



UXE Command Reference Manual

Product Version 17.1

April 2017

© 2017 Cadence Design Systems, Inc. All rights reserved.
Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Product UXE contains technology licensed from, and copyrighted by: Lucent Technologies, Inc, and is © 1991-1998, Lucent Technologies, Inc, All rights reserved; Carnegie Mellon University, and is © 1988, 1991 by Carnegie Mellon University, All rights reserved; loi Kim Lam, and is © 1993-1996 by loi Kim Lam, All rights reserved; Qlogic Fibre Channel Driver, © 1989, 1991 Free Software Foundation, All rights reserved; Emulex Fibre Channel Driver, © 1989, 1991 Free Software Foundation, Inc, All rights reserved; Prentice Hall, © 1995 by Prentice Hall, All rights reserved; © Regents of the University of California © Sun Microsystem, Inc. © Scriptics Corporation, Novus Copyright (C) 1996 - 2008 by Novas Software, Inc., Source III (C) Copyright 2006 Source III, Inc., U.C Riverside COPYRIGHT (C) 1995, 1996 by Nexsyn Design Technology Inc. All rights reserved.

Lucent Technologies disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness.

Associated third-party license terms may be found at <ROOTDIR>/install/Copyrights

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Patents: Cadence Product UXE, described in this document, is protected by U.S. Patents 4,914,612, 5,448,522, 5,551,013, 5,563,829, 5,622,520, 5,680,583, 5,761,488, 5,781,903, 5,819,065, 5,822,564, 5,864,838, 5,870,316, 5,884,066, 5,920,712, 5,940,603, 5,963,736, 5,970,240, 6,011,730, 6,011,744, 6,035,117, 6,051,030, 6,141,636, 6,151,258, 6,469,375, 6,618,698, 6,681,377, 6,782,355, 6,850,880, 6,901,359, 7,043,417, 7,047,179, 7,054,802, 7,089,538, and 7,107,203.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third-party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

1

<u>Introduction</u>	67
<u>Command Specification</u>	67
<u>Compile the design for ICE mode using xeCompile and run using xeDebug</u>	68
<u>Compile the design for SA mode using IXCOM tools and run using xeDebug (Simulation Acceleration)</u>	68
<u>Compile the design for SA mode using IXCOM tools and run using <i>irun</i> (Software Simulation)</u>	69
<u>Compile and run the design for SA mode using <i>irun</i> (Simulation Acceleration using <i>irun</i>)</u>	69
<u>Introducing XEL</u>	69
<u>Using XEL Shell</u>	70
<u>Using XEL Scripts</u>	72
<u>Conventions Used in Command Syntax</u>	72
<u>Tcl References</u>	73

2

<u>UXE Command List</u>	75
<u>Programs and Utilities</u>	76
<u>Compiler Programs and Utilities</u>	76
<u>Run-time Programs and Utilities</u>	77
<u>Command Set for SA Mode</u>	77
<u>Compile-time Command Set for SA Mode</u>	77
<u>xeDebug Command Set for SA Mode</u>	77
<u><i>irun</i> Command Set for SA Mode</u>	83
<u>Command Set for ICE Mode</u>	85
<u>xeCompile Command Set</u>	85
<u>xeDebug Command Set for ICE Mode</u>	89
<u>Common XEL Commands</u>	92

<u>vvmDebug Command Set</u>	93
3		
<u>Programs and Utilities</u>	101
<u>generateDYNPwaveform</u>	102
[-fsdb -sst2]	102
<phy name>	102
<waveform filename>	102
[-session <session number>]	102
<u>ipBuilder</u>	103
<input file>	103
<u>ModelChecker</u>	104
<design name>	104
[options]	104
<u>gateCount</u>	106
<u>libTo1801</u>	107
<libertyFilename> <libertyDirectory>	108
<u>primCount</u>	110
<u>test server</u>	111
<top cell name>	112
<top cell name> -reserve [-key <key value>] [-timeout <t> [<unit>]]	113
<top cell name> -location [-all] [-sahost <host>] [-nosahost] [-host <host>]	115
-rmkey [<key value>]	115
-hdver	116
- json [-output <jsonFile>]	116
-l <num boards> <starting board> [<top cell name>]	116
-l <top cell name>	117
-d <b.d>	117
-notime	118
-update [<sec>]	118
-sainfo [-update [<sec>]]	118
-listUxeData	119
-moreinfo	119
-noclean	119
-rsh <host> -ssh <host>	120

