flows/SpyGlassPower SG_POWER_OPTS <list_of_options>*” command.

- **Allow VCD compression**

Enable this option if the version of spy-glass you are using supports compressed VCD (.vcd.gz) the option ALLOW_VCD_COMPRESSION will use the compressed format. You can also set this option using the “*options set Flows/SpyGlassPower ALLOW_VCD_COMPRESSION <true_or_false>*” command.

- **“Wire Load Model override” and “Wire Selection Group override”**

The wire_load_model and the wire_load_selection_group settings are copied from the Catapult library and applied during the power analysis. The “Wire Load Model override” and “Wire Selection Group override” options allow you to override the wire_load_model and the wire_load_selection_group values specified in the library (or simply create them if they are not specified in the catapult library).

If wire_load_model and the wire_load_selection_group are not specified in library, and the “Wire Load Model override” and “Wire Selection Group override” options are empty, then SpyGlass will look for default values in the technology library (in Liberty format). If no values are found anywhere, SpyGlass will error out.



Note

Only the technology library default wire_load_model and wire_load_selection_group are supported in SpyGlassPower prior to version 4.

You can also set the “Wire Load Model override” option using the “*options set Flows/SpyGlassPower wire_load_model <list_of_models>*” command.

You can set the “Wire Selection Group override” option using the “*options set Flows/SpyGlassPower wire_load_selection_group <list_of_groups>*” command.

TalusDesign Flow Options

In the *TalusDesign* flow, Catapult C Synthesis produces scripts that run Magma Talus Design for the downstream RTL synthesis and library characterization. Catapult does not provide prebuilt and characterized Talus Design libraries, so you must create them with templates in the Catapult C Library Builder.

The following default setting options can be specified for the TalusDesign flow:

- **Library Search Path**

Specifies one or more paths to search for Talus Design libraries. Talus Design searches the specified paths in addition to the default library search paths. Select the **Add**, **Edit** and **Delete** buttons to modify the paths in the list. The library search path can also be set with the “*options set Flows/TalusDesign SearchPath <list_of_paths>*” command.

- **Talus Executable Name**

Specifies the name of the Talus Design executable. Click the folder to browse for the executable. You can also set this option using the “*options set Flows/TalusDesign Path <path>*” command.

- **Executable Path**

Specifies the path to the Talus Design executable. If none is found, the standard path is searched. Click the folder to browse for the desired path. You can also set this option using the “*options set Flows/TalusDesign Path <path>*” command.

- **Command-Line Flags**

Specifies command line switches for the Talus Design executable. Command line switches can also be specified with the “*options set Flows/TalusDesign Flags <list_of_flags>*” command.

- **License Server**

Specifies the servers for Talus Design licenses. Catapult assigns the specified list to the FLEXnet environment variable MAGMA_LICENSE_FILE prior to launching Talus Design. Refer the *FLEXnet Licensing End User Guide* for information about the syntax of FLEXnet environment variables. The license servers can also be specified with the “*options set Flows/TalusDesign LicenseServer <list_of_servers>*” command.

- **Additional Licenses to Check out**

Specifies additional Talus Design licenses available for use. Select the **Add**, **Edit** and **Delete** buttons to modify the list. Additional licenses can also be specified with the “*options set Flows/TalusDesign Licenses <list_of_composites>*” command.

- **Output netlist name**

Specifies a basename for netlist output by Talus Design. The netlist name can also be specified with the “*options set Flows/TalusDesign OutNetlistName <string>*” command.

- **Output netlist format**

Specifies the format for the netlist output by Talus Design. Select an option from the dropdown menu. The netlist format can also be specified with the “*options set Flows/TalusDesign OutNetlistFormat <string>*” command.

- **Flatten Design**

Flattens the design when generating the output netlist. The flatten design option can also be specified with the “*options set Flows/TalusDesign FlattenDesign <true_or_false>*” command.

- **Enable Clock Gating**

Directs the Talus Design compiler to perform clock gating algorithm. For more information about clock gating, refer to “[Clock Gate Insertion](#)” on page 589. You can also enable clock gating with the “*options set Flows/TalusDesign EnableClockGating <true_or_false>*” command.

- **Enable Power Reporting**

Directs the Talus Design compiler to generate a power analysis report. You can also enable power reporting with the “*options set Flows/TalusDesign EnablePowerReporting <true_or_false>*” command.

Valgrind Flow Options

Set the default values for the Valgrind flow options. The Valgrind utility can be used by the SCVerify flow to check your original C++ design and testbench to check for memory issues such as uninitialized memory reads and out of bounds array accesses. Refer to “[C/C++ Designs and SCVerify](#)” on page 516 for more information.

- **Path to Valgrind (linux only)**

The path to the valgrind executable. You can also set this option using the “*options set Flows/Valgrind VALGRIND <path>*” command.

- **Valgrind Options (linux only)**

Specify command line options for valgrind. Refer to the Valgrind documentation for information about available options. You can also set this option using the “*options set Flows/Valgrind VALGRIND_OPTS {<list_of_options>}*” command.

VCS Flow Options

The VCS dialog box specifies default invocation values for the Synopsis VCS simulator flow options. The VCS simulator can be used to run simulations for the SCVerify flow. Refer to “[C/C++ Designs and SCVerify](#)” on page 516 for more information.

- **VCS Install Directory**

Specifies the path to the Synopsis VCS executable. The path can be set with the “*options set Flows/VCS VCS_HOME <path>*” command. The default value is \$VCS_HOME.

- **VCS Compiler Flags**

Specifies command line switches for the VCS executable. See the VCS documentation for information about available options. You can also set this option using the “[*options set Flows/VCS COMP_FLAGS {<list_of_flags>}*](#)” command.

- **Force VCS 32-bit simulation mode**

Forces the use of the 32-bit simulation mode even when a 64-bit simulation mode is available. The 32-bit simulation mode can also be set with the “[*options set Flows/VCS FORCE_32BIT*](#)” command.

- **VCS GNU Compiler version**

Specifies the GNU compiler version used with VCS. The compiler can also be specified with the “[*options set Flows/VCS VCS_GCC_VER*](#)” command.

Vista Flow Options

The Vista dialog box specifies the default locations of the Vista software.

- **Vista installation directory**

The path to the bin directory of the Vista software tree. The factory default value, “VISTA_HOME,” is a Catapult variable set to environment path “\$VISTA_ROOT/bin.” Enter a valid path if the environment variable is not set or to override the variable. You can also set this option using the “[*options set Flows/Vista VISTA_HOME <path>*](#)” command.

- **ModelBuider installation directory**

The path to the Model_Builder subdirectory inside the Vista software tree (\$VISTA_ROOT/Model_Builder). You can also set this option using the “[*options set Flows/Vista MODEL_BUILDER_HOME <path>*](#)” command.

XPower Flow Options

Power analysis flow using the Xilinx Xpower tool. Refer to “[Power Analysis and Optimization](#)” on page 583 for information about how to run the Xpower flow.

- **XILINX**

The path to the top-level directory of the Xilinx installation tree. By default, the flow reads the value of the XILINX environment variable. Enter a valid path if the environment variable is not set or to override the variable. You can also set this option using the “[*options set Flows/XPower XILINX <path>*](#)” command.

- **Xilinx XPower Options**

Specify options for XPower to use when performing power analysis. Refer to the Xilinx Xpower documentation for information about valid command options. You can also set

this option using the “`options set Flows/XPower XPWR_OPTS {<list_of_options>}`” command.

Saving and Restoring Session Options

Saving your session option settings allows you to:

- Maintain consistent option settings between sessions and software upgrades.
- Save any number of alternate configurations.

By default, option settings are saved in the Catapult *registry* unless you explicitly save them to an alternate file, such as the catapult.ini file. Refer to “[Catapult Registry](#)” on page 224 and “[The Catapult Initialization File](#)” on page 224 for more information.

Saving Options

You can use either the **Tools > Save Options** or **Tools > Save Options As...** pulldown menus. Alternatively, you can use the `options save` command. When using **Save Options As...**, name the output file catapult.ini if you want Catapult to load it automatically at startup.

- When saving settings specific to a project, use **Save Options As...** and save them to a catapult.ini file in the project directory. (To see an example catapult.ini file, refer to “[The Catapult Initialization File](#)” on page 224.)
- **Save Options** will save the settings to the source location from which they were loaded when the session started. The default location is the Catapult registry.

Additionally, you can have Catapult automatically save your settings each time you exit the tools. Refer to the section “[General Options](#)” on page 175 for information about this option.

Restoring Options

During startup, Catapult searches for saved option settings in the following order. The transcript lists the source location of the option settings loaded for a session.

1. A catapult.ini file in the current working directory
2. A catapult.ini file in the user’s HOME directory
3. The Catapult registry

Use the following command to load an alternate configuration file:

```
options load -file <file>.ini
```

Catapult Registry

Depending on the operating system, Catapult uses a registry to maintain the default session option settings as follows:

- **Windows** — option settings are stored in the Windows registry.
- **Linux** — a registry is created in \$HOME/.catapult. The registry contains a separate file for each platform and software version.

The Catapult Initialization File

The Catapult initialization file (catapult.ini) is an ASCII text file created by the **Tools > Save Options As...** command. The file contains the default settings for all Catapult C Synthesis system options. The settings are grouped into sections corresponding to the groupings in the Catapult Synthesis Options window and section names are enclosed in square brackets as shown in [Figure 4-3](#).

Figure 4-3. Excerpt from catapult.ini File

```
[General]
RestoreCWD = false
StartInWD =
SaveSettings = false
ShowToolBar = true
ShowFlowWindow = true
PdfViewer = acroread

[Message]
ErrorOverride = ASSERT-1 CIN-17 CIN-48 CIN-46 CIN-49 CIN-50 CIN-54
WarningOverride =
InformationalOverride =
Hide = CRD-177
```

Chapter 5

Design Optimization

This chapter describes optimization features and how loops, arrays, memories and general design constraints are handled within Catapult. For more information, see the following topics:

- “[Loop Constraints](#)” on page 225
- “[Memories and Arrays](#)” on page 248
- “[Advanced I/O Control for Memories](#)” on page 271
- “[General Design Constraints](#)” on page 273

Loop Constraints

Constraining loops is one of the primary means to optimize a design. You can constrain loops in the following ways.

- **Loop iterations:** Loop iterations refers to the number of times the loop runs before it exits. You can constrain the number of loop iterations if the number estimated by Catapult is not satisfactory.
- **Loop unrolling:** Loop unrolling refers to the number of times the loop body is copied. After the loop iterations are determined, the loop is unrolled based on the unrolling constraints. This changes the number of iterations of the loop but does not change the iterations directive. You can view the results in the report output.
- **Loop merging:** Loop merging is a technique that reduces latency and area consumption in a design by allowing parallel execution of loops that normally execute in series. Loop merging is applied only after loop iterations are established and loop unrolling and memory mapping are done.
- **Loop pipelining:** Loop pipelining is how often to start the next iteration of the loop. After loop unrolling and several other transformations are complete, the scheduler uses the pipelining constraints to build a pipelined loop. This happens much later than the other two transformations, and pipelining does not occur if the loop is optimized away because of the first two constraints.

Figure 5-1 shows source code with the loops highlighted in red font. Note that the function call in the design is also considered a loop for synthesis.

Figure 5-1. Source Code with Loop

```
#define NUM_TAPS 8

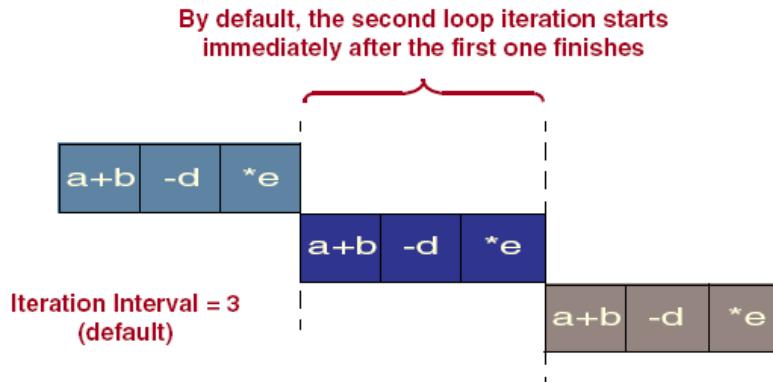
#pragma design top
void fir_filter ( ac_int<8> *input,
                  ac_int<8> coeffs[NUM_TAPS],
                  ac_int<8> *output ) {
    static ac_int<8> regs[NUM_TAPS];
    int temp = 0;
    int i;
    SHIFT:for ( i = NUM_TAPS-1; i>=0; i-- ) {
        if ( i == 0 )
            regs[i] = *input;
        else
            regs[i] = regs[i-1];
    }
    MAC:for ( i = NUM_TAPS-1; i>=0; i-- ) {
        temp += coeffs[i]*regs[i];
    }
    *output = temp>>7;
}
```

Loops

Loops and Timing

When Catapult synthesizes a loop, the loop takes at least one clock cycle per iteration. Figure 5-2 shows the default loop iteration.

Figure 5-2. Default Loop Iteration



Every loop that is not completely unrolled has at least one wait and the following consequences:

- This affects the minimum latency of the design. For more information, see “[Minimum Latency and Loop Unrolling](#)” on page 233.
- Consecutive loops run one at a time unless loop merging is employed (not all loops are suitable for merging). For more information, see “[Consecutive Loops](#)” on page 228 and “[Loop Merging](#)” on page 234, respectively.

- The placement of I/O in a loop can result in duplicate I/O accesses because redundant I/O accesses are not optimized across waits. For more information, see “[Loops and I/O Access Placement](#)” on page 229.

The appropriate action varies depending on whether the goal is area or performance.

The code in Example 5-1 shows how Catapult inserts “virtual waits” in the loops.

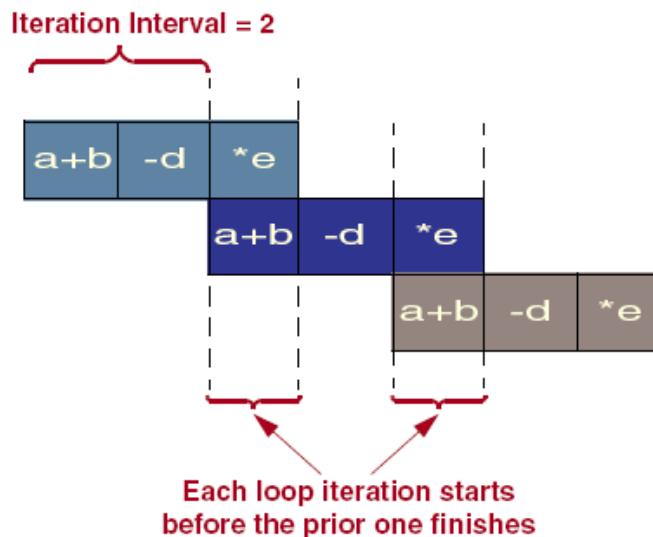
Example 5-1. Code Timing Example

```
for (int i = 0; i < 10; i++) {  
    <Virtual Wait>  
    ...  
    if ( input == 55 ) break;  
    ...  
}  
  
int i = 1;  
while (i != 64) {  
    <Virtual Wait>  
    ...  
    i = i << 1;  
}
```

Timing and Loop Pipelining

In loop pipelining, Catapult starts the second loop iteration before the first one finishes as shown below. For more information on loop pipelining, see “[Loop Pipelining](#)” on page 239.

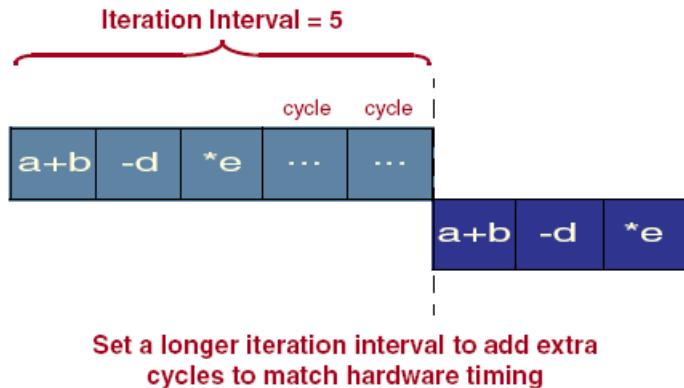
Figure 5-3. Timing using Loop Pipelining



Loop Iteration Interval Delay

You can also match hardware timing by setting a longer iteration interval, as shown in Figure 5-4. For more information on loop iterations, see “[Constraining Loop Iterations](#)” on page 229.

Figure 5-4. Iteration Loop Delay



Consecutive Loops

The design goal, area or performance, determines how consecutive loops are used. Catapult takes the following actions in regard to consecutive loops:

- Unless Catapult can merge the loops, consecutive loops cannot be scheduled in parallel.
- Consecutive loops are put in two separate finite state machine states.
- Regularity is usually lost between consecutive loops, which results in increased hardware size.

If the goal is minimizing area, you should consider using loop merging or revise the C source code to remove consecutive loops. If the goal is design performance, then you may want to incorporate consecutive loops. An example of source code with consecutive loops is shown in Example 5-2.

Example 5-2. Consecutive Loop

```
for (int i=0; i<10; i++) {
    <Virtual Wait>
    ...
    if (input == 55) break;
    ...
}
```

```
int i = 1;
while (i != 64) {
    <Virtual Wait>
    ...
    i = i << 1;
}
```

Loops and I/O Access Placement

Within Catapult, the placement of I/O accesses results in different hardware as follows:

- An I/O read inside of a rolled loop is read one time for each iteration.
- An I/O read outside of the loop is read one time, but now an input register is required.

An example of source code with different I/O access pointers is shown in Example 5-3.

Example 5-3. I/O Access Pointer

```
for (int i = 0; i < 3; i++) {
    <Virtual Wait>
    ...
    // 3 unregistered reads
    if a = *b;
    ...
}
```

```
// One unregistered read
int temp = *b;
for (int i = 0; i < 3; i++) {
    <Virtual Wait>
    ...
    a = temp;
    ...
}
```

Constraining Loop Iterations

Iteration count is very important for optimization and latency estimates provided by Catapult. If Catapult can determine an iteration count of a loop, that value displays in the Iteration Count field in the Architectural Constraints editor. If the tool cannot determine a count, the field is blank. Refer to Figure 5-5.

Whenever possible, the C/C++ code should provide a default iteration count so Catapult can use this information during optimization.

Entering the iteration count in the Iteration Count field is also an option, but it should be avoided if the C/C++ code can be changed. You can change the iteration count:

- For loops that can not be calculated.
- To decrease loops that always finish early.

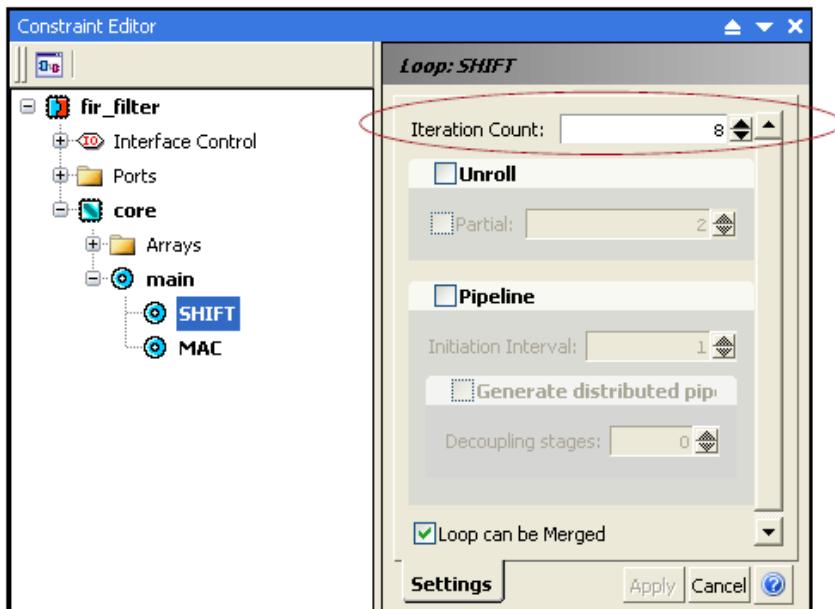
Note

If you enter a value, and the result produced by Catapult does not look realistic, then you should check the iteration count value.

Setting Iteration Count in Catapult

Within Catapult, loops are listed in the Architectural Constraints editor window. Click on the **Architectural Constraints** task to open the editor. Selecting a loop displays the dialog box, shown in Figure 5-5, with which you can enter the iteration count for the loop. When the Iteration Count setting is applied, Catapult sets the **ITERATIONS** directive to the specified value on the selected loop.

Figure 5-5. Loop Iteration Count in Architectural Constraints Editor



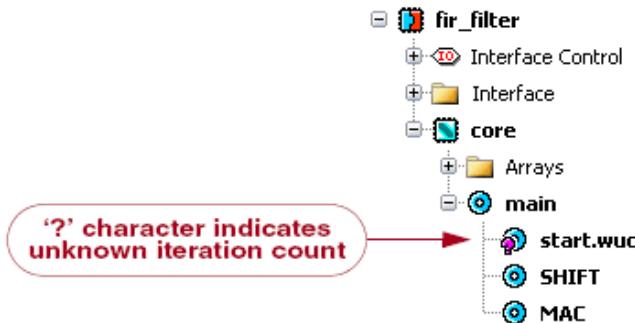
When an Iteration Count Cannot be Determined

Catapult automatically determines the number of loop iterations for some loops. If the iteration count for a loop cannot be determined, the loop displays with a question mark in the following windows:

- The Architectural Constraints editor window
- The Schedule window
- Loop Report
- XY-Plot (and other views)

Figure 5-6 shows the special loop icon displayed if it the loop has an unknown iteration count.

Figure 5-6. Loop Icon for Unknown Iteration Count



An example of a Loop Report showing an unknown iteration count is shown in Example 5-4.

Example 5-4. Unknown Loop Iteration Count Shown in Cycle Report

Process	Loop	Iterations	C-Steps	Total Cycles	Duration
/fir_filter/core	core:rlp	Infinite	0	11	110.00 ns
/fir_filter/core	main	Infinite	2	11	110.00 ns
/fir_filter/core	start.wuc	?	1	1	10.00 ns
/fir_filter/core	SHIFT	8	1	8	80.00 ns

Process	Loop	% of Overall Design Cycles Thr.

Unknown Iteration Count and Loop Unrolling

When a loop does not have an iteration count, the loop can only be partially unrolled. If you set the iteration count, complete unrolling is then available from the tool.

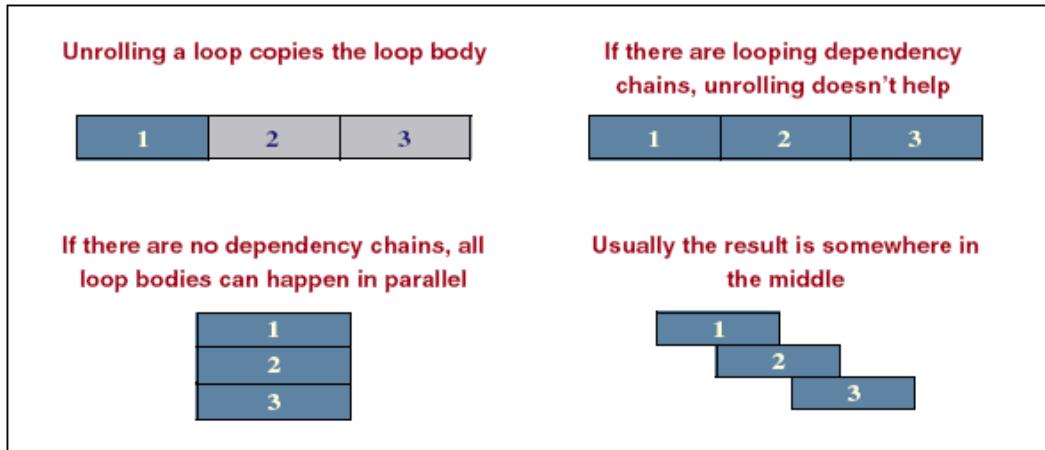
Loop Unrolling

Loop unrolling is a technique that transforms the serial iterations of a loop body into individual code units allowing them to execute in parallel whenever possible. It is controlled by applying the **UNROLL** directive to some or all of the loops in your design. Unrolling is a good option if the loop has few iterations (less than eight) or few operations. Unrolling loops larger than this causes runtime to increase dramatically.

After the number of loop iterations is determined, the loop is unrolled based on the unrolling constraints. This changes the number of loop iterations, which is reflected in the reporting, but it won't change the setting of the **ITERATIONS** directive.

During loop unrolling, the body of the loop is copied several times. The result is that the loop is either completely dissolved, or a new loop is created that iterates fewer times. Figure 5-7 shows what happens during loop unrolling.

Figure 5-7. Loop Unrolling



Dependency Chains

Dependency chains play an important role in what results when a loop is unrolled. In Example 5-5, the code on the left doesn't have any dependency chains between iterations, so the unrolled loop can run completely in parallel. In the code on the right, $a[i]$ depends on $a[i-1]$ from the previous iteration so there is a looping dependency chain. This means that the adders can't run in parallel when the loop is unrolled.

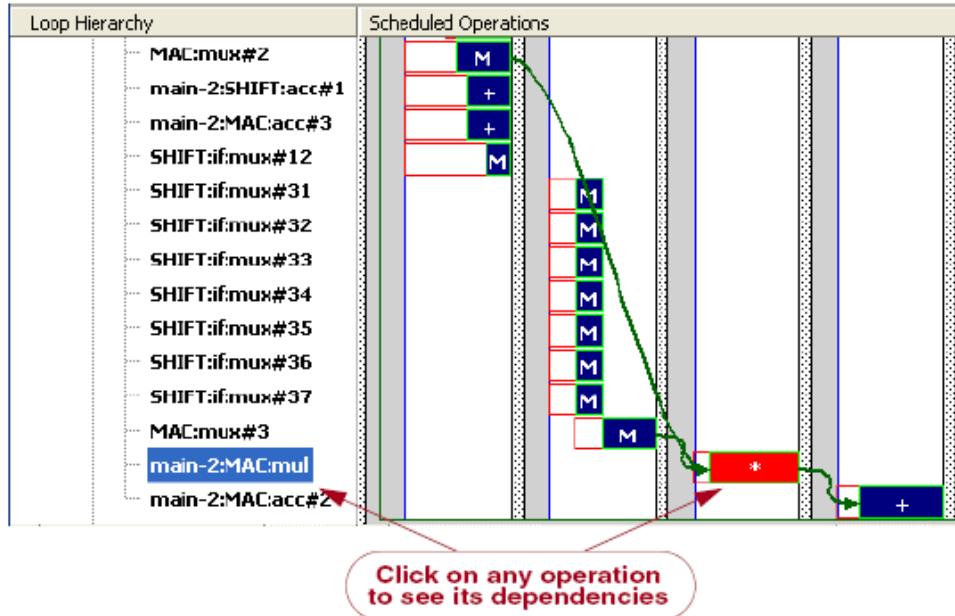
Even if there are looping dependency chains, an unrolled loop will probably run faster because a rolled loop requires at least one clock cycle per iteration.

Example 5-5. Loop Unrolling Code Example

No Dependency Chain (Operations can be parallel)	Dependency Chain (Operations cannot be parallel)
<p>Rolled</p> <pre>for (int i = 0; i < 3; i++) a[i] = b[i] + c[i];</pre> <p>Unrolled</p> <pre>a[0] = b[0] + c[0]; a[1] = b[1] + c[1]; a[2] = b[2] + c[2];</pre>	<p>Rolled</p> <pre>for (int i = 1; i < 4; i++) a[i] = a[i-1] + c[i-1];</pre> <p>Unrolled</p> <pre>a[1] = a[0] + c[0]; a[2] = a[1] + c[1]; a[3] = a[2] + c[2];</pre>

To view dependencies in your design, click on one of the operations in the Gantt chart. Once you have set up your design, and added input files, click the **Schedule** task on the Task Bar to display the Gantt chart. A looping dependency chain shows up as a line that starts on the left and points to an object on the right, as shown in Figure 5-8.

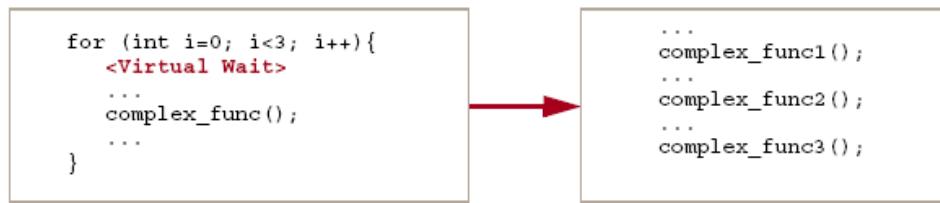
Figure 5-8. Viewing Data Dependencies in the Gantt Chart



Minimum Latency and Loop Unrolling

Within Catapult, each loop iteration takes at least one cycle. However, a loop that is unrolled doesn't have this constraint. To optimize the design and increase performance, you should unroll a loop. An example of source with loop unrolling is shown below.

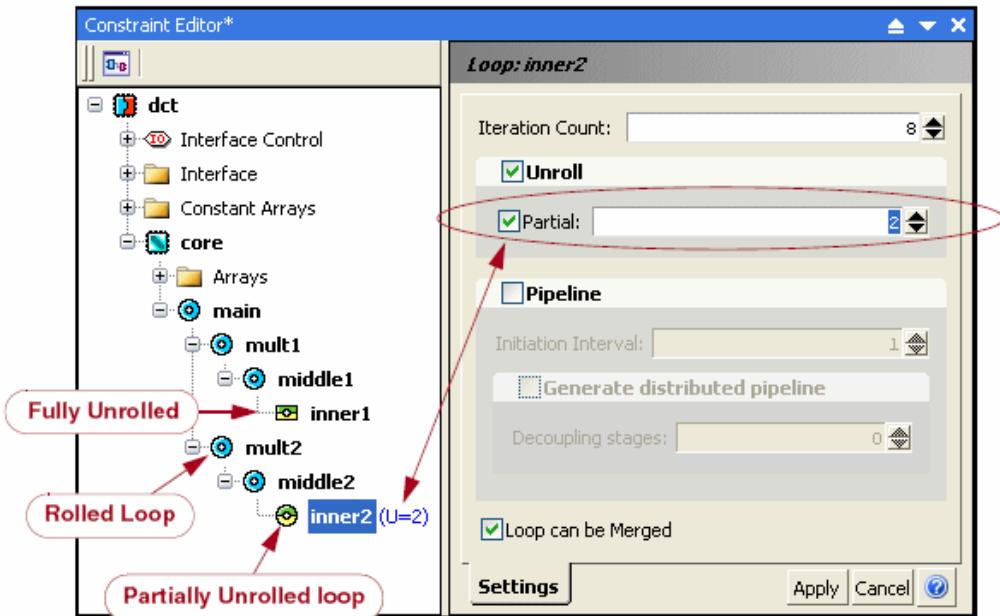
Figure 5-9. Loop Unrolling Example



Setting Loop Unrolling in Catapult

Click on the **Architectural Constraints** task on the Task Bar to display Architectural Constraints editor. Loops are at the bottom of the design browser in the editor. Selecting a loop displays the loop constraint options, including the unroll options. Figure 5-10 shows the loop constraint options and the various loop icons used to indicate rolled/unrolled status of a loop.

Figure 5-10. Loop Unrolling in Architectural Constraints Editor



To unroll a loop completely, enable the **Unroll** option and disable the **Partial** option. In order to partially unroll the loop, enable both options and then specify the number of times the loop body should be copied. Any loop can be partially unrolled, but only loops that have an iteration count can be completely unrolled.

Loop Merging

Loop merging reduces latency and area consumption in a design by executing some loops in parallel instead of in series. The **MERGEABLE** directive controls loop merging in your design.

Loop merging is applied after loop iterations are established and loop unrolling and memory mapping are done. Loops that meet the following criteria are merged:

- MERGEABLE directive set to “true” applied (directly or indirectly).
- Not fully unrolled.
- Iteration count greater than 1.

Note

Certain types of data dependencies between loops prevent merging. Data dependencies occur when one loop reads data from a variable that is written in another loop. Dependent loops are merged if Catapult can determine a viable pattern of reads/writes between the loops.

Loop Merging and Architectural Constraints

Loop merging takes place after architectural constraints are set, so you can select the loops to be merged. The merge algorithm operates in hierarchical fashion, allowing you to apply the MERGEABLE directive to any combination of the following hierarchy objects: DESIGN, SOLUTION, PROCESS, or LOOP.

For high level control, you can set a default MERGEABLE value on the SOLUTION, DESIGN, or PROCESS, thereby implicitly applying the directive setting to all hierarchically subordinate loops. For low level control, set the MERGEABLE directive on individual LOOPS. Note that the lower level settings override the higher level settings. The order of precedence is as follows:

1. LOOP
2. PROCESS
3. DESIGN
4. SOLUTION

Loops Containing Wait Statements or I/O Function Calls

By default, the MERGEABLE directive is set to “true”. However, for SystemC, the default is set to “true” except when loops contain a wait statement or a modular I/O function call as shown in the following examples:

```
wait_loop: for ( int i = 0; i < 10; i++ )
    wait();
```

or

```
io_loop: for ( int i = 0; i < 10; i++ )
    out.write(in.read());
```

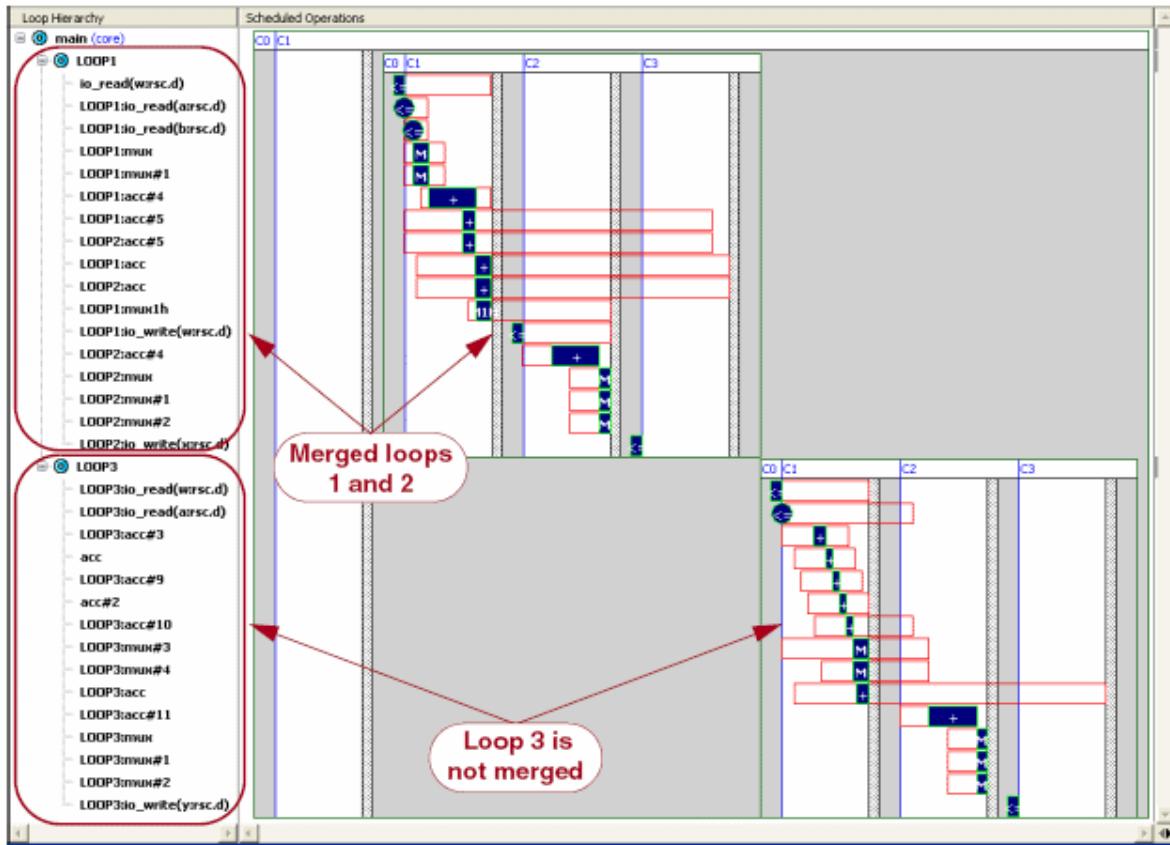
In both cases, the MERGEABLE directive is set to “false” so neither can run concurrently with other loops. You can use architectural constraints to override this behavior and enable loop merging.

Example 1: Data Dependency

Given the example code below, assume that the **MERGEABLE** directive is set to “true” and applied to all of the loops.

```
#pragma hls_design top
void test (int a[3], int b[3], int w[3], int x[3], int y[3]) {
    LOOP1:for ( int i = 0; i < 3; i++)
        w[i] = a[i] + b[i];
    LOOP2:for ( int j = 0; j < 3; j++)
        x[j] = b[j] + w[j];
    LOOP3:for ( int k = 0; k < 3; k++)
        y[k] = a[k] - w[(k+1)%3];
}
```

The result of loop merging in this example is that loops 1 and 2 are merged, but loop 3 is not due to a data dependency on `w[]`. Loop 3 cannot be merged because it reads a value of `w[]` that is not available yet. Notice that loop 2 also has a dependency on `w[]`, but because it reads `w[]` in the same order as loop 1 writes to it, they can be merged. Figure 5-11 shows how the merged loops appear in the Schedule window.

Figure 5-11. Schedule View of Example 1 Code


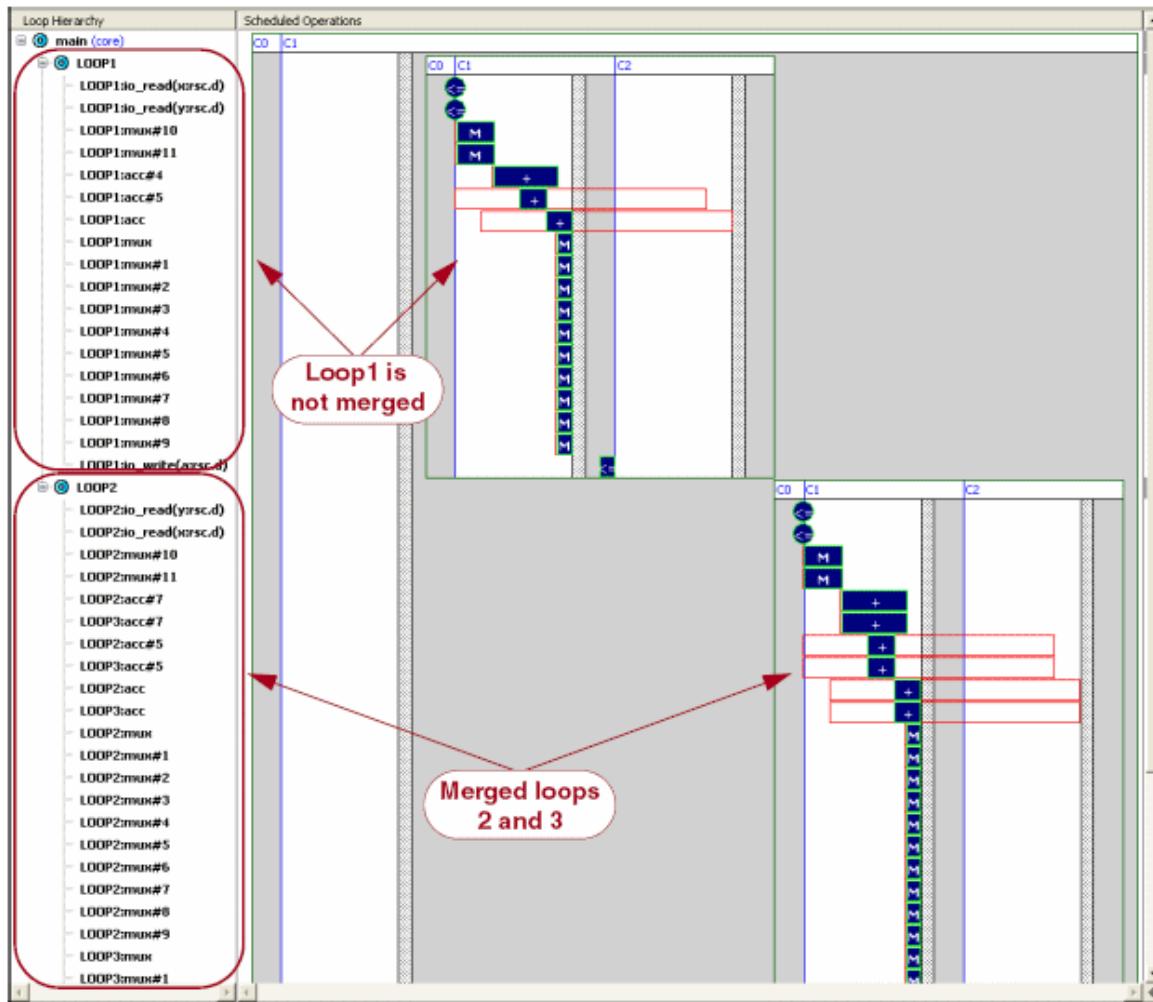
Example 2: Combination of MERGEABLE Directives

Given the example code below, assume that a **MERGEABLE** directive is set to “true” for the solution, and a **MERGEABLE** directive is explicitly applied to loop 1 and its value is “False”.

```
#pragma hls_design top
void test (int a[10], int b[10], int c[10], int x[10], int y[10]) {
    LOOP1:for (int i = 0; i < 10; i++) /* (MEARGEABLE is False) */
        a[i] = x[i] + y[i];
    LOOP2:for (int j = 0; j < 10; j++)
        b[j] = x[j] - y[j];
    LOOP3:for (int k = 0; k < 10; k++)
        c[k] = y[k] - x[k];
}
```

All three loops satisfy the merge criteria and would be merged, except that Loop 1 is explicitly excluded from being merged. The result is that the merge algorithm skips over loop 1, and merges loops 2 and 3, as illustrated in Figure 5-12.

Figure 5-12. Schedule View of Example 2 Code



Setting MERGEABLE directive from the GUI

To enable/disable solution-wide loop merging, first choose **Tools > Set Options...** to open the Catapult Synthesis Options window. Then expand **Project Initialization** and select **Architectural** to access the **Loop Merging** option checkbox. The Loop Merging checkbox corresponds to the DefaultLoopMerge option, which appears as the following command line in the transcript window:

```
options set Architectural DefaultLoopMerging <true or false>
```

To enable/disable loop merging on individual loops, open the Architectural Constraints editor and select a loop to access the **Loop can be Merged** option checkbox. The option will issue a directive `set` command, appearing in the transcript window similar to the following:

```
directive set /test/core/main/LOOP1 -MERGEABLE <true or false>
```

The loop icon in the hierarchy pane of the Architectural Constraints editor indicates the mergeable status of the loop. If not mergeable, the loop icon has a square box around it. Otherwise, the box is omitted.

Loop Pipelining

Loop Pipelining is not the same pipelining used in RTL. Loop pipelining is similar to the pipelining done in CPUs where the second iteration/operation is started before the first one finishes. If loops are nested, internal loops can be pipelined to reduce latency. The outermost loop can be pipelined to produce a throughput driven design.

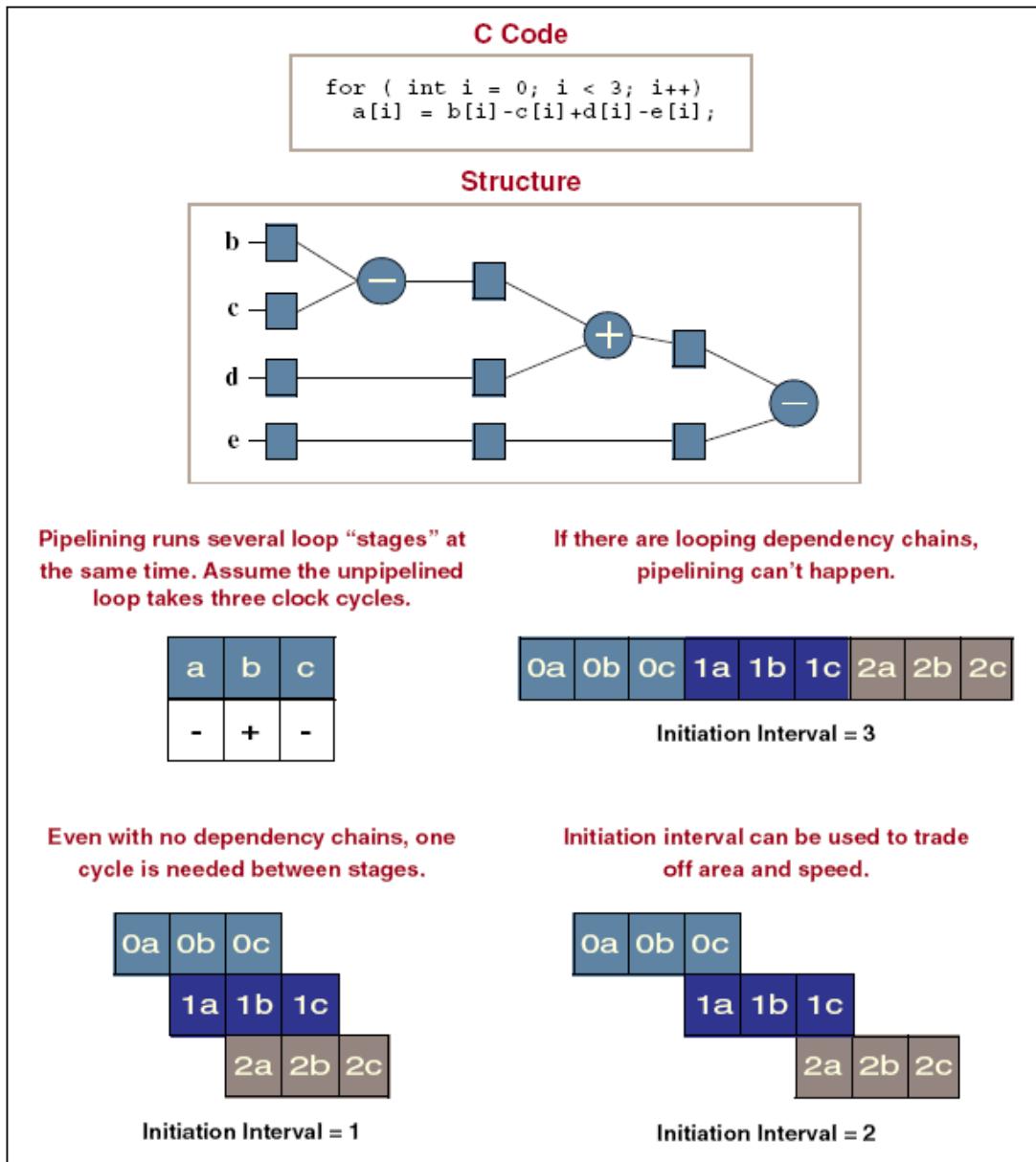
Loop pipelining is controlled by applying the `PIPELINE_INIT_INTERVAL` directive to some or all of the loops in your design. After loop unrolling and several other transformations have been completed by Catapult, the scheduler applies the pipelining constraints to build a pipelined loop. Just like in loop unrolling, loop pipelining is limited by the looping dependency chains in your loop.

This section provides information on how Catapult handles loop pipelines and how to enter pipeline information using the Catapult graphical user interface. Figure 5-13 illustrates how Catapult handles pipelining.

Within Catapult, you should pipeline a loop:

- If it contains poorly utilized components.
- If the loop has many iterations. Pipelining is useful for any loop that takes two or more C-Steps (assuming no looping dependency chains).
- To increase throughput of nested loops by pipelining the outermost loop. Only one loop in a group of nested loops can have an initiation interval set.

Figure 5-13. Loop Pipelining Overview



Conditional Data Dependencies in Pipelines

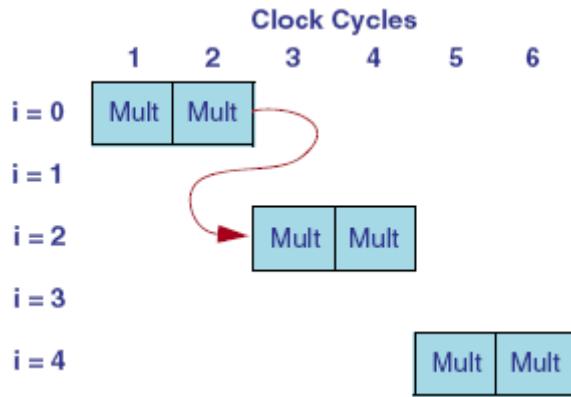
Conditional data dependency refers to a dependency that might not occur in every iteration of the pipeline because of conditional logic in the loop body. For example, the following loop iterates 100 times, but the assignment to “temp” only occurs every 2nd iteration. Furthermore, the first multiplication operation is dependent on the value of temp from the “previous” assignment.

```
int temp;
for (i=0; i<100; i++)
```

```
if (i%2 == 0)
    temp = temp * input_1 * input_2;
```

In this example, if the initiation interval (II) is set to 1, the normal scheduling algorithm would attempt to schedule MULT operations in every clock cycle. But the data dependency does not allow that. Instead, the Catapult scheduler automatically adjusts the schedule to compensate for the data dependency. The MULT operations in this case, are scheduled in every other clock cycle, as illustrated in Figure 5-14.

Figure 5-14. Schedule Optimized for Conditional Data Dependency



Note This optimization does not optimize for area.

Nested Loop Pipelining

Setting the pipeline initiation interval (II) on a set of nested loops will NOT unroll the loops inside. Instead, the innermost loop is pipelined with the correct initiation interval and the loops around that loop are flattened. The result is that the innermost loop cycles always start based on the initiation interval. This is useful for algorithms that need to have a continuous throughput, and that are easy to describe as a set of nested loops. Consider the following nested loop:

```
void pipeline (...) {
    ...
    Outer: for ( i = 0; i < 10; i++)
        Middle: for ( j = 0; j<10; j++)
            Inner: for (k = 0; k < 10; k++) {
                core_function();
            }
    }
```

- Setting the initiation interval (II) on Inner will pipeline as described
- Setting the II on Middle will pipeline Inner and conditionally execute Middle every 10 cycles (but Inner will still start every II cycles)

- Setting the II on Outer will still pipeline Inner and conditionally execute Middle every 10 cycles and Outer every 100 cycles

How Catapult Handles the Nested Pipeline

Applying loop pipelining on an outer loop in a set of nested loops automatically applies it to all of the inner loops. The design is then built up so that the innermost loops run in a regular way. This is different than unrolling the inner loops and setting the same pipelining constraint on Main. For consecutive loops, Catapult will build hardware that ramps up the second sequential loop while the first loop is ramping down.

Consider the following code example which contains a pair of three-level nested loops.

```
Outer: for (...) {
    middle_1: for ( int i = 0; i < 2; i++)
        inner_1: for ( int j = 0; j < 3; j++)
            a[j] = b[i]-c[i]+d[i]-e[i];
    middle_2: for ( int k = 0; k < 2; k++)
        inner_2: for ( int l = 0; l < 3; l++)
            a2[k] = b[k]-c[k]+d[k]-e[k];
}
```

Setting an II constraint on the outermost loop, really sets the II for the innermost loops over multiple iterations of the outer loops, as illustrated in Figures 5-15, 5-16 and 5-17.

Figure 5-15. Three-Level Nested Loops with No Pipelining

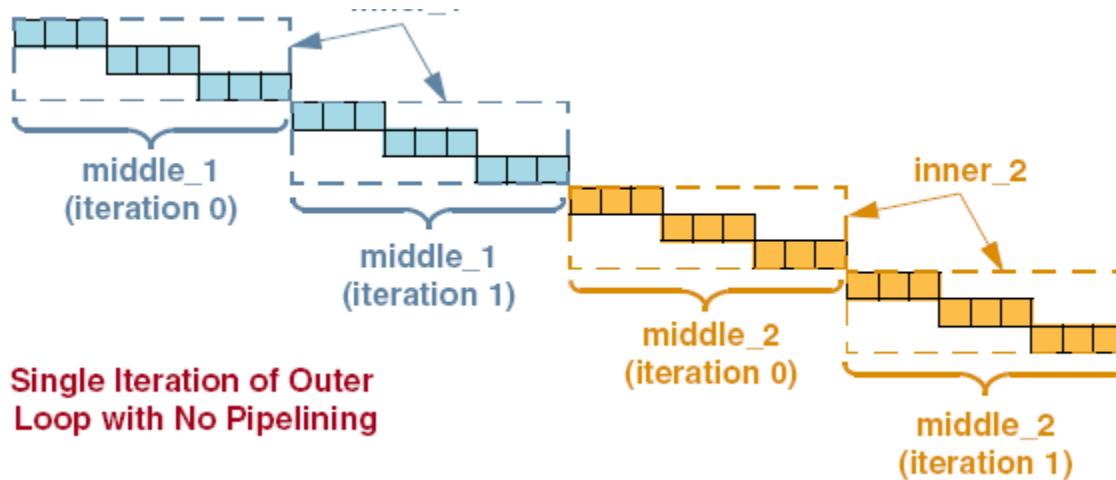
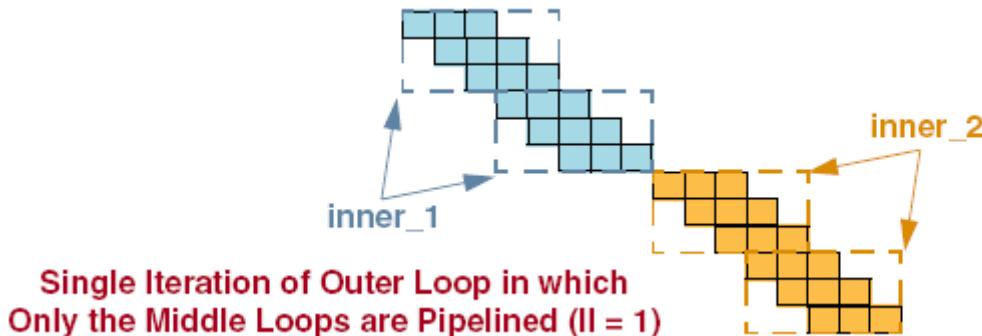
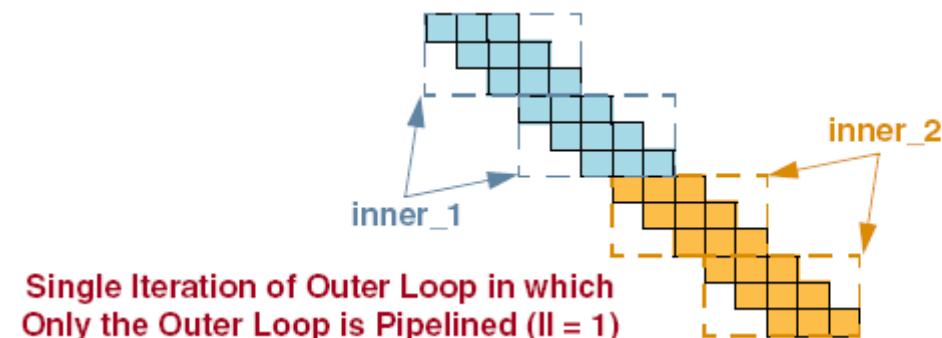
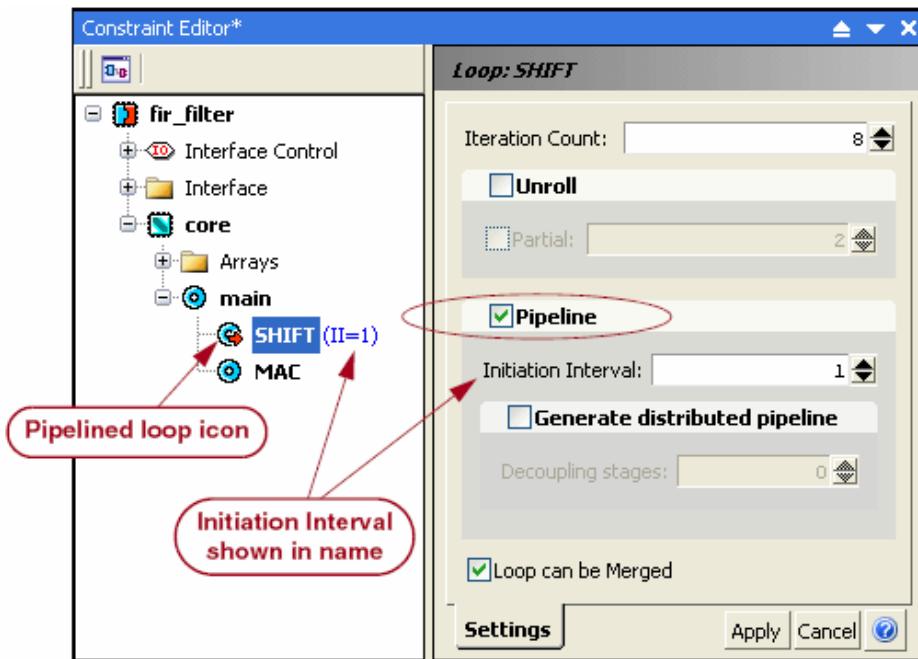


Figure 5-16. Three-Level Nested Loops with Middle Loops Pipelined**Figure 5-17. Three-Level Nested Loops with Outer Loop Pipelined**

Setting Loop Pipelining in Catapult

Click on the **Architectural Constraints** task on the Task Bar to display Architectural Constraints editor. Loops are at the bottom of the design browser in the Architectural Constraints window. Clicking on a loop displays the loop constraint options, as shown in Figure 5-18, with which you can set up the pipelining option.

Figure 5-18. Loop Pipelining in Architectural Constraints Editor



To pipeline a loop, enable the **Pipeline** option, then set the **Initiation Interval** for the loop. The initiation interval controls how many cycles to wait between starting each iteration of the loop. Refer to “[Loop Pipelining](#)” on page 239 for a discussion of how the iteration interval works.

If a loop is completely unrolled, then it can't be pipelined. If the loop is left rolled or partially unrolled then it can be pipelined, but only if the partial unrolling number is less than the iteration count (if there is one). For instructions on how to set an iteration count in Catapult, see “[Setting Iteration Count in Catapult](#)” on page 230.

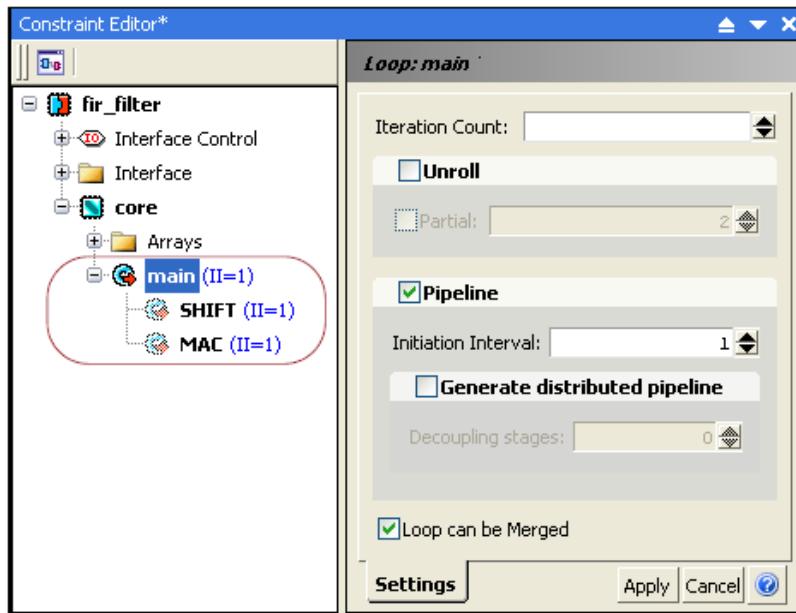
Pipelining Nested Loop

If the outermost loop of a set of nested loops is pipelined, then the inner loops are dissolved. This means that the pipelining constraints for the inner loops cannot be applied. Two things need to be changed with nested loop pipelining:

1. If a pipeline directive is set on a loop and that loop is completely unrolled, then the directive should be propagated to all the loops inside that loop during loop unrolling. This guarantees the pipeline constraint is not lost during loop unrolling.
2. If a loop has an initiation interval, then the initiation interval is not available on the loops inside it. The Architectural Constraints editor displays the initiation interval of the currently selected loop and displays the same initiation interval on all sub-loops that are completely unrolled.

In Figure 5-19, the top loop “main” has an initiation interval of two. This initiation interval is propagated to the remaining loops, “SHIFT” and “MAC,” and these loops are fully unrolled and are not available as shown by the grayed out symbol.

Figure 5-19. Nested Loop Pipelining Symbols



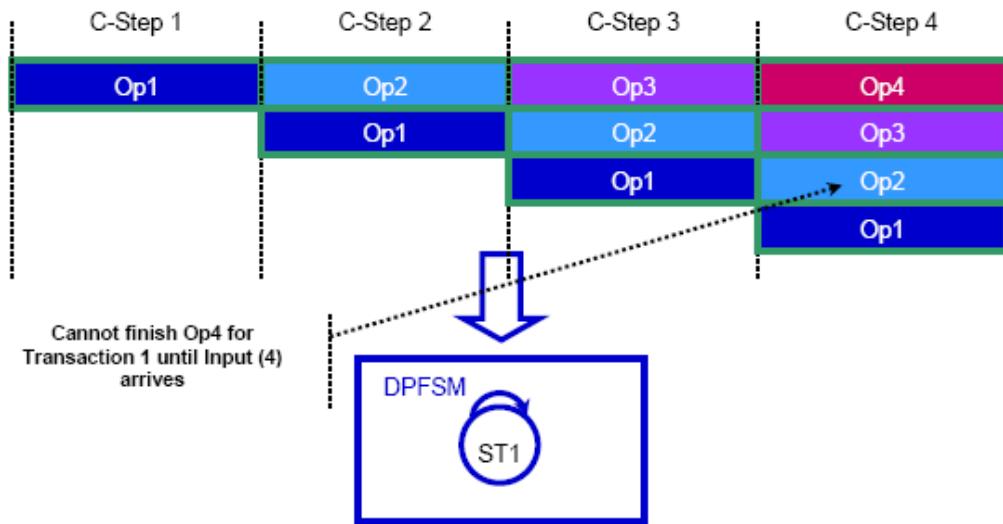
Distributed Pipeline Synthesis

Distributed pipeline synthesis is an optional method for constructing pipelined loops in a way that provides the benefits of pipeline “flushing” and “bubble compression.” Both pipeline flushing and bubble compression are described later. A non-distributed method is also available in Catapult, and the two methods are compared and contrasted below.

In the non-distributed method, Catapult uses a centralized approach with fixed operation cycle assignments when building pipelines. This is illustrated in Figure 5-20 which shows a design that contains four sequential single-cycle operations, and is pipelined with initiation interval (II) of one.

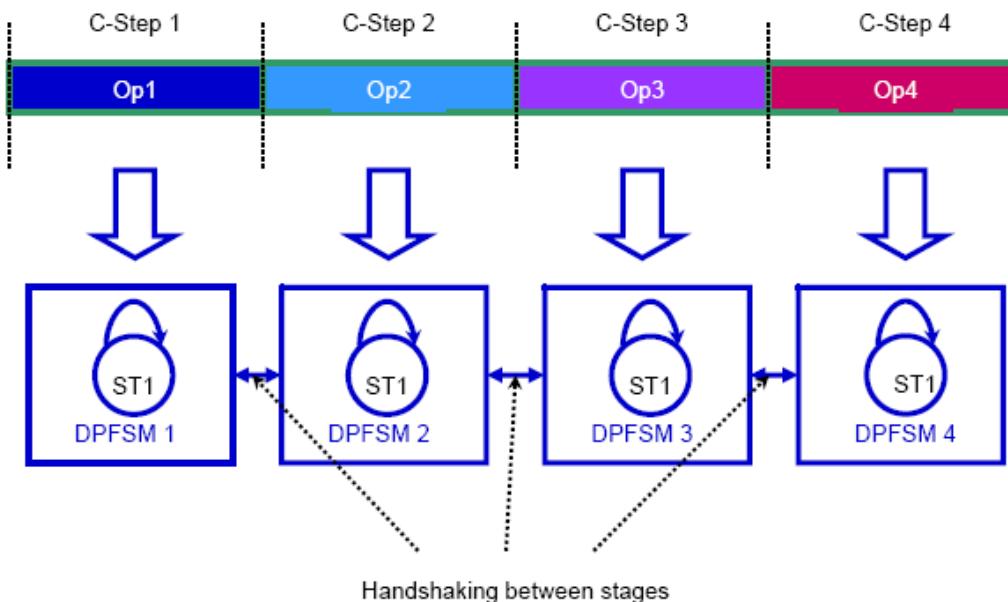
The non-distributed pipelining method maps such a design into a datapath that can handle simultaneous execution of four consecutive transactions (C function calls) and a datapath control finite state machine (DPFSM) that has only one state. The benefit of this approach is that it allows generation of relatively compact hardware. Its disadvantage is that neighboring transactions cannot be decoupled since execution of all the transactions in progress is controlled by one central DPFSM.

Figure 5-20. Non-Distributed Pipelining Illustration



The distributed pipelining algorithm maps C functions into a pipeline with distributed control — a technique widely used in manual RTL designs today. As shown in Figure 5-21, instead of creating a single DPFMSM that handles multiple transactions at once, the new pipelining method generates a cascaded chain of independent DPFMSMs (or pipeline stages) connected by handshaking signals. Each pipeline stage executes only the operations scheduled in adjacent C-steps. The maximum number of adjacent C-steps assigned to one pipeline stage is equal to the desired design initiation interval ($II=1$ is shown in the picture for simplicity).

Figure 5-21. Distributed Pipelining Illustration



Since all the DPFSMs are independent, the hardware does not require input data for the next transaction to be present in order to complete execution of the current transaction. In other words, the pipeline "flushes" freely.

If distributed pipelining (pipeline flushing) is not enabled, then the design stops (because the inputs are stopped) and no additional outputs will appear. When the “Turn off inputs when empty” option is disabled, the inputs are not stalled, the design will continue to read whatever value was last placed on the input (with a warning from SCVerify indicating that the input FIFO has run empty) and the outputs will presumably come out assuming that the testbench iterates enough times.

Pipeline flushing shut offs each input resource (assuming it has full handshaking) as soon as the last input is read. The result is that when all inputs are read from the testbench, all handshake signals (the 'vz' signals) should then be inactive and stall Catapult from reading those inputs. However, the remaining data in the pipeline should still come out even though the inputs are stopped.

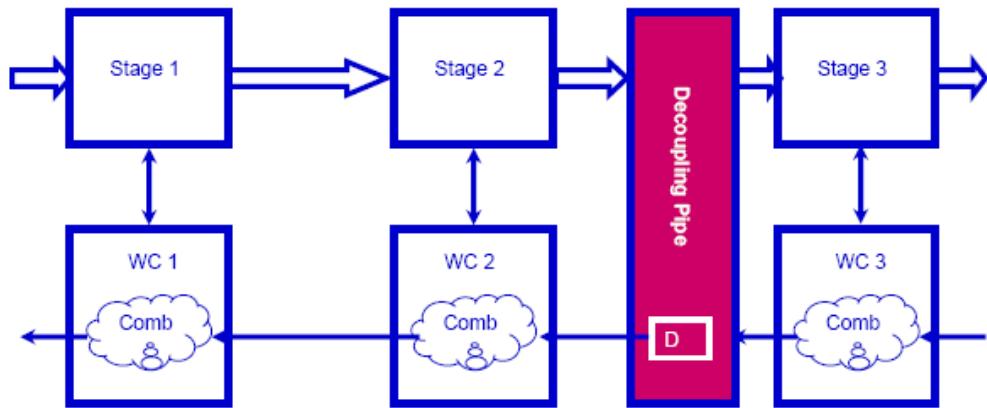
The other important feature this structure can deliver is "bubble compression", which means that if all the stages of a pipeline but the last are in the idle state, the pipeline can continue to accept and process input data even if the last stage is blocked by external logic. This process will continue until all the elasticity elements in the pipeline are filled and a "bubble" (empty space between subsequent transactions) is fully compressed.

Decoupling Stages of Distributed Pipelines

For distributed pipelines that have a large number of pipeline stages, partitioning the total number of stages into smaller groups might improve the timing results. The DECOUPLING_STAGES directive specifies the maximum number of back-to-back pipeline stages allowed in the design. If the number of stages exceeds that threshold, handshake decoupling pipes are inserted (refer to Figure 5-22). A decoupling pipe breaks combinational path into sections, which otherwise exists through the chain of Wait Controllers (WC).

Only timing for datapath control logic can be improved, not for the datapath itself. In other words, you may want to use this directive if the clock frequency reported by the RTL synthesis tools (such as Design Compiler) is less than the clock frequency reported by Catapult, and the critical path runs through the datapath control logic.

Figure 5-22. Decoupling Pipe Between Pipeline Stages



For information about how to enable the distributed pipelining algorithm and decoupling stages, refer to “[Architectural Constraints on Loops](#)” on page 120, “[DISTRIBUTED_PIPELINING](#)” on page 317 and “[DECOUPLING_STAGES](#)” on page 316.

For designs using distributed pipelining associated with wait controllers, we recommend you enable the [REDUNDANT_MUX_OPT](#) directive in the final production run of Catapult. This directive optimizes pipeline ramp-up logic in order to achieve smaller area and perhaps shorter critical paths through wait controller elements.

I/O and Cycle Flow Analysis

Catapult analyzes the C++ code to guarantee that all operations occur in the correct cycle. The scheduler looks at implicit scheduling constraints inferred from the C++ code and constraints set using the [cycle set](#) and [cycle add](#) commands. For more information on how the tool analyzes constraints and issues with I/O scheduling, see the “[Set Cycle Constraints](#)” section in the *Catapult C Synthesis C++ to Hardware Concepts* book.

Memories and Arrays

This section describes how memories and arrays are handled in Catapult. It is very important to understand that Catapult determines whether memories are internal or external based on where they are with respect to the top-level design block—this doesn’t relate to on or off chip.

The topics in this section are:

- “[Array Naming](#)” on page 249
- “[Resources and Memory Arrays](#)” on page 250
- “[Mapping Arrays in Physical Memory](#)” on page 251

- “Packing Mode Algorithms” on page 253
- “Word Width Constraint” on page 255
- “Automatic Memory Optimizations” on page 258
- “Splitting Memory Resources” on page 260
- “Merged Read/Write Memory Operations” on page 264
- “Bytewise Write Enable Memories” on page 265
- “Memory Streaming” on page 271
- “DirectInput Resource Type” on page 270

Array Naming

This section describes how Catapult C Synthesis names arrays.

- **Default Naming**

If a parameter is an array and the array is split, then each element of the array generates a port suffixed by the number of the element in the array.

For example, the C function:

```
int56 func(uint54 A[5]) { ... }
```

generates the VHDL entity:

```
ENTITY func IS
PORT (
    A_rsc_4 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
    A_rsc_3 : OUT STD_LOGIC_VECTOR (53 DOWNTO 0);
    A_rsc_2 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
    A_rsc_1 : INOUT STD_LOGIC_VECTOR (53 DOWNTO 0);
    A_rsc_0 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
    func_out : OUT STD_LOGIC_VECTOR (55 DOWNTO 0);
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC
);
END func;
```

- **Multi-dimensional Array**

For multi-dimension input arrays that are split, the suffixes are written in the same order as the original array dimensions. For example, the C function:

```
void func(uint54 A[3][2]) { ... }
```

generates the VHDL entity:

```
ENTITY func IS
PORT (
```

```

A_rsc_2_1 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
A_rsc_2_0 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
A_rsc_1_1 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
A_rsc_1_0 : INOUT STD_LOGIC_VECTOR (53 DOWNTO 0);
A_rsc_0_1 : OUT STD_LOGIC_VECTOR (53 DOWNTO 0);
A_rsc_0_0 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
clk : IN STD_LOGIC ;
rst : IN STD_LOGIC
);
END func;

```

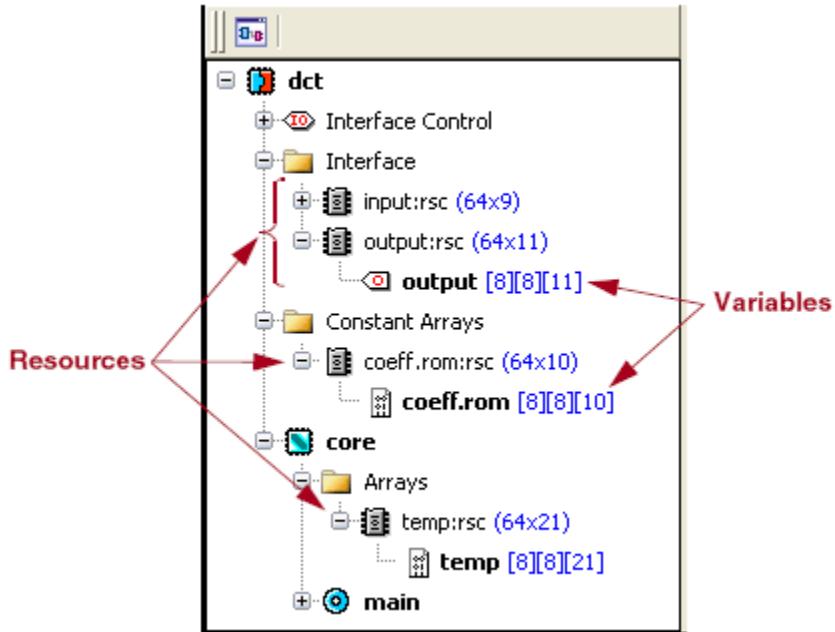
Resources and Memory Arrays

One of the most important concepts in Catapult is the architectural *resource*. This type of resource represents a hardware component that has data moving through it. Catapult automatically creates such resources for the following kinds of variables in the design:

- I/O parameters of the top-level function (I/O ports)
- Global constant array variables (ROMs)
- Local array variables within a process

Figure 5-23 shows how Catapult displays variables and resources in the Architectural Constraints editor.

Figure 5-23. Variable and Resources



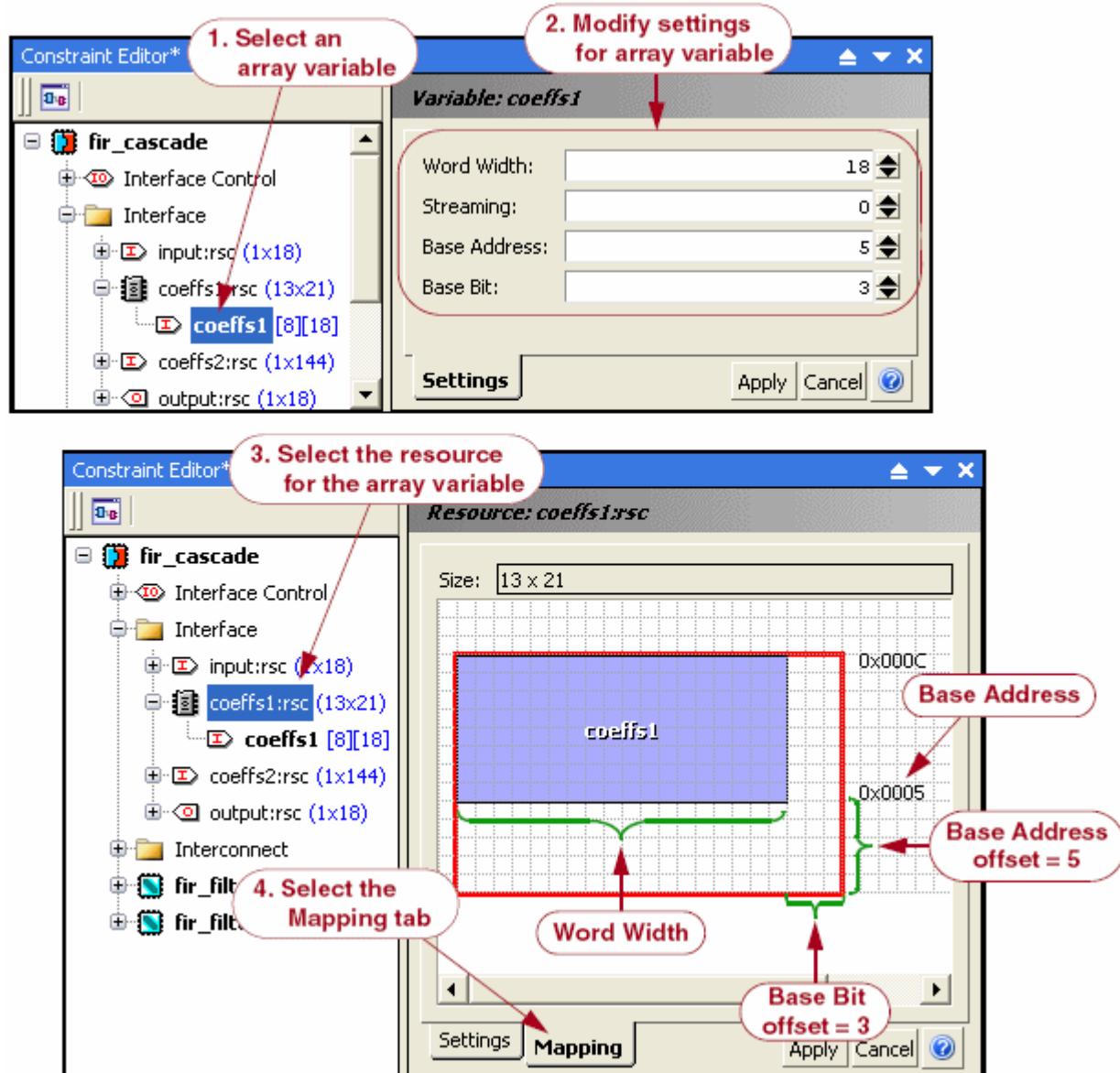
Resource variables can be moved from one resource to another, and multiple variables can share a single resource. Within the Architectural Constraints editor, simply drag and drop the variable on the destination resource. Variables cannot be made more local than where they are initially

declared, but they can always be made more global. Once a variable is moved onto a resource, the width of the resulting resource will be the maximum of the variable bitwidths.

Mapping Arrays in Physical Memory

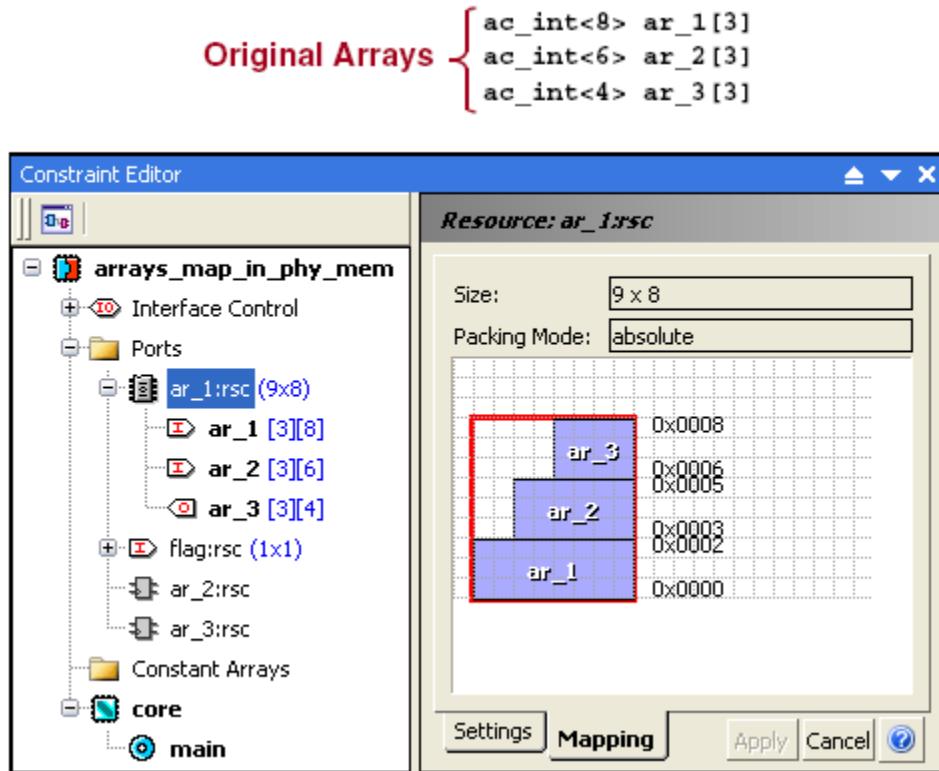
Catapult gives you direct control over where an array variable should be mapped into memory. Any location in the two dimensional space can be specified. You can use the Architectural Constraints editor to set the base bit offset, base address offset, and word width, as shown in Figure 5-24. Or you can set the corresponding directives **BASE_BIT**, **BASE_ADDR** and **WORD_WIDTH** from the command line.

Figure 5-24. Setting a Variable Inside Memory

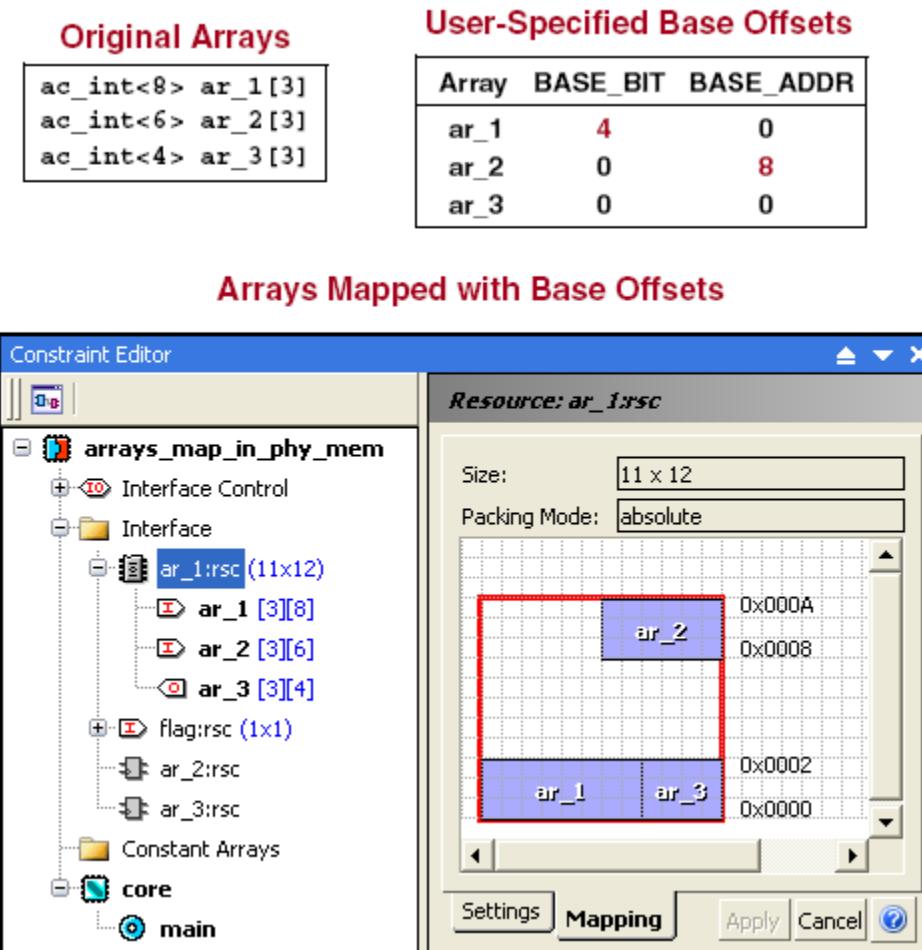


If multiple arrays share the same memory resource, by default the arrays are mapped to memory in the same order that they are declared in the C code. The first variable declared starts at memory location zero.

Figure 5-25. Default Addresses of Arrays Sharing a Resource



You can override the default mapping and arrange the variables in any configuration by specifying explicit base address offsets or word widths. Figure 5-26 shows how to specify the address spaces of three arrays by adjusting their base offsets and word width settings. If gaps are created between variables, the memory in the gap is not used.

Figure 5-26. Bit and Word Location Mapping Example

Packing Mode Algorithms

When multiple variables share the same resource and their memory addresses overlap, Catapult uses one of four “packing mode” algorithms to resolve the address conflict. For information about how to set the packing mode, refer to [“Architectural Constraints on Resources”](#) on page 114 and the [PACKING_MODE](#) directive on page 330.

The algorithms are:

- **Absolute mode**

This mode only applies to memory resources, and it is the default mode. In absolute mode, Catapult attempts to position the resource variables in memory according to their *Base Bit*, *Base Address* and *Word Width* settings. If those settings cause an overlap, then compact mode is used by default. Figure 5-26 illustrates how absolute packing mode works.

- **Side by Side mode**

All variables begin at the same base address, and are packed together along the word-width boundaries. The first variable starts at base bit position zero, and each subsequent variable is offset to begin immediately adjacent to the prior one. See Figure 5-27.

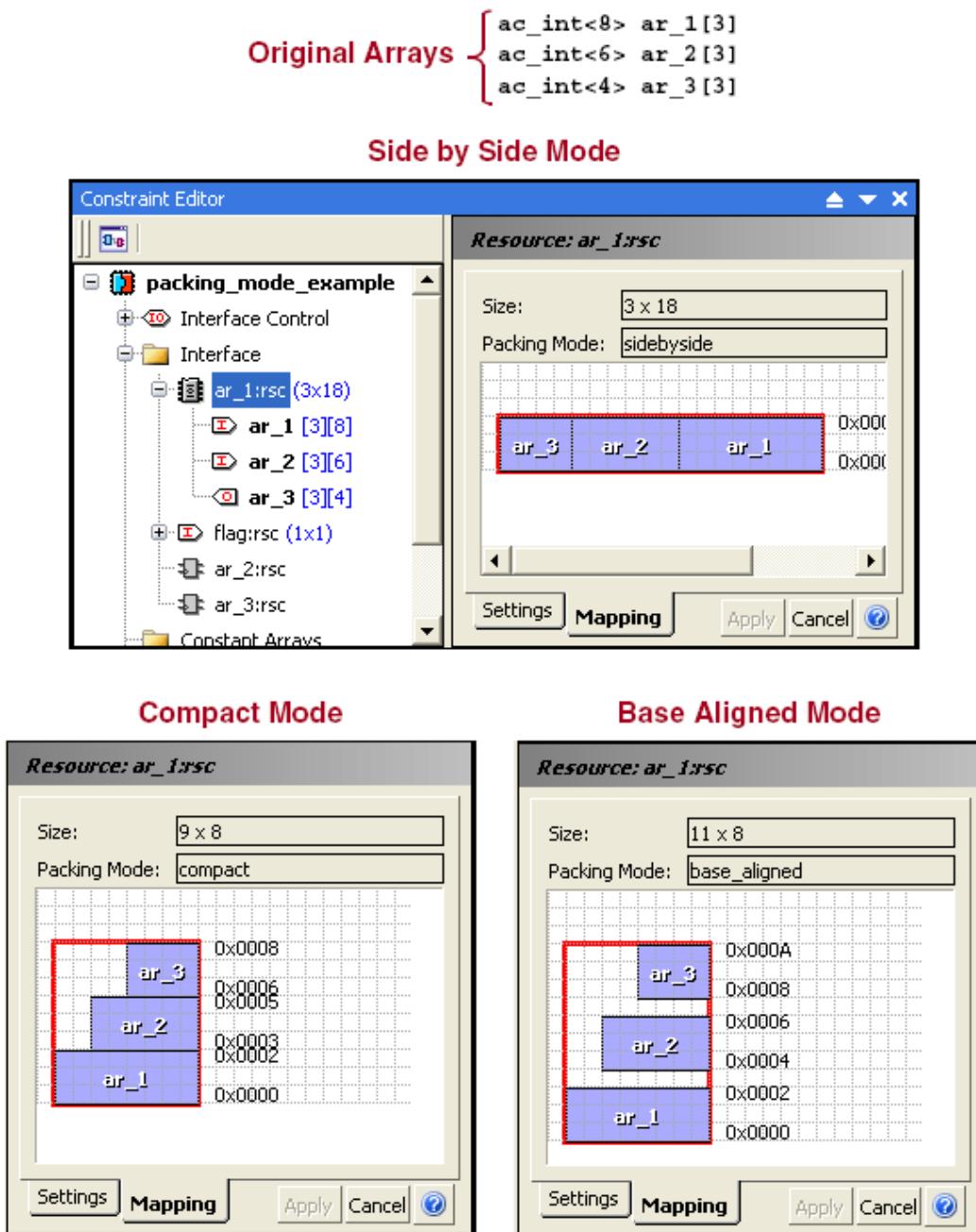
This is the default mode for I/O resources (`mfc_ioprt.*`). For more information about the I/O component types, refer to “[Basic I/O Components](#)” on page 288.

- **Compact mode**

This mode makes the most efficient use of the memory by putting the arrays one after another in the memory. The first variable starts at position zero, and each subsequent variable begin immediately adjacent to the prior one. The disadvantage of this approach is that an adder is needed to access any array after the first one. See Figure 5-27.

- **Base Aligned mode**

This mode puts gaps between the arrays in order to remove the need for the adders to access the arrays. The base address offset of each array is always a power of 2. For example, in Figure 5-27 array “ar_1” starts at address zero, “ar_2” starts at 2^2 (0x0004), and “ar_3” starts at 2^3 (0x0008).

Figure 5-27. Packing Modes for Arrays Mapped to Memories


Word Width Constraint

The word width constraint alters the default word width of resource variables. In the case of memory arrays, this constraint, along with the base bit and base address offset constraints, allows you to control how an array is mapped in the memory resource. In the case of I/O resources, the word width constraint splits the variable into multiple I/O ports.

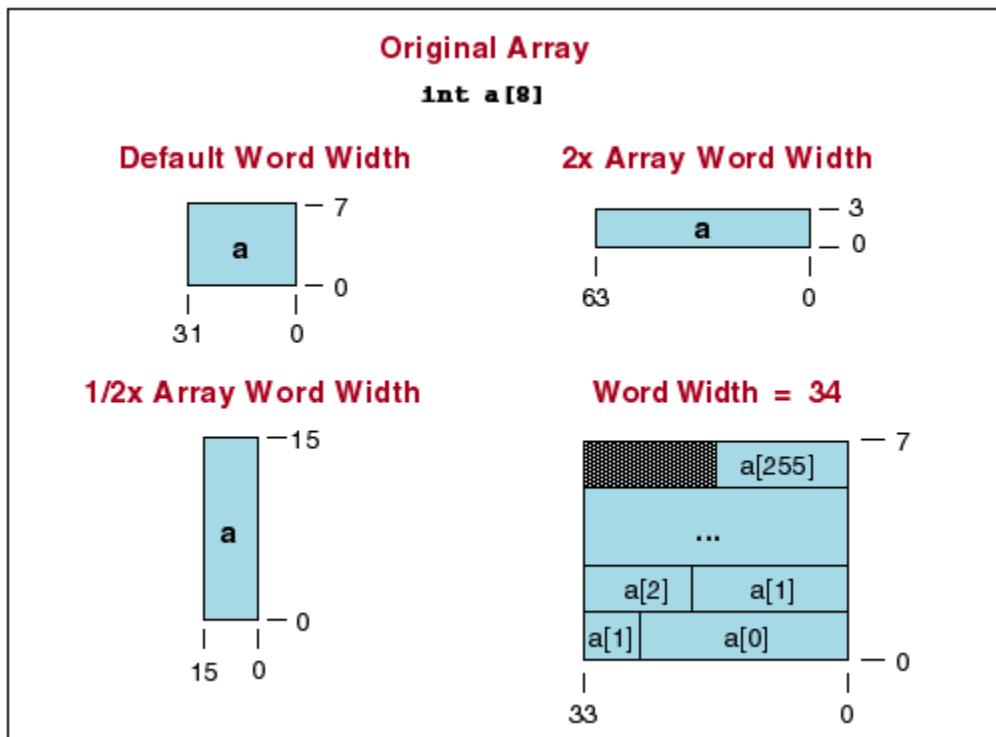
To edit the word width in the GUI, select the resource variable and change the **Word Width** field setting. You can also use the **WORD_WIDTH** directive.

Constraining Word Width of Memory Resources

When mapping an array to memory, you can control the word size by setting the **Word Width** constraint on the array. The default word width of an array is the bit size of its base data type. For example, the array variable “`ac_int<12> a[10][10][10]`” has 1000 words and a default word width of 12 bits.

The word width constraint allows the array to be re-mapped with a different number of bits per memory word, as shown in Figure 5-28. The most common use of the word width constraint is to map more than one array word to one memory word to improve memory bandwidth. The word width can also be used to maximize the memory utilization.

Figure 5-28. Array Word Width



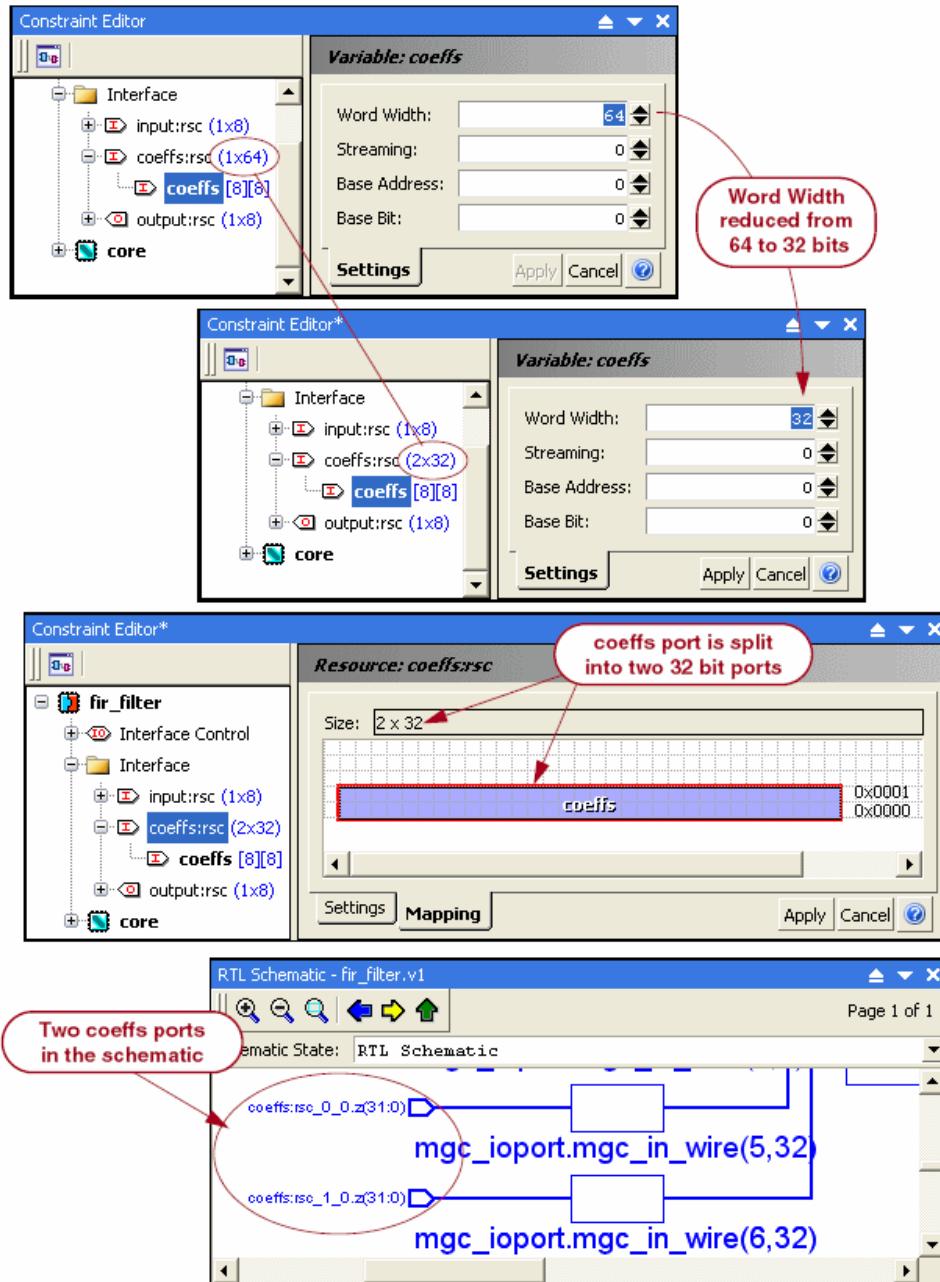
Constraining Word Width of I/O Resources

By default, I/O array variables that are mapped to one of the “`mgc_ioport.*`” resource types are flattened to single word (or port). Therefore the default word width is total number of bits in the array. Setting the word width to smaller number will split the array into multiple ports.

Figure 5-29 illustrates this feature with the example `fir_filter` design provided with the Catapult software.

By default, the array “ac_int<8> coeffs[8]” is mapped to the input resource coeffs:rsc. The resource has a single port that is 64 bits wide (the default word width). As shown in the figure, when the Word Width setting is changed to 32 bits, the array is split into two 32 bit ports. That can be seen in the Mapping tab for coeffs:rsc. Finally, the RTL schematic shows that two coeffs ports appear in the design.

Figure 5-29. Using Word Width Constraint To Split I/O Ports



Automatic Memory Optimizations

Catapult performs a number of memory optimizations automatically.

Minimize Number of Memory Reads/Writes Within a Loop.....	258
Predictive Resource and Memory Mapping	258
Constant Arrays Mapped to ROMs	259

Minimize Number of Memory Reads/Writes Within a Loop

If a memory location is read and written several times within the same loop, the memory is read and written once and the intermediate value is stored in a register.

Catapult can also determine whether two memory reads and writes are always addressing different memory locations. This is important for dual-port memories.

Predictive Resource and Memory Mapping

Catapult automatically derives a default component type for each resource based on factors listed below. The default components are mapped to their respective resources the first time the project enters the Architectural Constraints task. You can use an architectural constraint to change the default component mapped to a resource as described in “[Architectural Constraints on Resources](#)” on page 114, or you can change the [MAP_TO_MODULE](#) directive setting on the resource.

The following factors determine the default component selected for a resource:

- **Available IP Libraries**
The component is selected from the set IP libraries loaded in the project. For information on specifying libraries, see “[Setting Up the Design](#)” on page 96.
- **Type of Resource**
The type of component must be compatible with the type of resource as listed in [Table 5-1](#). For more information about the various component types, refer to “[Basic I/O Components](#)” on page 288 and “[DirectInput Resource Type](#)” on page 270.

Table 5-1. Compatible Resource and Component Types

Resource Type	Component Types
Ports	[DirectInput], mgc_ioport.mgc_chan_in, mgc_ioport.mgc_in*, mgc_ioport.mgc_out*, mgc_ioport.mgc_inout*, RAM
Channels	mgc_ioport.mgc_pipe*

Table 5-1. Compatible Resource and Component Types (cont.)