UXE Command Reference Manual

<u>help</u>	120
<u>vCompare</u>	121
<u>-in <stim filename></u>	121
<u>-result <res filename></u>	121
<u>-log <log filename></u>	121
<u>-comp <comp filename></u>	121
<u>-start <vecNumber></u>	121
<u>-count <numVectors></u>	121
<u>-interact [Y N]</u>	122
<u>-mm <maxMismatches></u>	122
<u>verigen</u>	123
<u>-inst <path1> <path2> <path3> ...</u>	123
<u>-net <path1> <path2> <path3> ...</u>	123
<u>-net -f <nets file></u>	123
<u>-path <net1> <net2></u>	124
<u>-path -f <pair file></u>	124
<u>-levels <num></u>	124
<u>-forward <num></u>	124
<u>-back <num></u>	124
<u>-stop [atStorage atStorageData]</u>	124
<u>-rename <rename string></u>	124
<u>-out <output></u>	124
<u>-bus</u>	125
<u>-force</u>	125
<u>-help</u>	125
<u>Examples</u>	125
<u>vvmDebug</u>	126
<u>-gui</u>	126
<u>-noecho</u>	126
<u>-shm</u>	127
<u>-fsdb</u>	127
<u>-vcd</u>	127
<u>-init -input <script file> @<command></u>	127
<u>-log <log filename></u>	128
<u>-append_log</u>	128
<u>-append_key</u>	128

UXE Command Reference Manual

<u>checksyntax</u>	128
<u><script> [<script_option> ...][probelist][rcfile]</u>	128
<u>clean</u>	129
<u>cleanall</u>	129
<u>version</u>	129
<u>help</u>	129
fvCompute	130
<u>fvCompute <phy_file> [-nice <n>]</u>	131
xeCompile	132
<u>-init <script_file> @<command></u>	132
<u>checksyntax</u>	132
<u><script> [<script_option> ...]</u>	133
<u>version</u>	133
<u>clean</u>	133
<u>cleanall</u>	133
<u>help</u>	134
xeDebug	135
<u>-gui</u>	135
<u>-noecho</u>	135
<u>-shm</u>	136
<u>-fsdb</u>	136
<u>-vcd</u>	136
<u>-init -input <script_file> @<command></u>	136
<u>-host <host_name></u>	137
<u>-dynp</u>	137
<u>-log <log_filename></u>	137
<u>append_log</u>	137
<u>append_key</u>	137
<u>checksyntax</u>	137
<u>--ncsim <ncsim_option> ... [--]</u>	138
<u><script> [<script_option> ...]</u>	138
<u>wave <phyfile> [<probescrpt> <probelist> <rcfile>]</u>	138
<u>offline <session> <phyfile> [<probescrpt> <probelist> <rcfile>]</u>	139
<u>simlogsize <value> [-simincfile <value>]</u>	139
<u>clean</u>	140
<u>cleanall</u>	140

UXE Command Reference Manual

<u>version</u>	140
<u>help</u>	140
<u>xeclient</u>	141
<u>xeGenReport</u>	142
<u>-host <configmgr_host1> [<configmgr_host2> ... <configmgr_hostn>]</u>	143
<u>-period <year>.<start_month> [<year>.<end_month>]</u>	143
<u>[-output <directory name>]</u>	143
<u>[-maxUser <number>]</u>	143
<u>[-maxDesign <number>]</u>	144
<u>[-daytimeHour <startHour> <endHour>]</u>	144
<u>[-alias <file>]</u>	144
<u>[-help]</u>	145