Resource Type	Component Types
Constant Arrays	[Register], ROM
Local Arrays	[Register], RAM

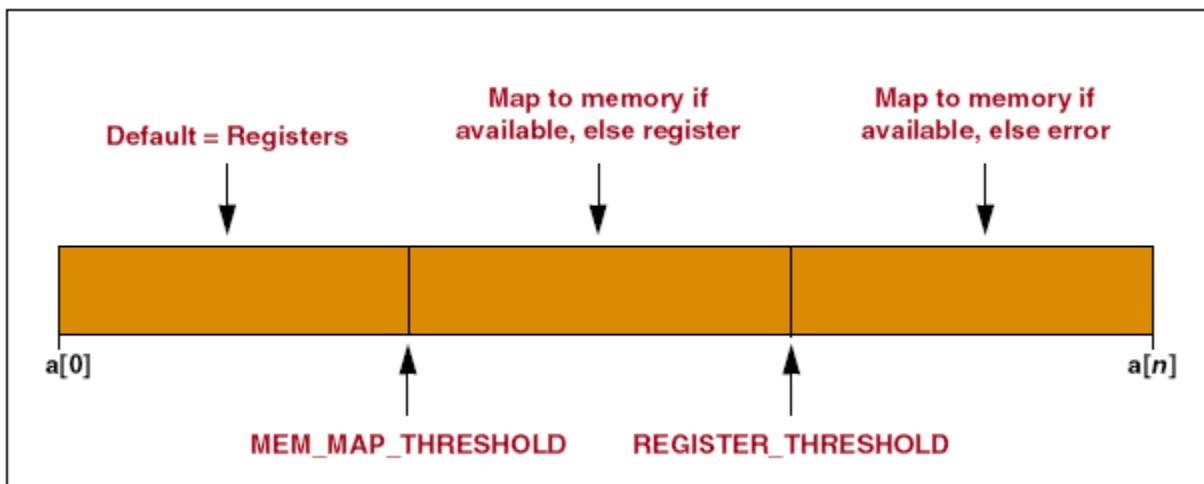
- **Array Variables**

If the resource variable is an array, Catapult maps the resource to a RAM/ROM memory component if the following conditions exist.

- a. Memory components are available in the IP libraries.
- b. The number of elements in the array is greater than the memory threshold setting ([MEM_MAP_THRESHOLD](#) directive).

If the number of array elements is less than the [MEM_MAP_THRESHOLD](#) setting, then the resource is mapped to the “[Register]” resource type.

If no memory components are available and the number of array elements is greater than the register threshold setting ([REGISTER_THRESHOLD](#)), an error is issued.

Figure 5-30. Catapult Thresholds

Constant Arrays Mapped to ROMs

If a ROM library is available to the project, Catapult automatically maps constant array variables to the ROM library component. These appear in the Architectural Constraints editor as global resources inside the **Constant Arrays** folder.

Constant arrays initialized in a loop are not automatically mapped to ROMs. For example, the following code is not be mapped to a ROM:

```
int rom[256];
```

```
for(int i=0; i<256; i++)
    rom[i]=i;
```

The arrays must be initialized in aggregate fashion, as follows:

```
const ac_int<10> coeff [XYSIZE] [XYSIZE] = {
    362, 362, 362, 362, 362, 362, 362, 362,
    502, 425, 284, 99, -99, -284, -425, -502,
    473, 195, -195, -473, -473, -195, 195, 473,
    425, -99, -502, -284, 284, 502, 99, -425,
    362, -362, -362, 362, 362, -362, -362, 362,
    284, -502, 99, 425, -425, -99, 502, -284,
    195, -473, 473, -195, -195, 473, -473, 195,
    99, -284, 425, -502, 502, -425, 284, -99
};
```

Splitting Memory Resources

The memory splitting feature allows you to split a memory resource into an arbitrary number of new memory resources of the same type as the original memory resource. When splitting a memory resource, the words of the original resource are distributed over the new resources according to the specified splitting algorithm(s). All other attributes of the new resource are inherited from the original resource. The specification of the mapping of a variable onto a split resource always refers to the original resource.

Note

 Splitting can only be applied to memories (RAMs or ROMs) and resources mapped to streamed variables cannot be split.

Catapult provides two splitting algorithms that are implemented by the directives [INTERLEAVE](#) and [BLOCK_SIZE](#). The new resources resulting from a split are organized in columns and rows, and their names are of the form <var>:rsc_<row>_<col>. The [INTERLEAVE](#) directive increases the column count, the [BLOCK_SIZE](#) directive increases the row count. Examples of how these directives work are shown in the sections “[INTERLEAVE Splitting Algorithm](#)” and “[BLOCK_SIZE Splitting Algorithm](#)” below.

When Catapult splits a resource, an informational message is sent the transcript during the allocation stage. The message will be similar to the following:

```
# $PROJECT_HOME/matrix_mult.cxx(4): Resource split - resource 'ar_1:rsc'
  (Id: 2) splits into 4 x 1 blocks (MEM-11)
# $PROJECT_HOME/matrix_mult.cxx(4): Memory inferred - resource
  'ar_1:rsc_0_0' (from var: ar_1) mapped to 'ram_Xilinx-VIRTEX-II-
  4_RAMS.B.singleport' (size: 4 x 8). (MEM-4)
# $PROJECT_HOME/matrix_mult.cxx(4): Memory inferred - resource
  'ar_1:rsc_1_0' (from var: ar_1) mapped to 'ram_Xilinx-VIRTEX-II-
  4_RAMS.B.singleport' (size: 4 x 8). (MEM-4)
# $PROJECT_HOME/matrix_mult.cxx(4): Memory inferred - resource
  'ar_1:rsc_2_0' (from var: ar_1) mapped to 'ram_Xilinx-VIRTEX-II-
  4_RAMS.B.singleport' (size: 4 x 8). (MEM-4)
```

```
# $PROJECT_HOME/matrix_mult.cxx(4): Memory inferred - resource
'ar_1:rsc_3_0' (from var: ar_1) mapped to 'ram_Xilinx-VIRTEX-II-
4_RAMSBS.singleport' (size: 4 x 8). (MEM-4)
```

Also, information will appear in the cycle report describing how the addresses of the original resource are distributed over the new blocks

INTERLEAVE Splitting Algorithm

The INTERLEAVE algorithm distributes the addresses of the original resource over the new resources in an interleaving fashion. The format of the command is:

```
directive set <design_path_to_resource> -INTERLEAVE <value>
```

Interleave example 1. Use the **INTERLEAVE** directive to split the resource “ar_var:rsc,” size 16, into 2 new resources.

```
int8 ar_var[16];
directive set /top_func/sub_func_proc/ar_var:rsc -INTERLEAVE 2
```

The result would be two new resources of size 8. The names of the new resources would be “ar_var:rsc_0_0”, and “ar_var:rsc_0_1”. The distribution of words from the original resource would be as follows:

ar_var:rsc_0_0	ar_var:rsc_0_1
-----	-----
ar_var[0]	ar_var[1]
ar_var[2]	ar_var[3]
ar_var[4]	ar_var[5]
ar_var[6]	ar_var[7]
ar_var[8]	ar_var[9]
ar_var[10]	ar_var[11]
ar_var[12]	ar_var[13]
ar_var[14]	ar_var[15]
-----	-----

Interleave example 2. Split “ar_var:rsc” into 3 new resources.

```
directive set /top_func/sub_func_proc/ar_var:rsc -INTERLEAVE 3
```

The result would be three new resources of size 6. The distribution of words from the original resource would be as follows:

ar_var:rsc_0_0	ar_var:rsc_0_1	ar_var:rsc_0_2
-----	-----	-----
ar_var[0]	ar_var[1]	ar_var[2]
ar_var[3]	ar_var[4]	ar_var[5]
ar_var[6]	ar_var[7]	ar_var[8]
ar_var[9]	ar_var[10]	ar_var[11]
ar_var[12]	ar_var[13]	ar_var[14]
ar_var[15]	-----	-----

Interleave example 3. Splitting a multi-dimensional array resource works the same way.

```
int8 ar_var[4][3];  
directive set /top_func/sub_func_proc/ar_var:rsc -INTERLEAVE 3
```

The result would be three new resources of size 4.

ar_var:rsc_0_0	ar_var:rsc_0_1	ar_var:rsc_0_2
-----	-----	-----
ar_var[0][0]	ar_var[0][1]	ar_var[0][2]
ar_var[1][0]	ar_var[1][1]	ar_var[1][2]
ar_var[2][0]	ar_var[2][1]	ar_var[2][2]
ar_var[3][0]	ar_var[3][1]	ar_var[3][2]
-----	-----	-----

BLOCK_SIZE Splitting Algorithm

The BLOCK_SIZE algorithm splits the original memory into new memories of the given size. In contrast to INTERLEAVE, consecutive elements are kept together.

When an array is mapped to a memory (RAM or ROM), the block size must be either zero or greater than 1. If it is mapped to an I/O component, such as a wire, the block size value must be either zero or one. Invalid settings will generate an error later in the flow.

Block Size example 1. Use the **BLOCK_SIZE** directive to split the resource “ar_var:rsc,” size 16, into blocks of size 8:

```
int8 ar_var[16];  
directive set /top_func/sub_func_proc/ar_var:rsc -BLOCK_SIZE 8
```

The result would be two new resources of size 8. The names of the new resources would be “ar_var:rsc_0_0”, and “ar_var:rsc_1_0”. The distribution of words from the original resource would be as follows:

ar_var:rsc_0_0	ar_var:rsc_1_0
-----	-----
ar_var[0]	ar_var[8]
ar_var[1]	ar_var[9]
ar_var[2]	ar_var[10]
ar_var[3]	ar_var[11]
ar_var[4]	ar_var[12]
ar_var[5]	ar_var[13]
ar_var[6]	ar_var[14]
ar_var[7]	ar_var[15]
-----	-----

Block Size example 2. Split “ar_var:rsc,” size 16, into blocks of size 6:

```
directive set /top_func/sub_func_proc/ar_var:rsc -BLOCK_SIZE 6
```

The result would be three new resources of size 6. The distribution of words from the original resource would be as follows:

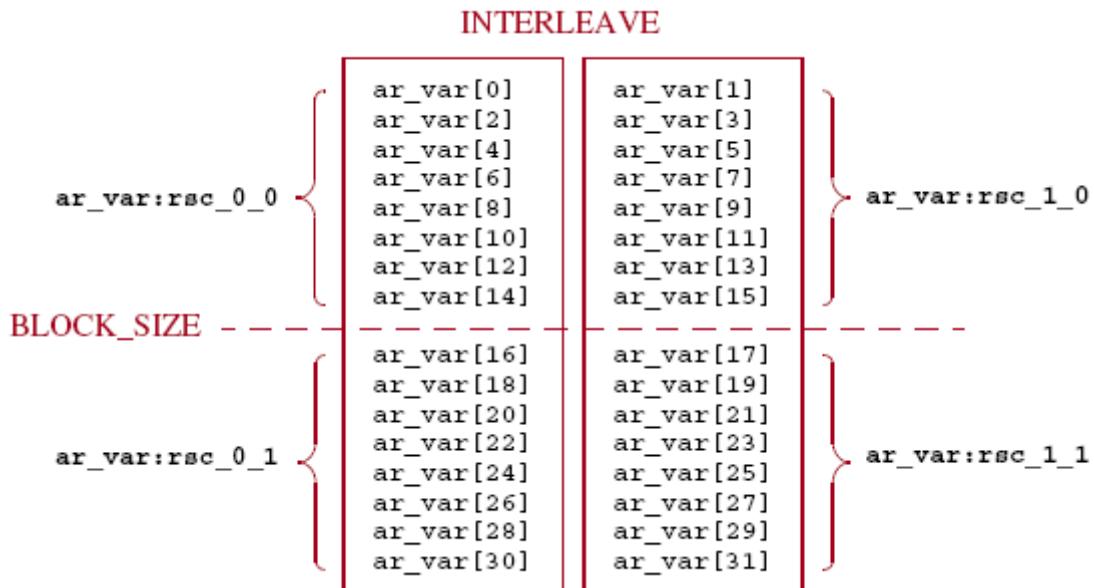
ar_var:rsc_0_0	ar_var:rsc_1_0	ar_var:rsc_2_0
-----	-----	-----
ar_var[0]	ar_var[6]	ar_var[12]
ar_var[1]	ar_var[7]	ar_var[13]
ar_var[2]	ar_var[8]	ar_var[14]
ar_var[3]	ar_var[9]	ar_var[15]
ar_var[4]	ar_var[10]	
ar_var[5]	ar_var[11]	
-----	-----	-----

Combining INTERLEAVE and BLOCK_SIZE

If both directives are applied to a resource, Catapult first applies INTERLEAVE, then applies BLOCK_SIZE to each of the resulting resources from INTERLEAVE. For example, assume a resource “a” has a size of 32 and the following directives are applied:

```
directive set /top_func/sub_func_proc/ar_var:rsc -INTERLEAVE 2
directive set /top_func/sub_func_proc/ar_var:rsc -BLOCK_SIZE 8
```

The result would be four new resources of size 8. First, the INTERLEAVE algorithm creates two resources, “ar_var:rsc_0_0” and “ar_var:rsc_1_0”, each of size 16. Next the BLOCK_SIZE algorithm splits each of those into two size 8 resources, creating “ar_var:rsc_0_1”, “ar_var:rsc_1_1”. The final distribution of words from the original resource would be as follows.



Merged Read/Write Memory Operations

Catapult can optimize an ASIC/FPGA design by automatically merging eligible read/write memory operations into a single cycle for both single and dual port RAM. Merging the memory read and write means they are scheduled in the same cycle as shown in the Gantt chart in Catapult.

To enable the merging behavior, you must do both of the following:

- Configure the RAM component as described in “[Enabling Memory Read/Write Merging](#)” on page 264.
- Satisfy all the rules listed in “[Memory Read/Write Merging Rules](#)” on page 265.

The following example shows how to code a *read-before-write* circular buffer for memory merging when the merging behavior is enabled:

```
void delay_line (
    int &data_in,
    int &data_out
) {

    static int mem[256] ;
    static unsigned char index =0;

    // read/write in a single cycle
    data_out = mem[index] ;
    mem[index] = data_in ;

    index++ ;
}
```

Enabling Memory Read/Write Merging

To configure the RAM component for the read/write merging behavior, use Library Builder to add/modify the RdWrResolution property in the ALL binding. Depending on the value of the property, the eligible read/write operations are merged as follows:

- **0** — Read-before-write (default for RAM components in the Catapult ASIC libraries).
- **1** — Don’t know. Merging is disabled (default for RAM components in the Catapult FPGA libraries).
- **2** — Write-before-read.

For more information, see [Editing RAM Library Properties](#) in the *Catapult C Library Builder User’s and Reference Manual*.

Memory Read/Write Merging Rules

In addition to configuring the RAM component, the following conditions must also be true before Catapult automatically merges memory read/write operations:

- **Read and write operations happen at the same address** — For example, assuming the following data types: `int mem[1024], int inp1, int inp2, int temp`, the following operations are merged:

```
temp = mem[addr] + inp2;
mem[addr] = inp1 + 1;
```

- **No intervening memory accesses between the read and write operations** — The following is an example of an intervening memory access that cannot be merged by Catapult:

```
temp = mem[addr] + inp2;
mem[3] = inp3;
mem[addr] = (inp1 + 1); is not be merged because of intervening
write at index '3'
```

- **One operation dominates the other** — The memory read and write must happen in the same `if` or `else` branch as follows:

```
if (!(i %2)) {
    temp = mem[addr] + inp2;
    mem[addr] = inp1 + 1; is merged by catapult because Mr/Mw are
happening in the same conditional branch.
}
```

- **Old scheduler behavior is disabled** — The old scheduler behavior does not support the merging functionality, and the `OLD_SCHED` directive should be set to false. This is the default setting.

Bytewise Write Enable Memories

RAM libraries created with Catapult C Library Builder can be configured to provide bytewise write capability. All of the RAM components in the libraries shipped with Catapult support bytewise write operations. When a bytewise enable memory is mapped to a resource (the resource must be either an I/O port or a local array), the resource word can be partitioned into a set of slices that can be accessed independently during write operations. A bytewise write operation can write to one or more slices as opposed to writing to the entire memory word.

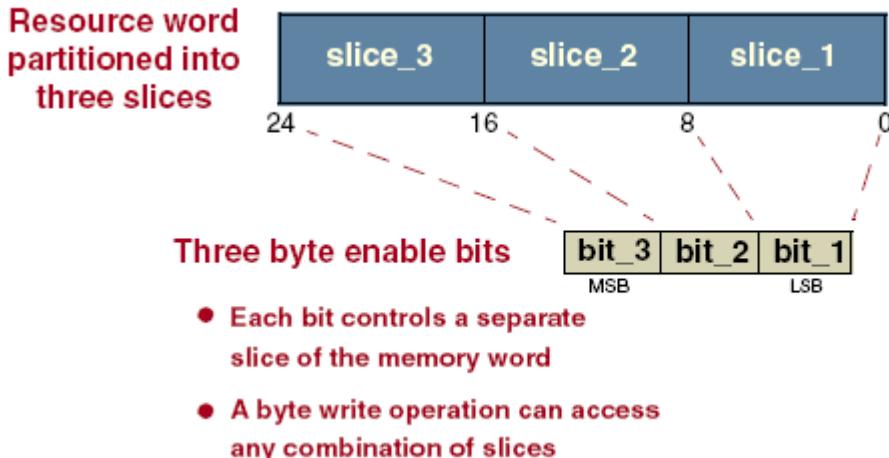
The number of slices can be specified in the Architectural Constraints editor by setting the “`num_byte_enables`” field. Based on that setting, Catapult generates a set of bit flags, one bit for each slice, to control write access. The bit flags are referred to as “byte enables.” The number of byte enables must be a divisor of the resource word width.

`Slice size = Memory Word size / Number of byte enables.`

The i-th byte enable bit controls the slice bits in the range:

```
[ (i + 1)*Slice size - 1 : i*Slice size of the memory word ]
```

Figure 5-31. Byte Write Enabled Memory



The recommended style for coding byte enable write operations is to use the Algorithmic C (AC) Datatype method, `set_slc()`, to write only a subset of the bits in the memory word. The `set_slc()` method takes two arguments, *position* and *value*. The position argument specifies the start bit in the memory word, and is either a constant value or dynamic expression. The value argument specifies the data to be written.

In the following example, assume `mem[]` is a bytewise enable memory. Its word width is 32 bits and the number of byte enables (slices) is four. The example writes to the third slice, which starts at the 16th bit in the word and is 8 bits long.

```
mem[addr].set_slc(16, data_in.slc<8>(8));
```

The *byte enable* inferred by Catapult is: 0100.

In the example above, if you map a non-bytewise enable component to the `mem[]` resource, the write operation is implemented as a read-modify-write sequence of operations and is scheduled over two cycles. The memory read is scheduled in the first cycle and the memory write in the second.

Using bytewise enable functionality make it possible for Catapult to optimize away the memory read operation. If the `set_slc()` *position* aligns with the start of a slice and the size of the *value* argument is a multiple of the slice size, then the memory read operation is not generated. The byte enable bits control which slices are written and the memory write operation can be scheduled in one cycle.

The memory read operation is optimized away if all the following four conditions are true:

- For every slice, either all the bits in the slice are modified by the memory write or they are left unchanged. The following examples assume memory word width = 32 bit, number of byte enables = 4, the *byte enable* inferred by Catapult is: 0010.

Example:

```
mem[addr].set_slc(8, data_in.slc<8>(8))
```

The memory read operation is optimized away because the position is aligned with start of the second slice and all eight bits of the slice are written.

The following two examples show conditions in which the read operation cannot be optimized away.

Example 1:

```
mem[addr].set_slc(8, data_in.slc<4>(8))
```

Catapult generates a read-modify-write sequence because the slice size is 8 and only 4 bits are written.

Example 2:

```
mem[addr].set_slc(10, data_in.slc<6>(8))
```

Catapult performs read-modify-write because position is not aligned with start of any slice.

- For dynamic *position* arguments, if Catapult can statically infer that the start position specified by the set_slc() method is a power of two and is a multiple of slice size, then the read operation can be optimized. For example, given the following code, Catapult can statically prove that “pos” can only be 0, 8, 16, or 24, assuming pos = 5 bits, memory word width = 32 bits, number of byte enables = 4, the *byte enable* inferred by Catapult is: 0010.

```
pos = pos << 3
mem[addr].set_slc(pos, data_in.slc<8>(8))
```

- For dynamic *position* arguments, if the set_slc() method is called directly on the memory write operation, then the read operation can be optimized. The following examples first the coding style that cannot be optimized, followed by the style that can. The examples assumes temp = 32 bits, memory word width = 32 bits, number of byte enables = 2. The inferred byte enable is the concatenation {i(0), !(i(0))}.

Example 1:

```
ac_int<32, true> temp = mem[addr];
for (int i = 0; i < 10; ++i)
{
    if (i % 2)
        temp.set_slc(16, data_in.slc<16>(0));
    else
        temp.set_slc(0, data_in.slc<16>(0));
```

```
    mem[addr] = temp;  
}
```

In the example above, Catapult will not optimize away the memory read. It will only be able to derive the dynamic enable condition. Since the use of “temp” is not necessary, the recommended coding style, shown below, would omit “temp” and call set_slc() directly on memory write operation.

```
for (int i = 0; i < 10; ++i)  
{  
    if (i % 2)  
        mem[addr].set_slc(16, data_in.slc<16>(0));  
    else  
        mem[addr].set_slc(0, data_in.slc<16>(0));  
}
```

The memory read will now be optimized. Also the two mutually exclusive memory write operations alternately write the upper and the lower 16 bits of the memory word. Catapult automatically combines the writes into a single memory write resulting in an unconditional write controlled by the dynamic enable resulting in the following:

```
for (int i = 0; i < 10; ++i)  
{  
    mem[addr] = data_in & data_in  
}
```

The inferred byte enable is the concatenation {i(0), !(i(0))}.

4. For dynamic *position* arguments, the memory word size is a power of two. For the following example, assumes pos = 5 bits, memory word width = 20 bits, number of byte enables = 4.

```
mem[address].set_slc(pos<<2, data_in.slc<8>(0))
```

In this case, 2^{pos} can never equal 20 (word width), therefore the read will not be optimized away.

For cases in which memory reads cannot be optimized away, it is still possible for Catapult to infer the enable condition, which is activated dynamically. The following example assumes temp = 32 bits, memory word width = 32 bits, number of byte enables = 2 .

The if/else logic assigns a new value (16824) to “temp” only if “i” is an even number ((i%2)==0). If “i” is odd, the current value of “mem[addr]” is assigned, effectively leaving “mem[addr]” unchanged. The inferred byte enable is the concatenation { !i(0), !(i(0))}. The “i(0)” refers to the lowest bit of “i” and “!i(0)” implies “i” is an even number.

```
for (int i = 0; i < 10; ++i)  
{  
    if (i % 2)  
        temp = mem[addr]; // i is odd, read mem[addr] and assign to temp  
    else
```

```

        temp = 16824;           // i is even, temp gets a new value
        mem[addr].set_slc(3, temp.slc<29>(3)); // Write temp to mem[addr]
    }
    
```

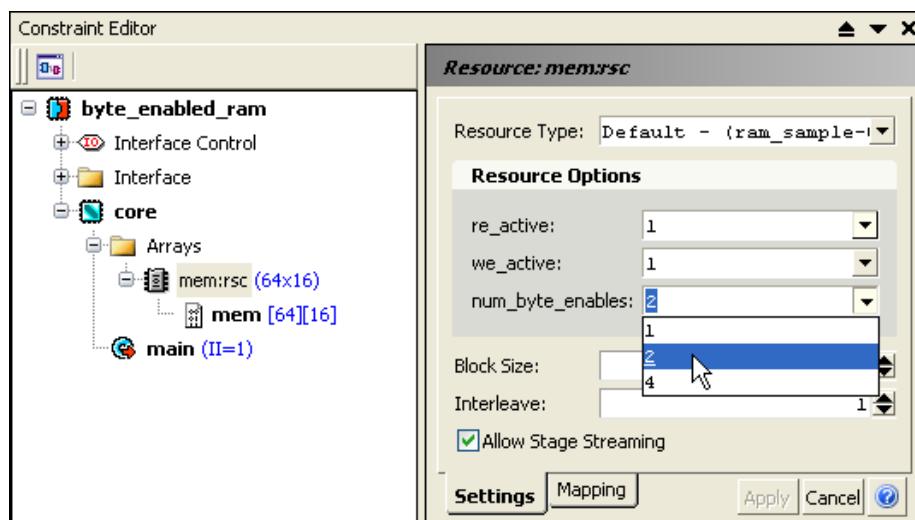
Bytewise enable memories dynamically enables/disables enable bits to perform byte enable write. For each enable bit, Catapults automatically infers the enable condition under which the write operation is performed on the corresponding slice. It then generates the glue logic for the enable condition. If Catapult cannot automatically infer the bytewise enable from the C code, it continues to run as if there was no bytewise enable.

To set the byte enable options from the command line, use the “[directive set](#)” command to set the [MAP_TO_MODULE](#) directive, and include the byte enable options as show in the following example.

```

directive set /byte_enabled_ram/core/mem:rsc \
-MAP_TO_MODULE {{} re_active=1 we_active=1 num_byte_enables=2}
    
```

Figure 5-32. Byte Enable Option in Architectural Constraints Window



Bytewise Example

In the following example the loop cannot be scheduled with an initiation interval (II) of 1 in the absence of bytewise enable. With bytewise enable, the read-modify-write is not generated for the memory write and the design can be scheduled with an II of 1.

```

#pragma hls_design top
void top (uint6 &address, uint1 &ctrl, uint2 &bit_pos, uint4 &data_in,
          uint4 &data_out)
{
    static uint4 mem[64];
    static bool mem_uninit = ac::init_array<AC_VAL_0>(mem, MEMSIZE);
    if (ctrl)
        mem[address] = data_in;
    else
        data_out = mem[address];
}
    
```

```
    mem[address].set_slc(bit_pos, data_in.slc<1>(0));
} else
    data_out = mem[address];
}

directive set /top/top_proc/top_main -PIPELINE_INIT_INTERVAL 1
directive set /top/top_proc/top_mem_rsc \
-MAP_TO_MODULE {{}} num_byte_enables=4}
```

The memory read in the above design is mutually exclusive from the memory write and can be scheduled in the same cycle.

Catapult Library Builder and Bytewise Enable

The RAM components in libraries shipped with Catapult support bytewise enable. They support up to a maximum of four byte enables. Catapult library builder can be used for building RAM components that support more than four byte enables. For more information, see "[Editing RAM Components Parameters](#)" in the *Catapult C Library Builder User's and Reference Manual*. Catapult library builder can also be used for building custom RAM components with byte enables.

SCVerify and Bytewise Enable

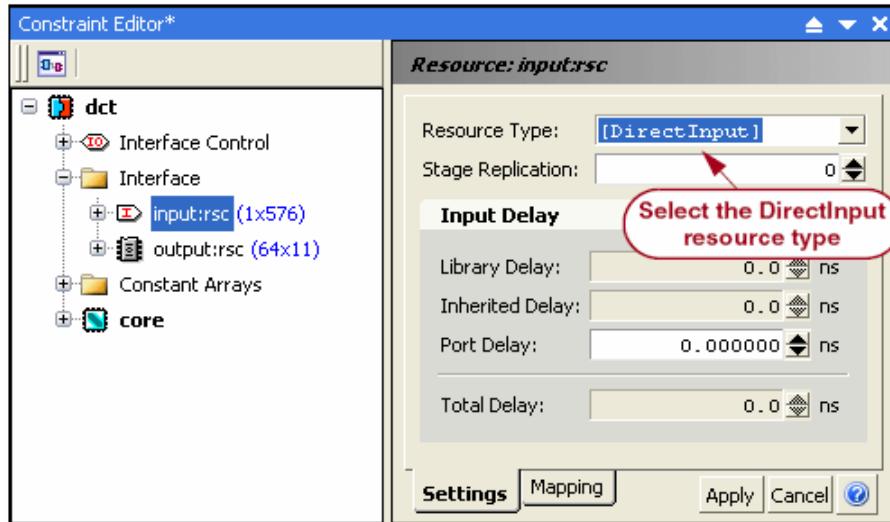
The bytewise enable RAM components in libraries shipped with Catapult are compatible with the SCVerify flow. For verifying custom memory components with bytewise enable capability, you must supply appropriate memory models (for internal memories) and transactors (for external memories) that support the bytewise enable. For more information, refer to "[Using Custom Interfaces with SCVerify](#)" in the *Catapult C Library Builder User's and Reference Manual*.

DirectInput Resource Type

Catapult provides a special input component called a DirectInput that can be used if an initialization signal needs to feed multiple pipeline stages. You may need to modify your C++ code in order to use the DirectInput resource type for this purpose. For more information, see "[Creating a Signal to Feed Multiple Pipeline Stages](#)" in the *Catapult C Synthesis C++ to Hardware Concepts* manual.

To select the DirectInput component, select an I/O resource in the Architectural Constraints editor, then select the DirectInput component on the drop-down list in the Resource Type field. See Figure 5-33.

Figure 5-33. DirectInput Resource Type



Advanced I/O Control for Memories

Catapult has the following advanced I/O control features for memories:

- **Streaming:** This feature lets you convert one or more dimensions on an array into “time.” For more information, see [“Memory Streaming”](#) on page 271.
- **Map-to-Memory Threshold:** Arrays with more elements than the user-specified threshold value are automatically mapped to memory. For more information, see [“Automatic Memory Optimizations”](#) on page 258.
- **Bit and Word Location Control:** This feature gives you the ability to place a variable inside of a memory. For more information, see [“Mapping Arrays in Physical Memory”](#) on page 251.
- **I/O Delay Controls:** This feature allows you to override the default delay values for I/O resources. Adjusted values can be applied globally to all I/O ports, and/or to individual ports. For more information, refer to [“Timing Constraints on Input/Output Ports”](#) on page 294.

Memory Streaming

The **STREAM** directive can be set on an I/O port variable in order to specify that one or more array dimensions are “time.” Setting STREAM to 0 will disable streaming for that variable. When writing C++ code to access a streamed I/O port, be aware of the following rules:

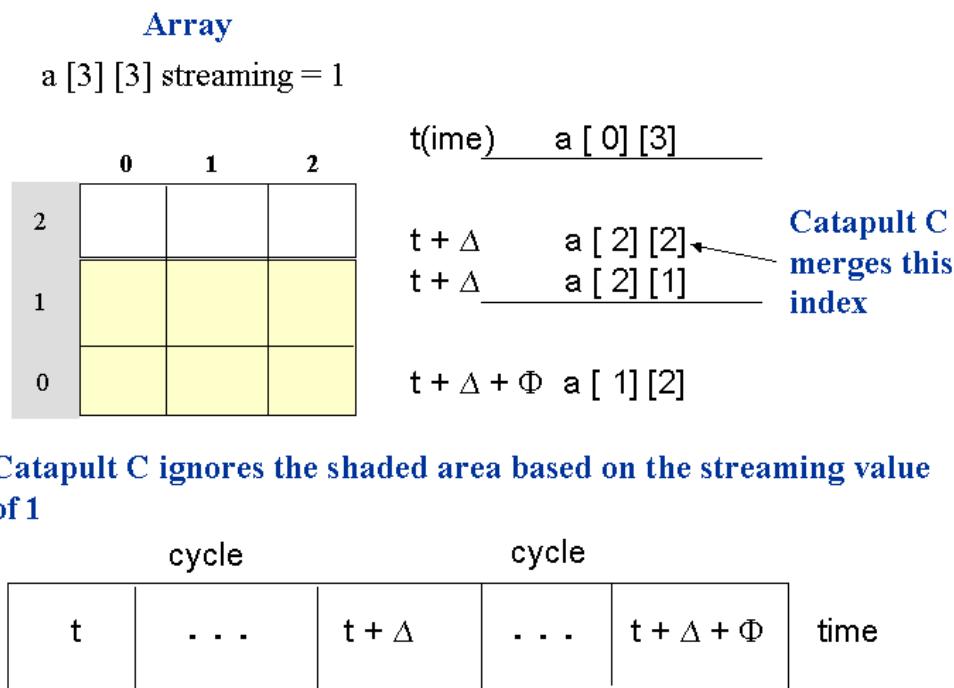
- Comply with all of the same access rules as for pointers
- Accessing the same index twice may merge the reads

- Accessing different indexes will schedule the accesses in different cycles

Streaming Examples

In the first example, we use an array of “ $a[3][3]$ ” and set the streaming value to 1, as shown in Figure 5-34. The tool could read “ $a[2]$ ” twice in different cycles, so it merges the index as shown in the shaded area. Catapult schedules the run as shown in the example below and inserts extra cycles between the runs.

Figure 5-34. Streaming Example



Now, let's look at another example. In this example: $a[3] = a[1] + a[1] + a[6]$, the result could be:

Cycle 1: Read $a[1]$ and add $a[1] + a[1]$

Cycle 2: Read $a[6]$ and add

Cycle 3: Write $a[3]$

But Catapult could read $a[1]$ twice in two different cycles. The only constraint for the access to “ a ” is that $a[x]$ will happen in a different cycle than $a[y]$ if Catapult can't determine that they're equal or that they are not accessed under mutually exclusive conditions.

Setting Streaming in Catapult

To enable streaming on an I/O resource variable, select the variable in the Architectural Constraints editor, then set the value of the “Streaming” field to the number of array dimensions.

General Design Constraints

This section describes how the following general design constraints are handled within Catapult:

- **Clock, reset and enable information:** See “[Setting Up the Interface Control Signals](#)” on page 98, and “[Clocks](#)” on page 273.
- **Scheduler optimizations for area and latency:** See “[Scheduler Optimization](#)” on page 276.
- **Register replication to lower fanout of register outputs:** See “[Register Fanout Optimization](#)” on page 279.
- **Handshake signals:** See “[Process-Level Handshake](#)” on page 280 and “[Transaction-Level Handshake](#)” on page 281 and “[Handshake Signals and SystemC Designs](#)” on page 281.

Clocks

This section describes how Catapult handles clocks. The features covered in this section are clock frequency, percent sharing allocation, and clock reset/enable.

Clock Frequency

When using Catapult, it is important to understand how different elements of the design affect clock speed. One of the most significant impacts on clock speed is muxes, especially the muxes used for control in the design. Note that muxes cannot be made multi-cycle by Catapult. The following items relate to clock frequency within Catapult:

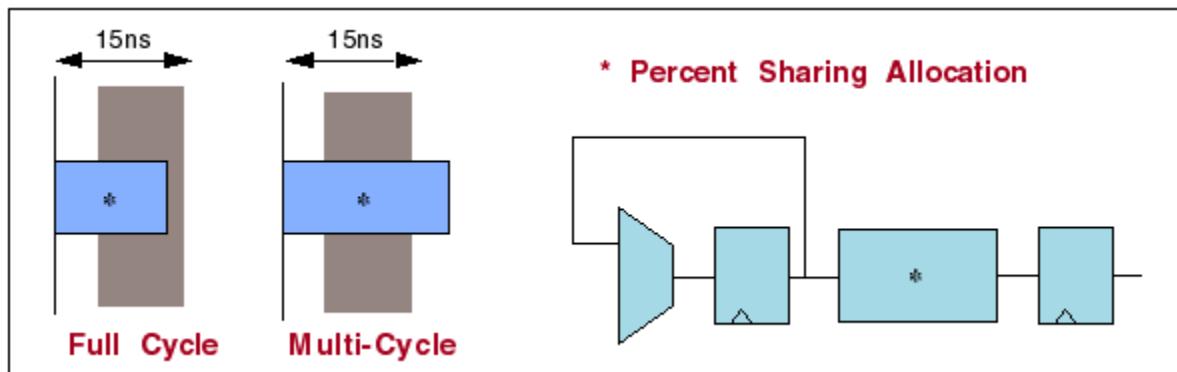
- The clock period is usually limited by the control muxes in your datapath. That means that the control logic usually limits clock speed.
- Multi-cycle synthesis components are supported in downstream RTL compilers supported by Catapult. Constraints are automatically set up within Catapult and passed to RTL synthesis tools.
- Higher clock speed does not mean faster design speed.

Percent Sharing Allocation

Percent Sharing Allocation (PSA) is an estimate for the logic needed to share components and ports. It corresponds to the [CLOCK_OVERHEAD](#) directive. Components that go into the overhead are full-cycle. PSA is the percentage of the overhead held in reserve so the Catapult scheduler does not use it. It is usually 20-30% for FPGAs and 10-20% for ASICs. You can use PSA to improve the clock speed of the design or to reduce the size of the design by reserving more of the clock period for sharing muxes.

PSA can also be used to automatically pipeline some components. If the component takes more than the clock period minus the PSA, then input and output registers are added. If the component takes longer than the clock cycle, it will be scheduled multi-cycle.

Figure 5-35. Percent Sharing Allocation



To modify the PSA setting, open the Architectural Constraint editor, select the process to be changed, enter the new setting in the **Percent Sharing Allocation** field, and click the Apply button to save the change. The command line equivalent is:

```
directive set /dct/core -CLOCK_OVERHEAD <new_value>
```

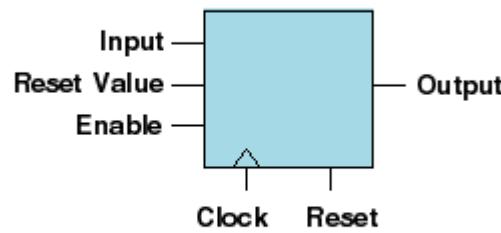
Clock, Reset and Enable Information

C code doesn't include any information about clocks, resets, or enable signals. This information is all added to the design in Catapult. The tool always adds a clock and reset (either synchronous or asynchronous) and an optional enable. Figure 5-36 shows the priority of the different inputs.

There is also an option for you to specify whether the reset clear all of the registers in the design, or only those that need to be reset for correct simulation results. Clearing all of the registers is the safest approach, but, if area is very tight, this can be turned off and non-essential registers will not be reset. Refer to [“Architectural Constraints on the Design”](#) on page 106 and [“RESET_CLEAR_ALL_REGS”](#) on page 336.

Figure 5-36. Clock, Reset and Enable Information

- Always adds a Clock and Reset
 - Either synchronous or asynchronous reset
- Optional Enable
- Clear all registers is safest approach
- Priority:
 - a. Asynchronous Reset
 - b. Synchronous Reset
 - c. Enable
 - d. Input



Clk and Reset Ports Naming

This section describes how Catapult C Synthesis handles port naming for clocks and resets.

- **Default Naming**

By default, Catapult generates two STD_LOGIC ports. Their default names are clk and rst. For example, the C function:

```
void func() { ... }
```

generates the VHDL entity

```

ENTITY clk IS
PORT(
    clk : IN STD_LOGIC ;
    rst : IN STD_LOGIC
);
END func;
  
```

- **Naming if Clk and Rst Name Exists**

If one of the parameters in a design is named clk or rst, Catapult reuses the name. This allows you to connect the clock and reset in non-standard ways. For example, the pre-generated clk is replaced by the defined clk. The C function:

```
void func(int1 clk){ ... }
```

generates the VHDL entity:

```

ENTITY func IS
PORT(
    clk : IN STD_LOGIC_VECTOR (0 DOWNTO 0) ;
    rst : IN STD_LOGIC
);
END func;
  
```

Setting Up Clock, Reset and Enable Information in Catapult

You set up the clock, reset and enable information in Catapult using the Interface Control options in the Constraint Editor window. Click the **Setup Design** task on the Task Bar to open this window. Refer to “[Setting Up the Design](#)” on page 96. To configure reset and enable signals from the command line, use the “[directive set -CLOCKS](#)” command.

Setting Up and Reporting Clock Uncertainty

You can specify clock uncertainty values to account for variances in clock propagation times for post layout designs. The clock uncertainty value specifies a global margin of error to compensate for the variance in the clock pulse due to jitter, skew, and so on. This value is subtracted from the total clock period before other clock constraints are applied. Specify an uncertainty value as a floating number in nanoseconds. The clock uncertainty can be specified for the entire design or for a specific process.

This option is available from both the command line and GUI modes of Catapult C. For more information, see the following topics

- [Setting Up the Interface Control Signals](#)
- [CLOCK_UNCERTAINTY](#)

When reporting the clock uncertainty values in *rtl.rpt*, Catapult rounds the value to 4 digits. For example, when the clock uncertainty value is set to 0.0000521ns, Catapult rounds it off to the nearest 4 digits and reports the value as 0.0001ns. This rounding behavior only affects the uncertainty value reported in the *rtl.rpt* and not the calculation of delay values (where the actual specified values are used).

Scheduler Optimization

This section describes the algorithms used by the scheduler to find a schedule that has the best balance of area and latency. Scheduling is an iterative process that generates a series of increasing optimized schedules. Each iteration produces a viable schedule that is an improvement over the prior iteration. The process ends when no further improvement can be achieved, or if a user-specified limit is reached.

Four directives, [EFFORT_LEVEL](#), [DESIGN_GOAL](#), [AREA_GOAL](#) and [MAX_LATENCY](#), control the amount of effort the scheduler should spend on optimization, the priority goal (area or latency) of the optimizations, and the optimization limits for area size and/or maximum latency. However, the initial schedule (first iteration) does not consider any of the four directives mentioned above. It is simply establishes a starting point by generating the fastest possible schedule regardless of the area size.

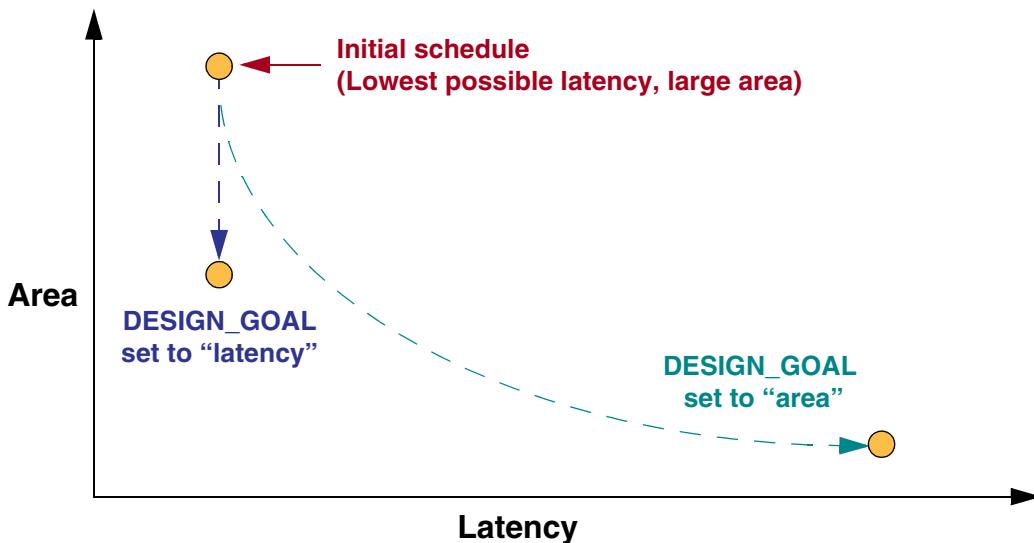
Note

 All other design constraints defined during the design setup and architectural constraints are applied before the initial schedule is generated and they do not change during subsequent iterations.

Figures 5-37 through 5-40 contain a series of graphs that illustrate how the directives **EFFORT_LEVEL**, **DESIGN_GOAL**, **AREA_GOAL** and **MAX_LATENCY** affect the scheduling algorithm. The graphs plot the progression of scheduling iterations, or *design space exploration*, in terms of latency (X axis) and area (Y axis).

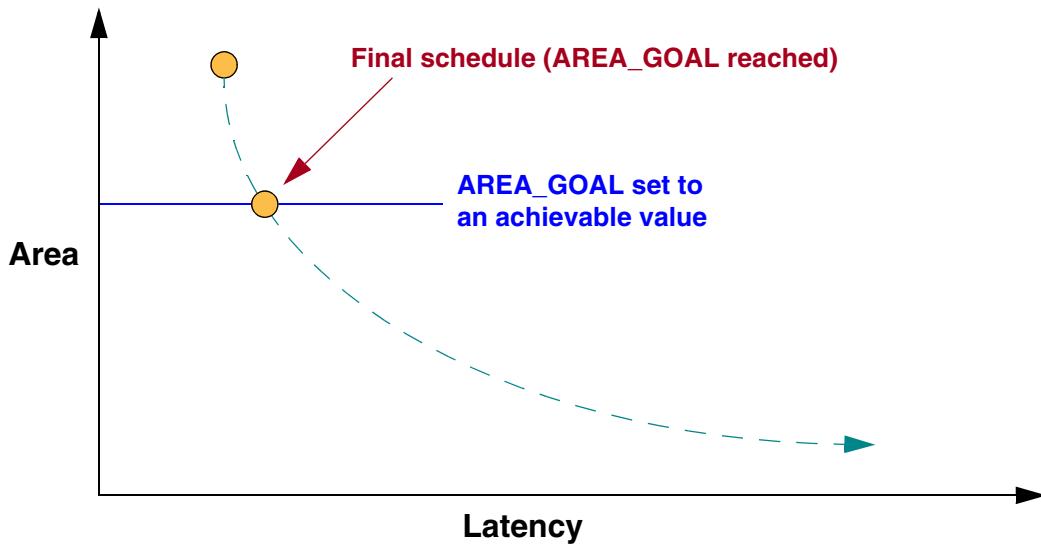
The graph in Figure 5-37 shows the behavior of the **DESIGN_GOAL** directive. The directive specifies whether the scheduler will prioritize area or latency during optimization. When **DESIGN_GOAL** is set to latency, only the area size can be improved because the initial schedule is the fastest possible schedule. When the directive is set to area, the scheduler trades-off latency for smaller area.

Figure 5-37. Affect of DESIGN_GOAL on Scheduling



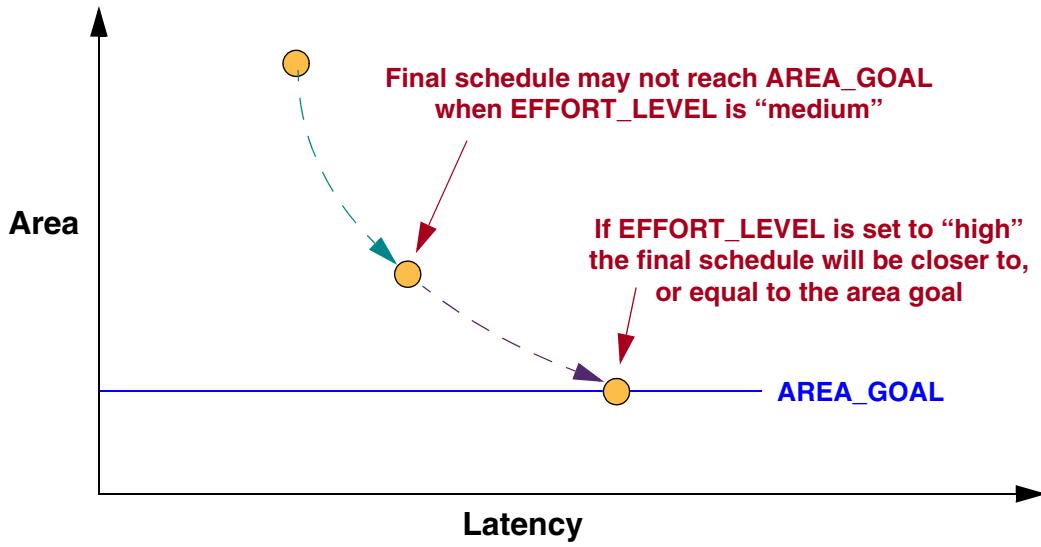
For designs that do not need full area optimization, use the **AREA_GOAL** directive to relax the area size target and thereby reduce the scheduling time. The **AREA_GOAL** specifies a target area size, and the scheduler stops optimizing as soon as it reaches the specified target. Refer to the graph in Figure 5-38.

Figure 5-38. AREA_GOAL Limits Scheduler Optimizations



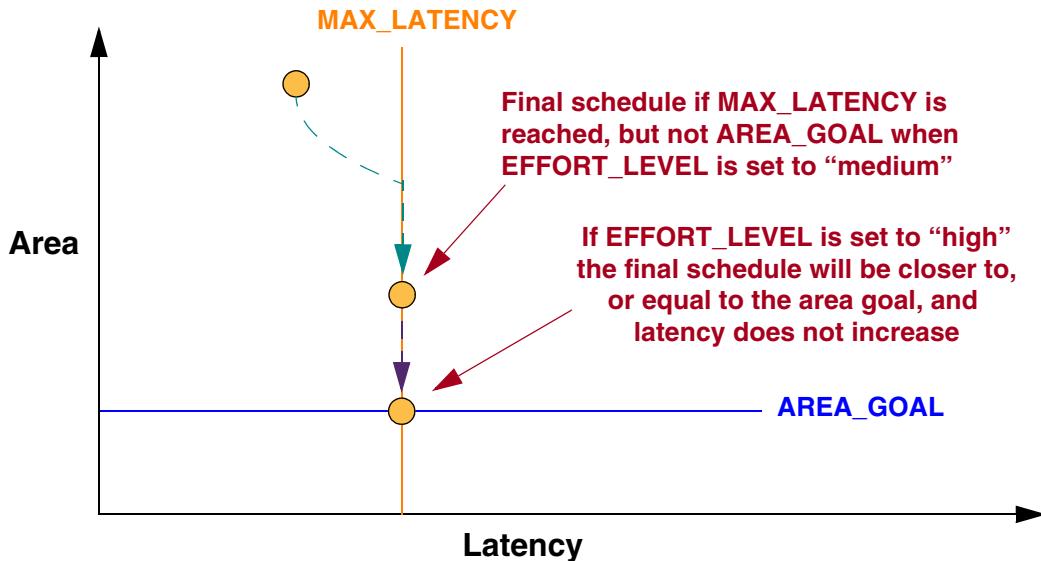
If the **AREA_GOAL** target is too small (unachievable), the scheduler will optimize as much as possible and finish when no further improvements can be made. In that case, the **EFFORT_LEVEL** directive can be set to “high” to increase the optimization effort and perhaps achieve better area results. Refer to Figure 5-39. The default setting for **EFFORT_LEVEL** is not “high” because, in some cases, high effort mode can greatly extend the scheduling time.

Figure 5-39. Affect of EFFORT_LEVEL on AREA_GOAL



It is recommended that the **MAX_LATENCY** directive is always set. That directive limits the scheduler to a user-specified number of cycles (latency). If the maximum latency is reached, the scheduler will continue to optimize the area. Refer to Figure 5-40.

Figure 5-40. Affect of MAX_LATENCY on Scheduling



Register Fanout Optimization

This optimization performs register replication to lower fanout of register outputs to be below a certain threshold. The fanout threshold is specified with the directive **REG_MAX_FANOUT**. It is an integer value that reflects a fanout count rather than an actual fanout load. It does not take into account expected gate sizing or even internal fanout count for components such as multipliers, multiplexers etc. In other words, it assumes that each input bit of a component is already buffered so that the fanout contribution for it is just one. A **REG_MAX_FANOUT** setting of less than 2 will not trigger this optimization.

Usage example:

```
directive set REG_MAX_FANOUT 40
```

The default value is zero. You can change the default value by modifying the “**DefaultRegMaxFanout**” option. In the Catapult Options window (**Tools > Set Options...**), expand “Project Initialization” and select “Hardware” to access the “**Default Max Fanout For Register Replication**” field. Refer to “[Hardware Options](#)” on page 186.

The registers that are replicated fall into two categories:

- FSM outputs: Such outputs tend to have a high degree of fanout and in most cases they feed control logic. The replicated registers are contained within the FSM module.
- Datapath registers.

Note



Registers that are internal to components (such as the last register in a pipelined multiplier) are not targeted by register replication.

Computation of the fanout is done at the bit-level. If any bit of a datapath register or an FSM output surpasses the threshold:

- Fanout is reported.
- The whole register is replicated as many times as necessary to divide the fanout to be below the threshold.
- The fanout for any bit that exceeds the threshold is distributed among the replicated registers.
- Unused bits of the replicated registers are optimized away.

Summary of Limitations:

- Only register replication is performed. Insertion of buffer gates to address fanout is left to the RTL synthesis to perform.
- Fanout threshold is only a count and may not reflect fanout as seen by the RTL synthesis tool.
- Register replication does not take into account additional fanout introduced by register replication.
- Fanout distribution may not break down fanout to a fine enough level of granularity to guarantee an even distribution and thus does not guarantee that all fanout will indeed be below the threshold.
- Fanout distribution does not take into account likely physical placement of fanout components
- RTL synthesis directives maybe required to preserved replicated registers.

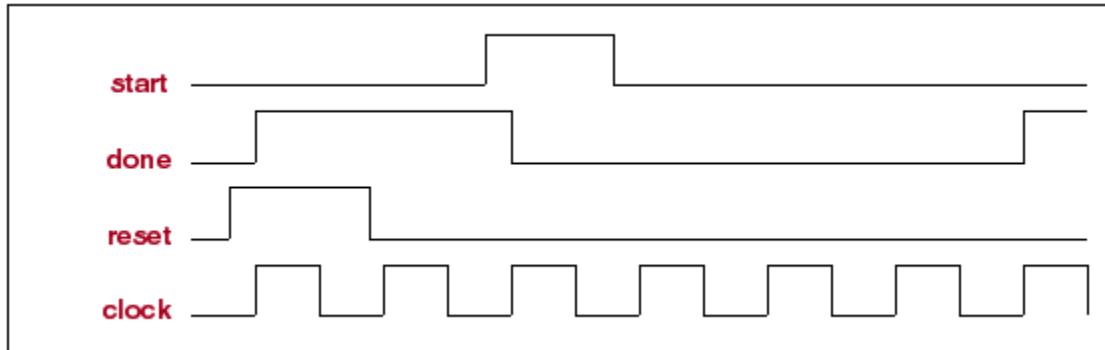
Process-Level Handshake

Catapult can add a start/done handshake to a C++ design. This is similar to a CPU-like handshake. The design sets done to true after reset and waits for the start signal. After the start signal goes high, the done signal goes low until the hardware is “done” processing. The optional start and done flags are:

- Start is an input and the process waits for start to begin
- Done is an output, the process sets done to “1” when it’s waiting for start
- Reset sets done to “1”

Start and/or Done adds one cycle to the design. Figure 5-41 shows the process-level handshake.

Figure 5-41. Process Level Handshake



Transaction-Level Handshake

Enabling the Transaction Done Signal option (on the Interface tab of the Setup Design dialog box) will add corresponding transaction signals for each variable in every function in the design. Each transaction signal is set to “1” whenever its variable is written or read. The signals are identified by the string “_triosy_” in their names. The **TRANSACTION_DONE_SIGNAL** can be useful in getting detailed feedback about when interface pins are written or read.

You can also enable this feature by setting the following directive:

```
directive set -TRANSACTION_DONE_SIGNAL true
```



Note

When the Transaction Done Signal is enabled, the Done flag is ignored.

Handshake Signals and SystemC Designs

SystemC designs define all synchronization signals in the source code, so any additional handshake signals set in Catapult are unnecessary and ignored including:

- Start flag
- Ready flag
- Done flag
- Transaction Done Signal

For more information on writing SystemC for Catapult, see the *Catapult C Synthesis SystemC User’s Guide*.

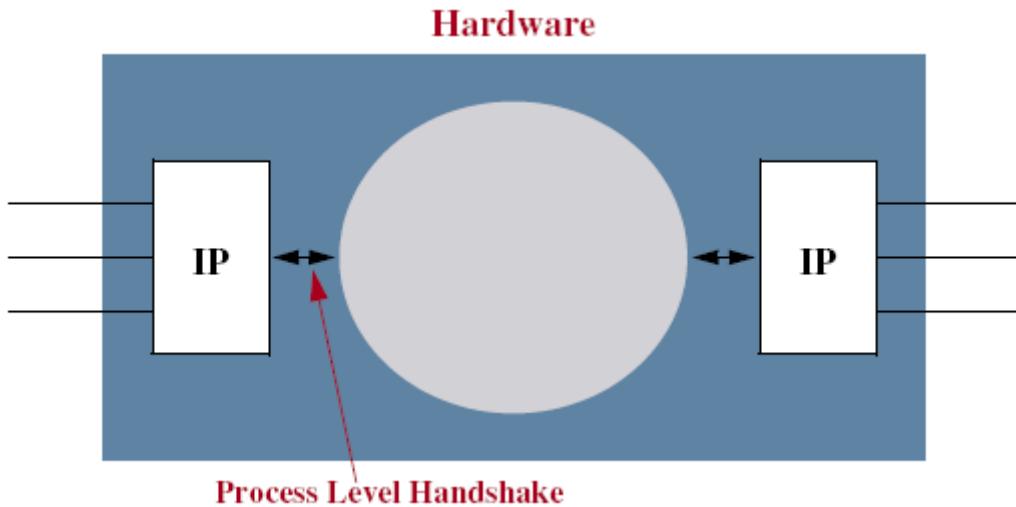
Chapter 6

Interface Synthesis

Interface Synthesis Protocols

Interface synthesis allows for any interface on a design by combining a set of simple internal protocols with RTL interface components. The result is the ability to target any design interface. This chapter covers the process-level handshake protocol, as shown in Figure 6-1.

Figure 6-1. Process-Level Handshake Protocol



Catapult automatically infers interface accesses from pointer parameters of the top level function if the object referenced by the pointer is accessed in the function. The direction of the transfer is derived from the way a port is used in the function. For example:

```
#pragma hls_design top
int top(int *inp, int *outp, int *inout)
{
    ...
    int *p1 = inp;          // no interface operation
    int x = *p1;            // infers read operation on port 'inp'
    int y = *inp;           // no additional interface operation (x == y)
    ...
    *outp = x + 1;          // output operation
    ...
    ++*inout;              // increment infers write after read
}
```

Since an interface access is modeled as a simple assignment, there is no place to specify hardware detail of the data transfer (such as a specific communication protocol). Catapult allows you to select the hardware details of the ports by mapping the port to specific interface

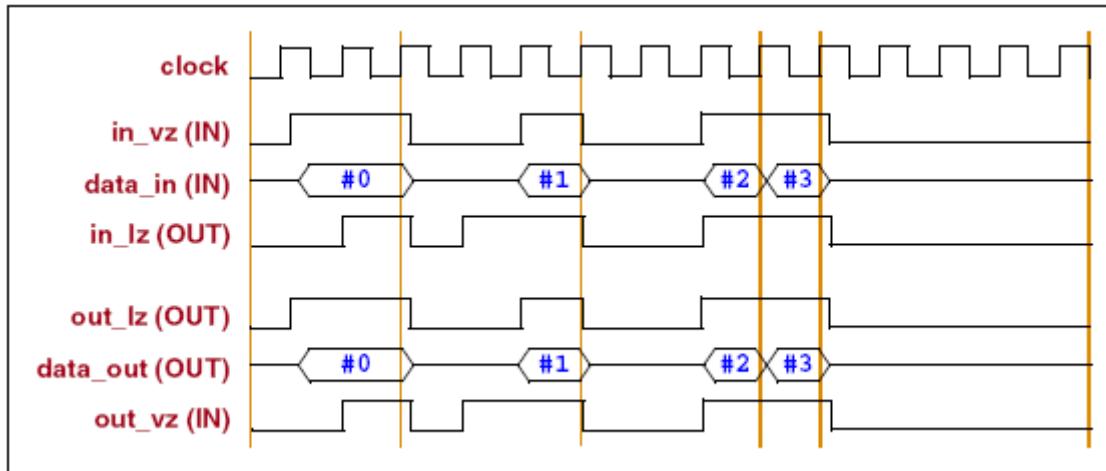
components. The interface component determines the details of the interface and can be selected in the Architectural Constraints editor for each interface port.

The communication between the synthesized data path and the interface component (Process Level Handshake) is built into the tool and consists of any number of control and data lines.

Within Catapult, all of the variables on the hardware interface are automatically mapped to I/O resources and the resources are mapped to components. The tool default is no handshake, the component is just a wire. Reads are read directly from a port and writes translate into signal assignments. For more information, see “[Basic I/O Components](#)” on page 288 and “[Handshake](#)” on page 292.

Figure 6-2 shows the available I/O signals and the timing for the I/O in the process-level handshake. The Data line can be made up of any number of discrete data groups, but the timing of all these groups is the same.

Figure 6-2. I/O Signals and Timing



Rules for I/O Signals

- To connect two processes together, connect **in_vz** to **out_lz** and **in_lz** to **out_vz**.
- The **lz** and **vz** signals are completely symmetrical.
- The design sets **lz** high when it is ready to read/write.
- The outside world sets **vz** high when it is ready to read/write.
- The writer will typically setup the data line whenever his control signal is high. In inout structures, such as buses, the data line will only be valid when both **lz** and **vz** are high.
- The data transfer occurs when both signals are high at the clock edge.

- After this edge, both partners set their control signal to low unless they are immediately ready for the next transfer. (If one partner is ready all the time, it can set its control signal to constant high).
- Both partners have to have a common clock and there is no asynchronous handshaking. Therefore, there can be a data transfer every cycle.

The Catapult-generated design requires that the inputs be stable for the full cycle and the design expects the inputs to be registered by the outside world. This handshake is only valid for data transfers over short distances within the same clock domain. If more complex transfers are required, you are expected to write RTL to handle the communication protocol. Calypto Design Systems also provides RTL source code for the most common types of these protocols.

Port Naming Conventions

Each component is instantiated in the top level design and will create certain signals in the top level interface Entity/Module. The signals names are concatenations of the resource name and the port names. Resource names can be specified in the Architectural Constraints editor window. Port names for the simple provided interface components have the following conventions.

In the following tables, the **vz** port only exists on I/O components that have “_wait” in their names. If the design is ready for a transfer, then a ‘0’ at the **vz** input port will block the process until the data is valid or consumed. The simpler components do not have a **vz** port. For these components, the design expects the outside world to be continuously ready for transfer.

Table 6-1. Input Port Naming Conventions

Name Suffix	Description
<rsc>_z	n bit input data port
<rsc>_lz	1 bit output, design is ready to consume data at clock edge
<rsc>_vz	1 bit input, input is valid at clock edge

Table 6-2. Output Port Naming Conventions

Name Suffix	Description
<rsc>_z	n bit output data port
<rsc>_lz	1 bit output, output is valid at clock edge
<rsc>_vz	1 bit input, outside world is ready to consume data at clock edge

For inout ports, the data port is shared, but each direction has its own handshake. The **z** port tri-state-driver is controlled by (**lzout AND vzout**).

Table 6-3. Inout Port Naming Conventions

Name Suffix	Description
<rsc>_z	n bit tri-state data port
<rsc>_lzin	1 bit output, design is ready to consume data at clock edge
<rsc>_vzin	1 bit input, input is valid at clock edge
<rsc>_lzout	1 bit output, output is ready at clock edge
<rsc>_vzout	1 bit input, outside world is ready to consume data at clock edge

Input Parameters

For each input parameter, an input port is generated. The name of each port is the same as the parameter in the C code.

Default Port

The following example illustrates the default input port for non-pointer datatypes.

Example C Function:

```
void func(int1 A, uint47 B, char C){ ... }
```

Generated VHDL entity:

```
ENTITY func IS
  PORT(
    A : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
    B : IN STD_LOGIC_VECTOR (46 DOWNTO 0);
    C : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    clk : IN STD_LOGIC ;
    rst : IN STD_LOGIC
  );
END func;
```



Note

‘A’ is not a `STD_LOGIC`, but a `STD_LOGIC_VECTOR(0 DOWNTO 0)`. Only the “bool” data type infers a `STD_LOGIC` on the port.

Parameter Contains an Illegal Character

If a C++ variable is illegal in the target language or if its name is a keyword of VHDL (example LOOP, BEGIN, ...) or Verilog, then the prefix “mgc_” will be added to the port.

Example C Function:

```
void func(int3 _a){ ... }
```

Generated VHDL entity:

```
ENTITY func IS
  PORT(
    mgc_a : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
    clk : IN STD_LOGIC ;
    rst : IN STD_LOGIC
  );
END func;
```

 **Note** Catapult displays “_a” within the tool, but in the netlist output it will be “mgc_a” so that it will be legal in the target language.

Input Port Already Exists

If a port prefixed by “mgc_” already exists, then the port will be suffixed by the lowest “_<n>” that does not exist.

Example C Function:

```
void func(uint54 _A , char mgc_A, int1 mgc_A_3){ ... }
```

Generated VHDL entity:

```
ENTITY func IS
  PORT(
    mgc_A_1 : IN STD_LOGIC_VECTOR (53 DOWNTO 0);
    mgc_A : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    mgc_A_3 : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
    clk : IN STD_LOGIC ;
    rst : IN STD_LOGIC
  );
END func;
```

In this example, Catapult displays _A within the tool, but the netlist output will start with mgc_A_1.

Output Parameters

This section describes Catapult output parameter rules.

General Output Rule

If the C function returns a value, the output port name is the name of the function with the suffix “_out” appended.

Example C Function:

```
int3 func() { ... }
```

Generated VHDL entity:

```
ENTITY func IS
  PORT(
    func_out : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    clk : IN STD_LOGIC ;
    rst : IN STD_LOGIC
  );
END func;
```

In this example, the return value for int3 is <function name>_out.

Output Port Already Exists

If an output port already exists, then the port will be suffixed by the lowest “_<n>” that does not exist.

Example C Function:

```
int8 func(uint64 func_out,int func_out_0, int1 func_out_1){ ... }
```

Generated VHDL entity:

```
ENTITY funcIS
  PORT(
    func_out : IN STD_LOGIC_VECTOR (63 DOWNTO 0);
    func_out_0 : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    func_out_1 : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
    func_out_2 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    clk : IN STD_LOGIC ;
    rst : IN STD_LOGIC
  );
END func;
```

In this example, the netlist output will start with “func_out_2” for the return value.

Basic I/O Components

The Catapult synthesis library provides some basic components, which mainly pass the process level handshake to the outside world. These components can be selected in the Architectural Constraints editor. The following predefined I/O components are available in the Catapult default library. Refer to “[I/O Components](#)” on page 647 for descriptions of all component parameters.

Input Components:

```
'mgc_in_wire' (default for inputs)
```

Results in a single input signal ‘z’ without control signal. The input is assumed to be registered outside of the design and ready whenever the process reads it, as shown in the following figure.

```
'mgc_in_wire_en'
```

Results in an input signal ‘z’ and a single control output lz. The input is assumed to be registered outside the design and ready whenever the design reads it. Reading the input is acknowledged with setting lz to ‘1’ at the clock, where the data is taken over.

```
'mgc_in_wire_wait'
```

Results in an input signal ‘z’ with both control signals lz and vz. The input is assumed to be registered outside the design. The design waits until input is valid.

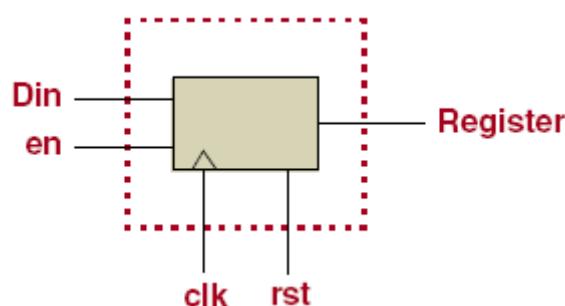
Input Components Examples:

This section provides examples of simple I/O components that can have their interface mapped within Catapult C Synthesis. The dotted lines represent the edges of these components. The process hooked up to the I/O component may have an output register. If the process has a register, it is shown outside of the I/O component. The Process is to the left of the component and the outside world is to the right of the component. Figures 6-3 and 6-4 show examples of input components:

Figure 6-3. Input Component - wire



Figure 6-4. Input Component - register



Output Components:

```
'mgc_out_stdreg' (default for outputs)
```

Results in an output signal ‘z’ without any control signals. The output is registered in the data path. The register value is stable until the next write operation occurs.

`'mgc_out_stdreg_en'`

Results in an output signal ‘z’ and a single control output lz. Each output value is valid for one full cycle and each write is flagged with the signal lz set to ‘1’. The output is undefined in all other cycles, when the control lz is ‘0’ (the output register in the data path could be shared with other registers.)

`'mgc_out_stdreg_wait'`

Results in an output signal ‘z’ with both control signals lz and vz. The output is registered and valid while lz is ‘1’. While the output is valid, the design waits until the signal vz is set to ‘1’.

`'mgc_out_buf_wait'`

Results in an output signal ‘z’ with both control signals lz and vz. The output is registered and valid while lz is ‘1’. The interface component has storage for one output value and serves as a buffer. The process can continue after writing a value and only blocks at the next write if the first value is still not consumed. The interface component has a combinational path through it and does not defer output data by one cycle.

`'mgc_out_fifo_wait'`

Results in an output signal ‘z’ with both control signals lz and vz. The output is registered and valid while lz is ‘1’. The interface component has storage for output values and serves as a buffer. A generic parameter ‘fifo_sz’ determines how many values can be held. The design can continue until the buffer is filled. If fifo_sz is 1, then the component is identical to a “mgc_out_buf_wait.”

`'mgc_out_reg'`

Results in an output signal ‘z’ and a single control output lz. The output is registered in the interface component (not in the data path). Each new output value is flagged with lz = 1 and the output is stable until the next write.

`'mgc_out_prereg_en'`

Results in an output signal ‘z’ and a single control output lz. The output (and the control) is not registered and arrives one cycle earlier than compared to component mgc_out_reg. The output can depend on combinational path through data path and controller and can be combinationally dependent on an input. It will only be valid for the setup time of a register if lz = 1.

Output Components Examples:

This section provides examples of simple I/O output components. Figure 6-5 shows a wire component with output that is registered by a process.

Figure 6-5. Output Component O1

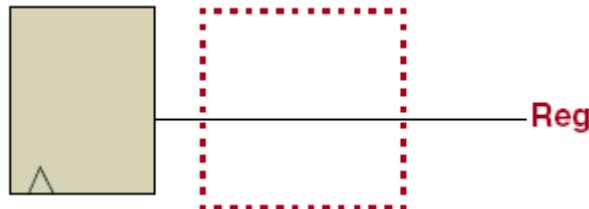


Figure 6-6 shows a wire component in which the output register in the process removed.

Figure 6-6. Output Component O2

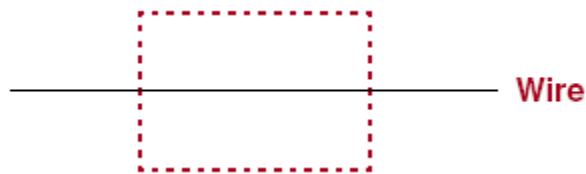
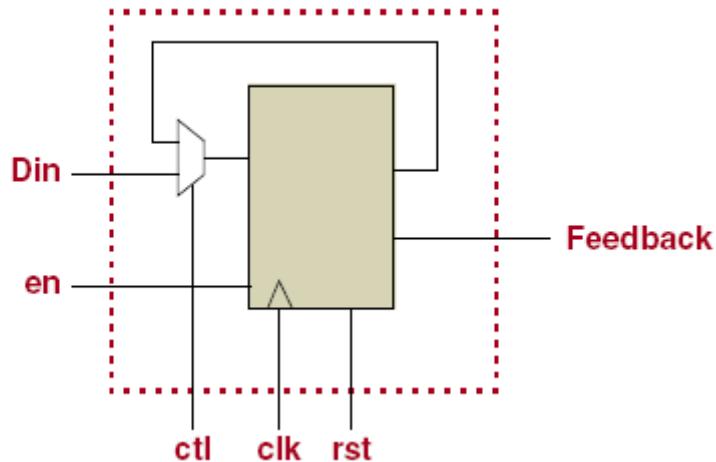


Figure 6-7 shows a register with a feedback mux. The register in the component will need to be connected to the correct clock, reset and enable signals. The output register in the process is removed.

Figure 6-7. Output Component O3



Inout Components:

```
'mgc_inout_stdreg_en' (default for inout)
'mgc_inout_stdreg_wait'
'mgc_inout_buf_wait'
'mgc_inout_fifo_wait'
```

Result in an input port with control signals `ldin` and `ldout`. The components with “wait” property also have `vdin` and `vdout` inputs. The behavior is equivalent to the combination of their matching output components combined with `mgc_in_wire_en` or `mgc_in_wire_wait`. “buf” and “fifo” only apply to the output part of the component.

Inout Components Examples:

Figure 6-8 shows a tri-state component. The “ctl” input must be reset so that the output is high impedance during reset. The output registers are left in the process.

Figure 6-8. InOut Component I/O1

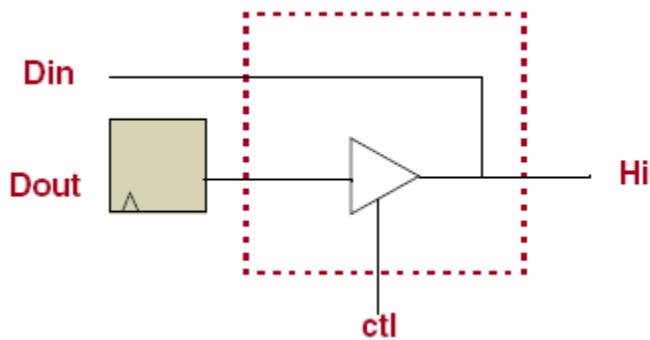
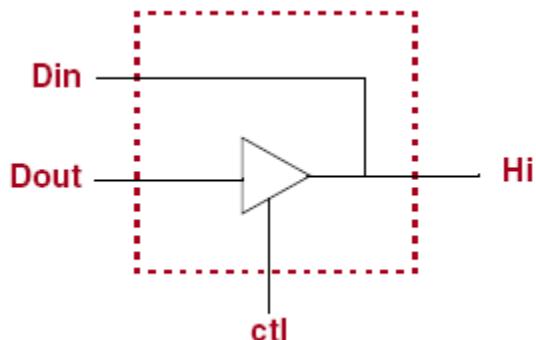


Figure 6-9 shows a tri-state component. The “ctl” input must be reset so that the output is high impedance during reset. The output registers are removed from the process.

Figure 6-9. InOut Component I/O2



Handshake

This section describes the handshake rules for the I/O components examples in the previous section. All of these handshake I/Os are available for all of the basic components. Examples in this section only show the handshake from the process side. However, the I/O component can have a much more complex handshake by having more complex I/O components.

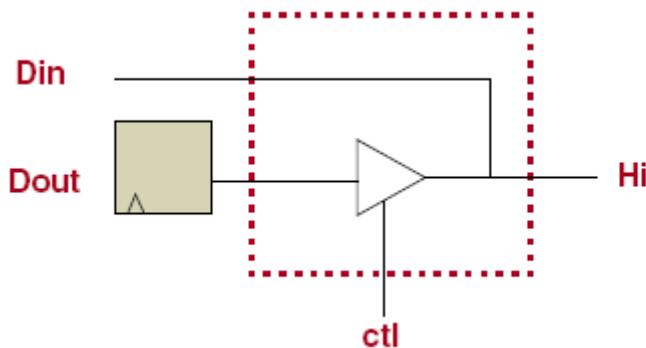
Inputs

In addition to data inputs, the Catapult C Synthesis tool allows any number of control outputs and one control input. All control outputs are set to their active state and the control input is also checked in the first state. The following pseudo-code illustrates this behavior.

```
control_outputs = control_out_active_value;
wait;
while ( control_input != control_in_active)
    wait;
Data = data_input;
control_outputs = control_out_inactive_value;
```

Figure 6-10 shows the implementation of the input and handshake.

Figure 6-10. Input + Handshake



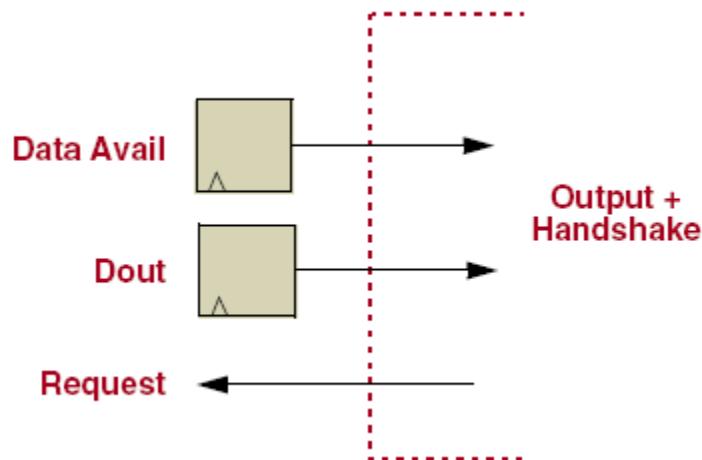
Outputs

In addition to data outputs, the Catapult C Synthesis tool allows any number of control outputs and one control input. All outputs are set to their active state and the control input is also checked in the first state. The following pseudo-code illustrates this behavior:

```
control_outputs = control_out_active_value;
data_output = data_output_value;
wait;
while ( control_input != control_in_active_value)
    wait;
control_outputs = control_out_inactive_value;
```

Figure 6-11 shows the implementation of the output and handshake.

Figure 6-11. Output + Handshake

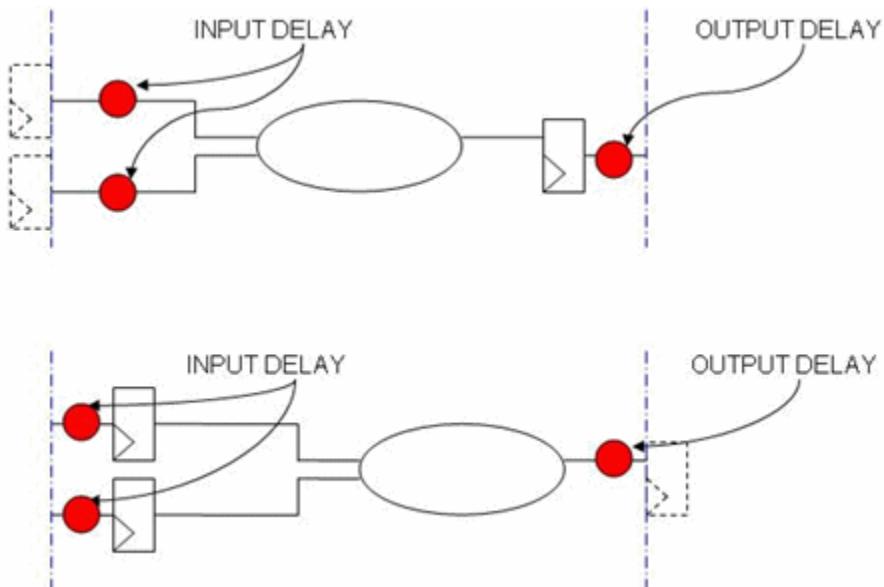


Timing Constraints on Input/Output Ports

Input and output delays can be specified globally or individually on data I/O resources (interface ports). The delay values are used to constrain synthesis, then are passed to the downstream RTL synthesis tool in a timing constraints file.

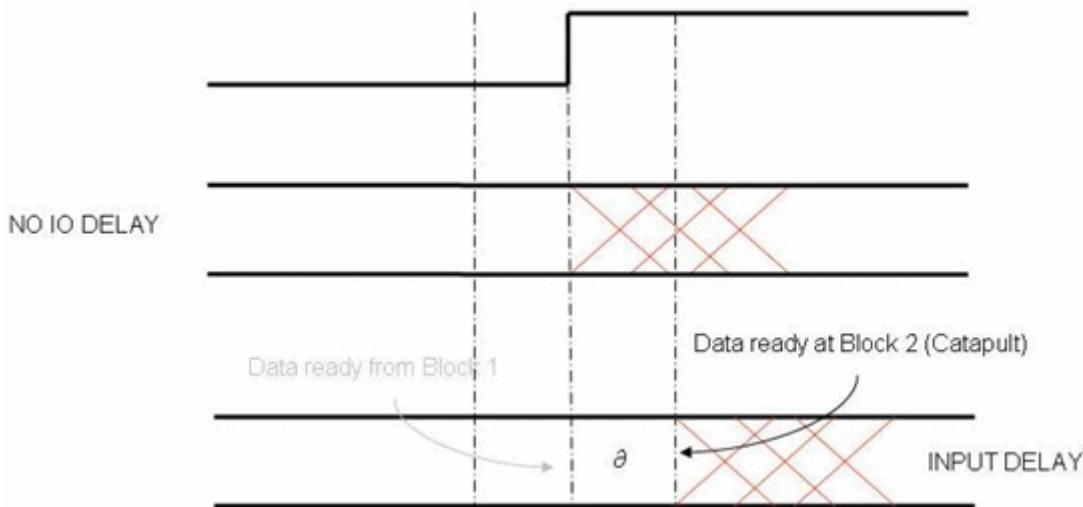
In Catapult, input and output delays are interpreted as shown in Figure 6-12 (dotted lines represent Catapult block boundary).

Figure 6-12. Input and Output Delay Illustration



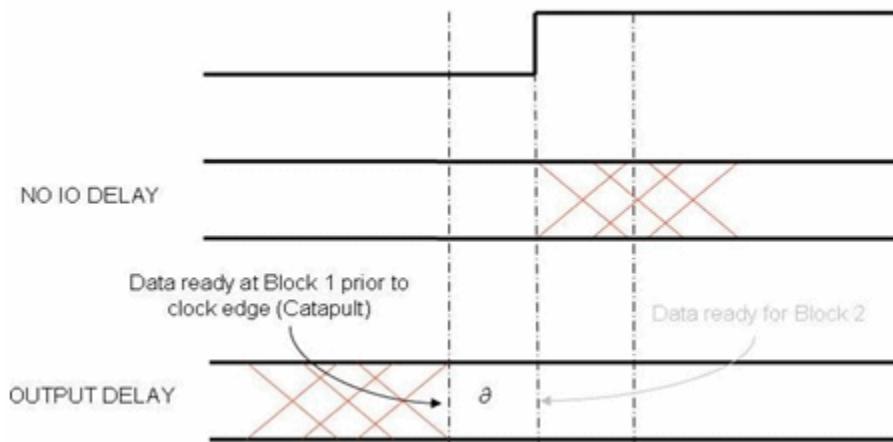
Input delay is the delay after the clock edge when the input is ready at a specific interface. It can be set to constrain the input-to-register and input-to-output paths in the design. Figure 6-13 illustrates the input delay definition.

Figure 6-13. Input Delay Wave Form Diagram



Output delay is the delay prior to the clock edge when the output has to be ready at a specific interface. Output delay can be set to constrain the register-to-output and input-to-output paths in the design. Figure 6-14 illustrates the output delay definition.

Figure 6-14. Output Delay Wave Form Diagram



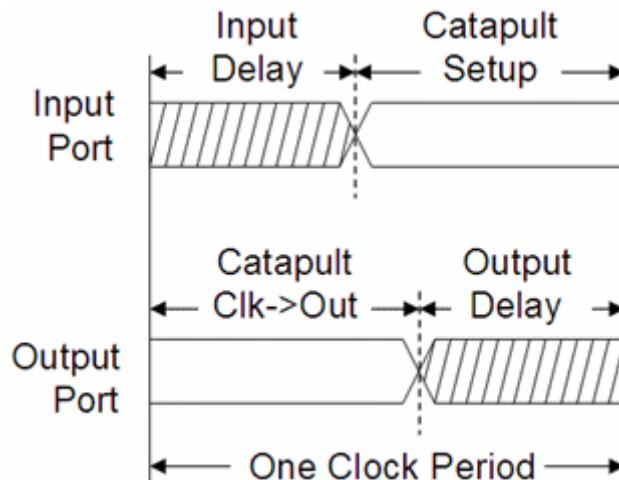
Setting I/O Delay Constraints

The delay constraint value on each I/O port is the sum of three elements: Library Delay + Inherited Delay + Port Delay.

- Library Delay is the default delay constraint predefined in the I/O component library.
- Inherited Delay is the user-specified global delay constraint inherited by all I/O ports.
- Port Delay is the user-specified constraint value set on the individual port.

The total delay constraint values correspond to the portion of the clock period that is unavailable to Catapult. This is illustrated in Figure 6-15 where the shaded regions represent the delay constraints.

Figure 6-15. Timing Diagram Showing Input and Output Delay



The directives **INPUT_DELAY** and **OUTPUT_DELAY** specify global and port-specific delay constraint values. The command line syntax is as follows:

```
directive set /<design_path> -INPUT_DELAY <value>
```

where **<design_path>** is either the name of the top-level process (global setting) or the path to an I/O port resource. The **<value>** argument is a real number. For example:

```
# Global Input Delay
directive set /top_fn -INPUT_DELAY 2.1

# Port Output Delay
directive set /top_fn/out:rsc -OUTPUT_DELAY 0.02
```

The total I/O delay constraint on any interface must be less than the clock period. Values greater than the clock period will generate an error after the “compile” stage in the Catapult work flow. To reduce the total I/O delay for a particular I/O port, set the resource delay (INPUT_DELAY or OUTPUT_DELAY directive) to a negative number. If the total I/O delay for a resource is negative, that I/O delay will be ignored (undefined).

For information about how to set I/O delay constraints from the Catapult GUI, refer to “Architectural Constraints on Processes” on page 112 and “Architectural Constraints on Resources” on page 114.

I/O Delay Constraints Transferred to RTL Synthesis Tools

The I/O delay constraints are automatically transferred to the downstream RTL synthesis tools. For all supported RTL synthesis tools except Precision RTL Synthesis, the constraints are placed in the tool's timing files (*.<tool_name>_timing).

When using Precision RTL Synthesis for FPGA synthesis, the constraints are NOT in the timing file, but are embedded in the Tcl run script (*.psr). The constraints are listed under the heading “## IO TIMING CONSTRAINTS.”

Chapter 7

Directives

Introduction

This chapter describes the directives and pragmas used within Catapult. Directives can be set from the graphical user interface or the command line. Catapult uses the following priority when applying directive and pragma settings:

1. **Directives** — summarized in [Table 7-2](#) and detailed in “[Pre-Defined Directives](#)” on page 307”. Directives can be embedded in source code to direct Catapult on operations such as low-power clock gating.
2. **Pragmas** — summarized in [Table 7-4](#) and detailed in “[Pragmas](#)” on page 341”. Pragmas can be embedded in source code to define hardware control, such as pipelining. Pragmas are interpreted when the algorithm is loaded.
3. **Options** — applied within Catapult to control things such as interface and architecture. For more information on options, see “[Setting Up Catapult Default Options](#)” on page 173.

Directives

Directives constitute a control mechanism used for guiding the synthesis process. The control information is specified by assigning a directives to particular object in the Catapult in-memory design database. To set a directive, you must specify the directive name, the target object, the object type, and a value.

How to Set Directives

There are four ways to set directives.

1. You can declare and set directives in your source files.
2. You can source a Tcl file that contains [directive set](#) commands.
3. You can enter the [directive set](#) command on the Catapult command line.
4. Many directives are implicitly set by the dialog box options in the Catapult GUI. Refer to “[Directives Set from the Graphical User Interface](#)” on page 305 for more information.

Valid Object Types

Table 7-1 identifies the types of objects on which directives can operate.

Table 7-1. Valid Object Types for Directives

Scope	Object Type	Description
Design Hierarchy	Solution	The current (active) solution.
	Design	The top function in the current solution.
	Process	The outermost loop in a design, called the process loop.
	Loop	A loop in the design.
Design Objects	Operation	A scheduled operation.
	Resource	A memory resource. Can be an I/O port, channel or array.
	Signal	A signal in the design, usually on a design port.
	Variable	A resource variable in the design.

Alphabetical List of Directives

Table 7-2 contains a summary of the directives that Catapult supports.

Table 7-2. Alphabetical Summary of Directive

Directive	Description
ARRAY_SIZE	Specifies the default array size to be created for pointer variables on the interface.
AREA_GOAL	Specifies a size goal for area optimization.
ASSIGN_OVERHEAD	Specifies a percentage of the clock period for which sharing is not allowed.
BASE_ADDR	Sets the base address offset of an array mapped to memory. Refer to “Mapping Arrays in Physical Memory” on page 251 and “Packing Mode Algorithms” on page 253.
BASE_BIT	Sets the base bit offset of an array mapped to memory. Refer to “Mapping Arrays in Physical Memory” on page 251 and “Packing Mode Algorithms” on page 253.
BLOCK_SIZE	Splits a memory resource into several new resources of the specified size.
CLOCK_EDGE	Defines the active edge of the clock.

Table 7-2. Alphabetical Summary of Directive (cont.)

Directive	Description
CLOCK_NAME	Defines the name of the clock. Can be any string, but may be modified to be a legal HDL name during netlisting.
CLOCK_OVERHEAD	Specifies the percent of the overhead to reserve for control logic and routing times.
CLOCK_PERIOD	Defines the time, in the units defined by the library, between active edges of the clock. The clock is always assumed to have a 50% duty cycle.
CLOCK_UNCERTAINTY	Specifies a margin of error to compensate for the variance in the clock pulse due to jitter, skew, and so on.
CLOCKS	Defines the clock for a solution.
COMPGRADE	Sets the component selection criteria (speed vs. size) used during allocation.
CREATE_COMBINATIONAL	Specifies whether Catapult synthesizes combinational or sequential logic for a CCORE.
CSTEPS_FROM	This directive sets the number of C-Steps between two operations.
DECOUPLING_STAGES	Insert decoupling pipes when using distributed algorithm for loop pipelining.
DEFAULT_VALUE_OPT	Optimizes design based on default values of component ports.
DESIGN_GOAL	Sets the scheduling optimization goal (area versus latency).
DISTRIBUTED_PIPELINING	Use the distributed pipelining algorithm when pipelining a loop.
DONE_FLAG	Sets the name for the handshaking “done” signal.
EFFORT_LEVEL	Sets the level of effort Catapult applies to design optimization during scheduling.
ENABLE_ACTIVE	Sets the active level of the enable, if there is an enable in the design.
ENABLE_NAME	Sets the name of the enable in the design.
EXTERNAL_MEMORY	Specifies that the memory is external to the design. It can still be on the same chip as the design.
FSM_ENCODING	Set low-power state encoding method for FSMs.

Table 7-2. Alphabetical Summary of Directive (cont.)

Directive	Description
GATE_EFFORT	Specifies the level of optimization for low-power clock gating.
GATE_EXPAND_MIN_WIDTH	Specifies a minimum register bit width for expanded (sequential) clock gate optimization.
GATE_MIN_WIDTH	Specifies a minimum register bit width for clock gate optimization.
GATE_REGISTERS	Enables clock gating optimization.
GEN_EXTERNAL_ENABLE	Adds an enable signal for external memory.
IDLE_SIGNAL	Add idle signals to process blocks and specify a string to be appended to the signal name.
IGNORE_DEPENDENCY_FROM	This directive performs the underlying functionality of the ignore_memory_precedences command.
IGNORE_PROCESS	When set to “true”, the process will not be scheduled. (Flow is not fully implemented.)
INCR_IO_CYCLES	Control the reference data applied to an incremental solution.
INCR_OP_CYCLES	Control the reference data applied to an incremental solution.
INCR_REG_SHARING	Control the reference data applied to an incremental solution.
INCR_RES_SHARING	Control the reference data applied to an incremental solution.
INPUT_DELAY	This directive specifies the global and/or port-specific input delay values.
INPUT_REGISTERS	Determines if registers are created on CCORE inputs.
INTERLEAVE	Splits a memory resource into several new resources and distributes elements among new resources in interleaving fashion. For information about how to use this directive, refer to “Splitting Memory Resources” on page 270. Also see the BLOCK_SIZE directive.
ITERATIONS	Specifies the number of times a loop iterates.
MAP_TO_MODULE	Maps a resource to the specified library component or other special resource types.
MAXLEN	Specifies the maximum number of cycles allowed in a process.

Table 7-2. Alphabetical Summary of Directive (cont.)

Directive	Description
MAX_LATENCY	Specifies the maximum latency (cycles) allowed for the design.
MEM_MAP_THRESHOLD	Any array that has a number of elements greater than or equal to this number is mapped to memory by default. For more information, refer to “Predictive Resource and Memory Mapping” on page 258.
MERGEABLE	Specifies whether or not loops are eligible to be merged.
MIN_CSTEPS_FROM	This directive is replaced by the CSTEPS_FROM directive.
NO_X_ASSIGNMENTS	Specifies whether or not Catapult-generated HDL can assign ‘X’ values (“don’t care” values).
OLD_SCHED	Set the default scheduling algorithm.
OPT_CONST_MULTS	Enables or disables the optimization of constant multipliers.
OUTPUT_DELAY	This directive specifies the global and/or port-specific output delay values.
OUTPUT_REGISTERS	Determines if registers are created on CCORE outputs.
PACKING_MODE	Specifies the default packing mode for packing when several arrays share the same RAM.
PIPELINE_INIT_INTERVAL	Specifies the initiation interval for pipelined loops.
PIPELINE_RAMP_UP	If true, the outputs of the pipeline will be set to zero while the pipeline is filling.
PRESERVE_STRUCTS	Controls whether or not structs that pass through ac_channels are split into separate FIFOs for each data element.
REDUNDANT_MUX_OPT	Removes unnecessary muxes that were added to hold register inputs stable.
REG_MAX_FANOUT	Limits fanout of register outputs to the specified threshold value.
REGISTER_IDLE_SIGNAL	Adds registers to idle signals created by the IDLE_SIGNAL directive.
REGISTER_NAME	Specify a register and variables for sharing.

Table 7-2. Alphabetical Summary of Directive (cont.)

Directive	Description
REGISTER_THRESHOLD	Any array with a number of elements equal to or greater than this threshold value and which is also mapped to a register bank (not mapped to a memory component), will produce an error in order to prevent long runtime. For more information, refer to “Predictive Resource and Memory Mapping” on page 258.
RESET_ASYNC_ACTIVE	Specifies the active level of the asynchronous reset in the design.
RESET_ASYNC_NAME	Specifies the name for the asynchronous reset signal.
RESET_CLEAR_ALL_REGS	If true, all the registers in a design are to be reset.
RESET_KIND	Specifies whether the reset is synchronous, asynchronous or both.
RESET_SYNC_ACTIVE	Specifies the active level of the synchronous reset in the design.
RESET_SYNC_NAME	Specifies the name for the synchronous reset signal.
RESOURCE_NAME	Specify a shared resource name and the operations that should share the resource.
SAFE_FSM	Enables/disables the support for safe FSM flows.
SPECULATE	Enables/disables the speculative execution feature.
STAGE_IO_CNS	Controls the generation of soft constraints for I/O in pipelined designs.
STAGE_REPLICATION	Specifies the maximum number of times a port can be replicated.
START_FLAG	Sets the name for the handshaking “start” signal.
STREAM	Sets the number of dimensions for an array mapped to time.
TECHLIBS	Specifies a list of component libraries used to generate the RTL.
TRANSACTION_DONE_SIGNAL	Adds signals to processes that indicate completion of I/O transactions.
UNROLL	Specifies how many times to unroll a loop.
WORD_WIDTH	Specifies the word width of a resource variable.

Directives Set from the Graphical User Interface

The following lists identifies the directives that can be set from each Catapult GUI element.
Note that some directives can be set in multiple places.

Architectural Constraints Editor

Table 7-1.

GATE_EFFORT	
GATE_EXPAND_MIN_WIDTH	
GATE_MIN_WIDTH	
EFFORT_LEVEL	OUTPUT_DELAY
FSM_ENCODING	REGISTER_IDLE_SIGNAL
GATE_REGISTERS	RESET_CLEAR_ALL_REGS
GEN_EXTERNAL_ENABLE	SAFE_FSM
IDLE_SIGNAL	SPECULATE
INPUT_DELAY	
When the Design is selected	
ASSIGN_OVERHEAD	OLD_SCHED
EFFORT_LEVEL	OUTPUT_DELAY
FSM_ENCODING	SAFE_FSM
GATE_REGISTERS	SPECULATE
GEN_EXTERNAL_ENABLE	REGISTER_IDLE_SIGNAL
IDLE_SIGNAL	RESET_CLEAR_ALL_REGS
INPUT_DELAY	
When a resource is selected	
BLOCK_SIZE	OUTPUT_DELAY
INTERLEAVE	PACKING_MODE
INPUT_DELAY	STAGE_REPLICATION
MAP_TO_MODULE	
When a resource variable is selected	
BASE_ADDR	STREAM
BASE_BIT	WORD_WIDTH
When a process is selected	
AREA_GOAL	GATE_REGISTERS
CLOCK_OVERHEAD	IDLE_SIGNAL
COMPGRADE	MAX_LATENCY

Table 7-1.

DESIGN_GOAL	MAXLEN
EFFORT_LEVEL	SAFE_FSM
FSM_ENCODING	SPECULATE
When a loop is selected	
DECOUPLING_STAGES	MERGEABLE
DISTRIBUTED_PIPELINING	Pipeline_INIT_INTERVAL
ITERATIONS	UNROLL

Setup Design Dialog Box

Table 7-2.

Technology settings	
ARRAY_SIZE	OPT_CONST_MULTS
CLOCK_PERIOD	TECHLIBS
Interface Control	
CLOCK_EDGE	RESET_ASYNC_NAME
CLOCK_NAME	RESET_CLEAR_ALL_REGS
CLOCK_UNCERTAINTY	RESET_KIND
DONE_FLAG	RESET_SYNC_ACTIVE
ENABLE_ACTIVE	RESET_SYNC_NAME
ENABLE_NAME	START_FLAG
RESET_ASYNC_ACTIVE	TRANSACTION_DONE_SIGNAL

Pre-Defined Directives

The following is a list of pre-defined directives that you can set from the source code, a Tcl script, or by entering them from the command line.

ARRAY_SIZE

Specifies the default array size to be created for pointer variables on the interface.

When initially loading design source code, Catapult creates arrays for each pointer variable on the design interface. ARRAY_SIZE specifies the initial size of each array. Catapult then tries to reduce the size of the arrays by assessing how each pointer is used in the design. For example,

the arrays for the "in" and "out" arguments in the following design will be reduced to six element each.

```
#pragma design top
void example (int *in, int *out) {
    for ( int i = 0; i < 6; i++ )
        out[i] = in[i];
}
```

Typically, the arrays size will be reduced to a single element. Catapult issues a warning message if an array cannot be reduced to one element. If the array size cannot be reduced at all, Catapult issues an error message.

Note



You can downgrade the message severity level from an error to a warning. Refer to “[Message Options](#)” on page 176 for more information.

Setting ARRAY_SIZE to 1 will create a single object for each pointer. Setting it to zero will generate an error.

Allowed Values: Positive integers (greater than zero)

Default Value: 1024

Allowed Objects: Design

Last State Allowed: memories

AREA_GOAL

Specifies a size goal for area optimization.

This directive controls the amount of effort Catapult will spend optimizing for area. Valid settings are positive floating point numbers. A setting of “0.0” (default) mean no area goal is set, and the scheduler applies a standard level of effort. If set to a positive number, the scheduler will stop optimizing the area when the total area is less than or equal to the AREA_GOAL.

If the area goal is not met after “medium” EFFORT_LEVEL optimization iterations, then Catapult will issue the following warning message and continue:

```
Warning: The AREA_GOAL of "<num>" could not be met with medium effort
level. Please try setting EFFORT_LEVEL to high.
```

If the EFFORT_LEVEL is “high”, then the warning message is:

```
Warning: The AREA_GOAL of "<num>" could not be met with high effort level.
```

For more information, refer to “[Scheduler Optimization](#)” on page 276. The AREA_GOAL directive applies only when the new scheduler is enabled (see “[Use Old Scheduling and](#)

[Allocation Algorithms](#)” on page 111). See also related directives [EFFORT_LEVEL](#) and [MAX_LATENCY](#).

Allowed Values: Positive floating point numbers

Default Value: 0.0

Allowed Objects: Design -> Process

Last State Allowed: architect

ASSIGN_OVERHEAD

Specifies a percentage of the clock period for which sharing is not allowed.

Positive values help to work around situations in which sharing violates timing. Negative values allow more sharing at the cost of worse slack, which can be corrected in RTL or Physical synthesis.

Allowed Values: Integers less than 100, including negative numbers

Default Value: 0

Allowed Objects: Solution -> Design

Last State Allowed: dpfsm

BASE_ADDR

Sets the base address offset of an array mapped to memory. Refer to “[Mapping Arrays in Physical Memory](#)” on page 251 and “[Packing Mode Algorithms](#)” on page 253.

Allowed Values: Integers greater than or equal to zero

Default Value: 0

Allowed Objects: Variable

Last State Allowed: compile

BASE_BIT

Sets the base bit offset of an array mapped to memory. Refer to “[Mapping Arrays in Physical Memory](#)” on page 251 and “[Packing Mode Algorithms](#)” on page 253.

Allowed Values: Integers greater than or equal to zero

Default Value: 0

Allowed Objects: Variable

Last State Allowed: compile

BLOCK_SIZE

Splits a memory resource into several new resources of the specified size.