4

General-purpose Commands

<u>exit</u>	148
<u>[code]</u>	148
<u>help</u>	149
<u>-gui</u>	149
<u><command></u>	149
<u>msgControl</u>	150
<u><message_type></u>	150
<u>on off</u>	150
<u>quit</u>	151
<u>[code]</u>	151
<u>redirect</u>	152
<u><command></u>	152
<u><redirection_operator></u>	152
<u><filename></u>	153

5

Import Commands

<u>designImport</u>	156
<u>-noautomem</u>	156
<u>-keepExistingLinks</u>	156

UXE Command Reference Manual

<u><file></u>	156
<u>-noparallel</u>	156
<u>-useLSF</u>	157
<u>Examples</u>	157
<u>impDumpConfig</u>	158
<u><file></u>	158
<u>Examples</u>	158
<u>implLoadConfig</u>	159
<u><file></u>	159
<u>Examples</u>	159
<u>importOption</u>	160
{mode incr full}	160
{compatibility sl ns ci cs cd ir}	161
{library <lib>}	162
{libType design asic generic ref}	162
{memFile <file>}	162
{includes <dir>}	162
{special <arg>}	162
{DEFINE {<variable>[=<value>] ...}}	163
{search on off}	163
{protection on off}	163
<u>Examples</u>	164
<u>linkRefLib</u>	165
<u>-keepExistingLinks</u>	165
<u><lib></u>	165
<u>Examples</u>	166
<u>netlistFile</u>	167
<u>verilog <netlist></u>	167
<u>netlists <netlist></u>	167
<u>attribute <file></u>	167
<u>exclude <filename></u>	168
<u>Examples</u>	168
<u>refLib</u>	169
<u>-keepExistingLinks</u>	169
<u><lib> ...</u>	169
<u>Examples</u>	169

6

<u>HDL Compiler Commands</u>	171
<u>hdlDefineLib</u>	172
<u>-add <lib> <dir></u>	172
<u>-rm <lib></u>	172
<u>-clr</u>	172
<u>Examples</u>	172
<u>hdlImport</u>	174
<u>-full</u>	174
<u>-u</u>	174
<u>-2001</u>	174
<u>-v2008</u>	176
<u>-l <lib></u>	176
<u>-sv</u>	176
<u>-atb</u>	177
<u>-ignorePragma+<module_name>[+...]</u>	177
<u>Examples</u>	178
<u>hdlInputFile</u>	179
<u>-add <file></u>	179
<u>-errormax <arg></u>	179
<u>-rm <file>...</u>	180
<u>-clr</u>	180
<u>-synth_prefix <pragma_name></u>	180
<u>-rm_synth_prefix <pragma_name></u>	180
<u>-f <commandFile></u>	180
<u>-v <filename></u>	180
<u>-y <directory></u>	180
<u>+define+<var1>+..</u>	181
<u>+libext+<ext1>+..</u>	181
<u>+incdir+<path1>+..</u>	181
<u>-u</u>	181
<u>Examples</u>	181
<u>hdlLang</u>	182
<u>verilog</u>	182
<u>vhdl</u>	182