Allowed Values: 0 : Disables the directive.
1 : Only valid for I/O components such as a wire.
>1 : Only valid for memory arrays.

Default Value: 0

Allowed Objects: Resource

Last State Allowed: compile

Note

 If the specified block size is greater than the size of the original resource, a new resource of the specified size will be created. In this case, the extraneous elements in new resource will cause corresponding extraneous address bits in the output design.

For information about how to use this directive, refer to “[Splitting Memory Resources](#)” on page 260. Also see the [INTERLEAVE](#) directive.

CHARACTERIZE_OPERATOR

Flag for top-down CCORE flow.

This directive specifies whether or not the top-down CCORE flow will automatically characterize CCOREs when they are created. Refer to “[CCORE Design Flow](#)” on page 597 for more information about CCOREs.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: RESOURCE

Last State Allowed: compile

CLOCK_EDGE

Defines the active edge of the clock.

This directive is a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the CLOCK_EDGE directive is by setting the [CLOCKS](#) directive. Setting the CLOCK_EDGE directive independently is supported for backward compatibility but is not recommended.

Allowed Values: “rising”, “falling”

Default Value: “rising”

Allowed Objects: Solution -> Design -> Process, hierarchical

Last State Allowed: compile

CLOCK_NAME

Defines the name of the clock. Can be any string, but may be modified to be a legal HDL name during netlisting.

Allowed Values: Any string

Default Value: “clk”

Allowed Objects: Solution -> Design -> Process, hierarchical

Last State Allowed: compile

CLOCK_OVERHEAD

Specifies the percent of the overhead to reserve for control logic and routing times.

Typically 20-30% for FPGAs and 10-20% for ASICs. This directive can be set from the GUI by using the “Percent Sharing Allocation” field in the Architectural Constraints editor window.

Allowed Values: Numbers from 0.0 to 100.0

Default Value: 0.0

Allowed Objects: Solution -> Design -> Process, hierarchical

Last State Allowed: compile

CLOCK_PERIOD

Defines the time, in the units defined by the library, between active edges of the clock. The clock is always assumed to have a 50% duty cycle.

This directive is a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the CLOCK_PERIOD directive is by setting the [CLOCKS](#) directive. Setting the CLOCK_PERIOD directive independently is supported for backward compatibility but is not recommended.

Allowed Values: Positive numbers

Default Value: 0.0

Allowed Objects: Solution -> Design -> Process -> Signal -> Qualified Component

Last State Allowed: compile

CLOCK_UNCERTAINTY

Specifies a margin of error to compensate for the variance in the clock pulse due to jitter, skew, and so on. This value is subtracted from the total clock period before other clock constraints are applied. Use a floating number in nanoseconds to define the clock uncertainty value.

The CLOCK_UNCERTAINTY directive applies to all clocks in a design and is supported for backward compatibility.

For new applications, you should use the [CLOCKS](#) directive and CLOCK_UNCERTAINTY as a subordinate property to define clock uncertainty for each clock in a design.

Allowed Values: Positive numbers

Default Value: 0.0

Allowed Objects: Solution -> Design -> Process -> Signal

Last State Allowed: compile

CLOCKS

Defines the clock for a solution.

This directive takes a list argument containing a clock definition. The definition consists of a clock name and a list of properties for that clock. The following properties can be defined:

[CLOCK_PERIOD](#), [CLOCK_EDGE](#), [CLOCK_HIGH_TIME](#), [CLOCK_OFFSET](#),
[CLOCK_UNCERTAINTY](#), [RESET_KIND](#), [RESET_SYNC_NAME](#),
[RESET_SYNC_ACTIVE](#), [RESET_ASYNC_NAME](#), [RESET_ASYNC_ACTIVE](#),
[ENABLE_NAME](#) and [ENABLE_ACTIVE](#).

Example:

```
directive set -CLOCKS { \
    Clock_1 { \
        -CLOCK_PERIOD 20.0 \
        -CLOCK_EDGE falling \
        -CLOCK_HIGH_TIME 25.000000 \
        -CLOCK_OFFSET 0.000000 \
        -CLOCK_UNCERTAINTY 2.0 \
        -RESET_KIND sync \
        -RESET_SYNC_NAME Clock_1_rst \
        -RESET_SYNC_ACTIVE high \
        -RESET_ASYNC_NAME Clock_1_arst_n \
        -RESET_ASYNC_ACTIVE low \
        -ENABLE_NAME {} \
        -ENABLE_ACTIVE high \
    } \
    "Clock_2" \
    { ... } \
    ... }
```

Default values are used for any omitted properties. The default values are derived as follows:

-
1. A new project uses global default values. See “[Interface Options](#)” on page 181 for information about changing the default value.
 2. Whenever the CLOCKS directive is set, the property values for that clock become the default values for the project.

Allowed Values: See description above.

Default Value: See description above.

Allowed Objects: Solution -> Design

Last State Allowed: analyze

COMPgrade

Sets the component selection criteria (speed vs. size) used during allocation.

Allowed Values: “fast”: Selects the fastest component
“small”: Selects the smallest component
“product”: Selects the component with the lowest “product” value, where product = Area x Delay.

Default Value: “fast”

Allowed Objects: Solution -> Design

Last State Allowed: architect

CREATE_COMBINATIONAL

Specifies whether Catapult synthesizes combinational or sequential logic for a CCORE.

When this directive is set to true, Catapult automatically unrolls all loops in the C++ function and removes all output registers. An error will be generated if the function contains any static variables. This directive is most useful in conjunction with the automated CCORE flow (Refer to “[CCORE Design Flow](#)” on page 597).

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Design, CCORE

Last State Allowed: assembly

CSTEPS_FROM

Defines the length of an operation/and or the clock period it occurs via C-Steps.

The arguments for the CSTEPS_FROM directive is a list of operations and C-Step pairs as shown below:

```
directive set <operation_path>
    CSTEPS_FROM {{<operation_path> <comparator> <#C-steps>}}
```

You can specify the length of an operation within a parent loop by defining the number of C-Steps it spans as follows:

```
directive set op3 CSTEPS_FROM {{. == 3}}
```

Where ‘.’ specifies the *op3* operation and 3 specifies the number of C-Steps the *op3* spans.

 **Note** The ‘.’ option only applies to loops.

You can define which clock period within a parent loop that an operation occurs in two ways:

- Specify the number of C-steps between two operations which makes the occurrence of an operation relative to the occurrence of another operation. For example:

```
directive set op3 CSTEPS_FROM {{op2 = 3}}
```

Where *op3* occurs 3 C-Steps after *op2*.

- Explicitly specify a C-Step within the parent loop that an operation occurs. For example:

```
directive set op3 CSTEPS_FROM {{.. == 7}}
```

Where .. specifies the operation currently being defined (*op3*) and 7 explicitly specifies C-Step 7 within the parent loop. This example sets *op3* to occur in C-Step 7 of the parent loop.

Use the Cycle Command to Manage Constraints

The CSTEPS_FROM constraint requires a basic constraint manager to avoid overwriting of constraints as well as being able to query existing constraints. We recommend that you use the [cycle](#) command as your “constraint manager” because you can add, get, set, remove and list operations using the cycle command.

Be aware that it is easy to accidentally overwrite constraints when setting the CSTEPS_FROM directive. The following example illustrates why:

```
directive set op3 CSTEPS_FROM {{op2 = 3}}
directive set op3 CSTEPS_FROM {{loop = 5}}
```

The second constraint will overwrite the first constraint on *op3*. The constraints must be merged as:

```
directive set op3 CSTEPS_FROM {{op2 = 3} {loop = 5}}
```

Allowed Values: <comparator> can be: ==, =, <= , >=, >

Default Value: =

Allowed Objects: Operation

Last State Allowed: architect

DECOUPLING_STAGES

Insert decoupling pipes when using distributed algorithm for loop pipelining.

This directive specifies the maximum number of back-to-back pipeline stages allowed in the pipeline. If the number of stages exceeds that threshold, handshake decoupling pipes are inserted. You may want to use this directive in designs that have a large number of pipeline stages as it may improve the timing results. Only timing for datapath control logic can be improved, not for the datapath itself.

This directive works in conjunction with the [DISTRIBUTED_PIPELINING](#) directive. For a detailed discussion about the distributed pipelining algorithm, refer to “[Distributed Pipeline Synthesis](#)” on page 245.

Allowed Values: Integers greater than or equal to 0.

Default Value: 0

Allowed Objects: Loop

Last State Allowed: architect

DEFAULT_VALUE_OPT

Optimizes design based on default values of component ports.

When set to true, this directive allows Catapult to set the value of component ports to “don’t-care” in those C-steps where the component is inactive or its port-values have no consequence on the design output. The directive does not affect component ports that have a default value specified in the component library. If the directive is set to false, the optimization is disabled. In that case the previous value of the port is allowed to flow through.

Turning on this optimization can improve quality of results in terms of area, latency and timing. However, it could possibly have a negative impact on power.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Design

Last State Allowed: schedule

DESIGN_GOAL

Sets the scheduling optimization goal (area versus latency).

This directive applies only to the new scheduling algorithm introduced in release 2007a. For more information about the DESIGN_GOAL directive, refer to “[Scheduler Optimization](#)” on page 276. The DESIGN_GOAL setting is similar to, but mutually exclusive to the COMPGRADE setting for the old scheduling algorithm. The COMPGRADE setting is ignored when the new algorithm is enabled. Similarly, the DESIGN_GOAL setting is ignored with the old algorithm is enabled.

Allowed Values: “area”, “latency”

Default Value: “latency”

Allowed Objects: Solution -> Process

Last State Allowed: architect

DISTRIBUTED_PIPELINING

Use the distributed pipelining algorithm when pipelining a loop.

This directive can be set on a loop that also has the [PIPELINE_INIT_INTERVAL](#) directive set. Valid settings are “true” or “false” (default). This directive works in conjunction with the [DECOUPLING_STAGES](#) directive. For a detailed discussion about the distributed pipelining algorithm, refer to “[Distributed Pipeline Synthesis](#)” on page 245.

Allowed Values: “true” or “false”

Default Value: “false”

Allowed Objects: Loop

Last State Allowed: architect

DONE_FLAG

Sets the name for the handshaking “done” signal.

If DONE_FLAG is set to an empty value (“ ”), the handshaking signal that indicates “done” is not added to the process.

Allowed Values: Any string

Default Value: Disabled. When enabled, default name is “done”.

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

EFFORT_LEVEL

Sets the level of effort Catapult applies to design optimization during scheduling.

Valid settings are “high” or “medium” (default). It can be set on the design or a process. This directive also influences the [AREA_GOAL](#) and [MAX_LATENCY](#) directives. For more information, refer to “[Scheduler Optimization](#)” on page 276.

The EFFORT_LEVEL directive also controls the activity of the [REDUNDANT_MUX_OPT](#) directive. EFFORT_LEVEL must be set to “high” in order for REDUNDANT_MUX_OPT to be enabled. If EFFORT_LEVEL is “medium,” REDUNDANT_MUX_OPT will have no effect, regardless of its setting.

The EFFORT_LEVEL directive applies only when the new scheduler is enabled (see “[Use Old Scheduling and Allocation Algorithms](#)” on page 111).

Allowed Values: “medium” or “high”

Default Value: “medium”

Allowed Objects: Design -> Process

Last State Allowed: architect

ENABLE_ACTIVE

Sets the active level of the enable, if there is an enable in the design.

This directive is a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the ENABLE_ACTIVE directive is by setting the [CLOCKS](#) directive. Setting the ENABLE_ACTIVE directive independently is supported for backward compatibility but is not recommended.

Allowed Values: “high”, “low”

Default Value: “high”

Allowed Objects: Solution -> Design -> Process, hierarchical

Last State Allowed: analyze

ENABLE_NAME

Sets the name of the enable in the design.

Setting the ENABLE_NAME to an empty string ({ }) will result in no enable in the design. Can be set to any string, but may be modified to be a legal HDL name during netlisting.

This directive is a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the ENABLE_NAME directive is by setting the [CLOCKS](#) directive. Setting the ENABLE_NAME directive independently is supported for backward compatibility but is not recommended.

Allowed Values:	Any string
Default Value:	{ }
Allowed Objects:	Solution -> Design -> Process, hierarchical
Last State Allowed:	analyze

EXTERNAL_MEMORY

Specifies that the memory is external to the design. It can still be on the same chip as the design.

Allowed Values:	“true”, “false”
Default Value:	“false”
Allowed Objects:	Solution -> Resource
Last State Allowed:	architect

FSM_ENCODING

Set low-power state encoding method for FSMs.

FSMs can potentially improve their power consumption numbers by employing the following low-power state encoding methods. Sub-processes inherit the setting on the top process unless individually overridden. For more information, refer to “[FSM State Encoding](#)” on page 591.

Allowed Values:	“none” “binary” “grey” “onehot”
Default Value:	“none”
Allowed Objects:	Design, Process
Last State Allowed:	architect

GATE_EFFORT

Specifies the level of effort used for low-power clock gating optimization. Options include:

- **Normal** — Catapult performs combinational clock gating and uses clock-gated data storage core component models (FIFO/Register file/Channel), where explicit enable signals are placed on all design sequential nodes.
- **High** — In addition to the normal effort optimizations, Catapult performs sequential clock gating by taking into account the clock cycles between registers and propagating enable signals to downstream gating control registers. This results in a strengthened or more selective enable condition. It may also result in additional flip flops being added to the design to store intermediate clock enable states. In addition to the choice of the clock-gated data storage elements, control registers holding the gating information for the main data storage elements are sequentially clock gated (in groups of eight, 1-bit registers).

For more information, refer to “[Gate Register Optimization](#)” on page 605.

Allowed Values: “normal”, “high”

Default Value: “normal”

Allowed Objects: Design, Process

Last State Allowed: architect

GATE_EXPAND_MIN_WIDTH

Specifies a minimum register bit width for expanded (sequential) clock gate optimization. Any registers that fall below the specified value are not targeted for expanded clock gates. Options include:

- **0** — Indicates a 4-bit width, so all registers 4-bits and larger are targeted for clock gating. Default.
- ***positive integer value*** — Specifies a value for the minimum register bit width.
- ***negative integer value*** — Disables clock gating for a given partition or design.

Allowed Values: <0>, <positive_integer>, <negative integer>

Default Value: 4 (minimum width is 4-bits)

Allowed Objects: Design, Process

Last State Allowed: architect

GATE_MIN_WIDTH

Specifies a minimum register bit width for clock gate optimization. Any registers that fall below the specified value are not targeted for clock gates. Options include:

- **0** — Indicates a single bit width, so all registers are targeted for clock gating. In terms of clock gating behavior, a value of 1 is identical to 0. Default.
- ***positive integer value*** — Specifies a value for the minimum register bit width.
- ***negative integer value*** — Disables clock gating for a given partition or design. You can set the GATE_MIN_WIDTH to a negative value and the GATE_EXPAND_MIN_WIDTH to a finite value to enable the expanded clock gating optimization only. For more information, refer to “[Gate Register Optimization](#)” on page 605.

Allowed Values: <0>, <positive_integer>, <negative integer>

Default Value: 0 (no minimum width constraint)

Allowed Objects: Design, Process

Last State Allowed: architect

GATE_REGS

Enable clock gating optimization. Clock gating is a widely used technique for reducing power consumption. When this directive is enabled, Catapult adds explicit clock enable signals to registers allowing clock gate insertion. Clock gates are actually inserted by downstream RTL synthesis tools. For more information, refer to “[Clock Gate Insertion](#)” on page 589.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Design, Process

Last State Allowed: architect

GEN_EXTERNAL_ENABLE

Adds an enable signal for external memory.

This directive applies only to external memories that have an enable pin and have to be stalled from the synthesized design because of wait components. If GEN_EXTERNAL_ENABLE is set to false or the memory does not have an enable pin, the memory will not be stalled and buffer logic inside the Catapult design will be generated. Setting GEN_EXTERNAL_ENABLE to true will drive the enable pin of the memory and remove buffer logic inside the Catapult block.

Catapult will no longer drive the memory’s enable pin to a constant or pass through an input enable signal if GEN_EXTERNAL_ENABLE is set to true, as it did in Catapult 2006 releases.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Solution -> Resource

Last State Allowed: compile

IDLE_SIGNAL

Add idle signals to process blocks and specify a string to be appended to the signal name.

Idle signals are used in design that require low power consumption. Refer to “[Idle Signal Insertion](#)” on page 592 for detailed information about using idle signals. See also “[REGISTER_IDLE_SIGNAL](#)” on page 333 for adding registers to idle signals.

Allowed Values: User-specified suffix string appended to signal name.

Default Value: { }

Allowed Objects: Design -> Process

Last State Allowed: architect

The command syntax is:

```
directive set <path_to_block> -IDLE_SIGNAL <suffix_string>
```

where <path_to_block> is the design path to either the top-level block or a sub-process. The resulting signal name will have the form:

```
<process_name>_<suffix_string>
```

where <process_name> is the name of the process on which the directive is being set. This form applies to all sub-processes, and for single clock designs it applies to the top block also.

<suffix_string> is a user-defined string appended to the signal.

IGNORE_DEPENDENCY_FROM

This directive performs the underlying functionality of the [ignore_memory_precedences](#) command.

You should not set this directive directly. The ignore_memory_precedences command is the user interface for the directive. When the command is used, the directive setting will appear in the `directives.tcl` file in the solution directory.

IGNORE_PROCESS

When set to “true”, the process will not be scheduled. (Flow is not fully implemented.)

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Process

Last State Allowed: architect

INCR_IO_CYCLES

Control the reference data applied to an incremental solution.

When set to “false” the I/O access cycle constraints are excluded from the incremental solution.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Solution > Design > Process, hierarchical

Last State Allowed: architect

INCR_OP_CYCLES

Control the reference data applied to an incremental solution.

When set to “false” the operation cycle constraints are excluded from the incremental solution.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Solution > Design > Process, hierarchical

Last State Allowed: architect

INCR_REG_SHARING

Control the reference data applied to an incremental solution.

When set to “false” the register sharing constraints are excluded from the incremental solution.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Register variables

Last State Allowed: schedule

INCR_RES_SHARING

Control the reference data applied to an incremental solution.

When set to “false” the resource sharing constraints are excluded from the incremental solution.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Operations

Last State Allowed: dpfsm

INPUT_DELAY

This directive specifies the global and/or port-specific input delay values.

When INPUT_DELAY is set on the top design, the delay value is globally inherited by all input ports in the design. When the directive is set on an input port resource, it modifies the total delay for the resource as follows:

Total delay = library delay (from component library) + inherited delay + port delay.

To reduce the total delay for a port resource, specify a negative port delay value.

Allowed Values: Real numbers

Default Value: 0.0

Allowed Objects: Design, Resource

Last State Allowed: compile

See also “[OUTPUT_DELAY](#)” on page 330. For an overview of how I/O delays are defined and applied in Catapult, refer to “[Timing Constraints on Input/Output Ports](#)” on page 294.

INPUT_REGISTERS

Inserts registers on CCORE inputs.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Process

Last State Allowed: compile

INTERLEAVE

Splits a memory resource into several new resources and distributes elements among new resources in interleaving fashion. For information about how to use this directive, refer to “[Splitting Memory Resources](#)” on page 260. Also see the [BLOCK_SIZE](#) directive.

Allowed Values: Integers greater than 1 specify the number blocks to be created.
Less than or equal to 1 disables the directive.

Default Value: 0

Allowed Objects: Resource

Last State Allowed: compile

ITERATIONS

Specifies the number of times a loop iterates.

The default for this value is set during compilation. Future optimizations use this number to optimize the loop iterator and changing this number may cause simulation failures. Setting this number to 0 means that the number of iterations is unknown and the loop iterator will not be optimized.

Allowed Values: Integers greater than or equal to 0

Default Value: 0

Allowed Objects: Loop

Last State Allowed: architect

MAP_TO_MODULE

Maps a resource to the specified library component or other special resource types.

Typically this directive maps a resource to a library component. The alternative is specify one of three special purpose keywords, described in Table 7-3.

Table 7-3. Keyword Values for MAP_TO_MODULE Directive

Keyword	Description
[CCORE]	Used for the CCORE design flow. Refer to “ CCORE Design Flow ” on page 597.
[DirectInput]	Used for inputs that feed multiple pipeline stagesRefer to “ DirectInput Resource Type ” on page 270.
[Register]	Applies only array variables. It splits array into separate elements and maps each element to separate registers. Refer to “ Predictive Resource and Memory Mapping ” for more information.

In the resource constraints editor GUI, if the specified value is the default setting, Catapult performs computations based on the **MEM_MAP_THRESHOLD** and **REGISTER_THRESHOLD** directives to determine a default component.

Allowed Values: A keyword ([DirectInput], [Register] or [Component]) or a valid library component that is compatible with the target resource.

Default Value: { }

Allowed Objects: Solution -> Design -> Operation -> Resource

Last State Allowed: compile

The following example maps the library component “`mgc_ioport.mgc_out_fifo_wait`” to the resource “`/fir_filter/output_rsc`” and sets the FIFO size to 5.

```
directive set /fir_filter/output:rsc
    -MAP_TO_MODULE "mgc_ioport.mgc_out_fifo_wait fifo_sz=5"
```

MAXLEN

Specifies the maximum number of cycles allowed in a process.

The number of cycles in a process may depend on the number of iterations in a loop as defined in the **ITERATIONS** directive.

Allowed Values: Integers greater than or equal to 0

Default Value: 0

Allowed Objects: Solution -> Process

Last State Allowed: architect

MAX_LATENCY

Specifies the maximum latency (cycles) allowed for the design.

The scheduler will not increase the latency of the design if the current latency is equal to or greater than MAX_LATENCY. However, the scheduler will continue to optimize for area, even if the design latency is above MAX_LATENCY.

If MAX_LATENCY cannot be met by the scheduler, then Catapult will issue a warning message of the form:

```
Warning: The design "<design name>" cannot be scheduled within the design
goal of <num> cycles.
```

If the **DESIGN_GOAL** directive is set to “latency”, then the latency of the initial design created by Catapult will not be increased.

Valid settings for this directive are either undefined (-1) or a positive integer specifying the maximum number of cycles. If undefined, Catapult will consider the maximum latency to be infinite.

Allowed Values: -1 or a positive integer

Default Value: -1

Allowed Objects: Design -> Process

Last State Allowed: architect

For more information, refer to “[Scheduler Optimization](#)” on page 276. The MAX_LATENCY directive applies only when the new scheduler is enabled (see “[Use Old Scheduling and Allocation Algorithms](#)” on page 111). See also related directives [EFFORT_LEVEL](#) and [AREA_GOAL](#).

MEM_MAP_THRESHOLD

Any array that has a number of elements greater than or equal to this number is mapped to memory by default. For more information, refer to “[Predictive Resource and Memory Mapping](#)” on page 258.

Allowed Values: Positive integers

Default Value: 32

Allowed Objects: Solution -> Design -> Process

Last State Allowed: compile

MERGEABLE

Specifies whether or not loops are eligible to be merged.

The MERGEABLE directive is only one of several criteria that determine whether or not a loop is actually mergeable. For more information, see “[Loop Merging](#)” on page 234.

Allowed Values: “true”, “false”, { }
(empty string takes the default setting for hierarchy)

Default Value: “true”
 (“false” for loops in systemC that contain a wait statement or modular I/O function call)

Allowed Objects: Solution -> Design -> Process -> Loop, hierarchical, skips loops

Last State Allowed: compile

MIN_CSTEPS_FROM

This directive is replaced by the [CSTEPS_FROM](#) directive.

NO_X_ASSIGNMENTS

Specifies whether or not Catapult-generated HDL can assign ‘X’ values (“don’t care” values).

This project-level directive supports components that will not generate “X” assignments. When not enabled (default), Catapult may generate Mux code using a case statement, such as the following, in which X’s can occur in simulation:

```
CASE sel_alias IS
    WHEN '1' =>
        result := inputs_alias(31 DOWNTO 0);
    WHEN '0' =>
        result := inputs_alias(63 DOWNTO 32);
    WHEN others =>
        result := (others => 'X');
END CASE;
```

When enabled, Muxes are generated using loops and range slices and will not create X's in simulation. Mux code similar to the following will be generated:

```
or_inputs := (OTHERS=>'0');
FOR i IN 0 TO 1 LOOP
    IF (1 - CONV_INTEGER((sel))) = i THEN
        or_inputs((i+1)*32-1 DOWNTO i*32) := inputs_alias((i+1)*32-1 DOWNTO
i*32);
    END IF;
END LOOP;
result := (OTHERS=>'0');
FOR i IN 0 TO 1 LOOP
    result := result or (or_inputs((i+1)*32-1 DOWNTO i*32));
END LOOP;
```

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Solution -> Design

Last State Allowed: analyze

OLD_SCHED

Set the default scheduling algorithm.

The Catapult 2007a release introduced a new, combined scheduling and allocation algorithm that is replacing the old scheduling algorithm. This directive will be deprecated when the old scheduler is retired.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Solution

Last State Allowed: architect

When running the old scheduler, the following warning message displays in the transcript:

Warning: This scheduling mode will not be supported in future releases of Catapult. Please set directive "OLD_SCHED" to false.



Note

If you use a “directives.tcl” file generated by Catapult 2007a or 2007a_update1, the old scheduler will be enabled because that was the default behavior in those versions. You can change the file to use the new scheduler by changing the OLD_SCHED setting to false.

OPT_CONST_MULTS

Enables or disables the optimization of constant multipliers.

The default value is -1, which defers the setting to the value that is defined in the technology library. In FPGA technologies shipped with Catapult, the library setting is off (0). In sample ASIC technology libraries shipped with Catapult, the library setting is on (1).

Allowed Values: -1 : Use the default settings in the library
1 : On
0 : Off

Default Value: -1

Allowed Objects: Solution -> Design

Last State Allowed: analyze

OUTPUT_DELAY

This directive specifies the global and/or port-specific output delay values.

When OUTPUT_DELAY is set on the top design, the delay value is globally inherited by all output ports in the design. When the directive is set on an output port resource, it modifies the total delay for the resource as follows:

$$\text{Total delay} = \text{library delay (from component library)} + \text{inherited delay} + \text{port delay}.$$

To reduce the total delay for a port resource, specify a negative port delay value.

Allowed Values: Real numbers

Default Value: 0.0

Allowed Objects: DESIGN, RESOURCE

Last State Allowed: compile

See also “[INPUT_DELAY](#)” on page 324. For an overview of how I/O delays are defined and applied in Catapult, refer to “[Timing Constraints on Input/Output Ports](#)” on page 294.

OUTPUT_REGISTERS

Determines if registers are created on CCORE outputs.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Process

Last State Allowed: compile

PACKING_MODE

Specifies the default packing mode for packing when several arrays share the same RAM.

In “compact” mode the arrays are packed as closely together as possible. The “base aligned” mode puts arrays on two-bit boundaries. For more information, refer to “[Packing Mode Algorithms](#)” on page 253 and “[Architectural Constraints on Resources](#)” on page 114.

Allowed Values: “absolute”, “compact”, “sidebyside”, “base aligned”

Default Value: “absolute”

Allowed Objects: Solution -> Design -> Resource

Last State Allowed: compile

PIPELINE_INIT_INTERVAL

Specifies the initiation interval for pipelined loops.

Set this directive to a number between 1 and the latency of a loop in order to pipeline the loop. This directive can also be used to slow down loop processing by setting it to a value greater than the loop latency. A value of zero disables pipelining.

Loops nested inside of a pipelined loop are automatically pipelined also. If this directive is set on a loop inside of a pipelined loop, the nested directive will be ignored. When Catapult applies the pipelining constraint, it is first applied to the innermost loop that isn't unrolled and then propagated outward.

Allowed Values: Integers greater than or equal to 0.

Default Value: 0

Allowed Objects: Solution -> Design -> Process -> Loop, hierarchical, skips loops

Last State Allowed: compile

PIPELINE_RAMP_UP

If true, the outputs of the pipeline will be set to zero while the pipeline is filling.

If false, the outputs of the pipeline will not be controlled, resulting in smaller hardware. Be aware that when PIPELINE_RAMP_UP is set to false, the generated circuit will generate invalid values while the pipeline is ramping up. Your testbench must account for these invalid values.

A setting of “false” will invalidate the SCVerify flow. For example, if the output is mapped to a RAM, the contents of the RAM will be different in the C model and the post-synthesis model. The first entries of the RAM are written during the ramp-up period and the rest of the values of the RAM will be offset by the number of samples written during the startup phase.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Solution -> Design -> Process -> Loop, hierarchical, skips loops

Last State Allowed: compile

PRESERVE_STRUCTS

Controls whether or not structs that pass through ac_channels are split into separate FIFOs for each data element.

If the directive is set to false, separate FIFOs are used for each data element in a struct when it is written to an ac_channel. When set to true, the entire struct is kept intact and written to a single

FIFO. In many cases preserving structs can reduce complexity and area of the resulting RTL.
See also pragma “[“hls_preserve_struct”](#) on page 345.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Design

Last State Allowed: analyze

Note

-  1. The SCVerify flow does not allow structs with embedded arrays to be preserved.
 - 2. The SLEC flow does not support preserved structs and will issue an error message.
-

To see an example design that demonstrates this directive, refer to the “Preserving Structs in AC Channels” toolkit. From the GUI, select the **Help > Toolkits...** menu item to open the Toolkits window.

READY_FLAG

Enables the “ready” handshaking signal and allows you to rename it. The new name must be compatible with the target RTL.

If the READY_FLAG is set to an empty value (“ ”), the handshaking signal that indicates “ready” is omitted from the process.

Allowed Values: Any string

Default Value: Disabled. When enabled, default name is “done”.

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

REDUNDANT_MUX_OPT

Removes unnecessary muxes that were added to hold register inputs stable.

This optimization improves the quality of RTL that Catapult produces for designs using hierarchy or distributed pipelining associated with wait controllers. Smaller area is achieved by optimizing pipeline ramp-up logic. Also critical paths through wait controller elements may be shorter. The recommended flow for all customers is to enable this optimization for the final production run of Catapult.

The [EFFORT_LEVEL](#) directive also controls the activity of the REDUNDANT_MUX_OPT directive. EFFOR_LEVEL must be set to “high” in order for REDUNDANT_MUX_OPT to be

enabled. If EFFORT_LEVEL is “medium,” REDUNDANT_MUX_OPT will have no effect, regardless of it setting.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Design

Last State Allowed: compile

REG_MAX_FANOUT

Limits fanout of register outputs to the specified threshold value.

It is an integer value that reflects a fanout count rather than an actual fanout load. It does not take into account expected gate sizing or even internal fanout count for components such as multipliers, multiplexers etc. In other words, it assumes that each input bit of a component is already buffered so that the fanout contribution for it is just one. A REG_MAX_FANOUT setting of less than 2 will not trigger this optimization. For more information, refer to [“Register Fanout Optimization”](#) on page 279.

Allowed Values: Integers greater than or equal to 0

Default Value: 0

Allowed Objects: Design

Last State Allowed: schedule

REGISTER_IDLE_SIGNAL

Adds registers to idle signals created by the IDLE_SIGNAL directive.

Idle signals are used in designs that require low power consumption. See [“Idle Signal Insertion”](#) on page 592 for more information. See also the [IDLE_SIGNAL](#) directive.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Design -> Process

Last State Allowed: architect

REGISTER_INPUT

Insert input registers on a component (Catapult SL only). The input registers are created external to the component. For example:

```
directive set {/top/core/adder\.v1.adder()} -REGISTER_INPUT true
```

Registers are often required when using memories in a design. Adding the extra registers may help alleviate timing problems due to routing delays.

You can also place registers on inputs internal to a CCORE component with the [INPUT_REGISTERS](#) directive.

The REGISTER_INPUT directive is applied as a resource constraint.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Qualified Component

Last State Allowed: architect

REGISTER_NAME

Specifies a register and variables for sharing.

The arguments for the REGISTER_NAME directive include *variable pathname* and *group name* as follows:

```
directive set <variable pathname> REGISTER_NAME <group_name>
```

Where the *variable pathname* is the pathname to the variable, and the *group name* is the name of the shared register.

For example, you can share a register between two variables as follows:

```
directive set /top/top:core/core/for:c_addr.sval REGISTER_NAME for.reg  
directive set /top/top:core/core/for:i#1.lpi.dfm#1 REGISTER_NAME for.reg
```

Allowed Values: Pathname string

Default Value: None

Allowed Objects: Variables

Last State Allowed: schedule

REGISTER_THRESHOLD

Any array with a number of elements equal to or greater than this threshold value and which is also mapped to a register bank (not mapped to a memory component), will produce an error in

order to prevent long runtime. For more information, refer to “[Predictive Resource and Memory Mapping](#)” on page 258.

Allowed Values: Positive integers

Default Value: 256

Allowed Objects: Solution -> Design -> Process

Last State Allowed: compile

RESET_ASYNC_ACTIVE

Specifies the active level of the asynchronous reset in the design.

This directive is a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the RESET_ASYNC_ACTIVE directive is by setting the [CLOCKS](#) directive. Setting the RESET_ASYNC_ACTIVE directive independently is supported for backward compatibility but is not recommended.

Allowed Values: “high”, “low”

Default Value: “low”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

RESET_ASYNC_NAME

Specifies the name for the asynchronous reset signal.

Can be any string, but may be modified to be a legal HDL name during netlisting.

This directive became a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the RESET_ASYNC_NAME directive is by setting the [CLOCKS](#) directive. Setting the RESET_ASYNC_NAME directive independently is supported for backward compatibility but is not recommended.

Allowed Values: Any string.

Default Value: “rst_n”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

RESET_CLEAR_ALL_REGS

If true, all the registers in a design are to be reset.

If false, only registers that need to be reset for correct simulation results are reset. This value should be set to “true” to guarantee that the C++ and RTL simulations match.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: schedule

RESET_KIND

Specifies whether the reset is synchronous, asynchronous or both.

This directive is a subordinate property of the **CLOCKS** directive. The recommended method for setting the RESET_KIND directive is by setting the **CLOCKS** directive. Setting the RESET_KIND directive independently is supported for backward compatibility but is not recommended.

Allowed Values: “sync”, “async”, “both”

Default Value: “sync”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

RESET_SYNC_ACTIVE

Specifies the active level of the synchronous reset in the design.

This directive is a subordinate property of the **CLOCKS** directive. The recommended method for setting the RESET_SYNC_ACTIVE directive is by setting the **CLOCKS** directive. Setting the RESET_SYNC_ACTIVE directive independently is supported for backward compatibility but is not recommended.

Allowed Values: “high”, “low”

Default Value: “high”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

RESET_SYNC_NAME

Specifies the name for the synchronous reset signal.

Can be any string, but may be modified to be a legal HDL name during netlisting.

This directive is a subordinate property of the [CLOCKS](#) directive. The recommended method for setting the RESET_SYNC_NAME directive is by setting the [CLOCKS](#) directive. Setting the RESET_SYNC_NAME directive independently is supported for backward compatibility but is not recommended.

Allowed Values: Any string.

Default Value: “rst”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

RESOURCE_NAME

Specify a shared resource name and the operations that should share the resource. The arguments for the RESOURCE_NAME directive are operation path and resource name as follows:

```
directive set <operation path> RESOURCE_NAME <resource name>
```

Where the *operation path* is a list of operations and the *resource name* is the name of the shared component.

For example, you can share three multiplier operations on a resource as follows:

```
directive set /top/top:core/top:core:conc/else:mul#3 RESOURCE_NAME \
             else:mgc_sample-090nm_beh_dc.mgc_mul(32,0,32,0,32,4)
directive set /top/top:core/top:core:conc/else:mul#1 RESOURCE_NAME \
             else:mgc_sample-090nm_beh_dc.mgc_mul(32,0,32,0,32,4)
directive set /top/top:core/top:core:conc/else:mul RESOURCE_NAME \
             else:mgc_sample-090nm_beh_dc.mgc_mul(32,0,32,0,32,4)
```

Allowed Values: Valid component name string

Default Value: None

Allowed Objects: An operation in the design

Last State Allowed: dpfsm

SAFE_FSM

Enables/disables the support for safe FSM flows.

Catapult allows for a default state that is not reachable from other states during RTL simulation. The default state will unconditionally set the state to the first state in the FSM or the reset state, if such a state exists.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: schedule

SPECULATE

Enables/disables the speculative execution feature.

Speculative execution allows Catapult to pre-compute operations before conditions.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Solution -> Design -> Process -> Operation

Last State Allowed: architect

STAGE_IO_CNS

Controls the generation of soft constraints for I/O in pipelined designs.

If this directive is set to true, Catapult gives a high priority to scheduling inputs in the same stage and outputs in the same stage. If this directive is set to false, Catapult moves I/O into different pipeline stages to save area. Pipelined designs in which the inputs are in the same stage and outputs are in the same stage are easier to integrate, so this directive is set to true by default.

Allowed Values: “true”, “false”

Default Value: “true”

Allowed Objects: Solution -> Design

Last State Allowed: architect

STAGE_REPLICATION

Specifies the maximum number of times a port can be replicated.

I/O ports can be replicated to increase parallelism for pipelining. Set this directive on resources containing ports to specify the maximum number of times it can be replicated.

Allowed Values: 0 = No replication for I/O ports (default)
 1 = No replication
 2 = Ping-pong replication
 3 and greater = Round-robin

Default Value: 0

Allowed Objects: Resources containing I/O ports

Last State Allowed: architect

START_FLAG

Sets the name for the handshaking “start” signal.

If START_FLAG is set to an empty value ({ }), the handshaking signal that indicates “start” is not added to the process.

Allowed Values: Any string

Default Value: Disabled. When enabled, default name is “start”.

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

STREAM

Sets the number of dimensions for an array mapped to time.

If an I/O resource variable is a multi-dimensional array, and the port reads/writes streaming data, this option can be used to specify the number of array dimensions to that are mapped to “time.”

Allowed Values: Any integer.
 Zero means no streaming.
 Non-zero number specifies the number of dimensions.
 (A positive number uses the old streaming functionality. A negative number uses the new streaming functionality.)

Default Value: 0

Allowed Objects: Design -> Process -> Variable

Last State Allowed: compile

TECHLIBS

Specifies a list of component libraries used to generate the RTL.

These libraries includes all the basic components (adders, subtractors, etc.) and more complex components (RAMs, etc.).

Allowed Values: A list of files, including full paths or files in the library search path.
Refer to “[Component Libraries Options](#)” on page 180 for information about the library search path.

Default Value: { }

Allowed Objects: Solution

Last State Allowed: analyze

TRANSACTION_DONE_SIGNAL

Adds signals to processes that indicate completion of I/O transactions.

If this global setting is enabled then each process in a design will have one or more signals that go high for one cycle at the completion of an I/O transaction.

Allowed Values: “true”, “false”

Default Value: “false”

Allowed Objects: Solution -> Design -> Process

Last State Allowed: analyze

UNROLL

Specifies how many times to unroll a loop.

Setting this value “yes” or a number greater than the number of [ITERATIONS](#) in a loop will completely unroll the loop. Setting the value to “no” or to 0 will not unroll the loop.

Allowed Values: Integers greater than or equal to 0, “yes”, “no”

Default Value: “no”

Allowed Objects: Solution -> Design -> Process -> Loop, (skips loops)

Last State Allowed: compile

WORD_WIDTH

Specifies the word width of a resource variable.

For memory resources, use this directive to configure how the array variable is mapped in memory. This allows several array words to be read at the same time or for only parts of an

array word to be read. This value is set by during compilation. It is ignored for array channels that are streamed.

For I/O resources, use this directive to split the resource variable into multiple I/O ports. For more information, refer to “[Word Width Constraint](#)” on page 255 and “[Architectural Constraints on Resources](#)” on page 114.

Allowed Values: Integers greater than 0

Default Value: 0

Allowed Objects: Variable

Last State Allowed: compile

Pragmas

Pragmas are synthesis controls embedded in source code. They are higher priority than options specified in the GUI. Many pragmas are associated with directive, and these directives override the pragmas. All pragmas are declared in the source with the following format:

```
#pragma <pragma name> <pragma options>
```

All pragmas can have an optional “hls_” before the pragma declaration. The Pragma name is not case sensitive, but many of the options are. The following sections lists the available pragmas.

Table 7-4. Alphabetical Summary of Pragmas

Pragma	Description
hls_builtin	Reserved pragma used by the Catapult front-end to recognize bit-accurate class structures.
hls_design	This pragma is used to define the top-level function for synthesis.
hls_direct_input	Removes the register from any direct input signal.
hls_fixed	Sets the default fixed point type that will be used for all float and double types.
hls_map_to_operator	Directs Catapult to either use a specified library component instead of synthesizing the associated C++ function, or to synthesize the function as a CCORE.
hls_noglobals	Removes the start and done flags by default.
hls_pipeline_init_interval	Sets the default PIPELINE_INIT_INTERVAL directive for a loop.
hls_preserve_struct	Controls whether or not structs that pass through ac_channels are split into separate FIFOs for each data element.

Table 7-4. Alphabetical Summary of Pragmas

Pragma	Description
<code>hls_remove_out_reg</code>	Removes the register from any output signal.
<code>hls_resource</code>	Sets the default resource and variable mapping.
<code>hls_stream</code>	Sets the default streaming value to the specified for a list of variables.
<code>hls_unroll</code>	Sets the default UNROLL directive for a loop.

hls_builtin

Reserved pragma used by the Catapult front-end to recognize bit-accurate class structures.

This pragma is used for the Algorithmic C (AC), SystemC and mc_bitvector types. This pragma takes no arguments.

Example: `#pragma hls_builtin`



Note

Do not add or remove this pragma from your source code.

hls_design

This pragma is used to define the top-level function for synthesis.

This pragma must be set on a function, but it may not be set on the function call or on a template. The design pragma has one required argument “top” to define the top level of hierarchy.

Example:

```
#pragma hls_design top
void my_function (...) {...}
```

hls_direct_input

Scope: SystemC

Removes the registers normally added during scheduling from an input to a module/process or a signal feeding a process.

Usage

```
#pragma direct_input
sc_in<int>input1, input2
```

Description

This pragma removes any pipeline registers from an input signal to a process. By default, Catapult schedules all I/O, which typically results in one or more pipeline registers.

This pragma can be declared on the input ports of an SC_MODULE and locally on internal signals.

Note



You cannot use this pragma on a signal that drives an SC_MODULE.

You can also use this pragma on signals with complex data types such as structs, including structs containing arrays. When this pragma is used for structs with arrays, it applies to all data members.

Example

The following example removes the input register from the din signal:

```
#include <ac_int.h>
#include <systemc.h>

#pragma hls_design top
SC_MODULE(example_reg_removal) {
public:
    // Make sure clock name matches directive in Catapult
    sc_in <bool> clk;
    // Make sure reset name matches directive in Catapult
    sc_in <bool> rst;

    #pragma direct_input
    sc_in<bool> din;

    SC_CTOR(example_reg_removal) :
        clk("clk"),
        rst("rst"),
        din("din"),
        dout("dout")
    {
        SC_METHOD(process1);
        sensitive << clk.pos();
    }

    void process1() {
        dout = din;
    }
};
```

For more information about specifying the top-level function in your design, refer to “[Preparing the Source Files](#)” on page 95.

hls_fixed

Sets the default fixed point type that will be used for all float and double types.

If more than one fixed pragma is declared, then only one will be used (i.e. you cannot have different options for different parts of your C++ code). The fixed pragma takes the same arguments as the sc_fixed datatype from SystemC. We strongly recommend using the sc_fixed datatypes instead of this pragma because there may be simulation differences when the floating point C++ code is converted to fixed point.

For more information, refer to the section “[SystemC Data Types](#)” in the *Catapult C Synthesis C++ to Hardware Concepts*. The default if this pragma is not defined is:

`32,16,SC_TRN,SC_WRAP,0`

Example:

```
#pragma hls_fixed 32 16 SC_TRN SC_WRAP 0
```

hls_map_to_operator

Directs Catapult to either use a specified library component instead of synthesizing the associated C++ function, or to synthesize the function as a CCORE.

This pragma statement must immediately precede the function definition in the source code. If the function is templatized, the pragma must also precede the template statement. The syntax for the pragma is:

```
#pragma map_to_operator <string>
```

where <string> can be one of the following:

[CCORE]

Use the top-down CCORE method to synthesize the function. Refer to “[CCORE Design Flow](#)” on page 597 for more information about CCOREs.

Library component name

If a Catapult library component of the specified name is available in the Catapult project, that component will be used in the design instead of synthesizing the function. The component interface must match the function interface.

If no library component of that name is found, an error will occur during the “architect” stage of the Catapult work flow.

Example 1: This example uses the library component “my_oper” in place of the function “my_function.”

```
#pragma hls_map_to_operator my_oper
ac_int<8> my_function (...) {...}
```

The file system location of the library containing the “my_oper” component must be included in Component Libraries search path. Refer to “[Component Libraries Options](#)” on page 180. Also,

the library must be accessible in the Catapult project by including it in the [TECHLIBS](#) directive setting. From the GUI, use the “Compatible Libraries” field of the Setup Design constraints editor to select the library.

Example 2: This example uses the top-down CCORE flow to synthesize “my_function:”

```
#pragma hls_map_to_operator [Component]
ac_int<8> my_function (...) {...}
```

[hls_noglobals](#)

Removes the start and done flags by default.

This pragma takes no arguments. Example:

```
#pragma hls_noglobals
```

[hls_pipeline_init_interval](#)

Sets the default [PIPELINE_INIT_INTERVAL](#) directive for a loop.

Legal values are the same as for the directive (greater than zero)

Example:

```
#pragma hls_pipeline_init_interval 5
my_loop: for ( int i = 0; i < 10; i++ ) {...}
```

[hls_preserve_struct](#)

Controls whether or not structs that pass through ac_channels are split into separate FIFOs for each data element.

If the pragma is set to false, separate FIFOs are used for each data element in a struct when it is written to an ac_channel. When set to true, the entire struct is kept intact and written to a single FIFO. In many cases preserving structs can reduce complexity and area of the resulting RTL.

This pragma must be declared in the global scope of the C++ design code.

```
#pragma hls_preserve_struct true
```

See also “[PRESERVE_STRUCTS](#)” on page 331.



Note

1. The SCVerify flow does not allow structs with embedded arrays to be preserved.
2. The SLEC flow does not support preserved structs and will issue an error message.

[hls_remove_out_reg](#)

Scope: SystemC

Removes the register on the output of a process or SC_MODULE.

Usage

```
#pragma remove_out_reg
sc_signal<int>my_sig1, my_sig2
```

Description

This pragma can be used to remove registers from an SC_MODULE or output signals inside the processes: SC_CTHREAD, SC_THREAD, and SC_METHOD. It can also be used on signals connecting two of these processes. By default, Catapult places registers on all output signals.

This pragma can be declared locally on the output port of an SC_MODULE to remove the register on the driving process. You cannot use this pragma on a signal driven by an SC_MODULE.

You can also use this pragma on signals declared with complex data types such as structs, including those containing arrays. When this pragma is used for signals of type struct, it applies to all data members.

Example

The following example removes the output register from the *dout* signal:

```
#include <ac_int.h>
#include <systemc.h>

#pragma hls_design top
SC_MODULE(example_reg_removal) {
public:
    // Make sure clock name matches directive in Catapult
    sc_in <bool> clk;
    // Make sure reset name matches directive in Catapult
    sc_in <bool> rst;

#pragma remove_out_reg
    sc_out<bool> dout;

    SC_CTOR(example_reg_removal) :
        clk("clk"),
        rst("rst"),
        din("din"),
        dout("dout")
    {
        SC_METHOD(process1);
        sensitive << clk.pos();
    }

    void process1() {
        dout = din;
    }
};
```

hls_resource

Sets the default resource and variable mapping.

This pragma must be declared in the function scope even if the variable being mapped is declared on the function interface. The arguments to the resource pragma are case sensitive and must exactly match the C++ variable declaration and the library name.

The arguments for the resource pragma are:

```
#pragma hls_resource <resource name> variables=<comma separated variable list> map_to_module="<component name> <component options>"
```

The following example maps the variables “out1” and “out2” to the resource “output_resource”. It also sets the default resource type to “mhc_out_fifo_wait” and sets the FIFO size to 10.

```
void func ( int8 *in, int8 *out1, int8 *out2) {
    // DO NOT declare pragma before a static declaration
    static int static_variable;
    #pragma hls_resource output_resource variables="out1, out2" \
        map_to_module="mhc_out_fifo_wait fifo_sz=10"
    int non_static_variable;
    ...
}
```

hls_stream

Sets the default streaming value to the specified for a list of variables.

This pragma must be declared in the function scope, even though the variable being streamed is declared on the function interface. The “depth” argument specifies the number of array dimensions to be streamed.

The arguments for the stream pragma are:

```
#pragma hls_stream variables=<var_list> depth=<num_dimensions>
```

Example:

```
void func ( int8 in1[10], int8 in2[10], int8 in3[10][3], int16* out) {
    // DO NOT declare pragma before a static declaration
    static int static_variable;
    #pragma hls_stream variables="in1,in2" depth=1
    #pragma hls_stream variables="in3" depth=2
    int non_static_variable;
    ...
}
```

hls_unroll

Sets the default **UNROLL** directive for a loop.

Legal values for this pragma are the same as for the directive (Yes, No, or a positive number).

Example:

```
#pragma hls_unroll yes
my_loop: for ( int i = 0; i < 10; i++ ) { ... }
```

Chapter 8

Commands

Catapult provides a shell command line interface that allows you to enter commands interactively and use scripts for batch processing. For more information, see the following topics:

- “[General Command Syntax](#)” on page 349
- “[Using Tcl Commands in Scripts](#)” on page 358
- “[Command Reference](#)” on page 360 page 360

General Command Syntax

The command interface is based on the Tcl language and accepts all standard Tcl commands. Standard Tcl provides the foundation for the command syntax, including variable assignment, handling of lists and arrays, sorting, string manipulation, arithmetic operations, (if/case/foreach/while) statements, and procedures. [Table 8-1](#) describes the basic Tcl syntax rules.

Table 8-1. Basic Tcl Syntax

Syntax	Description
command arg1 ... argN	A command string consists of a command name followed by zero or more arguments. White space delimits each element.
\$my_variable	The dollar sign (\$) substitutes the value of a variable. In this example, the variable name is “my_variable”.
[solution get]	Square brackets are used to execute a nested command. For example, if you want to pass the result of one command as the argument to another, use this syntax. In this example, the nested command is “ solution get ”, which is used to obtain information about a solution.
"some stuff"	Double quotation marks group words as a single argument to a command. Dollar signs and square brackets are interpreted inside double quotation marks.
{some stuff}	Curly braces also group words into a single argument, but elements within the braces are not interpreted.

Table 8-1. Basic Tcl Syntax (cont.)

Syntax	Description
\	The backslash (\) is used to quote special characters. For example, \n generates a newline. The backslash also is used to “turn off” the special meanings of the dollar sign, quotation marks, square brackets, and curly braces.

Note

 Reference help files about the Tcl language are available in the Catapult software tree:

UNIX man pages: \$MGC_HOME/pkgs/tcl_msg/man

Windows help files: \$MGC_HOME/pkgs/tcl_msg/doc

Catapult commands typically have the form “<object> <operator>”, where <object> is the command name and <operator> is the action to be performed. The operator is first argument. Some examples are:

Table 8-2. Typical Command Form Examples

Object	Operators
component	add, remove
directive	get, remove, set
project	get, load, new, report, save, set, close

Some commands can operate on a set of objects. In these cases, the initial argument(s) specify the target object. For example:

Table 8-3. Other Command Form Examples

Object	Operators
flow	get, run
flow package	provide, require, present, forget, names, versions, script, vcompare, versions, vsatisfies
flow package option	add, get, remove, set

Documentation Conventions for Catapult Commands

For clarity when referring to Catapult commands by name, the documentation uses a composite name consisting of the object and operator components of the command string. For example the documentation might refer to the “[project new](#)” command or the “[flow package option add](#)” command. Similarly, the command reference pages use the composite form of the command names.

Command Reference Page Format

All command reference pages begin on a new page and use the same structure to present the information. Each command page contains the six sections described in the following list.

- At the top of the page is the **command name** and a **short description** of what the command does.
- The “**Syntax**” section shows the proper usage of all of the command arguments and switches. For example:

```

solution get ?<path>? ?<switches>? ?<args>?
    Get solution information
<path>                  Hierarchical database path (Optional)
<switches>               Valid switches: (Optional)
    --                      End <switches> parsing
    -recurse <string>     Everything under
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                           Return data format
        value              just matching values
        path               just matching paths
        pathvalue          path and value combination for array set
        leaf                just matching leaves
        leafvalue           leaf and value combination for array set
        advanced            hierarchical list structure
        none                no return value
    -checkpath <bool>      Error on path not found
    -match <exact|glob>    Path match type
        exact              exact paths only
        glob               glob paths
    -info <bool>           Return object info
    -solution <string>     Specify solution
    -state <new|analyze|compile|architect|allocate|
                           schedule|dpfsm|extract|last>
                           Pick design state, default is current
        initial            Initial state of new project
        new                Design state 'new'
        analyze            Design state 'analyze'
        compile            Design state 'compile'
        architect          Design state 'architect'
        allocate            Design state 'allocate'
        schedule            Design state 'schedule'
        dpfsm              Design state 'dpfsm'
        extract             Design state 'extract'
        last               Last valid design state for specified
                           solution
<args>                  Database subpath and value combinations

```

Table 8-4 describes the meaning of the special characters used to express command line syntax.

Table 8-4. Documentation Conventions for Command Syntax

Symbol	Description
< >	Fields to be completed with your values.

Table 8-4. Documentation Conventions for Command Syntax

Symbol	Description
?< >?	Optional argument.
	The “or” symbol. Indicates mutually exclusive arguments.

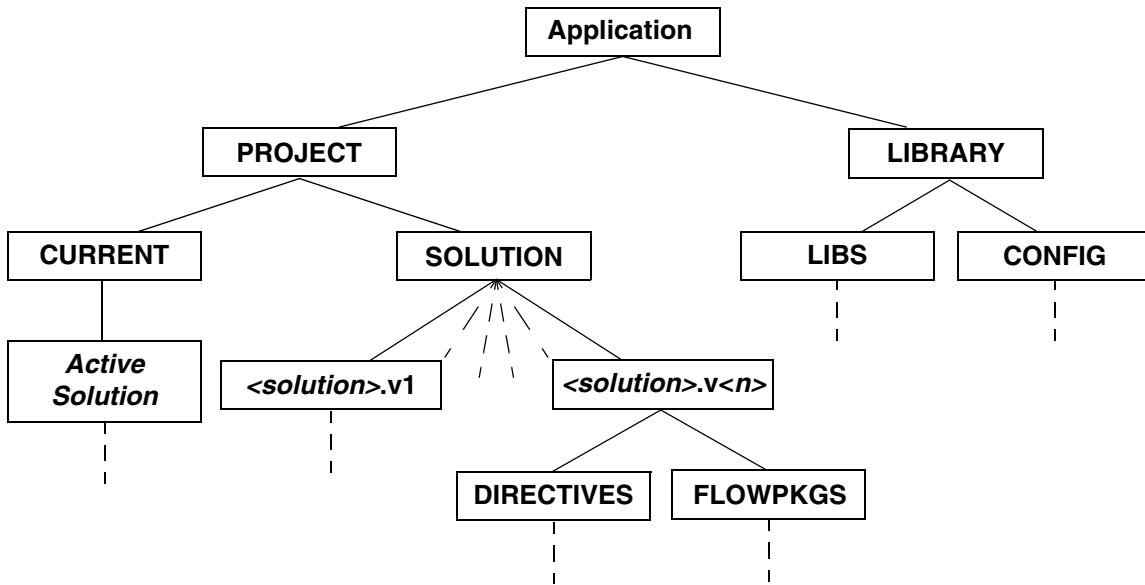
- The “**Arguments**” section provides a detailed description of each command argument. Note that there is a group of command switches common to almost all Catapult commands. Those switches are documented separately in the section “[Common Command Switches](#)” on page 356, rather than repeating the information throughout the command reference pages. A link to that section is provided from each command reference page that uses those switches.
- The “**Description**” section describes the purpose and usage of the command.
- The “**Examples**” section provides one or more examples of how the command can be used, and includes a description.
- The “**Related Commands**” section provides links to other command reference pages related to the current command.

Command Interface to the SIF Database

Every Catapult project has a SIF (synthesis internal format) database that stores the state of the project and its solutions. The database also includes information about the Catapult session and loaded libraries. Data values are stored as *key* and *value* pairs, and the keys are organized hierarchically. Data values are accessed by specifying the database path to their corresponding keys. In general, key names and hierarchical node names are in all uppercase letters. One notable exception is the key name “*name*” that appears in many places throughout the database.

The diagram in Figure 8-1 is a simplified representation of the SIF database hierarchy. It shows only the primary nodes of interest with respect to the set of Catapult commands that interface with the database. Those commands are listed in [Table 8-5](#) on page 353.

Figure 8-1. Hierarchy of Objects in the SIF Database



The SIF database structure is not static. When a new project is created, the database is populated with the minimum set of key/value pairs (default settings) required by the system. Catapult dynamically creates, removes and modifies key/value pairs throughout the session.

Path and Sub-Path Argument Syntax Rules

The Catapult commands listed in Table 8-5 interact with the SIF database. These commands accept database path and sub-path arguments in order to access specific keys in the database. Many of the switches described in the section “[Common Command Switches](#)” on page 356 can be used to control and refine how the path and sub-path arguments are evaluated by Catapult.

Table 8-5. Commands That Take Database Path Arguments

Command Name	Operators	Path Type	Path Root
application	get	SIF	/ (Root of SIF)
directive	get, remove, set	HDL-style	Design top
flow	get	SIF	/FLOWPKGS
library	get	SIF	/LIBRARY
project	get	SIF	/PROJECT
resource	add, remove	HDL-style	Design top
solution	get	SIF	/PROJECT/CURRENT

As indicated in the “Path Type” column, a few of the commands take path arguments in the form of HDL design paths. HDL paths are the same type of paths that you would see in a simulation tool like ModelSim. For example:

```
directive set /dct/core/main/mult1 -MERGEABLE false
```

For commands that take a SIF database path, the root of the path argument is the node in the SIF hierarchy corresponding to the command name. In other words, the path argument for the “project” command is rooted at the “PROJECT” node in the hierarchy. Similarly, the path for the “solution” command is rooted at the “PROJECT/CURRENT” node, which contains the active solution. Refer to the “Path Root” column of the table.

The following examples illustrate how the “project” command and the “solution” command require slightly different paths in order to access the same data value.

```
project get /CURRENT/INPUTFILES/1/name
# $PROJECT_HOME/dct.cpp

solution get /INPUTFILES/1/name
# $PROJECT_HOME/dct.cpp
```

Note

 The set of key/value pairs in the database changes dynamically throughout the session as you work on the design. Use a “get” command to check for the existence of a key/value before attempting to modify or remove it.

Using Wildcard Characters in Path Arguments

Wildcard characters can be used in Path and sub-path arguments. Wildcard expansion is enabled by the “-match glob” switch (see also “[-match](#)” on page 357). The following wildcard reserved characters are supported:

- ‘*’ : Asterisk matches one and only one level.

Example:

```
solution get /INPUTFILES/*/name -match glob
#
# {$PROJECT_HOME/dct.cpp} {$PROJECT_HOME/tb_dct.cpp}
```

- “...” : Ellipsis matches all sub-levels.

Example:

```
solution get /TOP/INTERFACE/.../name -match glob -return pathvalue
```

In this example, the ellipsis wildcard expanded the search to include the entire sub-tree below the “/TOP/INTERFACE” node. The return values were found at the following paths:

```
/TOP/INTERFACE/0/name clk
```

```
/TOP/INTERFACE/0/DATATYPE/name std_logic
/TOP/INTERFACE/1/name rst
/TOP/INTERFACE/1/DATATYPE/name std_logic
/TOP/INTERFACE/2/name input_rsc_dualport_data_in
/TOP/INTERFACE/2/DATATYPE/name unsigned_17_0
/TOP/INTERFACE/3/name input_rsc_dualport_addr
/TOP/INTERFACE/3/DATATYPE/name unsigned_11_0
/TOP/INTERFACE/4/name input_rsc_dualport_we
/TOP/INTERFACE/4/DATATYPE/name unsigned_1_0
/TOP/INTERFACE/5/name input_rsc_dualport_data_out
/TOP/INTERFACE/5/DATATYPE/name unsigned_17_0
/TOP/INTERFACE/6/name dct_rsc_dualport_data_in
/TOP/INTERFACE/6/DATATYPE/name unsigned_21_0
/TOP/INTERFACE/7/name dct_rsc_dualport_addr
/TOP/INTERFACE/7/DATATYPE/name unsigned_11_0
/TOP/INTERFACE/8/name dct_rsc_dualport_we
/TOP/INTERFACE/8/DATATYPE/name unsigned_1_0
/TOP/INTERFACE/9/name dct_rsc_dualport_data_out
/TOP/INTERFACE/9/DATATYPE/name unsigned_21_0
```

For each interface node (0, 1, ... 9) two “name” keys were returned. The first is the interface name, and one level deeper is the DATATYPE name of the interface.

The ellipsis is not valid at the root level. For example “`../INTERFACE`” is invalid.

Using Sub-Path Arguments

Commands that take database path arguments also accept optional sub-path arguments. Sub-paths make it possible to specify multiple variations of the path argument in a single command. The general form of the command line is shown below.

```
command_name <sif_path> ?<switches>? <sub-path_1> ... <sub-path_n>
```

The `<sif_path>` argument is a partial database path that is common to all of the target objects. A set of unique paths are formed by appending each sub-path to the partial path. In the following example the sub-paths “name” and “VERSION” are appended to the partial path “/FLOWPKGS/ModelSim” to form two complete paths. The “`-return pathvalue`” switch is used in this example so that the resulting paths are displayed in the return value.

```
solution get /FLOWPKGS/ModelSim -return pathvalue name VERSION
#
# /FLOWPKGS/ModelSim/name ModelSim /FLOWPKGS/ModelSim/VERSION 6.0
```

Error Messages Caused by Invalid Paths

If an invalid path or sub-path is specified, the command will not be executed and an error message will be displayed for each invalid path. For example, the following command tries to get the names of two input files, but only one exists. The command aborts and returns a path error.

```
solution get /INPUTFILES -return value 1/name 2/name
# Error: solution get: Unknown path '/INPUTFILES/2'
```

The error can be suppressed by including the “`-checkpath false`” switch in the command line. In this case, the valid file name is returned.

```
solution get /INPUTFILES -return value -checkpath false 1/name 2/name
# {$PROJECT_HOME/dct.cpp}
```

Script writers can use the tcl `catch` command to catch the return value and then parse the results to determine which parts of the command passed. For example:

```
if ([catch {solution get /INPUTFILES -return value 1/name 2/name} msg]) {
    // parse $msg for errors
    ...
}
```

Common Command Switches

The command switches described in this section are common to most of the Catapult commands. The switches qualify how the database is searched and how the returned data is displayed. The same command switch may be set several times on the same line. If the same switch is used, the last switch on the line will take precedence.

The following switches are described in this section:

```
-return
-checkpath
-match
-info
-recurse
-- (switch)
-help
--help
```

-return

The `-return` switch filters and formats the data returned by the command.

```
-return <value|path|pathvalue|leaf|leafvalue|advanced|none>
```

Arguments:

- **value**

Return only the data values stored at the database paths that match the search paths. For example:

```
solution get /INPUTFILES/*/name -match glob -return value
# {$PROJECT_HOME/tb_dct.cpp} {$PROJECT_HOME/dct.cpp}
```

- **path**

Return only the database paths that match the search paths. For example:

```
solution get /INPUTFILES/*/name -match glob -return path
```

```
# /INPUTFILES/1/name /INPUTFILES/2/name
```

- **pathvalue**

Return the database paths and the data values. For example:

```
solution get /INPUTFILES/*/name -match glob -return pathvalue
# /INPUTFILES/1/name {$PROJECT_HOME/tb_dct.cpp} /INPUTFILES/2/name
{$PROJECT_HOME/dct.cpp}
```

- **leaf**

Return only the leaf names of the database paths that match the search paths. For example:

```
solution get /INPUTFILES/*/name -match glob -return leaf
# name name
```

- **leafvalue**

Return the leaf names and the data values. For example:

```
solution get /INPUTFILES/*/name -match glob -return leafvalue
# name {$PROJECT_HOME/tb_dct.cpp} name {$PROJECT_HOME/dct.cpp}
```

- **advanced**

Return a hierarchical representation of the database paths and the data values. Each level of hierarchy is enclosed in braces, and sub-levels are nested. For example:

```
solution get /INPUTFILES/*/name -match glob -return advanced
# INPUTFILES {1 {name {$PROJECT_HOME/tb_dct.cpp}} 2 {name
{$PROJECT_HOME/dct.cpp}}}
```

- **none**

Return nothing. Use this argument to suppress transcripting of the return value.

-checkpath

The “-checkpath” switch enables/disables error checking of paths.

```
-checkpath <true|false>
```

If the “-checkpath” switch is set to true, then the command will issue an error if any of the paths in the command line, including paths generated by globbing, do not match an existing path.

-match

The “-match” switch can use *glob* or *exact* matching rules.

```
-match <glob|exact>
```

The following TCL rules are used with the “-match” switch. If the “-match” switch is set to glob, then the TCL glob rules are used to resolve the path argument. If set to exact, then the TCL exact rules are used to resolve the path argument. Refer to “[Using Wildcard Characters in Path Arguments](#)” on page 354 for more information about the glob option.

-info

If the “-info” switch is true, then the type information on each non-value node will be included in the return value if the return value includes the specified value.

```
-info <true|false>
```

-recurse

If the “-recurse” switch is true, then Catapult will perform a recursive path search.

```
-recurse <true|false|+<depth>>
```

Valid values for enabling full recursion are case insensitive and include: ‘1’, “yes”, or “true” (‘y’ and ‘t’ are sufficient). To disable it, use any other string or no string at all.

You can also limit the depth of recursion by specifying the number of levels to search. The argument format is “+<n>”, where the plus character is required and <n> is any whole number.

```
-recurse +3
```

Note that “+1” and “1” do not mean the same thing. Without the plus character, “1” evaluates to “true.”

-- (switch)

The “--” switch disables switch parsing for the rest of the line. You can set this switch so that you can access data objects that have names that conflict with reserved words (i.e. switches) and reserved characters.

-help

The “-help” switch displays help information about the commands.

--help

The “--help” switch displays recursive help for hierarchical commands.

Using Tcl Commands in Scripts

A good method for creating a basic Tcl command script is to make a copy of the “directives.tcl” file that is automatically generated by Catapult for each solution, and modify it as needed to suit your purposes. For more information about the directives.tcl file, refer to the “[Migrating Design Solutions to Newer Catapult Versions](#)” on page 44.

After you create a Tcl script, you can use one of the following methods to *source* your Tcl script from within a Catapult session:

- Interactive Command Line Shell
- GUI
- Command Line Invocation Argument
- Tcl Startup Script

Interactive Command Line Shell

Type the following syntax to *source* your Tcl script:

```
source <my_tcl_script>
```

or type the following command to execute your Tcl script:

```
dofile {C:/6tap filter/scripts/run1.tcl}
```

Note

 Microsoft Windows operating systems allow pathnames to contain spaces. Therefore it is a good practice to enclose pathnames in curly braces ({ }) on Windows.

The **dofile** command is similar to the **source** command in that they both execute the Tcl commands contained in a specified file. An additional feature of the dofile command is that it also sends a message to the standard output device each time a Tcl command is executed. This is a useful feature that you can use to help debug the Tcl script.

All UNIX/Linux commands and many DOS commands are accessible from the Catapult command line shell as long as the command can be found in your search path (PATH environment variable). Another helpful Tcl feature is history tracking. Type the command “history” to view your previous commands. Any previous command can be re-executed typing “!*<history_number>*” or “!*<beginning_of_cmd>*”. You simply type “!!” to re-execute the last command.

GUI

On the menu bar select **File > Run Script**. Use the file navigator to locate, select and execute your script. Your script file runs in the Catapult Command/Transcript window.

Command Line Invocation Argument

When invoking Catapult from a shell window, use the “-file” argument to source your script:

```
% <catapult_dir>/Mgc_home/bin/catapult -file <my_tcl_script>
```

Tcl Startup Script

Catapult startup scripts contain Tcl commands to be executed immediately after the tool is up and running. When creating scripts, remember that some commands cannot be executed until Catapult has reached a certain stage in the work flow. For example, commands such as “**flow package require /SCVerify**” will cause errors if it is called before a design has been loaded.

Catapult searches for two different startup scripts in the following order. Both scripts are run if they both exist.

1. **\$MGC_HOME/shared/etc/appl/sif/startup.tcl**

This script is specific to the MGC_HOME tree in which it resides. When you invoke Catapult from that tree, the script is run. The default file system permission of the \$MGC_HOME/shared/etc/appl/sif directory might be read-only. Change the permissions as needed to create, modify or delete the startup file.

2. **\$HOME/catapult.tcl** (Linux)

%USERPROFILE%/.catapult.tcl (Windows)

This script is specific to single user account and only runs when Catapult is invoked from the same user account. It runs after the startup script in 1 above. To make the script hidden in the file system, prefix a period on the filename (.catapult.tcl).

Command Reference

This section begins with the “[Command Summary](#)” table which contains an alphabetical listing and short description of Catapult commands. Use the command summary table as an index for locating and jumping to command reference pages. A second table shows the mapping of commands to flow icons in the GUI. The actual command reference pages, also presented in alphabetical order, begin after the tables.

Command Summary

Clicking on any item in Table 8-6 will bring you to the command reference page for that item.

Table 8-6. Alphabetical Command Summary

Command	Description
application	General command to access all data in the application and to exit the application.
application get	Gets information from the Catapult project, solution, or library databases and gets Catapult system information.
application exit	Closes the Catapult session and returns an exit code.

Table 8-6. Alphabetical Command Summary (cont.)

Command	Description
application report	Generates a report of the Catapult licenses in use in the current session.
catapult	Shell-level command that invokes Catapult.
cycle	Used to organize I/O timing, but it may also be used to resolve timing or pipelining issues. The cycle command lets you perform a variety of actions relating to cycle constraints.
cycle add	Adds cycle constraints on operations to control scheduling.
cycle find_op	Returns full path of operations in the cycle database.
cycle get	Gets the cycle constraint settings on the specified operation.
cycle remove	Removes cycle constraint(s) set on the specified operations.
cycle set	Sets the value of the specified cycle constraint(s).
directive	In Catapult, directives are used as a control mechanism to guide the synthesis process. Refer to Chapter 7, “Directives,” for complete information about Catapult directives.
directive get	Gets the value of the specified directives.
directive remove	Resets the specified directives to their default values.
directive set	Sets the specified directive to the specified value.
dofile	Executes the specified Tcl script and prints messages as each command in the script is executed.
flow	The flow commands are used to load and manage user-defined flows. For conceptual information about creating and using custom flows, refer to “Flow Customization” on page 609.
flow get	Queries flow database information.
flow package forget	Removes the specified package from the active solution.
flow package names	Returns the names of all flow packages that are available to Catapult.
flow package option add	Creates a new option for a flow package.
flow package option get	Returns the value of the specified flow package option.
flow package option remove	Deletes the specified flow package option.
flow package option set	Sets the value of the specified option.
flow package present	Returns the version string of the specified package if the package has been loaded.
flow package provide	Registers a flow package name and compatible version(s).

Table 8-6. Alphabetical Command Summary (cont.)

Command	Description
flow package require	Loads the specified flow package into the current Solution.
flow package script	Returns the file system path to the flow package file.
flow package vcompare	Compares two version numbers and determines which one is newer.
flow package versions	Returns a list of all available versions of the specified flow package.
flow package vsatisfies	Compares two version strings and determines whether they are compatible.
flow provide	Adds a flow to a package index.
flow run	Executes a flow procedure.
go	Starts the synthesis flow and moves to the specified state.
help command	Gets help on command syntax and usage.
help message	Gets help on system message codes.
ignore_memory_precedences	Ignores data flow dependencies between the specified read/write operations of a memory array.
logfile close	Closes the Catapult session log file.
logfile isopen	Reports whether or not a Catapult session log file is open.
logfile message	Sends messages to the session log file and the Transcript window in the Catapult GUI.
logfile move	Relocates and/or renames the session log file.
logfile name	Returns the full pathname of the current session log file.
logfile open	Opens a Catapult session log file at the specified path.
logfile save_commands	Saves all session commands to a file.
options	Performs tasks associated with Catapult options that are stored in files or within the in-memory database.
options defaults	Resets all flow options to default settings.
options exists	Checks whether an option exists in the database.
options get	Returns the value of the specified option.
options load	Loads saved option settings.
options save	Save the in-memory options to the Catapult registry or an alternate file.
options set	Sets the value of an option in the project database.

Table 8-6. Alphabetical Command Summary (cont.)

Command	Description
project	Performs tasks relating to Catapult projects.
project close	Closes the current project without exiting the Catapult session.
project get	Gets information from the Catapult project database.
project load	Loads a previously saved project.
project new	Creates a new project.
project report	Generates a report on the current project.
project save	Saves the current project to a file.
project set	Sets the project name displayed in the GUI and/or create a new project directory.
quit	Terminate operations and close the Catapult session.
resource	Resource objects such as inputs, outputs and registers are tracked in Catapult and can have constraints applied to them.
resource add	Adds resource objects to a process in the design.
resource remove	Removes resource objects from the design.
set_working_dir	Sets the working directory to the specified pathname.
solution	Performs various tasks on the active solution.
solution get	Gets information about the files, directories, design data and general information associated with a solution.
solution file add	Add an input or output file to the solution.
solution file remove	Remove a file from the Input File list.
solution file restore	Restores input files from the Version Control System.
solution file set	Set options on an input or output file.
solution new	Create a new solution branch from the specified solution.
solution netlist	Generate an RTL netlist and tool flow scripts for the current solution (not available in the Catapult BL product).
solution remove	Removes a solution from the project.
solution rename	Renames a solution.
solution report	Generates a report on the active solution.
solution restore	Restores input files from the Version Control system.
solution select	Makes the specified solution the active solution.

Table 8-6. Alphabetical Command Summary (cont.)

Command	Description
solution timing	Generate timing reports for critical paths and user-specified data paths in the design.
view schedule	Opens the Gantt chart schedule of operations for viewing.
view schematic	Opens the RTL schematic window. Displays critical paths and path timing data.
view file	Opens the specified file in the DesignPad Editor window.
view source	Opens the C++ source file(s) in the Input Files list.

Commands Mapped to Flow Tasks

Table 8-7 describes the flow tasks in Task Bar window.

Table 8-7. Commands Mapped to Flow Tasks

Flow Task	Function	Implicit Commands
Set Working Directory	Opens a file system browser with which you find and select your working directory.	<code>set_working_dir</code>
Add Input Files	Opens a file system browser with which you find and add input files to the project.	<code>solution_file add</code>
Setup Design	Displays the Setup Design settings in the Constraint Editor window. This task allows you to set global constraints related to the target technology, design interface and design hierarchy.	<code>go analyze</code>
Architectural Constraints	Opens the Constraint Editor with which you select design-specific architectural options, like loops and memories.	<code>go compile</code> (Compilation, conversion to synthesis internal format, and optimization)
Schedule	Opens the Schedule window (Gantt chart) with which you analyze and set scheduling constraints.	<code>go allocate</code> (Resource scheduling)
Generate RTL	Generates RTL (for synthesis and simulation), timing reports and simulation scripts.	<code>go extract</code> (Output file generation)

After running **Generate RTL** (or “[go extract](#)” command), you can synthesize or simulate the design by double-clicking the appropriate file in the Output Files folder (Project Files window). Alternatively, you can run the “[flow run](#)” command. For example:

```
flow run /ModelSim/launch ./cycle.vhdl.msim
```

and

```
flow run /Precision/precision -file ./rtl.vhdl.psr
```

application

General command to access all data in the application and to exit the application.

The following *application* commands are available.

- ***application get***

Retrieves data in the application including the solution, library, project, and system databases.

- ***application exit***

Exits Catapult with optional return code.

- ***application report***

Reports the Catapult licenses used in the current session.

application get

Gets information from the Catapult project, solution, or library databases and gets Catapult system information.

Syntax

```
application get ?<path>? ?<switches>? ?<args>?

<path>          Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --
        End <switches> parsing
    -recurse <string>   Everything under
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
        Return data format
            value           just matching values
            path            just matching paths
            pathvalue       path and value combination for array set
            leaf             just matching leaves
            leafvalue        leaf and value combination for array set
            advanced         hierarchical list structure
            none             no return value
    -checkpath <bool>   Error on path not found
    -match <exact|glob> Path match type
        exact           exact paths only
        glob            glob paths
    -info <bool>        Return object info
<args>          Database subpaths (Optional)
```

Arguments

- **<path>s**

Path into the Catapult SIF database. For detailed information about specifying database paths, refer to “[Command Interface to the SIF Database](#)” on page 352.

- **<switches>**

Use these switches to control how the database is searched and how the returned data is displayed. These switches are common to all commands that take database path arguments. For detailed descriptions of the switches, refer to the section, “[Common Command Switches](#)” on page 356.

- **<args> s**

Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

This command is used to get information from the project, solution, or library databases, or to get system information.

Commands

application get

Examples

Example 1:

The following example queries the Catapult database for the company name. The switch “-return pathvalue” causes the command to return both the database path that was queried and the value stored at that path.

```
application get /SYSTEM/COMPANY_NAME_VERBOSE -return pathvalue
```

The return value is:

```
# /SYSTEM/COMPANY_NAME_VERBOSE {Calypto Design Systems Corporation}
```

Example 2:

This example queries for all of the path nodes under the SYSTEM node. The “-match glob” switch is required in order to evaluate and expand the ‘*’ wildcard. The “-return” switch in this case is set to “path” so that only the database paths are returned and not the values stored there.

```
application get /SYSTEM/* -match glob -return path
```

The return value is:

```
# /SYSTEM/COMPANY_NAME /SYSTEM/COMPANY_NAME_VERBOSE  
/SYSTEM/COMPANY_LOCATION /SYSTEM/DET_VERSION /SYSTEM/SHORT_VERSION  
/SYSTEM/BUILD_NUMBER /SYSTEM/RELEASE_DATE /SYSTEM/RELEASE_VERSION  
/SYSTEM/RELEASE_TYPE /SYSTEM/RELEASE_COPYRIGHT  
/SYSTEM/APPLICATION_SYN_PROJECT_EXT /SYSTEM/APPLICATION_LB_PROJECT_EXT  
/SYSTEM/EMAIL_SUPPORT /SYSTEM/EMAIL_LICENSE /SYSTEM/WEBPAGE  
/SYSTEM/DATE_CURRENT /SYSTEM/DATE_START /SYSTEM/PW_NAME /SYSTEM/PW_PASSWD  
/SYSTEM/PW_UID /SYSTEM/PW_GID /SYSTEM/PW_GECOS /SYSTEM/PW_DIR  
/SYSTEM/PW_SHELL /SYSTEM/UNAME_NODENAME /SYSTEM/UNAME_SYSNAME  
/SYSTEM/UNAME_VERSION /SYSTEM/UNAME_RELEASE /SYSTEM/UNAME_MACHINE  
/SYSTEM/SYS_PID /SYSTEM/ENV_HOME /SYSTEM/ENV_MGC_HOME  
/SYSTEM/ENV_MODEL_TECH /SYSTEM/ENV_CXX_HOME /SYSTEM/ENV_EXIT_ON_EXCEPTION  
/SYSTEM/ENV_LIMIT_LICENSE_PID /SYSTEM/ENV_VCO /SYSTEM/ENV_APPVAR  
/SYSTEM/ENV_APPHOME /SYSTEM/ENV_APP_INI /SYSTEM/ENV_KEEP_TMPS  
/SYSTEM/INI_DEBUG /SYSTEM/INI_LANGUAGE /SYSTEM/INI_CACHE_DIR  
/SYSTEM/INI_TCLSH_CMD /SYSTEM/ENV_TMPDIR /SYSTEM/APPLICATION_NAME  
/SYSTEM/APPLICATION_TITLE /SYSTEM/APPLICATION_TITLE_SHORT  
/SYSTEM/APPLICATION_SUBTYPE /SYSTEM/APPLICATION_PROJECT_EXT  
/SYSTEM/APPLICATION_EXECUTABLE /SYSTEM/BANNER
```

Related Commands

[application exit](#)

[application report](#)

application exit

Closes the Catapult session and returns an exit code.

Syntax

```
application exit ?<code>?
```

Arguments

- *<code>*

An integer value that specifies the status of the application upon exit. The set of valid values is platform dependent, but typically a ‘0’ (zero) means a normal status and any other number represents an error status.

Description

This command immediately terminates the Catapult session. Any unsaved work will be lost. The exit code, if specified, is passed to the operating system shell from which the application was invoked.

Example

The example below closes the Catapult session and returns a status code of ‘2’ to the operating system.

```
application exit 2
```

Related Commands

[application get](#)
[application report](#)

application report

Generates a report of the Catapult licenses in use in the current session.

Syntax

```
application report ?<options>?

<options>          Report options (Optional)
    -filename <string>  Write report to file
    -transcript <bool>   Send report to transcript
    -window <bool>      View report in a window
    -license <bool>      Include license in report
```

Arguments

- **-filename <string>**

Write the report to the specified file path. The path is relative to the current working directory for the Catapult session.

- **-transcript <bool>**

Send the report to the session transcript, which is also captured in the session log file. This is the default option when the application is running in “shell” mode.

- **-window <bool>**

Open the report in a document window in the Catapult session. This is the default option in the graphical user interface.

- **-license <bool>**

Include license information in the report.

Description

This command generates a license report and sends it to the specified output display or file.

Example

This example writes the report to file “MyReport.txt” in the current working directory.

```
application report -license true -filename MyReport.txt
```

Related Commands

[application get](#)
[application exit](#)

catapult

Shell-level command that invokes Catapult.

Syntax

```
catapult ?<switches>? ?<project>?

<switches>          Valid switches: (Optional)
    -shell           Start the application in shell mode
    -product <system_level|block_level|university|library_builder|setup>
                    Start specified product
        system_level   System Level Synthesis [sl] (Default)
        block_level     Block Level Synthesis [bl]
        university      University Version [uv]
        library_builder Library Builder [lb]
        setup           Setup Wizard [sw]
    -file <file>       Source the specified Tcl command file after invoking
    -logfile <file>    Create and open a log file at the specified pathname
    -version          Display the release version banner and exit
    -mglsl_license_fileSpecify license file location
    -license          Specify license checkout preferences
        <string>        Specifies a preferred license
        +<string>       Restricts Catapult to a license
        -<string>       Restricts Catapult from a license
    -stacksize <int>   Specify the default stack size in MB
<project>           Project file to open (Optional)
```

Arguments

- **-shell**

Invokes Catapult in the shell command line user interface. This interface is most useful for running script-driven batch jobs. By default, the GUI is invoked.

- **-product**

Specifies the Catapult product to invoke. The corresponding Catapult license must be installed and available. The *setup* argument launches the Catapult Setup Wizard that configures the Catapult software installation. For more information, refer to “[Using the Catapult Setup Wizard](#)” in the *Catapult C Synthesis Installation Guide*.

- **-file <Tcl_script_pathname>**

Sources the specified Tcl command file after invoking.

- **-logfile <logfile_pathname>**

Creates and opens a log file at the specified pathname.

- **-version**

Returns the release version banner and exits.

- **-mglslicense_file <string>**

Specifies the path to a license file containing the Catapult licenses. This switch overrides the MGLS_LICENSE_FILE environment variable. Refer to the “[Licensing Mentor Graphics Software](#)” manual for information about the MGLS_LICENSE_FILE variable.

- **-license <string>**

Specifies license requests and restrictions for the license server. By default, the first valid composite found in the license file is checked out. Options include:

- **-license <string>** — Specifies a preferred license for Catapult to use. If the specified license is not available, an alternate license is checked out and a warning message displays.
- **-license -<string>** — Restricts Catapult from using a specified license.
- **-license +<string>** — Restricts Catapult to using only a specified license.

For a line-by-line description of a license file, refer to the [FLEXnet Licensing End User Guide](#).

You can use multiple instances of the **-license** switch to specify a prioritized list of target licenses. The order of priority is the order the switches appear on the command line. The first valid license found is checked out.

You can use the [application report](#) command to display a list of the license features currently checked out by Catapult:

```
application report -license true
```

- **-stacksize <int>**

Specifies the default stack size in MB for the Catapult core process. The range of valid values is 8 to 64. The default value is 16.

- **<project>**

Specifies a Catapult project file (.css) to load at invocation.

Description

Invokes Catapult from a shell command line. On Windows, the option switches can be added to the Target field of the Windows Shortcut.

Examples

Example 1

The following example opens the default Catapult product in GUI mode and then *sources* the run.tcl script in the current working directory:

```
catapult -file run.tcl
```

For information on specifying a default product, see [Using the Catapult Setup Wizard](#)

Example 2

The following example opens the Catapult Block-Level product in shell mode and then *sources* the run.tcl script in the current working directory:

```
catapult -product block_level -shell -file run.tcl
```

Example 3

This example uses the *-license* switch twice to specify two preferred licenses. Catapult tries to use *catapultslasc_c* first. If it is not available, *catapultsllpld_c* is tried:

```
catapult -license catapultslasc_c -license catapultsllpld_c
```

Example 4

The following example uses the *-license* switch to restrict the license selection to the *catapultsllpld_c* license:

```
catapult -license +catapultsllpld_c
```

Example 5

The following example uses the *-license* switch to restrict Catapult from selecting the *catapultsllpld_c* license:

```
-license -catapultslasc_c
```

cycle

Used to organize I/O timing, but it may also be used to resolve timing or pipelining issues. The cycle command lets you perform a variety of actions relating to cycle constraints.

The following are valid cycle command options:

- ***cycle add***

Adds cycle constraints on operations. You may specify a range, the minimum or maximum cycles between operations, or the exact cycles between operations.

- ***cycle find_op***

Returns full path of operations in the cycle database.

- ***cycle get***

Gets the value of the cycle constraints on specified operations.

- ***cycle remove***

Removes cycle constraints set on one or more operations.

- ***cycle set***

Sets cycle constraints on operations, control the placement of operators during scheduling, and specify the placement of operators within a loop iteration.

This command replaces the CSTEP_FROM directive. For more information, see “[CSTEPS_FROM](#)” on page 314.

cycle add

Adds cycle constraints on operations to control scheduling.

Syntax

```
cycle add <op1> ?<options>?

<op1>      Operation to be constrained (Required)
<options>   Valid options: (Optional)
    -from    Link constraint on <op1> 'from' <op2>
    -min     Specify minimum cycles between ops
    -max     Specify maximum cycles between ops
    -equal   Specify exact cycles between ops
```

Arguments

- **<op1>**

This required argument specifies the operation name to be constrained. If <op1> is expressed as a *glob* expression, only the first occurrence found will be constrained. The argument can be expressed as a full path of the form used in the Gantt chart, or just the leaf name of the operation. For example:

- Operation name only: **io_read(c:rsc.d)**
- Operation full path: **/cycle_ex/core/main/io_read(c:rsc.d)**

- **-from <op2>**

Specifies an operation, <op2>, relative to which <op1> will be constrained. If this argument is omitted, the target operation is constrained relative to the first cycle of the loop.

- **<-min | -max | -equal > <number_of_cycles>**

This required argument specifies a condition and the number of cycles to be constrained by that condition. Valid conditions are:

- **-min** : The minimum number of cycles to be scheduled.
- **-max** : The maximum number of cycles to be scheduled.
- **-equal** : The exact number of cycles to be scheduled.

Description

This command adds a cycle constraint to the specified operation to control the number of cycles that Catapult will schedule for the operation. The constraint is implemented by setting the CSTEPS_FROM directive on the operation.

Operations are constrained relative to the first cycle of the loop unless the optional argument “-from” is used. The required arguments for the command are the name of the operation to be constrained, a constraint condition (-min, -max or -equal) and the number of cycles for that condition.

To add more than one constraint condition (e.g. both a min and a max), execute the command once for each condition. Do not use this command to modify the value of an existing cycle constraint. Instead, use the “[cycle set](#)” command.

Use the “cycle add” command in conjunction with the Gantt chart, which allows you view the effects of the command. The Gantt chart also enables you to copy operation paths which can be pasted into the cycle command line.

The procedure is as follows:

1. From the Gantt chart view, right click on the operation you want to constrain and select “**Copy Operation Path to Clipboard**” from the popup menu.
2. The cycle commands can only be used when Catapult is in the “architect” state. In the Command window, enter the “go architect” command to set the state.
3. Type the cycle command and paste the operation path you copied from the Gantt chart. Only the leaf (operation name) is necessary if the operation name is unique.
4. Execute the cycle command. Catapult immediately branches a new solution.
5. Reschedule the design and view the results in the Gantt chart by clicking on the “**Schedule**” command icon in the Design Bar, or by entering the following commands:

```
go allocate
view schedule
```

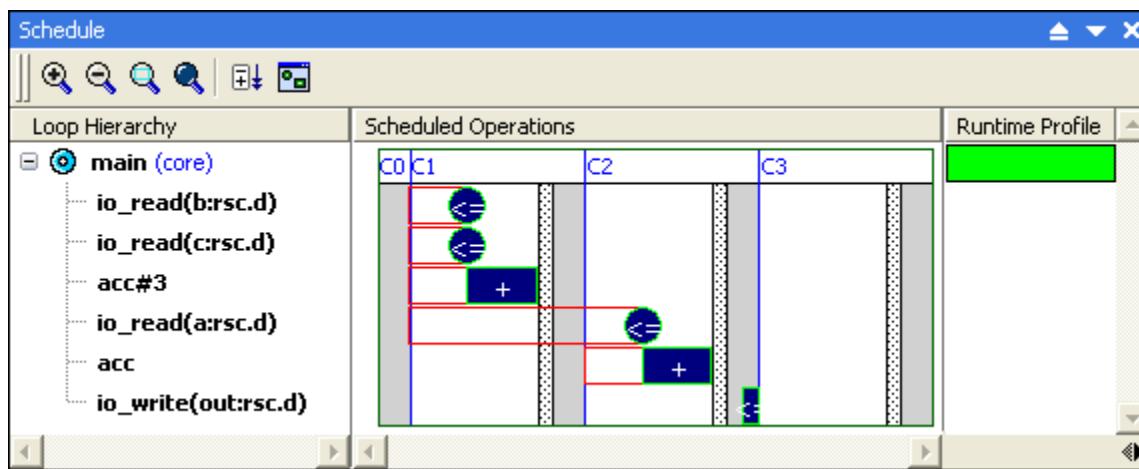
Examples

The examples in this section are based on the following simple design:

```
#pragma design top
void cycle_ex( int a, int b, int c, int *out )
{
    *out = a + b -c;
}
```

For each command example, a Gantt chart is included to show how the command affected the default scheduling of the design. To begin, Figure 8-2 shows the default Gantt chart with no cycle constraints added.

Figure 8-2. Catapult Default Schedule without Constraints

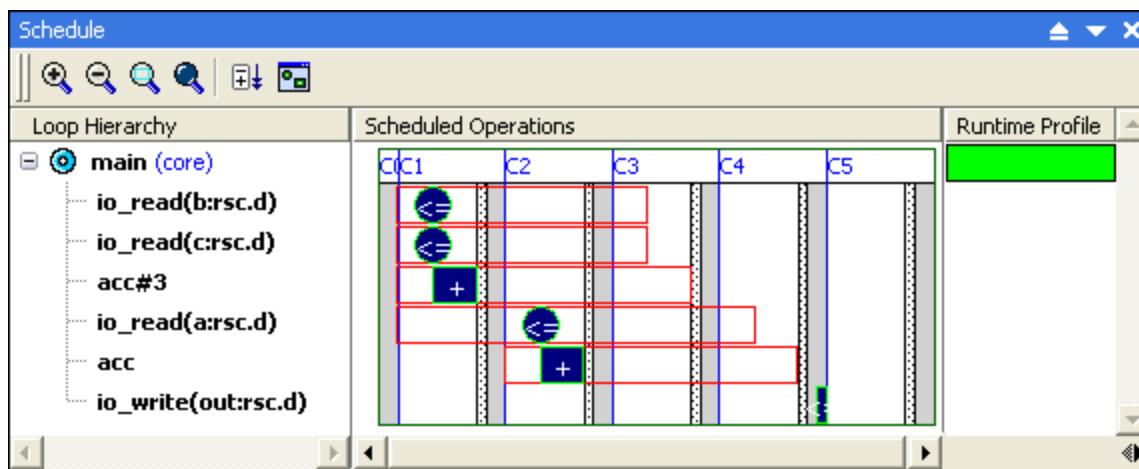


Example 1:

This example executes the “cycle add” command twice in order to establish a minimum and a maximum bound on the number of cycles for the final I/O write operation, “io_write(out:rsc.d)”. The first command sets the minimum bound of 4 cycles from the beginning of the loop, and the second command sets the maximum.

```
cycle add io_write(out:rsc.d) -min 4
# /cycle_ex/core/main/io_write(out:rsc.d)/CSTEPS_FROM
{{/cycle_ex/core/main >= 4} }

cycle add io_write(out:rsc.d) -max 6
# /cycle_ex/core/main/io_write(out:rsc.d)/CSTEPS_FROM
{{/cycle_ex/core/main >= 4} {{/cycle_ex/core/main <= 6}}
```



Example 2:

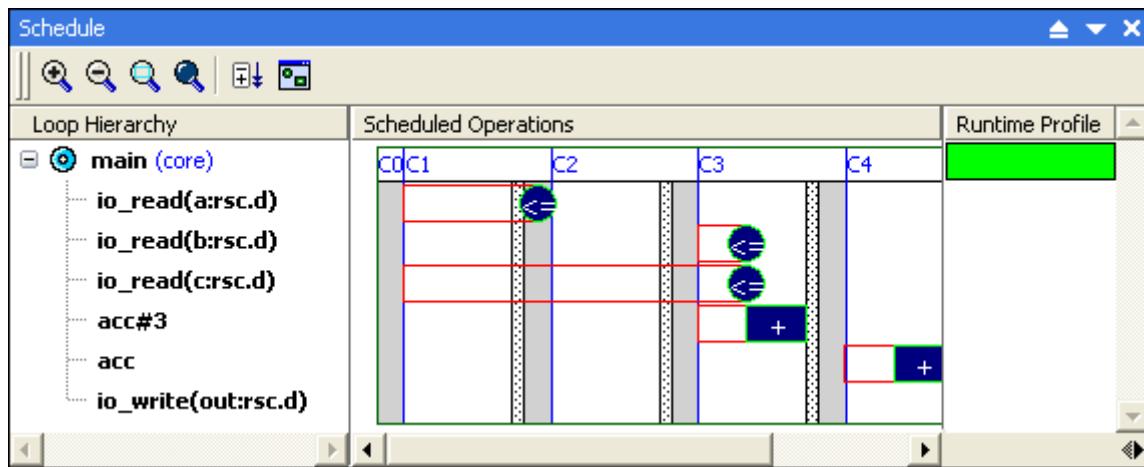
This example constrains the read operation “io_read(b:rsc.d)” so that it is scheduled a minimum of 2 cycles after the read operation “io_read(a:rsc.d)”. The example also shows the use of glob expressions.

```
cycle add io_read(b:*) -from io_read(a:*) -min 2
```

Commands

cycle add

```
# /cycle_ex/core/main/io_read(b:rsc.d) /CSTEPS_FROM  
{ {/cycle_ex/core/main/io_read(a:rsc.d) >= 2}  
{/cycle_ex/core/main/io_read(a:rsc.d) >= 2}}
```



Related Commands

[cycle find_op](#)
[cycle get](#)

[cycle remove](#)
[cycle set](#)

cycle find_op

Returns full path of operations in the cycle database.

Syntax

```
cycle find_op <op_name>  
<op_name>           Name of operation (Required)
```

Arguments

- **<op_name>**

This required argument specifies the operation name to search for. If **<op_name>** is expressed as a *glob* expression, only the first occurrence found will be returned. The argument can be expressed as a full path of the form used in the Gantt chart, or just the leaf name of the operation. For example:

- Operation name only: **io_read(c:rsc.d)**
- Operation full path: **/cycle_ex/core/main/io_read(c:rsc.d)**

Description

This command returns the full path to the specified operation in the database.

Example

This first example returns the path to the “acc” operation.

```
cycle find_op acc  
# /cycle_ex/core/main/acc
```

Related Commands

[cycle add](#)
[cycle get](#)

[cycle remove](#)
[cycle set](#)

cycle get

Gets the cycle constraint settings on the specified operation.

Syntax

```
cycle get <op1> ?<options>?  
  
<op1>           Name of operation (Required)  
<options>        Valid options: (Optional)  
    -from          Get constraint 'from' <op1> to <op2>  
    -return         Specify return value format
```

Arguments

- **<op1>**

This required argument specifies the name of the constrained operation. If **<op1>** is expressed as a *glob* expression, only the first occurrence found will be reported. The argument can be expressed as a full path of the form used in the Gantt chart, or just the leaf name of the operation. For example:

- Operation name only: **io_read(c:rsc.d)**
- Operation full path: **/cycle_ex/core/main/io_read(c:rsc.d)**

- **-from <op2>**

Specifies an operation, **<op2>**, relative to which **<op1>** has been constrained.

Description

This command gets the cycle constraint settings on the specified operation. The return value is a list of constraint settings found on **<op1>**. If the “-from” switch is used, the command returns only the constraints between **<op1>** and **<op2>**. Each element in the list has the form:

```
<origin point for constraint> <condition> <number of cycles>
```

For example:

```
# {{/cycle_ex/core/main >= 6}}
```

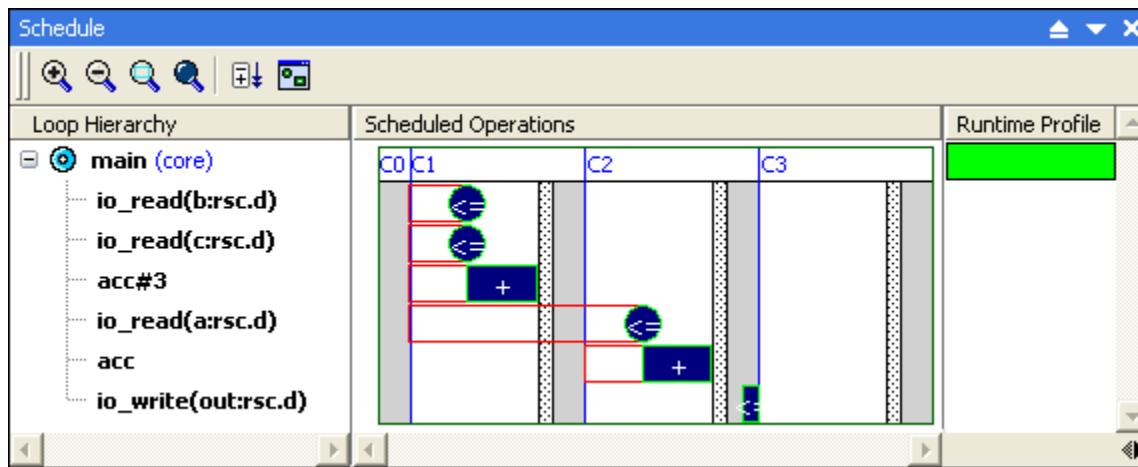
Read this value as follows: **<op1>** must be scheduled six or more (\geq) cycles after the first cycle of the loop **(/cycle_ex/core/main)**.

Examples

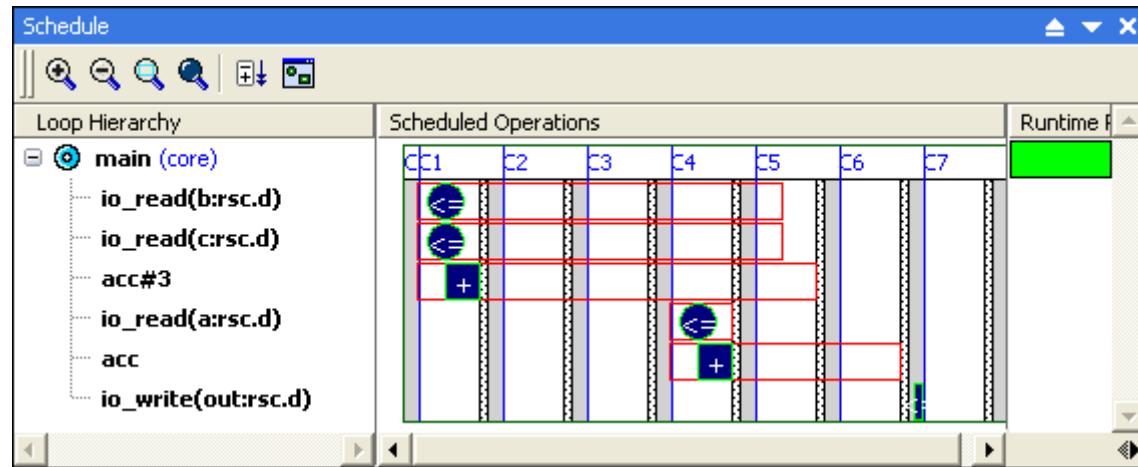
The examples in this section are based on the following simple design:

```
#pragma design top
void cycle_ex( int a, int b, int c, int *out )
{
    *out = a + b -c;
}
```

Figure 8-3 shows the default Gantt chart for this design with no cycle constraints added.

Figure 8-3. Catapult Default Schedule without Constraints

For the purposes of the examples below, two constraints have been set on the “io_write(out:rsc.d)” operation. One constraint requires a minimum of six cycles for the operation. The other constraint requires that the operation be scheduled two cycles after the “io_read(a:rsc.d)” operation. The Gantt chart in Figure 8-4 shows the constrained schedule.

Figure 8-4. Schedule with Cycle Constraints**Example 1:**

This example returns a list of all cycle constraints on the “io_write(out:rsc.d)” operation. In this case, two constraints are listed.

```
cycle get io_write(out:rsc.d)
# {{/cycle_ex/core/main >= 6} {/cycle_ex/core/main/io_read(a:rsc.d) == 2}}
```

The first constraint in the list means that “io_write(out:rsc.d)” must be scheduled at least to 6 cycles after the beginning of the loop. The second means “io_write(out:rsc.d)” must also be scheduled exactly two cycles after the “io_read(a:rsc.d)” operation.

Example 2:

This example returns only those cycle constraints on “io_write(out:rsc.d)” that are relative to the “io_read(a:rsc.d)” operation. The return list contains only one element.

```
cycle get io_write(out:rsc.d) -from io_read(a:rsc.d)  
# {{/cycle_ex/core/main/io_read(a:rsc.d) == 2}}
```

Related Commands

[cycle add](#)
[cycle find_op](#)

[cycle remove](#)
[cycle set](#)

cycle remove

Removes cycle constraint(s) set on the specified operations.

Syntax?

```
cycle remove <op1> ?<options>?  
  
<op1>           Name of operation (Required)  
<options>        Valid options: (Optional)  
    -from          Remove constraint 'from' <op1> to <op2>
```

Arguments

- **<op1>**

This required argument specifies the name of the constrained operation. If <op1> is expressed as a *glob* expression, only the first occurrence found will be reported. The argument can be expressed as a full path of the form used in the Gantt chart, or just the leaf name of the operation. For example:

- Operation name only: **io_read(c:rsc.d)**
- Operation full path: **/cycle_ex/core/main/io_read(c:rsc.d)**

- **-from <op2>**

Specifies an operation, <op2>, relative to which <op1> has been constrained.

Description

The “cycle remove” command removes all cycle constraints on <op1> that are relative to the first cycle of the loop or relative to an operation specified by the “-from <op2>” switch. Any other cycle constraints on <op1> are not affected.

Examples

For the purposes of these examples, assume the operation “io_write(out:rsc.d)” starts out with five cycle constraints, as shown by the “cycle get” command below:

```
cycle get io_write(out:rsc.d)  
  
# {{/cycle_ex/core/main/io_read(c:rsc.d) <= 4}  
#   {{/cycle_ex/core/main/io_read(c:rsc.d) >= 2}  
#     {{/cycle_ex/core/main/io_read(a:rsc.d) >= 5}  
#       {{/cycle_ex/core/main <= 8}  
#         {{/cycle_ex/core/main >= 6}}
```

The first two constraints are relative to the “io_read(c:rsc.d)” operation. The third is relative to the “io_read(a:rsc.d)” operation. The last two constraints are relative to the start of loop.

Example 1:

This example removes the constraints relative to the start of the loop. The return value shows that the CSTEPS_FROM directive value is reset to omit the two targeted constraints and preserve all of the others. The remaining constraints are listed in the return value.

Commands

cycle remove

```
cycle remove io_write(out:rsc.d)

# /cycle_ex/core/main/io_write(out:rsc.d)/CSTEPS_FROM {
    {/cycle_ex/core/main/io_read(c:rsc.d) <= 4}
    {/cycle_ex/core/main/io_read(c:rsc.d) >= 2}
    {/cycle_ex/core/main/io_read(a:rsc.d) >= 5}
}
```

Example 2:

This example removes the constraints relative to the “io_read(c:rsc.d)” operation. The return value shows that the CSTEPS_FROM directive value is reset to omit the two targeted constraints and preserve any remaining constraints.

```
cycle remove io_write(out:rsc.d) -from io_read(c:rsc.d)

# /cycle_ex/core/main/io_write(out:rsc.d)/CSTEPS_FROM
    {{/cycle_ex/core/main/io_read(a:rsc.d) >= 5}}
```

Related Commands

[cycle add](#)
[cycle find_op](#)

[cycle get](#)
[cycle set](#)

cycle set

Sets the value of the specified cycle constraint(s).

Syntax

```
cycle set <op1> ?<options>?

<op1>      Operation to be constrained (Required)
<options>   Valid options: (Optional)
    -from    Link constraint on <op1> 'from' <op2>
    -min     Specify minimum cycles between ops
    -max     Specify maximum cycles between ops
    -equal   Specify exact cycles between ops
```

Arguments

- **<op1>**

This required argument specifies the operation name to be constrained. If <op1> is expressed as a *glob* expression, only the first occurrence found will be constrained. The argument can be expressed as a full path of the form used in the Gantt chart, or just the leaf name of the operation. For example:

- Operation name only: **io_read(c:rsc.d)**
- Operation full path: **/cycle_ex/core/main/io_read(c:rsc.d)**

- **-from <op2>**

Specifies an operation, <op2>, relative to which <op1> will be constrained. If this argument is omitted, the target operation is constrained relative to the first cycle of the loop.

- **<-min | -max | -equal > <number_of_cycles>**

This required argument specifies a condition and the number of cycles to be constrained by that condition. Valid conditions are:

- **-min** : The minimum number of cycles to be scheduled.
- **-max** : The maximum number of cycles to be scheduled.
- **-equal** : The exact number of cycles to be scheduled.

Description

This command modifies a cycle constraint setting on the specified operation. Catapult implements the cycle constraint by setting the CSTEPS_FROM directive on the operation. It is important to understand how loop latency works in relation to c-steps. For more information, see “[Understanding How Loop Latency Works with C-Steps](#)” on page 387.

Operations are constrained relative to the first cycle of the loop unless the optional argument “-from” is used. The required arguments for the command are the name of the operation to be constrained, a constraint condition (-min, -max or -equal) and the number of cycles for that condition.

To modify more than one constraint condition (e.g. both a min and a max), execute the command once for each condition. Use this command in conjunction with the Gantt chart, which allows you view the effects of the command. The Gantt chart also enables you to copy operation paths which can be pasted into the cycle command line.

The procedure is as follows:

1. From the Gantt chart view, right click on the operation you want to constrain and select “**Copy Operation Path to Clipboard**” from the popup menu.
2. The cycle commands can only be used when Catapult is in the “architect” state. In the Command window, enter the “go architect” command to set the state.
3. Type the cycle command and paste the operation path you copied from the Gantt chart. Only the leaf (operation name) is necessary if the operation name is unique.
4. Execute the cycle command. Catapult immediately branches a new solution.
5. Reschedule the design and view the results in the Gantt chart by clicking on the “**Schedule**” command icon in the Design Bar, or by entering the following commands:

```
go allocate
view schedule
```

Examples

Example 1:

Setting the constraint from the beginning of a loop to an operator. The constraint shown below will force three c-steps from the beginning of the default loop to op2.

```
cycle set op2 -equal 3
```

Example 2:

Setting a constraint from an operator to the end of the loop. The constraint shown below will force three c-steps from op2 to the end of the loop. Notice that if only “-from” is specified, op1 is assumed to be the end of the loop.

```
cycle set loop1 -from op2 -equal 3
cycle set -from op2 -equal 3
```

Example 3:

Setting a constraint between two operators. The constraint shown below forces three c-steps from op1 to op2.

```
cycle set op2 -from op1 -equal 3
```

Example 4:

Setting a constraint on the loop. The constraint show below will set the number of c-steps in a loop iteration:

```
cycle set loop1 -equal 7
```

Understanding How Loop Latency Works with C-Steps

The constraint in Example 4 above can lead to confusion when looking at the expected cycle latency of the loop. The cycle latency can be less than (#Iterations * #C-steps). This is because the last iteration may terminate before the last c-step in the loop. If you want to guarantee the exact loop latency, there must be a single TERMINATE that occurs in the last c-step of the loop iteration. If the TERMINATE does not occur in the last cycle and can be moved, you can use the following constraint. However, the Gantt chart should be checked, since you will not always be guaranteed that the TERMINATE can be moved.

```
cycle set terminate_op -equal 0
```

Related Commands

[cycle add](#)
[cycle find_op](#)

[cycle get](#)
[cycle remove](#)

directive

In Catapult, directives are used as a control mechanism to guide the synthesis process. Refer to Chapter 7, “[Directives](#),” for complete information about Catapult directives.

The following are valid directive command options:

- [**directive get**](#)

Returns the value of one or more directives in the specified path.

- [**directive remove**](#)

Removes a directive at the specified path.

- [**directive set**](#)

Sets the value of one or more directives at the specified path.

directive get

Gets the value of the specified directives.

Syntax

```
directive get ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -recurse <string>   Everything under
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
        value      just matching values
        path       just matching paths
        pathvalue  path and value combination for array set
        leaf       just matching leaves
        leafvalue  leaf and value combination for array set
        advanced   hierarchical list structure
        none      no return value
    -checkpath <bool>     Error on path not found
    -match <exact|glob>   Path match type
        exact      exact paths only
        glob       glob paths
    -info <bool>          Return object info
    -inherited_from       Location directive value is inherited from
    -solution <string>   Specify solution
    -state <new|analyze|compile|architect|allocate|
                        schedule|dpfsm|extract|last>
                        Pick design state, default is current
        new        Design state 'new'
        analyze   Design state 'analyze'
        compile   Design state 'compile'
        architect Design state 'architect'
        allocate  Design state 'allocate'
        schedule  Design state 'schedule'
        dpfsm    Design state 'dpfsm'
        extract   Design state 'extract'
        last      Last valid design state for specified solution
    <subpaths>        Database subpaths (Optional)
```

Arguments

- **<path>**

Database path to a directive. The format is:

```
<object_path>/<directive_name>
```

The <object_path> portion specifies the path to a design object on which the directive is set, such as a process, a loop or a component. The path has the form of a HDL design path. The root of the path is the top-level of the design. The <directive_name> portion specifies the directive key in the database. For example:

```
/fir_filter/core/main/UNROLL
```

- **-solution <string>**

Use this switch to access directive settings in a solution other than the active solution.

- **-state <string>**

Use this switch to access directive settings in a state other than the current state of the solution. For more information about Catapult states, refer to [in the section](#).

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section “[Common Command Switches](#)” on page 356.

- **-inherited_from**

Returns the path to the design object from which the directive was inherited instead of the directive setting. If the directive was inherited from the top-level design object, “/” is returned.

- **<subpaths>**

One or more database sub-paths to directives. Sub-paths are relative to the *<path>* argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

The “directive get” command returns the value of the specified directive on the specified design object. By default, only the directive value is returned. Use the “-return” switch to get additional information in the return value.

Examples

Example 1:

Get the value of the [CLOCK_OVERHEAD](#) directive.

```
directive get /fir_filter/CLOCK_OVERHEAD -return leafvalue
```

The return value is:

```
# CLOCK_OVERHEAD 20.0
```

Example 2:

This example gets the path of the design element from which the [CLOCK_OVERHEAD](#) directive was inherited.

```
directive get /fir_filter/CLOCK_OVERHEAD -return leafvalue -inherited_from
```

The return value shows that the directive was inherited from the top-level element. The directive value is not returned.

```
# CLOCK_OVERHEAD /
```

Example 3:

If the <path> argument is omitted and the directive name is supplied as sub-path argument in the <subpaths> field, the command will return the first instance of the directive found in the database. For directives such as CLOCK_OVERHEAD that have only one instance, this is a convenient form of the command. Note that directive names should always be prefixed with a ‘-’ character when specified as a sub-path argument.

```
directive get -CLOCK_OVERHEAD
# 20.0
```

Example 4:

This example returns the value of all instances of the UNROLL directive below the “main” loop. It uses a wildcard expression to simplify the <path> argument, and the directive is supplied as a sub-path argument.

```
directive get /.../main/* -match glob -return pathvalue -UNROLL
```

Two matches are returned, one for each loop (“SHIFT” and “MAC”) inside of “main”.

```
# /fir_filter/core/main/SHIFT/UNROLL no
/fir_filter/core/main/MAC/UNROLL no
```

Example 5:

This example gets all of the directives set on the “MAC” loop. Wildcard expressions are used in both the path argument and the sub-path argument. The path argument expands to the “MAC” loop path. The “-*” sub-path expands to match all directive names on the loop.

```
directive get /.../MAC -match glob -return leafvalue -*
```

The return value lists of all directives for the specified path. The example below shows only the beginning portion of a return list.

```
# CSTEPS_FROM {} DECOUPLING_STAGES 0 DISTRIBUTED_PIPELINE false
IGNORE_DEPENDENCY_FROM {} ITERATIONS 8 ...
```

Related Commands

[directive](#)
[directive remove](#)

[directive set](#)

directive remove

Resets the specified directives to their default values.

Syntax

```
directive remove ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                    Return data format
        value      just matching values
        path       just matching paths
        pathvalue   path and value combination for array set
        leaf        just matching leaves
        leafvalue   leaf and value combination for array set
        advanced    hierarchical list structure
        none       no return value
    -checkpath <bool>     Error on path not found
    -match <exact|glob>   Path match type
        exact      exact paths only
        glob       glob paths
    -info <bool>          Return object info
    -solution <string>    Specify solution
    -state <new|analyze|compile|architect|allocate|
                           schedule|dpfsm|extract|last>
                    Pick design state, default is current
        new        Design state 'new'
        analyze   Design state 'analyze'
        compile   Design state 'compile'
        architect Design state 'architect'
        allocate  Design state 'allocate'
        schedule  Design state 'schedule'
        dpfsm     Design state 'dpfsm'
        extract   Design state 'extract'
        last      Last valid design state for specified solution
    <subpaths>       Database subpaths (Optional)
```

Arguments

- **<path>**

Database path to a directive. The format is:

```
<object_path>/<directive_name>
```

The **<object_path>** portion specifies the path to a design object on which the directive is set, such as a process, a loop or a component. The path has the form of a HDL design path. The root of the path is the top-level of the design. The **<directive_name>** portion specifies the directive key in the database. For example:

```
/fir_filter/core/main/UNROLL
```

- **-solution <string>**

Use this switch to access directive settings in a solution other than the active solution.

- **-state <string>**

Use this switch to access directive settings in a state other than the current state of the solution. For more information about Catapult states, refer to [in the section](#).

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section [“Common Command Switches”](#) on page 356.

- **<subpaths>**

One or more arguments specifying database sub-paths to directives. Sub-paths are relative to the *<path>* argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 355.

Description

The “directive remove” command resets the specified directive on the specified design object to its default value.

Examples

Example 1:

In this example, the “directive set” command first sets the value of the **SAFE_FSM** directive to “true”. Then the “directive remove” command to resets the directive to its default value. Finally, the “directive get” command is used to check the value.

```
directive set /fir_filter/core -SAFE_FSM true
# /fir_filter/core/SAFE_FSM true

directive remove /fir_filter/core/SAFE_FSM
# /fir_filter/core/SAFE_FSM

directive get /fir_filter/core/SAFE_FSM -return pathvalue
# /fir_filter/core/SAFE_FSM false
```

Example 2:

This example resets the UNROLL directive for all sub-loops of the “main” loop:

```
directive remove /fir_filter/core/main/* -match glob
                           -return pathvalue -UNROLL
```

The return value shows that two sub-loops were reset.

```
# /fir_filter/core/main/SHIFT/UNROLL {} /fir_filter/core/main/MAC/UNROLL
{}
```

Related Commands

[directive](#)
[directive set](#)

[directive get](#)

directive set

Sets the specified directive to the specified value.

Syntax

```
directive set ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                    Return data format
        value      just matching values
        path       just matching paths
        pathvalue   path and value combination for array set
        leaf        just matching leaves
        leafvalue   leaf and value combination for array set
        advanced    hierarchical list structure
        none       no return value
    -checkpath <bool>  Error on path not found
    -match <exact|glob> Path match type
        exact      exact paths only
        glob       glob paths
    -info <bool>     Return object info
    -solution <string> Specify solution
    -state <new|analyze|compile|architect|allocate|
                    schedule|dpfsm|extract|last>
                    Pick design state, default is current
        new        Design state 'new'
        analyze   Design state 'analyze'
        compile   Design state 'compile'
        architect Design state 'architect'
        allocate  Design state 'allocate'
        schedule  Design state 'schedule'
        dpfsm    Design state 'dpfsm'
        extract   Design state 'extract'
        last      Last valid design state for specified solution
    <subpaths>     Database subpath and value combinations (Optional)
```

Arguments

- *<path>*

Database path to a directive. The format is:

```
<object_path>/<directive_name>
```

The *<object_path>* portion specifies the path to a design object on which the directive is set, such as a process, a loop or a component. The path has the form of a HDL design path. The root of the path is the top-level of the design. The *<directive_name>* portion specifies the directive key in the database. For example:

```
/fir_filter/core/main/UNROLL
```

- *-solution <string>*

Use this switch to access, and branch from, a solution other than the active solution.

- **-state <string>**

A new solution is branched from the specified state of the current solution or the solution specified by the “-solution” switch. The new solution is set to the specified state and the directive is set in the new solution.

Use this switch to set the solution to a valid state before setting the directive. An invalid state generates an error. The set of valid states depends on which directive is being set. For more information about Catapult states, refer to [in the section](#).

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section [“Common Command Switches”](#) on page 356.

- **<subpaths>**

One or more arguments specifying database sub-paths to directives and their values. Sub-paths are relative to the [`<path>`](#) argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 355.

Description

The “directive set” command sets the value of the specified directive on the specified design object. The default return value includes the path and value arguments. Use the “-return” switch to control the return value.

Examples

Example 1:

This example sets the value of the CLOCK_OVERHEAD directive to a value of 30. Note that the [`<path>`](#) argument is omitted. That is allowable for global directives such as CLOCK_OVERHEAD.

```
directive set -return pathvalue -CLOCK_OVERHEAD 30
# /CLOCK_OVERHEAD 30
```

Example 2:

The following example sets the “MAP_TO_MODULE” directive on the resource “/fir_filter/output:rsc.” The directive maps the library component “mgc_ioprt.mgc_out_fifo_wait” to the resource and sets the FIFO size to 5.

```
directive set /fir_filter/output:rsc
-MAP_TO_MODULE "mgc_ioprt.mgc_out_fifo_wait fifo_sz=5"
```

Related Commands

[directive](#)
[directive remove](#)

[directive get](#)

dofile

Executes the specified Tcl script and prints messages as each command in the script is executed.

Syntax

```
dofile ?options? <filename>
    -verbose      -- Transcript commands and return values
    -quiet        -- Do not transcript commands or return values
    -noerrors     -- Continue processing commands even if error occurs
    -startline    -- Start processing at specified line
```

Arguments

- **<filename>**

This required argument specifies the pathname of a Tcl file to be executed. If only the leaf name is specified, Catapult looks in the current working directory for the file.

- **-verbose**

During execution of the Tcl file, display commands being executed and their return values in the transcript. Enabled by default. The “-verbose” option can be set in the tcl command file by adding the following pragma at the top of the file. The “command” setting turns on verbose messaging and “none” turns it off.

```
// pragma dofile( verbose : <command|none> )
```

- **-quiet**

Do not display commands or return values in the transcript.

- **-noerrors**

Continue processing commands even if an error occurs. The “-noerrors” option can be set in the tcl command file by adding the following pragma at the top of the file:

```
// pragma dofile( noerrors : true )
```

- **-startline <integer>**

Begin processing commands at the specified line number of the file. All lines preceding the specified line are ignored.

Description

The dofile command is an extension of the Tcl source command. In addition to executing a Tcl file, the dofile command sends a message to the standard output device as each command executes. This is very helpful when debugging a Tcl script.

Example

```
dofile -startline 4 -quiet my_file.tcl
```

Related Commands

None.

flow

The flow commands are used to load and manage user-defined flows. For conceptual information about creating and using custom flows, refer to “[Flow Customization](#)” on page 609.

Flow Management:

```
flow get  
flow provide  
flow run
```

Flow Package Management:

```
flow package forget  
flow package names  
flow package present  
flow package provide  
flow package require  
flow package script  
flow package vcompare  
flow package versions  
flow package vsatisfies
```

Flow Package Options Management:

```
flow package option add  
flow package option get  
flow package option remove  
flow package option set
```

flow get

Queries flow database information.

Syntax

```
flow get ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -recurse <string>   Everything under
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
        value      just matching values
        path       just matching paths
        pathvalue  path and value combination for array set
        leaf       just matching leaves
        leafvalue  leaf and value combination for array set
        advanced   hierarchical list structure
        none      no return value
    -checkpath <bool>   Error on path not found
    -match <exact|glob> Path match type
        exact     exact paths only
        glob      glob paths
    -info <bool>    Return object info
<subpaths>        Database subpaths (Optional)
```

Arguments

- **<path>**

A hierarchical path into the SIF database. The root node of <path> is /SOLUTION/FLOWPKGS. For more information, refer to “[Command Interface to the SIF Database](#)” on page 352.

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section “[Common Command Switches](#)” on page 356.

- **<subpaths>**

One or more arguments specifying database sub-paths to flows. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

The “flow get” command queries the Catapult SIF database for information about currently loaded flow packages. Refer to “[Flow Customization](#)” on page 609 for information about how flows operate in Catapult and how to create custom flow packages.

Examples

Example 1:

This example returns the SIF database paths for all of the flows currently loaded:

```
flow get /* -match glob -return pathvalue
```

The return value will be similar to the following:

```
# ModelSim {} NCSim {} OSCI {} Report {} Schedule {} Schematic {}
Netlist {} Precision {} SCVerify {} HDL_Tcl {} Scanner {} Make {}
```

Example 2:

This example gets the version number and option settings of the ModelSim flow package. The <path> argument supplies a partial path that is common to both of the target data paths. The sub-path arguments “-VERSION” and “-OPTIONS” provided the remaining portions of the paths. The “-recurse” switch is enabled in order to traverse the OPTIONS sub-tree.

```
flow get /ModelSim -recurse true -return pathvalue -VERSION -OPTIONS
```

The return value shown below has been reformatted for clarity:

```
# /ModelSim/VERSION 6.0
/ModelSim/OPTIONS {
    FOLDERNAME {
        name FOLDERNAME
        type string
        DESCRIPTION {Output File Folder Name}
        DEFAULT ModelSim
        VALUE ModelSim
    }
    RADIX {
        name RADIX
        type string
        DESCRIPTION {Default radix for WAVE window}
        DEFAULT hex
        VALUE hex
    }
    ...
}
```

Example 3:

This example returns the names and values of all SCVerify flow options:

```
flow get /SCVerify/OPTIONS/* -match glob -return value -name -VALUE
```

The return value will be similar to the following:

```
# RESET_CYCLES 2 SYNC_ALL_RESETS true TB_STACKSIZE 64000000 INVOKE_ARGS {}
REPLAY_ARGS {} MSIM_DEBUG false MAX_ERROR_CNT 0 DEADLOCK_DETECTION true
INCL_DIRS {} LINK_LIBPATHS {} LINK_LIBNAMES {} USE_MSIM true USE_OSCI true
USE_NCSIM false USE_VISTA false USE_VCS false DISABLE_EMPTY_INPUTS false
ENABLE_REPLAY_VERIFICATION false IDLE_SYNCHRONIZATION_MODE false
```

Related Commands

[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package forget

Removes the specified package from the active solution.

Syntax

```
flow package forget <package>
```

Arguments

- **<package>**

A flow package that is currently loaded in the active solution.

Description

The “flow package forget” command removes the specified flow package from the active solution. Other solutions in the project are not affected. The flow package remains in the project’s flow index and can be reloaded with the [flow package require](#) command.

Refer to [“Flow Customization”](#) on page 609 for information about how flows operate in Catapult and how to create custom flow packages.

Example

Suppose the flow package My_Package.flo exists in the project and has been loaded into the current solution. The following command will remove the flow package.

```
flow package forget My_Package
```

Related Commands

[flow get](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package names

Returns the names of all flow packages that are available to Catapult.

Syntax

```
flow package names
```

Arguments

None

Description

The “flow package names” command returns a list of the flow packages in the flow index. When initializing a new project, Catapult scans all of the flow files (.flo) in the search path and constructs an index of the flow packages it finds. The flows are not loaded at this time, but the index enables Catapult to load them dynamically when they are required. Refer to “[Loading Flows](#)” on page 610 for more information. Refer to “[Flow Customization](#)” on page 609 for information about how flows operate in Catapult and how to create custom flow packages.

Example

```
flow package names
```

Returns a list similar to the following:

```
# ArtisanMemories DesignCompiler GIDEL HDL_Tcl Make ModelSim NCSim Netlist  
OSCI Precision Report SCVerify SLEC Scanner Schedule Schematic XPower
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package option add

Creates a new option for a flow package.

Syntax

```
flow package option add <path> ?<switches>?

<path>           Hierarchical database path (Required)
<switches>       Valid switches: (Optional)
    -default <value>     default value of option
    -description <string>   description of option
    -type <type>         option type
```

Arguments

- **<path>**

Hierarchical database path of the new option in the SIF database. The path is rooted at the FLOWPKGS node in the database and has the form /<package_name>/<option_name>. Only the <option_name> portion is required when this command is called from within a flow package script. When called from the command line, the whole path is required, including the leading forward slash.

For more information about the SIF database, refer to “[Command Interface to the SIF Database](#)” on page 352.

- **-default <value>**

Initializes the new option to the specified default value. If the “-default” switch is omitted, no default value is assigned and the option is not initialized.

- **-description <string>**

Prompt string that appears next to the input field on the options dialog page. If the “-description” switch is omitted, the default description text is the value of the <path> argument.

- **-type <value>**

Defines the data type of the new option. Valid types are “string”, “bool”, “integer”, “double”, “inputfile”, “outputfile”, and “directory”.

Description

The “flow package option add” command creates a new option in the database for the specified flow package. The new option is also added to the Flows page of the options dialog (**Tools > Set Options... > Flows > package_name**). If this command is implemented in a flow script, it is only executed during the *flow-indexing* process, which is initiated by calling [flow package require](#). For more information about flow-indexing, see “[Loading Flows](#)” on page 610. Refer to “[Flow Customization](#)” on page 609 for information about how flows operate in Catapult and how to create custom flow packages.

Example

The example below creates and initializes a new option named ReportFile in the flow package MyFlowPackage. The option is initialized to my_flow.rpt, its default value. In the GUI, the new option appears on the options page for the specified flow package, and the description text appears next to the ReportFile option field.

```
flow package option add /MyflowPackage/ReportFile -default "my_flow.rpt"  
-description "Output Report Filename"
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package option get

Returns the value of the specified flow package option.

Syntax

```
flow package option get <path> ?-nocomplain?  
  
<path>           Hierarchical database path (Required)  
<switches>       Valid switches: (Optional)  
    -nocomplain   Do not error if package or option is not defined
```

Arguments

- **<path>**

A pseudo path into the SIF database that has the form /<package_name>/<option_name>. The pseudo path is a shorthand form of the true database path to flow option values: /<package_name>/OPTIONS/<option_name>/VALUE.

The path is rooted at the FLOWPKGS node in the database. The leading forward slash is required.

- **-nocomplain**

Disable error reporting. If the command encounters an error, no error message is reported and the command does not terminate.

Description

The “flow package option get” command returns the current value of the specified flow package option. This command is the preferred method of accessing flow option settings because it takes a simplified form of the <path> argument, as compared to the “[flow get](#)” command.

An error status is returned if the option value is not defined or if <path> is not valid.

Example

The example below returns the value of the option ReportFile in package MyflowPackage. Error reporting is disabled by the -nocomplain switch. The return value is “my_flow.rpt“.

```
flow package option get /MyflowPackage/ReportFile -nocomplain  
# my_flow.rpt
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package option remove

Deletes the specified flow package option.

Syntax

```
flow package option remove <path> ?-nocomplain?

<path>           Hierarchical database path (Required)
<switches>       Valid switches: (Optional)
    -nocomplain   Do not error if package or option is not defined
```

Arguments

- **<path>**

A pseudo path into the SIF database that has the form /<package_name>/<option_name>. The pseudo path is a shorthand form of the true database path to flow option values: /<package_name>/OPTIONS/<option_name>/VALUE.

The path is rooted at the FLOWPKGS node in the database. The leading forward slash is required.

- **-nocomplain**

Disable error reporting. If the command returns an error, no error message is reported and the script does not terminate.

Description

The “flow package option remove” command deletes the option from the SIF database. However, it is not deleted from the Flows options dialog in the GUI.

This command can be used in flow package scripts only if it is in global context (not inside a procedure), and it is valid only in the *flow-indexing* safe interpreter (See “[Safe Interpreters](#)” on page 613). This command can also be called from the command line.

Example

The example below deletes the option ReportFile in package MyflowPackage. Error reporting is disabled by the -nocomplain switch.

```
flow package option remove /MyflowPackage/ReportFile -nocomplain
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option set](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package option set

Sets the value of the specified option.

Syntax

```
flow package option set <path> <value>  
<path>           Hierarchical database path (Required)  
<value>          value of option (Required)
```

Arguments

- **<path>**

A pseudo path into the SIF database that has the form /<package_name>/<option_name>. The pseudo path is a shorthand form of the true database path to flow option values: /<package_name>/OPTIONS/<option_name>/VALUE.

The path is rooted at the FLOWPKGS node in the database. The leading forward slash is required.

- **<value>**

String value assigned to the flow package option.

Description

The “flow package option set” command assigns a string value to the specified flow package option. This command can be used in flow package scripts only if it is in global context (not inside a procedure), and it is valid only in the *flow-indexing* safe interpreter (See “[Safe Interpreters](#)” on page 613). This command can also be called from the command line.

Example

The example below assigns the value project.rpt to option ReportFile in package MyflowPackage.

```
flow package option set /MyflowPackage/ReportFile "project.rpt"
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package present](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package present

Returns the version string of the specified package if the package has been loaded.

Syntax

```
flow package present ?<switches>? <package> ?<version>?

<switches>           Valid switches: (Optional)
    -exact            versions must exactly match
<package>           package name (Required)
<version>            version (Optional)
```

Arguments

- **-exact**

The “-exact” switch is used in conjunction with the <version> argument. It forces Catapult to match the exact <version> specified. For more information about flow package versioning, refer to “[Package Versioning](#)” on page 614.

- **<package>**

The name of a flow package that is currently in the package index. To get a list of all indexed packages use [flow package names](#).

- **<version>**

Specifies a version string to search for. Use the [flow package versions](#) command to get a list of available versions for the package. For more information about flow package versioning, refer to “[Package Versioning](#)” on page 614.

Description

The “flow package present” command returns the version number of the specified flow package if that package has been loaded. Otherwise, an error message

Example

For the following examples, assume two versions of the ModelSim flow (6.0 and 5.7a) have been found in the index, but only the 6.0 version has been loaded. The “[flow package versions](#)” command is used to get the list of the ModelSim flow versions in the index.

```
flow package version ModelSim
# 6.0 5.7a
```

The example below gets the version of the ModelSim flow package that is loaded.

```
flow package present ModelSim
# 6.0
```

The next example explicitly checks to see if version 5.7a is loaded. Because it is not, an error message is returned.

```
flow package present -exact ModelSim 5.7a
# Error: Version conflict for flow package "ModelSim": have "6.0", need
"5.7a"
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package provide](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package provide

Registers a flow package name and compatible version(s).

Syntax

```
flow package provide <path> <package> ?<switches>?

<path>           path (Required)
<package>        package name (Required)
<switches>       Valid switches: (Optional)
    -require <package> ?version ?exact??>
                    require additional flow package
    -description <string>   flow package description
    -author <string>        author of flow package
    -support <string>       support email for author
    -website <string>       website for author
    -icons <string>         icons for website
    -hidden <bool>          set hidden flag
```

Arguments

- **<path>**

The file system path to a flow package file (*.flo). This argument is only required if the “flow package provide” command is called from the Catapult command line. Typically the command is called from within a flow package file. In that context this argument superfluous.

- **<package>**

The name of the flow package. This is the name by which the package will be indexed and referenced in the Catapult system database.

- **-require <package> ?version ?exact??>**

Require an additional flow package. If the current flow package depends on another flow package, the option will “require” the other package if it is not already loaded. Optionally specify the version label for the dependent flow package. Refer to “[flow package require](#)” on page 412.

For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

- **-description <string>**

A string describing the flow package. This string is stored in the SIF database and displayed in the “Description” field of the Flow Manager window when the flow is selected.

- **-author <string>**

The name of the flow author. This string is stored in the SIF database and displayed in the “Author” field of the Flow Manager window when the flow is selected.

- **-support <string>**

An email address to which customer issues can be reported. This string is stored in the SIF database and displayed in the “Support” field of the Flow Manager window when the flow is selected.

- **-website <string>**

A website providing documentation, support and downloads for the flow package. This string is stored in the SIF database and associated with the “Homepage” button on the Flow Manager window when the flow is selected.

- **-icons <string>**

The name of an icon file that will substitute for the “Homepage” button. Catapult expects the icon file to reside in the same directory as the flow file. This string is stored in the SIF database and will appear in the Flow Manager window when the flow is selected.

- **-hidden <bool>**

Enable this switch to hide the flow in the Catapult GUI. Default value is ‘0’ (false).

Description

The “flow package provide” command registers a package name, version(s) and other details about authorship in the Catapult system database. Each time a new project is created, Catapult locates all flow package files in the Flows Search Path, then generates an index of flow packages based on the “flow package provide” command in each file. See “[Loading Flows](#)” on page 610.

Example

The example below adds the package My_Report to the flow package index, and declares that it is compatible with scripts written for the listed versions.

```
flow package provide My_Report -description "Generates custom reports"  
v2.4 v2.5 v3.0
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package require

Loads the specified flow package into the current Solution.

Syntax

```
flow package require ?<switches>? <package> ?<version>?

<switches>      Valid switches: (Optional)
                  -exact           versions must exactly match
                  -source          loads the package source into the current package
<package>        package name (Required)
<version>         Version (Optional)
```

Arguments

- **<package>**

Name of a flow package that exists in the Catapult index. Use the [flow package names](#) command to get a list of all indexed flow packages.

- **-exact**

The “-exact” switch must be used in conjunction with the <version> argument. It forces Catapult to use the exact <version> specified. If the switch is omitted, Catapult will load the latest compatible version of the package. For more information about flow package versioning, refer to [“Package Versioning”](#) on page 614.

- **-source**

When the [flow package require](#) command is called form inside of a flow package file, the “-source” switch causes the required package to be loaded into the current package. Thus, flow from the required package and the current package will run in the same Tcl interpreter.

- **<version>**

Specifies the version of the flow package to load. Use the [flow package versions](#) command to get a list of available versions for the package. If no version is specified, the latest version will be loaded. For more information about flow package versioning, refer to [“Package Versioning”](#) on page 614.

Description

The “flow package require” command makes the flows in the named package available in the solution. Packages must be *required* before their flows can be run. If the specified package is already in the solution, no change is made.

A flow package can access flows from another flow package in two ways:

1. `flow package require <package_name>`
`flow run <package_name>/<flow_name>`
2. `flow package require <package_name> -source <flow_name>`

In the first method, the required package remains external to the current package and you must use the flow run command to invoke its flows. Furthermore, its flows are run in a separate Tcl

interpreter. In the second method, the required package is inlined into the current package, and flows from both packages share the same Tcl interpreter. Be aware that sharing the same interpreter can lead to errors caused by conflicting names of global variables or processes.

Flow code is not loaded into the tool, but is read from the file system each time it is run. See “[Loading Flows](#)” on page 610 for more information. To get a list of available flow packages, use the [flow package names](#) command.

Example

The example below loads flow package My_Report, version “v3.0”.

```
flow package require -exact My_Report v3.0
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package script

Returns the file system path to the flow package file.

Syntax

```
flow package script <package> <version>
<package>      package name (Required)
<version>       package version (Required)
```

Arguments

- **<package>**

The name of a flow package that is currently in the package index. To get a list of all indexed packages use [flow package names](#).

- **<version>**

Specifies the version of the flow package to query. Use the [flow package versions](#) command to get a list of available versions for the package. For more information about flow package versioning, refer to [“Package Versioning”](#) on page 614.

Description

The “flow package script” command returns the full file system path to the flow package file for the specified package name.

Example

```
flow package script ModelSim 6.0
```

Returns the following path:

```
# /Catapult/mgc_home/pkgs/sif/userware/En_na/flows/app_msim.flo
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package require](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package vcompare

Compares two version numbers and determines which one is newer.

Syntax

```
flow package vcompare ?<switches>? <version1> <version2>  
  
<switches>      Valid switches: (Optional)  
    -nocomplain   Do not error if either version is invalid  
<version1>      version for comparison (Required)  
<version2>      version for comparison (Required)
```

Arguments

- **<version1> and <version2>**

Flow package versions to be compared. Use the [flow package versions](#) command to get a list of available versions for the package. For more information about flow package versioning, refer to “[Package Versioning](#)” on page 614.

- **-nocomplain**

Do not report an error if either <version> argument is invalid.

Description

The “flow package vcompare” command compares the ordinal value of <version1> to <version2> and returns a status as follows:

Table 8-8. *flow package vcompare* return states

Status	Description
-1	<version1> is older (lower ordinal value)
0	<version1> and <version2> are equivalent
1	<version1> is newer (higher ordinal value)

For more information about flow package versions, refer to “[Package Versioning](#)” on page 614.

Example

The example below compares the version strings “2007a.12” and “2007a.15” and returns the status “-1” that indicates string 2 (2007a.15) is the more recent version.

```
flow package vcompare 2007a.12 2007a.15  
# -1
```

Commands

flow package vcompare

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package require](#)
[flow package script](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package versions

Returns a list of all available versions of the specified flow package.

Syntax

```
flow package versions <package>  
<package>          package name (Required)
```

Arguments

- **<package>**

The name of a flow package that is currently in the package index. To get a list of all indexed packages use [flow package names](#).

Description

The “flow package versions” command returns the list of all versions of the specified flow package. For more information about flow package versions, refer to [“Package Versioning”](#) on page 614.

Example

The example below returns the list of versions for the ModelSim flow package

```
flow package version ModelSim  
# 6.0 5.7a
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package vsatisfies](#)
[flow provide](#)
[flow run](#)

flow package vsatisfies

C.compares two version strings and determines whether they are compatible.

Syntax

```
flow package vsatisfies ?<switches>? <version1> <version2>

<switches>      Valid switches: (Optional)
    -exact        versions must be equivalent
<version1>      version for comparison (Required)
<version2>      version for comparison (Required)
```

Arguments

- **-exact**
Specifies that both versions must be equivalent.
- **<version1> and <version2>**
Flow package versions to be compared. Use the [flow package versions](#) command to get a list of available versions for the package. For more information, refer to “[Package Versioning](#)” on page 614.

Description

The “flow package vsatisfies” command compares the ordinal value of <version1> to <version2>. If <version1> is lower, it is not compatible with <version2> and the return status is ‘0’. If <version1> is greater than, or equal to <version2>, the versions are compatible and the return status is ‘1’.

Example

This example compares two versions of the Precision flow package to demonstrate their ordinal relationship and compatibility. First, version1 is compared to version2. Then the order is reversed and version2 is compared to version1.

```
flow package vsatisfies 2005c.151 2005c.128
# 1
flow package vsatisfies 2005c.128 2005c.151
# 0
```

In the first case, version “2005c.151” is compatible with version “2005c.128.” The second case, however, shows that the reverse is not true.

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow provide](#)
[flow run](#)

flow provide

Adds a flow to a package index.

Syntax

```
flow provide ?<path>? ?<switches>? ?<subpaths>?  
<switches>  
--  
-return <value|path|pathvalue|leaf|leafvalue|advanced|none>  
-checkpath <bool>  
-match <exact|glob>  
-info <bool>  
  
<path>           Hierarchical database path (Optional)  
<switches>       Valid switches: (Optional)  
--                End <switches> parsing  
-return <value|path|pathvalue|leaf|leafvalue|advanced|none>  
                  Return data format  
  value            just matching values  
  path             just matching paths  
  pathvalue        path and value combination for array set  
  leaf              just matching leaves  
  leafvalue        leaf and value combination for array set  
  advanced         hierarchical list structure  
  none             no return value  
-checkpath <bool>    Error on path not found  
-match <exact|glob>  Path match type  
  exact            exact paths only  
  glob             glob paths  
-info <bool>        Return object info  
<subpaths>        Database subpath and value combinations (Optional)
```

Arguments

- **<path>**

A hierarchical path into the SIF database. The root node of <path> is /SOLUTION/FLOWPKGS. The leaf of the path is name of a flow. For more information, refer to “[Command Interface to the SIF Database](#)” on page 352.

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section “[Common Command Switches](#)” on page 356.

- **<subpaths>**

One or more arguments specifying database sub-paths to flows and option values. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Database options of interest are -states, -enabled, and -netlists. The first two are described in the “flow provide” example below. The “-netlists <file_type>” option causes the flow to run while netlists are generated. Valid <file_type> values are: all, vhdl, verilog, systemc.

Description

The “flow provide” command registers a flow in the Catapult flow index. By default, a flow is only accessible from within its own flow package unless the “-states <stage_list>” argument is supplied. Once associated with one or more design flow stages, the flow can be called from the Catapult command line and from other flow packages.

The ‘flow provide’ command is available only in the *flow-indexing* safe interpreter (See “[Safe Interpreters](#)” on page 613). When a “[flow package require](#)” command is issued, Catapult reads the flow package script file and processes each “flow provide” command in the file. See “[Loading Flows](#)” on page 610.

Example

The example below uses the “flow provide” command to add the flow gen_report to the Catapult flow index. The “-states {any}” argument makes the procedure available at any stage in the design flow. The “-enabled {extract}” argument enables the procedure to automatically execute when the *extract* stage completes.

```
1  ## Package Name: My_reports
2
3  flow provide gen_report -states {any} -enabled {extract}
4
5  proc gen_report {
6      ...
7  }
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow run](#)

flow run

Executes a flow procedure.

Syntax

```
flow run <flow>  
<flow>    flow name (Required)
```

Arguments

- **<flow>**

Hierarchical database path to the flow procedure. The path is rooted at the FLOWPKGS node in the database and has the form /<package_name>/<flow_name>. The leading forward slash is required. The /<package_name> portion can be omitted if <flow_name> is in the same package with the “flow run” command.

Description

The “flow run” command launches the specified flow procedure. The flow package defining <flow> must be in the Catapult index (See [flow package provide](#) and [flow package require](#)).

Example

The example below launches

```
flow run /My_Report/gen_report
```

Related Commands

[flow get](#)
[flow package forget](#)
[flow package names](#)
[flow package option add](#)
[flow package option get](#)
[flow package option remove](#)
[flow package option set](#)
[flow package present](#)

[flow package provide](#)
[flow package require](#)
[flow package script](#)
[flow package vcompare](#)
[flow package versions](#)
[flow package vsatisfies](#)
[flow provide](#)

go

Starts the synthesis flow and moves to the specified state.

Syntax

```
go ?<state>?

<state>
  new      Set state 'new'
  analyze  Set state 'analyze'
  compile  Set state 'compile'
  architect Set state 'architect'
  allocate  Set state 'allocate'
  schedule  Set state 'schedule'
  dpsfm    Set state 'dpsfm'
  extract   Set state 'extract'
```

Arguments

- **<state>**

Specifies the design state to go to. For more information about the Catapult states, see the section “[The Catapult Work Flow](#)” on page 46.

Description

The C synthesis flow goes through several transformation states. While a design is in a specified state, only certain commands and directives can be executed.

The go command starts the flow execution at the end of the current state and moves toward the specified state. Going forward in the flow executes all the states that are required to get to and complete the target state. Going back in the flow enables you to branch a new solution by changing a constraint that affects that state.

Example

Given a design in any state prior to the “extract” state, the following command will advance the design to the “extract” state by executing all of the necessary intermediate commands.

```
go extract
```

Related Commands

None.

help command

Gets help on command syntax and usage.

Syntax

```
help command ?<args>?  
<args>      <pattern> | <command> <args> ... (Optional)
```

Arguments

- **<args>**

This argument specifies a Catapult command name for which help information is returned. The “<pattern> | <command>” field represents the first word in the command name and can be expressed as a literal string or a wildcard glob expression using the asterisk (“*”) character. The “<args> ...” field represents any additional words in the command name. For information about Tcl command syntax with respect to Catapult command names, refer to “[General Command Syntax](#)” on page 349.

Description

The help command returns help on the syntax and usage of Catapult commands. If no command name is specified, a list of command names is returned.



Note

Note, help about command usage is also accessible by invoking the command with the “-help” switch. For example, “resource -help” or “resource add -help”. The “--help” (double dash) option displays recursive help for hierarchical commands.

Examples

Example 1:

This example returns only the list of command names because *<args>* is omitted.

```
help command  
  
# User commands:  
# application  
# component  
# cycle  
# directive  
# flow  
# go  
# help  
# ignore_memory_precedences  
# library  
# logfile  
# options  
# project  
# resource  
...
```

Example 2:

This example returns the syntax for all of the “view” command operators (schedule, schematic, file and source):

```
help command view
# Usage: view <type>      View or edit design files
#   type <type> ?<args>?
#   <type>          Operation for <object>
#   <type>          Operation for <object> (Required)
#
#   schedule        Gantt Chart Schedule of Operations
#
#   schematic ?<type>? ?<switches>?
#   type            RTL Schematic
#   rtl             schematic type
#   critical        RTL schematic
#   datapath        Critical path schematic
#   criticalmap    Datapath schematic
#   criticalmap    Critical map schematic
#   <switches>      Valid switches: (Optional)
#   -count <int>   Critical path count (valid for 'critical' type)
#   -to <string>   Highlight path to operation (valid for
'critical'
#               mode)
#   -from <string>
#               Highlight path from operation (valid for
#               'critical' mode)
#   -through <string>
#               Highlight path through operation (valid for
#               'critical' mode)
#   -map <int>     Display timing map for paths exceeding % of
clock
#               period (valid for 'criticalmap' mode)
#
#   file <filepath> ?<switches>?
#               Any file type
#   <filepath>      Relative or absolute file path (Required)
#   <switches>      Valid switches: (Optional)
#   -filetype <string>
#               file type
#   -branch <string>
#               name
#
#   source          C/C++ Source files
#   <args>          Operator specific arguments (Optional)
```

Example 3:

This example returns the syntax for “view file” command only:

```
help command view file
# Usage: view file <filepath> ?<switches>?
#               Any file type
#   <filepath>      Relative or absolute file path (Required)
#   <switches>      Valid switches: (Optional)
#   -filetype <string> file type
#   -branch <string> name
```

Commands

help command

Related Commands

[help message](#)

help message

Gets help on system message codes.

Syntax

```
help message <id> ?<switches>?

<id>                  message identifier (Required)
<switches>             Valid switches: (Optional)
    -description        return message long description
    -hasdescription     return true if message has long description
    -minseverity        return message minimum severity
    -severity           return message severity
```

Arguments

- **<id>**

Each system message has a unique identifier associated with it, which is displayed in the transcript window at the end of the short description. The identifiers are composed of two elements, a text label that indicates type of operation that was being performed when the message was generated, and an integer to uniquely identify each message associated with a label. Some examples are:

```
ASSERT-1
CIN-6
LOOP-4
PRJ-1
```

For more information about Catapult system messages IDs, refer to “[Message Identifiers](#)” on page 161.

- **-description | -hasdescription | -minseverity | -severity**

One of these four optional switches can be used to specify the type of information that is returned. If none is specified, “-description” is the default.

Description

The “help message” command returns help information about Catapult system message codes (IDs). Catapult displays a brief form of system messages (error, warning and info) in the transcript window. At the end of the brief message is a message ID that, in many cases, corresponds to a longer, more detailed description of the message. The “-hasdescription” switch tells you whether or not a long description is available. Use the “-description” switch to get the long description. (In the GUI you can simply double-click on the ID to the view the long description.)

The “-severity” and “-minseverity” return the present *severity* setting and the minimum severity setting of the message. The severity level (“error”, “Warning” or “Information”) of a message is user-configurable. Refer to “[Severity Classification of Messages](#)” on page 161, and “[Message Options](#)” on page 176 for more information.

Examples

Example 1:

This example first shows the how the short form of the “LOOP-4” message would appear in the transcript window. Following that, the “help message” command is used to get the long description.

Short form of message:

```
# fir_filter.cpp(26): Loop '/fir_filter/fir_filter_proc/fir_filter_main/SHIFT' is left rolled.  
(LOOP-4)
```

Get the long form of the message:

```
help message LOOP-4 -description  
# The loop listed in this message is not being unrolled. This could be  
because the UNROLL directive is set to "no" for this loop. Or it could be  
because the UNROLL directive is set to "yes" but the number of iterations  
is not known.
```

Example 2:

This example calls the “help message” command twice to the get severity information about the “LOOP-4” message.

Current severity:

```
help message ASSERT-1 -severity  
# error
```

Minimum severity:

```
help message ASSERT-1 -minseverity  
# warning
```

Related Commands

[help command](#)

ignore_memory_precedences

Ignores data flow dependencies between the specified read/write operations of a memory array.

Syntax

```
ignore_memory_precedences [-process <proc_name>] -from <op1> -to <op2>  
[-verbose]
```

Arguments

- **-process <process_name>**

Optional switch that specifies the pathname of the hierarchical block containing `<op1>` and `<op2>`. Typically Catapult includes the process name in the operation name, so this option is not needed. However, you can use this option to resolve any ambiguity. You can determine process pathnames from the Gantt chart by right-clicking on a process and choosing Copy Operation Path to Clipboard.

- **-from <op1> -to <op2>**

Specifies two read/write operations that have a data flow dependent relationship. The `<op1>` and `<op2>` values are pathnames of the operations. You can determine operation pathnames from the Gantt chart by right-clicking on an operation and choosing Copy Operation Path to Clipboard.

The data flow dependencies are only ignored in the *from to* direction specified. To ignore the data flow dependencies in both directions, you must specify the complimentary command.

Description

In certain cases, Catapult may generate a false data flow dependency between two memory operations on an array because the index analysis in Catapult cannot determine that the index values for the two array accesses are never the same. In these cases, if you are certain that the two array accesses are truly independent, you can direct Catapult to ignore such false data flow dependencies and possibly save time.

Caution



Ignoring true data flow dependencies may produce an incorrect design.

This command cannot be used until the architectural constraints are applied to the design (go architect).

Note



The effect of this command is not inherited by solutions subsequently created by Catapult.

Examples

The following example shows how to ignore the data dependencies progressively between the `read_mem_dct_proc_temp_rsc_1_i21_153` and `write_mem_dct_rsc_1_u704_154` operations:

```
ignore_memory_precedences -from read_mem_dct_proc_temp_rsc_1_i21_153  
                           -to write_mem_dct_rsc_1_u704_154
```

The following example specifies the process block and shows how to ignore the data dependencies in both directions between the `read_mem_dct_proc_temp_rsc_1_i21_153` and `write_mem_dct_rsc_1_u704_154` operations:

```
ignore_memory_precedences -process /dct_proc/dct_proc_proc/dct_proc_main  
                           -from read_mem_dct_proc_temp_rsc_1_i21_153  
                           -to write_mem_dct_rsc_1_u704_154
```

```
ignore_memory_precedences -process /dct_proc/dct_proc_proc/dct_proc_main  
                           -from write_mem_dct_rsc_1_u704_154  
                           -to read_mem_dct_proc_temp_rsc_1_i21_153
```

Related Commands

None

logfile

The logfile commands are used for managing the Catapult session log file, and for adding content to log file. Messages sent to the logfile are also sent to the Transcript window in the Catapult GUI.

File Management:

[logfile close](#)
[logfile isopen](#)
[logfile move](#)
[logfile name](#)
[logfile open](#)

Add Content:

[logfile message](#)
[logfile save_commands](#)

logfile close

Closes the Catapult session log file.

Syntax

```
logfile close
```

Arguments

None.

Description

The “logfile close” command closes the session log file. An error is returned if no log file is currently open. If the GUI is open, the Transcript window continues to record all executed commands and system messages.

Example

```
logfile close
```

Related Commands

[logfile isopen](#)
[logfile message](#)
[logfile move](#)

[logfile name](#)
[logfile open](#)
[logfile save_commands](#)

logfile isopen

Reports whether or not a Catapult session log file is open.

Syntax

```
logfile isopen
```

Arguments

None.

Description

If a session log file is currently open, the “logfile isopen” command returns 1 (true). Otherwise it returns 0 (false).

Example

```
logfile isopen  
# 1
```

Related Commands

[logfile close](#)
[logfile message](#)
[logfile move](#)

[logfile name](#)
[logfile open](#)
[logfile save_commands](#)

logfile message

Sends messages to the session log file and the Transcript window in the Catapult GUI.

Syntax

```
logfile message <string> ?<severity>? ?<options>?
  <string>      Message to post (Required)
  <severity>    Message severity level
    comment      Message severity "comment"
    info         Message severity "info"
    warning      Message severity "warning"
    error        Message severity "error"
    command      Message severity "command"
    subcommand   Message severity "subcommand"
    noprefix     Message severity "noprefix"
    unknown      Message severity "unknown"
  <options>     Message options (Optional)
    -xref <filepath> ?<line>? ?<column>? ?<length>?
      Message file cross reference information
      <filepath>   Relative or absolute file path (Required)
      <line>       line number in file (Optional)
      <column>     column number of line (Optional)
      <length>     length from column number (Optional)
    -index <id>   Message index
```

Arguments

- **<string>**

The text of the message being logged. The string must be terminated with a “\n”.

- **<severity>**

Use one of the severity keywords described in Table 8-9 to apply predefined format to the message <string>. In the log file, each keyword prefixes a predefined string to the message to indicate its severity level. In the GUI, icons and font colors are used instead of prefix strings.

Table 8-9. Log Message Severity Formatting Options

Severity Keyword	Prefix String in Log File	Color of Message in GUI
command	(No prefix)	Blue
comment	#	Black
error	# Error:	Red
info	# Info:	Green
warning	# Warning:	Orange
subcommand	# >	Blue
noprefix	(No prefix)	Blue

Table 8-9. Log Message Severity Formatting Options

Severity Keyword	Prefix String in Log File	Color of Message in GUI
unknown	Dynamically determined. See description below.	Dynamically determined.

The “unknown” keyword tries to determine the severity level of the message by searching for severity keywords embedded in the message string. Embedded keywords must be all uppercase in order to be recognized. If multiple keywords are found, the last one found is used.

- **-xref <filepath> ?<line>? ?<column>? ?<length>?**

This option only applies to the Transcript window in the Catapult GUI. It adds a cross-probe link in the “File(line)” column of the Transcript window. Double-clicking the link will open the specified file in a separate window.

Table 8-10. Arguments for the -xref Option

Argument	Description
<filepath>	Specifies the cross-probe destination file. It can be an absolute path or relative to the project directory.
?<line>?	Line number to be highlighted in the destination file.
?<column>?	Column number at which highlighting begins.
?<length>?	Number of columns in the highlighted area.

Only the <filepath> argument is required. If the optional arguments are omitted, the cross-probe link opens the destination file, but does not highlight any text.

- **-index <id>**

This option only applies to the Transcript window in the Catapult GUI. The option adds a reference to additional information (also referred to as a “long description”) about the message <string>. The <id> value is a unique ID of a registered long description. The ID will appear in the “Id” column of the Transcript window.

Description

The “logfile message” command sends text strings to the session log file and the Transcript window in the GUI. The <severity> option applies formatting attributes to the message corresponding to predefined severity levels, such as error, warning, comment, and so on. If the Catapult GUI is open, the “-xref” and “-index” options can be used to add a cross-probe target and additional help information, respectively, to the Transcript window. Refer to “[Understanding Messages in the Transcript](#)” on page 159 for more information about message severity levels and message ID tags.

Examples

Example 1:

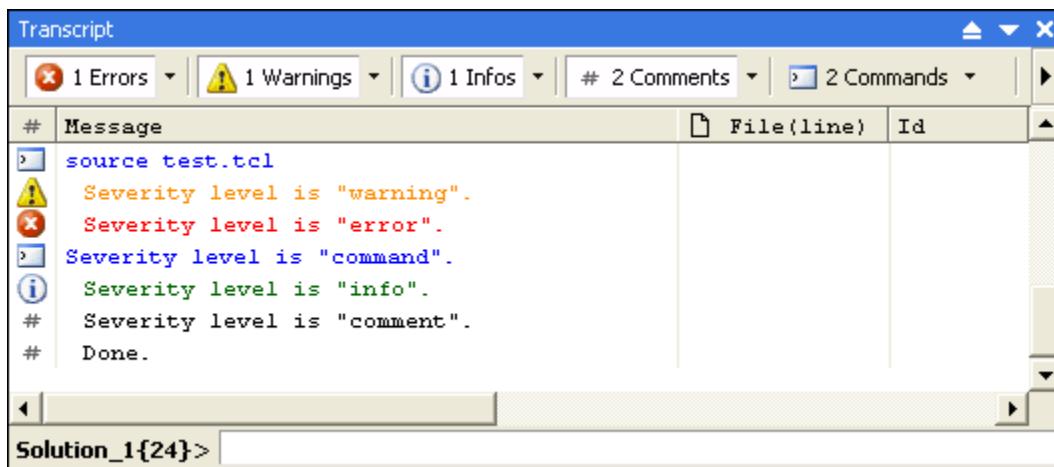
The following Tcl code demonstrates how the various <severity> settings affect the appearance of the message string in the log file and in the Transcript window. Assume the Tcl code was saved in a file named “test.tcl” and the *source* command is used to run the script from the Catapult command line.

```
set level "warning error command info comment"
foreach severity $level {
    logfile message "Severity level is \"$severity\".\n" $severity
}
puts "Done."
```

Each message in the log file has the appropriate prefix string corresponding to its severity.

```
source test.tcl
# Warning: Severity level is "warning".
# Error: Severity level is "error".
Severity level is "command".
# Info: Severity level is "info".
# Severity level is "comment".
# Done.
```

Similarly in the Transcript window, the appropriate severity icon is appears next to each message.



Example 2:

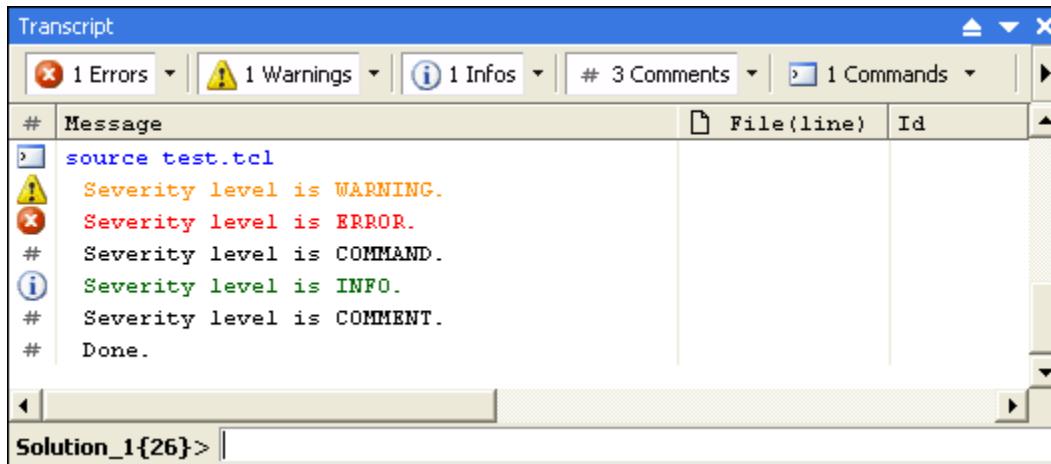
This example demonstrates how the severity option “unknown” is used to dynamically determine the severity level of the message string. The Tcl code is essentially the same as in example 1, except that the “unknown” option is used and the severity keywords in each message string are upper case in order for the “unknown” option to recognize them.

```
set level "WARNING ERROR COMMAND INFO COMMENT"
foreach severity $level
    logfile message "Severity level is $severity.\n" unknown
}
puts "Done."
```

The log file output is shown below. Note that the “COMMAND” keyword is not recognized by the “unknown” parser, so the message is treated as a comment, the default severity level.

```
source test.tcl
# Warning: Severity level is WARNING.
# Error: Severity level is ERROR.
# Severity level is COMMAND.
# Info: Severity level is INFO.
# Severity level is COMMENT.
# Done.
```

The output in the Transcript window is shown in the figure below.



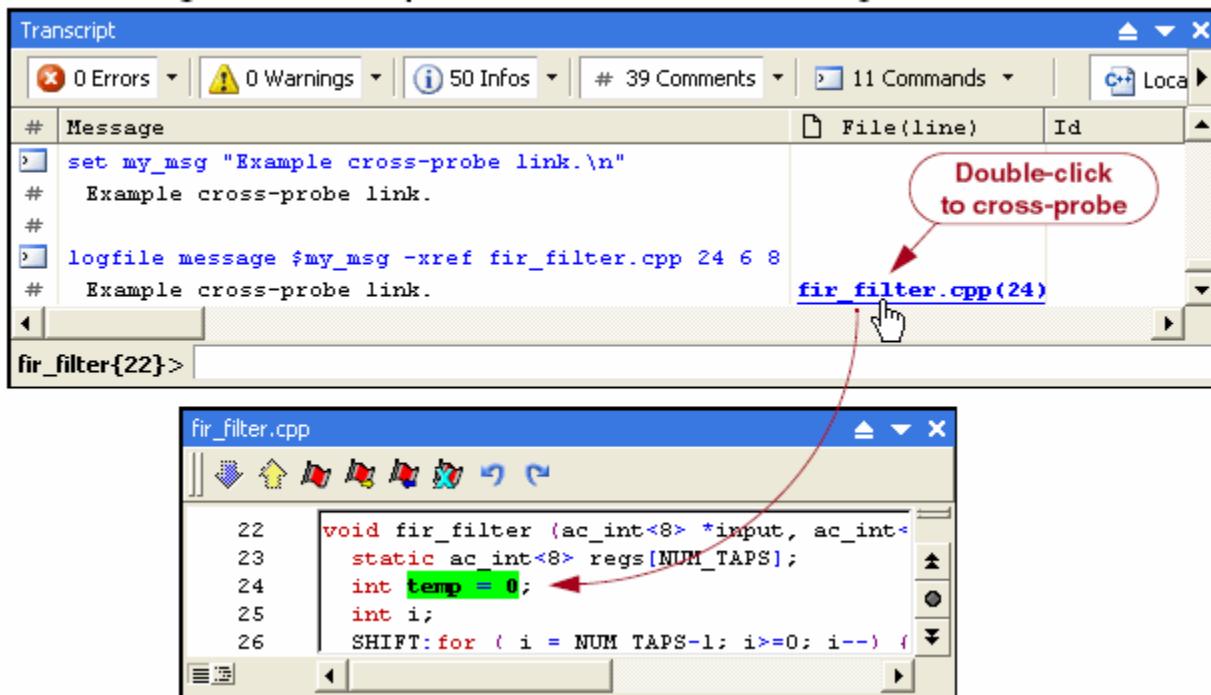
Example 3:

This example demonstrates how to insert a cross-probe link in the Catapult Transcript window. The target file, fir_filter.cpp, is in the project directory, so only the filename is required. The link will open the target file and highlight a substring of text on line 24. The string begins at column 6 and is 8 characters long.

```
set my_msg "Example cross-probe link.\n"
logfile message $my_msg -xref fir_filter.cpp 24 6 8
```

Figure 8-5 shows how the message appears in the Transcript window and the highlighted text in the target file.

Figure 8-5. Example Cross-Probe Link from Logfile Command



Related Commands

[logfile close](#)
[logfile isopen](#)
[logfile move](#)

[logfile name](#)
[logfile open](#)
[logfile save_commands](#)

logfile move

Relocates and/or renames the session log file.

Syntax

```
logfile move ?<switches>? ?<file>?  
          move open logfile to new location  
<switches>      Valid switches: (Optional)  
    -enable        Enable automatic logfile move  
    -disable       Disable automatic logfile move  
<file>         Move logfile to file (Optional)
```

Arguments

- **-enable**

Allows Catapult to automatically move the log file when the working directory is set (or reset) with the [set_working_dir](#) command. The [set_working_dir](#) command is only available when Catapult prior to the creation of the current project directory. That condition only occurs when the tool is first invoked or immediately after the [project new](#) command has been called, and prior to calling the “go analyze” command. This option is on by default.

- **-disable**

Does not allow Catapult to automatically move the log file when the working directory is set (or reset) with the [set_working_dir](#) command.

- **<file>**

File system path to the new log file. The path specification can be relative to the current working directory or an absolute path.

Description

The “logfile move” command relocates and/or renames the current Catapult session log file to the specified file path, <file>.

Example

```
logfile move "/Catapult/dct/my_logfile.log"  
# Moving session transcript to file "/Catapult/dct/my_logfile.log"
```

Related Commands

[logfile close](#)
[logfile isopen](#)
[logfile message](#)

[logfile name](#)
[logfile open](#)
[logfile save_commands](#)

logfile name

Returns the full pathname of the current session log file.

Syntax

```
logfile name
```

Arguments

None.

Description

The “logfile name” command returns the full pathname of the current session log file.

Example

```
logfile name
# /Catapult/dct/catapult.log
```

Related Commands

[logfile close](#)
[logfile isopen](#)
[logfile message](#)

[logfile move](#)
[logfile open](#)
[logfile save_commands](#)

logfile open

Opens a Catapult session log file at the specified path.

Syntax

```
logfile open ?<switches>? ?<file>?  
          open a logfile for writing  
<switches>           Valid switches: (Optional)  
          -append            Append existing file  
<file>              Log filename to open (Optional)
```

Arguments

- **-append**

If opening an existing file, use this switch to preserve the existing contents of the file and append new log output to the end of the file.

- **<file>**

File system path to a new or existing log file. The path specification can be relative to the session working directory ([set_working_dir](#) command) or an absolute path. If <file> is omitted, a log file is created in the directory <catapult_install_tree>/tmp/<file>, where <file> is a unique generated name. Use the “[“logfile name”](#) command to get the name of the currently open log file.

Description

This command creates and opens a Catapult session log file at the specified path. If a log file is currently open, this command will return an error and leave the current log file open.

Example

```
logfile open /Catapult/dct/my_catapult.log  
# Logging session transcript to file "/Catapult/dct/my_catapult.log"
```

Related Commands

[logfile close](#)
[logfile isopen](#)
[logfile message](#)

[logfile move](#)
[logfile name](#)
[logfile save_commands](#)

logfile save_commands

Saves all session commands to a file.

Syntax

```
logfile save_commands <file>
    save commands from open logfile
    <file>           Save commands to file (Required)
```

Arguments

- <file>

File system path to a new or existing file. The path specification can be relative to the session working directory ([set_working_dir](#) command) or an absolute path.

Description

The “logfile save_commands” command writes out all of the Tcl commands you enter during the current project session. Commands executed in scripts are not written in the save_commands file. Such scripts include Catapult startup scripts and scripts called by the [dofile](#) and “source” commands.

Example

```
logfile save_commands /Catapult/dct/my_session_commands.tcl
# Commands saved to '/Catapult/dct/my_session_commands.tcl'
```

Related Commands

[logfile close](#)
[logfile isopen](#)
[logfile message](#)

[logfile move](#)
[logfile name](#)
[logfile open](#)

options

Performs tasks associated with Catapult options that are stored in files or within the in-memory database.

For information about setting options by using the Catapult dialog box interface, refer to “[Setting Up Catapult Default Options](#)” on page 173.

The following option commands are available:

- [*options defaults*](#)

Resets options to default settings.

- [*options exists*](#)

Checks whether an option exists in the database.

- [*options get*](#)

Returns the current value of the specified option in the in-memory design database.

- [*options load*](#)

Loads saved option settings.

- [*options save*](#)

Saves the current in-memory options.

- [*options set*](#)

Sets the value of an option in the project database.

options defaults

Resets all flow options to default settings.

Syntax

```
options defaults ?<switches>?  
  
<switches>           Valid switches: (Optional)  
      -clean            restore to predefined options
```

Arguments

- **-clean**

Removes all flow options from the project. Flows must be reloaded in order to re-establish the options in the project.

Description

The “options defaults” command resets the default option values for all flows loaded in the session. The values are reset to their original values that are predefined in the flow files. All user-specified overrides are discarded. This command does not update the flow option values saved in the [Catapult Registry](#). If old option settings were saved previously, you may want to save the options again after resetting them.

Because flow options are saved in the Catapult registry, they can become out of synch with the flow if a new version of the flow is loaded and that new version has changed its default options. Use the “-clean” switch to purge all obsolete and conflicting options, the reload the flow to re-synchronize the flow options, and finally save the new options to the registry.

Examples

Example 1:

Suppose you have modified many of the default settings across several flows, and you now want to revert the settings to the factory defaults. Use the following command:

```
options defaults
```

Example 2:

Suppose you have saved options from an older version of Catapult and you have just installed a newer version. And let’s say, for example, that the SCVerify flow in the new version has added a new option and deleted an old option. It is possible that when you enable the new SCVerify flow, the old and new options will both appear. In this situation, use the following commands to re-synchronize the flow options.

```
options defaults -clean  
flow package require /SCVerify  
flow run /SCVerify
```

Related Commands

[options exists](#)
[options get](#)
[options load](#)

[options save](#)
[options set](#)

options exists

Checks whether an option exists in the database.

Syntax

```
options exists <name>  
<name>      option name (Required)
```

Argument

- **<name>**

The name of an option in the project database. This argument has the form:

```
<section>/<option>
```

The **<section>** portion is a section name found in the Catapult Synthesis Options window or the Catapult initialization file (see “[Setting Up Catapult Default Options](#)” on page 173 and “[The Catapult Initialization File](#)” on page 224). The **<option>** portion is the target option name in that section.

In the case of hierarchical sections (“ProjectInit” and “Flows”), this argument has the form:

```
<section>/<sub-section>/<option>
```

Description

The “options exists” command reports whether or not an option exists in the database. The command returns either ‘1’ (true) or ‘0’ (false).

Example

This example checks whether for the existence of the MSIM_DEBUG option in the SCVerify flow. In this case the return value is false, indicating that the SCVerify flow has not been loaded.

```
options exists Flows/SCVerify/MSIM_DEBUG  
# 0
```

Related Commands

[application get](#)
[options defaults](#)
[options get](#)
[options load](#)

[options save](#)
[options set](#)
[project get](#)
[solution get](#)

options get

Returns the value of the specified option.

Syntax

```
options get <name> ?<switches>?

<name>                                option name (Required)
<switches>                               Valid switches: (Optional)
    -all                                 all options
    -default                            default value
    -hidden ?<bool>?                  hidden options
```

Arguments

- **<name>**

The name of an option in the project database. This argument has the form:

```
<section>/<option>
```

The `<section>` portion is a section name found in the Catapult Synthesis Options window or the Catapult initialization file (see “[Setting Up Catapult Default Options](#)” on page 173 and “[The Catapult Initialization File](#)” on page 224). The `<option>` portion is the target option name in that section.

In the case of hierarchical sections (“ProjectInit” and “Flows”), this argument has the form:

```
<section>/<sub-section>/<option>
```

- **-all**

Performs a recursive search and returns a list of all option names in the specified section and its sub-sections. Option values are not returned.

- **-default**

Returns the factory default value of the specified option.

- **-hidden ?<bool>?**

This switch includes hidden option names in the return value. Option values are not returned. The `<bool>` argument is true by default. Set `<bool>` to false to explicitly exclude hidden options from the return value.

Description

The “options get” command returns the value of an option in the in-memory database. If the `<name>` argument is omitted, the command returns a list of valid section names. If only a section name is specified, a list of valid option names is returned.

Examples

Example 1

In this example the `<name>` argument is omitted and the command returns the list of valid section names that can be used.

Commands
options get

```
options get

# General Message ProjectInit Input Output TextEditor ScheduleViewer
SchematicViewer Flows
```

Example 2:

This example provides only a section name in order to get a list of valid option names in that section.

```
options get Output

# OutputVHDL OutputVerilog SplitExpressions OutputSystemC SystemCUseBigint
SystemCUseLogicForClk RTLSChem Architecture
```

Example 3:

This example illustrates the use of the “-default” switch. The first command returns the current value of the “OutputVHDL” option. The second command returns the factory default value.

```
options get Output/OutputVHDL
# false

options get Output/OutputVHDL -default
# true
```

Example 4:

The three commands in this example demonstrate the difference between the “-all” switch and the “-hidden” switch. The first command is the default case in which neither switch is used. It simply returns the visible set of sub-section names under the “ProjectInit” section.

```
options get ProjectInit
# ProjectNamePrefix SolutionNamePrefix VersionControl ComponentLibs
Interface Architectural Hardware
```

In the second command the “-hidden” switch exposes all of the sub-section names.

```
options get ProjectInit -hidden
# ProjectNamePrefix DefaultProjectName SolutionNamePrefix VersionControl
ComponentLibs Interface Architectural Hardware
```

Finally, the “-all” switch in the third command returns every sub-section as well as every option name under each sub-section. Each value in the list is a complete hierarchical path.

```
options get ProjectInit -all

# ProjectNamePrefix DefaultProjectName SolutionNamePrefix VersionControl
VersionControl/Enabled VersionControl/Verbose
VersionControl/QueryOverwrite VersionControl/ExcludeCompilerIncludes
VersionControl/ExcludePaths ComponentLibs ComponentLibs/SearchPath
ComponentLibs/DefaultSynthesisTool ComponentLibs/CacheDir
ComponentLibs/DefaultLibs Interface Interface/DefaultClockName
Interface/CLOCK_NAME Interface/DefaultClockEdge Interface/CLOCK_EDGE
Interface/DefaultClockPeriod Interface/CLOCK_PERIOD
Interface/DefaultClockOverhead Interface/CLOCK_OVERHEAD
Interface/DefaultResetName Interface/RESET_NAME
Interface/DefaultSyncResetName Interface/RESET_SYNC_NAME
Interface/DefaultAsyncResetName Interface/RESET_ASYNC_NAME
Interface/DefaultResetKind Interface/RESET_KIND
```

```
Interface/DefaultResetActive Interface/RESET_ACTIVE
Interface/DefaultSyncResetActive Interface/RESET_SYNC_ACTIVE
Interface/DefaultAsyncResetActive Interface/RESET_ASYNC_ACTIVE
Interface/DefaultResetClearsAllRegs Interface/RESET_CLEAR_ALL_REGS
Interface/DefaultEnableName Interface/ENABLE_NAME
Interface/DefaultEnableActive Interface/ENABLE_ACTIVE
Interface/DefaultStartFlag Interface/START_FLAG Interface/DefaultDoneFlag
Interface/DONE_FLAG Interface/DefaultTransactionDoneSignal
Interface/TRANSACTION_DONE_SIGNAL Architectural
Architectural/DefaultMemMapThreshold Architectural/MEM_MAP_THRESHOLD
Architectural/DefaultRegisterThreshold Architectural/REGISTER_THRESHOLD
Architectural/DefaultLoopMerging Architectural/MERGEABLE
Architectural/DefaultSpeculativeExec Architectural/SPECULATE
Architectural/DefaultResetClearsAllRegs Architectural/OldSchedule
Architectural/OLD_SCHED Architectural/DesignGoal
Architectural/DESIGN_GOAL Hardware Hardware/DefaultClockPeriod
Hardware/ClockPeriod Hardware/DefaultClockOverhead Hardware/ClockOverhead
Hardware/IOMode Hardware/Unroll Hardware/DefaultRealloc Hardware/REALLOC
Hardware/DefaultMuxpath Hardware/MUXPATH Hardware/DefaultTimingChecks
Hardware/TIMING_CHECKS Hardware/DefaultAssignOverhead
Hardware/DefaultRegisterSharingLimit Hardware/REGISTER_SHARING_LIMIT
Hardware/ASSIGN_OVERHEAD Hardware/RESET_CLEAR_ALL_REGS
Hardware/DefaultSafeFSM Hardware/SAFE_FSM Hardware/DefaultNoXAssignments
Hardware/NO_X_ASSIGNMENTS Hardware/DefaultRegMaxFanOut
Hardware/REG_MAX_FANOUT Hardware/DefaultSafeMux
```

Related Commands

[application get](#)
[options defaults](#)
[options exists](#)
[options load](#)

[options save](#)
[options set](#)
[project get](#)
[solution get](#)

options load

Loads saved option settings.

Syntax

```
options load ?<switches>?

<switches>           Valid switches: (Optional)
    -file <string>      name of file to load
    -registry <string>   registry location to load
    -quiet               suppress messages
```

Arguments

- **-file <string>**

A pathname to an options file. Use this switch to load option settings from the specified file instead of the default location. If only the leaf name is specified, Catapult looks for the file in the current working directory.

- **-registry <string>**

(For Windows operating systems only) A Windows registry path where the Catapult option settings are stored. The standard location for Catapult options is as follows, where <version> is the Catapult version number.

```
{HKEY_CURRENT_USER\Software\Mentor Graphics\Catapult Synthesis\<version>}
```

- **-quiet**

Suppress informational messages returned by the command. Does not suppress error or warning messages.

Description

The “options load” command loads initialization options from the default location unless the “-file” or “-registry” switch is used to specify an alternate location. The default location is determined during invocation of the Catapult session. Catapult searches for an initialization file in the following search order:

1. A catapult.ini file in the current working directory
2. A catapult.ini file in the user’s HOME directory
3. The Catapult registry (see “[Catapult Registry](#)” on page 224)

The first one found becomes the default location for the duration of the catapult session, or until a different location is configured by using “options save” command with the “-default” switch.

Examples

Example 1:

This example simply loads option settings from the default location, which is the Catapult registry in this case.

```
options load
# Loading options from '$HOME/.catapult/2007a.ixl.reg'.
```

Example 2:

This example loads option settings from a file named “opt_settings.ini” in the current working directory.

```
options load -file opt_settings.ini
```

Example 3:

This example is the same as above, but specifies the full path to the file.

```
options load -file /jdoe/opt_settings.ini
```

Related Commands

[options defaults](#)
[options exists](#)
[options get](#)

[options save](#)
[options set](#)

options save

Save the in-memory options to the Catapult registry or an alternate file.

Syntax

```
options save ?<switches>?

<switches>           Valid switches: (Optional)
    -default <load message> <save message>
                    configures the default load and save location
    -file <string>          name of file to save
    -registry <string>       name of registry key
    -section <string>        starting section name
    -hidden ?<bool>?       option is hidden
    -quiet                  suppress messages
```

Arguments

- **-default <load message> <save message>**

Configures default settings for the “options load” and “options save” commands. It sets the default location where Catapult option settings are stored, and sets the message strings that are displayed by the “options load” and “options save” commands. The location setting is specified by either the “-file” or “-registry” switches, one of which must be used in conjunction with this switch.

When this switch is used, the “options save” command does not save options settings. It only configures the default settings. The new settings remain in effect for the duration of the current Catapult session. New Catapult sessions are always initialized to the system default settings.

- **-file <string>**

A pathname to a file in which the option settings will be saved. Use this switch to save option settings to a file other than the default location. If only the leaf name of the file is specified, then the file is saved to the current working directory. If the file exists, it is overwritten. Otherwise, the file is created.

Use the filename “catapult.ini” in either the project directory or in the \$HOME directory in order to have Catapult automatically load the options at during invocation. Refer to “[Restoring Options](#)” on page 223 for more information.

- **-registry <string>**

(For Windows operating systems only) A Windows registry path where the Catapult option settings will be stored. A hierarchy of keys and values is created below the specified location in the registry. The standard location for Catapult options is as follows, where <version> is the Catapult version number.

```
{HKEY_CURRENT_USER\Software\Mentor Graphics\Catapult Synthesis\<version>}
```

- **-section <string>**

Saves only those options stored in the specified section of the file or registry. Section names correspond to the names in the Catapult Synthesis Options window. This switch must be used in conjunction with the “-file” or “-registry” switch. When saving to existing file, the entire contents of the file are overwritten and only the specified section is saved. Saving to the Windows registry overwrites only the specified section.

- **-hidden ?<bool>?**

This switch includes hidden options in the save operation. The <bool> argument is true by default. Set <bool> to false to explicitly exclude hidden options.

- **-quiet**

Suppress informational messages returned by the command. Does not suppress error or warning messages.

Description

This command saves the in-memory option settings to default location, unless the “-file” or “-registry” switch is used to specify an alternate location. The system default location is determined during invocation of the Catapult session. Refer to the section “[Restoring Options](#)” on page 223 for more information.

When “-file” or “-registry” is used in conjunction with the “-default” switch, the specified path becomes the default location used by the “options load” and “options save” commands. In addition, the default return string can be changed by using the “-default” switch. Refer to the “-default” switch above for more details.

 **Note**
Enable the “General/SaveSettings” option to have Catapult automatically save the options database when exiting.

Examples

Example 1:

This example saves the options to a Catapult initialization file in the user’s HOME directory.

```
options save -file /user/johnd/catapult.ini
```

Example 2:

This example uses the “-default” switch to configure the default location and messages for the “options load” and “options save” commands.

```
options save -default "Loading options from file: opt_settings.ini"  
"Saving options to file: opt_settings.ini" -file "/jdoe/opt_settings.ini"
```

As a result, the “options load” and “options save” commands now read/write the options in the file “/jdoe/opt_settings.ini” by default. And the commands return the specified message strings as demonstrated in the following command calls:

```
options load
```

Commands

options save

```
# Loading options from file: opt_settings.ini  
options save  
# Saving options to file: opt_settings.ini
```

Related Commands

[options defaults](#)
[options exists](#)
[options get](#)

[options load](#)
[options set](#)

options set

Sets the value of an option in the project database.

Syntax

```
options set <name> <value> ?<switches>?  
<name>          option name (Required)  
<value>          option value (Required)
```

Arguments

- **<name>**

The name of an option in the project database. This argument has the form:

<section>/<option>

or

<section> <option>

The <section> portion is a section name found in the Catapult Synthesis Options window or the Catapult initialization file (see “[Setting Up Catapult Default Options](#)” on page 173 and “[The Catapult Initialization File](#)” on page 224). The <option> portion is the target option name in that section.

In the case of hierarchical sections (“ProjectInit” and “Flows”), this argument has the form:

<section>/<sub-section>/<option>

or

<section>/<sub-section> <option>

Use the “[options get](#)” command to get a list of option names in the database.

- **<value>**

The new value(s) to be assigned to the option. List values and string values containing spaces must be enclosed in double quotes or braces.

Description

The “options set” command modifies the value of an option in the project database. If the command succeeds, the new value is returned as a comment. Otherwise, an error message is returned. The modified values are valid for the duration of the Catapult session. To preserve the values for future Catapult sessions, use the “[options save](#)” command. For information about setting options from the Catapult GUI, refer to “[Setting Up Catapult Default Options](#)” on page 173.

Examples

Example 1:

The following examples show the different forms of the <name> argument.

```
options set Interface/DefaultResetKind async
```

```
# async

options set Interface DefaultResetKind async
# async

options set ProjectInit/Hardware/DefaultSafeFSM true
# true

options set ProjectInit/Hardware DefaultSafeFSM true
# true
```

Example 2:

This example shows how to specify a list of values. The first string in the list is enclosed in double quotes because it contains a space. The entire list is enclosed in braces.

```
options set Input/LibPaths {"/Calypto Design Systems/Catapult/libs"
                           /catapult/projlibs}

# "/Calypto Design Systems/Catapult/libs" /catapult/projlibs
```

Related Commands

[options defaults](#)
[options exists](#)
[options get](#)

[options load](#)
[options save](#)
[project set](#)

project

Performs tasks relating to Catapult projects.

The following project commands are available:

- [**project close**](#)
Closes the current project without exiting Catapult.
- [**project get**](#)
Gets database information about the current project.
- [**project load**](#)
Loads a previously saved project.
- [**project new**](#)
Creates a new project.
- [**project report**](#)
Generates a report on the active project.
- [**project save**](#)
Saves the open project to the current working directory.
- [**project set**](#)
Sets the names and directories to be associated with the project.

project close

Closes the current project without exiting the Catapult session.

Syntax

```
project close
```

Arguments

None.

Description

The “project close” command closes the current project and keeps the Catapult session open.

Example

```
project close
```

Related Commands

[project get](#)
[project load](#)
[project new](#)

[project report](#)
[project save](#)
[project set](#)

project get

Gets information from the Catapult project database.

Syntax

```
project get ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -recurse <string>   Everything under
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
        value      just matching values
        path       just matching paths
        pathvalue  path and value combination for array set
        leaf       just matching leaves
        leafvalue  leaf and value combination for array set
        advanced   hierarchical list structure
        none      no return value
    -checkpath <bool>  Error on path not found
    -match <exact|glob> Path match type
        exact     exact paths only
        glob      glob paths
    -info <bool>   Return object info
<subpaths>        Database subpaths (Optional)
```

Arguments

- **<path>**

Path into the Catapult SIF database. For detailed information about specifying database paths, refer to “[Command Interface to the SIF Database](#)” on page 352.

- **<switches>**

Use these switches to control how the database is searched and how the returned data is displayed. These switches are common to all commands that take database path arguments. For detailed descriptions of the switches, refer to the section, “[Common Command Switches](#)” on page 356.

- **<subpaths> s**

Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

This command queries the project database. Refer to “[Command Interface to the SIF Database](#)” on page 352 for general information about the structure of the SIF database.

Examples

Example 1:

This example returns the top-level data items in the project database. The “-match glob” switch is required in order to evaluate and expand the “*” wildcard. The “-return pathvalue” switch causes the command to return both the database path that was queried and the value stored at that path.

```
project get /* -match glob -return pathvalue
```

Line breaks have been added to the return value below for readability. Each line contains a database key and its corresponding value. Key values that are empty braces indicate that the key is a hierarchical node in the database. To access multiple levels of the hierarchy, use the “-reurse” switch.

```
# /LOGFILE {C:\Catapult\fir_filter\catapult.log}
/PROJECT_DIR C:\\\\Catapult\\\\fir_filter\\\\Catapult_2\\\\
/SIF_DIR C:\\\\Catapult\\\\fir_filter\\\\Catapult_2\\\\SIF\\\\
/CURRENT_STATE extract
/STATES {new analyze techlibs compile loops memories architect allocate
schedule dpfsm instance extract}
/MODIFIED 1
/name Catapult
/CURRENT {}
/SOLUTION {}
/SID {}
/MESSAGES {}
/XREFS {}
/FLOWPKGS {}
/DIRECTIVES {}
/name Catapult
```

Example 2:

This example gets the names of all directives in the project. The “-return” switch in this case is set to “value” so that only the database values are returned and not the paths to the keys.

```
project get /DIRECTIVES/*/name -match glob -return value
# ARRAY_SIZE ASSIGN_OVERHEAD AREA_GOAL BASE_ADDR BASE_BIT ...
```

Example 3:

This example returns the basename and the version suffix of the currently active solution.

```
project get /CURRENT -return value -- name VERSION
# v3 fir_filter
```

Related Commands

[project get](#)
[project load](#)
[project new](#)

[project report](#)
[project save](#)
[project set](#)

project load

Loads a previously saved project.

Syntax

```
project load <filepath>  
<filepath>           Relative or absolute file path (Required)
```

Arguments

- **<filepath>**

The pathname to a saved Catapult project file (.ccs). If only the leaf name of the file is specified, then Catapult looks in the current working directory for the project file.

Description

The “project load” command opens a saved project. The design transformation state is re-established at the point at which the project was saved. If a project is currently open, it is closed before loading the new project.

Example

```
project load /filter_6tap/filter_6tap.ccs
```

Related Commands

[project close](#)
[project new](#)
[project report](#)

[project save](#)
[project set](#)

project new

Creates a new project.

Syntax

```
project new ?<switches>?  
  
<switches>           Valid switches: (Optional)  
    -name <string>      Specifies project name  
    -directory <string>   Specifies project directory to create  
    -description <string> Description
```

Arguments

- **-name <string>**

Specifies the name of the new project. If omitted, the default name has the form “Catapult_<nnn>” where <nnn> is a number that is incremented to make the filename unique.

- **-directory <string>**

A pathname specifying the file system location where Catapult will create the project directory. The specified location cannot be an existing directory. If this option is not used, the new project is created in the current working directory.

- **-description <string>**

Adds the specified string to the new project. The string is stored in the project database at the /DESCRIPTION node. To query for a project’s description, use the command:

```
project get -DESCRIPTION
```

Description

The “project new” command closes the current project and creates a new project.

Examples

Example 1:

Creates a new project named “atm_filter” in the current directory working directory:

```
project new -name atm_filter
```

Example 2:

Creates a new project named “atm_filter” in the directory “/my_catapult_projects/”:

```
project new -dir /my_catapult_projects/atm_filter
```

Related Commands

[project close](#)
[project get](#)
[project report](#)

[project save](#)
[project set](#)

project report

Generates a report on the current project.

Syntax

```
project report
```

<code><options></code>	Report options (Optional)
<code>-filename <string></code>	Write report to file
<code>-transcript <bool></code>	Send report to transcript
<code>-window <bool></code>	View report in window
<code>-all</code>	Include all reports
<code>-area <bool></code>	Include Area in report
<code>-bom <bool></code>	Include Bom in report
<code>-clock <bool></code>	Include Clock in report
<code>-filters <bool></code>	Include Filters in report
<code>-header <bool></code>	Include Header in report
<code>-iorange <bool></code>	Include Iorange in report
<code>-libs <bool></code>	Include Libs in report
<code>-license <bool></code>	Include License in report
<code>-loops <bool></code>	Include Loops in report
<code>-memories <bool></code>	Include Memories in report
<code>-multicycle <bool></code>	Include Multicycle in report
<code>-profiles <bool></code>	Include Profiles in report
<code>-registers <bool></code>	Include Registers in report
<code>-summary <bool></code>	Include Summary in report
<code>-timing <bool></code>	Include Timing in report

Arguments

- **`-filename <string>`**

Saves the report to the specified file in addition to displaying it in the GUI. The `<string>` argument specifies pathname of the output file. If only the leaf name is specified, then Catapult creates the file in the current working directory. If a file by that name already exists, the command does not generate a report.

- **`-transcript <bool>`**

Enable this switch to send the report to the Catapult session transcript. By default the report is displayed in a text window. Enable the “-window” switch in addition to this switch in order to send the report to both destinations.

- **`-window <bool>`**

Enable/disable this switch to control whether or not Catapult displays the report in a text window in the GUI. By default the report is displayed in a window unless the “-transcript” switch is enabled. You may want to disable the “-window” switch when using the “-file” switch.

- **`-all`**

Include all categories of the project data in the report. Individual “category” switches can be used in conjunction with this switch to selectively disable particular categories of data.

- ***Data categories***

All remaining switches listed in the “Syntax” section allow you to selectively include or exclude particular types of data in the report. By default, all categories are disabled.

Description

This command generates a report containing information about the current project and all of its solutions. If Catapult is running in GUI mode, the report is displayed in a DesignPad editor window. In non-GUI mode, the report is sent to the session transcript.

Related Commands

[project close](#)
[project get](#)
[project new](#)

[project save](#)
[project set](#)

project save

Saves the current project to a file.

Syntax

```
project save ?<filepath>?  
    <filepath>          Relative or absolute file path (Optional)
```

Arguments

- **<filepath>**

The path to a Catapult project file (*.ccs). If only the leaf name is specified, then the file is saved in the current working directory. If <filepath> is omitted, the default path is “<project_dir>/<project_name>.ccs”.

Description

The “project save” command writes the current project to a Catapult project file (*.ccs). The return value displays the full path to the saved file.

Example

```
project save  
# Saving project file 'C:/Catapult/fir_filter/Catapult_3.ccs'.
```

Related Commands

[project close](#)
[project get](#)
[project new](#)

[project report](#)
[project set](#)

project set

Sets the project name displayed in the GUI and/or create a new project directory.

Syntax

```
project set ?<switches>?  
  
<switches>           Valid switches: (Optional)  
    -incr_directory <string>  Set the project directory generating a  
                                unique name  
    -directory <string>       Set the project directory  
    -name <string>           Set the name of this project
```

Arguments

- **-incr_directory <string>**

Creates the specified directory in the file system to hold the Catapult project files. The *<string>* argument can be either a path or a leaf name. If only a leaf name is given, the directory is created in the current working directory. If the directory already exists, the new directory name is made unique as follows:

- If *<string>* contains the substring “_<n>”, where *<n>* is an integer value, then *<n>* is incremented sequentially until the name is unique within the directory. Note that “_<n>” can appear anywhere in the string.
- If *<string>* does not contain the substring “_<n>”, then “_<n>” is appended to *<string>* and incremented sequentially until the name is unique within the directory.

- **-directory <string>**

Creates the specified directory in the file system to hold the Catapult project files. The *<string>* argument can be either a path or a leaf name. If only a leaf name is given, the directory is created in the current working directory. Specifying an existing directory results in an error.

- **-name <string>**

Changes the project name displayed in the GUI to *<string>*. The name of the project directory in the file system is not changed.

Description

Use this command to set the project name displayed in the GUI and/or create a new project directory in the file system. The “-name” option simply changes the name of the current project displayed in the GUI. It does not change the default the system default name. To change the default project name, use the “options set ProjectInit ProjectNamePrefix” command.

The “-directory” and “-incr_directory” options immediately create a new project directory in the file system, and update the project name displayed in the GUI.

Note

 When this command is used to create a project directory, it can only be used if the current project has not yet been written to disk. That means that the current project must be either the default project created at invocation or a new project created by the “[project new](#)” command and is still in the “New” state. For more information about project states, refer to [“The Catapult Work Flow”](#) on page 46.

Example

This example creates a new project directory named “my_proj_dir_1” in the current working directory.

```
project set -incr_dir my_proj_dir_1
# Creating project directory 'C:\Catapult\fir_filter\my_proj_dir_1'.
```

Related Commands

[project close](#)
[project get](#)
[project new](#)

[project report](#)
[project save](#)

quit

Terminate operations and close the Catapult session.

Syntax

```
quit ?<switches>?  
-force    -- quit without saving changes  
-code     -- specifies the application return code
```

Arguments

- **-force**

Forces Catapult to terminate operations and exit immediately without saving changes.

- **-code <integer>**

Returns the specified integer to the shell as an exit status. If omitted, a zero is returned.

Description

Catapult will quit the application only if the project has been saved. Use the “-force” switch to terminate operations and close without saving changes. The “-code” option returns an integer status code to the shell.

Example

```
quit -force -code 2
```

Related Commands

None.

resource

Resource objects such as inputs, outputs and registers are tracked in Catapult and can have constraints applied to them.

The following resource command is available.

- *resource add*

Add new resource objects to the resources list. Once a new resource is added, you can set constraints on it within Catapult.

- *resource remove*

Removes resource objects from the design.

resource add

Adds resource objects to a process in the design.

Syntax

```
resource add ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                    Return data format
        value      just matching values
        path       just matching paths
        pathvalue   path and value combination for array set
        leaf        just matching leaves
        leafvalue   leaf and value combination for array set
        advanced    hierarchical list structure
        none       no return value
    -checkpath <bool>  Error on path not found
    -match <exact|glob> Path match type
        exact      exact paths only
        glob       glob paths
    -info <bool>       Return object info
    -solution <string> Specify solution
    -state <new|analyze|compile|architect|allocate|
                    schedule|dpfsm|extract|last>
                    Pick design state, default is current
        new        Design state 'new'
        analyze   Design state 'analyze'
        compile   Design state 'compile'
        architect Design state 'architect'
        allocate  Design state 'allocate'
        schedule  Design state 'schedule'
        dpfsm     Design state 'dpfsm'
        extract   Design state 'extract'
        last      Last valid design state for specified solution
    <subpaths>     Database subpath and value combinations (Optional)
```

Arguments

- **<path>**

Database path to a resource. The format is:

```
<object_path>/<resource_name>
```

The <object_path> portion specifies the path to a design process for which the resource is created. The path has the form of a HDL design path. The root of the path is the top-level of the design. The <resource_name> portion is a user-defined string. For example:

```
/dct/core/a_new_resource
```

- **-solution <string>**

Use this switch to access, and branch from, a solution other than the active solution.

- **-state <string>**

A new solution is branched from the specified state of the current solution or the solution specified by the “-solution” switch. The new solution is set to the specified state and the resource is added to the new solution. For more information about Catapult states, refer to in the section “[The Catapult Work Flow](#)” on page 46.

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section “[Common Command Switches](#)” on page 356.

- **<subpaths>**

One or more arguments specifying database sub-paths of resources and their values. Sub-paths are relative to the *<path>* argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

This command is used to add resources to processes in the design. It creates the resource in the SIF database and initializes the resource constraints to their default values. Adding a resource means that Catapult is free to use the resource and that you can add constraints on it within the Catapult graphical user interface. You can also use the “[directive get](#)” and “[directive set](#)” commands to access and modify constraints on the resources.

 **Note** During the “replay” that's used to re-apply constraints in a new solution, Catapult will apply all of the “resource add” constraints before any of the directives.

Examples

Example 1:

This example adds the resource “temp_rsc” to the “/fir_filter/core” sub-process.

```
resource add /fir_filter/core/temp_rsc -return pathvalue
# /fir_filter/core/temp_rsc {}
```

The “[directive get](#)” command can be used as follows to see the initial values of the resource constraints.

```
directive get /fir_filter/core/temp_rsc/* -match glob -return pathvalue

# /fir_filter/core/temp_rsc/BLOCK_SIZE 0
/fir_filter/core/temp_rsc/EXTERNAL_MEMORY false
/fir_filter/core/temp_rsc/FIFO_DEPTH -1
/fir_filter/core/temp_rsc/GEN_EXTERNAL_ENABLE false
/fir_filter/core/temp_rsc/INPUT_DELAY 0.000000
/fir_filter/core/temp_rsc/INTERLEAVE 1
/fir_filter/core/temp_rsc/MAP_TO_MODULE {}
```

Commands

resource add

```
/fir_filter/core/temp_rsc/OUTPUT_DELAY 0.000000
/fir_filter/core/temp_rsc/PACKING_MODE absolute
/fir_filter/core/temp_rsc/ALLOW_STAGE_STREAMING true
/fir_filter/core/temp_rsc/STAGE_REPLICATION 0
```

Example 2:

This example adds two resources to the top-level process. The resource names are specified as sub-path arguments at the end of the command line.