UXE Command Reference Manual

<u>Examples</u>	182
<u>hdlLogFile</u>	183
<u><log_file></u>	183
<u>Examples</u>	183
<u>hdlOutputFile</u>	184
<u>-add -f [verilog vhdl lec debussy qel] <file></u>	184
<u>-rm <file></u>	186
<u>-clr</u>	186
<u>Examples</u>	186
<u>hdlSkipModule</u>	187
<u>-add [<library>.]<module></u>	187
<u>-rm [<library>.]<module></u>	187
<u>-clr</u>	187
<u>Examples</u>	187
<u>hdlSynthesize</u>	189
<u>-all</u>	189
<u>-only <module_name></u>	189
<u><top_module></u>	190
<u>-alu</u>	190
<u>-alu<arg></u>	190
<u>-dontStopOnError</u>	190
<u>-memory</u>	190
<u>-memory<N></u>	190
<u>-keepVhdlCase</u>	191
<u>-vhdlToLowercase</u>	191
<u>-noNamePreservation</u>	191
<u>-fanoutNetOnly</u>	191
<u>-keepAllFlipFlop</u>	191
<u>-keepRtlSymbol</u>	191
<u>-parallel</u>	192
<u>-parallel<number></u>	192
<u>-parallelLSF<number></u>	192
<u>-append</u>	192
<u>-portNaming <pattern></u>	192
<u>-D<generic>=<value></u>	193
<u>-rename <module_name></u>	193

UXE Command Reference Manual

<u>-displayBuffer[+depth=<num>] [+timeOut=<num>] [+port=<num>]</u>	194
<u>-siliconSynthesisCompatible</u>	194
<u>Examples</u>	194
<u>hdlWorkPath</u>	196
<u><dir></u>	196
<u>Examples</u>	196
 <u>7</u>	
<u>Database Commands</u>	197
<u>db</u>	198
<u>cd <instance></u>	198
<u>pwd</u>	198
<u>ls [-insts] [-nets] [-all] [<glob_pattern>...]</u>	198
<u>find</u>	198
<u>-name <pattern></u>	199
<u>-scope <scope></u>	199
<u>-type net term inst module</u>	199
<u>-style glob regex</u>	199
<u>-command</u>	199
<u>-case 0 1</u>	200
<u>-levels <n></u>	200
<u>-max <n></u>	200
<u>-showgen 0 1</u>	200
<u>Examples</u>	200
<u>moduleNet</u>	202
<u><lib></u>	202
<u><mod></u>	202
<u><net></u>	202
<u>Examples</u>	202
<u>getDatabaseInfo</u>	204
 <u>8</u>	
<u>User Data Commands</u>	205
<u>aluTransform</u>	208
<u>-add {*} alu2gates OFF}</u>	208

UXE Command Reference Manual

<u>-add { * alu2gates ON}</u>	208
<u>-rm { * alu2gates}</u>	208
<u>-add { * aluMaxUtilization <number>}</u>	208
<u>-rm { * aluMaxUtilization}</u>	208
<u>andMultiDrv</u>	209
<u>-add {<signal>}</u>	209
<u>Examples</u>	209
<u>breakNet</u>	210
<u>-add <net_name></u>	210
<u>-add {<net_name> <number_of_units>}</u>	211
<u>-rm <net_name></u>	211
<u>Examples</u>	211
<u>breakNetHalfCycle</u>	212
<u>breakPin</u>	213
<u>-add</u>	213
<u><hierarchical_pin_name></u>	214
<u>IFO EFO AFO</u>	214
<u>breakPinHalfCycle</u>	216
<u>cableConnection</u>	217
<u>-add {<cable_label> <conn_loc> [I/O_technology]}</u>	217
<u>-add {<cable_label> LTA_A LTA_B <conn_loc> [I/O_technology]}</u>	218
<u>-add {<cable_label> LTA_C LTA_D <conn_loc>}</u>	219
<u>-rm <cable_label></u>	219
<u>-dump [<filename>]</u>	219
<u>-D2Y -Y2D <cable_label></u>	219
<u>-clear</u>	219
<u>Examples</u>	219
<u>clockAssign</u>	221
<u>-add {<signal1>[~] <signal2> [<N>]}</u>	221
<u>-rm <signal1></u>	221
<u>Examples</u>	221
<u>clockDelay</u>	224
<u><clockname></u>	224
<u><nDly></u>	224
<u>clockFrequency</u>	225
<u>-add {<clock_source_name> <frequency>}</u>	226