```
resource add /fir_filter -return path res_1 res_2
# /fir_filter/res_1 /fir_filter/res_2
```

Related Commands

[cycle add](#)
[directive get](#)

[directive set](#)
[resource remove](#)

resource remove

Removes resource objects from the design.

Syntax

```
resource remove ?<path>? ?<switches>? ?<subpaths>?

<path>           Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --           End <switches> parsing
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                    Return data format
        value      just matching values
        path       just matching paths
        pathvalue   path and value combination for array set
        leaf        just matching leaves
        leafvalue   leaf and value combination for array set
        advanced    hierarchical list structure
        none       no return value
    -checkpath <bool>  Error on path not found
    -match <exact|glob> Path match type
        exact      exact paths only
        glob       glob paths
    -info <bool>     Return object info
    -solution <string> Specify solution
    -state <new|analyze|compile|architect|allocate|
                    schedule|dpfsm|extract|last>
                    Pick design state, default is current
        new        Design state 'new'
        analyze   Design state 'analyze'
        compile   Design state 'compile'
        architect Design state 'architect'
        allocate  Design state 'allocate'
        schedule  Design state 'schedule'
        dpfsm     Design state 'dpfsm'
        extract   Design state 'extract'
        last      Last valid design state for specified solution
    <subpaths>   Database subpaths (Optional)
```

Arguments

- **<path>**

Database path to a resource. The format is:

```
<object_path>/<resource_name>
```

The <object_path> portion specifies the path to a design process containing the resource.

The path has the form of a HDL design path. The root of the path is the top-level of the design. For example:

```
/dct/core/a_new_resource
```

- **-solution <string>**

Use this switch to access, and branch from, a solution other than the active solution.

- ***-state <string>***

A new solution is branched from the specified state of the current solution or the solution specified by the “-solution” switch. The new solution is set to the specified state and the resource is removed from the new solution. For more information about Catapult states, refer to in the section “[The Catapult Work Flow](#)” on page 46.

- ***Common Switches***

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section “[Common Command Switches](#)” on page 356.

- ***<subpaths>***

One or more arguments specifying database sub-paths to resources. Sub-paths are relative to the *<path>* argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

This command deletes the specified resource(s) from the design database.

Example

This example removes two resources from the top-level process. The resource names are specified as sub-path arguments at the end of the command line.

```
resource remove /fir_filter -return path res_1 res_2  
# /fir_filter/res_1 /fir_filter/res_2
```

Related Commands

[cycle remove](#)
[directive get](#)

[directive remove](#)
[resource add](#)

set_working_dir

Sets the working directory to the specified pathname.

Syntax

```
set_working_dir <path>
```

Arguments

- **<path>**

Specifies a file system path to an existing directory.

Description

The `set_working_dir` command makes the specified directory the current working directory of the Catapult session. The working directory is where all output sub-directories and files are written.

Example

```
set_working_dir "E:/designs/filter_6tap"
```

Related Commands

None.

solution

Performs various tasks on the active solution.

The following `solution` commands are available:

- ***solution get***
Gets database information about a solution.
- ***solution file add***
Adds an input or output file to the solution.
- ***solution file remove***
Removes input or output files.
- ***solution file restore***
Restores input file(s) from the Version Control System.
- ***solution file set***
Sets options on an input or output file.
- ***solution new***
Creates a new solution branch from the specified solution.
- ***solution remove***
Removes a solution from the project.
- ***solution rename***
Renames the active solution.
- ***solution report***
Generates a report for the active solution.
- ***solution restore***
Restores input files from the Version Control System.
- ***solution select***
Use the command to select a solution.
- ***solution timing***
Generate timing reports for critical paths and user-specified data paths in the design.

solution get

Gets information about the files, directories, design data and general information associated with a solution.

Syntax

```
solution get ?<path>? ?<switches>? ?<subpaths>?

<path>          Hierarchical database path (Optional)
<switches>       Valid switches: (Optional)
    --
    --           End <switches> parsing
    -recurse <string>   Everything under
    -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
        value      just matching values
        path       just matching paths
        pathvalue  path and value combination for array set
        leaf       just matching leaves
        leafvalue  leaf and value combination for array set
        advanced   hierarchical list structure
        none      no return value
    -checkpath <bool>  Error on path not found
    -match <exact|glob> Path match type
        exact     exact paths only
        glob      glob paths
    -info <bool>      Return object info
    -solution <string> Specify solution
    -state <new|analyze|compile|architect|allocate|
                        schedule|dpfsm|extract|last>
                        Pick design state, default is current
        initial   Initial state of new project
        new      Design state 'new'
        analyze  Design state 'analyze'
        compile  Design state 'compile'
        architect Design state 'architect'
        allocate Design state 'allocate'
        schedule Design state 'schedule'
        dpfsm    Design state 'dpfsm'
        extract  Design state 'extract'
        last     Last valid design state for specified solution
    <subpaths>        Database subpaths (Optional)
```

Arguments

- **<path>s**

A hierarchical path into the SIF database. The root node of <path> is /PROJECT/CURRENT when accessing the active solution. When accessing other solutions (“-solution” switch), the root node is /PROJECT/SOLUTION/<solution_name>. For more information, refer to “[Command Interface to the SIF Database](#)” on page 352.

- **-solution <string>**

Use this switch to access settings in the specified solution. If omitted, the active solution is queried.

- ***-state <string>***

Use this switch to access settings in a state other than the current state of the solution. For more information about Catapult states, refer to in the section “[The Catapult Work Flow](#)” on page 46.

- ***Common Switches***

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section “[Common Command Switches](#)” on page 356.

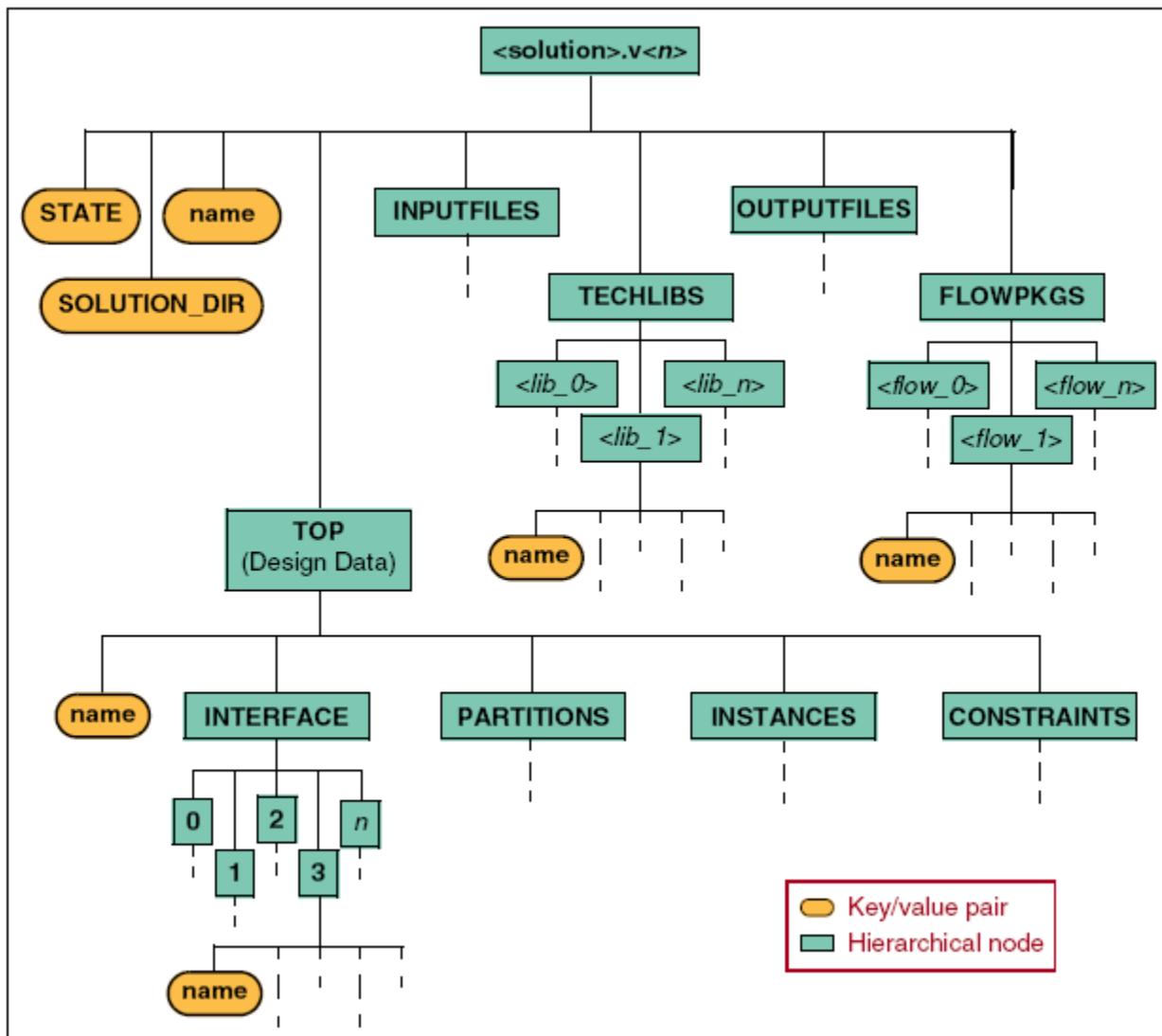
- ***<subpaths>***

Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the *<path>* argument. For detailed information about specifying database sub-paths, refer to “[Using Sub-Path Arguments](#)” on page 355.

Description

This command queries the Catapult SIF database for information about the specified solution. Data for each solution in the project is stored in a separate sub-tree in the database. All solutions in the database have the same general hierarchical structure. Figure 8-6 is a simplified representation of a solution sub-tree.

The actual set of nodes in any particular solution is unique because of the dynamic nature of the SIF database. When a solution is first created, the database is populated with the minimum set of nodes to represent the solution. As you work on a design, Catapult adds/removes nodes accordingly. Most of the data nodes, such as INPUTFILES, OUTPUTFILES, INTERFACES, and so on, contain lists of items. Each item in the list is a separate child node numbered sequentially starting at zero. The actual name of the item in the list is found in the “name” field below the child node. In the cases of TECHLIBS and FLOWPKGS, the children take the name of the library or flow being added.

Figure 8-6. Nodes of Interest in the Solution Database Hierarchy**Examples:****Example 1:**

This example simply gets the directory path for the currently active solution. The **SOLUTION_DIR** key is at the top-level of the solution sub-tree.

```
solution get /SOLUTION_DIR
# C:\Catapult\fir_filter\Catapult_7\fir_filter.v1
```

Example 2:

This example gets the names of all design interfaces. The asterisk (*) wildcard is used to match all nodes under the **INTERFACE** node in the solution sub-tree.

```
solution get /TOP/INTERFACE/*/name -match glob
# input:rsc.d coeffs:rsc.d output:rsc.d clk rst
```

To get the database path for each interface name, include the “-return pathvalue” switch. (Note: The return value is a single string. Line breaks have been added in the example below to improve readability.)

```
solution get /TOP/INTERFACE/*/name -match glob -return pathvalue

# /TOP/INTERFACE/0/name input:rsc.d
/TOP/INTERFACE/1/name coeffs:rsc.d
/TOP/INTERFACE/2/name output:rsc.d
/TOP/INTERFACE/3/name clk
/TOP/INTERFACE/4/name rst
```

Example 3:

This example explores the hierarchy below an INTERFACE node.

```
solution get /TOP/INTERFACE/2/* -match glob -return pathvalue

# /TOP/INTERFACE/2/DATAMODE OUT
/TOP/INTERFACE/2/WIDTH 8
/TOP/INTERFACE/2/HPATH
/fir_filter/output:rsc.d
/TOP/INTERFACE/2/DATATYPE {}
/TOP/INTERFACE/2/name output:rsc.d
/TOP/INTERFACE/2/LOC_INFO 331
/TOP/INTERFACE/2/PROPERTIES {}
```

The return value shows that “DATAMODE,” “name” and “LOC_INFO” are key/value pairs, and that “DATATYPE” and “PROPERTIES” are hierarchical nodes. The hierarchy of the DATATYPE node is as follows:

```
solution get /TOP/INTERFACE/2/DATATYPE/* -match glob -return pathvalue

# /TOP/INTERFACE/2/DATATYPE/FORMAT UNSIGNEDBITVECTOR
/TOP/INTERFACE/2/DATATYPE/LENGTH 8
/TOP/INTERFACE/2/DATATYPE/INDEX {}
```

Example 4:

This example returns the names and datatypes of all interfaces. The ellipsis wildcard (...) is used to recurse down all sub-trees under the INTERFACE node.

```
solution get /TOP/INTERFACE/.../name -match glob -return pathvalue

# /TOP/INTERFACE/0/name input:rsc.d
/TOP/INTERFACE/0/PROPERTIES/0/name DEPRECATED
/TOP/INTERFACE/0/PROPERTIES/1/name RESOURCE
/TOP/INTERFACE/1/name coeffs:rsc.d
/TOP/INTERFACE/1/PROPERTIES/0/name DEPRECATED
/TOP/INTERFACE/1/PROPERTIES/1/name RESOURCE
/TOP/INTERFACE/2/name output:rsc.d
/TOP/INTERFACE/2/PROPERTIES/0/name DEPRECATED
/TOP/INTERFACE/2/PROPERTIES/1/name RESOURCE
/TOP/INTERFACE/3/name clk
/TOP/INTERFACE/4/name rst
```

Related Commands

[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution file add

Add an input or output file to the solution.

Syntax

```
solution file add <filepath> ?<switches>?

<filepath>           Relative or absolute file path (Required)
<switches>          Valid switches: (Optional)
    -solution <string>      Specify solution
    -type <string>          Specify file type
    -input                  File added to input file list
    -output                 File added to output file list
    -folder <string>        Add file to specified folder dbpath
    -exclude <bool>         Exclude from C synthesis
    -description <string>   Description of file
    -dependencies <string>  List of file dependencies
    -flow <Flow Description> <Flow Command>
                            Flow that can be run on this file (Multiple)
        <Flow Description>  Flow Description value (Required)
        <Flow Command>     Flow Command value (Required)
    -flow_data <Flow Name> <Flow Data>
                            Flow data for this file (Multiple)
        <Flow Name>        Flow Name value (Required)
        <Flow Data>        Flow Data value (Required)
    -property <name value> <Property>
                            Property for file (Multiple)
        <name>             name value (Required)
        <value>            value value (Required)
    -args <string>          Compilation arguments
    -preserve_path <bool>  Preserve path as it was entered into the tool.
```

Argument

- **<filepath>**

A file system path to the file being added. Specify either an absolute path or a path relative to the Catapult working directory.

- **-solution <string>**

Adds the file to the specified solution. When adding an input file the target solution must be in the “New” state. The solution must be at the “Passed Compile” state or later when adding an output file.

- **-type <string>**

Specifies the type of file being added. Table 8-11 lists the most common type names. Each file type has a corresponding icon that is displayed in the Catapult GUI.

Table 8-11. File Type Names Known to Catapult

C++	LOG	SYNTHTIMING
CHEADER	MAKEFILE	SYSTEMC
EXECUTABLE	REPORT	TCL

Table 8-11. File Type Names Known to Catapult

FOLDER	SHELL	VERILOG
HTML	SYNTHESIS	VHDL

- **-input**

Adds the file to the “Input Files” list. The target solution must be in the “New” state when adding input files. If the target solution has passed the “New” state, use the “[go new](#)” command to reset its state before adding the file. In this situation, a new solution is branched from the target solution and the file is added to the new solution.

- **-output**

Adds an output file or folder to the solution. The target solution must be at the “Passed Compile” state or further when adding output files.

- **-folder <string>**

Specifies the database path to an output folder in which the file will be added. For output files only.

- **-exclude <bool>**

When set to “true”, this switch excludes the input file from C synthesis compilation. For input files only.

- **-description <string>**

Specifies a text string that is displayed in the GUI in place of the file name.

- **-dependencies <string>**

Specifies a list of files that are dependencies of the <filepath> file.

- **-flow <Flow Description> <Flow Command>**

Specifies a flow command that operates on the file. The command is accessible from the file’s popup menu in the GUI. The “<Flow Description>” string appears as a menu item on the popup menu. The “<Flow Command>” string specifies a the database path to flow command and any arguments for the command.

- **-flow_data <Flow Name> <Flow Data>**

This option provides a mechanism to store transient flow related information in the database.

- **-property <name value> <Property>**

Specifies a database property name and value for the <filepath> node in the SIF database.

- **-args <string>**

Use this switch to specify compilation arguments for the file. The arguments are stored in the database with the file and are available to flow commands that operate on the file. For input files only.

- ***-preserve_path <bool>***

When set to true, this option preserves the path as it was entered into the tool. When false, paths are made relative to the \$PROJECT_HOME directory.

Description

This “solution file add” command can add an input file, an output file, or an output folder to the solution. The file or folder is added to the SIF database and is reflected in the “Project Files” window in the Catapult GUI. The default return value is the database path to the folder or file that was added. Use the “-return” switch (see “[-return](#)” on page 356) to control the content and format of the return value.

Examples

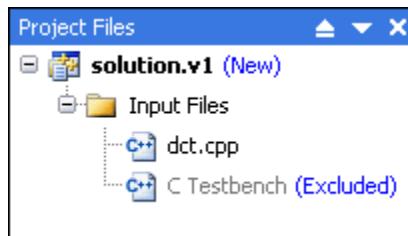
Example 1:

This example adds two input files. The first is a C++ design file and the second is a C++ testbench file.

```
solu file add dct.cpp -input
# /INPUTFILES/1
solu file add tb_dct.cpp -input -exclude true -description "C Testbench"
# /INPUTFILES/2
```

Figure 8-7 shows how the new files appear in the Catapult GUI. Note that the file types were not explicitly specified in the command, but Catapult inferred the type from file extensions. The second command uses the “-exclude” switch to prevent the testbench from being compiled during C synthesis. Also, the string specified by the “-description” switch in the second command replaces the actual testbench file name in the GUI.

Figure 8-7. Adding Input Files Example



Example 2:

The following example script demonstrates how to 1) create hierarchical output folders in the solution, 2) add an output file to a folder, and 3) assign flow commands to the file (accessed from the popup menu). Figure 8-8 shows the “Project Files” window before and after the script is executed.

```
// INITIALIZE VARIABLES
// File to be added.
set file_name precision_cmd.tcl
// Path to the active solution
set solu_dir [utility path hard [solution get /SOLUTION_DIR]]
// Captures return value (i.e. SIF database path to newly created folder)
```

```
set folder_sif_path {}

// CREATE PRECISION COMMAND FILE
// Create a TCL file containing commands for the Precision Synthesis tool
set fh [open $solu_dir/$file_name w]
puts $fh {help}; // Execute the "help" command in the Precision tool
close $fh

// CREATE FOLDER HIERARCHY
// Note that the <filepath> argument is an empty string and
// the folder name is specified by the -description argument
set folder_sif_path [solution file add {} -output -type FOLDER \
    -description "Top-Level Folder"]
set folder_sif_path [solution file add {} -output -type FOLDER \
    -folder $folder_sif_path -description "Sub-Folder"]

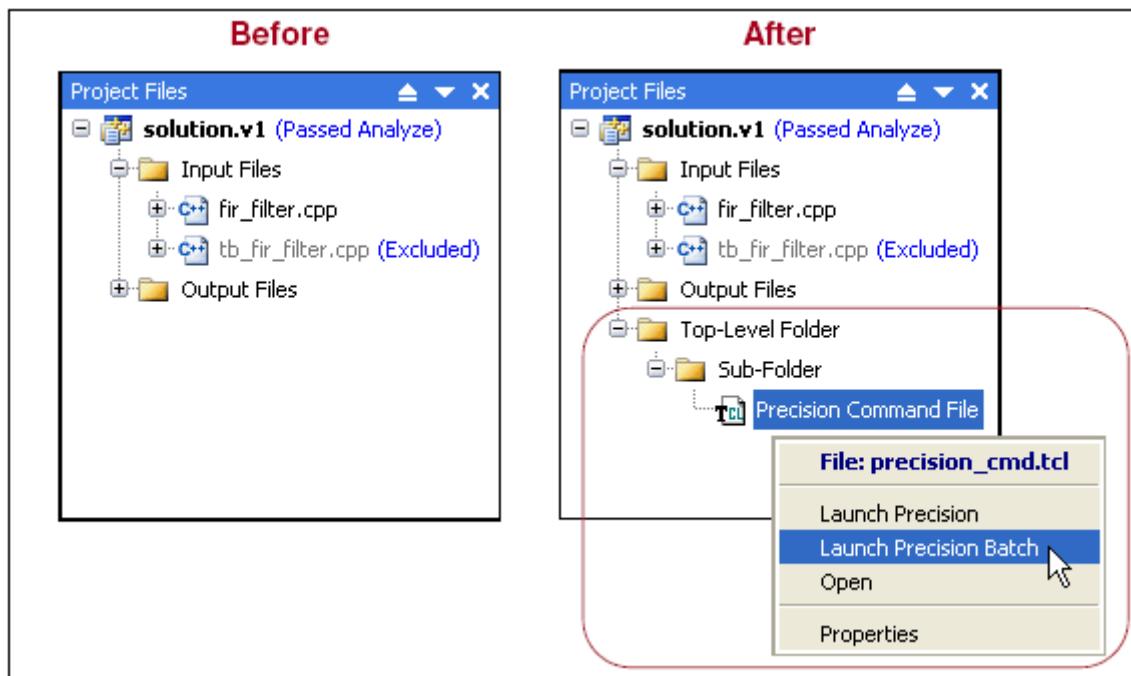
// ADD OUTPUT FILE AND ASSOCIATE FLOW COMMANDS
// Note that the "list" command embedded in the "-flow" arguments
// simply provides a convenient way of grouping the flow command args.
solution file add $solu_dir/$file_name -output -type TCL \
    -folder $folder_sif_path \
    -description "Precision Command File" \
    -flow {Launch Precision} [list /Precision/precision -file $file_name] \
    -flow {Launch Precision Batch} [list /Precision/precision -shell \
        -file $file_name]
```

The script contains three “solution file add” commands. The first command creates a folder at the top-level named “Top-Level Folder”. The second command creates a child folder named “Sub-Folder”. Note that the <filepath> argument is an empty string when adding a folder, and the folder name is specified by the “-description” argument.

The last command adds the “precision_cmd.tcl” to “Sub-Folder”. Because the file type is “TCL”, Catapult automatically displays the Tcl icon next to the file in the GUI. The two “-flow” arguments specify flow commands to be associated with the file. The “list” command embedded in the “-flow” arguments is a convenient way to group the flow command arguments. Selecting either “**Launch Precision**” or “**Launch Precision Batch**” from the popup menu will invoke the Precision Synthesis tools and execute the commands in the “precision_cmd.tcl” file. The file contains one command (“help”) in this example.

Commands
solution file add

Figure 8-8. Adding Output Files and Folder Hierarchy to the Solution



Related Commands

[solution get](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution file remove

Remove a file from the Input File list.

Syntax

```
solution file remove ?<filepath>? ?<switches>?  
  
<filepath>          Relative or absolute file path (Optional)  
<switches>          Valid switches: (Optional)  
    -solution <string>  Specify solution  
    -all                All files in the input files list
```

Arguments

- **<filepath>**

The filename and pathname of the input file you want to remove. This argument is required unless the “-all” switch is used.

- **-solution <string>**

Removes the input file(s) from the specified solution. Use this option to operate on a solution other than the active solution.

- **-all**

Remove all input files.

Description

The “solution file remove” command removes the specified file, or all files, from the list of input files in the SIF database. The “Project Files” window in the GUI is updated accordingly. This command cannot be used after the solution has passed the “compile” state.

Example

This example removes second file in the list of input files. The “[solution get](#)” command is used to get the file pathname from the SIF database. The second file is in the “INPUTFILES/2” node of the database.

```
solution file remove [solution get /INPUTFILES/2/name -ret value]
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution file restore

Restores input files from the Version Control System.

Syntax

```
solution file restore <filepath> ?<switches>?  
    Restore input/output file(s) from the Version Control System  
<filepath>           Relative or absolute file path (Required)  
<switches>           Valid switches: (Optional)  
    -solution <string> Specify solution  
    -force             Overwrite existing file(s)  
    -recurse           Also restore dependencies  
    -location <string> Specify the restore location
```

Argument

- **<filepath>**
File and pathname associated with the file to be restored.
- **-solution <string>**
Specifies the name of the solution to be restored.
- **-force**
Specifies that you want to overwrite existing files with the files stored in the Version Control System.
- **-recurse**
Specifies that you also want to restore all dependencies.
- **-location <string>**
Specifies location where you want to restore the files.

Description

Use the “solution file restore” command to restore solution files from the Version Control System. You use the “-force” and “-recurse” switches to indicate how the files are to be restored. Use the “-location” switch to indicate where you want to restore the files.

Example

```
solution file restore {F:/VersionCtrol/solution1} -force  
-{F:/design/src/solution1}
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution file set

Set options on an input or output file.

Syntax

```
solution file set <filepath> ?<switches>?

<filepath>                               Relative or absolute file path (Required)
<switches>                                Valid switches: (Optional)
    -solution <string>                  Specify solution
    -type <string>                     Specify file type
    -input                            File in input file list
    -output                           File in output file list
    -filenum                          Path argument is a file number
    -dbpath                           Path argument is a database path
    -recurse                          Recurse everything under
    -updated                          Specify file has changed
    -exclude <bool>                   Exclude from C synthesis
    -description <string>             Description of file
    -dependencies <string>            List of file dependencies
    -flow <Flow Description> <Flow Command>
        <Flow Description>           Flow that can be run on this file (Multiple)
        <Flow Command>              Flow Description value (Required)
        <Flow Data>                Flow Command value (Required)
    -flow_data <Flow Name> <Flow Data>
        <Flow Name>                Flow data for this file (Multiple)
        <Flow Data>                Flow Name value (Required)
        <Flow Data>                Flow Data value (Required)
    -property <name> <value>
        <name>                      Property for file (Multiple)
        <value>                      name value (Required)
        <value>                      value value (Required)
    -args <string>                   Compilation arguments
```

Arguments

- **<filepath>**

A file system path to the file. Specify either an absolute path or a path relative to the Catapult working directory. Alternatively, the **<filepath>** argument can specify a SIF database path if the **-dbpath** switch is set.

- **-solution <string>**

Specifies the name of the target solution.

- **-type <string>**

Modifies the file type setting. Table 8-11 lists the most common type names. Each file type has a corresponding icon that displays in the Catapult GUI.

- **-input**

Specifies the file is an input file. The target solution must be in the “New” state when operating on input files. If the target solution has passed the “New” state, use the “[go new](#)” command to reset its state before running the “solution file set” command. Forcing a

solution into the “New” state causes a new solution to be branched. The new file settings are applied in the new solution.

- **-output**

Specifies that the file is an output file. The target solution must be at the “Passed Compile” state or further when adding output files.

- **-filenum**

Specifies that the <filepath> argument is a numerical identifier and not a file system pathname. In Catapult’s SIF database, all files are given a unique numerical ID number for internal processing.

- **-dbpath**

Specifies that the <filepath> argument is a SIF database path and not a filesystem path.

- **-recurse**

When operating on SIF database path, use this switch to recursively search all nodes under <filepath>.

- **-updated**

Specifies that the file has changed. This will branch a new solution.

- **-exclude <bool>**

When set to “true”, this switch excludes the input file from C synthesis compilation. For input files only.

- **-description <string>**

Specifies a text string that is displayed in the GUI in place of the file name.

- **-dependencies <string>**

Specifies a list of files that are dependencies of the <filepath> file.

- **-flow <Flow Description> <Flow Command>**

Specifies a flow command that operates on the file. The command is accessible from the file’s popup menu in the GUI. The “<Flow Description>” string appears as a menu item on the popup menu. The “<Flow Command>” string specifies a the database path to flow command and any arguments for the command. (For an example of how to use this switch, refer to the “[solution file add](#)” command examples on page 482.)

- **-flow_data <Flow Name> <Flow Data>**

This option provides a mechanism to store transient flow related information in the database.

- **-property <name value> <Property>**

Specifies a database property name and value for the <filepath> node in the SIF database.

- **-args <string>**

Use this switch to specify compilation arguments for an input file. The arguments are stored in the database with the file and are available to flow commands that operate on the file. For input files only.

Description

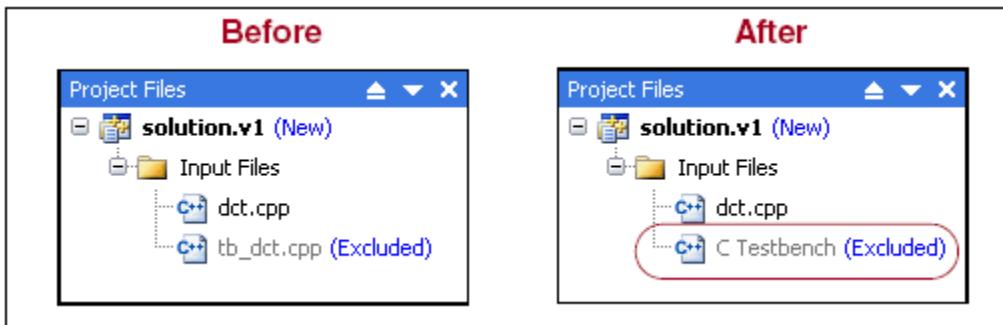
The “solution file set” command modifies a file’s options in the solution. The command writes to the SIF database and the changes are reflected in the “Project Files” window in the Catapult GUI.

Example

This example modifies three options on the input file “tb_dct.cpp”: 1) The description string is set, 2) the file is excluded from C synthesis, and 3) a compiler argument is set. Figure 8-9 shows the changes appear in the “Project Files” window in the GUI.

```
solution file set "tb_dct.cpp" -input -description "C Testbench" \
-exclude "true" -args "-DCCS_SCVERIFY"
```

Figure 8-9. Changing File Options with the Solution File Set Command



Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution new

Create a new solution branch from the specified solution.

Syntax

```
solution new ?<switches>?

<switches>           Valid switches: (Optional)
    -solution <string>  Specify solution

    -incremental        Create an incremental solution

    -state <initial|new|analyze|compile|architect|allocate|
                           schedule|dpfsm|extract|last>
                           Pick design state, default is current
        initial          Initial state of new project
        new              Design state 'new'
        analyze          Design state 'analyze'
        compile          Design state 'compile'
        architect         Design state 'architect'
        allocate          Design state 'allocate'
        schedule          Design state 'schedule'
        dpfsm             Design state 'dpfsm'
        extract            Design state 'extract'
        last              Last valid design state for specified solution
```

- **-solution <string>**

Specifies the name of a solution in the current session from which to branch the new solution. By default, currently active solution is the parent of the new solution.

- **-incremental**

Creates an incremental solution that is branched from the specified solution. For more information, see “[Step 7: Incorporating Post-Synthesis Design Changes](#)” on page 34.

- **-state <string>**

Specifies a state in the parent solution will be the branch point. If the option is omitted, the current state of the parent solution will be the branch point.

Description

The “solution new” command branches a new solution (child) from an existing solution (parent) in the current session. The child solution is brought up to the same design stage as the parent solution, unless the “-state” option is specified.

Examples

In example1 and 2, assume three solutions exist in the current session: dct.v1, dct.v2 and dct.v3. Also assume that dct.v3 is active solution and its state is “Passed Architect.”

Example 1:

This command used default values to branch a new solution (dct.v4) from the current state of the active solution. The new solution is automatically brought to the “Passed Architect” state.

```
solution new
```

Example 2:

This command specifies which solution will be the parent, and design state in the parent to branch from.

```
solution new -solution dct.v2 -state compile
```

Example 3:

The following command branches/creates an incremental solution from the active solution:

```
solution new -state new -solution incr.v1 -incremental
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution netlist

Generate an RTL netlist and tool flow scripts for the current solution (not available in the Catapult BL product).

Syntax

```

solution netlist ?<switches>? ?<filename>?
                                Generate a netlist for solution
<switches>          Valid switches: (Optional)
    -solution <string>  Specify solution
    -state
<new|analyze|compile|architect|allocate|schedule|dpfsm|extract|last>
                                Pick design state, default is current
    new                  Design state 'new'
    analyze              Design state 'analyze'
    compile              Design state 'compile'
    architect            Design state 'architect'
    allocate              Design state 'allocate'
    schedule              Design state 'schedule'
    dpfsm                Design state 'dpfsm'
    extract               Design state 'extract'
    last                 Last valid design state for specified solution
    -window               Display the netlist in a window
    -replace              Overwrite existing file
    -systemc              Generate SystemC netlist
    -vhdl                Generate VHDL netlist
    -verilog              Generate Verilog netlist
    -library              Generate custom component library
    -append               Append to existing library file
    -93                  Write VHDL-93 style netlist
    -configs              Generate VHDL component configurations
    -attributes            Netlist database properties as VHDL attributes
    -insert <string>      Insert text before entity declarations
    -delay <value> <units> Delay value for each register assignment
    -linenumber            Add source cross-ref information to netlist
    -name <string>        Toplevel design name
    -prefix <string>      Prefix for all sub-blocks
    -max_name_len <int>   Maximum length for names
    -architecture <string>
                            Toplevel design architecture name
    -synfile <string>     Name of flow generated synthesis file
    -syntimingfile <string>
                            Name of flow generated synthesis timing file
    -msimfile <string>    Name of flow generated simulation file
    -property_map <propname> <propval>
                            Adds a property to the All binding for the
                            generated module
    -input_register <bool> Generate custom component with input registers
    -ignore_init           Ignore init delay restrictions when writing
                            library
    <filename>            Netlist filename to generate (Optional)

```

Arguments

- **<filename>**

Optional argument specifying the path and filename for the output files. Include the file extension for the netlist file, if desired. File extensions are automatically added to the generated flow scripts. If only the leaf filename is specified, the output files are created in the \$PROJECT_HOME directory. If the <filename> argument is omitted, a unique name is generated and the output files are created in the user's temporary directory.

- **-state <state_name>**

Optional switch specifying the Catapult state from which the output files will be generated. The default is the current state. You cannot specify a state that has not yet been passed. This switch does not change the current state. For more information about Catapult design states, refer to in the section “[The Catapult Work Flow](#)” on page 46.

- **-window**

Opens the netlist file in a text editor window.

- **-replace**

Overwrite an existing netlist file. If not specified, an error message is generated and no output is produced.

- **-vhdl | verilog | systemc**

Optional switch indicating whether a VHDL, Verilog or systemc netlist should be generated. The default is VHDL.

- **-library**

Generate custom component library. Refer to “[Bottom-Up CCORE Methodology](#)” on page 602 for a detailed description of the procedure. See also “-input_register” and “-property_map” switches.

- **-append**

This switch is used in conjunction with the “-library” switch to add the current CCORE module to an existing library.

- **-93**

Optional switch writes VHDL-93 style netlist.

- **-configs**

Optional switch indicating that VHDL component configuration specifications should be generated. The default is to not generate component configuration specifications.

- **-attributes**

Optional switch indicating that some database properties should be netlisted as VHDL attributes.

- **-insert { <string> }**
Optional switch that causes the specified text to be inserted immediately before all entity declarations except those produced by the generators.
- **-delay <value> <units>**
Optional switch to specify the delay and delay unit values on each register assignment in the design. By default, no delay is used.
- **-linenumber**
Optional switch indicating that source cross-reference information (file and line number) should be added as VHDL comments to operations, signals and components. The default is to not add source cross-reference comments.
- **-name <string>**
Optional switch specifying the name of the top-level design.
- **-prefix <string>**
Optional switch specifies a string that will be prefixed to every block name in the generated HDL, except the top-level entity for the design. Block names in IP files are not affected.
- **-max_name_len <int>**
Optional switch controls whether or not variable names are truncated in the netlist, and if so how long they can be. The <int> argument specifies the maximum name length. The factory default length is 255 characters. The minimum allowed length is sixteen characters. If <int> is zero, truncation is disabled.

The truncation algorithm preserves the first [Max Name Length - 6] characters of the original name and replaces the rest with a unique 6 digit number. For example, if the maximum name length is set to 20, then the variable name “st_fir_filter2_proc_loop_1” will become “st_fir_filter2xxxxxx, where “xxxxxx” is a unique 6 digit number.
- **-architecture <architecture_name>**
Optional switch to specify the name of the top-level architecture.
- **-synfile <filename>**
Optional switch specifying the name of the Synthesis file to be generated.
- **-syntimingfile <filename>**
Optional switch specifying the name of the Synthesis timing file to be generated.
- **-msimfile <filename>**
Optional switch specifying the name of the ModelSim file to be generated.
- **-property_map <propname> <propval>**
This switch is used only in conjunction with the -library switch. Adds a property to the “All” binding on the library module and overrides the properties shown in the following table.

Refer to “[Netlisting the Component and Setting Library Component Properties](#)” on page 605 for more information and an example.

Table 8-12. Netlist Properties

Combinational	Sequential
Area	Area
Delay	SeqDelay
	InitDelay
	MinClkPrd

- **-input_register<bool>**

Used in conjunction with the -library switch to specify whether the input ports on the library module have input_registers. By default, all input ports on a generated library module have input registers and SeqDelay is adjusted automatically.

- **-ignore_init**

This switch is used only in conjunction with the -library switch. Ignore errors generated because the throughput of sequential custom components is not equal to one. Suppressing the errors will allow Catapult to generate a netlist that can be used for simulation purposes. Be aware that synthesizing such components may produce bad logic.

Description

The “solution netlist” command is only available in the Catapult SL product. This command generates RTL netlists. If <filename> is not specified, the netlist is written to a temporary file (with a unique name). If Catapult is invoked without “-window” specified, then the temporary file is left on disk. If “-window” is specified, the contents of the file is copied into the window and the temporary file is deleted. If <filename> is specified, then the file is left on disk regardless of the “-window” setting. If a temporary filename was left on disk, the name of the temporary file is written to the transcript.

Examples

Generates an RTL netlist, *samp2.vhd*.

```
netlist -l -insert "library special; use special.all;" samp2.vhd
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution remove

Removes a solution from the project.

Syntax

```
solution remove ?<switches>?  
  
<switches>           Valid switches: (Optional)  
    -solution <string>  Specify solution
```

Arguments

- **-solution <string>**

Specifies the name of the solution to be deleted.

Description

The “solution remove” command deletes the current solution or the solution specified by the “-solution” option. If only one solution exists, the command generates an error and does not delete the solution.

Example

```
solution remove -solution dct.v3
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution rename

Renames a solution.

Syntax

```
solution rename ?<switches>? <name>  
  
<switches>           Valid switches: (Optional)  
    -solution <string>  Specify solution  
<name>              New name (Required)
```

Arguments

- **<name>**

A string value that will be the new name for the solution. This argument only affects the basename portion of the solution name, not the version portion.

- **-solution**

Specifies the name of the solution to be renamed. Use this switch to rename a solution other than the active solution. The specified name must include the version portion of the name.

Description

The solution rename command renames the basename portion of the specified solution. Catapult automatically sets the version portion of the new solution name. The solution directory name on disk is also changed. You can use “solution get /SOLUTION_DIR” to find the solution directory path.

Example

Example 1:

This example renames the active solution to “pipelined.”

```
solution rename pipelined  
# pipelined.v1
```

Example 2:

This example renames the specified solution.

```
solution rename -solution dct.v3 pipelined  
# pipelined.v1
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution report](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution report

Generates a report on the active solution.

Syntax

```
solution report ?<options>?

<options>           Valid switches: (Optional)
  -filename <string>      Write report to file
  -transcript <bool>       Send report to transcript
  -window <bool>          View report in window
  -all                  Include all reports
  -area <bool>            Include Area in report
  -bom <bool>             Include Bom in report
  -clock <bool>           Include Clock in report
  -description <string>   Include Description in report
  -filters <bool>          Include Filters in report
  -header <bool>           Include Header in report
  -iorange <bool>          Include Iorange in report
  -libs <bool>             Include Libs in report
  -loops <bool>            Include Loops in report
  -memories <bool>         Include Memories in report
  -multicycle <bool>      Include Multicycle in report
  -processes <string>     Include Processes in report
  -profiles <bool>          Include Profiles in report
  -registers <bool>        Include Registers in report
  -solution <string>       Include Solution in report
  -summary <bool>           Include Timing in report
  -timing <bool>            Include Summary in report
```

Arguments

For switches that take a `<bool>` argument, Table 8-13 shows the valid values.

Table 8-13. Valid Values for bool Arguments

Enable	Disable
“yes” or “y”	“no” or “n”
“true” or “t”	“false” or “f”
Number other than zero	Zero

- **-filename <string>**

Saves the report in the specified file path. The path is relative to the Catapult “working directory”. When specifying a path to another directory, the target directory must exist and be writable. If the file already exists, it is overwritten. Otherwise a new file is created.

When a new file is created, it is automatically added to the “Output Files/Reports” folder in the “Project Files” window of the GUI. The default name of the file in the GUI is “Report”, but it is configurable by using the “-description” switch.

- **-transcript <bool>**

When this switch is enabled the report is sent to the transcript window.

- **-window <bool>**

When this switch is enabled (default) the report is opened in a text viewer window in the GUI.

- **-all**

Includes all optional data in the report.

- **-area <bool>**

Reports the total area of the design as an aggregate of its component. The area data is reported under the heading “Functional Unit Area“.

- **-bom <bool>**

Reports the Bill of Materials Constraint settings under the heading “Bill Of Materials (Datapath)“.

- **-clock <bool>**

Reports the Clock Constraint settings under the heading “Clock Information“.

- **-description <string>**

Use this switch to specify a descriptive name for the report file created by the “-filename” switch that will appear in the Catapult GUI. The value of <string> appears next to the report file icon in the “Output Files/Reports” folder in the “Project Files” window. By default, the string “Report” is used.

- **-filters <bool>**

Reports the settings of the “Pseudo-Operation Filter” and the “Allocation Filter” under the heading “Filter Settings”.

- **-header <bool>**

Disable this switch to suppress some of the default header information.

- **-iorange <bool>**

Reports I/O data range information under the heading “I/O Data Ranges”.

- **-libs <bool>**

Reports the list of component libraries loaded for this project. This information is found under the heading “Component Libraries In Use“.

- **-loops <bool>**

Reports the hierarchy of loops, their constraint settings and the total latency of the design. This information is found under the heading “Loops“.

- **-memories <bool>**

Reports constraint settings for all memory resources. This information is found under the heading “Memory Resources“.

- **-multicycle <bool>**

Reports multi-cycle components under the heading “Multi-Cycle (Combinational) Component Usage”.

- **-profiles <bool>**

Reports a profile of how much time is spent in each loop in the design. This information is found under the heading “Loop Execution Profile”.

- **-registers <bool>**

Reports the register-to-variable mapping and how many register bits are used. This information is found under the heading “Register-to-Variable Mappings”.

- **-summary <bool>**

Reports a summary of the total area estimate.

- **-timing <bool>**

Reports timing estimates for instances, components and the critical path. This information is found under the heading “Timing Report”.

Description

The “solution report” command generates a report about the active solution containing the requested information. The report can be sent to the transcript, a text viewer window (GUI only) or a file.

Example

```
solution report -summary y
```

This command requests a Summary Report for the active solution as shown below.

```
solution report -transcript 1 -summary 1
#
# -- Catapult System Level Synthesis: Report
# -----
# -- Version:                                <version and release>
# -- Build Date:                             <date of software>
#
# -- Generated by:                           <user name>
# -- Generated date:                         <date when report was generated>
#
# Solution Settings: dct.v1
# Current state: extract
# Project: Catapult
#
# Design Input Files Specified
#     $MGC_HOME/shared/examples/catapult/dct/dct.cpp
#     $MGC_HOME/shared/examples/catapult/dct/global_dct.h
#     $MGC_HOME/shared/include/ac_int.h
#     $MGC_HOME/shared/examples/catapult/dct/tb_dct.cpp
#     $MGC_HOME/shared/examples/catapult/dct/global_dct.h
#     $MGC_HOME/shared/include/ac_int.h
#     $MGC_HOME/shared/include/mc_scverify.h
```

```
# Processes/Blocks in Design
# Process      Real Operation(s) count Latency Throughput Reset Length II
Commen
#
# -----
# /dct/core          16    1158     1170        0   0
# Design Total:      16    1158     1170        0   0
#
# Area Scores
#           Post-Scheduling  Post-DP & FSM Post-Assignment
#
# Total Area Score: 6773.3      12559.6    8329.0
# Total Reg:       1000.6 (15%)  1236.8 (10%) 1236.8 (15%)
#
# DataPath:        6773.3 (100%) 12419.6 (99%) 8189.0 (98%)
# MUX:             0.0          74.3 (1%)   277.7 (3%)
# FUNC:            5313.9 (78%)  9758.4 (79%) 5313.9 (65%)
# LOGIC:           0.0          559.5 (5%)   570.1 (7%)
# BUFFER:          0.0          0.0          0.0
# MEM:             0.0          0.0          0.0
# ROM:             458.8 (7%)   917.6 (7%)  917.6 (11%)
# REG:             1000.6 (15%) 1109.8 (9%)  1109.8 (14%)
#
#
# FSM:             0.0          140.0 (1%)  140.0 (2%)
# FSM-REG:         0.0          127.0 (91%) 127.0 (91%)
# FSM-COMB:        0.0          13.0 (9%)   13.0 (9%)
#
#
# End of Report
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution restore](#)
[solution select](#)
[solution timing](#)

solution restore

Restores input files from the Version Control system.

Syntax

```
solution restore ?<switches>?  
  
<switches>           Valid switches: (Optional)  
    -solution <string>  Specify solution
```

Arguments

- **-solution <string>**

Specifies the name of the solution to be restored.

Description

The “solution restore” command restores input files from Catapult version control system for the specified solution.

Example

```
solution restore -solution dct.v3
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution select](#)
[solution timing](#)

solution select

Makes the specified solution the active solution.

Syntax

```
solution select ?<switches>? ?<solution>?  
  
<switches>           Valid switches: (Optional)  
    -solution <string>  Specify solution  
<solution>          Solution name (Optional)
```

Arguments

- **-solution <string>**
The <string> argument specifies the name of a solution in the current project.
- **<solution>**
The name of a solution in the current project.

Description

The “solution select” command makes the specified solution the active (working) solution. Use either the “-solution” switch or the <solution> argument to specify an existing solution name in the project.

Example

The following two commands are equivalent.

```
solution select -solution dct.v3
```

and

```
solution select dct.v3
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution timing](#)

solution timing

Generate timing reports for critical paths and user-specified data paths in the design.

Syntax

```
solution timing ?<options>?

<options>           Timing options (Optional)
  -filename <string>  Write report to file
  -from <string>      Report critical path from <op>
  -to <string>        Report critical path to <op>
  -through <string>   Report critical path through <op>
  -count <int>         Report <num> of critical paths
  -map <int>          Report critical paths map for paths exceeding
                      clock period by <value>%
  -variable <string>  Write report to string variable
```

Arguments

- ***-filename <string>***

Save the timing report in the file named <string> in the filesystem. The <string> argument can be an absolute pathname or a path relative to the current working directory.

- ***-from <string>***

Report timing for all instances in the data path on the output side of the instance specified by the <string> argument. The <string> argument must be the full path to an instance.

- ***-to <string>***

Report timing for all instances in the data path on the input side of the instance specified by the <string> argument. The <string> argument must be the full path to an instance.

- ***-through <string>***

Report timing for all instances in the data path that runs through the instance specified by the <string> argument. This option is not applicable if <string> specifies a register. The <string> argument must be the full path to an instance.

- ***-count <int>***

Report timing for the <int> most critical paths in the design. The list of paths in the report is ordered from most critical to least critical.

- ***-map <int>***

Report timing of all the critical paths having negative slack that exceeds the clock period by a specified threshold value (<int>). Valid threshold values are integers greater than or equal to 100. Setting a higher threshold (greater negative slack) will filter out paths whose delay is less than the threshold. If no paths have negative slack, or if none is greater than the threshold, the command returns an empty list.

Description

The “solution timing” command reports the individual delay for all instances in the specified path(s) and the cumulative delay and slack of the entire path. Registers are treated as path boundaries because they can have more than one path flowing through them. The “-count” and “-map” options report multiple paths and lists them in the order of their delay values (largest to smallest). Reports appear in the transcript window and can also be save in a file.

The format of the report is hierarchical. At the top-level is the “Max Delay” and “Slack” values of the most critical path (first path listed in the report). Below that, each path is listed. For each path, the first line identifies the start and end points of the path, and the total path delay and slack. Below that is a sequential list of each instance in the path. Each line includes the following four columns of data:

- **Instance:** The instance name. The format is <design_path>/<instance_name>.
- **Component:** The name of the library component (if applicable).
- **Delta:** The incremental delay of the instance.
- **Delay:** The cumulative delay of the path up to and including the instance.

Below is the format of the timing reports.

```
Max Delay:  <number>
Slack:      <number>

Path  Start Point          End Point          Delay   Slack
-----><inst_path>           <inst_path>       <number> <number>
1
    Instance      Component      Delta   Delay
    -----><inst_path>     <comp_name>   <number> <number>
    <inst_path>     <comp_name>   <number> <number>
    ...
    ...
2
    <inst_path>           <inst_path>       <number> <number>
    Instance      Component      Delta   Delay
    -----><inst_path>     <comp_name>   <number> <number>
    <inst_path>     <comp_name>   <number> <number>
    ...
    ...
```

Examples

The command examples below do not show their return values because the report format is too wide for the page. In order to try the example commands in a Catapult session, run the following Tcl script first.

```
project new
solution file add \
    {$MGC_HOME/shared/examples/catapult/fir_filter/fir_filter.cpp}
go analyze
directive set -TECHLIBS {{mfc_lcbg1lp_beh_dc.lib mfc_lcbg1lp_beh_dc} \
```

Commands

solution timing

```
{ram_lcbg11p-dualport_beh_dc.lib ram_lcbg11p-dualport_beh_dc} \
{ram_lcbg11p-separate_beh_dc.lib ram_lcbg11p-separate_beh_dc} \
{ram_lcbg11p-singleport_beh_dc.lib ram_lcbg11p-singleport_beh_dc}
directive set -CLOCKS {clk {-CLOCK_PERIOD 1.42857142857 \
-CLOCK_EDGE rising -CLOCK_HIGH_TIME 1.42857142857 \
-CLOCK_OFFSET 0.000000 -RESET_KIND sync -RESET_SYNC_NAME rst \
-RESET_SYNC_ACTIVE high -RESET_ASYNC_NAME arst_n \
-RESET_ASYNC_ACTIVE low -ENABLE_NAME {} -ENABLE_ACTIVE high}}
go extract
```

i **Tip:** To get the full path of an instance, right-click on it and choose “Copy Path to Clipboard” from the popup menu.

Example 1:

The following command reports the path delay for all instances up to, and including, the MAC:acc#2 instance.

```
solution timing -to
/fir_filter/fir_filter:core/fir_filter:core:conc/MAC:acc#2
```

Example 2:

The following command reports the path delay for all instances on the same path as the MAC:acc#2 instance.

```
solution timing -through \
/fir_filter/fir_filter:core/fir_filter:core:conc/MAC:acc#2
```

Example 3:

The following command reports the path delay for three most critical paths.

```
solution timing -count 3
```

Example 4:

The following command reports all paths whose delays exceed 110% of the clock period.

```
solution timing -map 110
```

Related Commands

[solution get](#)
[solution file add](#)
[solution file remove](#)
[solution file restore](#)
[solution file set](#)
[solution new](#)

[solution remove](#)
[solution rename](#)
[solution report](#)
[solution restore](#)
[solution select](#)

view schedule

Opens the Gantt chart schedule of operations for viewing.

Syntax

```
view schedule
```

Arguments

None.

Description

The “view schedule” command opens the Gantt chart file in the active solution directory (\$PROJECT_HOME/<active_solution>/schedule.gnt) for viewing in the schedule viewer window.

Example

```
view schedule
```

Related Commands

[view schematic](#)
[view file](#)

[view source](#)

view schematic

Opens the RTL schematic window. Displays critical paths and path timing data.

Syntax

```
view schematic ?<type>? ?<switches>?

<type>           schematic type (Optional)
    rtl            RTL schematic
    critical        Critical path schematic
    datapath        Datapath schematic
    criticalmap     Critical map schematic
<switches>       Valid switches: (Optional)
    -count <int>   Critical path count (valid for 'critical' type)
    -to <string>    Highlight path to operation (valid for 'critical'
                     mode)
    -from <string>  Highlight path from operation (valid for
                     'critical' mode)
    -through <string> Highlight path through operation (valid for
                     'critical' mode)
    -map <int>      Display timing map for paths exceeding % of clock
                     period (valid for 'criticalmap' mode)
```

Arguments

- **<type>**

This optional argument specifies the initial *view state* of the schematic viewer when it is opened. Valid values for the view state are:

rtl : Opens in the “RTL schematic” view state. (Default)

critical : Opens in the “Critical Path” view state.

datapath : Opens in the “Data Path” view state.

criticalmap : Opens in the “Critical Map” view state.

For a detailed description of each of these view states, refer to “[View States in the Schematic Viewer](#)” on page 149.

- **-count <int>**

Specifies the number of critical paths to display in the path table of the Critical Path schematic. The table contains the most critical paths in the design listed in order from most critical to least critical. This switch must be used in conjunction with the “critical” type argument (above).

- **-to <string>**

In the Critical Path schematic, reports timing data for and highlights all instances in the path up to and including the instance specified by the <string> argument. The <string> argument must be the full path to an instance. This switch must be used in conjunction with the “critical” type argument (above).

- **-from <string>**

In the Critical Path schematic, reports timing data for and highlights all instances in the path on the output side of the instance specified by the <string> argument. The <string> argument must be the full path to an instance. This switch must be used in conjunction with the “critical” type argument (above).

- **-through <string>**

In the Critical Path schematic, reports timing data for and highlights all instances in the path that runs through the instance specified by the <string> argument. This option is not applicable if <string> specifies a register. The <string> argument must be the full path to an instance. This switch must be used in conjunction with the “critical” type argument (above).

- **-map <int>**

In the Critical Map schematic window, highlight all the critical paths having negative slack that exceeds the clock period by a specified threshold value (<int>). Valid threshold values are integers greater than or equal to 100. Setting a higher threshold (greater negative slack) will filter out paths whose delay is less than the threshold. If no paths have negative slack, or if none is greater than the threshold, the command returns an empty list. The default threshold value is 100. This switch must be used in conjunction with the “criticalmap” type argument (above).

Description

The “view schematic” command opens the RTL schematic file (\$PROJECT_HOME/<active_solution>/schematic.nlv) for viewing in the schematic viewer window. If a schematic window is already open, it will be updated according to the command options and switches. For more information about the Schematic window, refer to “[Using the Schematic Viewer Window](#)” on page 148.

Examples

Example 1:

This example opens the Critical Path schematic (“critical” type) and sets the number of critical paths to 3.

```
view schematic critical -count 3
```

Example 2:

This example the data path through the instance named “mux#39.”

```
view schematic critical -through \
/fir_filter/fir_filter:core/fir_filter:core:conc/mux#39
```

Related Commands

[view schedule](#)
[view file](#)

[view source](#)

view file

Opens the specified file in the DesignPad Editor window.

Syntax

```
view file <filepath> ?<switches>?

<filepath>           Relative or absolute file path (Required)
<switches>          Valid switches: (Optional)
    -filetype <string>   file type
    -branch <string>     name
```

Arguments

- **<filepath>**

A file system path to the file being opened. Specify either an absolute path or a path relative to the Catapult working directory. If the specified filepath does not exist, a new document of is opened in the DesignPad editor which can be saved to the specified filepath.

- **-filetype <string>**

Specifies the type of file. Generally, Catapult can determine the type based on the filename extension. [Table 8-11](#) on page 482 lists the most common file type names that are known to Catapult.

Description

The “view file” command opens the specified file in the DesignPad editor.

Example

This example opens the VHDL file “rlt.vhdl” in the dct.v1solution directory. The path is relative to the current working directory.

```
view file Catapult_7/dct.v1/rtl.vhdl
```

Related Commands

[view schedule](#)
[view schematic](#)

[view source](#)

view source

Opens the C++ source file(s) in the Input Files list.

Syntax

```
view source
```

Arguments

None.

Description

The “view source” command opens all of the C++ design files in the Input Files list. Input files that are excluded from compilation are not opened.

Example

```
view source
```

Related Commands

[view schedule](#)
[view schematic](#)

[view file](#)

Commands
view source

Chapter 9

Design Verification

This chapter describes the three design verification flows available in Catapult:

- **SCVerify Flow** — validates the RTL netlist output from Catapult against the original design source code input using simulation. This flow also generates files that can be used to verify and debug the source code prior to synthesis.
- **Formal Verification Flow** — checks the functional equivalence between the RTL netlist from Catapult and the original design source code with Calypto Sequential Logic Equivalence Checking (SLEC).
- **MATLAB/Simulink Flow (C/C++)** — verifies C/C++ source code using numerical simulation in Mathworks Simulink/MATLAB.

SCVerify Flow

The SCVerify flow is an automated verification flow designed for use with the Catapult synthesis flow. SCVerify is set up/enabled at the beginning of the synthesis project and then, the files needed to run the SCVerify flow are generated at various stages of the synthesis process including:

- **Initial design compilation** — When **Task Bar: Architectural Constraints** is clicked, Catapult compiles the input source code in preparation for setting architectural constraints for synthesis. At this stage, Catapult also generates an SCVerify makefile that drives a compiler and simulator for the purpose of verifying/debugging the compiler setups and your input source code.
- **RTL generation** — When **Task Bar: Generate RTL** is clicked, Catapult generates an RTL netlist. At this stage, Catapult also generates several SCVerify makefiles, test infrastructure files, compilation makefiles, and simulation Tcl scripts for the purpose of verifying your final RTL against the design input source code.

The number and types of scripts generated for SCVerify depend on the verification tools, input source code format, and synthesis settings. For more information, see “[Makefiles Generated by SCVerify](#)” on page 526 and “[Infrastructure Files Generated by SCVerify \(C/C++\)](#)” on page 528.

Supported simulators include: ModelSim®/QuestaSim®, OSCI, NCSim, Vista™, and VCS. Supported compilers vary depending on the input source code and system platform. See the latest version of the Catapult Release Notes for a list of currently supported platforms and tools.

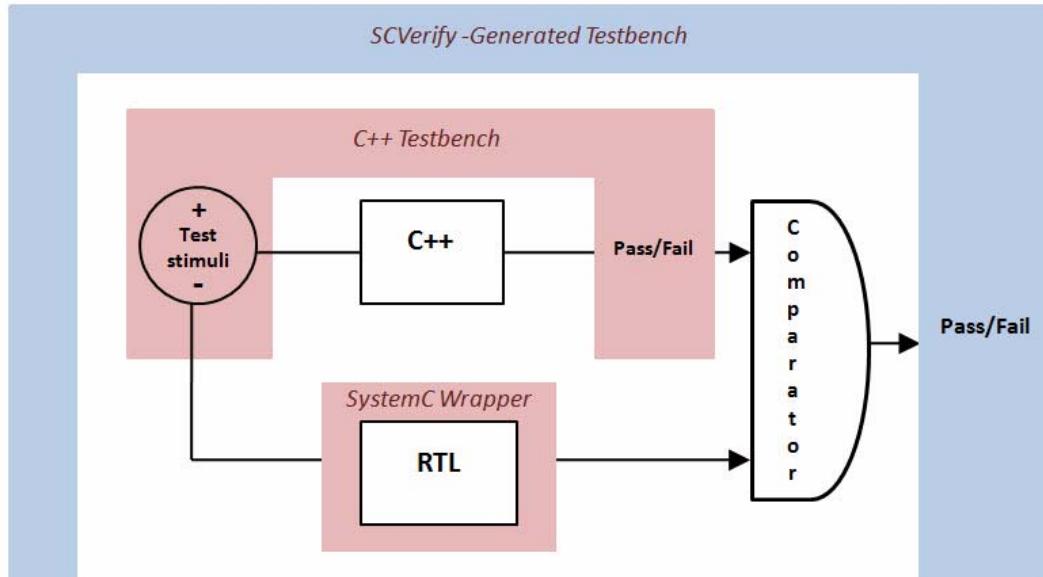
Setting up and running SCVerify is the same regardless of the input source code, however there are differences in the test infrastructure, testbench files, and clock/synchronization signal requirements. Depending on the input source code, see the following topics for more information:

- [C/C++ Designs and SCVerify](#)
- [SystemC Designs and SCVerify](#)

C/C++ Designs and SCVerify

The SCVerify flow validates the RTL netlist output from Catapult against the original C/C++ input source code using a manually-created C/C++ testbench and a SCVerify-generated testbench as shown in [Figure 9-1](#).

Figure 9-1. SCVerify: C/C++ and RTL Verification



This is accomplished by placing the RTL netlist in a SystemC wrapper and instantiating it along with the original C/C++ source code and testbench in a SystemC testbench generated by the SCVerify flow. The same input stimuli are then applied to both the original C/C++ source code and the RTL netlist and a comparator validates the output from both are identical.

Catapult also generates SystemC code to provide interconnect and synchronization signals including interface translator objects. The interface translator objects provide interface connections to the RTL block and handle the translation of the data types into the logic vectors of the RTL netlist. Because the RTL netlist has timed behavior, the translators use the following threads:

- *generate_sync* — provides time synchronization events.

- ***generate_reset*** — handles the initial assertion of the reset signal.
- ***watch_comparators*** — waits for the input FIFOs to empty and comparators to finish comparing outputs and then reports the number of compares made and the number of incorrect comparisons encountered for each comparator.

All of the blocks shown in [Figure 9-1](#) are automatically generated with the exception of the original C/C++ source code and testbench. The C/C++ testbench must provide the test stimuli and one or more call(s) to the original function to get a calculated output. See “[Preparing a Testbench for SCVerify \(C/C++\)](#)” on page 530.

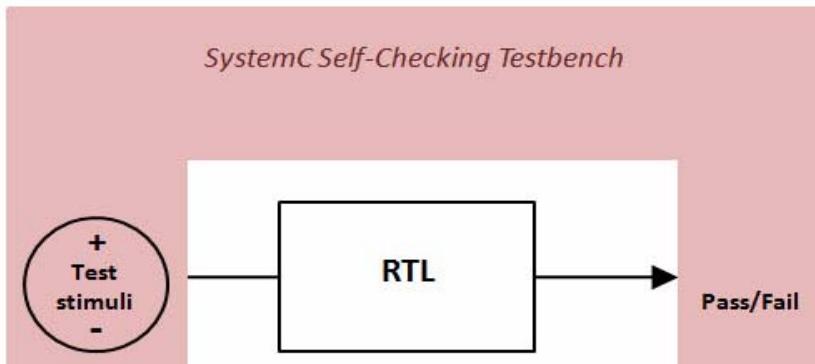
When the simulation is finished, a final pass/fail result is reported to the transcript.

SystemC Designs and SCVerify

The SCVerify flow allows you to validate the RTL netlist output from Catapult against the original SystemC source code input using a manually-created self-checking SystemC testbench.

This is accomplished by connecting the RTL netlist inside the self-checking testbench in place of the original SystemC design source code as shown in [Figure 9-2](#).

Figure 9-2. SCVerify: SystemC and RTL Verification



The SystemC source code provides all the synchronization signals necessary for simulation, and the self-checking testbench contains the test stimuli and target output signal values for validation.

When the simulation is finished, a final pass/fail result is reported to the transcript.



Note

If Modelsim is used for verification, the makefiles generated by SCVerify can also be used to debug your SystemC source code in the ModelSim debug environment.

Setting Up SCVerify

Use this procedure to set up/enable SCVerify at the beginning of a project so the verification files are generated at various stages during the synthesis process. You can set up SCVerify at anytime during the synthesis process and depending on the stage of the synthesis process, verification files will be generated.

Prerequisites

- **Testbench file for the design** — Catapult must have access to the uncompiled testbench file in order to generate the SCVerify files. Depending on the design source code, you must provide a testbench that proves the input design source code as follows:
 - SystemC requires a self-checking testbench.
 - C/C++ requires the original design testbench extended to work with SCVerify. For more information, see “[Preparing a Testbench for SCVerify \(C/C++\)](#)” on page 530.

Procedure

1. Invoke Catapult and select **View > Other Windows > Flow Manager**. The Flow Manager window displays the flows available in Catapult.
2. Select SCVerify and click **Enable**. The [Flow Package: SCVerify Dialog Box](#) displays.
3. Select the desired SCVerify options and click **Apply**.

Note



If the SCVerify options are changed after the RTL is generated, the SCVerify makefiles must be regenerated in order to apply the change. Right-click on makefile and choose Regenerate Makefile.

4. Add the input design source code, header file, and testbench. For more information, see “[Adding and Removing Input Files](#)” on page 93.

Note



C/C++ designs must have synchronization signals for data transfer set up before verification makefiles are generated. For more information, see “[Design Synchronization \(C/C++\)](#)” on page 529.

5. In the Project Files window, right-click on the testbench file and select **Exclude from Compilation**.

The testbench should be excluded from compilation with the design because the SCVerify flow needs access to the uncompiled testbench. The makefiles generated by the SCVerify flow will compile the testbench based on the simulator/compiler selected for verification.

Results

SCVerify is set up and the verification makefiles will be generated as your design progresses through the various synthesis stages.

Depending on your work flow, you are ready to proceed to one of the following steps:

- Verify/Debug your testbench and input source code.
- Verify your final RTL netlist.

Related Topics

flow package require

[Verifying Input Source Code](#)

[Verifying RTL Model](#)

Verifying Input Source Code

Use this procedure to verify your input source code before/during the synthesis process using the makefiles generated by SCVerify. This process should be used to verify that your design compiles whenever changes are made to the input source code.



Note

If ModelSim is installed and a SystemC design is loaded, the verification makefile can also be used to invoke ModelSim debug environment to analyze your design.

Prerequisites

- SCVerify is set up and enabled. See “[Setting Up SCVerify](#)” on page 518.
- Target simulator/compiler applications are installed and set up.
- Project is open and the design technology, frequency, and synchronization signals are set up.



Note

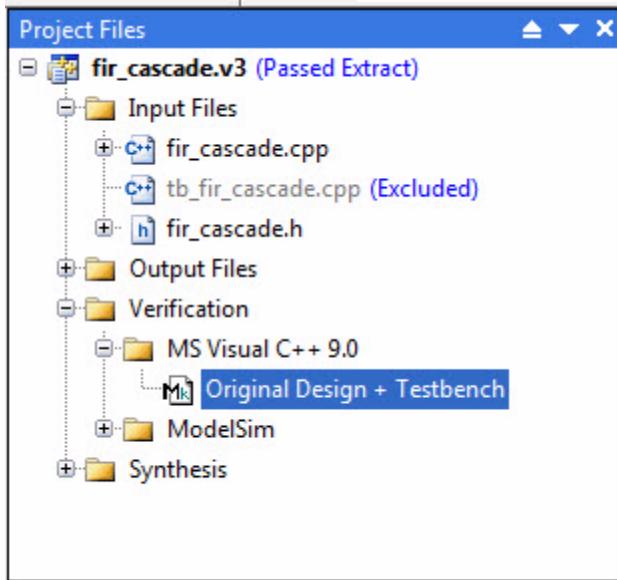
C/C++ designs must have synchronization signals set up before verification makefiles are generated. For more information, see “[Design Synchronization \(C/C++\)](#)” on page 529.

Procedure

1. Click **Task Bar: Architecture Constraints**. The verification makefile is generated and saved to a Verification folder in the Project Files window.
2. Expand the Verification folder and the compiler subfile to access the verification makefile. The name of the compiler subfolder reflects the name of the compiler set up/installed for your project.

Figure 9-3 highlights the makefile generated for C/C++ input source code and the OSCI simulator.

Figure 9-3. SCVerify: Makefile for C/C++ Simulation



3. Double-click the **Original Design + Testbench** makefile to compile the design.
Right-click on the makefile to display more options for executing/processing the makefile. For more information on the makefile options, see “[SCVerify Makefile Popup Menu](#)” on page 658.
4. If verification fails, do the following:
 - o Verify the source code using Valgrind or Purify.
 - o Check the design synchronization. For more information, see “[Design Synchronization \(C/C++\)](#)” on page 529.

Results

The design is compiled and the results are output to the transcript.

Related Topics

[SCVerify Flow Troubleshooting](#)

[Verifying RTL Model](#)

Verifying RTL Model

Use this procedure to simulate and verify your RTL netlist against the input design source code with files generated by SCVerify.

Prerequisites

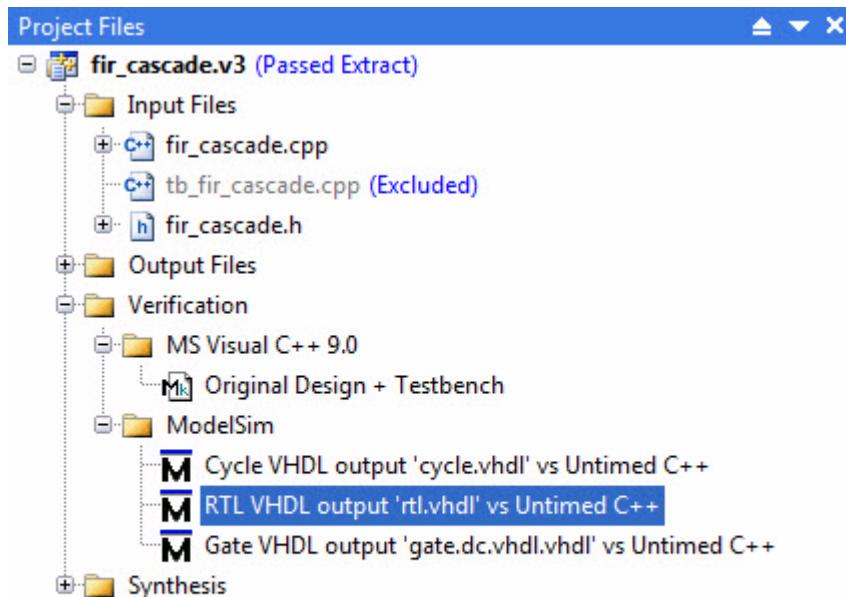
- The SCVerify flow is enabled. For more information, see “[Setting Up SCVerify](#)” on page 518.
- Target simulator/compiler applications are installed and set up.
- Project is open and an RTL netlist is generated. For more information, see “[Generating the Output Files](#)” on page 141.

Procedure

1. Double-click the **RTL <hdl> output <RTL_file.hdl> vs <source_code>** verification makefile saved in the simulator-specific folder of the Project Files window.

[Figure 9-4](#) highlights the makefile generated for C/C++ input source code, RTL VHDL netlist, and ModelSim.

Figure 9-4. SCVerify: Makefile for VHDL, C/C++, ModelSim



Right-click on the makefile to display more options for executing/processing the makefile. For more information on the makefile options, see “[SCVerify Makefile Popup Menu](#)” on page 658.

The makefile commands display in the Catapult transcript window, a ModelSim session is opened and the *cubed* design is loaded and simulated.

Results

The makefile compiles the design and testbench, and launches the simulation in the interactive mode.

Related Topics

[SCVerify Flow Troubleshooting](#)

Linking External Libraries into SCVerify

Use this procedure to set up the SCVerify environment when your design links in external software libraries for compilers and/or simulators.

Prerequisites

- SCVerify is enabled. See [“Setting Up SCVerify”](#) on page 518.

Procedure

1. Click the Flow Manager tab and select SCVerify. The [Flow Package: SCVerify Dialog Box](#) displays.

2. Specify the paths to the directories containing the object code using the **Additional include directory paths** option.

For example, for a software package that supplies a header file *my_sw.h* and a GNU C++ archive library *libmy_sw.a* in a directory named */software*, you would enter */software* here.

3. Specify the library paths that Catapult should search for the object code during compilation in the **Additional link library paths:** option.

For the example in Step 1, you would enter */software* here.

To create makefiles that can be used across multiple platforms/compilers, use makefile variables in this option. For more information, see [“External Libraries and Multiple Platforms/Compilers”](#) on page 523.

4. Specify the names of the libraries containing the object code in the **Additional link libraries:** option.

For the example in Step 1, you would enter *my_sw* here. Notice that the *lib* prefix and the *a* suffix are omitted. Catapult adds these values back in based on the target software platform.

5. Click **Apply** to apply the configuration to the current project only or **Apply and Save** to apply it to the current project and save it as default for all new projects.

Related Topics

[options set](#)

[External Libraries and Multiple Platforms/Compilers](#)

External Libraries and Multiple Platforms/Compilers

The makefile variables described in [Table 9-1](#) allow you to manage multiple platforms and/or compilers for designs that link to external software libraries.

Table 9-1. Makefile Variables

Variable	Description
<code>\$(CXX_OS)</code>	Variable used in a makefile/script that assumes the value of the operating system on which the makefile is executed.
<code>\$(CXX_TYPE)</code>	Variable used in a makefile/script that assumes the value of the compiler available in the system environment in which the makefile is executed. Either msvc or gcc depending on the

For example, you can set up a single directory structure that all system platforms can access for external object software such as:

<code>/software/include</code>	- common include directory
<code>/software/lib/Linux/gcc/libmy_sw.a</code>	- Linux GNU g++ archive
<code>/software/lib/Windows_NT/msvc/libmy_sw.lib</code>	- Microsoft VC++ archive
<code>/software/lib/Windows_NT/gcc/libmy_sw.a</code>	- MinGW g++ archive

and set a single SCVerify option that works on any of these platforms:

```
LINK_LIBPATHS: /software/lib/$(CXX_OS)/$(CXX_TYPE)
```

Accelerated FPGA Devices and Verification

The SCVerify flow requires precompiled Altera and/or Xilinx simulation libraries to check the RTL-synthesis output of Precision or the PNR output, and anytime the target technology is one of the accelerated device families. The power estimation flows for Altera PowerPlay and Xilinx Xpower also depend on these precompiled libraries. The libraries must be compiled using ModelSim and placed in the `$XILINX_LIB` or `$QUARTUS_LIB` trees respectively. The list of library names is predefined.

The SCVerify flow provides the “`ccs_check_vndr.mk`” makefile that can automatically compile all of the necessary libraries and save them in their appropriate locations in `$XILINX_LIB` or

\$QUARTUS_LIB. In addition, the makefile can be used to check whether the precompiled libraries already exist and to display the current settings of key environment variables.

The makefile can be run from the Catapult command line or from the shell. The syntax for each form is shown below.

From inside Catapult:

```
flow run /<vendor>/precompiled_libs HDL=<lang> <action>
```

From the shell:

```
$MGC_HOME/bin/make -B \  
-f $MGC_HOME/shared/include/mkfiles/ccs_check_vndr.mk VNDR=<vendor> \  
HDL=<lang> <action>
```

where

```
<vendor> is either "Xilinx" or "Altera"  
<lang> is one of: "vhdl", "verilog" or "all"  
<action> is one of: "usage", "check" or "create":  
    "usage" displays this message  
    "check" verifies that simulation libraries can be mapped properly  
    "create" generates (compiles) the vendor simulation libraries:
```

 **Note**

1. The “flow run” command requires that a project is open and the current solution is passed the “compile” stage.
 2. If the selected technology in Catapult is not a vendor accelerated library, you must manually require the flow by calling the “flow package require /Altera” (or /Xilinx) command.
 3. The “check” action creates temporary files in your current working directory when you run from the shell.
-

The structure of the output library trees is as follows:

- For QUARTUS_LIB:

```
$QUARTUS_LIB/<lang>/<tech_name>/...
```

where <lang> is either “verilog” or “vhdl,” and <tech_name> contains the library tree for a particular technology. For example:

```
$QUARTUS_LIB/vhdl/stratixii/...
```

- For \$XILINX_LIB:

```
$XILINX_LIB/<lang>/mti_se/<tech_name>/...
```

where <lang> is either “verilog” or “vhdl,” and <tech_name> contains the library objects. For example:

```
$XILINX_LIB/vhdl/mti_se/XilinxCoreLib/...
```

Example 1: Check the VHDL Library Mappings for Altera:

Inside catapult	flow run /Altera/precompiled_libs HDL=vhdl check
From the shell	\$MGC_HOME/bin/make -B \ -f \$MGC_HOME/shared/include/mkfiles/ccs_check_vndr.mk \ VNDR=Altera HDL=vhdl check

The command transcript is similar to the excerpt shown below. It verifies whether each logical library name can be mapped to a physical library in the \$QUARTUS_LIB tree.

```
=====  
Mapping vendor logical library 'altera_mf' to physical path  
      '$QUARTUS_LIB/vhdl/altera_mf'  
$MODEL_TECH/vmap altera_mf $QUARTUS_LIB/vhdl/altera_mf  
Modifying ./modelsim.ini  
=====  
.  
.  
.  
=====  
Simulation libraries can be successfully mapped
```

Example 2: Create Precompiled VHDL Libraries for Altera:

Inside catapult	flow run /Altera/precompiled_libs HDL=vhdl create
From the shell	\$MGC_HOME/bin/make -B \ -f \$MGC_HOME/shared/include/mkfiles/ccs_check_vndr.mk \ VNDR=Altera HDL=vhdl create

The command invokes ModelSim to compile the VHDL sources in the \$QUARTUS_HOME tree and save the compiled libraries in the \$QUARTUS_LIB tree.

Example 3: Display Environment Variables that Specify Library Locations:

Inside catapult	flow run /Altera/precompiled_libs HDL=vhdl usage
From the shell	\$MGC_HOME/bin/make -B \ -f \$MGC_HOME/shared/include/mkfiles/ccs_check_vndr.mk \ VNDR=Altera HDL=vhdl usage

The command displays the command usage syntax and lists the current settings of relevant environment variable. The listing of environment variables will be similar to the following excerpt:

Current settings:

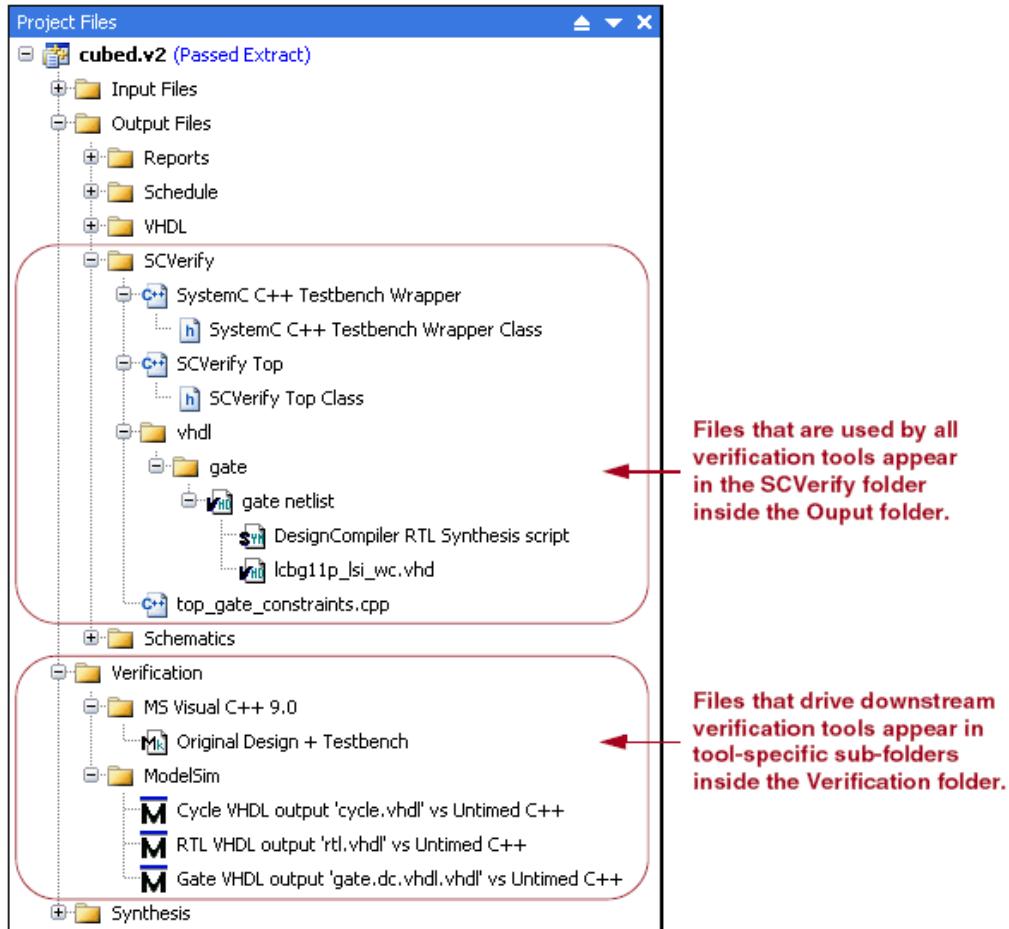
VNDR	= Altera
SIMTOOL	= msim
HDL	= vhdl
QUARTUS_ROOTDIR	= /syntools/pnr/quartus_latest/ixl
QUARTUS_LIB	= /scratch1/project_1/proj_quartus_lib
MODEL_TECH	= /syntools/modelsim_latest/ixl/modeltech/bin

For more information on generating precompiled simulation libraries for Altera and Xilinx, consult the vendor product manuals.

Makefiles Generated by SCVerify

SCVerify generates makefiles at various stages in the synthesis flow and saves them in various folders in the Project Files window as shown in [Figure 9-5](#).

Figure 9-5. SCVerify Makefiles



Makefiles files that drive a particular simulator/compiler are generated in subfolders named for the target simulator/compiler, such as ModelSim or MS VisualC and saved under Verification.

The names of the makefiles reflect their function. Depending on the type of makefile, the name is determined by the makefile purpose, target language (VHDL, Verilog or SystemC) and/or design source code.

Right-click on makefiles to access additional processing options. For more information, see “[SCVerify Makefile Popup Menu](#)” on page 658.

At various stages of the synthesis process, one or more of the following makefiles are output to the Verification folder:

- **Original Design + Testbench**

Makefile that compiles the testbench and input source code to verify they compile and execute correctly in the SCVerify environment. This makefile is generated when the compiler is determined by the “[Input Options](#)” on page 188. This makefile resolves to the *Verify_orig_cxx_osci.mk* or *Verify_orig_cxx_vista.mk* files in the *<solution_dir>/scverify* directory.

The Valgrind utility can be used with “*Verify_orig_cxx_osci.mk*” to check your original C++ design and testbench for memory issues such as uninitialized memory reads and out of bounds array accesses. Consult the Valgrind documentation for information on the reports generated by Valgrind.

The Valgrind-enabled SCVerify flow is only available on Linux and only applies to the makefile used for running the original design and testbench. It has no effect on the ModelSim/QuestaSim or NCSim simulation runs. To enable SCVerify to run the Valgrind utility, use the following command at the Catapult prompt:

```
flow run /SCVerify/launch_make Verify_orig_cxx_osci.mk {} \
SIMTOOL=osci valgrind
```

Refer to “[Valgrind Flow Options](#)” on page 221 for information about setting default options for the Valgrind utility.

- **Cycle <hdl_format> output ‘cycle.<hdl_ext>’ vs <input_source_code_format>**

Makefile that compiles and executes simulation comparing your original design to the cycle model including the creation of *work* libraries, compilation of VHDL/Verilog netlist and SystemC files. This makefile resolves to the *Verify_cycle_<hdl_ext>_<sim_tool>.mk* file in the *<solution_dir>/scverify* directory.

- **RTL <hdl_format> output ‘rtl.<hdl_ext>’ vs <input_source_code_format>**

Makefile that compiles and executes simulation comparing your original design to the RTL model including the creation of *work* libraries, compilation of VHDL/Verilog RTL netlist and SystemC files. This makefile resolves to the *Verify_rtl_<hdl_ext>_<sim_tool>.mk* file in the *<solution_dir>/scverify* directory.

- **Mapped <hdl_format> output ‘rtl.<hdl_ext>’ vs <input_source_code_format>**

Runs the Precision RTL Synthesis tool and then simulates the original design vs. the technology mapped netlist generated by Precision. This makefile resolves to the *Verify_mapped_<hdl_ext>_<sim_tool>.mk* file in the *<solution_dir>/scverify* directory. For configuration information, see “[Accelerated FPGA Devices and Verification](#)” on page 523.

- **Gate <hdl_format> output ‘rtl.<hdl_ext>’ vs <input_source_code_format>**
Runs Precision and Xilinx place and route (PNR) and then simulates the original design versus the Xilinx post-PNR netlist. This makefile resolves to the *Verify_gate_<hdl_ext>_<sim_tool>.mk* file in the *<solution_dir>/scverify* directory.

The valid targets for each makefile are:

<code>simgui</code>	Compile and Execute the simulation using the interactive mode of the simulator (if available)
<code>sim</code>	Compile and Execute the simulation using the batch mode of the simulator
<code>build</code>	Compile the models only
<code>clean</code>	Remove all compiled objects
<code>help</code>	Show this help text

Infrastructure Files Generated by SCVerify (C/C++)

SCVerify generates verification infrastructure files at various stages in the synthesis flow and saves them in the Output folder under SCVerify as shown in [Figure 9-5](#). These infrastructure files are only generated for C/C++ designs. For more information, see “[C/C++ Designs and SCVerify](#)” on page 516.

The verification infrastructure files are common to all simulation tools.

The names of the infrastructure files in the GUI are generic labels. To see the actual filename associated with a label, right-click on the label to display the popup menu. The filename is shown at the top of the menu. Select the **Properties** menu item to see more detailed information about the file.

The following infrastructure files are generated by SCVerify:

- **SystemC C++ Testbench:** This file resolves to the generated testbench source code file. The default file name is “mc_testbench.cpp” and is saved in the solution directory.
- **SystemC C++ Testbench Header:** This header file contains the SystemC declarations necessary to write the testbench. Since the testbench must drive both the original C++

and the SystemC wrapper, the flow creates the header to make the testbench easier to construct. The file is named “mc_testbench.h” and is saved in the solution directory.

- **SCVerify Top:** This is the SystemC code implementing the main modules in the top level of test infrastructure. In particular, the generate_reset, generate_sync, deadlock_watch and deadlock_notify threads are in this file. This file is named “scverify_top.cpp” and is saved in the solution directory.
- **SCVerify Top Class:** This header file contains the declaration of the top level of the infrastructure. This file is named “scverify_top.h” and is saved in the solution directory.
- **SystemC Interface Transactors:** This file defines transactors for memory interface component used in the verification flow. The file is named “mc_transactors.h” and resides in the “\$MGC_HOME/shared/include” directory.

Design Synchronization (C/C++)

Hierarchical designs must provide design synchronization for both synthesis and verification flows. SystemC defines design synchronization signals, however synthesis transforms C/C++ designs from sequential to parallel requiring that one of the following methods of data synchronization:

- **ac_channel** — Use the ac_channel class in your source code to explicitly describe the synchronization between hierarchy blocks. For more information, see “[Inferring Hierarchy in Sequential C Code](#)” in the *Catapult C Synthesis C++ to Hardware Concepts* manual. This method ensures that every transaction on the IO is the same between the C/C++ and RTL. You must also map to an interface component with synchronization, either an _en or _wait IO component. For more information, see “[Basic I/O Components](#)” on page 288.
- **Transaction Done Signal** — Enable the Transaction Done Signal in Catapult to add one or more handshaking signals to the interface of all processes in the design. The handshaking signals go high for one cycle at the completion of an I/O transaction to indicate the last read/write on the inputs/outputs. For more information, see “[Setting Up the Interface Control Signals](#)” on page 98.
- **Absolute time** — Set the exact initialization interval (II) and latency constraints on the design and make sure they are consistent in SCVerify.

Note

 For designs that use reset behavior longer than a single clock cycle (such as initializing all of the elements in a local array variable), the synchronization signals must be delayed for the duration of the initialization action. In this case, you must use the Transaction Done Signal for verification. For more information, see “[Setting Up the Interface Control Signals](#)” on page 98.

Preparing a Testbench for SCVerify (C/C++)

Your C/C++ testbench must be extended to work with the SCVerify flow. In addition to executing the original design, it must simulate the RTL netlist and utilize the SCVerify infrastructure files to capture the input stimuli, call the original function to calculate the golden output values, and capture the outputs for comparison.

Use this procedure to modify your C/C++ testbench for use with SCVerify.

Prerequisites

- C/C++ testbench that proves the original design source. You can use SCVerify to generate a template testbench for your design. See “[Generating a Random Vector Testbench \(C/C++\)](#)” on page 541.

Procedure

1. Include the auto-generated testbench header *mc_testbench.h* at the top of the testbench. For example:

```
#include "mc_testbench.h"
```

The testbench header file provides functions that initialize variables and #ifdef blocks, so the testbench can still be used outside of the SCVerify flow. If you are using SystemC data types, you should include the SystemC header file just before the *mc_testbench.h* header file.

2. Insert calls to capture the input stimuli for each input port.

SCVerify automatically creates special functions for capturing data values, one function for each input and output. These functions can be found in the *mc_testbench.h* file with names derived from the formal name of the function argument to be captured. For example, the input to the function *idx* is captured using the auto-generated function named *capture_idx()*. The capture function has the same data type as the original function argument and handles the details of capturing both scalar values and array structures. Insert the call to each appropriate *capture_X()* function, passing in the same variable that is passed to the original function.

3. Insert calls to capture the golden output value from the original C/C++ function.

Just as with the input function arguments in Step 2, special *capture_X()* functions are created for capturing the outputs. In [Example 9-4](#), the correct golden value returned by the original C/C++ function is captured and placed in the FIFO by calling:

```
capture_idx_cubed(&s_idx_cubed);
```

This must be done after calling the original function. If you passed a pointer to the variable when calling the original function, you must also pass the pointer to the *capture_X()* function.

If your design returns a value from the top function call, that output value should be captured as well. The capture function is derived from the name of the function as:

```
capture_<function name>_out()
```

Results

C/C++ testbench is extended and ready for use with SCVerify flow.

Related Topics

[Capturing INOUT Values \(C/C++\)](#)
[Design and Testbench Example \(C/C++\)](#)

[C/C++ Designs and SCVerify](#)

Design and Testbench Example (C/C++)

[Example 9-2](#) defines a simple design containing the *cubed* function. The *cubed* function computes the cubed result of the input *idx*, and returns the result in the output variable *idx_cubed*. The function is in a file named: *cubed.cpp*.

Example 9-1. *cubed* Function

```
#include "cubed.h"

#pragma hls_design top
void cubed( int idx, int *idx_cubed)
{
    *idx_cubed = idx * idx * idx ;
}
```

[Example 9-2](#) defines a testbench to test the *cubed* function. The testbench exercises the *cubed* function over five iterations and reports the output for each iteration. The testbench is in a file named: *cubed_testbench.cpp*.

Example 9-2. Testbench for *cubed* Function

```
// Include some standard headers for doing IO
#include <iostream>
#include <fstream>
// Include the SystemC standard header
#include <systemc.h>

// Include the C/C++ function header
#include "cubed.h"

int main(int argc, char *argv[])
{
    int s_idx, s_idx_cubed;
    // Test simuli. Five iterations.
```

```
for (s_idx = 0; s_idx < 5; s_idx++) {  
    cubed(s_idx,&s_idx_cubed);  
  
    // Generate some output  
    cout << " idx = " << s_idx  
        << " idx_cubed = " << s_idx_cubed  
        << endl;  
}  
}
```

[Example 9-3](#) shows the output values from the *cubed* testbench.

Example 9-3. Output of the *cubed* Testbench

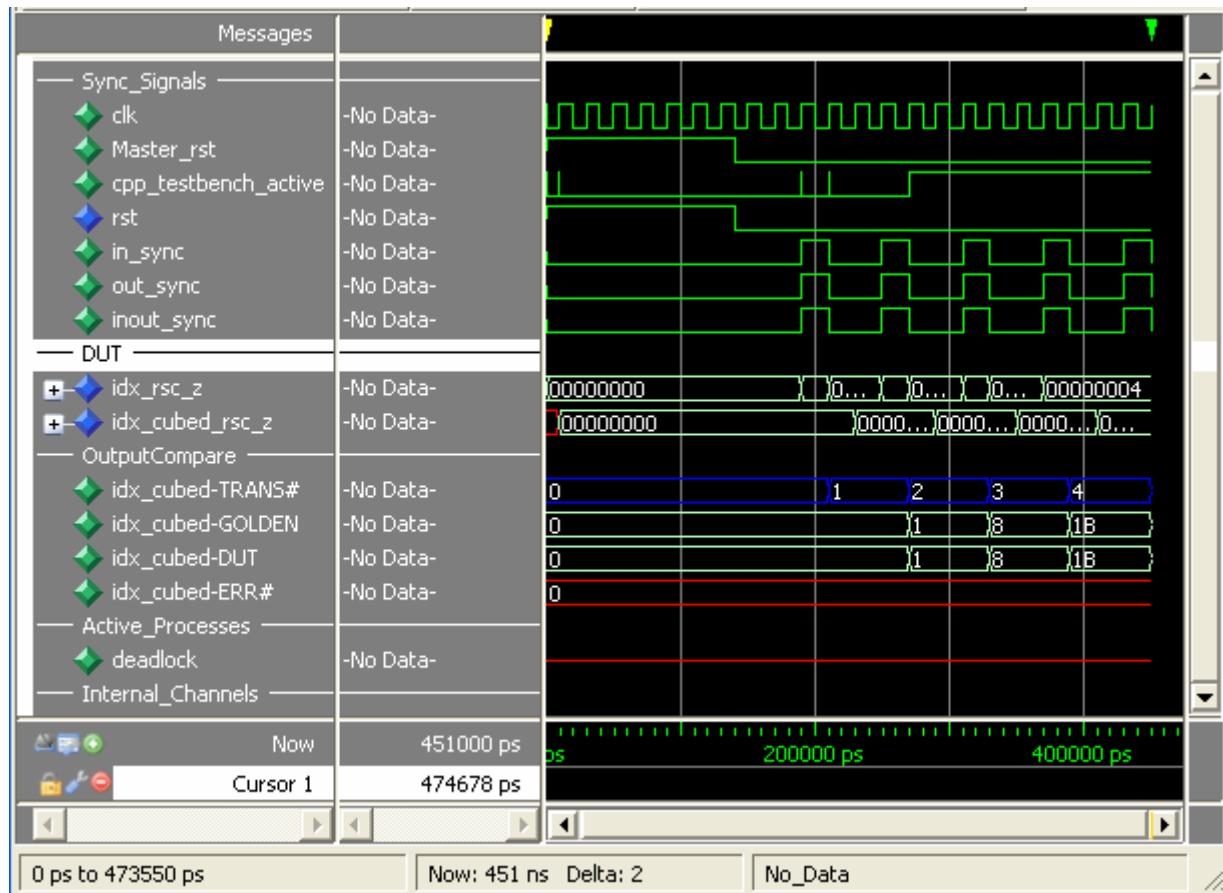
```
idx = 0 idx_cubed = 0  
idx = 1 idx_cubed = 1  
idx = 2 idx_cubed = 8  
idx = 3 idx_cubed = 27
```

[Example 9-4](#) shows the *cubed* testbench modified for the SCVerify flow with changes highlighted in red.

Example 9-4. Testbench for *cubed* Function extended for SCVerify

```
// Include some standard headers for doing IO  
#include <iostream>  
#include <fstream>  
  
// Include the C/C++ function header  
#include "cubed.h"  
  
// Include the SCVerify header  
#include "mc_testbench.h"  
  
CCS_MAIN(int argc, char *argv[])  
{  
    int s_idx, s_idx_cubed;  
    // Test simuli. Five iterations.  
    for (s_idx = 0; s_idx < 5; s_idx++) {  
        CCS DESIGN(cubed) (s_idx,&s_idx_cubed);  
  
        // Generate some output  
        std::cout << " idx = " << s_idx  
            << " idx_cubed = " << s_idx_cubed  
            << std::endl;  
    }  
    CCS RETURN(0);  
}
```

[Example 9-5](#) shows the *cubed* testbench output in the waveform window in ModelSim. For information on setting up the ModelSim session, see [“ModelSim Flow Options”](#) on page 199.

Example 9-5. *cubed* Testbench Waveform with 5 Iterations in ModelSim

In the wave window, the clock and reset signals are placed at the top, followed by the sync signals generated by the *generate_sync* thread. Next are the signals on the interface of the design under test (DUT). The final block of signals are the inputs to the comparators – the *GOLDEN* values coming from the original function call and the *DUT* values coming from the Catapult DUT. The *TRANS#* values show when each transaction occurs. The transaction periods show each time the cubed function begins and ends. The *ERR#* values identify the transaction number in which an error occurs. There are no errors in [Example 9-5](#).

From this simulation waveform, you can see the reset is asserted at 40 ns (2 clock cycles). Then on the fifth negative edge of the clock the input value (*idx_rsc_z*) is read. On the seventh edge the output (*idx_cubed_rsc_z*) is written and sampled (the *out_sync* signal is asserted).

In the example above, the *in_sync* signal is asserted at the same instant as the *out_sync*. This means that the testbench advances the next set of input stimuli into the FIFOs for the design to read as soon as the previous result is output.

The following text appears in the simulation transcript indicating that the DUT outputs matched the golden outputs:

```
run -all
# idx = 0 idx_cubed = 0
# idx = 1 idx_cubed = 1
# idx = 2 idx_cubed = 8
# idx = 3 idx_cubed = 27
# idx = 4 idx_cubed = 64
# Info: Execution of user-supplied C++ testbench 'main()' has completed
#
# Info: Collecting data completed
#     captured 5 values of idx
#     captured 5 values of idx_cubed
# Info: scverify_top/user_tb: Simulation completed
#
# Checking results
# 'idx_cubed'
#     capture count      = 5
#     comparison count   = 5
#     ignore count       = 0
#     error count        = 0
#     stuck in dut fifo = 0
#     stuck in golden fifo = 0
#
#
# Info: Simulation PASSED @ 350001 ps
```

Capturing INOUT Values (C/C++)

If the design being verified contains function arguments that Catapult infers as INOUT ports, then the testbench must capture the *input* value before calling the original function and then capture the *output* value after calling the original function. The SCVerify flow generates a separate function for capturing the input value. For example, if the original function is declared as:

```
void my_function(int *my_inout_var);
```

Then the testbench should capture the before and after values as follows:

```
int tb_my_inout_var = 5;
capture_my_inout_var_IN(&tb_my_inout_var);
my_function(&tb_my_inout_var);
capture_my_inout_var(&tb_my_inout_var);
```

For more information on INOUT ports, see the “[Design Interface](#)” section of the *Catapult C Synthesis C++ to Hardware Concepts* guide.

SCVerify TestBench Transaction Controls (C/C++)

The infrastructure testbench SCVerify generates for C/C++ design verification ([Figure 9-1](#)) defines a set of static variables. You can use these static variables to specify how transactions on individual interface variables are processed. Controls are per field of the formal arguments on the function interface.

The general form of the controls is:

```
testbench::<field_name>_<control_name>=<value>
```

where <field_name> is the name of an interface variable in the original C++ design, and <control_name> is one of the controls listed in [Table 9-2](#). Since these static variables are only defined in the SCVerify environment, you should enclose the controls with CCS_SCVERIFY precompiler directive so that your original testbench still compiles properly. For example:

```
#ifdef CCS_SCVERIFY
    testbench::<field_name>_<control_name>=<value>
#endif
```

For an example of how field names are derived from structs/classes, consider the class “complex” below:

```
class complex { int real; int imag }
...
void my_func (complex arg1, ... )
```

The resulting field names are *arg1_real* and *arg1_imag* for each *complex* type argument.

Table 9-2. SCVerify Testbench Transaction Controls

Control	Description
<u>_array_comp_first and _array_comp_last</u>	Specifies the range of array elements to compare.
<u>_ignore</u>	Ignores specified input or output values.
<u>_output_mask and _use_mask</u>	Masks a specified bit pattern when comparing output values.
<u>_skip</u>	Skips a non-channel interface transaction.
<u>_wait_ctrl.cycles</u>	Stalls an interface component for a specified number of cycles.
<u>_wait_ctrl.mode</u>	Sets the operating mode of the <u>_wait_ctrl.cycles</u> control.
<u>_wait_ctrl.elements</u>	Controls which elements of a streamed array are stalled.
<u>_wait_ctrl.interval</u>	Specifies the interval between waits applied by <u>_wait_ctrl.elements</u> .

[_array_comp_first and _array_comp_last](#)

Specifies the range of array elements compared for each output transaction. Only array elements within the range are compared and the remainder are ignored. These flags are reset to -1 at the beginning of each testbench iteration to indicate that all array elements need to be compared.

```
<field>_array_comp_first = <integer>;
<field>_array_comp_last = <integer>;
```

These controls help when verifying designs that only write to a subset of the output array on any given iteration. If the testbench data for a given iteration causes a smaller subset of the output array elements to be written, and the testbench modifies the contents of the output array on each iteration, then the elements outside of that subset range would likely fail comparison with the output of the design under test. You can use these controls to restrict the range of elements compared to the range that you expect the design to modify - ignoring all other elements.

The following testbench excerpt limits the range of elements to be compared during the fourth iteration of the loop. In all other iterations, every array element is compared.

```
int lb; // lowerbound array index
int ub; // upperbound array index
for (int iteration = 0; iteration < 10; ++iteration) {
    // clear output array each iteration
    for (int i=0; i < NUM_OF_ELEMS; i++) out_arr[i] = 0;

    if (iteration == 4) {
        lb = 5; ub = NUM_OF_ELEMS/2;
#ifdef CCS_SCVERIFY
        testbench::out_arr_array_comp_first = lb;
        testbench::out_arr_array_comp_last = ub;
#endif
    } else {
        lb = 0; ub = NUM_OF_ELEMS;
    }
    ...
}
```

The SCVerify flow issues a message in the simulation transcript when these controls are used. For example, if the NUM_OF_ELEMS in the excerpt above has a value of 40, the following message displays listing the original range and the restricted range.

```
# Info: scverify_top/user_tb/out_arr_comp: Array element range comparison
limited from [0...39] to [5...19]
```

ignore

Ignores values on a specified port for one transaction. Use this control when certain port values should not be tested because they are known to be invalid. Setting *_ignore* to true causes the values on the specified port to be ignored for the subsequent transaction only. All *<field>_ignore* flags are internally reset to false at the beginning of each testbench iteration (i.e. after every call to `exec_<design>`).

If outputs are ignored, the golden value and the RTL value are not compared. If inputs are ignored, the transaction event is not initiated on the input for a value. Ignoring inputs is useful when the input interface is a multicycle bus transaction that only needs to be performed once in the simulation to configure some registers.

The following example ignores the value of “output” during the first ten iterations only:

```
for (i=0; i < 100; i++) {
```

```
...
if (i < 10) testbench::output_ignore=true;
testbench::exec_MyDesign(&input,coeffs,&output);
...
}
```

_output_mask and _use_mask

These two controls work in conjunction to apply a bitmask to both the golden values and the values from the Catapult design under test (DUT). The `_output_mask` control specifies a bitmask to be applied, and the “`_use_mask`” control is a boolean flag that controls when the bitmask is to be applied.

```
<field>_output_mask = <integer>
<field>_use_mask = <true_or_false>
```

The C/C++ datatype for `_output_mask` should match the datatype of the formal arg to the function that was synthesized. In the case of an array, it should match the datatype of the array element. Applying an “`_output_mask`” to an array will apply the mask to every element of the array. The summary statistics at the end of simulation will report the number of masked comparisons.

Unlike the other testbench controls, `_output_mask` is NOT cleared on each iteration of the testbench. The reason is that SCVerify does not necessarily know how to set an arbitrary datatype to all ones (meaning no bitmasking). For this reason, use the “`_use_mask`” control to specify when “`_output_mask`” is to be applied. The “`_use_mask`” control is reset to false on each testbench iteration. If you set it to true for a given iteration, the value of “`<field_name>_output_mask`” is applied to the golden and DUT values.

Example using `_output_mask` and `_use_mask`:

```
for ( i = 0; i < NUM_TAPS; i++ ) {
    ...
#ifndef CCS_SCVERIFY
    if (i == 2) { outvar_output_mask = 9; outvar_use_mask = true; }
#endif
    CCS_DESIGN(fir_filter) (&input,coeffs,&outvar);
    ...
}
```

skip

Specifies that a transaction for a non-channel interface should be skipped for this iteration. This control is necessary when the interface of the C++ function being synthesized has both ac_channel and non-ac_channel interfaces. The following code example demonstrates the need for this control.

```
#include <ac_int.h>
#include <ac_channel.h>
```

```
#pragma hls_design top
void design ( ac_channel <ac_int<5> > &din, ac_int<5> *dout ) {
    if ( din.available(2) ) {
        *dout = din.read() + din.read();
    }
}
```

In this design, the input “din” is an ac_channel while the output “dout” is not. Since the SCVerify simulation assumes that each call to the function is a transaction, if the function were called 10 times by the C++ testbench it would appear that “dout” would have 10 transactions. However, since the write to “dout” is enclosed in the “din.available(2)” branch, then there will be actually only $10 / 2 = 5$ real transactions on “dout”. The testbench control _skip allows you to mark certain outputs as not having any transaction on a given iteration of the testbench. Note that this is different from the “_ignore” option which accepts the output transaction but simply ignores the result of the comparison. Also note that since the information needed to decide whether an output transaction should be skipped or not is actually in the C++ design itself, this testbench control must be accessible from anywhere in the design scope. Therefore, this control is available as a global function call “mc_testbench_<field>_skip()”. The modified design and testbench are shown below:

```
#include <ac_int.h>
#include <ac_channel.h>
#ifndef CCS_SCVERIFY
// Include this file to gain access to the global function
"mc_testbench_dout_skip()"
#include "mc_scverify.h"
#endif

#pragma hls_design top
void design ( ac_channel <ac_int<5> > &din, ac_int<5> *dout ) {
    if ( din.available(2) ) {
        *dout = din.read() + din.read();
    }
#ifndef CCS_SCVERIFY
    else {
        // since dout was not actually written in this call to the design, tell
        // the testbench to skip this output transaction
        mc_testbench_dout_skip(true);
    }
#endif
}

// C++ Testbench
CCS_MAIN(int argc, char *argv[]) {
    ac_channel<ac_int<5> > din;
    ac_int<5> dout;

    for ( int i = 0; i < 10; i++ ) {
        din.write(i);
        CCS_DESIGN(design)(din,&dout);
    }
    CCS_RETURN(0);
}
```

_wait_ctrl.cycles

Controls how long to de-assert the wait handshake signal (“vz”). Cycles are counted from when Catapult starts to attempt to read the input (or write the output), i.e. when Catapult asserts the request handshake signal (“lz”). The behavior of this control can be modified by the [_wait_ctrl.mode](#) control below.

```
<field>_wait_ctrl.cycles = <integer>
```

_wait_ctrl.mode

Controls whether the number of cycles specified in the [_wait_ctrl.cycles](#) control is to be a UNIFORM (fixed) value or a RANDOM value between 0 and [_wait_ctrl.cycles](#). The default is UNIFORM. This flag is internally reset to UNIFORM at the beginning of each testbench iteration. You must include the typename prefix (“mc_wait_ctrl”) in order to select the enum. For example:

```
<field>_wait_ctrl.mode = mc_wait_ctrl::UNIFORM
```

or

```
<field>_wait_ctrl.mode = mc_wait_ctrl::RANDOM
```

_wait_ctrl.elements

This control is for streamed arrays or ac_channels. It takes one of four values:

INITIAL	Delay applied only on the first element of the transaction. (Default)
ALL	Delay applied on every element in the transaction.
FIXED_INTERVAL	Delay applied to every “ _wait_ctrl.interval ” elements.
RANDOM_INTERVAL	Similar to FIXED_INTERVAL except that the interval between delayed elements is randomly chosen (up to the “ _wait_ctrl.interval ” value).

This control is internally reset to INITIAL at the beginning of each testbench iteration. You must include the typename prefix (“mc_wait_ctrl”) in order to select the enum. For example:

```
<field>_wait_ctrl.elements = mc_wait_ctrl::ALL
```

_wait_ctrl.interval

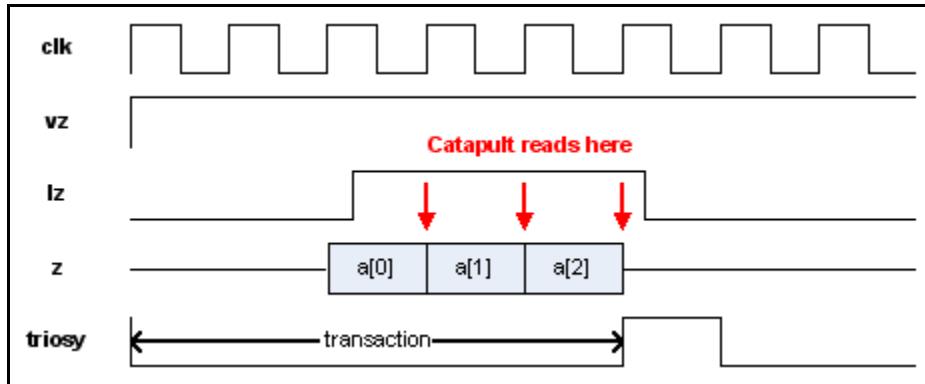
Specifies the interval between streamed element numbers to which the delay will be applied (if the “[_wait_ctrl.elements](#)” control is FIXED_INTERVAL or RANDOM_INTERVAL).

```
<field>_wait_ctrl.elements = mc_wait_ctrl::FIXED_INTERVAL
<field>_wait_ctrl.interval = <integer>
```

Wait Control Examples

The following examples show how the various wait controls are used in the testbench. The waveforms in the figures illustrate the effects of the controls. To begin, Figure 9-6 shows the waveform without any wait controls.

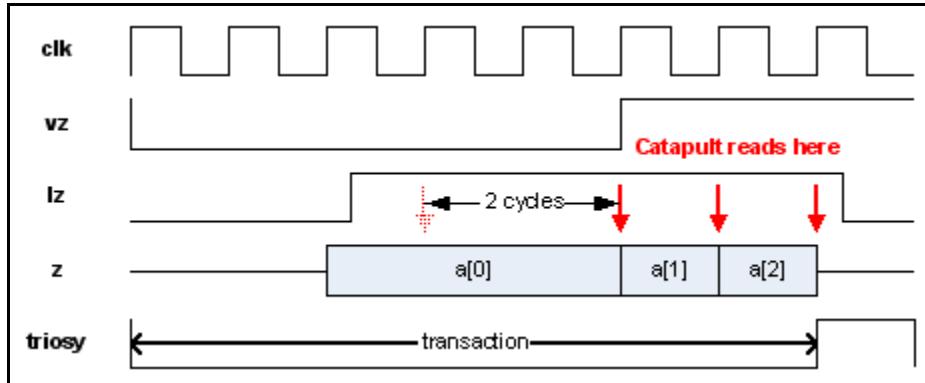
Figure 9-6. No Input Stalling



Example 1: Specifies an initial delay of 2 cycles at start of the transaction. (Figure 9-7)

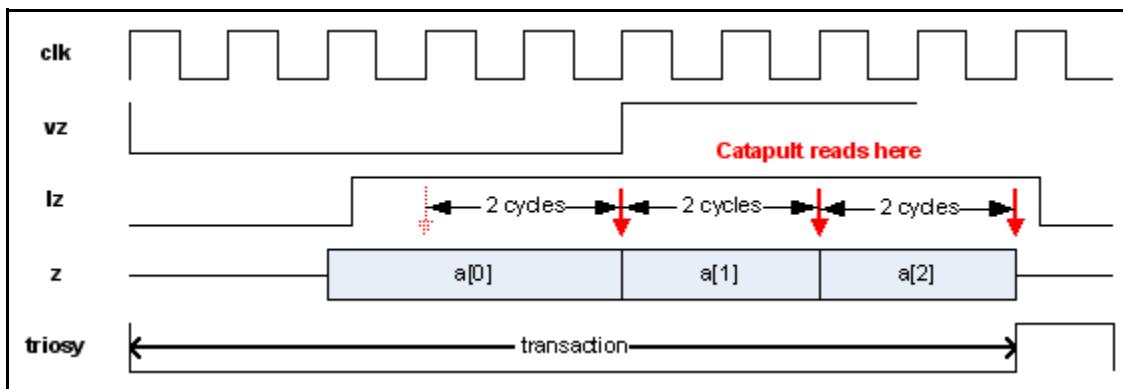
```
testbench::varA_wait_ctrl.cycles = 2;
testbench::varA_wait_ctrl.elements = mc_wait_ctrl::INITIAL;
```

Figure 9-7. Stalls Read of the First Element by 2 Cycles



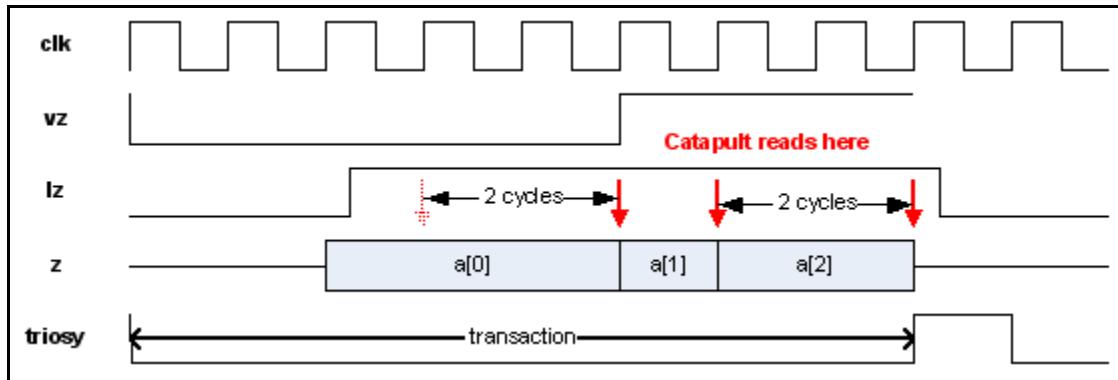
Example 2: Specifies a uniform delay of 2 cycles on each element streamed in. (Figure 9-8)

```
testbench::varA_wait_ctrl.cycles = 2;
testbench::varA_wait_ctrl.elements = mc_wait_ctrl::ALL;
testbench::varA_wait_ctrl.mode = mc_wait_ctrl::UNIFORM;
```

Figure 9-8. Stalls All Elements by 2 Cycles

Example 3: Specifies a delay of 2 cycles on the first element and every second element after that. (Figure 9-9)

```
testbench::varA_ctrl.cycles = 2;
testbench::varA_ctrl.elements = mc_wait_ctrl::FIXED_INTERVAL;
testbench::varA_ctrl.interval = 2;
```

Figure 9-9. Stalls Read of Every Other Element by 2 Cycles

Generating a Random Vector Testbench (C/C++)

This procedure generates a basic random vector testbench file based on the C/C++ input source code. The random vector testbench file is a ready-to-compile C/C++ file that contains the necessary functions and structure to run in the SCVerify flow and generate random test vectors. Use this procedure if you need to generate a basic testbench to verify your design.

Prerequisites

- Design has compiled successfully in Catapult

Procedure

1. Open the project in Catapult.

2. Enter the *flow run* command to generate the random vector testbench. For example:

```
flow run SCVerify/gen_testbench_template [filename] [iterations]
```

A testbench file is created in the project directory. If no filename is specified, the file is named *template_testbench.cpp*. The iterations argument specifies the number of test iterations. The default number of iterations is 1.

3. Modify the testbench as necessary and rename the file. You may need to constrain the ranges of the random data. For more information, see “[Random Vector Testbench Structure \(C/C++\)](#)” on page 542.
4. Enable the SCverify flow. For more information, see “[Setting Up SCVerify](#)” on page 518.

Results

A testbench is created that can be used with the SCVerify flow for design verification.

Related Topics

[Preparing a Testbench for SCVerify \(C/C++\)](#)

Random Vector Testbench Structure (C/C++)

The random vector testbench file generated by Catapult contains calls to the capture functions for each of the inputs and outputs in the design. The *mc_scverify.h* file provides functions to initialize variables and *#ifdef* blocks to enable the testbench to be used outside of the SCVerify flow. The following example shows the generated random vector testbench file for the *cubed.cpp* design shown in [Example 9-1](#).

```
// -----
// SCVerify Testbench Template
// -----
//
// Place system includes here
#include <iostream>

#include "mc_scverify.h"
#include "../cubed.h"

// some functions for generating random test vectors

#include "mc_random.h"

static void throw_dice_for_idx(int &idx) {
    mc_random(idx);
}

// some functions initializing values to zero
static void init_idx(int &idx) {
```

```
    idx = 0;
}

static void init_idx_cubed(int &idx_cubed) {
    idx_cubed = 0;
}

// -----
// Start of MAIN

CCS_MAIN(int argc, char *argv)
{
    // Place local testbench variables here
    int idx;
    int idx_cubed;
    int (*_idx_cubed) = &idx_cubed;

    // Initialize local variables to zero
    init_idx(idx);
    init_idx_cubed(idx_cubed);

    // Main test iterations start here
    for (int iteration = 0; iteration < 5; ++iteration) {

        // Set test values for this iteration
        throw_dice_for_idx(idx);

        // Progress indicator
        std::cout << "Iteration: " << iteration << std::endl;

        // Call original function and capture data
        CCS_DESIGN(cubed)(idx, _idx_cubed);
    }
    // Return success
    CCS_RETURN(0);
}
```

Non-Blocking Reads and Verification (C/C++)

The SCVerify flow provides the Trace/Replay option to accommodate the special challenges of verifying designs that use non-blocking reads. The non-blocking behavior enables C-to-RTL synthesis of designs in which “hierarchical functions” communicate through multiple channels (ac_channels). These designs can imply the behavior of concurrent (or multi-threaded) systems and will result in RTL with concurrency.

Design verification is made difficult when there is concurrency in the HDL design because concurrency can produce any number of different, but valid, results that differ from the (zero delay) C++ code's results. The challenge is to test the assumption that the specific HDL results of the synthesized design are a valid output of the C/C++ design when the temporal effects of the HDL simulation are taken into consideration.

The SCVerify flow now includes the new “Trace and Replay” approach to verify concurrent designs. During HDL simulation this approach *traces* the communication channels through which the hierarchical blocks of the design pass data, and then *replays* that data as input to the algorithmic functions of the C/C++ design, thus validating each block's discrete execution with “stimulus” from the corresponding event from the HDL run.

In other words, the trace/replay method can simulate how data moves from one C/C++ block to another with the same values, and in the same order, as in the synthesized RTL design even when the order of module evaluation differs from the HDL. When this is done on a correctly synthesized design, the outputs of the C/C++ and RTL will match. In addition, this method also makes it possible to debug problems with the non-blocking reads using C/C++.

Known Issues with SCVerify Replay Verification (C/C++)

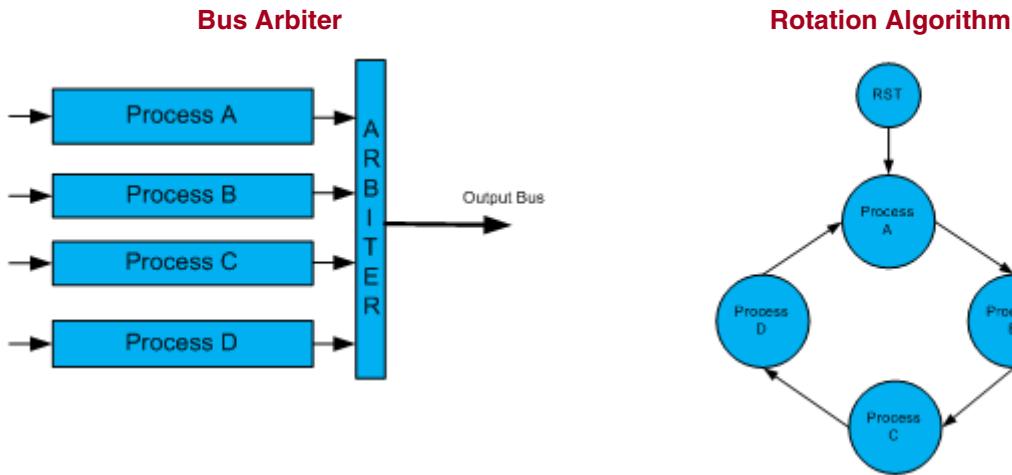
The following limitations and restrictions apply to this release:

- Overriding the methods of the ac_channel object may interfere with verification.
- In test designs where the testbench state is affected by the output state of the design under test (DUT), the untimed sequential C/C++ DUT output state is used. This may differ from the HDL/replay C/C++ DUT output state, which may result in unexpected and/or undesirable test behavior.
- A reset following data input will end the test. (Consistent with traditional SCVerify).
- As with SCVerify, false positive results may occur due to insufficient stimulus from the test bench or coincidental matching of C/C++/HDL data from non-equivalent logic.
- At this time, input arguments to the top level test function cannot be structs or arrays. Supported data types are limited to primitives (int, uint, bool, etc.), ac_channel, SC_INT and SC_FIXED. The arguments can be passed in as pointers or as references (&).
- SCVerify Testbench Transaction Controls (*<field>_array_comp_first*, *<field>_wait_ctrl.elements*, and so on) will not work correctly in this version of the Replay System. Designs that require such iteration specific controls will not work in this system.
- If ac_channels are members of C++ objects, the objects can only be instanced once.
- If the C++ test runs for a fixed number of iterations you may not get all of the result out by the time the test is done. This is because the RTL has output its results over a “longer” period, meaning more block evaluations in this clockless environment. This is not a failed test condition. You may want to run your test a bit longer with redundant inputs.

Trace/Replay Verification Example (C/C++)

This example arbiter design contains four processes that are all trying to write to a shared resource, the output bus. The arbiter function implements a rotation-based (or round robin) arbitration algorithm to read from each process, one after the other. Using non-blocking reads, if no data is available when the read is attempted, the arbiter moves on to the next process. Figure 9-10 illustrates the major features of the design.

Figure 9-10. Design Features



The C++ code implementation is shown in Example 9-6. In, `arbiter_example.cpp`, blocks A,B,C,D implement the client processes trying to access the output bus. Each of these blocks reads its input ac_channel using the blocking `read()` method, then writes the modified data to its output FIFO on each successful read.

The arbiter block first uses the non-blocking `size()` method to check for data availability in the output FIFO of each process block. Second, for those processes that have data in their FIFOs, the arbiter selects one process to be served based on the rotation arbitration algorithm. Third, it reads the selected process FIFO (blocking method) and writes the data to the bus. Finally, the arbiter updates the serving priority based on rotation algorithm.

Example 9-6. Arbiter Example Using Non-Blocking size() Method

```
#include <ac_channel.h>
#include <ac_int.h>
#include "arbiter_example.h"

#pragma hls_design
void block_A(ac_channel<uint8> &in, uint8 &offset, ac_channel<uint8> &out)
{
    if (in.available(1))
        out.write(in.read() + offset);
}

#pragma hls_design
```

```
void block_B(ac_channel<uint8> &in, uint8 &offset, ac_channel<uint8> &out)
{
    if (in.available(1))
        out.write(in.read() + offset);
}

#pragma hls_design
void block_C(ac_channel<uint8> &in, uint8 &offset, ac_channel<uint8> &out)
{
    uint8 acc;
    if (in.available(1))
    {
        acc = in.read();
        for (int i=0;i<4;i++)
            acc = acc + i + offset;
        out.write(acc);
    }
}

#pragma hls_design
void block_D(ac_channel<uint8> &in, uint8 &offset, ac_channel<uint8> &out)
{
    if (in.available(1))
        out.write(in.read() + offset);
}

#pragma hls_design
void arbiter(ac_channel<uint8> &pri0,
            ac_channel<uint8> &pri1,
            ac_channel<uint8> &pri2,
            ac_channel<uint8> &pri3,
            ac_channel<uint8> &out)
{
    static uint4 priority = 1;
    ac_int<4,false> p_rdy;

    /* Non-blocking "size()" function is used here to */
    /* check for output data in all process block FIFOs */
    p_rdy[0] = pri0.size() > 0;
    p_rdy[1] = pri1.size() > 0;
    p_rdy[2] = pri2.size() > 0;
    p_rdy[3] = pri3.size() > 0;

    uint4 temp_priority=priority;
    for (int i=0;i<4;i++){
        if (p_rdy & temp_priority){
            p_rdy = p_rdy & temp_priority;
            break;
        }
        uint1 temp = temp_priority[3];
        temp_priority = temp_priority << 1;
        temp_priority[0] = temp;
    }

    /* Arbiter read data from the selected process block */
    /* and write it to the bus (arbiter output FIFO) */
    if (p_rdy[0])
        { out.write(pri0.read()); }
```

```
else if (p_rdy[1])
    { out.write(pri1.read()); }
else if (p_rdy[2])
    { out.write(pri2.read()); }
else if (p_rdy[3])
    { out.write(pri3.read()); }

/* Update priority setting for next cycle. */
if (p_rdy) {
    priority = temp_priority;
    uint1 temp = priority[3];
    priority <= 1;
    priority[0] = temp;
}
}

#pragma hls_design top
void top( uint8 & offset1,
          uint8 & offset2,
          uint8 & offset3,
          uint8 & offset4,
          ac_channel <uint8> &in_A,
          ac_channel <uint8> &in_B,
          ac_channel <uint8> &in_C,
          ac_channel <uint8> &in_D,
          ac_channel <uint8> &out
)
{
    // The following line must be modified so that each
    // channel variable is wrapped in the AC_CHANCTOR macro.
    // ORIGINAL CODE: static ac_channel<uint8> pri0, pri1, pri2, pri3;

    // MODIFIED CODE:
    static ac_channel<uint8> AC_CHANCTOR(pri0), AC_CHANCTOR(pri1),
                                AC_CHANCTOR(pri2), AC_CHANCTOR(pri3);
    block_A(in_A, offset1, pri0);
    block_B(in_B, offset2, pri1);
    block_C(in_C, offset3, pri2);
    block_D(in_D, offset4, pri3);
    arbiter(pri0,pri1,pri2,pri3,out);
}
```

Example 9-7. Arbiter Example Header File

```
void top( uint8 & offset1, uint8 & offset2,
          uint8 & offset3, uint8 & offset4,
          ac_channel <uint8> &in_A,
          ac_channel <uint8> &in_B,
          ac_channel <uint8> &in_C,
          ac_channel <uint8> &in_D,
          ac_channel <uint8> &out
);
```

Using the following testbench and directives setup script (Examples 9-8 and 9-9), you can verify the arbiter design using the SCVerify flow.

Example 9-8. Testbench for Arbiter Example Design

```
#include "mc_scverify.h"
#include <ac_channel.h>
#include <ac_int.h>
#include "arbiter_example.h"

CCS_MAIN(int argc, char **argv) {
    static ac_channel<uint8> in_A, in_B, in_C, in_D;
    static ac_channel<uint8> out;

    uint8 tmp;
    uint8 offset = 0;
    for (int i=0; i<10;i++)
    {
        in_A.write(rand() % 255);
        in_B.write(rand() % 255);
        in_C.write(rand() % 255);
        in_D.write(rand() % 255);

        CCS_DESIGN(top)(offset, offset, offset, offset, in_A, in_B, in_C,
in_D, out);

        while(out.available(1))
        {
            tmp = out.read();
            printf("out = %d\n",tmp.to_int());
        }
    }
    CCS_RETURN(0);
}
```

Example 9-9. Setup Script for Arbiter Example Design

```
project new
flow package require /SCVerify
options set Flows/SCVerify ENABLE_REPLY_VERIFICATION true
solution file add ./arbiter_example.cpp -type C++
solution file add ./tb_arbiter_example.cpp -type C++ -exclude true
go analyze
directive set -CLOCK_NAME clk
directive set -CLOCKS {clk {-CLOCK_PERIOD 10.0 -CLOCK_EDGE rising \
-CLOCK_HIGH_TIME 5.0 -RESET_SYNC_NAME rst -RESET_ASYNC_NAME arst_n \
-RESET_KIND sync -RESET_SYNC_ACTIVE high -RESET_ASYNC_ACTIVE low \
-ENABLE_NAME {} -ENABLE_ACTIVE high}}
directive set -TECHLIBS {{mgc_sample-065nm-dw_beh_dc.lib \
mgc_sample-065nm-dw_beh_dc}}
directive set -TRANSACTION_DONE_SIGNAL true
go compile

directive set /top/arbiter/main/for -UNROLL yes
directive set /top/arbiter/main -PIPELINE_INIT_INTERVAL 1
directive set /top/block_D/main -PIPELINE_INIT_INTERVAL 1
directive set /top/block_C/main -PIPELINE_INIT_INTERVAL 1
directive set /top/block_B/main -PIPELINE_INIT_INTERVAL 1
directive set /top/block_A/main -PIPELINE_INIT_INTERVAL 1
```

```

directive set /top/in_D:rsc -MAP_TO_MODULE mgc_ioport.mgc_in_wire_wait
directive set /top/in_C:rsc -MAP_TO_MODULE mgc_ioport.mgc_in_wire_wait
directive set /top/in_B:rsc -MAP_TO_MODULE mgc_ioport.mgc_in_wire_wait
directive set /top/in_A:rsc -MAP_TO_MODULE mgc_ioport.mgc_in_wire_wait
directive set /top/pri3:cns -FIFO_DEPTH 10
directive set /top/pri2:cns -FIFO_DEPTH 10
directive set /top/pri1:cns -FIFO_DEPTH 10
directive set /top/pri0:cns -FIFO_DEPTH 10
go architect
go extract

```

In [Example 9-6](#) on page 545, block_C writes to its output FIFO every 4th clock cycle instead of every clock cycle like blocks A, B and D. This hardware timing detail cannot be modelled in untimed C execution, so we must use the Trace/Replay option of the SCVerify flow.

The following steps explain the replay verification process.

1. The replay system must be able to uniquely identify all ac_channel instances in the design. Macros are defined in ac_channel.h for this purpose:
 - a. Use the AC_CHAN_CTOR macro to wrap each static ac_channel instance in the design. In our example design, we make the following change:

```

OLD: static ac_channel<uint8> prio0, pri1, pri2, pri3;

NEW: static ac_channel<uint8> AC_CHAN_CTOR(prio0),
      AC_CHAN_CTOR(pri1), AC_CHAN_CTOR(pri2),
      AC_CHAN_CTOR(pri3);

```

Do not wrap ac_channels constructed in the testbench and passed in as formal arguments.

- b. The AC_CHAN_CPP macro is used for ac_channel data members of a class. In this case, wrap the constructor for the data member in the class definition. This macro is not demonstrated in the arbiter design in [Example 9-6](#) on page 545. Instead, refer to the following RAM_controller class definition. You can see that all of interface ports are ac_channel data members of the class. In the RAM_controller constructor, those data members are wrapped in the AC_CHAN_CPP macro.

```

class RAM_controller {
    typedef ac_int<num_data_bits> RAMC_data;
    typedef ac_channel<RAMC_data> RAMC_chan_data;
    typedef ac_int<num_address_bits, false> RAMC_address;
    typedef ac_channel<RAMC_address> RAMC_chan_address;
    typedef ac_channel<ac_int<1, false>> RAMC_chan_control;

    // Interface channels - Should be private
    RAMC_chan_data d_port1_data_in, d_port1_data_out,
                    d_port2_data_in, d_port2_data_out, d_port3_data_in,
                    d_port3_data_out;
    RAMC_chan_address d_port1_addr, d_port2_addr, d_port3_addr;
    RAMC_chan_control d_port1_control, d_port2_control,
                       d_port3_control;
    ...
}

```

```
public:  
    RAM_controller() :  
        AC_CHAN_CPP(d_port1_data_in),  
        AC_CHAN_CPP(d_port1_data_out),  
        AC_CHAN_CPP(d_port2_data_in),  
        AC_CHAN_CPP(d_port2_data_out),  
        AC_CHAN_CPP(d_port3_data_in),  
        AC_CHAN_CPP(d_port3_data_out),  
        AC_CHAN_CPP(d_port1_addr),  
        AC_CHAN_CPP(d_port2_addr),  
        AC_CHAN_CPP(d_port3_addr),  
        AC_CHAN_CPP(d_port1_control),  
        AC_CHAN_CPP(d_port2_control),  
        AC_CHAN_CPP(d_port3_control)  
    {  
        priority = 0;  
        priority[0] = 1;  
        d_array = input_RAM;  
    }  
    ...  
}
```

2. Create a Catapult project and load the design.
3. Enable the SCVerify flow. The command syntax is:

```
flow package require /SCVerify
```

From the GUI, double-click on “SCVerify” in the Flow Manager window, then click the “Enable” button.

4. Enable the Trace/Replay flow option. The command syntax is:

```
flow package option set /SCVerify/ENABLE_REPLAY_VERIFICATION true
```

From the GUI, toggle the “Use Trace/Replay verification” option on the SCVerify flow options page. The options page can be accessed from the Flow Manager window by double-clicking on the SCVerify.

Be sure this option is turned on before the compile stage. If you want to turn it on or off after synthesis you must run “Clean,” “Regenerate Makefile” AND “Regenerate Replay Makefile” in the test’s verification menu.

5. Generate RTL and the SCVerify verification files. The Trace/Replay system will generate an SCVerify testbench that will output the trace file (trace_data.txt) in the output directory (e.g. <solution_dir>/scverify/rtl_vhdl_msim) when verification is run. It will also produce the Replay testbench and its Makefile (Verify_orig_cxx_replay.mk).
6. Run the verification scripts. As shown in Figure 9-11, this is a two pass process. First launch standard SCVerify script. If the design verifies correctly, you are done. If data comparison errors are found, the test report will indicate that and recommend that you run the Replay test. For example:

```
# Info: output 'out' had comparison errors
```

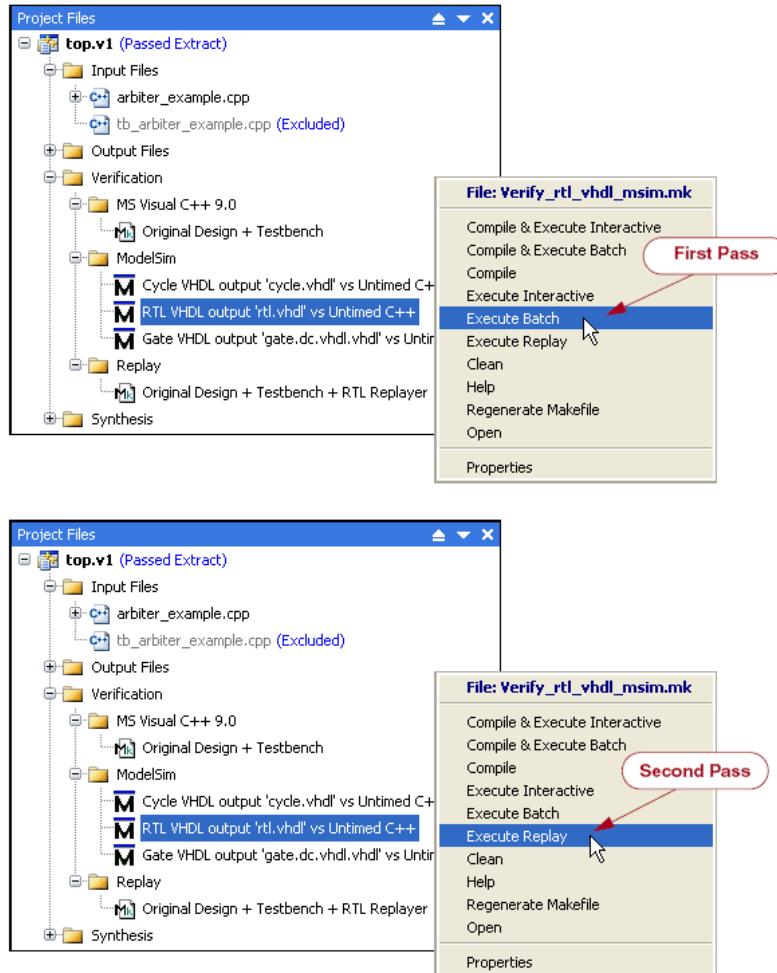
```

#
# Info: Simulation complete @ 156 ns with data mis-matches, REPLAY
test recommended.

```

To run the replay testbench use the “Execute Replay” popup menu item as shown in Figure 9-11. The testbench will run using the traced data and compare outputs with those traced. A report appears in the Catapult Transcript Window.

Figure 9-11. Trace/Replay Two Pass Verification



SCVerify Frequently Asked Questions

See also “[SCVerify Flow Troubleshooting](#)” on page 552.

How do I remove the SCVerify flow?

Answer: To remove the flow from the active solution, select the SCVerify flow in the **Flow Manager** window, and click the **Disable** button. Alternatively, you can enter the command “`flow package forget /SCVerify`”.

What is the proper setting for the MODEL_TECH environment variable?

During the “Generate RTL” task, Catapult issues an error message because it cannot find the ModelSim software even though ModelSim is installed.

Answer: On Windows, you do not need the MODEL_TECH variable because the location of ModelSim is stored in the Windows system registry after you run the tool one time. On UNIX platforms, you'll need to set your MODEL_TECH environment variable to point to the “bin” directory of the Modelsim installation. For example:

```
setenv MODEL_TECH /simtools/modelsim/6.4b/bin
```

Can Catapult support applications that use floating point math?

Answer: Catapult will not convert “float” or “double” into floating point hardware, they are converted to fixed point. The developer will need to write a floating point class to implement the portion of the floating point arithmetic that is required for the design.

Does Catapult support or generate divide functions?

Answer: Yes, Catapult fully supports divide and modulus operations for all C integer types and all supported bit-accurate integer types: ac_int, mc_bitvector, sc_int, sc_uint, sc_bigint and sc_bignum.

Catapult also fully supports divide operations for the bit-accurate fixed-point datatype ac_fixed and it supports the divide assign operation “/=” for the SystemC fixed-point datatype sc_fixed with some restrictions.

Division functions for ac_int, ac_fixed, sc_bigint, sc_bignum and sc_fixed are included in the Math packages “math/mgc_ac_math.h” and “math/mgc_sc_math.h” that are provided with Catapult. These functions allow exploring area vs. performance trade-offs for those functions. Additionally, the division functions for ac_fixed and sc_fixed provide full control of the desired precision and support all quantization and overflow modes.

SCVerify Flow Troubleshooting

This troubleshooting guide uses a decision tree format to analyze your situation and direct you to the appropriate solution. Begin by going to the section named “**START**” and find the problem description that best describes your situation. You will be guided to lower level sections containing more focused problem descriptions and/or a recommended solution.

START	553
Flow Execution Problems	554
No Output Files Generated	554
Flow Not Enabled	554

Flow Package Require Fails	555
Flow Messages in Transcript	555
Missing Testbench.cpp File	555
Error Streaming Without Handshake	556
Error Streaming Multiple Variables	556
ModelSim Launch.	556
ModelSim Compile Errors	557
“Capture” Function Undeclared	558
“Capture” Function Out of Scope.	558
Captured Port Optimized Away	558
Capturing a Pointer Value.	558
Warning Non-Debuggable Type.	559
Non-Public Data Members in Class on the Interface	559
Compiling with sc_fixed Types	559
Additional Compile Error Trouble	560
Picking Up STL Header From Working Directory	560
Overloading Window Macros/Typedefs.	561
ModelSim Link Errors	562
ModelSim Link Undefined Reference	562
ModelSim Link Multiple Definition.	562
ModelSim Design Load Errors.	562
Cannot Wave Non-Debuggable Objects.	563
Simulation Errors	563
Input FIFO Never Filled	564
Input FIFO Underflow	564
Output FIFO Overflow	564
Doubles/Sc_fixed Mismatch.	564
Simulation Mismatch	565

START

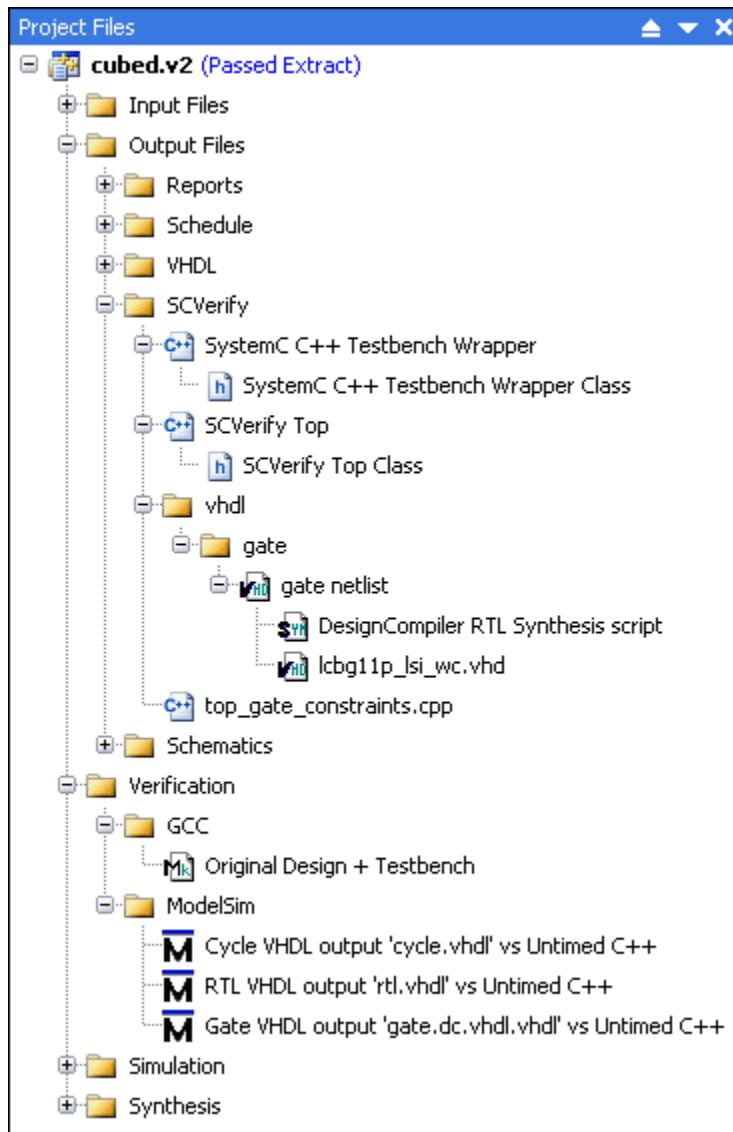
What is the nature of the problem you are having with the SCVerify flow:

- If you are not sure if the flow is being run for your solutions, go to [No Output Files Generated](#).
- If the flow is running (generating output files during RTL generation) but prints warnings or errors in the Catapult transcript, go to [Flow Messages in Transcript](#).
- If the flow is running (generating output files during RTL generation) but fails to launch ModelSim, go to [ModelSim Launch](#).
- If ModelSim launches correctly but then encounters compilation or simulation errors, go to [ModelSim Errors](#).

Flow Execution Problems

No Output Files Generated

Check to see if you have any of the SCVerify files in the **Verification** folder in the solution. Look for filenames like:



- If the files have never been present in any solution, go to [Flow Not Enabled](#).

Return to [START](#).

Flow Not Enabled

The SCVerify flow is probably not enabled. To enable the flow, select SCVerify in the Flow Manager window. See [“Setting Up SCVerify”](#) on page 518.

- If that command fails, go to [Flow Package Require Fails](#).

Back to [No Output Files Generated](#).

Return to [START](#).

Flow Package Require Fails

Check to make sure that your version of Catapult has the SCVerify flow installed. Type:

```
flow package names
```

You should see a list of names that includes SCVerify. If you do not see SCVerify listed, then try re-installing Catapult or contact Calypto Design Systems Customer Support.

Back to [Flow Not Enabled](#).

Return to [START](#).

Flow Messages in Transcript

- If you get a warning message like this:

```
# Warning: SystemC Verification flow could not locate user-supplied
C++ testbench driver testbench.cpp
```

then go to [Missing Testbench.cpp File](#).

- If the error message looks like this:

```
# Error: Streamed resource 'some_variable:rsc' must be mapped to an
interface synthesis component having an enable handshake signal
(i.e. mgc_in_wire_en/mgc_out_stdreg_en)
```

then go to [Error Streaming Without Handshake](#).

- If the error message looks like this:

```
# Error: Streamed resource 'a:rsc' has more than one variable mapped
to it. Revisit mappings in the Architectural Constraints editor.
```

then go to [Error Streaming Multiple Variables](#).

Return to [START](#).

Missing Testbench.cpp File

This message indicates that the SCVerify flow could not locate the user-supplied C/C++ testbench, typically added to the project input files list. Your testbench can be embedded in the same file as your original design if it is properly surrounded by “#ifndef __SYNTHESIS__” to prevent Catapult from synthesizing the testbench.

Back to [Flow Messages in Transcript](#).

Return to [START](#).

Error Streaming Without Handshake

This message indicates that an array on the interface was marked as a streaming input (or output) but not mapped to an interface synthesis component that has an enable pin. This typically happens when an array is mapped to a memory component and is then marked as a streamed port. To correct this condition, open the Architectural Constraints editor and select an interface component that has an enable handshake (such as mgc_in_wire_en, mgc_in_wire_wait, mgc_out_stdreg_en, mgc_out_stdreg_wait).

[Back to Flow Messages in Transcript.](#)

[Return to START.](#)

Error Streaming Multiple Variables

This message indicates that the streamed port had more than one interface variable mapped to it. To correct this condition, open the Architectural Constraints editor, select the interface port and drag-and-drop the variables on the ports such that only one variable is associated with the streamed interface.

[Back to Flow Messages in Transcript.](#)

[Return to START.](#)

ModelSim Launch

Check the Catapult transcript window for the error message shown here.

```
flow run /SCVerify/launch_make Verify_rtl_vhdl_msim.mk simgui
# Error: =====
# Error: errorInfo: Package: SCVerify, Version: 2008a.2, Flow: launch_make
# Error: Script: Mgc_home\pkgs\sif\userware\En_na\flows\app_scgen.flo
# Error: C:/Modeltech_6.4b/win32/junk/vsim.exe is not an executable
```

This error indicates that either ModelSim was not properly installed or that the Catapult options for Simulation are not set correctly. The path to your ModelSim installation should be set using the menu pick **Tools > Set Options...** and selecting **Flows > ModelSim** in the navigation pane on the left.

[Return to START.](#)

ModelSim Errors

- If you encounter errors during the compilation of the SCVerify wrappers (i.e. before “sccom -link”), go to [ModelSim Compile Errors](#).
- If you encounter errors during the linking of the SCVerify wrappers (i.e. before the design is loaded in “vsim”), go to [ModelSim Link Errors](#).
- If you encounter errors after the design is loaded in vsim but before the simulation is started, go to [ModelSim Design Load Errors](#).

- If you encounter errors during simulation, go to [Simulation Errors](#).

Return to [START](#).

ModelSim Compile Errors

Starting from the bottom of the compilation transcript in either the Catapult transcript window (if you launched ModelSim in batch mode) or the ModelSim transcript window, scroll back through the transcript looking for messages like this:

```
# Compiling C++ file: <filename>.cpp
```

Starting from this location in the transcript, look for the first error message.

- If the compiler error message refers to an undeclared capture function,

```
"# testbench.cpp:30: `capture_<some_variable>' undeclared (first use  
this function)"
```

then go to [“Capture” Function Undeclared](#).

- If the compiler error message refers to “protected” or “private” data members of a class,

```
# ../../ComplexNum.h(130): `DATA_TYPE Complex<DATA_TYPE>::data_Im'  
is protected  
or  
# ../../ComplexNum.h(132): `DATA_TYPE Complex<DATA_TYPE>::data_Re'  
is private
```

then go to [Non-Public Data Members in Class on the Interface](#).

- If the compiler error message looks like this:

```
# # .../Mgc_home/shared/include/mc_typeconv.h:11: parse error before  
'Q',
```

then go to [Compiling with sc_fixed Types](#).

- If the compile error message looks like this:

```
# # testbench.cpp:47: invalid conversion from `unsigned int' to  
'unsigned int*'
```

then go to [Capturing a Pointer Value](#).

- If the compiler issues warnings about “object ... is not debuggable”, go to [Warning Non-Debuggable Type](#).
- If the none of these are applicable, go to [Additional Compile Error Trouble](#).

Back to [ModelSim Errors](#).

Return to [START](#).

“Capture” Function Undeclared

Open the file named in the error message and go to the specified line. Determine the enclosing function containing this call to the capture function. The enclosing function should be testbench::main().

- If the call to the capture function is outside of the testbench::main() function, go to [“Capture” Function Out of Scope](#).
- If the call to the capture function is inside of the testbench::main() function, go to [Captured Port Optimized Away](#).

Back to [ModelSim Compile Errors](#).

Return to [START](#).

“Capture” Function Out of Scope

The capture_<x>() functions are only available in the scope of the C++ class “testbench”. Thus, you can only call them from within the function testbench::main(). Move your capture_<x>() calls inside of this function.

Back to [“Capture” Function Undeclared](#).

Return to [START](#).

Captured Port Optimized Away

Look in the cycle or RTL netlist for a port with the name specified in the name of the capture function.

- If it is not present, then the class/struct member was apparently unused and optimized away by Catapult. In this case, your testbench.cpp does not need to call a capture_data function for that member.
- If it is present and within testbench::main(), check the capitalization of the name.

Back to [“Capture” Function Undeclared](#).

Return to [START](#).

Capturing a Pointer Value

Your compile transcript should look something like the sample shown here:

```
# Compiling C++ file: testbench.cpp
# # Model Technology ModelSim SE sccom ...
# #
# # testbench.cpp: In member function `int testbench::main()':
# # testbench.cpp:47: invalid conversion from `unsigned int' to `unsigned
# # int*'
# # testbench.cpp:47:   initializing argument 1 of `void
# #     testbench::capture_sum(unsigned int*)'
# # ** Error: (sccom-6142) Compilation failed.
```

In this case, the capture function was expecting a pointer to an “unsigned int” and was instead called with an “unsigned int”. Simply place an “&” before the variable name in the call to the capture function. For example, change this:

```
unsigned int tb_sum;  
capture_sum(tb_sum);
```

to this:

```
unsigned int tb_sum;  
capture_sum(&tb_sum);
```

[Back to ModelSim Compile Errors.](#)

[Return to START.](#)

Warning Non-Debuggable Type

This warning from ModelSim indicates that there is insufficient information in the data type to be able to view this object in ModelSim, even though the simulation will work correctly. This warning can be safely ignored, though it may mean that some signals in the design will not be visible in the Wave window.

```
# Compiling C++ file: testbench.cpp  
# # Model Technology ModelSim SE sccom ...  
# #  
# # ** Warning: (sccom-6102) The object "data_vector" is not debuggable.  
(Catapult/Solution.v1/mc_testbench.h, 39)
```

[Back to ModelSim Compile Errors.](#)

[Return to START.](#)

Non-Public Data Members in Class on the Interface

Interface variables must be visible in order for ModelSim to access their values, and thereby enable the SCVerify comparator functions to compare their values. If a class variable contains data members that are “protected” or “private”, you will need to make them visible by making them “public”.

[Back to ModelSim Compile Errors.](#)

[Return to START.](#)

Compiling with sc_fixed Types

If the compiler error message looks like this:

```
# # In file included from Catapult/Solution.v1/mc_testbench.h:34,  
# #           from testbench.cpp:6:  
# # .../Mgc_home/shared/include/mc_typeconv.h:11: parse error before `Q'
```

then there is a reference in your testbench to a SystemC fixed-point data type (sc_fixed/sc_ufixed). In order to compile with SystemC fixed-point types, you must include the macro definition “SC_INCLUDE_FX” before every include of the systemc.h header file. For example, your code must look like this:

```
#define SC_INCLUDE_FX
#include <systemc.h>
```

[Back to ModelSim Compile Errors.](#)

[Return to START.](#)

Additional Compile Error Trouble

Look at the compile error messages carefully. The format of the message is:

```
<filename>:<linenumber>: <error message text>
```

- If the filename has no directory path (i.e. it is just a name with no “/” or “\”), then go to [Picking Up STL Header From Working Directory](#).
- If the filename is a Windows header file from the ModelSim include tree (such as “windows.h”, “winbase.h”, etc.), then go to [Overloading Window Macros/Typedefs](#).
- If neither of these choices apply, contact customer support.

[Back to ModelSim Compile Errors.](#)

[Return to START.](#)

Picking Up STL Header From Working Directory

Look closely at the error message. The format is:

```
<filename>:<linenumber>: <error message text>
```

If the filename is a leaf filename (no directory name) then it was picked up from the project directory. There are many C++/STL header files that have names with no “.h” extension which could conflict with like-named files in the project directory. If the file name in the error message is any of the names from the list below, then you should rename that file in your project directory to something else. Example: if there is a file name “new” in your project directory, you might get an error like this:

```
# # Compiling C++ file: testbench.cpp
# # Model Technology ModelSim SE sccom
#
# # In file included from .../bits/stl_algobase.h:69,
# #                 from .../memory:54,
# #                 from .../string:48,
# #                 from .../bits/localefwd.h:49,
# #                 from .../ios:48,
# #                 from .../ostream:45,
# #                 from .../iostream:45,
```

```
# #           from testbench.cpp:2:
# # new:1: parse error before numeric constant
```

Rename it to “new.txt” to prevent the compiler from thinking it is C++ code.

The list of C++/STL headers is:

Table 9-3. C++ and STL Headers

algorithm	backward	bitset	cassert	cctype
cerrno	cfloat	ciso646	climits	clocale
cmath	complex	csetjmp	csignal	cstdarg
cstddef	cstdio	cstdlib	cstring	ctime
cwchar	cwctype	deque	exception	ext
fstream	functional	hash_map	hash_set	iomanip
ios	iosfwd	iostream	istream	iterator
limits	list	locale	map	memory
mingw32	new	numeric	ostream	queue
rb_tree	rope	set	slist	sstream
stack	stdexcept	streambuf	string	typeinfo
utility	valarray	vector		

Back to [Additional Compile Error Trouble](#).

Return to [START](#).

Overloading Window Macros/Typedefs

If you get compile errors from windows header files such as windows.h, windef.h, winbase.h, etc., then it may be due to a compiler macro definition from your header/C files conflicting with the windows headers. In the example below, the testbench.cpp file has “#define INT,” but the windows headers use the macro INT in many of the function prototype declarations. Since the definition of INT now resolves to an empty string, the prototype is syntactically incorrect and results in an error. Check your source code for macro definitions like INT, UINIT, BYTE, WORD, FLOAT, LPBOOL, etc.

```
# # Compiling C++ file: testbench.cpp
# # Model Technology ModelSim SE sccom
# #
# # In file included from .../gcc-3.2.3-mingw32/include/Windows.h:48,
# #                   from .../include/systemc/sc_cmnhdr.h:116,
# #                   from .../include/systemc/systemc.h:41,
# #                   from testbench.cpp:4:
# # .../gcc-3.2.3-mingw32/include/windef.h:243: declaration
# #       does not declare anything
```

[Back to Additional Compile Error Trouble.](#)
[Return to START.](#)

ModelSim Link Errors

- If the link error message looks like this:

```
# # C.../testbench.cpp:31: undefined reference to `some_function()'
```

then go to [ModelSim Link Undefined Reference](#).

- If the link error message refers to “multiple definition of <var_name>”, then go to [ModelSim Link Multiple Definition](#).

[Back to ModelSim Errors.](#)
[Return to START.](#)

ModelSim Link Undefined Reference

If the link transcript looks something like this:

```
# # Model Technology ModelSim SE sccom ...
# #
# # Catapult/Solution.v1/work\_sc\testbench.o(.text+0xa5): In function
`ZN9testbench4mainEv':
# # C.../testbench.cpp:31: undefined reference to `some_function()'
# # ** Error: (sccom-6126) Linking failed. Creation of
Catapult/Solution.v1/work/systemc.so failed.
```

then the problem may be that some files required by the testbench were not compiled with the SystemC compiler. Add all required files to the Catapult project Input Files list and exclude them from synthesis compilation. To exclude files from compilation, right-click on the files and choose the “**Exclude from Compilation**” menu item.

[Back to ModelSim Link Errors.](#)
[Return to START.](#)

ModelSim Link Multiple Definition

This error can be caused by having global variables defined in a header file. The problem manifests because the header files for the design are also included in the mc_testbench.h for the SCVerify flow. Consequently, the global variables are redefined. As a general rule, avoid putting global variables in header files.

[Back to ModelSim Link Errors.](#)
[Return to START.](#)

ModelSim Design Load Errors

If the Modelsim error is:

```
"Error: (vish-4014) No objects found matching"
```

then go to [Cannot Wave Non-Debuggable Objects](#).

Back to [ModelSim Errors](#).

Return to [START](#).

Cannot Wave Non-Debuggable Objects

Your ModelSim transcript probably looks something like this:

```
# vsim -L mgc_hls -L mgc_ioport -L work -t ps top
# Loading Catapult/Solution.v1/work/systemc.so
...
# ** Error: (vish-4014) No objects found matching
'sim:/top/comparator_msg_out/data_golden'.
# Error in macro
././Catapult/Solution.v1/testbench_compile_cycle_vhdl.tcl line 98
# (vish-4014) No objects found matching
'sim:/top/comparator_msg_out/data_golden'.
```

The error message is caused by the attempt to add an object to the Wave window for which ModelSim lacks certain debug information (You probably also have warnings in your compile transcript that are described here). This error message can be safely ignored, though it means you will not be able to view values on that SystemC object. This is a limitation of the C++ compiler used by ModelSim and will be corrected in a future release of ModelSim.

Back to [ModelSim Design Load Errors](#).

Return to [START](#).

Simulation Errors

- If the final simulation summary reports looks like this:

```
# # Simulation completed
# # comparator_z_rsc_INST comparison count = 0
# # comparator_z_rsc_INST error count = 10
# # input FIFO c_rsc_INST was never filled
# # Simulation FAILED at time 1250
```

then go to [Input FIFO Never Filled](#).

- If the simulation fails and there are warnings such as this:

```
Warning: top/c:rsc no more input data for variable c, assuming pipe
is flushing
```

then go to [Input FIFO Underflow](#).

- If the simulation errors out early with the message:

```
# # ERROR: Hardware output FIFO top/z:rsc is full
```

then go to [Output FIFO Overflow](#).

- If the simulation mismatches involve variables that are fixed-point types (such as `sc_fixed`), then go to [Doubles/Sc_fixed Mismatch](#).
- If the simulation output of the cycle/rtl appears to be the same as the C/C++ but just delayed in time, then go to [Simulation Mismatch](#).

[Return to START](#).

Input FIFO Never Filled

This message indicates that at least one input variable had no values captured, i.e. no calls to "capture_x(x)" were made. Therefore, the input FIFO was never able to provide data to the cycle or RTL model for that variable. Check your testbench to ensure that you have a "capture_X()" function call for each parameter in the call to the original function.

[Back to Simulation Errors](#).

[Return to START](#).

Input FIFO Underflow

If this message occurs early in the simulation then some values were captured properly but others were not. Check the testbench to make sure that all of the "capture_X()" calls are being executed (not in conditional branches).

[Back to Simulation Errors](#).

[Return to START](#).

Output FIFO Overflow

This message indicates that the cycle/rtl model overflowed the output FIFO which feeds the comparison of the C/C++ result with the hardware result. This is caused by the testbench failing to capture the golden results from the C/C++. Check your testbench to ensure that you have a "capture_X()" function call for each parameter in the call to the original function.

[Back to Simulation Errors](#).

[Return to START](#).

Doubles/Sc_fixed Mismatch

Check to make sure that your testbench uses the correct fixed-point precision for intermediate values. For example, if you have a value that will be passed as an argument to the function with a type of "`sc_fixed<32,4>`", then declare a local variable of that type to hold the values. Do not use the C/C++ "double" or "float" types.

[Back to Simulation Errors](#).

[Return to START](#).

Simulation Mismatch

If you are getting simulation mismatches that appear to be delayed in time, check to make sure that the TRANSACTION_DONE_SIGNAL option is enabled in the "Setup Design" constraints window. This will provide the most accurate transaction handshaking between the C/C++ and the cycle/RTL model.

[Back to Simulation Errors.](#)

[Return to START.](#)

Formal Verification Flow

Catapult has integrated the Calypto SLEC (Sequential Logic Equivalence Checking) tool into the Catapult user interface in order to provide a seamless flow for checking the functional equivalence between system-level input source code models and the RTL designs generated by Catapult. In a typical usage scenario of the Formal Verification flow, Catapult is used to explore architectural trade-offs at the system-level. After generating a given RTL description, further ECO modifications for clock gating, power and re-timing may be done. The SLEC tool then verifies that RTL ECOs do not change the functionality of the design. SLEC verifies the functionality in spite of sequential design differences in throughput, latency and interfaces.

This flow provides the following features and benefits:

- Enables an design flow with system-to-RTL verification methodology
- Eliminates the manual effort of creating verification set-up scripts
- Automates the mapping of interfaces between the input source and the RTL design
- Provides an independent, automated verification path
- Eliminates the need for RTL, block-level testbenches
- Provides comprehensive design verification and synthesis

Note

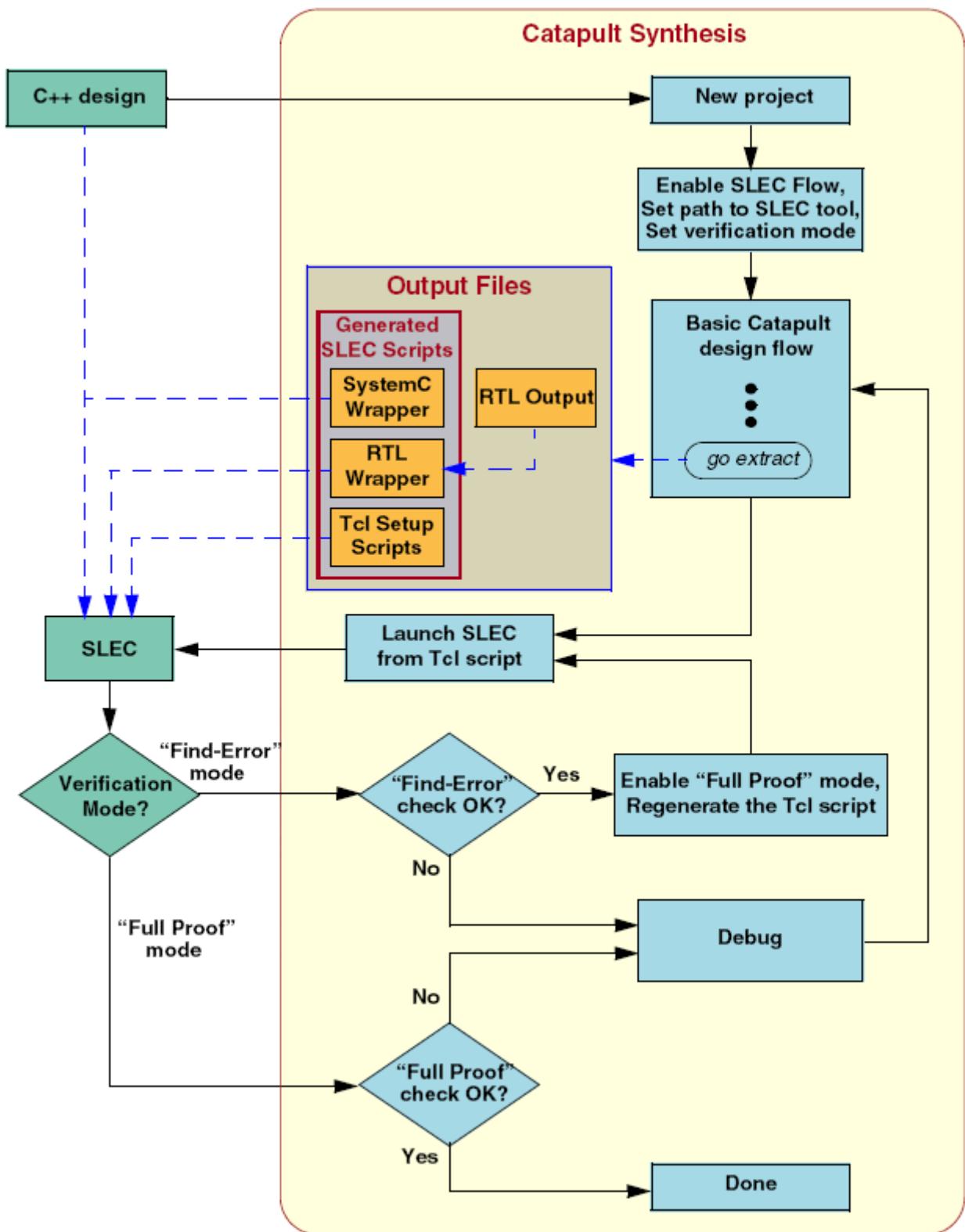
 The Formal Verification flow is an optional feature for Catapult that is licensed separately. It will not appear in the Catapult user interface unless the proper license is available. At this time the Formal Verification flow is available only on Linux.

The architecture of the flow is illustrated in Figure 9-12. If the Formal Verification flow is enabled, Catapult automatically generates three additional output files to interface with the SLEC tool. These files provide all the required setup for the SLEC tool:

- A Tcl script that launches and drives the SLEC tool
- A SystemC wrapper for the original C++ design
- A VHDL wrapper for the RTL design to align the interfaces between the two designs

Verification is performed in two passes. The first pass performs a “Find-Error” check which is a quick check for simple errors, such as syntax mistakes. The second pass performs a “Full Proof” which provides 100% coverage of the design.

Figure 9-12. Diagram of Integrated Formal Verification Flow



A Sample Formal Verification Flow Session

This sample session demonstrates how to use the Formal Verification flow by taking a simple FIR filter design through it.

Note

 The Formal Verification flow does not currently support Ping-Pong memories on the C++ interface of the design.

Step 1: Enable and Configure SLEC Options

Enable the Formal Verification flow either from GUI (described below) or by entering the [flow package require](#) command on the Catapult command line:

```
flow package require /SLEC
```

From the GUI, select the **SLEC** item in the Flow Manager window. Details about the flow appear in the window, as well as option to select a particular version of the flow. When you click the “Enable” button the flow package is loaded into the project, and the flow options appear in the window.

The Formal Verification flow can be disabled, if necessary, by selecting the SLEC item in the Flow Manager window and clicking the **Disable** button.

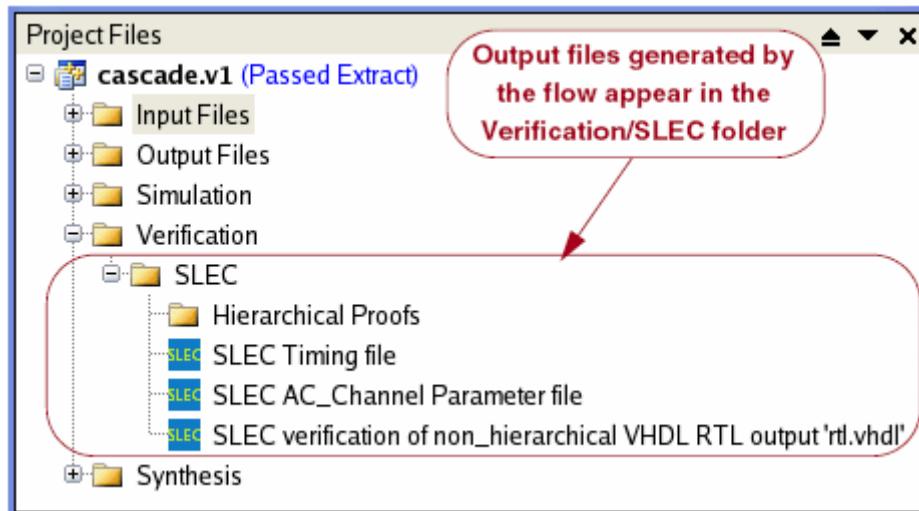
Setting SLEC Options

After the flow package is loaded, the flow package properties appear for you to configure the Formal Verification flow options. The path to the SLEC executable directory must be set. For the purposes of this sample session, we will use the default settings for the other options. The **Enable Full Proof Verification Mode** option will be changed later for the second pass through the flow.

You can access the Formal Verification flow options at anytime by selecting the **SLEC** item in the Flow Manager window. Alternatively, you can use the Catapult Synthesis Options window ([File > Set Options... > Flows > SLEC](#)). For more information about the Formal Verification Options and how to change the default enable setting, refer to [“SLEC Formal Verification Flow Options”](#) on page 215.

Step 2: Generate Output Files

After enabling the Formal Verification flow, follow the normal Catapult design flow all the way through to RTL generation step. The SLEC output files are automatically generated by the “go extract” command (or the **Generate RTL** command icon). The files are created in the Solution directory and appear in the Project Files window as shown in Figure 9-13.

Figure 9-13. Formal Verification Flow Output Files

The file names in the Project Files window are not the same as the actual file names in the Solution directory. You can see the actual file names in the “**Details**” window when a file is selected, or by right-clicking on a file.

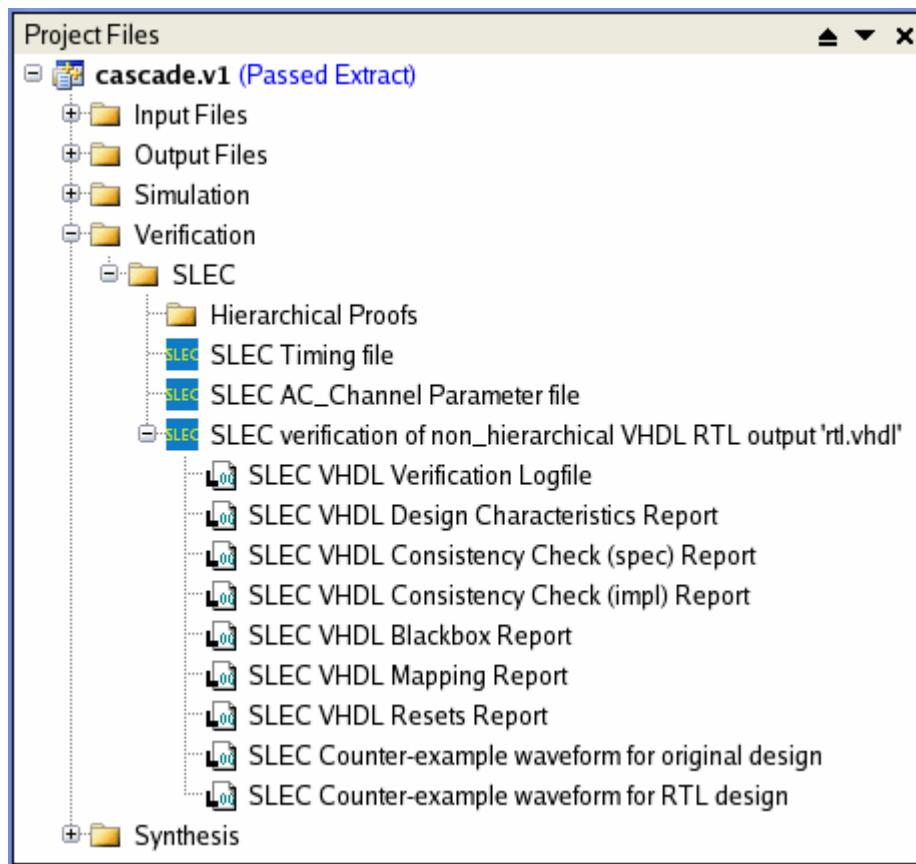
Step 3: Launch SLEC to Perform Find-Error Check

Double-click on the **SLEC verification of non_hierarchical VHDL RTL output ‘rtl.vhdl’** script in the GUI to launch the SLEC tool. SLEC performs the “Find-Error” check by default because the “Full Proof” verification mode option was not enabled in the SLEC options in “[Step 1: Enable and Configure SLEC Options](#)” on page 568. As the verification runs its progress is monitored in the Catapult transcript window.

When it is complete, the results files are created in the solution directory and appear in the Project Files window, as shown in Figure 9-14. The set of results files includes a logfile of the run, several report files, and a couple of waveform files. Double-clicking any of these files in the GUI will open the file for viewing.

If the design does not pass the Find-Error check, the problem is likely a simple error that can quickly be corrected. The report in the results file will help diagnose the problem. After the problem is fixed, simply regenerate the Tcl script and wrapper files and run the Find-Error check again. When the design passes, proceed to the next step.

Figure 9-14. SLEC Verification Results Output Files



Step 4: Enable Full Proof Option and Regenerate Script

To enable Full Proof mode, open the Formal Verification flow options dialog box by selecting **SLEC** in **Flow Manager** window. Then check the **Enable Full Proof Verification Mode** option and click the “Apply” button.

Next, the Tcl script must be regenerated in order to pick up the change. Right-click on the **SLEC verification of non_hierarchical VHDL RTL output ‘rtl.vhdl’** file in the SLEC output folder, then select **Regenerate Wrapper**.

Step 5: Launch Formal Verification for Full Proof Check

The final step is to run the Full Proof verification. Once again double-click the **SLEC verification of VHDL RTL output** file. The behavior is the same as for the Find-Error run. The same set of results files are saved in the Solution directory.

Supported Interface Synthesis Components

The following matrix shows the level of support for all of the Interface Synthesis components currently available in Catapult.

Table 9-4. SLEC Support for Catapult Interface Synthesis Components

Interface Synthesis Component	C Function Args			Interface Synthesis Options		
	Scalar	1D array	nD array	multi-var	streaming	word width
mgc_in_wire	X	X	X	X	Note 2	Note 4
mgc_in_wire_en	X	X	X	X	X	
mgc_in_wire_wait	X	X	X	X	X	
DirectInput	X	X	X	X	Note 2	
mgc_out_stdreg	X	X	X	X	Note 2	
mgc_out_stdreg_en	X	X	X	X	X	
mgc_out_stdreg_wait	X	X	X	X	X	
mgc_out_reg	X	X	X	X	Note 2	
mgc_out_buf_wait	X	X	X	X	X	
mgc_out_fifo_wait						
mgc_out_prereg_en	X	X	X	X	X	
mgc inout stdreg en						
mgc inout stdreg wait						
mgc inout buf wait						
mgc inout fifo wait						
singleport RAM in	Note 1	X	X	X	Note 3	X
multiport RAM in		X	X	X		X
singleport RAM out		X	X	X		X
multiport RAM out		X	X	X		X
singleport RAM inout		X	X	X		X
multiport RAM inout		X	X	X		X

Note 1: Scalar function args are typically not mapped to memory interface components.

Note 2: Streamed interfaces require a handshake signal from Catapult indicating when a sample is read.

Note 3: Mapping to a memory and mapping to a streamed interface are mutually exclusive in Catapult.

Note 4: The WORD_WIDTH directive can be used to split the I/O resource into multiple ports.

Flop Mappings for the SLEC Flow

Catapult-generated flop-map data provides SLEC with accurate information about the mapping of variables that keep state in the C specification to the corresponding registers in the generated RTL. The mappings are part of the standard SLEC script generated by Catapult in the Solution directory (named either SLEC_compile_rtl_vhdl.tcl or SLEC_compile_rtl_v.tcl).

The script uses the SLEC commands “create_map -flop” and “slice_flop” to provide the mappings. Before calling the SLEC commands, the script uses the SLEC “find” command to verify that the C variable is found by SLEC. If the variable is not found by SLEC, then a warning is issued, and the flop mapping is not applied. The “mapping.log” file generated by SLEC can be used to verify what mappings were applied.

The ENABLE_FLOP_MAPS option controls whether the flop mappings are utilized by SLEC. The option is enabled by default. To toggle the option on or off, use the following command:

```
options set Flows/SLEC ENABLE_FLOP_MAPS <true_or_false>
```

From the GUI, use the SLEC flow options dialog box to toggle the “Enable using FLOP mappings” option. Either select “SLEC” in the Flow Manager window or use the Catapult Options window (Tools > Set Options... > Flows > SLEC) as described in [“SLEC Formal Verification Flow Options”](#) on page 215.

Current known limitations:

1. Flop maps are not generated for global variables. A warning is placed in the script:

```
# WARNING: did not find source info for x while generating flop maps
for:
#      {x[31:31]} x_31_sva {x[15:0]} x_15_0_sva
```

2. Flop maps for static variables inside template functions or overloaded functions are not generated correctly. SLEC will issue a warning that it could not find the C variable.
3. SLEC may not find some C variables with some complex struct/class and array combinations. SLEC will issue a warning that it could not find the C variable.

Generating Proofs for Hierarchical Blocks

Enabling the SLEC flow option “Generate hierarchical proofs” will automatically generate SLEC wrapper scripts for each hierarchical sub-block in the design and its corresponding RTL sub-module (in addition to the regular top-level wrapper/script generation). The original C source and the generated RTL for the full block are used in the sub-verification runs. Each sub-block/sub-module pair can be verified separately. The appropriate sub-function and sub-module are set as the top module in the wrappers.

Figure 9-15 illustrates the concept. Using the example “cascade” design shown below, the hierarchical sub-block/sub-module pairs, fir_filter1 and fir_filter2, are verified independently.

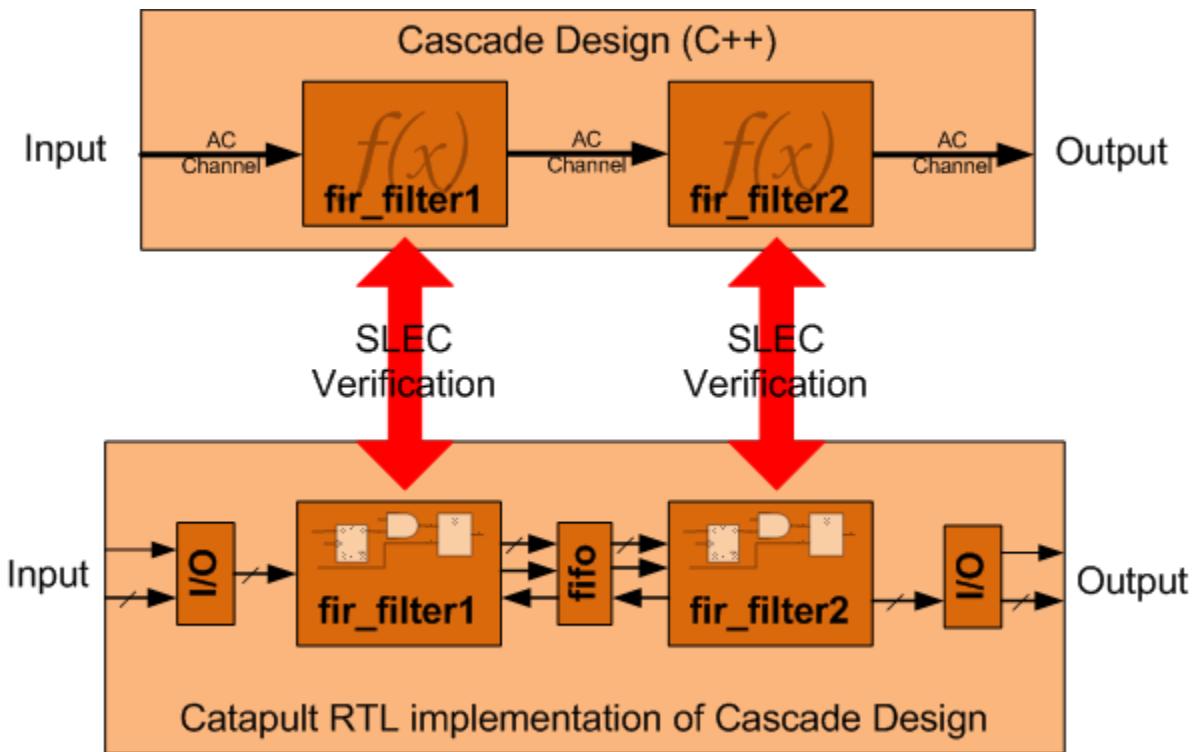
```

#pragma design
void fir_filter1 (ac_channel< ac_int<8> >& input, ..., ac_channel<ac_int<8>
& output ) {...design details...}

#pragma design
void fir_filter2 (ac_channel< ac_int<8> >& input, ..., ac_channel<ac_int<8>
& output ) {...design details...}

#pragma design top
void cascade (ac_channel< ac_int<8> >& input, ..., ac_channel<ac_int<8> >&
output) {
    static ac_channel< ac_int<8> > intermediate;
    fir_filter1(input, .... intermediate);
    fir_filter2(intermediate, ....output);
}
    
```

Figure 9-15. Hierarchical Block Verification



Catapult gives you the option of running SLEC on the entire design as a flat design or running each sub-block standalone.

Generating RTL output for the design will generate all the sub-verification scripts and directory structure for all the SLEC runs. Wrappers for each sub-block/sub-module pair are saved in separate sub-directories inside the solution directory. The directory names have the form **SLEC_<block_name>**. In the Catapult GUI, the sub-block wrappers are grouped under the folder "Verification/SLEC/Hierarchical Proofs" in the "Project Files" window.

The “Extended reset length” option allows you to artificially extend the duration of the reset state in SLEC to ensure that all blocks are well into their normal operating mode before SLEC starts to identify transaction intervals using simulation.

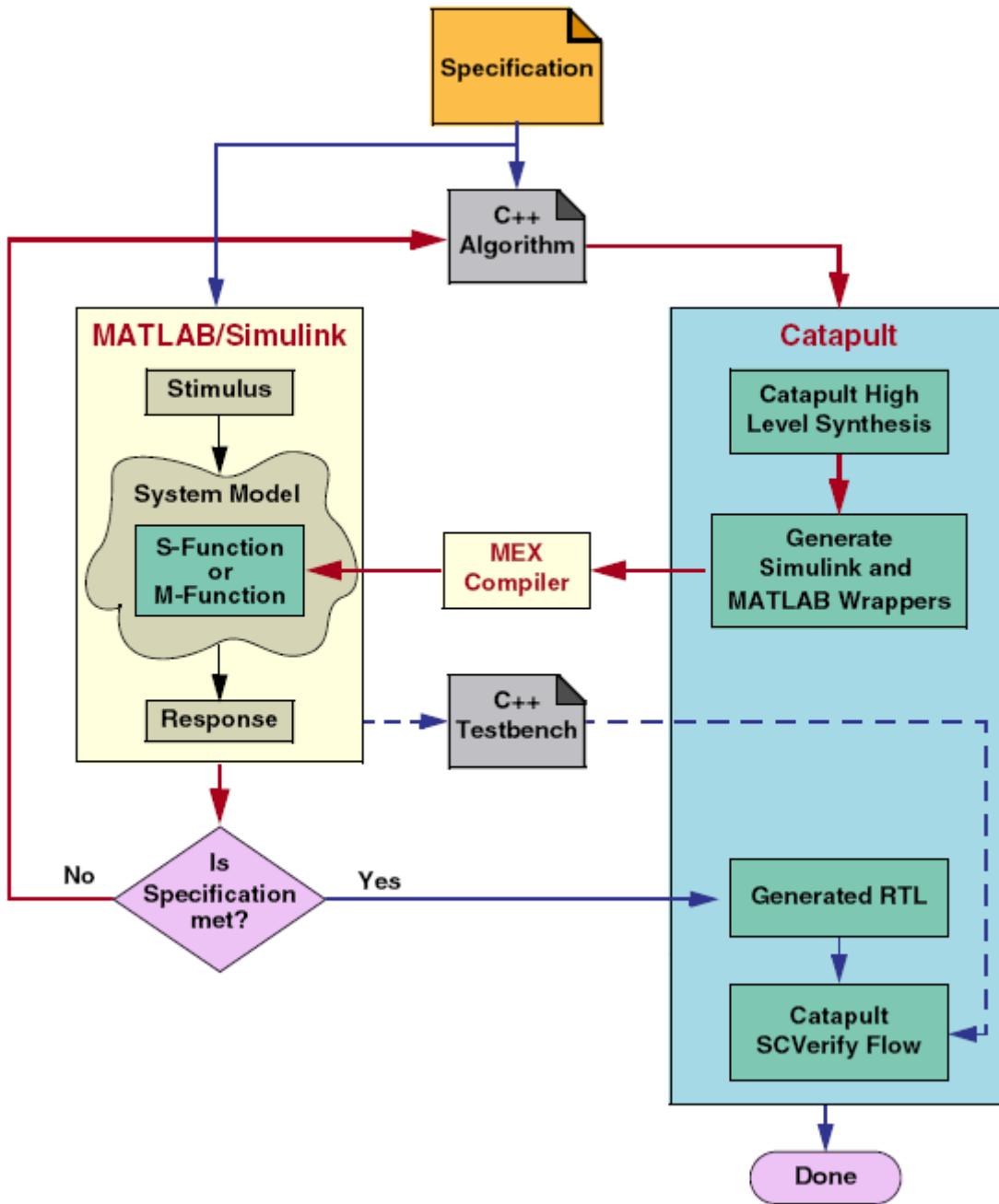
The “Maximum ac_channel size” option sets the size for all ac_channel objects in the C++ design. This must be set large enough to allow the C++ simulation to run without running into a deadlock. If overflow or underflow occurs during the SLEC simulation phase, a message is issued.

MATLAB/Simulink Flow (C/C++)

The “MATLAB” flow integrates the Mathworks tools Simulink and MATLAB into the Catapult design process. The flow is centered on the idea of using the same C/C++ code in numerical simulation (Simulink and MATLAB) and in high-level synthesis (Catapult). Figure 9-16 illustrates the integrated flow.

The flow takes advantage of the Mathworks tools’ ability to “import” existing C/C++ functions as blocks into its Simulink and MATLAB simulation environments. Simulink blocks (S-functions) and MATLAB blocks (M-functions) are natively compiled and linked against a dynamic library and are available to the simulators as dynamically loadable objects. Catapult automatically converts Calypto Design Systems ac_complex data types to Simulink complex types.

Figure 9-16. Integration of Simulink and Catapult



A typical combined session of Catapult and Simulink might go as follows:

1. Based on the design specification, the C/C++ source code for a function describing an algorithmic block is created. The MATLAB flow also requires a header file that contains all of the function prototypes for the design. The source code is modified for synthesis. For example, floating-point data types are replaced with fixed-point types, globals variables are eliminated, library calls are stripped, and so on.

The MATLAB flow does not support some data types on the design interface. Table 9-5 lists the support limitations. All data types, ac_channels, ac_complex, and so on, internal to the design are supported.

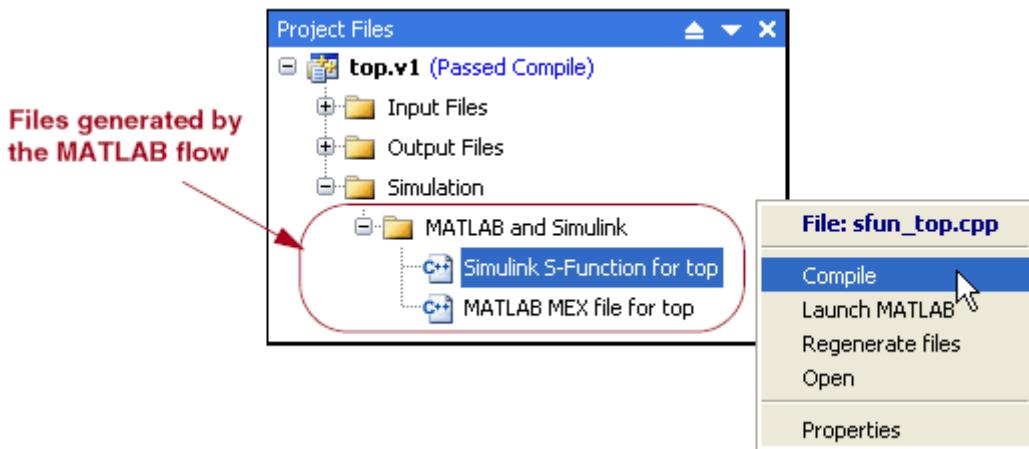
Table 9-5. Limited Support for Data Types on the Interface

Data Type on the Interface	Supported in Matlab	Supported in Simulink
ac_int	No	Yes
sc_int	No	No
ac_fixed	Yes	Yes
ac_channels	No	No
ac_complex	No	Yes
Bitwidths greater than 32 bits	No	No
Structs on the interface	No	No
Arrays on the interface	No (v2009a Update2) Yes (v2010a)	Yes

2. The C/C++ source code is synthesized with Catapult into RTL. Initially, the design constraints are relaxed in order to rapidly arrive at the first RTL implementation.
3. Once Catapult has generated the RTL, the MATLAB flow is run in Catapult (refer to [“Enabling the MATLAB/Simulink Flow \(C/C++\)”](#) on page 580 for information about setting the flow options). The flow generates two C++ wrapper files in the solution directory (Figure 9-17) that package the original C++ design in a Simulink S-function and a MATLAB M-function.

The flow also generates a Simulink Blockset library to hold icons for Catapult-generated S-Functions. The following files are generated by the flow as part of the blockset library. Both files are placed in the directory specified by the Matlab flow option **Simulink Model Directory**.

- **catapult_blocks_lib.mdl** - This file contains the Blockset definition for all blocks added to the Simulink model directory.
- **.blkssyms/sfun_<top-level design name>_lib.m** - This is the Simulink library file which describes the Simulink block and mask properties for the S-Function.

Figure 9-17. MATLAB Flow Interface

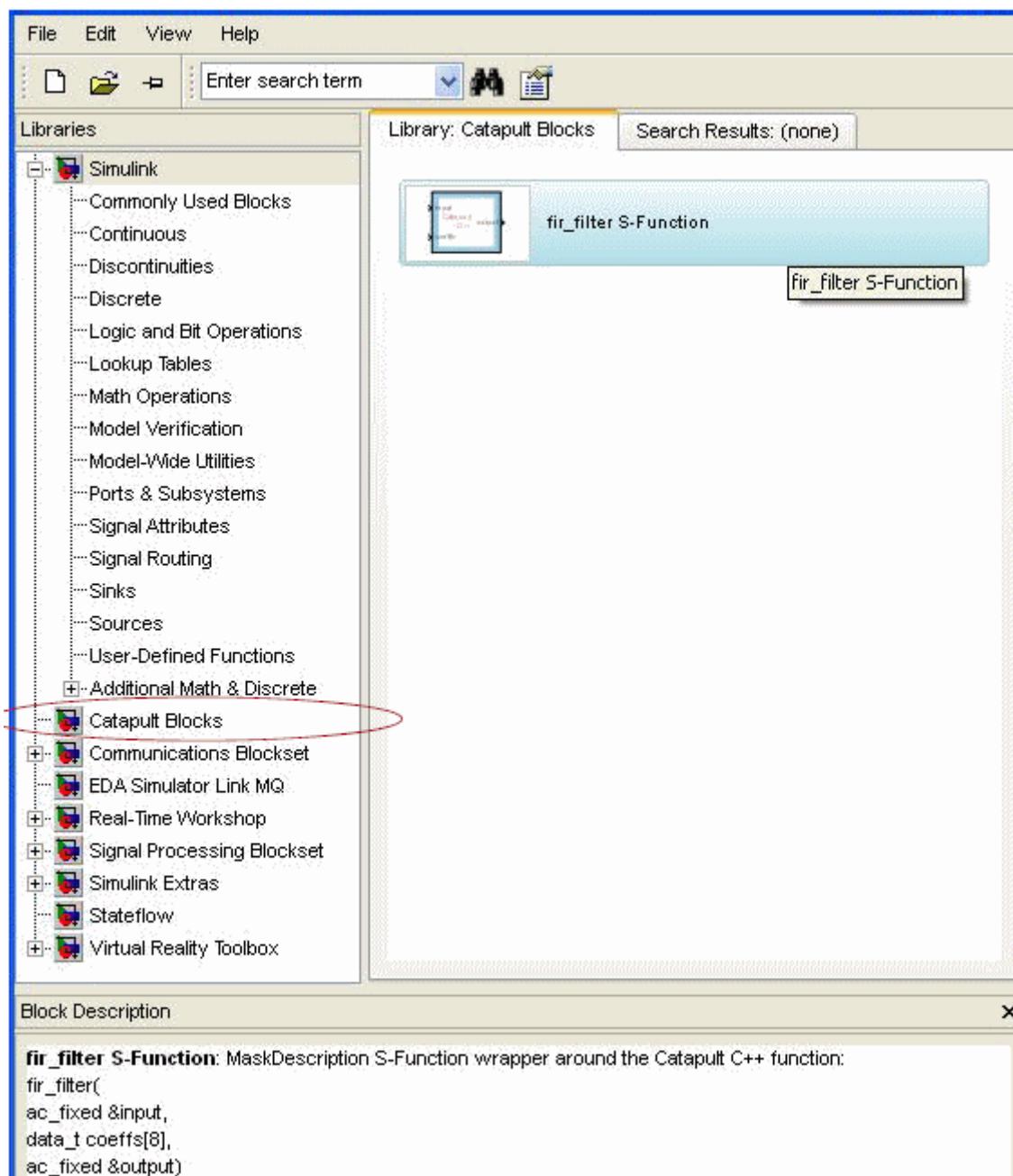
4. The wrapper is compiled with MEX compiler to generate a Simulink S-block. From the Catapult GUI, right-click on the S-Function wrapper and choose *Compile with MEX*.

**Note**

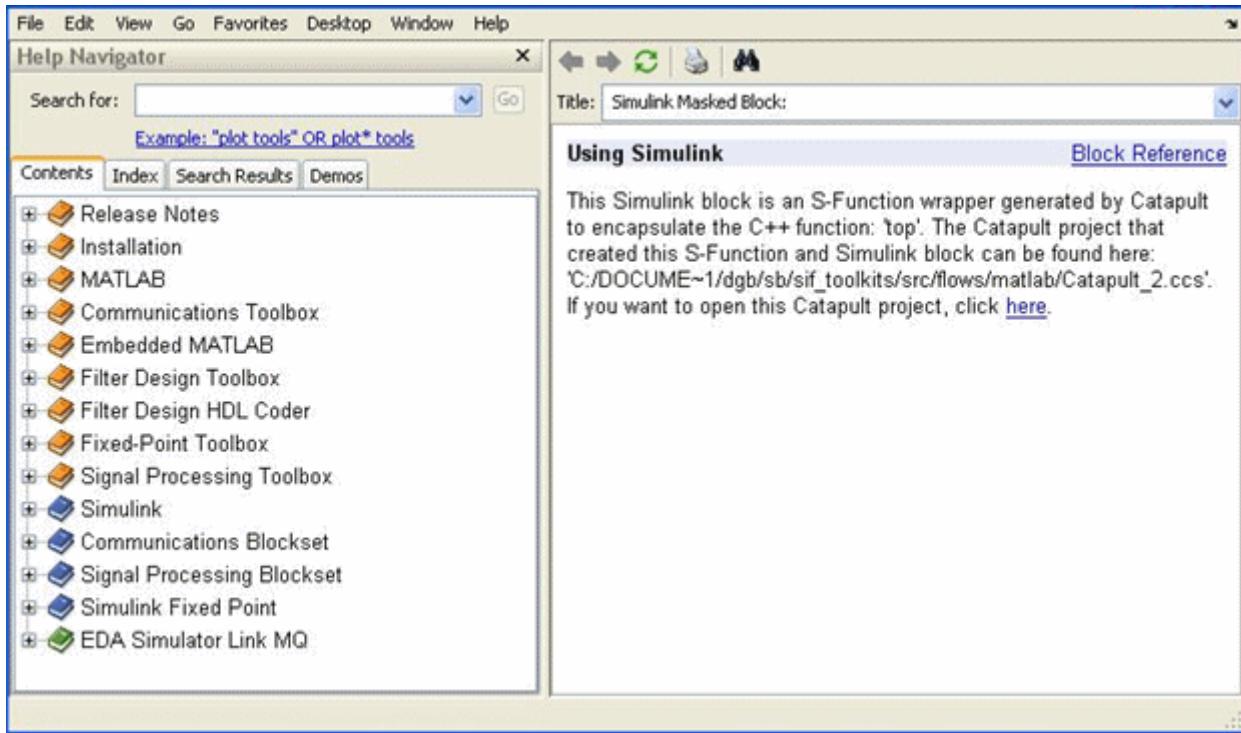
Before Matlab can be used to compile MEX files, a compiler options file must be created for the particular compiler(s) in the user environment. To create an options file, run the command “**MEX -setup**.” When you run this command, a series of questions are asked regarding the location of the compiler you would like to use to compile your code. Not all compilers are supported, so refer to the Matlab documentation for more information about which compilers are supported and how to generate compiler options files.

5. Invoke Simulink. In the Simulink Library Browser window, the Catapult library is named “Catapult Blocks,” as shown in Figure 9-18.

Figure 9-18. Catapult Blockset Library in Simulink



If you right-click on a Catapult block in the Simulink GUI and select help, it will bring up help text similar to that shown in Figure 9-19. The help message gives the path to the Catapult project that generated the block and provide an embedded link to reopen the project. Embedded in the help text is a hyperlink that can be used to reopen the design in Catapult (Note that you must have 'catapult' in your search PATH).

Figure 9-19. Figure 9.Help text for Catapult generated blocks

6. You need to select the Catapult blockset, find the new model, drags it onto the canvas and connect it to the rest of the simulation model
7. Run the Simulink model to exercise the block under test.
8. Examine the simulation results, and use them to identify errors and/or possible improvements in the model.
9. Modify the C/C++ in Catapult's editor (or in any external text editor).
10. Rerun Catapult on the modified model and regenerates the corresponding Simulink S-block.
11. Alternatively, the C/C++ code does not change, but the Catapult constraints do. Again, the S-block is regenerated to reflect the change in RTL.
12. Switch back to the still open Simulink and click on the 'Update Model' icon to absorb the changes to the S-block model.
13. Run the Simulink simulation again. This process iterates back and forth. At each iteration, you need to evaluate the numerical performance of the block under test (using metrics such as bit error rate (BER), signal-to-noise ratio (SNR) and so on).
14. Exit this iterative process when the design satisfies both the numerical metrics and the hardware QoR metrics.

Note



In reality, there would be an inter-disciplinary team of algorithm specialists and high-level synthesis engineers collaborating on iterations of this flow to achieve convergence.

Simulink is used here as a test bench. There is however a fundamental difference between comparing bits in an RTL test bench and evaluating numerical difference metrics in Simulink. RTL verification treats signals as bits – all bits have to match exactly or else the verification fails. Simulink on the other hand allows testing to a degree of tolerance. This is a key difference. One can make adjustments to the C/C++ code that will change the result at the bit level, but still satisfy a numerical quality metric, such as bit error rate (BER) or signal-to-noise ratio (SNR).

You can try the MATLAB flow by running the example provided in the Catapult Toolkits. From the Catapult session window, choose the **Help > Toolkits...** menu item, then select **“Integration Flows > Matlab Flow”**. Refer to [“The Toolkits Window”](#) on page 82 for information about how to run the toolkit.

Enabling the MATLAB/Simulink Flow (C/C++)

To enable the MATLAB/Simulink flow from the GUI, select the **Matlab** item in the Flow Manager window. Details about the flow appear in the window, as well as option to select a particular version of the flow. When you click the “Enable” button the flow package is loaded into the project, and the flow options appear in the window.

The Matlab flow can be disabled, if necessary, by selecting the **Matlab** item in the Flow Manager window and clicking the **Disable** button.

Flow options:

- **Matlab Installation Directory:**

Path to the top-level directory containing the Mathworks tools. The default setting relies on the value of the MATLABROOT environment variable. That environment variable must be set in order for the default option setting to work.

- **Simulink Model Directory:**

Optional output path for the compiled S-block/M-block models. By default, Catapult saves these files in the Solution directory.