UXE Command Reference Manual

<u>-rm <clock source name></u>	226
clockOption	227
<u>-add {technology [CAKE <number> SCM]}</u>	227
<u>-add {activeEdge both positive negative}</u>	227
<u>-add {ignoreOutput ON OFF}</u>	228
Examples of Clock Generation	228
clockSource	231
<u>-add <net name>...</u>	231
<u>-rm <net name>...</u>	232
compilerOption	233
<u>-add -rm {<parameter> <value>}...</u>	235
<u>-add -rm {allWiresParityCheck ON OFF}</u>	235
<u>-add {mode vd ice}</u>	235
<u>-add {ioDelayUnit pd2tif pd2step uatif step ns}</u>	235
<u>-add {visionMode DYNP FV}</u>	236
<u>-add {enableMultiSampledIO 0 1 2}</u>	237
<u>-add {delayBidirIO OFF ON}</u>	237
<u>-add {infiniTrace OFF ON}</u>	237
<u>-add {memoryWritethru OFF ON}</u>	238
<u>-add {noTailgate OFF ON}</u>	238
<u>-add {sdllInstances <num_instances>}</u>	238
<u>-add {sdltTraceDepth <sdl_trace_depth>}</u>	238
<u>-add {sdldDisplayDepth <sdl_display_depth>}</u>	239
<u>-add {sdldDisplayWidth <sdl_display_width>}</u>	240
<u>-add {sdlgGlobalDisplay <sdl_global_display>}</u>	240
<u>-add {traceDepth <value>}</u>	240
<u>-add {traceDepthDYNP <value>}</u>	241
<u>-add {numCapturedNetsPerDomain <n>}</u>	241
<u>-add {numExtraCapturedNetsPerDomain <n>}</u>	242
<u>-add {compilerEffort LOW HIGH}</u>	242
<u>-add {optimizationEffort LOW HIGH}</u>	242
<u>-add {optimizationAlgo CLASSIC MODERN}</u>	243
<u>-add {placementEffort LOW HIGH}</u>	243
<u>-add {powerdb <powerdb_directory>}</u>	243
<u>-add {min_dcc_percentage <n>}</u>	243
<u>-add {max_dcc_percentage <n>}</u>	243

UXE Command Reference Manual

<u>-add {limfanout <n>}</u>	243
<u>-add {retiming OFF ON}</u>	245
<u>-add {stepMode 256 512 auto}</u>	245
<u>-add {enhancedTerminalTiming ON OFF}</u>	246
<u>-add {symmetricPreserveCables ON OFF}</u>	246
<u>-add {symmetricPreserveMemCards ON OFF}</u>	246
<u>-add {wireDuplication <value>}</u>	247
<u>-add {ModelChecker OFF ON}</u>	247
<u>-add {write_pgm OFF ON}</u>	248
<u>-add {write_fnets OFF ON}</u>	248
<u>-add {ModelCheckerOmit "<check_type> <check_type> ..."}</u>	248
<u>-add {cpfLogBufferDepth <n>}</u>	249
<u>-add {1801LogBufferDepth <n>}</u>	249
<u>-add {1801LogType { [powerDomain] [supplySet] [isolation] [retention] [powerState] [stateTransition] [portState] [pstState] [illegalIsoState] [illegalRetState] }}</u>	249
<u>-add {1801CoverageWidth [0-8]}</u>	250
<u>-add {XP capacityFeatures <N>}</u>	250
<u>-add {symmetricXP only ON}</u>	250
<u>-add {maxStep <number>}</u>	251
<u>-add {minStep <number>}</u>	251
<u>-add {DPATopNInst N}</u>	251
<u>-add {subDPATopNInst <fileName>}</u>	251
<u>-add {dumpDPATopNInst N}</u>	252
<u>-rm {<compiler_option>}...</u>	252
<u>create assertion control</u>	254
<u> -name <string></u>	254
<u> -assertions <assertion_list></u>	254
<u> -domains <power_domain_list></u>	254
<u> -assertion_control expression</u>	254
<u> -supply_set</u>	254
<u> -type reset</u>	254
<u>customIsolationCell</u>	255
<u> -add {<libraryName> <cellName>}</u>	255
<u> -rm {<libraryName> <cellName>}</u>	255
<u>delayAsynClockInputs</u>	256
<u> -add {<sig1> <sig2> ... [<N>]}</u>	256

UXE Command Reference Manual

<u>-rm {<sig1> <sig2>...}</u>	256
<u>Examples</u>	257
<u>delayBox</u>	258
<u>-add {<output name> <delay> <list of input names>}</u>	258
<u><output name></u>	258
<u><delay></u>	258
<u><list of input names></u>	259
<u>-dump <filename></u>	259
<u>-rm {<output name>}</u>	259
<u>Examples</u>	259
<u>delayBoxInput</u>	261
<u>-add {<output name> <delay> <list of input names>}</u>	261
<u><output name></u>	261
<u><delay></u>	261
<u><list of input names></u>	261
<u>-rm <output name> [<list of input names>]</u>	262
<u>Examples</u>	262
<u>emptyCell</u>	263
<u>-add {<libraryname> <cellname>}</u>	263
<u>-rm {<libraryname> <cellname>}</u>	263
<u>emulatorConfiguration</u>	264
<u>-add {host <hostname>}</u>	264
<u>-add {host <hostname>} {boards <sub_board>}</u>	265
<u>-add {file <filename>}</u>	266
<u>-add {file <filename>} {boards <sub_board>}</u>	266
<u>-add {boards <sub_board>}</u>	266
<u>-help</u>	266
<u>Examples</u>	267
<u>globalNet</u>	268
<u>-add {<net> <term>}</u>	268
<u>-rm {<net>}</u>	268
<u>-clear</u>	268
<u>hierAssign</u>	269
<u>-add {<net> <value>} ...</u>	269
<u>-add {<net> <source_net>} ...</u>	269
<u>-rm <net> ...</u>	269

UXE Command Reference Manual

<u>hwInfo</u>	270
<u>-host <hostname></u>	270
<u>-file <filename></u>	270
<u>-design</u>	271
<u>-orion</u>	271
<u>-buffer</u>	271
<u>-cable</u>	271
<u>-connector</u>	271
<u>-memory</u>	271
<u>-help</u>	271
<u>Examples</u>	272
<u>identify_power_domain</u>	275
<u>-name <powerDomainName></u>	275
<u>-modules <moduleNameList> ...</u>	275
<u>-shutoff condition <expression></u>	275
<u>-clear</u>	275
<u>instanceFV</u>	276
<u>-add [<instance_name> enable disable]</u>	276
<u>-rm [<instance_name>]</u>	276
<u>instanceStub</u>	278
<u>-add {a.b.c.d.}</u>	278
<u>keepLoadlessInstance</u>	279
<u>-add {a.b.c.d.}</u>	279
<u>-rm {a.b.c.d.}</u>	279
<u>keepLoadlessModule</u>	280
<u>-add {<LIB> <cellName>}</u>	280
<u>-rm {<LIB> <cellName>}</u>	280
<u>keepLoadlessNet</u>	281
<u>-add <net>...</u>	281
<u>-rm <net>...</u>	281
<u>keepNet</u>	282
<u>-add <net>...</u>	282
<u>-add [-errorok] {net1} {net2} ... {netN}</u>	282
<u>-rm <net>...</u>	282
<u>-addNreport <net_name></u>	283
<u>-addNreport <net_name></u>	283

UXE Command Reference Manual

<u>Examples</u>	283
<u>lp (Low-power Compile-time Command)</u>	284
<u>Options of Compile-Time lp Command for CPF</u>	285
<u>Options of Compile-Time lp Command for IEEE 