You can access the MATLAB/Simulink flow options at anytime by selecting the **MATLAB** item in the Flow Manager window. Alternatively, you can use the Catapult Synthesis Options window (**File > Set Options... > Flows > Matlab**). For more information about the MATLAB flow Options and how to change the default enable setting, refer to [“MATLAB Flow Options”](#) on page 207.

The command line syntax for enabling the flow, setting the flow options, and running the flow:

```
flow package require /Matlab
options set Flows/Matlab MATLAB_ROOT <path>
options set Flows/Matlab SIMULINK_DIR <path>
flow run /Matlab
```


Chapter 10

Power Analysis and Optimization

This section describes how to use Catapult with third party tools to analyze power usage and the constraints used to optimize a design for low-power.

Analyzing a Design for Power Usage.....	583
Low Power Optimization	589

Analyzing a Design for Power Usage

Currently five power analysis tools are supported:

Table 10-1. Supported Power Analysis Tools

Technology	Flow Name	Third Party Tool
ASIC	DCPower	Synopsys Design Compiler
	RCPower	Cadence RTL Compiler
	SpyGlass-Power	Atrenta Spyglass-Power
FPGA	PowerPlay	Included with the Altera Quartus II software
	XPower	Included with the Xilinx ISE software

Note

 Design Compiler and SpyGlass-Power are not available for Windows operating systems.

All of these flows depend on the Catapult SystemC Verification flow (SCVerify) to generate driver scripts and prepare input files for the process. The supported simulators for generating the activity file are ModelSim and NC-Sim. The activity file is generated even if the simulation does not pass in the SCVerify flow. The appropriate Catapult license for the *Power Analysis Integration* option is required to run these flows.

Note

 All power estimation flows using Modelsim for generating the activity file now require version 6.2f or later versions of Modelsim.

Design Flow

Begin by enabling the flow for the target power analysis tool and the SCVerify flow. Catapult outputs an RTL description of the design including a hierarchical netlist of synthesizable components and state machines. Along with the RTL descriptions, Catapult generates constraints and a synthesis script for the downstream RTL synthesis tool. RAM and ROM blocks in the design may need special handling. After generating the RTL netlist, the design is simulated and a switching activity file is generated for use by the power analysis tool. You can use the resulting power estimates to compare architectures. A similar step could also be performed after RTL synthesis. The RTL synthesis tool generates a gate-level netlist and a switching activity file is created by simulating this gate-level netlist. The switching activity file and the netlist are then fed to the power analysis tool and a gate-level power analysis is then performed as well. This estimate is also fed into Catapult to correlate with the RTL-level power estimation.

Preparation Before Running Catapult

1. Set environment variables for the target power analysis tool.
 - **PowerPlay:**
 - QUARTUS_ROOTDIR: Directory containing the PowerPlay executable file.
 - QUARTUS_LIB: Top-level directory containing the technology libraries. If the Altera OEM version of ModelSim is installed, Catapult will automatically get the libraries from that installation tree. Otherwise, set the QUARTUS_LIB variable. The library directory structure should be as follows:

```
$QUARTUS_LIB/<lang>/<tech_name>/...
```

where <lang> is either “verilog” or “vhdl,” and <tech_name> contains the library tree for a particular technology. For example:

```
$QUARTUS_LIB/vhdl/stratixii/...
```
 - **SpyGlassPower:**
 - SPYGLASS_HOME: Top-level directory of the SpyGlass-Power installation tree.
 - NOVAS_INST_DIR: Optional. Used if you prefer using FSDB files instead of VCD files. The SpyGlassPower flow can use the Novas FsdbWriter tool to generate FSDB files. The NOVAS_INST_DIR environment variable specifies the top-level directory of the Novas installation tree. If NOVAS_INST_DIR is not set, you can specify the location of the Novas software from within Catapult by editing the Novas tool flow options (“[Novas Flow Options](#)” on page 208).
 - **XPower:**
 - XILINX: Directory containing the XPower executable file.

- XILINX_LIB: Directory containing the technology libraries. If the Xilinx compxlib script was used to create the Xilinx libraries, Catapult will automatically find them in the Xilinx installation tree. Otherwise, set the XILINX_LIB variable.

The library directory structure should be as follows:

```
$XILINX_LIB/<lang>/mti_se/<tech_name>/...
```

where <lang> is either “verilog” or “vhdl,” and <tech_name> contains the library tree for a particular technology. For example:

```
$XILINX_LIB/vhdl/mti_se/XilinxCoreLib/...
```

2. For the PowerPlay and Xpower flows, be sure that precompiled simulation libraries are available in \$QUARTUS_LIB and \$XILINX_LIB, respectively. For instructions about how to check and create the necessary libraries, refer to [“Accelerated FPGA Devices and Verification”](#) on page 523.
3. Set up the SCVerify testbench to exercise the design with the desired conditions for power measurement. Refer to [“C/C++ Designs and SCVerify”](#) on page 516 for information on how to set up the C++ testbench file for the SCVerify flow.
4. For the SpyGlassPower flow, add the variable "syn_library" in each Catapult library to indicate the Synopsys liberty file name. This is done using the Catapult C Library Builder command:

```
library add /LIBS/<Catapult library name>/VARS/syn_library --
-VALUE <DC library name>.lib
```

If not set in the technology library, the flows will infer the name from the name of the target_library (.db file). For memory libraries, the syn_library and the target_library root names are different so both of them must be specified. For example:

- Base Component Library:


```
syn_library: sample_090nm.lib
target_library: sample_090nm.db
```
- Library for 256x8 Single Port Memory:


```
syn_library: ram_sample_090nm-sp_256_16_8.lib
target_library: sample_090nm.db
```

The entity/module name of the memory as netlisted by Catapult should match the cell name in the liberty library.

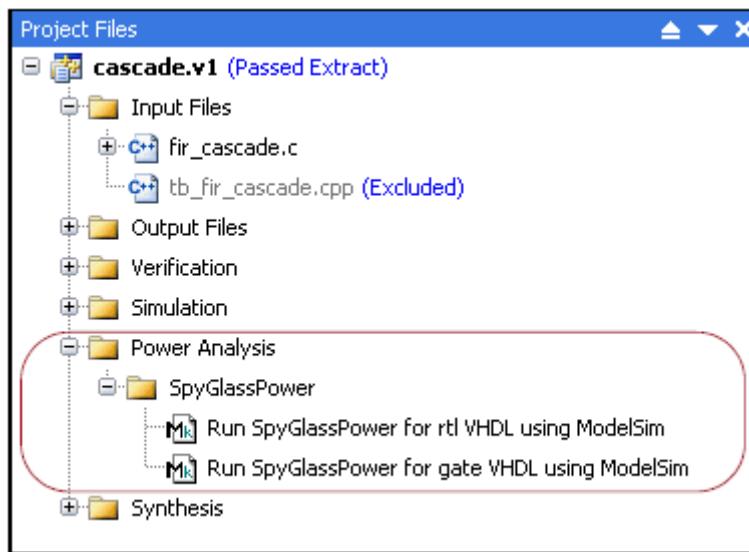
5. For the SpyGlassPower flow, make sure the SearchPath option for the DesignCompiler flow is set correctly (**Tools > Set Options > Flows > Design Compiler > Library Search Path**). Catapult will search this path for the following files:

- The db files for setting up the Design Compiler synthesis script.
- The VHDL or verilog for the technology gates for enabling gate-level simulation (gate-level power analysis flow). For instance if the target_library is sample_090nm.db, Catapult will search for sample_090nm.vhd or sample_090nm.v. If not found, the path “<search_path>/..sim” will also be searched.

Running a Power Flow in Catapult

1. Set the Catapult output options to netlist VHDL or Verilog (**Tools > Set Options...**, then select **Output**).
2. Enable the power flow. In the Flow Manager window, select the target power analysis tool name and click the “Enable” button. Optionally modify its flow package properties and click the “Apply” button. For descriptions of the available tool options, refer to the following sections:
 - [“DCPower Flow Options” on page 203](#)
 - [“PowerPlay Flow Options” on page 210](#)
 - [“SpyGlassPower Flow Options” on page 218](#)
 - [“XPower Flow Options” on page 222](#)
3. Enable the SCVerify flow in the Flow Manager window.
4. Add the input files and the testbench file for the SCVerify flow. Set the testbench to be excluded from synthesis compilation.
5. Setup the design constraints as usual. If needed for the SCVerify flow, enable the **“Transaction Done Signal”** on the **“Interface Control > Handshake”** page of the Setup Design window.
6. Proceed with architectural exploration and generate RTL. The power flow scripts will be generated in the Solution directory along with the RTL netlist. Among the files produced by the flows are:
 - a. Makefile(s) that first run the SCVerify flow to generate the activity file, and then run the target power analysis tool. In the Catapult GUI, the makefiles appear in the Project Files window in the folder **Power Analysis/<tool_name>**. For example, Figure 10-1 shows how the SpyGlassPower folder appears in the window.

Figure 10-1. Power Flow Folder and Makefiles



The makefile names, in the GUI and in file system, have four variable components: tool name, netlist type (gate or RTL), HDL language, and simulation tool. The netlist type of RTL applies only to the SpyGlassPower flow. In the file system the file names have the form:

```
<power_tool>_<netlist_type>_<language>_<simulation_tool>
```

Some examples are:

```
SpyGlassPower_rtl_vhdl_msim.mk
```

```
PowerPlay_gate_v_ncsim.mk
```

```
xpower_gate_v_msim.mk
```

- b. **scverify_msim.tcl** (or **scverify_ncsim.tcl**): script to run ModelSim (or NC-SIM) to generate a compressed activity file.
7. Run Power Analysis by double-clicking on the appropriate makefile. The power analysis flow does the following:
- a. Creates an output directory under the solution directory where the power estimations will be placed. The directory name corresponds to the same naming conventions as the makefile. For example:
- ```
SpyGlassPower_rtl_vhdl_msim
PowerPlay_gate_v_ncsim
xpower_gate_v_msim
```
- b. Runs the SCVerify flow to generate the activity file. The SpyGlassPower flow will run DesignCompiler, save the netlist and the SDF files, and then run gate-level

simulation. The PowerPlay and XPower flows run Precision RTL to generate the gate-level netlist.

- c. The power analysis report is added to the output files underneath the makefile in the Catapult GUI.
- d. Displays the resulting power estimation values in the Table window (See “[The Table Window](#)” on page 79). The first time the flow is run, additional data columns are appended to the right side of the table for the power data.

---

**Note**



The following commands can be used to force the regeneration of all the flow files generated by Catapult without having to rerun the design:

```
flow run /PowerPlay
flow run /SpyGlassPower
flow run /XPower
```

---

Warnings should be closely monitored in Catapult (warnings from the power analysis tools are also highlighted as warnings in the Catapult transcript).

## Power Analysis Flow Limitations

For all power flows:

1. The activity file does not capture activity for VHDL abstract types. For example, the activity of the FSM state is not captured because it's variable is an enumeration type. Otherwise, all other cases should be std\_logic or std\_logic\_vector (or signed/unsigned) and should be captured in the activity file.
2. The activity file may contain X-state activity that may have an impact in the accuracy of the estimate.

For SpyGlassPower flow only:

1. A clock control file is not generated by the flow, therefore clock trees will not be estimated for power.

For SpyGlassPower flow only:

1. NCSim VHDL RTL activity output will cause problems (Verilog RTL, Verilog/VHDL GATE Level should be fine).
2. Technology libraries that do not have a default wire load model will not work with Atrenta SpyGlass Power tool.

## Low Power Optimization

Catapult supports the following methods for reducing power consumption in a design:

- “Clock Gate Insertion” on page 589
- “FSM State Encoding” on page 591
- “Idle Signal Insertion” on page 592

### Clock Gate Insertion

Clock gates are commonly used on registers/memory elements to reduce switching and power consumption in low-power applications. The gates turn off clock signals to registers/memory elements when they are in an idle state to prevent needless switching. To get an estimate of power reduction related to clock gating for a design, see “[Estimating Clock Gate Power Savings](#)” on page 594.



#### Note

The Catapult gate insertion feature is licensed separately. Contact your Calypto Design Systems sales representative to obtain a license.

Depending on the low-power optimization level enabled, Catapult inserts explicit clock enable signals on registers and storage elements in a design, as shown in [Figure 10-2](#), so they can be gated by downstream RTL synthesis tools. As a rule, actual clock gate insertion is performed by RTL synthesis tools such as PowerCompiler.

**Figure 10-2. Explicit Clock Enable Signals**

#### VHDL

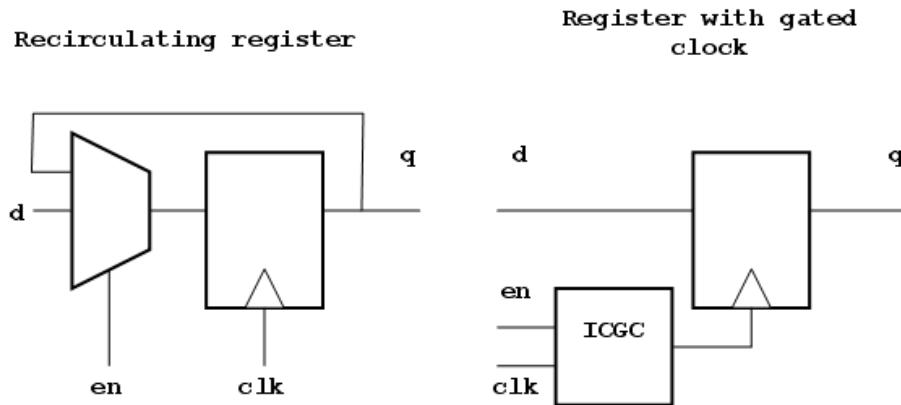
```
wait until clk'event and clk=1;
if (rst = '1') then
 q <= '0';
else if (en = '1')
 q <= d;
end if;
```

#### Verilog

```
always@(posedge clk)
begin
 if (rst == 1'b1)
 q <= 1'b0;
 else if (en == 1'b1)
 q <= d;
end
```

Then, during RTL synthesis, the explicit enable signal is used to implement either a recirculating register or a gated clock as shown in [Figure 10-3](#).

**Figure 10-3. Clock Gate Implementation in RTL Synthesis**



There are two levels of clock gating available in Catapult:

- **Normal** — Catapult performs combinational clock gating and uses clock-gated data storage core component models (FIFO/Register file/Channel) to place enable signals on all design sequential nodes.
- **High** — In addition to the *normal* effort optimizations, Catapult performs sequential clock gating by taking into account the clock cycles between registers and propagating enable signals to downstream gating control registers. This results in a strengthened or more selective enable condition. It may also result in additional flip flops being added to the design to store intermediate clock enable states. In addition to the choice of the clock-gated data storage elements, control registers holding the gating information for the main data storage elements are sequentially clock gated (in groups of eight, 1-bit registers).

You can also specify a minimum bit width value for both the normal and high levels of clock gate insertion. The minimum bit width option allows the exclusion of registers based on bit width to provide more control over the clock gating in a design.

The amount of power reduction delivered by clock gating is heavily dependant on the context of the candidate register and the test vectors applied to the design. You should analyze each candidate node in the design to determine whether clock gating would be beneficial. For more information, see “[Estimating Clock Gate Power Savings](#)” on page 594.

Depending on the application, the low-power options are set with architectural constraints at the top-level or process level of a design.

For information on using directives to set low-power optimization, see the [GATE\\_EFFORT](#), [GATE\\_EXPAND\\_MIN\\_WIDTH](#), [GATE\\_MIN\\_WIDTH](#), [GATE\\_REGISTERS](#) directives.

## Related Topics

[Architectural Constraints on the Design](#)  
[Architectural Constraints on Processes](#)

[I/O Components](#)

## FSM State Encoding

Catapult generates FSM (DPFSM) registers for datapath control. FSMs can potentially decrease power consumption by employing the following low-power state encoding methods:

- Binary encoding
- Gray code encoding
- One-hot encoding

The overall power savings due to low-power state encoding methods increase when the states in the DPFSM increase.



### Note

Currently this feature is only supported by the Design Compiler flow. The selected encoding method is passed to the Design Compiler as a synthesis constraint.

---

Use the [FSM\\_ENCODING](#) directive to enable this option. The directive can be set from the Catapult command line or from the GUI. The command syntax is:

```
directive set <path_to_block> -FSM_ENCODING <none|binary|grey|onehot>
```

where <path\_to\_block> is the design path to either the top-level block or a hierarchical sub-block. The default setting is “none.” Sub-blocks inherit the setting on the top block unless individually overridden. The following command example enables the directive for the top block of the hier\_dct design.

```
directive set /dct_proc -FSM_ENCODING onehot
```

To set the directive from the GUI, go to Architectural Constraints editor and select either the design top or one of the process blocks. Then choose one of the encoding methods from the drop-down menu in “FSM State Encoding” field.

To change the default setting of the FSM\_ENCODING directive, choose **Tools > Set Options...** pull-down menu item, then select **Project Initialization > Hardware** in the Catapult Options window. Modify the “FSM State Encoding” field. Or you can use the following command:

```
options set ProjectInit/Hardware DefaultFSMEncoding
<none|binary|grey|onehot>
```

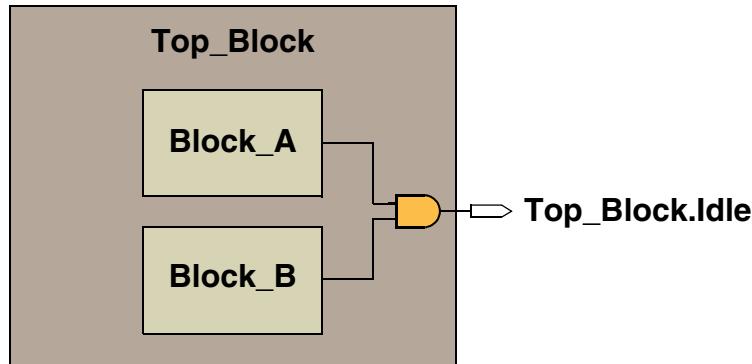
## Idle Signal Insertion

You can add idle indicator signals to process blocks in the design. An idle signal is driven HIGH if all three of the following conditions are true:

- The Catapult design is waiting for input data.
- No inputs are supplied to the design.
- All the output data has been read from the output ports of the design (non-pipelined designs and designs with distributed pipeline).

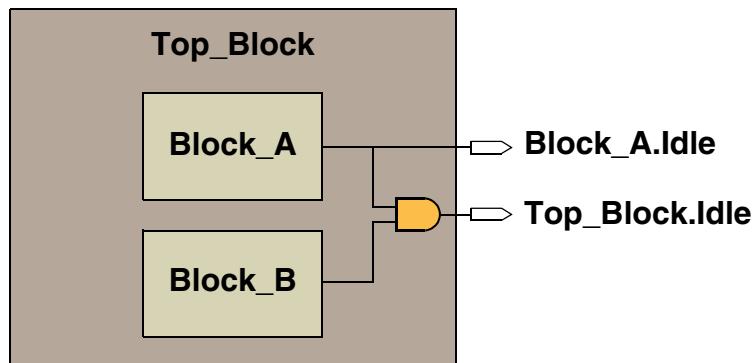
Enabling the Idle Signal option on the top block creates a *global* idle signal for the design by adding individual signals to all sub-blocks that feed into (ANDed together) the signal port at the top level. Figure 10-4 illustrates this.

**Figure 10-4. Global Idle Signal in Hierarchical Design**



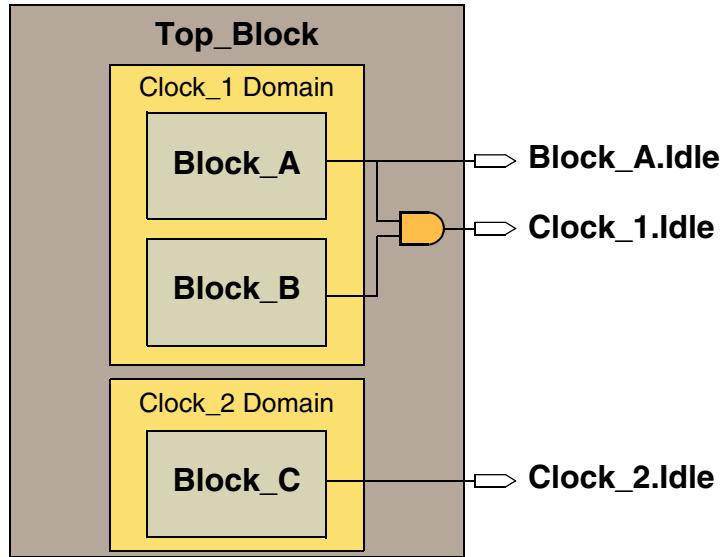
Separate idle signals can be enabled on individual sub-blocks as well as the top block, as illustrated in Figure 10-5. The idle signals from the sub-blocks are available individually at the design boundary, and they also feed into the global idle signal.

**Figure 10-5. Global and Block-Specific Idle Signals in Hierarchical Design**



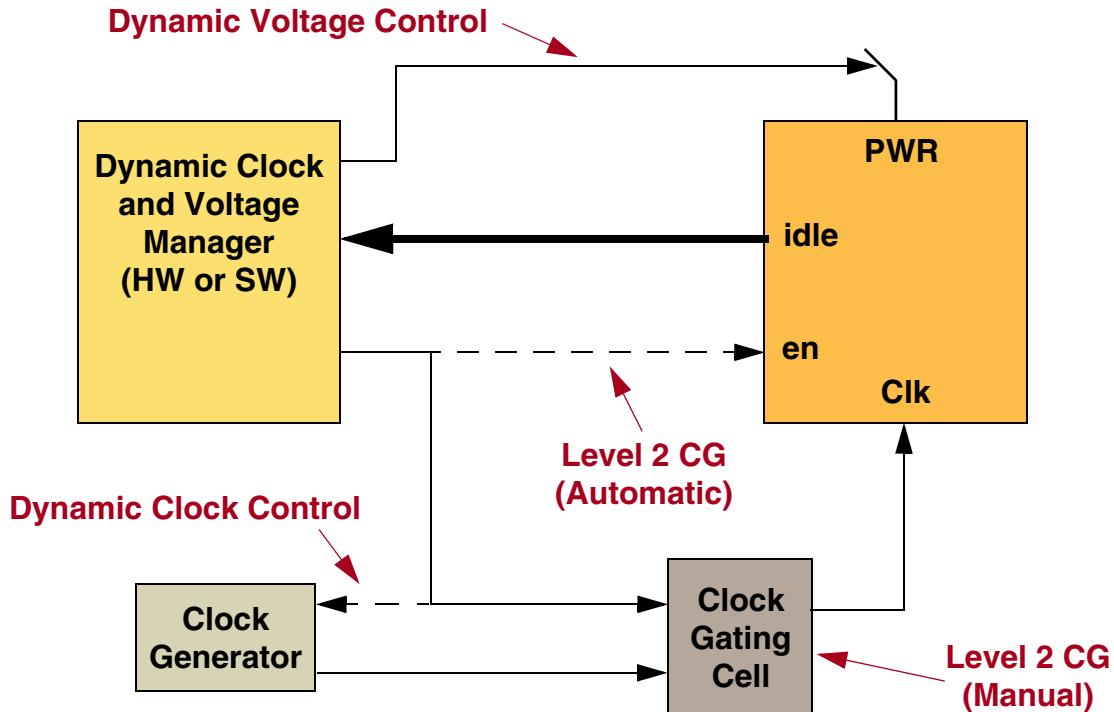
Enabling the top level idle signal on a design that uses multiple clocks will create separate signals for each clock domain. This is illustrated in Figure 10-6.

**Figure 10-6. Global and Block-Specific Idle Signals and Multiple Clock Domains**



Possible usage scenarios for the idle signal are illustrated in Figure 10-7.

**Figure 10-7. Idle Signal Synthesis**



Use the **IDLE\_SIGNAL** directive to add an idle signal to a process block and specify a string to be appended to the signal name. The directive can be set from the Catapult command line or from the GUI. The command syntax is:

```
directive set <path_to_block> -IDLE_SIGNAL <suffix_string>
```

where **<path\_to\_block>** is the design path to either the top-level block or a sub-process. The resulting signal name will have the form:

```
<process_name>.<suffix_string>
```

where **<process\_name>** is the name of the process on which the directive is being set. This form applies to all sub-processes, and for single clock designs it applies to the top block also.

**<suffix\_string>** is a user-defined string appended to the signal.

---

 **Note** The “.” separator in the signal name is replaced by an underscore character (“\_”) in the generated HDL netlist.

---

To unset the directive and remove the idle signal, use the same command, but omit the **<suffix\_string>** argument.

```
directive set <path_to_block> -IDLE_SIGNAL {}
```

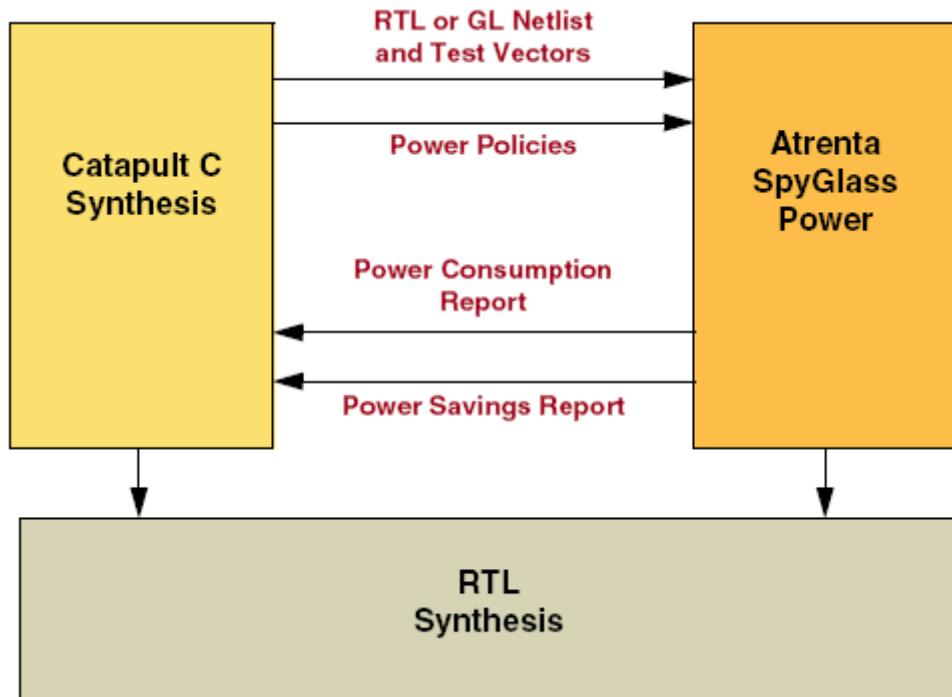
Example: The following command creates an idle signal on the Proc\_B sub-block in the design MyDesign. The resulting signal name is “Proc\_B.MyIdle.”

```
directive set /MyDesign/Proc_B -IDLE_SIGNAL MyIdle
```

## Estimating Clock Gate Power Savings

You can use the SpyGlassPower flow as shown in [Figure 10-8](#) to estimate the power savings provided by inserting clock gates in a design.

**Figure 10-8. Atrenta SpyGlassPower Flow**

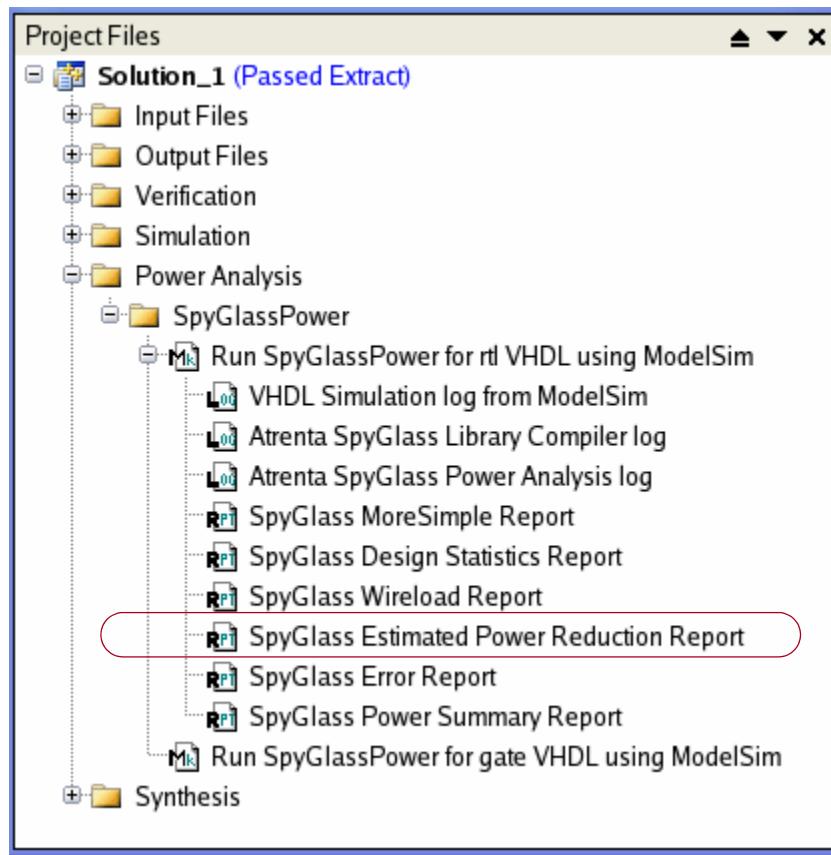


When the **GATE\_REGISTERS** directive is set to “true,” the SpyGlassPower flow includes the following policies when launching the SpyGlass Power tool. The first two apply to RTL power analysis, and the third policy applies to gate-level analysis. Refer to the Atrenta product documentation for more information about these policies.

- PEPWR03 - Estimates power savings due to explicit clock enables.
- PEPWR04 - Estimates power savings due to forward traversal and operand isolation.
- PEPWR05 - Estimates power savings due to already instantiated clock gates.

The SpyGlass Power tool generates a power savings report file, `pe_reduction.rpt`, which is saved in the Catapult solution directory. From the Catapult GUI, the report file is accessible from the Project Files window as shown in Figure 10-9. Double-click the file to view the report.

**Figure 10-9. Atrenta SpyGlassPower Flow Output Files**



The total clock gating power savings value reported in the pe\_reduction.rpt file is also included in the Catapult Table window, as shown in Figure 10-10.

**Figure 10-10. Clock Gating Power Savings Table Column**

Select “Atrenta SpyGlassPower”  
from the drop-down menu

| Solution /             | Leakage Power | Internal Power | Switching Power | Total Est Power | Clock Gating Power Savings |
|------------------------|---------------|----------------|-----------------|-----------------|----------------------------|
| ► Solution_1 (extract) | 164uW         | 738uW          | 4.74mW          | 5.64mW          | 1.81mW                     |
| ► Solution_2 (extract) |               |                |                 |                 |                            |

# Chapter 11

## CCORE Design Flow

---

A CCORE® (Catapult C Optimized Reusable Entities) is a custom operator synthesized from a single source code function that can be reused throughout the design. CCOREs provide several design improvements:

- Catapult runtime is shorter because the logic in a CCORE is optimized and synthesized once and reused.
- Area results are typically better because coarse-grain sharing optimization is more efficient.
- Better design performance is achieved when CCOREs are characterized by a downstream RTL synthesis tool during the Catapult design cycle. More accurate timing and area data enables Catapult to do a better job of optimizing the design.

Any function definition can become a CCORE with the `hls_map_to_operator` pragma. Catapult detects the `map_to_operator` pragma and creates a separate design partition for the function similar to the way hierarchical blocks are partitioned.

The `map_to_operator` argument must specify either the name of a valid CCORE component in an available library or the `CCORE` keyword. The value of the pragma determines the method Catapult uses to create the CCORE. For more information, see the following topics:

- [Top-Down CCORE Methodology](#) — creates CCOREs on the fly from function definitions.
- [Bottom-Up CCORE Methodology](#) — synthesizes a function directly into a Catapult library component that can then be targeted in a source code design.

The design in [Example 11-1](#) shows the two `map_to_operator` pragma statements. In both statements, the `map_to_operator` argument specifies a component name (`max_min` and `max_min_cross`). These CCORES were created with the bottom-up CCORE flow and saved in Catapult libraries. [Figure 11-1](#) shows how this design displays in the Architectural Constraints window. CCOREs are denoted by the  icon.

### Example 11-1. CCORE Example Design

```
#include "ccore_flow.h"

#pragma map_to_operator max_min
void max_min (Data & a, Data & b, Data & max, Data & min)
{
 Flag a_gt_b = a > b;
 max = a_gt_b ? a : b;
```

```
min = a_gt_b ? b : a;
}

#pragma map_to_operator max_min_cross
template <int ID>
void max_min_cross (Data & a, Data & b, Data & max,
 Data & min, Flag & cross)
{
 static Flag a_gt_b_prev = 0;
 Flag a_gt_b = 0;

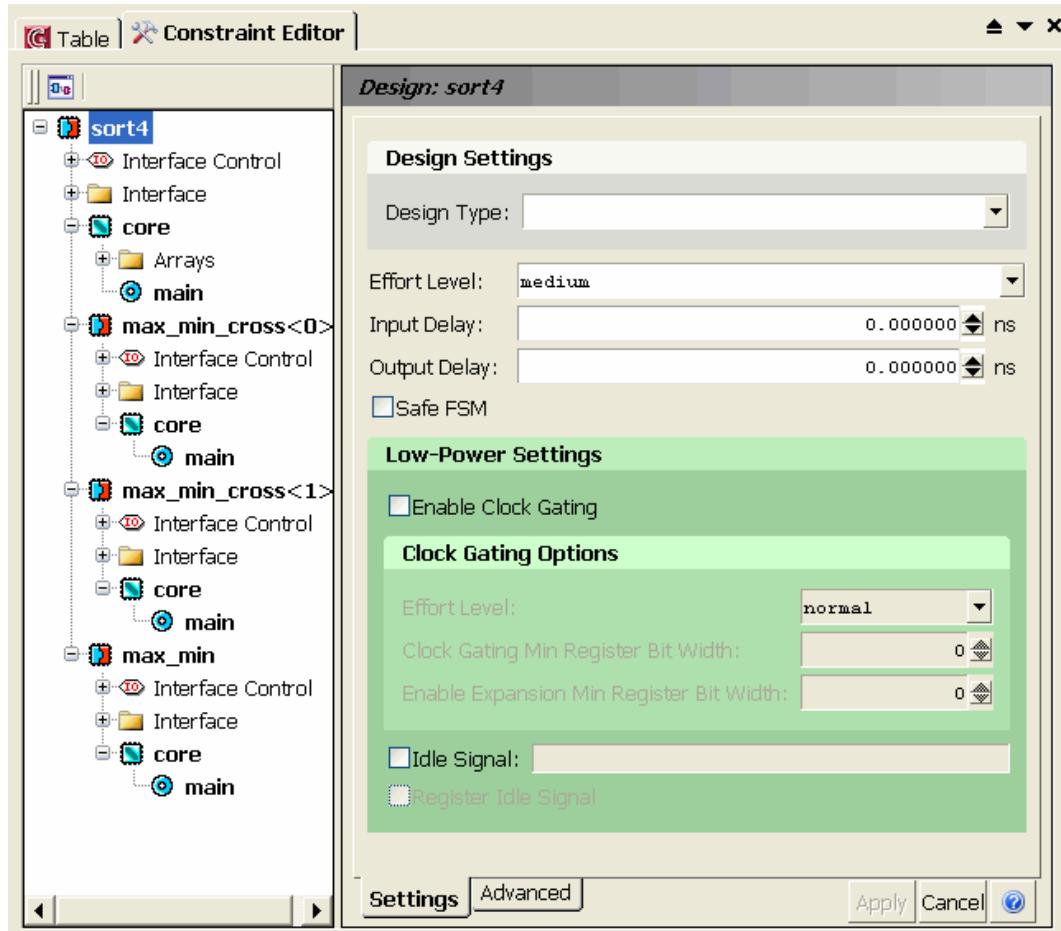
 a_gt_b = a > b;
 max = a_gt_b ? a : b;
 min = a_gt_b ? b : a;
 cross = a_gt_b ^ a_gt_b_prev;
 a_gt_b_prev = a_gt_b;
}

#pragma design top
void sort4
(
 Data & a,
 Data & b,
 Data & c,
 Data & d,
 Data & first,
 Data & second,
 Data & third,
 Data & fourth,
 Flag & cross_ab,
 Flag & cross_cd
)
{
 Data max_ab, min_ab, max_cd, min_cd;
 Data first_abcd, min_max_abcd, max_min_abcd, fourth_abcd;
 Data second_abcd, third_abcd;
 Flag x_ab, x_cd;

 max_min_cross<0>(a, b, max_ab, min_ab, x_ab);
 max_min_cross<1>(c, d, max_cd, min_cd, x_cd);
 max_min(max_ab, max_cd, first_abcd, min_max_abcd);
 max_min(min_ab, min_cd, max_min_abcd, fourth_abcd);
 max_min(min_max_abcd, max_min_abcd, second_abcd, third_abcd);

 first = first_abcd;
 second = second_abcd;
 third = third_abcd;
 fourth = fourth_abcd;
 cross_ab = x_ab;
 cross_cd = x_cd;
}
```

**Figure 11-1. CCOREs in the Architectural Constraints Window**



## CCORE Restrictions

The CCORE flow has the following limitations:

- Rolled loops inside of a CCORE cannot contain:
  - Conditional breaks. This causes a design to fail.
  - Variable bounds from inputs. Rolled loops must have constant bounds.
- Combinational components cannot contain static variables.
- Memories on the interface are not supported.
- Only wire interface components (mgc\_in\_wire for inputs and mgc\_out\_std\_reg for outputs) are allowed.
- Inout ports are not supported. Use an explicit output assignment at the end of the CCORE function to avoid false errors.

- OUTPUT\_REGISTERS require a latency greater than 2. Latency=1 results in a combinational path from input to output, which is unsupported.

## CCORE Variables

The following variables are added to the partition interface of the CCORE:

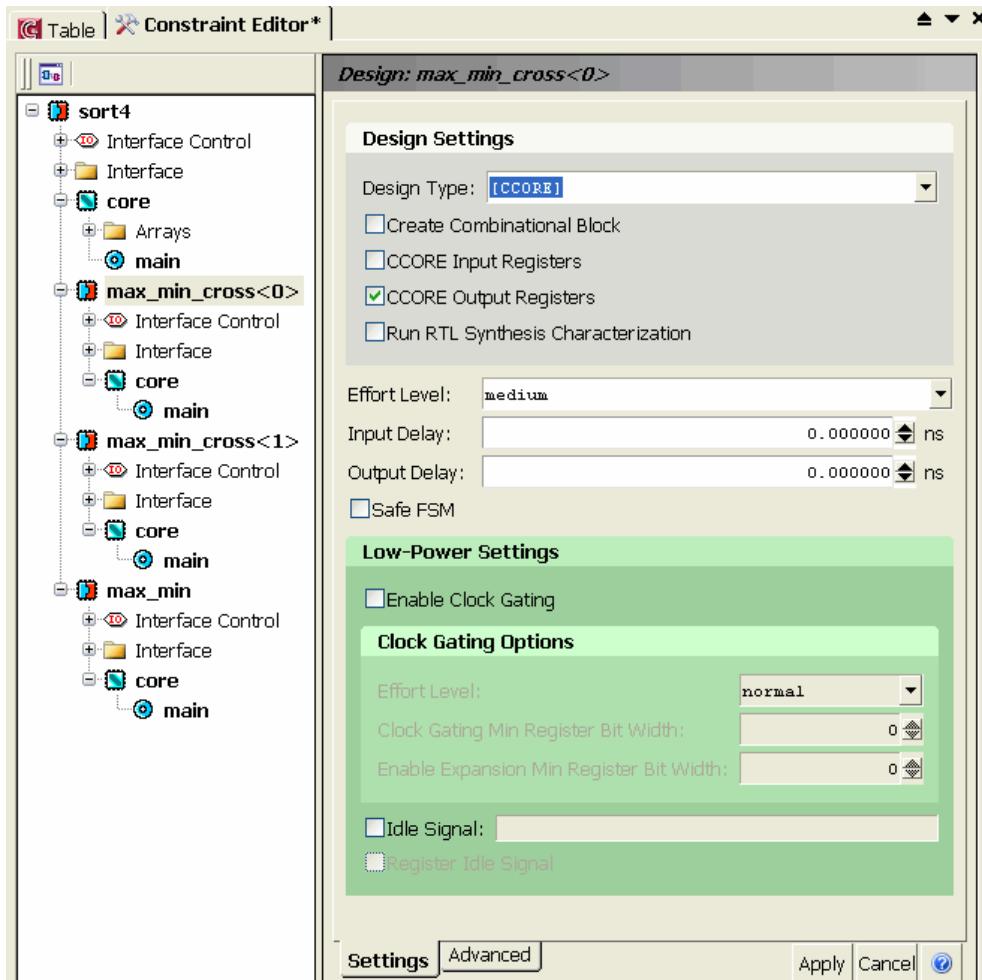
- **mgc\_ccore\_en** — this variable provides a global enable signal that is active high.
- **mgc\_ccore\_start** — this variable identifies the partition as a CCORE. The mgc\_ccore\_start variable is removed from the interface when the following CCORE conditions exist:
  - II=1
  - Throughput =1
  - no 'state'
  - clock gating is turned off

## Top-Down CCORE Methodology

In the top-down method, CCORE functions are individually synthesized in a single synthesis run along with the rest of the design.

For example, [Figure 11-2](#) shows the design in [Example 11-1](#) in the Constraint Editor window. When the max\_min\_cross function is selected, you can set the Design Type to CCORE. The equivalent command syntax is:

```
directive set /sort4/max_min -MAP_TO_MODULE { [CCORE] }
```

**Figure 11-2. Top-Down CCORE Architectural Constraints**

When the Design Type is set to CCORE, the Constraint Editor window expands to display additional CCORE options as follows:

- **CCORE Input Registers** — Determines if registers are created on the CCORE inputs. By default, no input registers are created. The INPUT\_REGISTERS directive controls this option.
- **CCORE output registers** — Determines if registers are created on the CCORE outputs. By default, output registers are created.
- **Run RTL Synthesis Characterization** — Synthesizes the CCORE with the downstream RTL synthesis tool and uses the characterization data instead of estimated data. The CHARACTERIZE\_OPERATOR directive controls this option.

Then, during the *go architect* stage, each specified CCORE function is synthesized in a separate solution for the current project and an RTL netlist and Catapult library component are generated. The solutions are named after the CCORE functions. If enabled, the downstream

RTL synthesis tool is also run to obtain characterization data at this time. Then Catapult returns to the top design, automatically maps the functions to their CCOREs, and continues with synthesis. Each CCORE solution is accessible in the current Catapult session.

Every time the design is resynthesized, all of the top-down CCOREs are also resynthesized (and recharacterized if that option is enabled) to ensure that the CCOREs are up-to-date with the design. After a CCORE is fully developed, it can be saved in a Catapult library and targeted by the source code in a design.

---

**Note**

---

 In the top-down CCORE method, you cannot add or modify CCORE constraints after the *compile* stage of the Catapult work flow. If you need to change CCORE constraints after the compile stage, use the bottom-up CCORE methodology.

---

## Bottom-Up CCORE Methodology

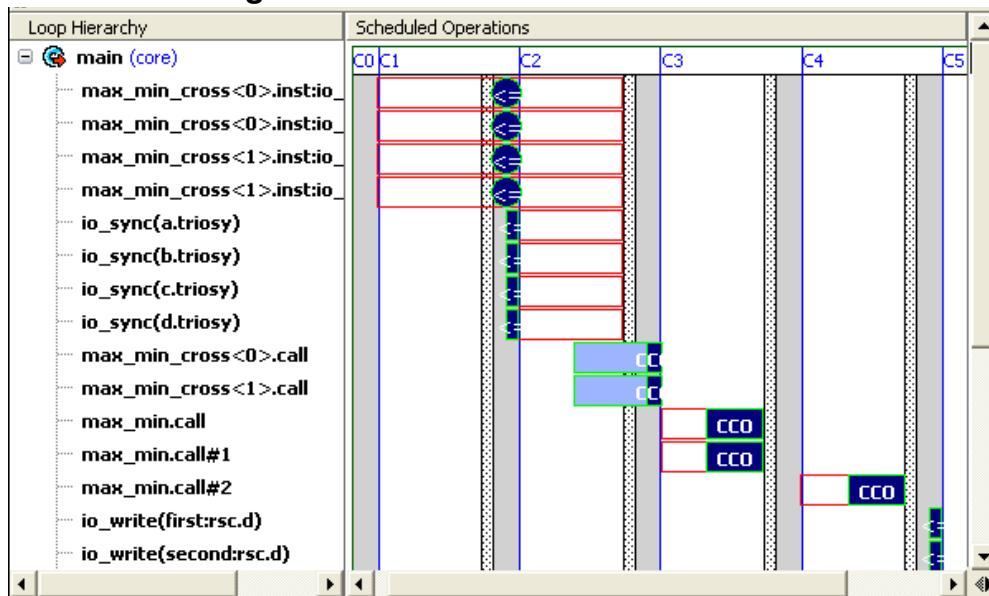
The bottom-up method utilizes pre-synthesized and characterized CCOREs saved in Catapult libraries. The advantage of using the bottom-up method is that the CCOREs do not need to be synthesized with the design. Instead, they are represented in the design as black boxes. The bottom-up procedure is as follows:

1. Create a CCORE library component as described in “[Creating CCORE Component Libraries](#)” on page 603. The general process is to first synthesize the target function as a stand-alone design. Next, save the resulting solution as a component in a Catapult library.
2. Modify the `hls_map_to_operator` pragma in the larger design to utilize the CCORE component. The `hls_map_to_operator` argument must specify the component name exactly as it exists in the library.
3. Add the CCORE library to the Component Libraries search path in Catapult. Choose **Tools > Set Options**. Then expand the Project Initialization group and select Component Libraries. Add the path to the Library Search Path field.
4. Create a new project in the Catapult session by executing the `project new` command.
5. When setting the `TECHLIBS` directive, include the new library in the list of component libraries. From the GUI, use the Setup Design constraints editor to select the new library in the Compatible Libraries field.
6. In the Architectural Constraints editor, verify that the Design Type is set to the correct library component.
7. Run the Catapult flow as usual. Catapult automatically maps the CCORE component to the corresponding function.

## CCOREs in the Gantt Chart

CCORE objects in the Gantt chart are identified by the CCO label. [Figure 11-3](#) shows the Gantt chart for the design from [Example 11-1](#). Notice that the max\_min\_cross CCOREs are full-cycle components while max\_min CCOREs are combinational and have red slack bars.

**Figure 11-3. Gantt Chart with CCOREs**



## Creating CCORE Component Libraries

This flow creates CCOREs directly from a source code function without using any existing RTL. You can also use the bottom-up CCORE flow to synthesize a function directly into a library component within Catapult. This flow allows you to implement simple combinational and sequential blocks as operators that can be used in a more complex design.

The [solution netlist](#) command has a *-library* switch that enables Catapult to generate component libraries from source code functions. With this feature, component libraries are synthesized from Catapult generated output. The libraries can then be used by Catapult along with other technology libraries

The procedure for creating library components from source code is a two-pass process. The first pass through Catapult generates the libraries. The second pass uses the libraries in the standard Catapult work flow.

|                                       |     |
|---------------------------------------|-----|
| Creating the Library . . . . .        | 604 |
| Sequential Components . . . . .       | 604 |
| Combinational Components . . . . .    | 604 |
| Running Downstream Synthesis. . . . . | 604 |
| Running Downstream Synthesis. . . . . | 604 |

|                                                                         |     |
|-------------------------------------------------------------------------|-----|
| Netlisting the Component and Setting Library Component Properties ..... | 605 |
| Examples.....                                                           | 605 |

## Creating the Library

Use the following procedure to create a CCORE library component directly from a source code function using Catapult:

1. Create a source code function that adheres to “[CCORE Restrictions](#)” on page 599.
2. Add the source code file that contains the target function to the Catapult Input Files.
3. Set technology constraints and the top-level design to the target function. By default, the library feature operates on the top-level function in the design. If the target function is not the default top, use the [DISTRIBUTED\\_PIPELINING](#) directive to make it the top-level function.  
Set the [CREATE\\_COMBINATIONAL](#) directive if creating a combinational component.
4. Set the architectural constraints.
5. Generate the RTL. You can also run a downstream synthesis tool to get area and timing data for the component instead of using the estimates provided by Catapult.
6. Generate the library with the [solution netlist -library](#) command. The CCORE library component (<design\_name>.lib) is automatically generated. The output path for the generated library is relative to the project directory unless an absolute path is specified.

## Sequential Components

Catapult allows you to build basic sequential components with *wire* interfaces to ensure that components never stall. The top-level function is automatically pipelined with Initiation Interval (II) = 1 (required), but internal loops can be left rolled or partially rolled.

## Combinational Components

Catapult allows you to create combinational components such as Galois field multipliers. The [CREATE\\_COMBINATIONAL](#) directive must be set to true in order for Catapult to create a combinational component. When this directive is issued, Catapult automatically unrolls all loops in the source function and removes all output registers.

## Running Downstream Synthesis

The CCORE flow uses the area and delay information from Catapult. A more accurate area and timing information can be obtained by running the standard RTL synthesis flow before netlisting the component. That CCORE option automatically annotates the area and timing results. If the RTL synthesis returns zero area, the Catapult area and timing is used.

# Netlisting the Component and Setting Library Component Properties

The component and library are created using the `solution netlist` command with the `-library` switch.

```
Solution netlist -library ?<switches>? <filename>
```

Switches:

- input\_register <bool>**: Determines if input ports on the library module have input registers. By default, `input_register=true` and `SeqDelay` is adjusted automatically.
- property\_map**: Adds a property to the All binding on the module and allows you to override the properties shown in the following table. For more information about these property mappings, refer to the *Catapult C Library Builder User's and Reference Manual*.

**Table 11-1. `solution netlist` Netlisting Properties**

| Combinational | Sequential |
|---------------|------------|
| Area          | Area       |
| Delay         | SeqDelay   |
|               | InitDelay  |
|               | MinClkPrd  |

To override the default property mappings for area and timing values, use the following command format:

```
solution netlist -library -property_map Area <value>
 -property_map SeqDelay <value>
 -property_map InitDelay <value>
 -property_map MinClkPrd <value> replace
my_add/comp.lib
```

## Examples

|                                                       |     |
|-------------------------------------------------------|-----|
| Example 1: Generating a Sequential Component.....     | 606 |
| Example 2: Generating a Combinational Component ..... | 606 |
| Example 3: Using the Custom Library .....             | 607 |
| Example 4: Component with State .....                 | 607 |

The first three examples in this section are based on the following C++ design in the file `my_add.cpp`. The first two create custom components generated from the `my_add` function. In the third example, the `hls_map_to_operator` pragma instructs Catapult to use a library component named `my_add` in place of the `my_add` function.

```
#include "ac_int.h"

#pragma hls_map_to_operator my_add
ac_int<8> my_add(ac_int<8> inputs, ac_int<8> coeffs)
{
 return inputs + inputs + inputs + coeffs;
}

#pragma hls_design top
void test(ac_int<8> inputs, ac_int<8> coeffs, ac_int<8> *outputs) {
 *outputs = my_add(inputs, coeffs);
}
```

## Example 1: Generating a Sequential Component

The following script loads the my\_add.cpp file and generates the my\_add component in the library named <project\_dir>/my\_add/comp.lib:

```
project new
solution file add ./my_add.cpp
go analyze
directive set -TECHLIBS {{mgc_sample-090nm_beh_dc.lib \
 mgc_sample-090nm_beh_dc}}
directive set -CLOCK_PERIOD 0.2
directive set -DESIGN_HIERARCHY my_add

go compile
directive set /my_add -MAP_TO_MODULE {[CCORE]}
directive set /my_add/core/main -PIPELINE_INIT_INTERVAL 1
go extract
solution netlist -library -replace my_add/comp.lib
```

HDL files generated by the solution netlist command are written to the <project\_dir>/my\_add directory along side the comp.lib library. They are also added as netlist dependencies.

By default, the library generation feature operates on the top-level function in the design. In our example, therefore, the *directive set -DESIGN\_HIERARCHY my\_add* command is required to override the #pragma hls\_design top statement in the C++ design file and make the my\_add function the top-level function.

For information about how to override the default property mappings for area and timing values, refer to “[Netlisting the Component and Setting Library Component Properties](#)” on page 605.

## Example 2: Generating a Combinational Component

The following script is essentially the same as the script in the previous example. The key difference is the use of the CREATE\_COMBINATIONAL directive. By default, Catapult generates a sequential component unless this directive is set. When the directive is set, all loops are unrolled and no registers are used in the generated component.

```
project new
```

```
solution file add ./my_add.cpp
go analyze
directive set -TECHLIBS {{mgc_sample-090nm_beh_dc.lib \
 mgc_sample-090nm_beh_dc}}
directive set -CLOCK_PERIOD 1
directive set -DESIGN_HIERARCHY my_add
directive set -CREATE_COMBINATIONAL true
go compile
directive set /my_add -MAP_TO_MODULE {[CCORE]}
go extract
solution netlist -library -replace my_add/comp.lib
```

For information about how to override the default property mappings for area and timing values, refer to “[Netlisting the Component and Setting Library Component Properties](#)” on page 605.

### Example 3: Using the Custom Library

The following script starts a new project, adds the generated library to the TECHLIBS, and generates RTL.

```
project new
solution file add ./my_add.cpp
directive set -TRANSACTION_DONE_SIGNAL true
go analyze
directive set -TECHLIBS {{my_add/comp.lib} \
 {mgc_sample-090nm_beh_dc.lib \
 mgc_sample-090nm_beh_dc}}
directive set -CLOCK_PERIOD 10
go extract
```

### Example 4: Component with State

This example shows how to create a component with *state*. If a C++ function uses static variables, it is considered to have *state* when the component is netlisted. The following example adds state to the adder tree example along with a signal to clear the state:

```
#pragma map_to_operator "add_tree_state"
ac_int<16> add_tree_state(ac_int<8> din[8], bool clr) {
 static ac_int<16> tmp=0; //STATE
 if(clr)//clear state
 tmp = 0;
 for(int i=0;i<8;i++)
 tmp += din[i];
 return tmp;
}

#pragma design top
void test_add_tree_state(ac_int<8> din[8], bool clr, ac_int<16> &dout) {
 dout = add_tree_state(din,clr);
}
```



# Chapter 12

## Flow Customization

---

A powerful feature of Catapult is its customizable design flow which enables tighter integration with your downstream tools and preferred design methods. You can define custom flows that launch other programs, generate specialized output files, or post-process output files. And you specify the design stage(s) at which each custom behavior occurs.

You implement your custom flows in the form of Tcl procedures. Related flows are grouped together in a *flow package* file (.flo file extension). The flow package also includes Catapult “[flow](#)” commands with which you load and manage the flows. The Catapult flow package management system is very similar to the Tcl package management system, employing the same *provide/require* usage model.

All flows will need to access data about the current design in order to dynamically construct commands that subsequently modify the design. Your flow should use the *FLOWS\_API* database for that purpose. Refer to “[FLOWs\\_API Database](#)” on page 615 for a description of the database and how the data is organized.

## Flow Package File Outline

The file outline below illustrates the basic organization and features of a flow package file.

```
1 ## Declare package name and version
2 flow package provide /My_Package 2.12
3
4 ## Define sub-processes for flow procedure(s)
5 proc write_file_header (args) {
6 ...
7 }
8
9 proc write_file_footer (args) {
10 ...
11}
12
13 ## Declare flow gen_report. The options specify that the flow is
14 ## enabled and it is available at any stage of the design process.
15 flow provide /My_Package/gen_report -enabled {all} -states {any}
16
17 ## Define flow procedure gen_report
18 proc gen_report (args) {
19 ...
20 write_file_header $arg_list
21 ...
22 write_file_footer $arg_list
23}
24
```

```
25 ## Access flows from another package
26 flow package require /Utility_Package -exact 4.3
27
28 ## Call flow from another package: /Utility_Package/generate
29 /Utility_Package/generate
```

The first statement, **flow package provide** (on line 2), is required as it declares the name and version of the package. Other optional information about the author of the flow can also be provided by the **flow package provide** command. Catapult searches for that statement when constructing its index of available packages. See “[Loading Flows](#)” on page 610 for more information about indexing

The next block (lines 5 to 10) defines sub-processes that are called by the flow process. The sub-processes cannot be accessed directly from within Catapult.

On line 15 the **flow provide** statement declares that procedure `gen_report` is a flow, that it is enabled by default, and that it is available at any stage of the design flow. Alternatively, a flow can be associated with specific stage(s). By default the flow will not be associated with any stages, making it inaccessible from the command line or from other packages. For information about the stages, refer to “[The Catapult Work Flow](#)” on page 46.

Lines 18 to 23 define the `gen_report` flow procedure.

The **flow package require** statement on line 26 declares a dependency on the flow package `Utility_Package` and makes the `Utility_Package` flows available to this package.

Line 29 calls the flow `/Utility_Package/generate`. Calling a procedure in another package requires that the other package has already been indexed. An alternative method of accessing flows in other packages is to *source* the other package into the current package. To source a package, you call the **flow package require** command with the **-source** switch. In that case you can call the flow procedure without specifying its package hierarchy. For example, lines 25 to 29 of the previous example would be written as follows:

```
Inline (source) flows from another package
flow package require -source /Utility_Package -exact 4.3

Call generate flow in package Utility_Package
generate
```

---

**Note**

 When sourcing other packages you must ensure that procedure names and global variable names do not collide with those in the current package.

---

## Loading Flows

Loading flows into Catapult involves the steps listed below. See also “[Safe Interpreters](#)” on page 613 for information about which flow commands are available in the context of each step.

1. Add the pathnames of your flows directories to the Flow Search Path in Catapult. This can be done interactively in the tool, or automatically via a startup script.

- Interactive Method:

- i. Start Catapult and prior to setting up a project, add the path to your flows directory to the Flow Search Path as follows:

**Tools > Set Options... > Flows > Flow Search Path:**

Or issue the command

```
options set Flows FlowSearchPath <dir_containing_flow_files>
```

- ii. Start a new project in order to load the new search path.

- Catapult Initialization File Method:

Add the following command to a Catapult startup script (See “[The Catapult Initialization File](#)” on page 224):

```
options set Flows FlowSearchPath \
 [lappend [options get Flows FlowSearchPath] \
 "<dir_containing_flow_files>"]
```

When Catapult initializes a new project, it scans all of the flow files (.flo) in the search path and constructs an index of the flow packages it finds. The flows are not loaded at this time, but the index enables Catapult to load them dynamically when they are required. Note that Catapult searches its default flow directories before the directories in the flow search path.

The flow indexing keys off of the [flow package provide](#) command inside the flow files. The [flow package provide](#) command specifies the package name and version(s). You can issue the [flow package names](#) command to see the list of packages that are currently indexed.

2. Issue the command [flow package require](#) <your\_package> to make the flow procedures available in the current Solution. The actual Tcl code is not loaded, but is read from the flow file every time the flow is run.

The [flow package require](#) command makes the flow procedures accessible from the current stage of the design process (such as *new*, *compile*, *extract*, etc.) and all subsequent stages. The flow is not accessible to prior stages. You must issue the [flow package require](#) command at or before the stage at which the flow will be used. In most cases, you can simply issue the command immediately after the project is created, thereby making it accessible from the first stage (*new*) and all other stages.

## Simple Flow Example

Below is a very simple example of valid flow package and instructions on how to run it a Catapult session.

```

flow package provide /My_Package 1.23

proc proc1 {} {
 puts "#####
 puts "Hello From the My_Package Flow #"
 puts "#####"
}

flow provide /My_Package/gen_report -STAGES {any} -ENABLED {extract}

proc gen_report {} {
 flow provide /My_Package/proc1 -STAGES {any}
 proc1
}

```

To use this flow from the Catapult “thirdparty/flows” directory, do the following:

1. Save the example to a file named “My\_Package.flo” in the “<install\_dir>/pkgs/thirdparty/flows” directory. The “.flo” file extension is required.
2. Launch a Catapult session.
3. Load the package:

```
flow package require My_Package 1.23
```

4. Load a design and progress to the “extract” stage:

```
go extract
```

The flow “/My\_Package/gen\_report” will automatically execute during the extract stage. Look in the transcript (and/or log file) for the message:

```
#####
Hello From the My_Package Flow
#####
```

To use this flow from a directory other than the Catapult “thirdparty/flows” directory, do the following:

1. Save the example to a file named “My\_Package.flo” in your directory.
2. Start a Catapult session and add the path to your flow directory to the Flow Search Path:

```
options set Flows FlowSearchPath <dirContaining_flow_package_file>
```

3. Create a new project so that Catapult will locate and index your flow package.

```
project new
```

4. Load the package:

```
flow package require My_Package 1.23
```

5. Load a design and progress to the “extract” stage:

```
go extract
```

The flow “/My\_Package/gen\_report” will automatically execute during the extract stage. Look in the transcript (and/or log file) for the message:

```

Hello From the My_Package Flow #
#####
```

## Safe Interpreters

To protect the integrity of the Catapult system, flows are launched in a Tcl *safe interpreter*. A safe interpreter is a Tcl environment with a limited set of commands. In fact, Catapult uses three different safe interpreters for handling flows. The command set allows read-only access to the design database. Each safe interpreter provides a base set of Tcl commands and a subset of Catapult commands.

---

**Note**

 For more information about the base set of Tcl commands available in a safe interpreter, refer to the *Tcl Reference Manual*. An online copy is installed in the Catapult software in the directory. On UNIX, add <install\_dir>/pkgs/tcl\_msg/man to your MANPATH search path. On Windows, the text is a WinHelp file in the directory <install\_dir>/pkgs/tcl\_msg/doc.

---

The following list describes the commands of interest available in each safe interpreter:

1. The *package-indexing* safe interpreter supports the following flow commands:

|                                                                                                          |                                                                                                                  |
|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| <code>flow package names</code><br><code>flow package provide</code><br><code>flow package script</code> | <code>flow package vcompare</code><br><code>flow package versions</code><br><code>flow package vsatisfies</code> |
|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|

2. The *flow-indexing* safe interpreter supports the same flow commands as the package-indexing interpreter, plus the following commands.

|                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------|
| <code>flow package option add</code><br><code>flow package option get</code><br><code>flow package option set</code> |
|----------------------------------------------------------------------------------------------------------------------|

3. The *run-flow* safe interpreter is for executing flows. It supports all flow commands *except* for the following:

|                                                                                                                         |                                                                |
|-------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| <code>flow package option add</code><br><code>flow package option set</code><br><code>flow package option remove</code> | <code>flow package provide</code><br><code>flow provide</code> |
|-------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|

It also supports the following list of Catapult commands:

|                       |                                 |
|-----------------------|---------------------------------|
| application get       | solution file add -output -type |
| directive get         | utility file launch             |
| library get           | utility fileaddexecutebit       |
| options get           | utility path hard               |
| project get           | utility path isabsolute         |
| solution get          | utiltiy path soft               |
| solution get_property | utility relative                |

The following Tcl built-in commands are also in the *run-flow* interpreter:

|                |                 |                |
|----------------|-----------------|----------------|
| exit           | file nativename | file separator |
| file dirname   | file normalize  | file split     |
| file extension | file pathtype   | file tail      |
| file join      | file rootname   | open           |

---

**Note**

 The `exit` command only exits the interpreter and not the application.

---

## Package Versioning

A group of flows can be combined into a single flow package, which can be versioned. You assign version numbers to packages with [flow package provide](#) command.

### Version Format

Version numbers are string values consisting of numbers, letters, ‘.’ and ‘\_’ characters. Catapult converts the string to a numerical value internally in order to do ordinal comparisons of versions. The following example shows how an alpha-numeric version string is represented internally

“2010a LB” = 2010.n.nn

The ‘a’ and “LB” fields are each converted to a number internally, represented on the right by  $n$ , and the space in “a LB” is changed to a dot. A dot is inserted to separate numbers from adjacent alphabetic or ‘\_’ characters. Alphabetic characters are case *sensitive*.

The left-most field is called the *major* number. Larger numbers correspond to later versions of a package, left-most numbers having the greatest significance. For example, version 2.1 is later than 1.3, and version 3.4.6 is later than 3.3.5. Missing fields are equivalent to zeroes. So version 1.3.0.2 is a later version than 1.3 (equivalent to 1.3.0.0).

A later version number is assumed to be compatible with earlier versions as long as both versions have the same major number. For example, Tcl scripts written for version 2.3 of a package should also work unchanged with versions 2.3.2, 2.4, and 2.5.1. Likewise, if code is compatible with version 2.3, it is not necessarily compatible with versions 1.7.3 or 3.1. If

changes made to a package make it incompatible with scripts written for the previous version, increment the major number field of the new version.

## Accessing Versions

You load a package by issuing the `flow package require` command. If you do not specify a version, Catapult will load the latest version it has indexed. To load an older version of a package, you must use the `-exact` switch and specify the version string. To get a list of all versions of a package in the index, issue the command:

```
flow package versions <package_name>
```

## FLOWS\_API Database

The FLOWS\_API database is a branch in the Catapult Solution database that contains information about the structure and implementation of the design (ports, resources, variables, technology, etc.). It is a central location where flow scripts can access all information about a design. The data is retrieved by using the “`solution get`” command. Each solution in the database has a FLOWS\_API node.

Figure 12-1 is a simple diagram of the database structure. It shows the essential elements that define the database hierarchy. The many key/value pair elements that are not shown in the figure are described later in this section.

**Figure 12-1. Nodes of Interest in the FLOWS\_API Database Hierarchy**

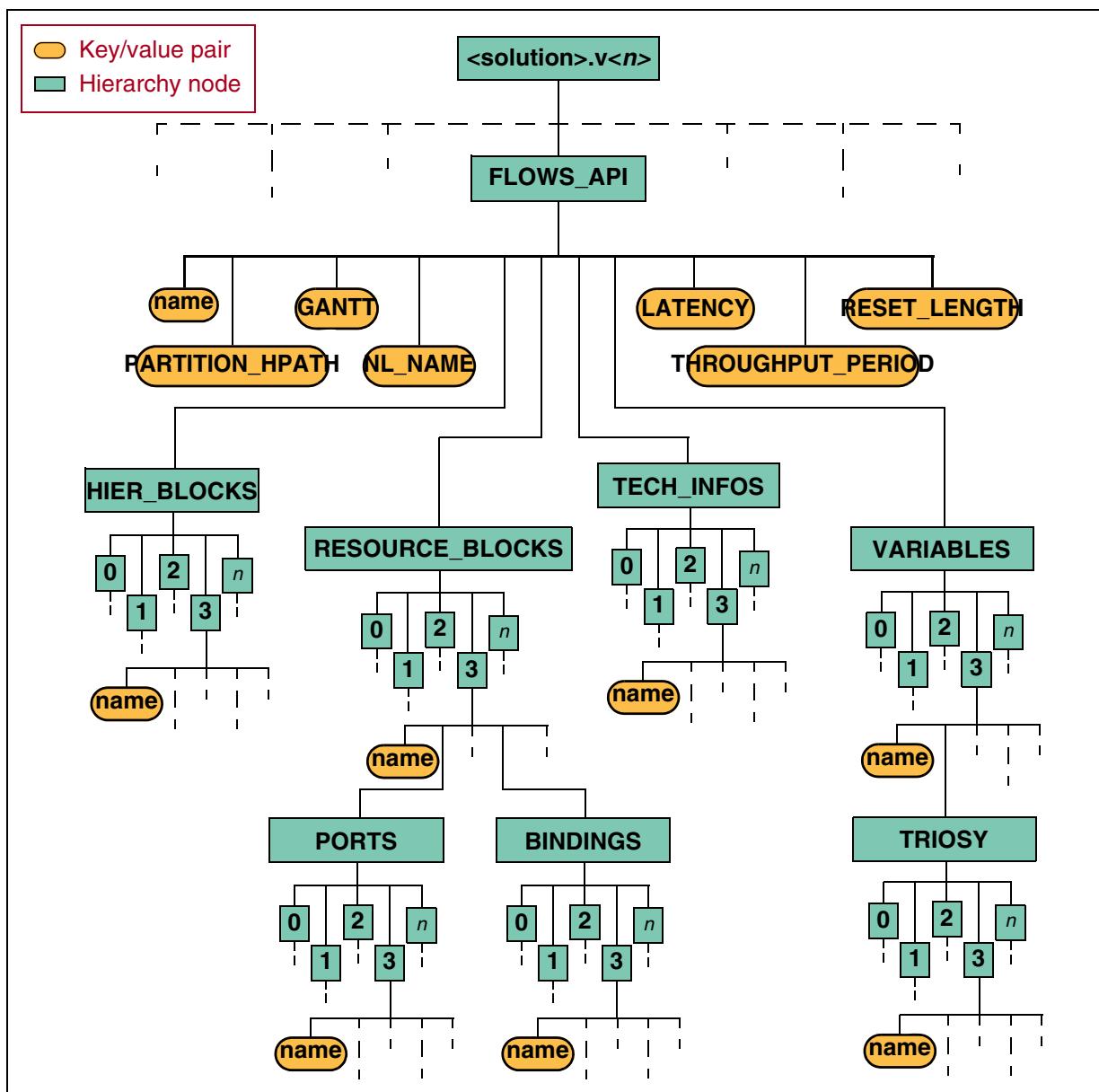
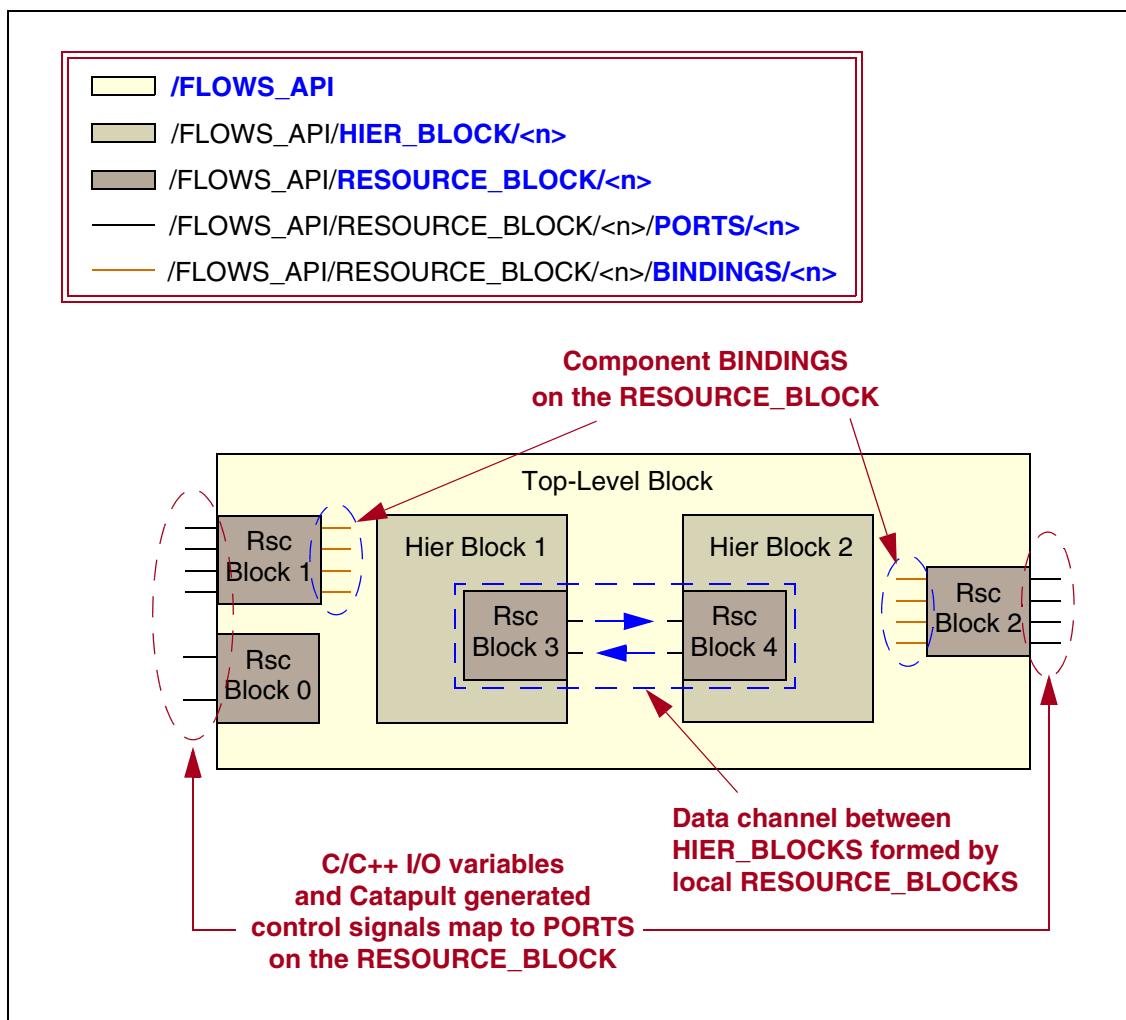


Figure 12-2 illustrates the correspondence of the primary database paths to elements in a design.

**Figure 12-2. Database Paths and Corresponding Design Blocks**



## FLOWS\_API Database Reference Tables

The following tables list the names and a brief description of the data elements contained in each of the hierarchical nodes in the FLOWS\_API database. A small number of elements in the database that are only for internal factory use are omitted from the tables.

|                                                                                            |                     |
|--------------------------------------------------------------------------------------------|---------------------|
| <a href="#">Table 12-1: Data Elements in the Top Level Node and HIER_BLOCK Nodes .....</a> | <a href="#">618</a> |
| <a href="#">Table 12-2: Data Elements in the RESOURCE_BLOCKS Node .....</a>                | <a href="#">619</a> |
| <a href="#">Table 12-3: Data Elements in the PORTS Node .....</a>                          | <a href="#">620</a> |
| <a href="#">Table 12-4: Data Elements in the BINDINGS Node .....</a>                       | <a href="#">622</a> |
| <a href="#">Table 12-5: Data Elements in the PROPERTIES Node .....</a>                     | <a href="#">622</a> |
| <a href="#">Table 12-6: Data Elements in the VARIABLES Node .....</a>                      | <a href="#">622</a> |
| <a href="#">Table 12-7: Data Elements in the TRIOSY Node .....</a>                         | <a href="#">623</a> |
| <a href="#">Table 12-8: Data Elements in the TECH_INFOS Node.....</a>                      | <a href="#">623</a> |

**Table 12-1. Data Elements in the Top Level Node and HIER\_BLOCK Nodes**

| Node Name         | Description                                                                                                                                                                                                                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name              | Name of the top design or a hierarchical block in the design. (string)                                                                                                                                                                                                                                                                    |
| PARTITION_HPATH   | Design path of the top level function or hierarchical block in the design. (string)                                                                                                                                                                                                                                                       |
| RESET_LENGTH      | Reset Cycle length.<br>1. For the top design holding hierarchical sub-blocks this value is the maximum of all sub-block RESET_LENGTH values. It represents a conservative approximation because it is the worst case value.<br>2. For an individual sub-block this value the actual number of cycles computed by the scheduler. (integer) |
| LATENCY           | Latency.<br>1. For the top design holding hierarchical sub-blocks this value is the sum of all sub-block LATENCY values.<br>2. For an individual sub-block this value the actual latency computed by the scheduler. (integer)                                                                                                             |
| THROUGHPUT_PERIOD | Throughput length.<br>1. For the top design holding hierarchical sub-blocks this value is the maximum of all sub-block THROUGHPUT_PERIOD values. It represents a conservative approximation because it is the worst case value.<br>2. For an individual sub-block this value the actual throughput computed by the scheduler. (integer)   |
| GANTT             | Name of the Gantt Structure in the Gantt Chart. (string)                                                                                                                                                                                                                                                                                  |
| NL_NAME           | Netlist name when the design was instantiated. (string)                                                                                                                                                                                                                                                                                   |
| RESOURCE_BLOCKS   | A hierarchical node containing a list of resource blocks corresponding to each resource (I/O, memory, pipe). Refer to Table 12-2.<br><br>Exception: Resource block named "__MGC_CTRL__" is a virtual block that holds together all the control signals (clk, a_rst, s_rst_, en, start, done) in its list of ports.                        |
| VARIABLES         | A hierarchical node containing a list of resource variables. The variable names in the list are netlist names generated when the resource block is instantiated. Refer to Table 12-6.                                                                                                                                                     |

**Table 12-1. Data Elements in the Top Level Node and HIER\_BLOCK Nodes**

| Node Name   | Description                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HIER_BLOCKS | A hierarchical node containing a list of the hierarchical blocks available in the design.                                                                           |
| TECH_INFOS  | A hierarchical node containing a list of technology libraries loaded for the design. The built-in libraries (STDOPS, mgc_ioprt) are not shown. Refer to Table 12-8. |

**Table 12-2. Data Elements in the RESOURCE\_BLOCKS Node**

| Node Name     | Description                                                                                                                                                                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name          | Name of the resource block assigned by Catapult. Exception: If the name is "__MGC_CTRL__" then it does not correspond to any actual resource. (string)                                                                                                                                                                  |
| HPATH         | Design path of the resource block. (string)                                                                                                                                                                                                                                                                             |
| RSCID         | Resource ID number. A unique resource ID for a given hierarchy. (integer)                                                                                                                                                                                                                                               |
| ORIG_RSC_ID   | Original RSCID number. Information available only for resource block(s) of the topmost FLOW_API (representing the top design), when an original resource is subjected to architectural constraints (such as BLOCK_SIZE, INTERLEAVE) that lead to splitting and addition of new resource blocks. (integer)               |
| ORIG_RSC_NAME | Name of the original parent resource. Information available only for resource block(s) of the topmost FLOW_API (representing the top design), when an original resource is subjected to architectural constraints (such as BLOCK_SIZE, INTERLEAVE) that lead to splitting and addition of new resource blocks. (string) |
| DATAMODE      | Direction of the resource (IN, OUT, INOUT).                                                                                                                                                                                                                                                                             |
| NL_NAME       | Netlist name when the resource block was instantiated. (string)                                                                                                                                                                                                                                                         |
| LIBRARY       | Name of the library containing the interface component. (string)                                                                                                                                                                                                                                                        |
| MODULE        | Name of module in the LIBRARY. (string)                                                                                                                                                                                                                                                                                 |
| QCOMP         | Qualified component name (has parameter values). (string)                                                                                                                                                                                                                                                               |
| PARAMETERS    | List of parameters used by QCOMP. A string composed of key/value pairs representing each parameter.                                                                                                                                                                                                                     |
| WIDTH         | Width of data. (integer)                                                                                                                                                                                                                                                                                                |
| GENERICS      | List of generic values used by QCOMP. A string composed of key/value pairs representing each generic.                                                                                                                                                                                                                   |
| FIRST         | First element number in block. (integer)                                                                                                                                                                                                                                                                                |

**Table 12-2. Data Elements in the RESOURCE\_BLOCKS Node**

| Node Name          | Description                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LAST               | Last element number in block. (integer)                                                                                                                                                                                                                                                                                                                             |
| STEP               | Step increment for element numbers in block. (integer)                                                                                                                                                                                                                                                                                                              |
| SIZE               | Number of elements in block. (integer)                                                                                                                                                                                                                                                                                                                              |
| STREAM             | Flag indicating variable is streamed in. (boolean)                                                                                                                                                                                                                                                                                                                  |
| MEMORY             | Flag indicating if the resource is a memory. (boolean)                                                                                                                                                                                                                                                                                                              |
| EXTERNAL           | Flag indicating if it is an external resource. (boolean)                                                                                                                                                                                                                                                                                                            |
| CLK_DRIVER         | Clock associated with this resource. It is useful only when there are multiple clocks and hierarchy, as each hierarchical part can directly access the primary I/O component, and in such cases, this field specifies the clock used by that hierarchical block. (string)                                                                                           |
| RESET_SYNC_DRIVER  | Synchronous reset associated with this resource. Similar to CLK_DRIVER in terms of meaning, corresponding to synchronous reset. (string)                                                                                                                                                                                                                            |
| RESET_ASYNC_DRIVER | Asynchronous reset associated with this resource. Similar to CLK_DRIVER in terms of meaning, corresponding to asynchronous reset. (string)                                                                                                                                                                                                                          |
| ENABLE_DRIVER      | Enable signal associated with this resource. Similar to CLK_DRIVER in terms of meaning, corresponding to enable. (string)                                                                                                                                                                                                                                           |
| PORts              | List of ports under the current resource block. Refer to Table 12-3.                                                                                                                                                                                                                                                                                                |
| BINDINGS           | List of bindings applicable to the current resource block. Refer to Table 12-4.                                                                                                                                                                                                                                                                                     |
| VARIABLES          | A string identifying the variable(s) mapped to this resource. The string is a list of items of the form "variable_name:LOC_INFO:field_name" separated by spaces, extracted out the /FLOW_API/VARIABLES list. LOC_INFO is used to disambiguate the variable inside /SCOPED_C_INTERFACE, as there can be multiple variables (in different scopes) with the same name. |

**Table 12-3. Data Elements in the PORTS Node**

| Node Name | Description                       |
|-----------|-----------------------------------|
| name      | Port name. (string)               |
| HPath     | Design path of the port. (string) |

**Table 12-3. Data Elements in the PORTS Node**

| <b>Node Name</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                         |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DEFAULT          | Default value of the port. NOTE: An empty string means no default value. Otherwise, the literal string is the default value. (string)                                                                                                                                                                                                                      |
| PHASENEG         | Flag indicating negative phase. TRUE, if negative phase. (boolean)                                                                                                                                                                                                                                                                                         |
| PORT_TYPE        | SystemC way of specifying the type of port. (string)                                                                                                                                                                                                                                                                                                       |
| DATAMODE         | Direction of port. (IN, OUT, INOUT)                                                                                                                                                                                                                                                                                                                        |
| WIDTH            | Width of port. (integer)                                                                                                                                                                                                                                                                                                                                   |
| IS_SCALAR        | Flag indicating scalar or bit-vector port. (boolean)                                                                                                                                                                                                                                                                                                       |
| IN_REG           | Flag indicating registered input port. (boolean)                                                                                                                                                                                                                                                                                                           |
| ASSOCS           | Pin Association. (SIGNAL:[EXTERNAL], CONSTANT:1, OPERATOR_PIN:D, OPERATION_PIN:I, SIGNAL:[CLOCK], SIGNAL:[A_RST], SIGNAL:[S_RST], SIGNAL:[ENABLE], SIGNAL:[GLOBAL]) (string)                                                                                                                                                                               |
| BINDINGS         | A string identifying all the applicable binding names separated by spaces. Each binding is prefixed with a number and a colon.                                                                                                                                                                                                                             |
| COUNT            | In the context of multiple R/W ports of memory ports, this field holds the number of ports represented by the given port. Default value = -1 (unset). (integer)                                                                                                                                                                                            |
| IS_CHILD_PORT    | True if the current port is not from Module of the library. This will be true in cases where there is sharing between multiple processes and/or when there is default value constraint on ports. This new port is automatically created by Catapult and represents a control signal. For example: *_sct represent select signal for 1-hot muxes. (boolean) |
| ORIG_PORT        | Name of the original port of Module in the library. Available only if IS_CHILD_PORT is true, and identifies the original port for which the current (child) port was created. (string)                                                                                                                                                                     |
| NL_NAME          | Netlist name for the port. NOTE: Only ports that are actually used in the design will have an NL_NAME field. (string)                                                                                                                                                                                                                                      |

**Table 12-4. Data Elements in the BINDINGS Node**

| Node Name  | Description                                                                                                                       |
|------------|-----------------------------------------------------------------------------------------------------------------------------------|
| name       | Name of the binding. Each binding name is prefixed by a number and a colon. For example, 1:read_port. (string)                    |
| PROPERTIES | List of selected properties from the library. Only the available properties are displayed by "solution get". Refer to Table 12-5. |

**Table 12-5. Data Elements in the PROPERTIES Node**

| Node Name | Description                                                    |
|-----------|----------------------------------------------------------------|
| name      | Name of the property. (string)                                 |
| type      | Type of the value, such as string, double, void, .... (string) |
| VALUE     | Value of the property. (string or double)                      |

**Table 12-6. Data Elements in the VARIABLES Node**

| Node Name  | Description                                                                                                             |
|------------|-------------------------------------------------------------------------------------------------------------------------|
| name       | Name of the variable as declared in the source code. (string)                                                           |
| LOC_INFO   | Cross-probe information. Can be used to identify the associated C++ variable inside /SCOPED_C_INTERFACE node. (integer) |
| HPATH      | Design path of the variable starting from the top design. (string)                                                      |
| CTYPE      | Original C++ data type of variable. (string)                                                                            |
| IS_SIGNED  | Flag indicating whether type is signed or unsigned. (boolean)                                                           |
| IS_CHANNEL | Flag indicating if variable is a channel. (boolean)                                                                     |
| WIDTH      | Width of variable. (integer)                                                                                            |
| SIZE       | Size of variable, if it is an array. (integer)                                                                          |
| DATAMODE   | Direction of variable. (IN, OUT, INOUT)                                                                                 |
| DIMENSIONS | Dimensions of the variable. {bitwidth dim1 dim2 ...}                                                                    |
| INDEX_1    | BASE_BIT directive. (integer)                                                                                           |
| INDEX_2    | BASE_BIT directive + width of variable. (integer)                                                                       |
| ADDR_1     | BASE_ADDR directive. (integer)                                                                                          |
| ADDR_2     | BASE_ADDR directive + size of variable. (integer)                                                                       |

**Table 12-6. Data Elements in the VARIABLES Node**

| <b>Node Name</b> | <b>Description</b>                                                                                                                                                                                                                                    |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TRIOSY           | List of transaction done signals. Refer to Table 12-7.                                                                                                                                                                                                |
| RESOURCE_BLOCKS  | A string that lists the names of the RESOURCE_BLOCKS corresponding to this variable. When a RESOURCE_BLOCK is split, a variable is spread across multiple RESOURCE_BLOCKS and this field identifies all those corresponding RESOURCE_BLOCKS. (string) |
| SIF_NAME         | Name of the variable assigned by Catapult. SIF_NAME can be same as the original name. (string)                                                                                                                                                        |

**Table 12-7. Data Elements in the TRIOSY Node**

| <b>Node Name</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                       |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name             | Transaction done signal name. (string)                                                                                                                                                                                                                                                                                                   |
| LOC_INFO         | Cross-probe information. Can be used to identify the associated C++ variable inside /SCOPED_C_INTERFACE node. (integer)                                                                                                                                                                                                                  |
| HPATH            | Design path of the variable starting from the top design. (string)                                                                                                                                                                                                                                                                       |
| NL_NAME          | Netlist name of the transaction I/O signal. (string)                                                                                                                                                                                                                                                                                     |
| RESOURCE_BLOCK   | A string that lists the names of the RESOURCE_BLOCKS corresponding to this transaction done signal. When a RESOURCE_BLOCK is split, each RESOURCE_BLOCK gets a transaction done signal for each associated variable and this field identifies the actual RESOURCE_BLOCK that is related to the current transaction done signal. (string) |

**Table 12-8. Data Elements in the TECH\_INFOS Node**

| <b>Node Name</b> | <b>Description</b>                                            |
|------------------|---------------------------------------------------------------|
| name             | Name of the technology library. (string)                      |
| SYN_TOOL         | Name of the synthesis tool, such as “Precision RTL”. (string) |
| SYN_FLOW         | Name of the flow, such as “Precision 2008a”. (string)         |
| FLOW_TYPE        | Value is always “Synthesis”. (string)                         |
| TECH_TYPE        | Either fpga or asic. (string)                                 |
| VENDOR           | Name of vendor. (string)                                      |
| MANUFACTURER     | Name of manufacturer. Relevant only in case of FPGA. (string) |

**Table 12-8. Data Elements in the TECH\_INFOS Node**

| <b>Node Name</b> | <b>Description</b>                                                        |
|------------------|---------------------------------------------------------------------------|
| FAMILY           | Name of technology family. Relevant only in case of FPGA.<br>(string)     |
| PART             | Device part name. Relevant only in case of FPGA. (string)                 |
| SPEED            | Device speed grade. Relevant only in case of FPGA. (string)               |
| IS_BASE_LIB      | TRUE, if the current library is the base technology library.<br>(boolean) |

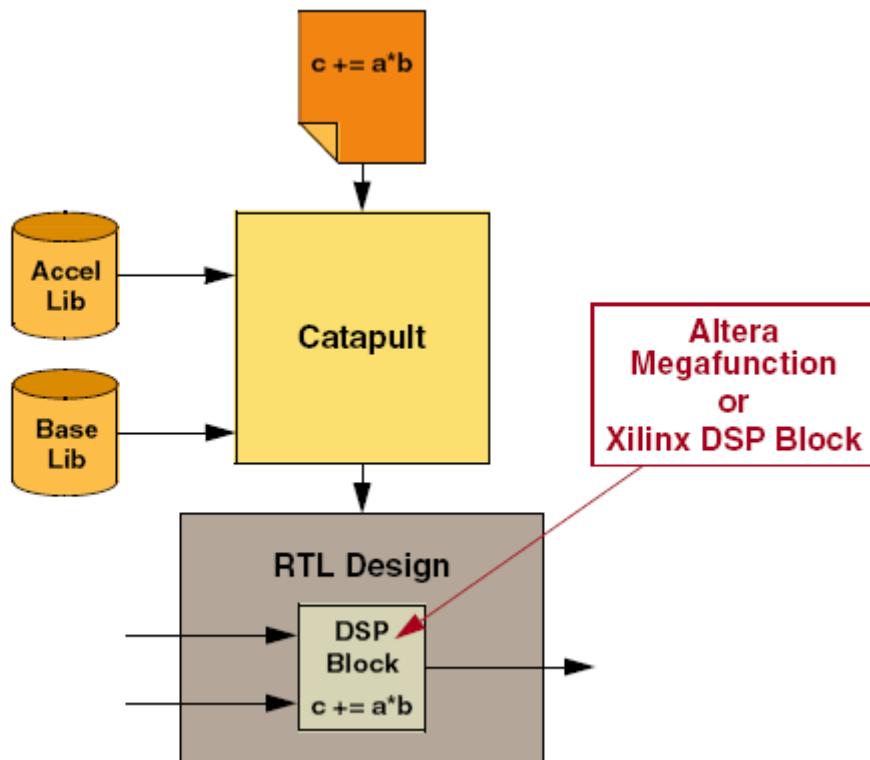
# Chapter 13

## Accelerated Component Libraries

You can use both Altera and Xilinx accelerated libraries to map directly to dedicated FPGA hardware resources using a library of C++ functions as shown in [Figure 13-1](#). Mapping C++ functions directly to hardware resources optimize the design area and performance. The C++ functions match the functionality of the hardware resources, such as the Altera Stratix II DSP and the Virtex-5 DSP blocks, and allows you to configure these hardware blocks via C++ function template parameters. The library supports the typical DSP structures such as multiply-add and multiply-accumulate as well as more exotic functions such as square-root, barrel shifters, and so on. The accelerated libraries are supported for both ac\_int and ac\_fixed data types. For more information, see the following topics:

- Using the Altera Accelerated Libraries . . . . .** **626**  
**Using the Xilinx Accelerated Libraries . . . . .** **637**

**Figure 13-1. Accelerated Library Flow**



## Using the Altera Accelerated Libraries

An example design using Altera accelerated libraries is available in the Accelerated Libraries toolkit. To access the Catapult Toolkits, select **Help > Toolkits...** from the Catapult session window.

**Table 13-1. Altera Accelerated Library Performance Benefits**

| Stratix-II-3               |                 |           |       |      |                    |           |       |       |               |           |       |
|----------------------------|-----------------|-----------|-------|------|--------------------|-----------|-------|-------|---------------|-----------|-------|
|                            | Custom Operator |           |       |      | No Custom Operator |           |       |       | % Improvement |           |       |
| Design                     | Fma x (MH z)    | DSP 9-bit | LUT s | Regs | Fma x (MH z)       | DSP 9-bit | LUT s | Regs  | Fmax          | Area LUTs | Regs  |
| <b>32-tap Parallel FIR</b> | 410             | 64        | 2500  | 3500 | 300                | 64        | 3000  | 7000  | 36.67         | 10.67     | 50.00 |
| <b>Edge Detector</b>       | 161             | 72        | 3723  | 2698 | 128                | 57        | 4903  | 5535  | 25.78         | 24.07     | 51.26 |
| <b>2-D DCT</b>             | 197             | 16        | 881   | 525  | 138                | 16        | 796   | 605   | 42.75         | -10.86    | 13.22 |
| <b>Alt_4Mutl _add</b>      | 390             | 16        | 290   | 84   | 330                | 16        | 514   | 1033  | 8.18          | 43.58     | 91.87 |
| <b>FFT</b>                 | 164             | 192       | 5550  | 4667 | 156                | 192       | 9638  | 13307 | 5.13          | 42.42     | 64.93 |

## Altera Custom Libraries and Header Files

The Altera Accelerated libraries are included as part of the Catapult installation. These Catapult libraries are included in the Catapult library search path by default. All Altera libraries, header files, and VHDL and Verilog files can be found in the MGC\_HOME tree at:

```
$MGC_HOME/pkgs/ccs_altera
```

## Use Model for Altera

To use the Altera Accelerated libraries, you must modify your source code to include the header file for the accelerated library and replace some of the algorithmic code with calls to the required function in the accelerated library. These code changes are straightforward and can yield significant performance and area improvements. The following FIR Filter example illustrates how the accelerated libraries can be used.

The FIR Filter design in [Example 13-1](#) illustrates how the accelerated libraries are used. The design implements a 32-tap filter. If the loops are completely unrolled there is 32 multipliers that feed an adder tree. Depending on the clock frequency, these multipliers and adders can be

scheduled over multiple clock cycles. For example, if the design is targeted at a Stratix-II, you can take advantage of the mult-add-add capabilities of the DSP block to map the FIR multipliers and part of the adder tree directly into these blocks.

### Example 13-1. Basic Design to be Modified to Use Accelerated Libraries

```
#include <ac_int.h>
#include "fir_filter.h"
#include "stdio.h"
#pragma design top
void fir_filter (ac_int<16> *input, ac_int<16> coeffs[NUM_TAPS] ,
ac_int<40> *output) {
 static ac_int<16> regs[NUM_TAPS];
 ac_int<40> temp = 0;
 int i;
 SHIFT:for (i = NUM_TAPS-1; i>=0; i--) {
 if (i == 0)
 regs[i] = *input;
 else
 regs[i] = regs[i-1];
 }
 MAC:for (i = 0; i<NUM_TAPS; i++) {
 temp += coeffs[i]*regs[i];
 }
 *output = temp;
}
```

Since the FIR filter in [Example 13-1](#) is 32-taps, we can use the Alt\_4Mult\_add function to map to the Stratix-II DSP block configured for 4 18x18 multiplies followed by two add stages. This function is part of the \$MGC\_HOME/shared/include/altera/accl.h header file.

```
#pragma map_to_operator "Alt_4mult_add"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned
int output_register, int width_a, bool sign_a, int width_b, bool
sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_4mult_add
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2,
 ac_int<width_a,sign_a> a3, ac_int<width_b,sign_b> b3
)
{ return ((a1*b1 + a0*b0) + (a3*b3 + a2*b2)); }
```

Where *#pragma map\_to\_operator* indicates that this function maps directly to the hardware.

Table 13-2 describes the template parameters:

**Table 13-2. Template Parameters for Altera Accelerated Libraries**

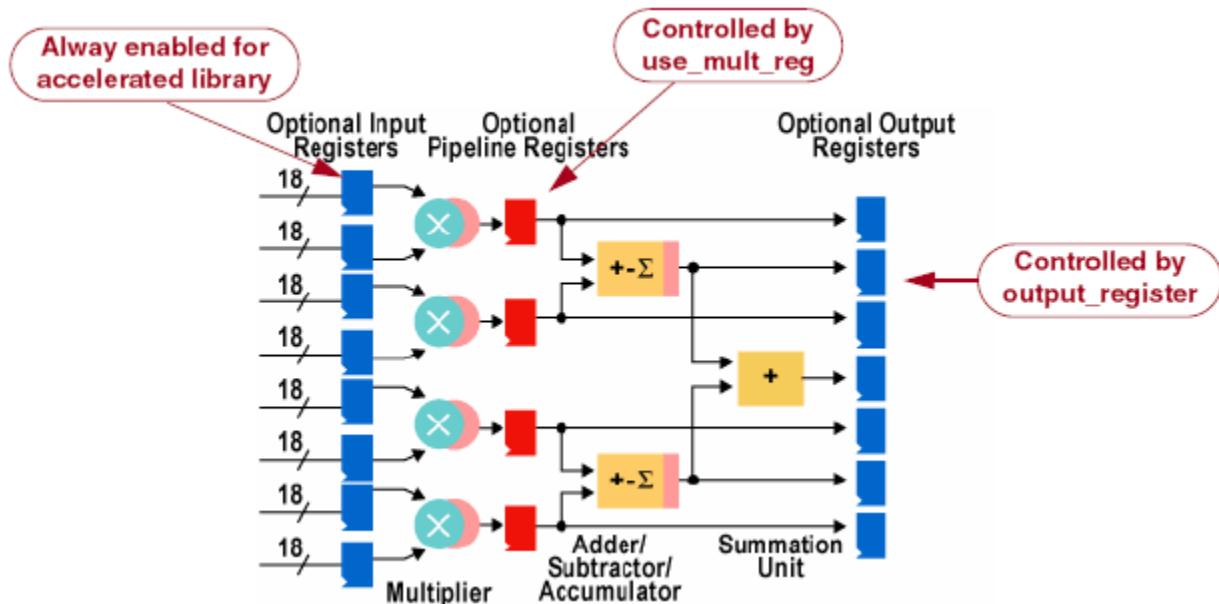
| Parameter                        | Description                                                                                             |
|----------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>use_extra_input_reg</code> | Adds an additional input register outside of the DSP block to help solve routing delay timing problems. |

**Table 13-2. Template Parameters for Altera Accelerated Libraries**

| Parameter              | Description                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------|
| <b>use_mult_reg</b>    | Registers the multiplier output. Valid values are 0 (no register) or 1 (1 register).                      |
| <b>output_register</b> | Sets the number of output registers. Valid values are 0 (no register), 1 (1 register) or 2 (2 registers). |
| <b>Width_a</b>         | Bit width of ‘a’ input. Bit widths of up to 18-bits wide are supported.                                   |
| <b>Sign_a</b>          | Sign of ‘a’ input.                                                                                        |
| <b>Width_b</b>         | Bit width of ‘b’ input. Bit widths of up to 18-bits wide are supported.                                   |
| <b>Sign_b</b>          | Sign of ‘b’ input.                                                                                        |

Figure 13-2 shows the Altera Stratix-II DSP block and how it can be configured based on the C++ function template parameters.

**Figure 13-2. Altera DSP Block Alt\_4mult\_add Structure**



The FIR code in Example 13-1 can be easily re-written to use the Altera accelerated library function. For example:

```
#include <ac_int.h>
#include "altera/accel.h" ← Include accelerated library
#include "fir_filter.h"
#include "stdio.h"
#pragma design top
void fir_filter (ac_int<16> *input, ac_int<16>
coeffs[NUM_TAPS], ac_int<40> *output) {
 static ac_int<16> regs[NUM_TAPS];
 ac_int<40> temp = 0;
 int i;
 SHIFT:for (i = NUM_TAPS-1; i>=0; i--) {
 if (i == 0)
 regs[i] = *input;
 else
 regs[i] = regs[i-1];
 }
 MAC:for (i = 0; i< NUM_TAPS; i+=4) {
 temp += Alt_4mult_add<1,1,1>(
 coeffs[i],regs[i],
 coeffs[i+1],regs[i+1],
 coeffs[i+2],regs[i+2],
 coeffs[i+3],regs[i+3]);
 }
 *output = temp;
}
```

For the FIR example shown above you can see that it is only necessary to specify the template parameters for the registers settings. The width and sign template parameters are automatically plugged in by template matching.

## Using or Compiling the Altera Simulation Libraries

Running SCVerify with the accelerated libraries requires that the Altera Megafunction simulation libraries have been compiled and mapped in Questa/Modelsim. Refer to “[Accelerated FPGA Devices and Verification](#)” on page 523 for instructions.

## Verifying the Design

To run SCVerify using the accelerated libraries, you must provide ModelSim with the location of the pre-compiled Altera Megafunction simulation libraries and set up the Modelsim flow to include this information in a modelsim.ini file.

Use the ModelSim Flow options to specify the default modelsim.ini file. Enter the path to the modelsim.ini file in the **optional dofile name to run on GUI startup** field. Relative paths are with respect to the Catapult project directory. Refer to “[ModelSim Flow Options](#)” on page 199 for more information. Alternatively, use the command:

```
options set Flows/ModelSim MSIM_DOFILE <path>/modelsim.ini
```

## Supported Operators and Functions

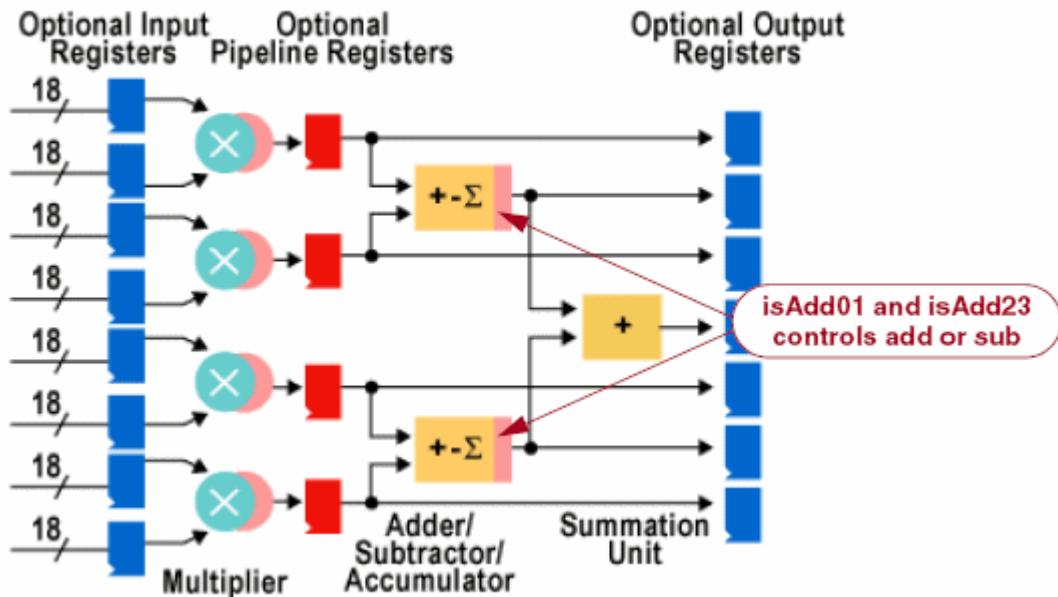
The library supports most of the possible configurations for the DSP blocks. It also includes support for square root, etc.. The supported functions are listed below. The ac\_int versions are listed for reference and there are equivalent ac\_fixed implementations. These can be found in \$MGC\_HOME/shared/include/altera/accel.h.

### Alt\_4mult\_addsub

Multiplies four sets of values and either adds or subtracts pairs of the four products. These add/sub results are then added together. The isAdd01 and isAdd23 inputs control the addition or subtraction of the products (0 == subtract, 1 == add). The output bit width must be the width of the "a" plus "b" inputs plus 2.

```
#pragma map_to_operator "Alt_4mult_addsub"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_4mult_addsub
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2,
 ac_int<width_a,sign_a> a3, ac_int<width_b,sign_b> b3,
 ac_int<1, false> isAdd01, ac_int<1, false> isAdd23
)
{
 return ((isAdd01 ? (a1*b1 + a0*b0) : (a0*b0 - a1*b1)) + (isAdd23 ?
(a3*b3 + a2*b2) : (a2*b2 - a3*b3)));
}
```

**Figure 13-3. Controlling Add/Sub for DSP Block Products**



## **Alt\_3mult\_addsub**

Multiplies three sets of values and either adds or subtracts pairs of the first two of the products. These add/sub results are then added together. The isAdd01 input controls the addition or subtraction of the products (0 == subtract, 1 == add). The output bit width must be the width of the "a" plus "b" inputs plus 2.

```
#pragma map_to_operator "Alt_3mult_addsub"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_3mult_addsub
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2,
 ac_int<1, false> isAdd01
)
{
 return ((isAdd01 ? (a1*b1 + a0*b0) : (a0*b0 - a1*b1)) + a2*b2);
}
```

## **Alt\_2mult\_addsub**

Multiplies two sets of values and either adds or subtracts the product. The isAdd01 input controls the addition or subtraction of the product (0 == subtract, 1 == add). The output bit width must be the width of the "a" plus "b" inputs plus 1.

```
#pragma map_to_operator "Alt_2mult_addsub"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+1, sign_a || sign_b> Alt_2mult_addsub
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<1, false> isAdd01
)
{
 return ((isAdd01 ? (a1*b1 + a0*b0) : (a0*b0 - a1*b1)));
}
```

## **Alt\_4mult\_add**

Multiplies four sets of values and adds the four products. The output bit width must be the width of the "a" plus "b" inputs plus 2.

```
#pragma map_to_operator "Alt_4mult_add"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register, int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_4mult_add
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2,
```

```
 ac_int<width_a,sign_a> a3, ac_int<width_b,sign_b> b3
)
{ return ((a1*b1 + a0*b0) + (a3*b3 + a2*b2)); }
```

## Alt\_3mult\_add

Multiplies three sets of values and adds the products. The output bit width must be the width of the "a" plus "b" inputs plus 2.

```
#pragma map_to_operator "Alt_3mult_add"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_3mult_add
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2
)
{ return ((a1*b1 + a0*b0) + a2*b2); }
```

## Alt\_2mult\_add

Multiplies two sets of values and adds the products. The output bit width must be the width of the "a" plus "b" inputs plus 1.

```
#pragma map_to_operator "Alt_2mult_add"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+1, sign_a || sign_b> Alt_2mult_add
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1
)
{ return (a1*b1 + a0*b0); }
```

## Alt\_4mult\_sub

Multiplies four sets of values and subtracts pairs of the four products. The output bit width must be the width of the "a" plus "b" inputs plus 2.

```
#pragma map_to_operator "Alt_4mult_sub"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_4mult_sub
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2,
 ac_int<width_a,sign_a> a3, ac_int<width_b,sign_b> b3
)
{ return ((a0*b0 - a1*b1) + (a2*b2 - a3*b3)); }
```

## Alt\_3mult\_sub

Multiplies three sets of values and subtracts pairs of the first two products. The output bit width must be the width of the "a" plus "b" inputs plus 2.

```
#pragma map_to_operator "Alt_3mult_sub"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_3mult_sub
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
 ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2
)
{ return ((a0*b0 - a1*b1) + a2*b2); }
```

## Alt\_2mult\_sub

Multiplies three sets of values and subtracts the products. The output bit width must be the width of the "a" plus "b" inputs plus 1.

```
#pragma map_to_operator "Alt_2mult_sub"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int
output_register,int width_a, bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+1, sign_a || sign_b> Alt_2mult_sub
(
 ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
 ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1
)
{ return (a0*b0 - a1*b1); }
Alt_abs
Returns the absolute value of the input
#pragma map_to_operator "Alt_abs"
template <int width_a, bool sign_a>
ac_int<width_a,sign_a> Alt_abs(
 ac_int<width_a,sign_a> a
)
{
 if (a < 0)
 {
 return -a;
 }
 return a;
}
```

## Alt\_sqrt

Returns the square root of the input.

```
#pragma map_to_operator "Alt_sqrt"
template <unsigned int output_register, int width_a>
ac_int<width_a,false> Alt_sqrt(
 ac_int<width_a,false> a
)
```

```
{
 ac_int<width_a, false> n1;
 sqrt(a, n1);
 return n1;
}
```

## Alt\_sqrt\_remainder

Returns the remainder of the square root of the input.

```
#pragma map_to_operator "Alt_sqrt_remainder"
template <unsigned int output_register, int width_a, bool sign_a>
ac_int<width_a,sign_a> Alt_sqrt_remainder(
 ac_int<width_a,sign_a> a
)
{
 ac_int<width_a,sign_a> n = 1;
 ac_int<width_a,sign_a> n1 = (n + a / n) >> 1;

 ac_int<width_a,sign_a> diff = Alt_abs<width_a,sign_a>(n1 - n);
 while(diff > 1) {
 n = n1;
 n1 = (n + a/n) >> 1;
 diff = Alt_abs<width_a,sign_a>(n1 - n);
 }
 while((n1*n1) > a) {
 n1 -= 1;
 }
 ac_int<width_a,sign_a> remain = a - n1*n1;
 return remain;
}
```

## Alt\_barrel\_shift

Dynamically controllable directional pipelined barrel shifter.

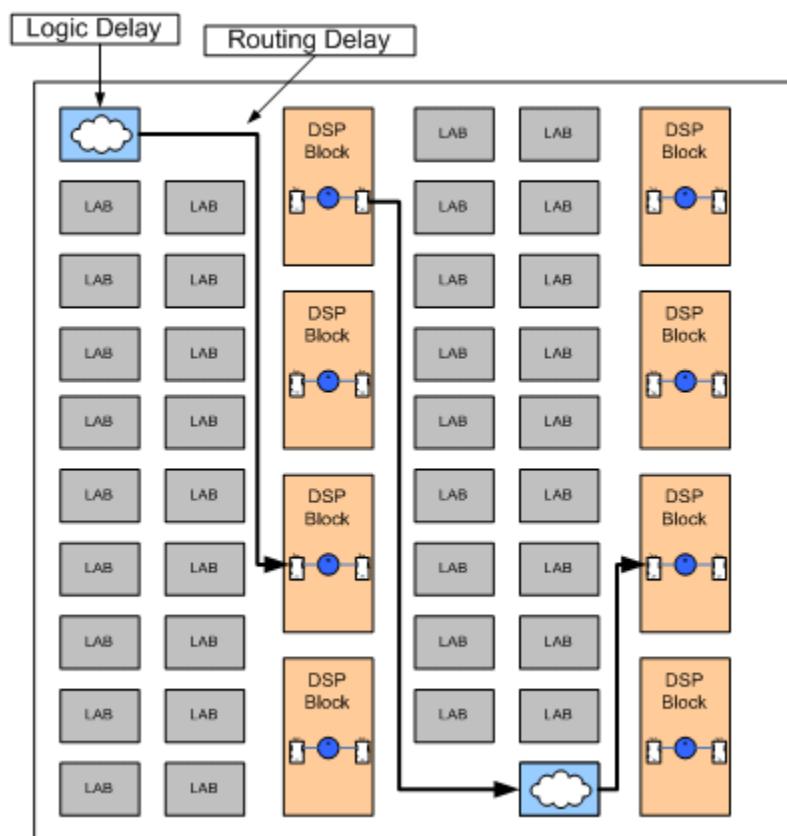
```
#pragma map_to_operator "Alt_barrel_shift"
template <unsigned int pipeline, int width_a, bool sign_a, int width_b,
bool sign_b>
ac_int<width_a,sign_a> Alt_barrel_shift(
 ac_int<width_a,sign_a> a,
 ac_int<width_b,sign_b> b,
 ac_int<1, false> dir
)
{
 ac_int<width_a,sign_a> c;
 if (dir)
 {
 c = a << b;
 }
 else
 {
 c = a >> b;
 }
 return c;
```

}

## Using Extra Input and Output Registers

In addition to mapping directly to the Altera DSP blocks and other hardware features, the Accelerated libraries allow you to solve tricky back-end timing problems due to routing delays and congestion. One of the most common sources for failing timing is due to routing to/from the DSP block. This is true even if the DSP block is fully pipelined with both input and output registers. There are cases where a LAB output may need to be routed to a DSP block that is placed on the opposite side of the chip (Figure 13-4). The required routing delay may cause the timing constraint to be violated.

**Figure 13-4. StratixII High-level Architecture**



The Accelerated Library functions have the ability to add an additional register to the inputs and the outputs of the hardware mapped operators. These extra registers solve the routing delay problems by providing P&R with the ability to balance the routing delay (Figure 13-5).

**Figure 13-5. Using Extra Input and Output Registers to Reduce Routing Delays**



Using the FIR filter in [Example 13-1](#) on page 627, we can set the `input_register` and `output_register` template parameters to achieve the desired results.

```
Setting to 1 adds the extra input register
for (i = 0; i < NUM_TAPS; i+=4) {
 temp += Alt_4mult_addr<1,1,2>(
 coeffs[i],regs[i],
 coeffs[i+1],regs[i+1],
 coeffs[i+2],regs[i+2],
 coeffs[i+3],regs[i+3]);
}

Setting to 2 adds an output register on the DSP block and an extra register outside of the DSP block
```

## Using the Xilinx Accelerated Libraries

An example design using Xilinx accelerated libraries is available in the “Accelerated Libraries” toolkit. To access the Catapult Toolkits, choose the “Help > Toolkits...” pulldown menu in the Catapult session window.

**Table 13-3. Xilinx Accelerated Library Performance Benefits**

| Design                              | Non-Accelerated | Accelerated |
|-------------------------------------|-----------------|-------------|
| <b>MAC-based FIR</b>                | 240             | 392         |
| <b>2-D DCT</b>                      | 229             | 395         |
| <b>4096-pt Complex FFT</b>          | 350             | 400         |
| <b>Rake Filter + Baseband Mixer</b> | 160             | 350         |

## Adding the Xilinx Custom Libraries and Header Files

The Xilinx Accelerated libraries are included as part of the Catapult installation. These Catapult libraries are included in the Catapult library search path by default. All Xilinx header files, libraries, and VHDL and Verilog files can be found in the install tree at:

```
$MGC_HOME/shared/include/xilinx
$MGC_HOME/pkgs/ccs_xilinx
```

## Use Model for Xilinx

To use the Xilinx Accelerated libraries, you must modify your source code to include the header file for the accelerated library and replace some of the algorithmic code with calls to the required function in the accelerated library. These code changes are very straightforward and can lead to huge performance and area improvements. The following FIR Filter example illustrates how the accelerated libraries can be used. The FIR filter example shown below implements a 32-tap filter. Depending on the clock frequency the accumulator can be scheduled over multiple clock cycles, limiting the amount of pipelining you can use.

```
#include <ac_int.h>
#include "fir_filter.h"
#include "stdio.h"
#pragma design top
void fir_filter (ac_int<16> *input, ac_int<16> coeffs[NUM_TAPS] ,
ac_int<40> *output) {
 static ac_int<16> regs[NUM_TAPS];
 ac_int<40> temp = 0;
 int i;
 SHIFT:for (i = NUM_TAPS-1; i>=0; i--) {
 if (i == 0)
 regs[i] = *input;
 else
```

```
 regs[i] = regs[i-1];
}
MAC:for (i = 0; i<NUM_TAPS; i++) {
 temp += coeffs[i]*regs[i];
}
*output = temp;
}
```

If we are planning on leaving the MAC loop rolled we can use the mac25x18 which will allow both the multiply and accumulate to be mapped to a single DSP48E. This function is part of the \$MGC\_HOME/shared/include/Xilinx/accel.h header file.

```
#pragma map_to_operator "macc25x18"
template<int opid, bool input_register, bool mult_register, bool
output_register>
ac_int<48> macc25x18(ac_int<25> A, ac_int<18> B, bool CLR, bool ADDSUB) {
 static ac_int<48> acc;

 if(CLR)
 acc = 0;
 if(ADDSUB)
 acc += A * B;
 else
 acc-= A * B;

 return acc;
}
```

Where "#pragma map\_to\_operator" indicates that this function will map directly to the hardware.

Table 13-4 describes the template parameters:

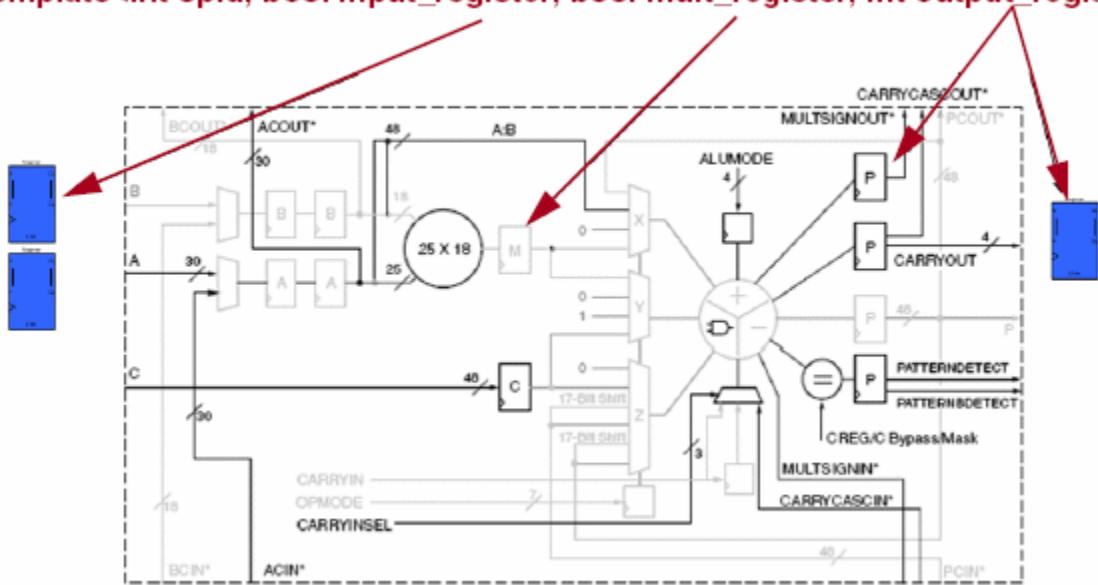
**Table 13-4. Template Parameters for Xilinx Accelerated Libraries**

| Parameter              | Description                                                                                                                                                                                                                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>opid</b>            | This template parameter allows you to specify unique instances of the C++ template function.                                                                                                                                                                                                           |
| <b>input_register</b>  | Adds an additional input register outside of the DSP block to help solve routing delay timing problems.                                                                                                                                                                                                |
| <b>mult_register</b>   | Registers the multiplier output. Valid values are 0 (no register) or 1 (1 register).                                                                                                                                                                                                                   |
| <b>output_register</b> | Sets the number of output registers. For the macc25x18 this allows an extra register to be added external to the DSP48E to help with routing delays. For the other non-accumulate functions this can be 0,1, or 2 registers where 1 indicates that the internal output register of the DSP48E is used. |

Figure 13-6 shows the Xilinx Virtex-5 DSP block and how it can be configured based on the C++ function template parameters.

**Figure 13-6. Xilinx DSP48E Block**

template<int opid, bool input\_register, bool mult\_register, int output\_register>



The original FIR code can be easily re-written to use the Xilinx Accelerated Library function.  
For example:

```
#include <ac_int.h>
#include "xilinx/accel.h"
#include "fir_filter.h"
#include "stdio.h"
#pragma design top
void fir_filter (ac_int<18> *input, ac_int<25>
coeffs[NUM_TAPS], ac_int<48> *output) {
 static ac_int<18> regs[NUM_TAPS];
 ac_int<48> temp = 0;
 int i;
 bool clr;
 SHIFT:for (i = NUM_TAPS-1; i>=0; i--) {
 if (i == 0)
 regs[i] = *input;
 else
 regs[i] = regs[i-1];
 }
 MAC:for (i = 0; i< NUM_TAPS; i+=1) {
 clr = (i==0)?1:0;
 temp =
macc25x18<0,true,true,true>(coeffs[i],regs[i],clr,true);
 }
 *output = temp;
}
```

Include accelerated library

## Using or Compiling the Xilinx Simulation Libraries

Running SCVerify with the accelerated libraries requires that the Xilinx simulation libraries are compiled and mapped in Questa/Modelsim. Refer to “[Accelerated FPGA Devices and Verification](#)” on page 523 for instructions.

## Supported Operators/Functions

The library supports many of the possible configurations for the DSP48E blocks. The supported functions are listed below. The ac\_int versions are listed for reference and there are equivalent ac\_fixed implementations. These can be found in \$MGC\_HOME/shared/include/Xilinx/accel.h.

### 25x18 Multiply-accumulate

```
#pragma map_to_operator "macc25x18"
template<int opid, bool input_register, bool mult_register, bool output_register>
ac_int<48> macc25x18(ac_int<25> A, ac_int<18> B, bool CLR, bool ADDSUB) {
 static ac_int<48> acc;

 if(CLR)
 acc = 0;
 if(ADDSUB)
 acc += A * B;
 else
 acc -= A * B;

 return acc;
}
```

### 25x18 Multiply-add (integer only)

```
#pragma map_to_operator "multadd25x18"
template<bool input_register, bool mult_register, int output_register>
ac_int<48> multadd25x18(ac_int<25> A, ac_int<18> B, ac_int<48> C, bool ADDSUB) {

 if(ADDSUB)
 return C + A * B;
 else
 return C - A * B;
}
```

### 25x18 Complex multiply-accumulate

```
template <int opid, bool input_register, bool output_register>
void cmacc25x18(
 ac_int<25> A_re,
 ac_int<25> A_im,
 ac_int<18> B_re,
```

```

 ac_int<18> B_im,
 bool CLR,
 ac_int<48> &P_re,
 ac_int<48> &P_im
)
{
 ac_int<50,false> A;
 ac_int<36,false> B;
 ac_int<96,false> P;

 A.set_slc (0, A_re);
 A.set_slc (25, A_im);

 B.set_slc (0, B_re);
 B.set_slc (18, B_im);
 P = cmacc25x18_int<opid,input_register,output_register>(A,B,CLR);

 P_re = P.template slc<48>(0);
 P_im = P.template slc<48>(48);
}

```

## 25x18 Complex Multiply

```

template <bool input_register, bool output_register>
void cmult25x18
(
 ac_int<25> A_re,
 ac_int<25> A_im,
 ac_int<18> B_re,
 ac_int<18> B_im,
 ac_int<43> &P_re,
 ac_int<43> &P_im
)
{
 ac_int<50,false> A;
 ac_int<36,false> B;
 ac_int<86,false> P;

 A.set_slc (0, A_re);
 A.set_slc (25, A_im);

 B.set_slc (0, B_re);
 B.set_slc (18, B_im);
 P = cmult25x18_int<input_register,output_register>(A,B);

 P_re = P.template slc<43>(0);
 P_im = P.template slc<43>(43);
}

```

## 48-bit Adder

```

template<bool input_register, int output_register>
ac_int<48> add48(ac_int<48> AB, ac_int<48> C){

 ac_int<30,false> A;
 ac_int<18,false> B;

```

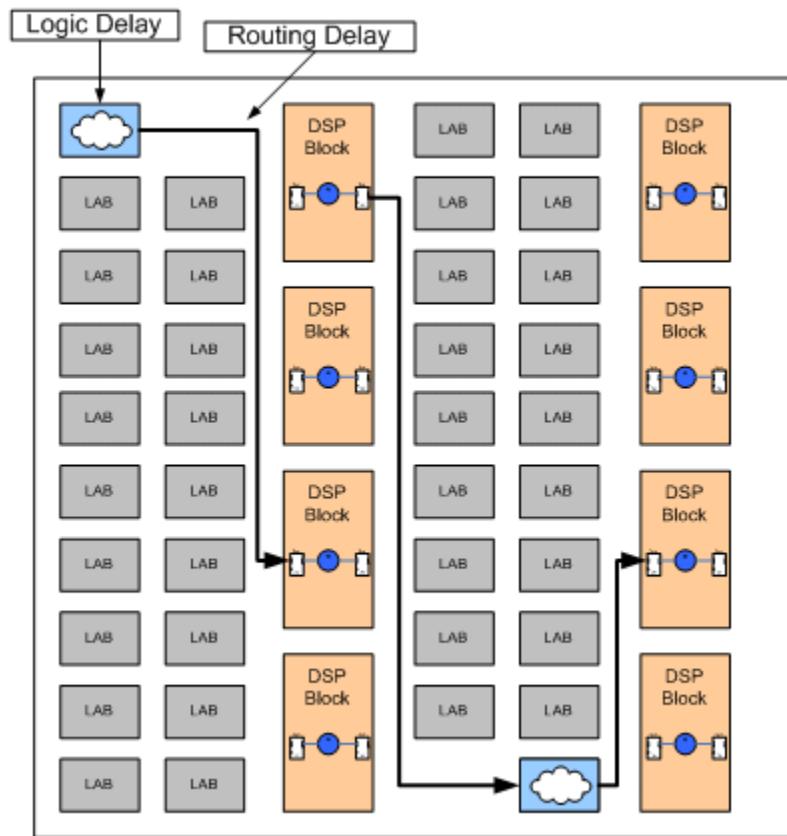
```
A = AB.template slc<30>(18);
B = AB.template slc<18>(0);

return add48_int<input_register,output_register>(A,B,C);
}
```

## Using Extra Input and Output Registers

In addition to mapping directly to the Xilinx DSP48E blocks and other hardware features, the Accelerated libraries allow you to solve tricky back-end timing problems due to routing delays and congestion. One of the most common sources for failing timing is due to routing to/from the DSP block. This is true even if the DSP block is fully pipelined with both input and output registers. There are cases where a slice output may need to be routed to a DSP block that is placed on the opposite side of the chip (Figure 13-7). The required routing delay may cause the timing constraint to be violated.

**Figure 13-7. Virtex-5 High-level Architecture**



The Accelerated Library functions have the ability to add an additional register to the inputs and the outputs of the hardware mapped operators. These extra registers solve the routing delay problems by providing P&R with the ability to balance the routing delay (Figure 13-8).

Figure 13-8. Using Extra Input and Output Registers to Reduce Routing Delays





# Appendix A

## Catapult Library Components

---

The following topics describe the Catapult library component parameters:

- [Base Components](#)
- [I/O Components](#)
- [Memory Components](#)
- [Channel Components](#)
- [Altera Accelerated Components](#)
- [Xilinx Accelerated Components](#)

For more information, see the component definitions in the HDL files. Copies of the HDL files are saved in the Solution directory after a design netlist is generated.

## Base Components

[Table A-1](#) describes the parameters for the base library components.

**Table A-1. Base Library Component Parameters**

| Component(s)                    | Parameters                                                                                                                                                                                                                                                                                                                    |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mgc_abs<br>mgc_equal<br>mgc_not | <b>width</b> — Bit width of single input<br><b>dataset</b> — Characterization data set for ASIC components                                                                                                                                                                                                                    |
| mgc_add                         | <b>width</b> — Bit width of single input<br><b>signd_a</b> — Signed input on ‘a’: 0=unsigned, 1=signed<br><b>width_b</b> — Bit width of input ‘b’<br><b>signd_b</b> — Signed input on ‘b’: 0=unsigned, 1=signed<br><b>width_z</b> — Bit width of output ‘z’<br><b>dataset</b> — Characterization data set for ASIC components |

**Table A-1. Base Library Component Parameters (cont.)**

| Component(s)                                                    | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mgc_add_pipe                                                    | <b>width_a</b> — Bit width of input ‘a’<br><b>signd_a</b> — Signed input on ‘a’: 0=unsigned, 1=signed<br><b>width_b</b> — Bit width of input ‘b’<br><b>signd_b</b> — Signed input on ‘b’: 0=unsigned, 1=signed<br><b>width_z</b> — Bit width of output ‘z’<br><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Enable signal polarity: 0=active low, 1=active high<br><b>ph_a_rst</b> — Asynchronous reset polarity: 0=active low, 1=active high<br><b>ph_s_rst</b> — Synchronous reset polarity: 0=active low, 1=active high<br><b>n_outreg</b> — Number of output registers: 0 to 1<br><b>stages</b> — Number of pipeline stages: 2 to 4<br><b>a_rst_used</b> — Asynchronous reset is used: 0=false, 1=true<br><b>s_rst_used</b> — Synchronous reset is used: 0=false, 1=true<br><b>en_used</b> — Enable signal is used: 0=false, 1=true<br><b>dataset</b> — Characterization data set for ASIC components |
| mgc_nand<br>mgc_nor<br>mgc_or<br>mgc_xnor<br>mgc_xor<br>mgc_rem | <b>width</b> — Bit width of single input<br><b>ninps</b> — Number of input data signals<br><b>dataset</b> — Characterization data set for ASIC components                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| mgc_div<br>mgc_mod                                              | <b>width_a</b> — Bit width of input ‘a’<br><b>width_b</b> — Bit width of input ‘b’<br><b>signd</b> — Signed data: 0=unsigned, 1=signed<br><b>dataset</b> — Characterization data set for ASIC components                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| mgc_mul                                                         | <b>width_a</b> — Bit width of input ‘a’<br><b>signd_a</b> — Signed input on ‘a’: 0=unsigned, 1=signed<br><b>width_b</b> — Bit width of input ‘b’<br><b>signd_b</b> — Signed input on ‘b’: 0=unsigned, 1=signed<br><b>width_z</b> — Bit width of output ‘z’<br><b>dataset</b> — Characterization data set for ASIC components                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| mgc_mul_pipe                                                    | <b>width_a</b> — Bit width of input ‘a’<br><b>signd_a</b> — Signed input on ‘a’: 0=unsigned, 1=signed<br><b>width_b</b> — Bit width of input ‘b’<br><b>signd_b</b> — Signed input on ‘b’: 0=unsigned, 1=signed<br><b>width_z</b> — Bit width of output ‘z’<br><b>clock_edge</b> — Active clock edge: 0=low, 1=high<br><b>enable_active</b> — Enable signal polarity: 0=active low, 1=active high<br><b>a_rst_active</b> — Asynchronous reset polarity: 0=active low, 1=active high<br><b>s_rst_active</b> — Synchronous reset polarity: 0=active low, 1=active high<br><b>stages</b> — Number of pipeline stages: 2 to 4<br><b>n_inreg</b> — Number of input registers: 0 to 2<br><b>dataset</b> — Characterization data set for ASIC components                                                                                                                                                                                     |

**Table A-1. Base Library Component Parameters (cont.)**

| <b>Component(s)</b>                                        | <b>Parameters</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mgc_mux                                                    | <b>width</b> — Bit width of single input<br><b>ctrlw</b> — Number of select lines<br><b>p2ctrlw</b> — Two to the power of the number of select lines<br><b>dataset</b> — Characterization data set for ASIC components                                                                                                                                                                                                                                                                                                                                          |
| mgc_mux1hot                                                | <b>width</b> — Bit width of single input<br><b>ctrlw</b> — Number of select lines<br><b>dataset</b> — Identifies the characterization data set for ASIC components                                                                                                                                                                                                                                                                                                                                                                                              |
| mgc_reg_neg<br>mgc_reg_pos                                 | <b>width</b> — Bit width of single input<br><b>has_a_rst</b> — Asynchronous reset is used: 0=false, 1=true<br><b>a_rst_on</b> — Asynchronous reset polarity: 0=active low, 1=active high<br><b>has_s_rst</b> — Synchronous reset is used: 0=false, 1=true<br><b>s_rst_on</b> — Synchronous reset polarity: 0=active low, 1=active high<br><b>has_enable</b> — Enable signal is used: 0=false, 1=true<br><b>enable_on</b> — Enable signal polarity: 0=active low, 1=active high<br><b>dataset</b> — Identifies the characterization data set for ASIC components |
| mgc_rot                                                    | <b>width</b> — Bit width of single input<br><b>width_s</b> — Bit width of input ‘s’<br><b>signd_s</b> — Signed data: 0=unsigned, 1=signed<br><b>sleft</b> — Defines rotation: 0=right shift, 1=left shift<br><b>log2w</b> — Log2 of width<br><b>l2wp2</b> — Two to the power of Log2 of width<br><b>dataset</b> — Identifies the characterization data set for ASIC components                                                                                                                                                                                  |
| mgc_shift_bl<br>mgc_shift_br<br>mgc_shift_l<br>mgc_shift_r | <b>width_a</b> — Bit width of input ‘a’<br><b>signd_a</b> — Signed input on ‘a’: 0=unsigned, 1=signed<br><b>width_s</b> — Bit width of input ‘s’<br><b>width_z</b> — Bit width of output ‘z’<br><b>dataset</b> — Identifies the characterization data set for ASIC components                                                                                                                                                                                                                                                                                   |

## I/O Components

Table A-2 describes the parameters for the I/O library components.

**Table A-2. I/O Library Component Parameters**

| <b>Component(s)</b> | <b>Parameters</b>                                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------------------------|
| mgc_chan_in         | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port<br><b>sz_width</b> — Bit width of the “sizez” port |

**Table A-2. I/O Library Component Parameters**

| Component(s)                                                                                                                                                                         | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mgc_in_wire<br>mgc_in_wire_en<br>mgc_in_wire_wait<br>mgc inout_stdreg_en<br>mgc inout_stdreg_wait<br>mgc_out_prereg_en<br>mgc_out_stdreg<br>mgc_out_stdreg_en<br>mgc_out_stdreg_wait | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port                                                                                                                                                                                                                                                                                                                                                                                                                 |
| mgc inout_buf_wait<br>mgc_out_buf_wait                                                                                                                                               | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port<br><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Enable signal polarity: 0=active low, 1=active high<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>pwropt</b> — Clock gating: 0=disabled, 1=normal, 2=high                                                                                      |
| mgc inout_fifo_wait<br>mgc_out_fifo_wait                                                                                                                                             | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port<br><b>fifo_sz</b> — Bit size of FIFO<br><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Enable signal polarity: 0=active low, 1=active high<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>ph_log2</b> — Log2 of FIFO size (fifo_sz)<br><b>pwropt</b> — Clock gating: 0=disabled, 1=normal, 2=high |
| mgc io_sync<br>mgc_sync                                                                                                                                                              | None                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| mgc_out_reg                                                                                                                                                                          | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port<br><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Enable signal polarity: 0=active low, 1=active high<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high                                                                                                                                                    |
| mgc_sync_cdc                                                                                                                                                                         | <b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Active enable signal polarity: 0=low, 1=high<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high                                                                                                                                                                                                                                    |

# Memory Components

Table A-3 describes the parameters for the I/O library components.

**Table A-3. Memory Library Component Parameters**

| Component                                 | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dualport                                  | <b>ram_id</b> — Unique resource ID<br><b>words</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>addr_width</b> — Number of address bits<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>enable_active</b> — Enable signal active polarity: 0=low, 1=high<br><b>re_active</b> — Read enable signal polarity: 0=active low, 1=active high<br><b>we_active</b> — Write enable signal polarity: 0=active low, 1=active high<br><b>num_byte_enables</b> — Number of byte enables for byte read/write<br><b>clock_edge</b> — Active clock edge: 0=falling, 1=rising<br><b>num_input_registers</b> — Number of input registers<br><b>num_output_registers</b> — Number of output registers<br><b>no_of_dualport_readwrite_port</b> — Number of dualport read/write ports |
| mgc_rom                                   | <b>rom_id</b> — Unique resource ID<br><b>size</b> — Number of words<br><b>width</b> — Bit width of each word                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| mgc_rom_sync_regin<br>mgc_rom_sync_regout | <b>rom_id</b> — Unique resource ID<br><b>size</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>ph_clk</b> — Active clock edge: 0=falling edge, 1=rising edge<br><b>ph_en</b> — Enable signal polarity: 0=active low, 1=active high<br><b>ph_a_rst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_s_rst</b> — Synchronous reset polarity: 0=active low, 1=active high<br><b>num_input_registers</b> — Number of input registers<br><b>num_output_registers</b> — Number of output registers<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>en_used</b> — Enable signal is used: 0=false, 1=true                                                                                                                                                        |

**Table A-3. Memory Library Component Parameters**

| Component      | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RAM            | <b>ram_id</b> — Unique resource ID<br><b>words</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>addr_width</b> — Number of address bits<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>enable_active</b> — Enable signal active polarity: 0=low, 1=high<br><b>re_active</b> — Read enable signal polarity: 0=active low, 1=active high<br><b>we_active</b> — Write enable signal polarity: 0=active low, 1=active high<br><b>num_byte_enables</b> — Number of byte enables for byte read/write<br><b>clock_edge</b> — Active clock edge: 0=falling, 1=rising<br><b>no_of_RAM_readwrite_port</b> — Number of RAM read/write ports                                                 |
| RAM_dualRW     | <b>ram_id</b> — Unique resource ID<br><b>words</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>addr_width</b> — Number of address bits<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>enable_active</b> — Enable signal active polarity: 0=low, 1=high<br><b>re_active</b> — Read enable signal polarity: 0=active low, 1=active high<br><b>we_active</b> — Write enable signal polarity: 0=active low, 1=active high<br><b>num_byte_enables</b> — Number of byte enables for byte read/write<br><b>clock_edge</b> — Active clock edge: 0=falling, 1=rising<br><b>no_of_RAM_dualRW_readwrite_port</b> — Number of RAM dual read/write ports                                     |
| RAM_separateRW | <b>ram_id</b> — Unique resource ID<br><b>words</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>addr_width</b> — Number of address bits<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>enable_active</b> — Enable signal active polarity: 0=low, 1=high<br><b>re_active</b> — Read enable signal polarity: 0=active low, 1=active high<br><b>we_active</b> — Write enable signal polarity: 0=active low, 1=active high<br><b>num_byte_enables</b> — Number of byte enables for byte read/write<br><b>clock_edge</b> — Active clock edge: 0=falling, 1=rising<br><b>no_of_RAM_read_port</b> — Number of RAM read ports<br><b>no_of_RAM_write_port</b> — Number of RAM write ports |

**Table A-3. Memory Library Component Parameters**

| Component                                    | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| REGISTER_FILE                                | <b>ram_id</b> — Unique resource ID<br><b>words</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>addr_width</b> — Number of address bits<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>enable_active</b> — Enable signal active polarity: 0=low, 1=high<br><b>re_active</b> — Read enable signal polarity: 0=active low, 1=active high<br><b>we_active</b> — Write enable signal polarity: 0=active low, 1=active high<br><b>num_byte_enables</b> — Number of byte enables for byte read/write<br><b>clock_edge</b> — Active clock edge: 0=falling, 1=rising<br><b>no_of_REGISTER_FILE_read_port</b> — Number of REGISTER_FILE read ports<br><b>no_of_REGISTER_FILE_write_port</b> — Number of REGISTER_FILE write ports                                             |
| singleport<br>singleport_rf<br>singleport_wf | <b>ram_id</b> — Unique resource ID<br><b>words</b> — Number of words<br><b>width</b> — Bit width of each word<br><b>addr_width</b> — Number of address bits<br><b>a_reset_active</b> — Active async reset polarity: 0=low, 1=high<br><b>s_reset_active</b> — Active sync reset polarity: 0=low, 1=high<br><b>enable_active</b> — Enable signal active polarity: 0=low, 1=high<br><b>re_active</b> — Read enable signal polarity: 0=active low, 1=active high<br><b>we_active</b> — Write enable signal polarity: 0=active low, 1=active high<br><b>num_byte_enables</b> — Number of byte enables for byte read/write<br><b>clock_edge</b> — Active clock edge: 0=falling, 1=rising<br><b>num_input_registers</b> — Number of input registers<br><b>num_output_registers</b> — Number of output registers<br><b>no_of_singleport_readwrite_port</b> — Number of singleport read/write ports |

# Channel Components

Table A-4 describes the parameters for the channel library components.

**Table A-4. Channel Library Component Parameters**

| Component    | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mgc_pipe     | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port<br><b>fifo_sz</b> — Bit size of FIFO<br><b>log2_sz</b> — Log2 of FIFO size (fifo_sz)<br><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Enable signal active polarity: 0=low, 1=high<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>pwopt</b> — Clock gating: 0=disabled, 1=normal, 2=high                                                                                                                                                                                                                                                                                                                                                                                     |
| mgc_pipe_mem | <b>rscid</b> — Unique resource ID<br><b>width</b> — Bit width of port<br><b>fifo_sz</b> — Bit size of FIFO<br><b>log2_sz</b> — Log2 of FIFO size (fifo_sz)<br><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising<br><b>ph_en</b> — Enable signal active polarity: 0=low, 1=high<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>input_register</b> — Number of input registers: 0 to 1<br><b>mem_read_delay</b> — Number of cycles from read to output: Constant value 1<br><b>mem_write_through</b> — For dual read/write ports: 0=read-first, 1=write-first<br><b>pipe_impl</b> — Underlying implementation type: Constant value 0<br><b>memory_impl</b> — Underlying implementation type: Constant value 0<br><b>pwopt</b> — Clock gating: 0=disabled, 1=normal, 2=high |

**Table A-4. Channel Library Component Parameters**

| Component            | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mgc_pipe_mem_cdc     | <p><b>rscid</b> — Unique resource ID</p> <p><b>width</b> — Bit width of port</p> <p><b>fifo_sz</b> — Bit size of FIFO</p> <p><b>log2_sz</b> — Log2 of FIFO size (fifo_sz)</p> <p><b>ph_clk_dout</b> — Active clock edge: 0=falling, 1=rising</p> <p><b>ph_en_dout</b> — Enable signal active polarity: 0=low, 1=high</p> <p><b>ph_arst_dout</b> — Active async reset polarity: 0=low, 1=high</p> <p><b>ph_srst_dout</b> — Active sync reset polarity: 0=low, 1=high</p> <p><b>ph_clk_din</b> — Active clock edge: 0=falling, 1=rising</p> <p><b>ph_en_din</b> — Enable signal active polarity: 0=low, 1=high</p> <p><b>ph_arst_din</b> — Active async reset polarity: 0=low, 1=high</p> <p><b>ph_srst_din</b> — Active sync reset polarity: 0=low, 1=high</p> <p><b>input_register</b> — Number of input registers: 0 to 1</p> <p><b>mem_read_delay</b> — Number of cycles from read to output: Constant value 1</p> <p><b>mem_write_through</b> — For dual read/write ports: 0=read-first, 1=write-first</p> <p><b>pipe_impl</b> — Underlying implementation type: Constant value 0</p> <p><b>memory_impl</b> — Underlying implementation type: Constant value 0</p> <p><b>pwopt</b> — Clock gating: 0=disabled, 1=normal, 2=high</p> |
| mgc_pipe_regfile     | <p><b>rscid</b> — Unique resource ID</p> <p><b>width</b> — Bit width of port</p> <p><b>fifo_sz</b> — Bit size of FIFO</p> <p><b>log2_sz</b> — Log2 of FIFO size (fifo_sz)</p> <p><b>input_register</b> — Number of input registers: 0 to 1</p> <p><b>ph_clk</b> — Active clock edge: 0=falling, 1=rising</p> <p><b>ph_en</b> — Enable signal active polarity: 0=low, 1=high</p> <p><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high</p> <p><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high</p> <p><b>pwopt</b> — Clock gating: 0=disabled, 1=normal, 2=high</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| mgc_pipe_regfile_cdc | <p><b>rscid</b> — Unique resource ID</p> <p><b>width</b> — Bit width of port</p> <p><b>fifo_sz</b> — Bit size of FIFO</p> <p><b>log2_sz</b> — Log2 of FIFO size (fifo_sz)</p> <p><b>input_register</b> — Number of input registers: 0 to 1</p> <p><b>ph_clk_dout</b> — Active clock edge: 0=falling, 1=rising</p> <p><b>ph_en_dout</b> — Enable signal active polarity: 0=low, 1=high</p> <p><b>ph_arst_dout</b> — Active async reset polarity: 0=low, 1=high</p> <p><b>ph_srst_dout</b> — Active sync reset polarity: 0=low, 1=high</p> <p><b>ph_clk_din</b> — Active clock edge: 0=falling, 1=rising</p> <p><b>ph_en_din</b> — Enable signal active polarity: 0=low, 1=high</p> <p><b>ph_arst_din</b> — Active async reset polarity: 0=low, 1=high</p> <p><b>ph_srst_din</b> — Active sync reset polarity: 0=low, 1=high</p> <p><b>pwopt</b> — Clock gating: 0=disabled, 1=normal, 2=high</p>                                                                                                                                                                                                                                                                                                                                        |

## Altera Accelerated Components

Table A-5 describes the parameters for Altera accelerated components.

**Table A-5. Altera Accelerated Component Parameters**

| Component                                                                                                                                                      | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Alt_1mult_add_accum_siii<br>Alt_1mult_addsub_accum<br>Alt_1mult_sub_accum<br>Alt_1mult_sub_accum_siii                                                          | <b>resc_id</b> — Unique resource ID<br><b>WIDTH_A</b> — Bit width of input ‘a’<br><b>WIDTH_B</b> — Bit width of input ‘b’<br><b>SIGNA</b> — Signed input of ‘a’: 0=unsigned, 1=signed<br><b>SIGNB</b> — Signed input of ‘b’: 0=unsigned, 1=signed<br><b>use_extra_input_reg</b> — Input register outside DSP block: 0=no register, 1=register<br><b>use_mult_reg</b> — Multiplier output register: 0=no register, 1=register<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high                                                                                                        |
| Alt_2mult_add<br>Alt_2mult_addsub<br>Alt_2mult_sub<br>Alt_3mult_add<br>Alt_3mult_addsub<br>Alt_3mult_sub<br>Alt_4mult_add<br>Alt_4mult_addsub<br>Alt_4mult_sub | <b>WIDTH_A</b> — Bit width of input ‘a’<br><b>WIDTH_B</b> — Bit width of input ‘b’<br><b>SIGNA</b> — Signed input of ‘a’: 0=unsigned, 1=signed<br><b>SIGNB</b> — Signed input of ‘b’: 0=unsigned, 1=signed<br><b>use_extra_input_reg</b> — Input register outside DSP block: 0=no register, 1=register<br><b>output_register</b> — Number of output registers: 0=no register, 1=one register, 2=two registers<br><b>use_mult_reg</b> — Multiplier output register: 0=no register, 1=register<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high                                        |
| Alt_2mult_add_accum<br>Alt_2mult_sub_accum<br>Alt_3mult_add_accum<br>Alt_3mult_sub_accum<br>Alt_4mult_add_accum<br>Alt_4mult_sub_accum                         | <b>resc_id</b> — Unique resource ID<br><b>WIDTH_A</b> — Bit width of input ‘a’<br><b>WIDTH_B</b> — Bit width of input ‘b’<br><b>SIGNA</b> — Signed input of ‘a’: 0=unsigned, 1=signed<br><b>SIGNB</b> — Signed input of ‘b’: 0=unsigned, 1=signed<br><b>use_extra_input_reg</b> — Input register outside DSP block: 0=no register, 1=register<br><b>output_register</b> — Number of output registers: 0=no register, 1=one register, 2=two registers<br><b>use_mult_reg</b> — Multiplier output register: 0=no register, 1=register<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high |

**Table A-5. Altera Accelerated Component Parameters**

| Component                                   | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Alt_abs                                     | <b>WIDTH</b> — Bit width of single input<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high                                                                                                                                                                                                                                                                                             |
| Alt_barrel_shift                            | <b>WIDTH</b> — Bit width of single input<br><b>WIDTHDIST</b> — Bit width of the distance port<br><b>pipeline</b> — Number of pipeline stages<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high                                                                                                                                                                                         |
| Alt_divide_quotient<br>Alt_divide_remainder | <b>WIDTH_D</b> — Bit width of denominator<br><b>WIDTH_N</b> — Bit width of numerator<br><b>SIGN_D</b> — Signed input of denominator: 0=unsigned, 1=signed<br><b>SIGN_N</b> — Signed input of numerator: 0=unsigned, 1=signed<br><b>output_register</b> — Number of output registers: 0=no register, 1= one register, 2=two registers<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high |
| Alt_shift_taps                              | <b>resc_id</b> — Unique resource ID<br><b>TAP_DISTANCE</b> — Number of delay elements<br><b>WIDTH</b> — Bit width of single input<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high                                                                                                                                                                                                    |
| Alt_sqrt<br>Alt_sqrt_remainder              | <b>WIDTH</b> — Bit width of single input<br><b>output_register</b> — Number of output registers: 0=no register, 1= one register, 2=two registers<br><b>arst_polarity</b> — Async reset active polarity: 0=low, 1=high<br><b>srst_polarity</b> — Sync reset active polarity: 0=low, 1=high<br><b>ena_polarity</b> — Enable signal active polarity: 0=low, 1=high                                                                                                                                                                                     |

## Xilinx Accelerated Components

Table A-6 describes the parameters for Xilinx accelerated components.

**Table A-6. Xilinx Accelerated Component Parameters**

| Component           | Parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| add48<br>cmult25x18 | <b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>ph_en</b> — Enable signal active polarity: 0= low, 1=high<br><b>input_register</b> — Input register outside the DSP48E block: 0=no register, 1=register<br><b>output_register</b> — Number of output registers: 0=no output register, 1=use output register that is inside the DSP48E, 2=use external output register                                                                                   |
| cmacc25x18          | <b>opid</b> — A unique instance of the C++ template function<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>ph_en</b> — Enable signal active polarity: 0= low, 1=high<br><b>input_register</b> — Input register outside the DSP48E block: 0=no register, 1=register<br><b>output_register</b> — Number of output registers: 0=no output register, 1=use output register that is inside the DSP48E, 2=use external output register                   |
| macc25x18           | <b>opid</b> — A unique instance of the C++ template function<br><b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>ph_en</b> — Enable signal active polarity: 0= low, 1=high<br><b>input_register</b> — Input register outside the DSP48E block: 0=no register, 1=register<br><b>mult_register</b> — Multiplier output register: 0=no register, 1=register<br><b>output_register</b> — Output register outside the DSP48E block: 0=no register, 1=register |
| multadd25x18        | <b>ph_arst</b> — Active async reset polarity: 0=low, 1=high<br><b>ph_srst</b> — Active sync reset polarity: 0=low, 1=high<br><b>ph_en</b> — Enable signal active polarity: 0= low, 1=high<br><b>input_register</b> — Input register outside the DSP48E block: 0=no register, 1=register<br><b>mult_register</b> — Multiplier output register: 0=no register, 1=register<br><b>output_register</b> — Number of output registers: 0=no output register, 1=use output register that is inside the DSP48E, 2=use external output register   |

# Appendix B

## GUI Reference

---

This appendix provides details on options available from the following GUI elements in Catapult:

- Project windows
- Taskbar
- Popup menus
- Dialog boxes

Most of the GUI option descriptions can also accessed from the Help button in the respective GUI element.

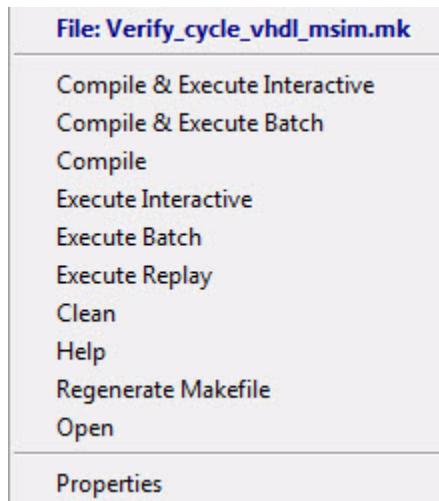
## SCVerify Makefile Popup Menu

To access: Right-click on any makefile.

### Description

Use this popup menu to perform optional makefile actions and display makefile information.

**Figure B-1. Makefile Popup Menu**



**Table B-1. Makefile Popup Menu Options**

| Option                                       | Description                                                                                                                                                           |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compile and Execute Interactive<br>(Default) | Compiles the design and testbench for the target simulation environment, invokes the simulator in interactive/GUI mode, and simulates the design.                     |
| Compile and Execute Batch                    | Compiles the design and testbench for the target simulation environment, invokes the simulator in batch/command line mode, and simulates the design.                  |
| Compile                                      | Recompiles the target design and testbench for the target simulation environment.                                                                                     |
| Execute Interactive                          | Invokes the target simulator in interactive/GUI mode and loads the design but does not launch simulation.                                                             |
| Execute Batch                                | Invokes the target simulator in batch/command line mode, loads the design, and launches simulation. The simulation results display in the Catapult transcript window. |

**Table B-1. Makefile Popup Menu Options**

| Option              | Description                                                                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Execute Replay      | Runs the Replay test part of the Trace/Replay verification flow. For more information, see “ <a href="#">Non-Blocking Reads and Verification (C/C++)</a> ” on page 543. |
| Clean               | Deletes all files generated by prior compilations.                                                                                                                      |
| Help                | Lists the set of valid makefile targets.                                                                                                                                |
| Regenerate Makefile | Regenerates the testbench wrapper files (C/C++) and makefiles.                                                                                                          |
| Open                | Displays the contents of the makefile in the Catapult text editor.                                                                                                      |
| Properties          | Displays information for the makefile including: location, application file type, options, and description.                                                             |

## Related Topics

[Design Verification](#)

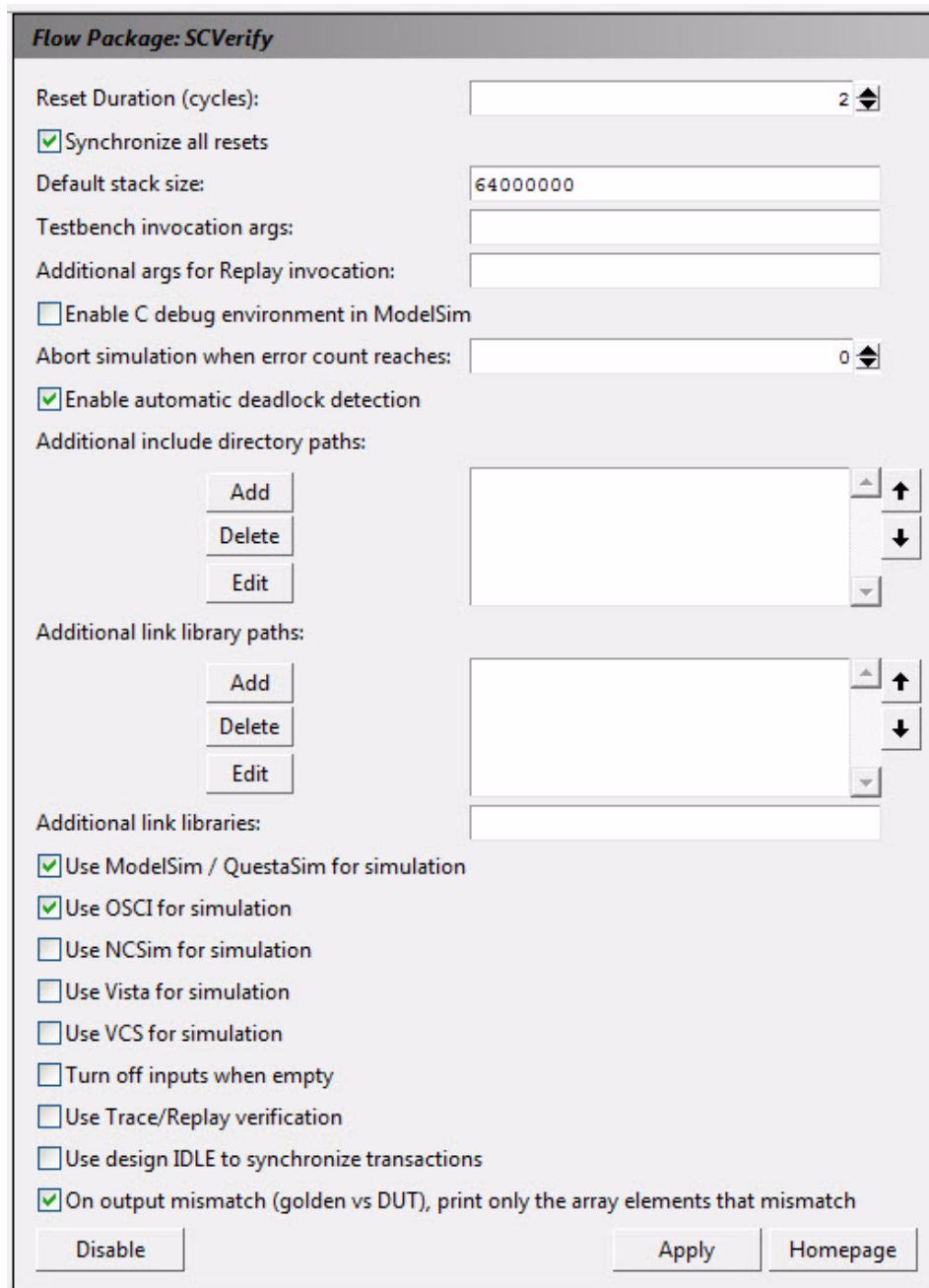
## Flow Package: SCVerify Dialog Box

To access: Select **View > Other Windows > Flow Manager**. Choose SCVerify, and click Enable.

### Description

Use this dialog to configure optional setting for SCVerify.

**Figure B-2. Flow Package: SCVerify Dialog Box**



**Table B-2. Flow Package: SCVerify Options**

| <b>Option</b>                              | <b>Description</b>                                                                                                                                                                                                                                                 | <b>Value</b>   |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| Reset Duration (cycles):                   | Specifies the number of clock cycles the reset signal is asserted. Default is 2.                                                                                                                                                                                   | floating point |
| Synchronize all resets                     | Overrides the individual reset durations of multi-clock designs and forces all resets to be inactive at the same time.<br>The inactive time period is determined by the slowest-clock-period. Default enabled.<br><br>(SL version only)                            | N/A            |
| Default stack size:                        | Specifies the default stack size for SystemC modules. Designs with many large data arrays may require a larger stack. Default is 64MB.<br><br>(SL version only)                                                                                                    | N/A            |
| Testbench invocation args:                 | Specifies command-line arguments passed to the testbench. For example: <code>int argc, char *argv[]</code> parameters of <code>main</code><br><br>Arguments are passed through ModelSim and available as local data in <code>testbench::main()</code>              | N/A            |
| Additional args for Replay invocation:     | Specifies input arguments for the <code>main</code> function in custom testbenches. Specified arguments affect all Catapult generated files.                                                                                                                       | N/A            |
| Enable C debug environment in ModelSim     | Invokes ModelSim in debug mode, sets an initial breakpoint in the reset function, and prompts you to specify a set of variables to report on.                                                                                                                      | N/A            |
| Abort simulation when error count reaches: | Specifies the number of simulation errors allowed before the simulation is terminated. When the specified number is reached, simulation is terminated and an error message displays in the transcript. Default is 0 and simulation continues regardless of errors. | Integer        |
| Enable automatic deadlock detection:       | Terminates simulation if a deadlock condition is detected and reports an error message to the transcript.                                                                                                                                                          | N/A            |

**Table B-2. Flow Package: SCVerify Options**

| <b>Option</b>                                                                   | <b>Description</b>                                                                                                                                                                                       | <b>Value</b>                                |
|---------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| Additional include directory paths:                                             | Specifies one or more directory paths containing include files.                                                                                                                                          | Multiple paths must be space separated.     |
| Additional link library paths:                                                  | Specifies one or more directory paths to search for libraries during compilation.                                                                                                                        | Multiple paths must be space separated.     |
| Additional link libraries:                                                      | Specifies one or more libraries to use during compilation.                                                                                                                                               | Multiple libraries must be space separated. |
| Use ModelSim / QuestaSim for simulation                                         | Generates makefiles that drive the ModelSim/QuestaSim simulation tools.                                                                                                                                  | N/A                                         |
| Use OSCI for simulation                                                         | Generates makefiles that drive the OSCI simulation tool. Catapult must be set up to generate SystemC output.                                                                                             | N/A                                         |
| Use NCSim for simulation                                                        | Generates makefiles to drive the NCSim simulation tool. Catapult must be set up to generate cycle VHDL, RTL VHDL or RTL Verilog.<br><br>Precision RTL Synthesis, TLM and SystemC are not supported       | N/A                                         |
| Use Vista for simulation                                                        | Generates makefiles to drive the VistaTM simulation tool.                                                                                                                                                | N/A                                         |
| Use VCS for simulation                                                          | Generates makefiles to drive the VCS simulation tool.                                                                                                                                                    | N/A                                         |
| Turn off inputs when empty                                                      | Enables <i>pipeline flushing</i> for distributed pipelines. For more information, see “ <a href="#">Distributed Pipeline Synthesis</a> ” on page 245.                                                    | N/A                                         |
| Use Trace/Replay for verification                                               | Enables verification of designs that model concurrency with non-blocking ac_channel read methods. For more information, see “ <a href="#">Non-Blocking Reads and Verification (C/C++)</a> ” on page 543. | N/A                                         |
| Use design IDLE to synchronize transactions                                     | Enables verification of designs that use idle signals to synchronize transactions between blocks. For more information, see “ <a href="#">Idle Signal Insertion</a> ” on page 592.                       | N/A                                         |
| On output mismatch (golden vs DUT), print only the array elements that mismatch | Reports only mismatch elements to the transcript. By default, all elements are reported.                                                                                                                 | N/A                                         |

## Usage Notes

Verification makefiles must be regenerated if options are changed after RTL is generated.

## Related Topics

[SCVerify Flow](#)

[SCVerify Flow Troubleshooting](#)



# Glossary

---

## Constraint

A user-specified synthesis control expressed as a value or a set of values for a particular parameter, such as maximum area or delay.

## C-Steps

Roughly equivalent to one state in a state machine. It is only different if there are several conditional branches in the same control step.

## Data Path

The portion of the design that computes data, from primary inputs or from intermediate input. The data path is usually controlled by the control unit.

## FIFO

First-In, First-Out memory, also known as a queue.

## FSM

Acronym for Finite State Machine also known as state machine. The FSM is a mathematical abstraction sometimes used to design digital logic or computer programs. It is a behavior model composed of a finite number of states, transitions between those states, and actions, similar to a flow chart that illustrates the way logic runs when certain conditions are met.

## Functional Unit

An arithmetic and logical operator written explicitly in the C code. It doesn't include data path muxes, the control logic, or FSM.

## Gantt Chart

A Gantt chart shows the actual time required to perform each operation and the data dependencies between the operations.

## Handshaking

The introduction of additional signals to indicate when data should be passed between models. This allows models to independently process data and only synchronize when it is necessary to pass data back and forth.

## HDL

This is an abbreviation for Hardware Description Language, meaning the textual representation of a circuit, usually using VHDL or Verilog. It is also sometimes a generic term for the Verilog language.

## Initiation Interval

The initiation interval of a pipelined component or loop specifies the rate at which the component or loop can begin processing new input data. Most pipelined components have an initialization interval of 1, which means that the component can begin processing new input data every clock cycle.

## Latency

The time required to execute the hardware equivalent of the C function call.

## Loop Iterations

Loop iterations is the number of times the loop runs before it exits. Loop iterations is a constraint to tell Catapult that its estimate for the number of iterations in a loop is incorrect.

## Loop Pipelining

Loop pipelining is how often to start the next iteration of the loop. After loop unrolling and several other transformations are complete, the scheduler uses the pipelining constraints to build a pipelined loop. This happens much later than the other two transformations, and pipelining will not occur if the loop has been optimized away because of the first two constraints.

## Loop Unrolling

Loop unrolling is the number of times to copy the loop body. After the loop iterations have been determined, the loop is unrolled based on the unrolling constraints. This will change the number of iterations of the loop, which is reflected in the reporting, but it won't change the iterations directive.

## Ping-Pong Memory

In a ping-pong configuration, two memories are shared by two processes in order to allow continuous processing of data. The data output by process A is consumed by process B. The two memories (ping and pong) allows process A to write to the ping memory while process B reads from the pong memory. In the next cycle they switch and process A writes to pong while process B reads from ping.

## Pragma

A user-specified synthesis tool directive that is placed directly in the VHDL source as a comment. Another tool, such as a VHDL simulator, will analyze this as a comment and ignore it.

## Round-Robin Memory

In a round-robin memory configuration, multiple memories are shared by multiple processes in order to allow continuous processing of data.

## RTL

Register-Transfer Level, a circuit abstraction that includes all clock edges, storage elements, and transformations of values.

## RTL Synthesis

The process of transforming an RTL description into a gate-level, technology netlist.

### Shared Memory

Memory shared by two or more processes and requiring a controller to ensure that memory access is mutually exclusive among the processes.

### Scheduling

The assignment of data operations to clock cycles such that constraints and data flow relationships are met.

### Symbolic Analysis

An analysis technique that uses what is known about the design to find out more about what is known. It is used to find the data-ranges of variables.

### Verilog HDL

Verilog HDL is a hardware description language used to design and document electronic systems defined by the IEEE standard 1364-2001 Revision C. Verilog HDL allows design activity at various levels of abstraction.

### VHDL

VHSIC hardware description language, the textual representation language for circuits defined by IEEE standard 1076-1993.



# Index

-- Switch, 358

## — Symbols —

.c file, 169  
.ccs file, 46, 169  
.cpp file, 169  
.cxx file, 169  
.h file, 169  
.hh file, 169  
.log file, 46, 169  
.msim file, 170  
.nlv file, 171  
.psr file, 171  
.rpt file, 170  
.siff file, 171  
.tcl file, 46, 169

## — A —

Adder tree optimization  
    OPT\_CONST\_MULTS directive, 329  
Advanced IO Control for Memories, 271  
application exit command, 369  
application get command, 367  
application report command, 370  
Architectural constraints  
    see also Constraint editor, 104  
Architectural options, 184  
AREA\_GOAL directive, 308  
ARRAY\_SIZE directive, 307  
Arrays  
    map in physical memory, 251  
    mapped to ROMs, 259  
    Optimization, 248  
    packing modes, 116, 253  
    PACKING\_MODE directive, 330  
    splitting, 260  
    word width, 255  
    WORD\_WIDTH directive, 340

## — B —

BASE\_ADDR directive, 251, 310

BASE\_BIT directive, 251, 310

Basic components, 288  
    inout components, 291  
    Input components, 288  
    output components, 289  
    parameter descriptions, 645  
BLOCK\_SIZE directive, 117, 260, 262, 310

## — C —

C Code Analysis, 155  
Catapult  
    architectural options, 184  
    catapult.ini file, 223  
    catapult.log file, 169  
    compiler options, 188, 189  
    component libraries options, 180  
    constraints editor, 195  
    Design Compiler synthesis options, 204  
    directives.tcl file, 44  
    flow command tasks, 364  
    general options, 175  
    hardware interface options, 181  
    hardware options, 186  
    Initialization file, 223, 224  
    input options, 188  
    invoking the GUI, 59  
    log file (.log), 44, 46, 93  
    messages, 159  
    messages.txt file, 44  
    object naming algorithm, 49  
    output file options, 189  
    power estimation flow options, 203, 210, 218, 222  
    Precision RTL Synthesis, 201  
    project file (.ccs), 43, 46, 169  
    project initialization options, 178  
    QoF bar chart, 80  
    QoF report tables, 79  
    QoF XY Plot, 80  
    recreating solutions, 44

- RTL Compiler Synthesis, 210  
saving and restoring options, 223  
saving and restoring projects, 44  
schedule viewer options, 196  
schematic viewer options, 197  
session window features, 60  
setting default options, 173  
simulation options, 199, 207, 208, 209  
solution directories, 44  
solution message log file, 44  
startup directory, 175  
state model, 47  
task bar, 67  
Tcl command file, 46  
text editor, 165, 195  
version control options, 179  
Windows shortcut, 60  
work flow, 46, 364  
working directory, 45  
catapult command, 59  
catapult.ini  
    format of, 224  
        Saving and restoring, 223  
catapult.log file, 44, 46, 93, 169  
catapult.tcl file, 46  
CCORE  
    bottom-up flow, 602  
    CHARACTERIZE\_OPERATOR  
        directive, 311  
    CREATE\_COMBINATIONAL directive,  
        314  
    in Gantt chart, 603  
    input registers, 324  
    MAP\_TO\_MODULE directive, 325  
    overview, 597  
    registers, 330  
    top-down flow, 600  
-checkpath Switch, 357  
Clock uncertainty, 276  
Clocks, 273  
    CLOCK\_EDGE directive, 311  
    CLOCK\_NAME directive, 312  
    CLOCK\_OVERHEAD directive, 312  
    CLOCK\_PERIOD directive, 312  
    CLOCK\_UNCERTAINTY directive, 313  
CLOCKS directive, 313  
configuring, 99  
percent sharing allocation, 113, 312  
uncertainty, 276  
Command input window, 72  
Command Line Shell, 359  
Commands  
    application exit, 369  
    application get, 367  
    application report, 370  
    catapult, 59  
    Command Summary Table, 360  
    cycle add, 375  
    cycle command, 374  
    cycle find\_op, 379  
    cycle get, 380  
    cycle remove, 383  
    cycle set, 385  
    directive, 388  
    directive get, 389  
    directive remove, 392  
    directive set, 394, 471  
    dofile, 396  
    flow control commands, 47, 423  
    flow get, 398  
    flow package forget, 401  
    flow package names, 402  
    flow package option add, 403  
    flow package option get, 405  
    flow package option remove, 406  
    flow package option set, 407  
    flow package present, 408  
    flow package provide, 410  
    flow package require, 412  
    flow package script, 414  
    flow package vcompare, 415  
    flow package versions, 417  
    flow package vsatisfies, 418  
    flow provide, 420  
    flow run, 422  
    general command syntax, 349  
    go command, 47, 423  
    help, 424  
    ignore\_memory\_precedences, 429  
    logfile close, 432

- logfile isopen, 433  
logfile message, 434  
logfile move, 439  
logfile name, 440  
logfile open, 441  
logfile save\_commands, 442  
options, 443  
options defaults, 444  
options exists, 446  
options get, 447  
options load, 450  
options save, 452  
options set, 455  
project, 457  
project close, 458  
project get, 459  
project load, 461  
project newproject commands  
    new, 462  
project report, 463  
project save, 465  
project set, 466  
quit, 468  
resource, 469  
resource add, 470  
resource remove, 473  
set\_working\_dir, 475  
solution file add, 482  
solution file remove, 487  
solution file restore, 488  
solution file set, 489  
solution netlist, 494, 605  
solution new, 492  
solution remove, 498  
solution rename, 499  
solution report, 500  
solution restore, 504  
solution select, 505  
solution timing, 506  
view file, 512  
view schedule, 509  
view schematic, 510  
view source, 513  
COMPGRADE directive, 314  
Compiler
- setting options, 188  
view settings, 189  
Component  
    automatic generation, 603  
    CREATE\_COMBINATIONAL directive, 314, 604  
    custom CCORE, 597  
    custom library, 604  
    grade, 314  
    libraries, 180  
    parameter descriptions, 645  
Constraint editor  
    architectural constraints, 104  
    on design, 106  
    on loops, 120  
    on processes, 112  
    on resource variables, 118  
    on resources, 114  
    block diagram view, 124  
    hardware interface, 98  
    hardware technology, 97  
    options, 195  
    set top-level funciton, 100  
    window features, 69  
Critical path in schematic, 143  
Cross-probing, 102  
    from Precision RTL, 103  
    tool bar buttons, 66  
Cycle  
    CSTEPS\_FROM directive, 314  
    cycle-accurate output files, 142, 170  
    flow, 248  
    MAX\_LATENCY directive, 326  
    MAXLEN directive, 326  
    MIN\_CSTEPS\_FROM directive, 328  
    reports, 141  
cycle commands, 374  
    cycle add, 375  
    cycle find\_op, 379  
    cycle get, 380  
    cycle remove, 383  
    cycle set, 385

— D —

- Data channels  
    stage replication, 116

- DECOUPLING\_STAGES directive, 316  
DEFAULT\_VALUE\_OPT directive, 316  
Deleting Solutions, 164  
Dependency chains, 232  
Design analysis, 100  
  checklist, 104  
Design flow  
  controlling, 47  
  state model, 47  
Design hierarchy  
  pragma hls\_design, 342  
Design object naming, 49  
Design optimization  
  general design constraints, 273  
  I/O access, 229  
  loop iterations, 229  
  loop merging, 234  
  loop pipeline flushing, 245  
  loop pipelining overview, 239  
  loop unrolling, 231  
  loops and timing, 226  
  loops overview, 225  
  memories and arrays overview, 248  
  overview, 225  
DirectInput resource type, 270  
Directive  
  MAP\_TO\_MODULE, 115  
directive command, 388  
directive get command, 389  
directive remove command, 392  
directive set command, 394, 471  
Directives  
  Alphabetic List, 300  
  Architectural Constraints, 306  
  AREA\_GOAL, 113, 308  
  ARRAY\_SIZE, 307  
  ASSIGN\_OVERHEAD, 110, 309  
  BASE\_ADDR, 251, 310  
  BASE\_BIT, 251, 310  
  BLOCK\_SIZE, 117, 260, 262, 310  
  CHARACTERIZE\_OPERATOR, 311  
  CLOCK\_EDGE, 311  
  CLOCK\_NAME, 312  
  CLOCK\_OVERHEAD, 113, 312  
  CLOCK\_PERIOD, 312  
  CLOCK\_UNCERTAINTY, 313  
  CLOCKS, 313  
  COMPGRADE, 114, 314  
  CREATE\_COMBINATIONA, 108  
  CREATE\_COMBINATIONAL, 314, 604  
  CSTEPS\_FROM, 314  
  DECOUPLING\_STAGES, 316  
  DEFAULT\_VALUE\_OPT, 316  
  DESIGN\_GOAL, 112, 317  
  DISTRIBUTED\_PIPELINING, 317  
  DONE\_FLAG, 317  
  EFFORT\_LEVEL, 109, 112, 318  
  ENABLE\_ACTIVE, 318  
  ENABLE\_NAME, 318  
  EXTERNAL\_MEMORY, 319  
  FSM\_ENCODING, 111, 114, 187, 319  
  GATE EFFORT, 319  
  GATE\_EXPAND\_MIN\_WIDTH, 320  
  GATE\_MIN\_WIDTH, 320  
  GATE\_REGISTERS, 109, 113, 321  
  GEN\_EXTERNAL\_ENABLE, 111, 321  
  IDLE, 110, 113  
  IDLE\_SIGNAL, 321  
  IGNORE\_DEPENDENCY\_FROM, 322  
  IGNORE\_PROCESS, 323  
  INPUT\_DELAY, 109, 117, 296, 324  
  INPUT\_REGISTERS, 324  
  INTERLEAVE, 117, 260, 261, 325  
  ITERATIONS, 325  
  MAP\_TO\_MODULE, 325  
  MAX\_LATENCY, 113, 326  
  MAXLEN, 114, 326  
  MEM\_MAP\_THRESHOLD, 259, 327  
  MERGEABLE, 238, 327  
  MIN\_CSTEPS\_FROM, 328  
  NO\_X\_ASSIGNMENTS, 328  
  OLD\_SCHED, 111, 329  
  OPT\_CONST\_MULTS, 97, 181, 329  
  OUTPUT\_DELAY, 109, 117, 296, 330  
  OUTPUT\_REGISTERS, 330  
  PACKING\_MODE, 253, 330  
  PIPELINE\_INIT\_INTERVAL, 331  
  PIPELINE\_RAMP\_UP, 331  
  Pre-defined directives, 307  
  PRESERVE\_STRUCTS, 331

- READY\_FLAG, 332  
REDUNDANT\_MUX\_OPT, 332  
REG\_MAX\_FANOUT, 187, 279, 333  
REGISTER\_IDLE\_SIGNAL, 333  
REGISTER\_THRESHOLD, 185, 259, 334  
RESET\_ASYNC\_ACTIVE, 335  
RESET\_ASYNC\_NAME, 335  
RESET\_CLEAR\_ALL\_REGS, 111, 336  
RESET\_KIND, 336  
RESET\_SYNC\_ACTIVE, 336  
RESET\_SYNC\_NAME, 337  
SAFE\_FSM, 109, 113, 187, 338  
setting  
    from the GUI, 305  
    how to, 299  
SPECULATE, 110, 113, 185, 338  
STAGE\_IO\_CNS, 338  
STAGE\_REPLICATION, 116, 339  
START\_FLAG, 339  
STREAM, 339  
TECHLIBS, 340  
TRANSACTION\_DONE\_SIGNAL, 340  
UNROLL, 340  
WORD\_WIDTH, 251, 256, 340  
directives.tcl file, 44  
DISTRIBUTED\_PIPELINING directive, 317  
dofile command, 396  
Done flag  
    directive, 317  
TRANSACTION\_DONE\_SIGNAL  
    directive, 340
- E —
- EFFORT\_LEVEL directive, 318  
Enable signal  
    configuring, 99  
ENABLE\_ACTIVE directive, 318  
ENABLE\_NAME directive, 318  
    for external memories, 111
- F —
- File extensions  
    .c, 169  
    .ccs, 44, 169  
    .cpp, 169  
    .css, 46  
    .cxx, 169  
    .h, 169  
    .hh, 169  
    .log, 46, 169  
    .msim, 170  
    .nlv, 171  
    .psr, 171  
    .rpt, 170  
    .sif, 171  
    .tcl, 46, 169  
    input files, 168  
    output files, 169  
    project control files, 169
- File Versioning System, 163  
    checking out backed-up files, 164  
    deleting solutions, 164  
    excluding directories, 163  
    handling input files, 163  
    how Catapult backs-up files, 164
- Files  
    .catapult.tcl, 360  
    adding to input, 93  
    cycle-accurate output, 170  
    directives.tcl, 44  
    in the working directory, 45  
    input, 168  
    log file, 169  
    messages.txt, 44  
    output reports, 171  
    project, 43, 169  
    project control, 169  
    removing from input, 93  
    RTL output, 170  
    schematic output, 171  
    SIF, 46, 171  
flow commands  
    flow get, 398  
    flow package forget, 401  
    flow package names, 402  
    flow package option add, 403  
    flow package option get, 405  
    flow package option remove, 406  
    flow package option set, 407  
    flow package present, 408  
    flow package provide, 410

- flow package require, 412
  - flow package script, 414
  - flow package vcompare, 415
  - flow package versions, 417
  - flow package vsatisfies, 418
  - flow provide, 420
  - flow run, 422
  - Flows
    - command tasks, 364
    - Concat, 145
    - customization, 609
    - CXXAnalysis for C code, 155
    - disabling, 78
    - editing properties, 78
    - enabling, 78
    - Flow Manager window, 77
    - FLOWS\_API database, 615
    - for verification, 515
    - infrastructure, 609
    - IP-XACT, 142
    - SCVerify, 515
    - SLEC formal verification, 565
  - FSM\_ENCODING directive, 111, 114, 187, 319
  - G —
    - GATE EFFORT directive, 319
    - GATE\_EXPAND\_MIN\_WIDTH directive, 320
    - GATE\_MIN\_WIDTH directive, 320
    - GATE\_REGISTERS directive, 321
    - GEN\_EXTERNAL\_ENABLE directive, 321
    - General command syntax, 349
    - General option, 175
    - go commands, 47, 423
    - gotolink catapult\_lb\_useref\_uv
      - manual title, 37
  - H —
    - Handshake
      - enabling signals, 98
      - I/O components, 292
      - process level, 280
      - SystemC designs, 281
      - transaction level, 281
    - handshaking, 183
  - Hardware
    - options, 186
  - HDL output, 170
  - Help
    - help command, 424
    - help message, 427
    - help switch, 358
    - help switch, 358
    - tool bar button, 67
  - history tracking, 359
  - hls\_builtin pragma, 342
  - hls\_design pragma, 342
  - hls\_direct\_input pragma, 342
  - hls\_fixed pragma, 344
  - hls\_map\_to\_operator pragma, 344
  - hls\_noglobals pragma, 345
  - hls\_pipeline\_init\_interval pragma, 345
  - hls\_remove\_out\_reg pragma, 345
  - hls\_resource pragma, 347
  - hls\_stream pragma, 347
  - hls\_unroll pragma, 347
- I —
- I/O constraints, 248
  - I/O delay, 294
  - stage replication, 116
- Idle signals
    - directive, 184
    - enable, 184
    - register, 184
  - IDLE\_SIGNAL directive, 321
  - IGNORE\_DEPENDENCY\_FROM directive, 322
  - ignore\_memory\_precedences command, 429
  - IGNORE\_PROCESS directive, 323
  - info Switch, 358
- Input files
    - adding to project, 93
    - removing from project, 93
  - Input options, 188
  - INPUT\_DELAY directive, 109, 117, 296, 324
  - INPUT\_REGISTER directive, 324
  - Interface options
    - hardware, 181
  - Interface Synthesis, 283
  - I/O signals, 284

INTERLEAVE directive, 260, 261

Iteration

count, 230

interval, 228

— L —

Libraries

custom component, 604

License report, 370

log file, 44, 93

logfile close command, 432

logfile isopen command, 433

logfile message command, 434

logfile move command, 439

logfile name command, 440

logfile open command, 441

logfile save\_commands command, 442

Loops, 225

constraints, 225

dependency chains, 232

distributed pipelining, 121, 316, 317

iteration count, 230

iteration interval, 228

ITERATIONS directive, 325

MERGEABLE directive, 238, 327

merging, 121, 234

merging and architectural constraints, 235

optimization, 225

pipeline flushing, 245

pipeline for throughput, 239, 241

pipeline to reduce latency, 239

PIPELINE\_INIT\_INTERVAL directive,  
331

PIPELINE\_RAMP\_UP directive, 331

pipelining, 121, 239

STAGE\_REPLICATION directive, 339

timing optimization, 226

UNROLL directive, 340

unrolling, 121, 231

wait statements and I/O function calls, 235

— M —

MAP\_TO\_MODULE directive, 115, 325

-match Switch, 356, 357

MEM\_MAP\_THRESHOLD directive, 259

Memories, 248

component parameters, 649, 652

export memory map for IP-XACT, 142

EXTERNAL\_MEMORY directive, 319

mapping arrays, 251

MEM\_MAP\_THRESHOLD directive, 327

optimizations, 258

packing modes, 116, 253

PACKING\_MODE directive, 330

STREAM directive, 339

streaming, 271

word width, 255

WORD\_WIDTH directive, 340

Memory splitting, 260

block size method, 117, 262

BLOCK\_SIZE directive, 310

INTERLEAVE directive, 325

interleave method, 117, 261

ModelSim scripts, 170

— N —

Netlist

configurable file basenames, 146, 192

configuring sub-block names, 192

custom component, 605

object name format, 49

packaging all output files, 190

Netlists

concat flow, 145

— O —

Operating procedures

adding input files, 93

creating a working directory, 45

generating output files, 141

invoking the GUI, 59

preparing the C source files, 95

removing input files, 93

scheduling the design, 127

setting iteration count, 230

setting the working directory, 92

setting up the design, 96

specifying architectural constraints, 104

Operational procedures

setting directives, 299

Optimizations

% assignment overhead, 110

- Adder trees, 329  
clock gating, 109, 113  
general design constraints, 273  
I/O access, 229  
loop dependency chains, 232  
loop iterations, 229  
loop merging, 234  
loop pipeline flushing, 245  
loop pipelining overview, 239  
loop unrolling, 231  
loops and timing, 226  
loops overview, 225  
low power, 110, 113, 319, 320, 321  
memories and arrays overview, 248  
memories, automatic, 258  
overview, 225  
speculative execution, 110, 113
- Options  
commands, 443  
constraint editor, 195  
DCPower, 203  
default values, 173  
Design Compiler options, 204  
for toolkits, 198  
general, 175  
input, 188  
messages, 176  
ModelSim, 199  
NCSim, 207  
Novas flow, 208  
OSCI flow, 209  
output files, 189  
PowerPlay, 210  
Precision RTL Synthesis, 201  
project initialization, 178  
    architectural, 184  
    component libraries, 180  
    hardware, 186  
    interface, 181  
    version control, 179  
RTL Compiler Synthesis, 210  
schedule viewer, 196  
schematic viewer, 197  
setting defaults, 173  
SpyGlassPower, 218
- synthesis  
    Design Compiler, 204  
    Precision RTL, 201  
    RTL Compiler, 210  
    text editor, 195  
    XPower, 222  
options commands, 443  
    defaults, 444  
    exists, 446  
    get, 447  
    load, 450  
    save, 452  
    set, 455
- Output file  
    configurable basenames, 146, 192
- Output file types, 169
- Output files  
    cycle reports, 141  
    cycle-accurate, 142  
    generating, 141  
    options, 189  
    packaging all netlist files, 190  
    RTL, 142  
    RTL report, 141  
    schematics, 143  
    synthesis scripts, 144  
    viewing in the GUI, 141
- OUTPUT\_DELAY directive, 109, 117, 296, 330
- OUTPUT\_REGISTER directive, 330
- P —
- Packing mode algorithms, 253  
PACKING\_MODE directive, 253  
Path and Sub-path Commands, 353  
Path and Sub-Path Search Rules, 353  
Path and Sub-Path Switches, 356  
Percent sharing allocation  
    CLOCK\_OVERHEAD directive, 113, 312
- Pipelining  
    PIPELINE\_INIT\_INTERVAL directive, 331
- Pipelining  
    distributed pipelining algorithm, 121  
    loops, 121  
    loops, distributed pipelining, 245

- loops, overview, 239
  - nested loops, 241
  - PIPELINE\_RAMP\_UP directive, 331
  - STAGE\_REPLICATION directive, 339
  - Ports
    - naming conventions, 285
    - splitting, 256
  - Power analysis tools, 583
  - Power estimation
    - DCPower flow options, 203
    - PowerPlay flow options, 210
    - SpyGlassPower flow options, 218
    - XPower flow options, 222
  - Power optimizations
    - clock gating, 109, 113, 319, 320, 321
    - idle signal, 110, 113
    - idle signals, 321
  - Pragmas, 341
    - hls\_builtin, 342
    - hls\_design, 95, 342
    - hls\_direct\_input, 342
    - hls\_fixed, 344
    - hls\_map\_to\_operator, 344
    - hls\_noglobals, 345
    - hls\_pipeline\_init\_interval, 345
    - hls\_remove\_out\_reg, 345
    - hls\_resource, 347
    - hls\_stream, 347
    - hls\_unroll, 347
  - purging pragmas from netlists, 190
  - Precision RTL Synthesis scripts, 171
  - Project
    - .ccs file, 44, 169
    - about, 43
    - commands, 457
    - initialization, 178
    - QoF bar chart, 80
    - QoF report tables, 79
    - QoF XY Plot, 80
    - restoring, 44
    - saving, 44
    - solutions, 44
    - view files in GUI, 70
  - project commands, 457
    - project close, 458
  - project get, 459
  - project load, 461
  - project report, 463
  - project save, 465
  - project set, 466
- Q —
- quit command, 468
- R —
- Ready flag, 183
    - directive, 332
  - recurse Switch, 358
  - REG\_MAX\_FANOUT directive, 187, 279, 333
  - REGISTER\_IDLE\_SIGNAL directive, 333
  - REGISTER\_THRESHOLD directive, 259
- Registers
- max fanout optimization, 187, 279
  - REG\_MAX\_FANOUT directive, 333
  - REGISTER\_IDLE\_SIGNALT directive, 333
- REGISTER\_THRESHOLD directive, 334
  - setting default threshold option, 185
- Reports, 141
  - clock uncertainty, 276
- Reset signal
  - clear all registers, 111
  - configuring, 99
  - RESET\_ASYNC\_ACTIVE directive, 335
  - RESET\_ASYNC\_NAME directive, 335
  - RESET\_CLEAR\_ALL\_REGS directive, 336
  - RESET\_KIND directive, 336
  - RESET\_SYNC\_ACTIVE directive, 336
  - RESET\_SYNC\_NAME directive, 337
- resource add command, 470
- resource command, 469
- resource remove command, 473
- Resources
  - about, 114
  - constraining, 114
  - map to module, 115
  - memory arrays, 250
  - packing mode, 116, 253
  - splitting, 260

-return Switch, 356  
RTL output files, 141, 142, 170  
RTL reports, 141

— S —

Safe FSM  
    directive, 187  
    SAFE\_FSM directive, 338  
    setting constraint, 109, 113  
    setting default option, 187  
Saving  
    a project, 44  
Schedule  
    viewer options, 196  
Schematics  
    critical path, 143  
    data path, 143  
    file type, 171  
    object name format, 49  
    output files, 143  
    view states, 143  
    viewer, 149  
    viewer options, 197  
Script  
    simulation output file, 170  
    synthesis output file, 171  
    Tcl commands, 358  
SCVerify flow  
    C++, 516  
    FAQ, 551  
    generated makefiles, 526  
    I/O transaction controls, 534  
    linking to external object code, 522  
    options, 212  
    SystemC, 517  
    troubleshooting, 552  
    using the Valgrind utility, 527  
set\_working\_dir command, 475  
SIF Files, 46, 171  
Simulation  
    ModelSim options, 199  
    NCSim options, 207  
    Novas flow options, 208  
    OSCI flow options, 209  
    script files, 170  
SLEC flow, 565

flop mapping, 572  
options, 215, 217  
supported I/O components, 571  
solution commands  
    solution file add, 482  
    solution file remove, 487  
    solution file restore, 488  
    solution file set, 489  
    solution netlist, 494  
    solution new, 492  
    solution remove, 498  
    solution rename, 499  
    solution report, 500  
    solution restore, 504  
    solution select, 505  
    solution timing, 506  
Solutions  
    about, 44  
    comparing results, 79, 80  
    directory structure, 46  
    selecting from tool bar, 66  
    view in GUI, 70  
SPECULATE  
    directive, 185  
Speculative execution  
    setting default option, 185  
    SPECULATE directive, 338  
STAGE\_IO\_CNS directive, 338  
Start flag  
    START\_FLAG directive, 339  
Startup file  
    .catapult.tcl file, 360  
    catapult.ini, 224  
State model, 47  
Streaming  
    memory, 271  
Synthesis  
    Design Compiler options, 204  
    Precision RTL options, 201  
    RTL Compiler options, 210  
    script files, 144, 171  
System messages, 159  
    help, 427  
SystemC  
    handshake signals, 281

— T —

Tcl  
language, 349  
Tcl script  
command line with path, 359  
Interactive Command Line Shell, 358  
LOG file, 358  
run script, 359  
TECHLIBS directive, 340  
Text editor  
features, 165  
options, 195  
Timing optimization  
loops, 226  
Timing Report  
clock uncertainty, 276  
Tool Bar, 65  
Toolkits options, 198  
Top-level function  
specify in GUI, 100  
specifying in C code, 95  
Transcript messages, 74  
identifiers, 161  
options dialog box, 176  
structure of, 159  
Transcript window, 72

— U —

User interface  
activating a solution, 66, 71, 79  
Bar Chart window, 80  
changing the window layout, 89  
command input window, 72  
Constraint editor window, 69  
Flow Manager window, 77  
Help button, 67  
opening a selected file, 70  
Project Files window, 70  
session window, 60  
Table window, 79  
Task bar, 67  
tool bar, 65  
transcript window, 72  
XY Plot window, 80

— V —

Valgrind utility  
for memory checks in SCVerify, 527  
Valgrind utility  
default options, 221  
Verification  
flows, 515  
SCVerify flow, 515  
SLEC flow, 565  
Version control, 179  
view command  
view file, 512  
view schedule (Gantt chart), 509  
view schematic, 510  
view source, 513

— W —

Window layout controls, 89  
Word width  
constraining, 255  
splitting I/O ports, 256  
WORD\_WIDTH directive, 251, 256  
Work flow, 46  
GUI interface, 364  
Task Bar interface, 67  
Working directory  
automatically setting, 93  
setting from Task bar, 68  
setting in Shortcut, 60  
structure of, 45

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

---

## Third-Party Information

This section provides information on third-party software that may be included in the Catapult® family of products, including any additional license terms.

- This software application may include GCC 4.2.2 third-party software. GCC 4.2.2 is distributed under the terms of the General Public License version 2 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <install\_directory>/Mgc\_home/shared/legal/gnu\_gpl\_2.0.pdf. To obtain a copy of the source code to the files licensed under the GNU GPL v2, send a request to request\_sourcecode@calypto.com. This offer shall only be available for three years from the date Calypto Design Systems first distributed GNU GPL v2 covered source code.
- This software application may include gdb version 7.0 third-party software. gdb version 7.0 is distributed under the terms of the GNU General Public License version 2.0 and version 3.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <your\_Mentor\_Graphics\_documentation\_directory>/legal/gnu\_gpl\_2.0.pdf and <your\_Mentor\_Graphics\_documentation\_directory>/legal/gnu\_gpl\_3.0.pdf. Portions of this software may be subject to the GNU Free Documentation License version 1.1. You can view a copy of the GNU Free Documentation License version 1.1 at: <your\_Mentor\_Graphics\_documentation\_directory>/legal/gnu\_free\_doc\_1.1.pdf. Portions of this software may be subject to the GNU Free Documentation License version 1.2. You can view a copy of the GNU Free Documentation License version 1.2 at: <your\_Mentor\_Graphics\_documentation\_directory>/legal/gnu\_free\_doc\_1.2.pdf. Portions of this software may be subject to the GNU Library General Public License version 2.0. You can view a copy of the GNU Library General Public License version 2.0 at: <your\_Mentor\_Graphics\_documentation\_directory>/legal/gnu\_library\_gpl\_2.0.pdf. To obtain a copy of the gdb version 7.0 source code, send a request to request\_sourcecode@calypto.com. This offer shall only be available for three years from the date Calypto Design Systems first distributed gdb version 7.0. gdb version 7.0 may be subject to the following copyrights:

© 1987 Regents of the University of California.  
All rights reserved.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the University of California, Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

© 1983, 1990 Regents of the University of California.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. [rescinded 22 July 1999]
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

© 1993, 1991, 1990, 1989, 1988, 1987 Carnegie Mellon University  
All Rights Reserved.

Permission to use, copy, modify and distribute this software and its documentation is hereby granted, provided that both the copyright notice and this permission notice appear in all copies of the software, derivative works or modified versions, and any portions thereof, and that both notices appear in supporting documentation.

CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS" CONDITION. CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.

Carnegie Mellon requests users of this software to return to

Software Distribution Coordinator or Software.Distribution@CS.CMU.EDU  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh PA 15213-3890

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

- This software application may include Tcl version 8.5.8 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Tcl version 8.5.8 may be subject to the following copyrights:

© 1988, 1993, 1994 The Regents of the University of California All rights reserved.  
All Rights Reserved

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:  
This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This software application may include Tcl Readline third-party software that may be subject to the following copyright:

© 1998 - 2000, Johannes Zellner johannes@zellner.org All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

\* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following

disclaimer in the documentation and/or other materials provided with the distribution.

\* Neither the name of Johannes Zellner nor the names of contributors to this software may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This software application may include Tk version 8.5.8 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Tk version 8.5.8 may be subject to the following copyrights:

© David Koblas

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

© 1998 Hutchison Avenue Software Corporation

<http://www.hasc.com>

info@hasc.com

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "AS IS." The Hutchison Avenue Software Corporation disclaims all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to this code and accompanying documentation.

© 1985, 1986, 1987, 1989, 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1987 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- This software application may include TKtreectrl version 2.2.8 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied TKtreectrl version 2.2.8 may be subject to the following copyrights:

This software is copyrighted by Tim Baker and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

- This software application may include gmake version 3.81 third-party software. Gmake version 3.81 is distributed under the terms of the General Public License version 2.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <install\_directory>/Mgc\_home/shared/legal/gnu\_gpl\_2.0.pdf. Portions of this software may be subject to the GNU Free Documentation License version 1.2. You can view a copy of the GNU Free Documentation License version 1.2 at: <install\_directory>/Mgc\_home/shared/legal/gnu\_free\_doc\_1.2.pdf. Portions of this software may be subject to the Library General Public License version 2.0. You can view a copy of the Library General Public License version 2.0 at: <install\_directory>/Mgc\_home/shared/legal.gnu\_library\_gpl\_2.0.pdf. To obtain a copy of the source code to gmake version 3.81, send a request to request\_sourcecode@calypto.com. This offer shall only be available for three years from the date Calypto Design Systems first distributed gmake version 3.81.
- This software application may include SystemC third-party software.

©2001 Dr. John Maddock. All rights reserved.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Dr. John Maddock makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

©1994 Hewlett-Packard Company. All rights reserved.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

©1996 Silicon Graphics Computer Systems, Inc. All rights reserved.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

- This software may include Info-Zip Unzip third-party software.

Copyright (c) 1990-2005 Info-ZIP. All rights reserved.

For the purposes of this copyright and license, "Info-ZIP" is defined as the following set of individuals:

Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly, Hunter Goatley, Ed Gordon, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs, Kai Uwe Rommel, Steve Salisbury, Dave Smith, Steven M. Schweda, Christian Spieler, Cosmin Truta, Antoine Verheijen, Paul von Behren, Rich Wales, Mike White

This software is provided "as is," without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software.

- This software application may include CUDD third-party software that may be subject to the following copyrights:

© 1995-2004, Regents of the University of Colorado. All rights reserved.  
© 1985 by Digital Equipment Corporation, Maynard, MA. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the University of Colorado nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The information in this software is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

no responsibility for the use or reliability of its software on equipment which is not supplied by Digital

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by Digital Equipment Corporation. The name of Digital Equipment Corporation may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Do not take internally. In case of accidental ingestion, contact your physician immediately.

- This software application may include getline third-party software that is distributed by Chris Thewalt and my be subject to the following copyrights

©1991, 1992, 1993 by Chris Thewalt (thewalt@ce.berkeley.edu)

- This software application may include Mersenne Twister third-party software. Mersenne Twister is distributed under the terms of the Mersenne Twister License Agreement. You can view the complete license at: <install\_directory>/Mgc\_home/shared/legal/mersenne\_twister.pdf.

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

- This software application may include SystemC version 2.2 third-party software. To obtain a copy of the SystemC source code, send a request to request\_sourcecode@calypto.com. SystemC software is distributed under the SystemC Open Source License Agreement (Download, Use and Contribution License Agreement Version 3.0) and is distributed on an

"AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. You can view a copy of the license at: <install\_directory>/Mgc\_home/shared/legal/systemc\_open\_source\_3.0.pdf. Portions of this software are subject to the Boost License v.1.0. You can view a copy of the license at: <install\_directory>/Mgc\_home/shared/legal/boost\_1.0.pdf. SystemC version 2.2 may be subject to the following copyrights:

Copyright (c) 1994 Hewlett-Packard Company

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright (c) 1996 Silicon Graphics Computer Systems, Inc.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Calling this script install-sh is preferred over install.sh, to prevent `make' implicit rules from creating a file called install from it when there is no Makefile.

This script is compatible with the BSD install script, but was written from scratch. It can only install one file at a time, a restriction shared with many OS's install programs.

Copyright (c) 1993 by David Keppel

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice and this notice appear in all copies. This software is provided as a proof-of-concept and for demonstration purposes; there is no representation about the suitability of this software for any purpose.

- This software application may include libxml2 version 2.6.31 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. libxml2 version 2.6.31 may be subject to the following copyrights:

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

Copyright (C) 2000 Bjorn Reese and Daniel Veillard.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

© 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 2000 Gary Pennington and Daniel Veillard.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

© 1998, 2000, 2001 Bjorn Reese and Daniel Stenberg.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

© 2001 Bjorn Reese <breese@users.sourceforge.net>

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

- This software application may include TLM version 2.0 third-party software. TLM version 2.0 is distributed under the terms of the **SystemC Open Source License Agreement v3.0** and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <install\_directory>/Mgc\_home/shared/legal/systemc\_open\_source\_3.0.pdf. To obtain a copy of the TLM version 2.0 source code, send a request to request\_sourcecode@calypto.com.
- This software application may include tcbload version 1.7 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.

- This software application may include portions of Boost Spirit version 1.8.5 third-party software. Boost Spirit version 1.8.5 is distributed under the terms of the Boost Software License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <install\_directory>/Mgc\_home/shared/legal/boost\_1.0.pdf. Boost Spirit version 1.8.5 may be subject to the following copyrights:

© 1996, 1997 Silicon Graphics Computer Systems, Inc.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1994 Hewlett-Packard Company

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty .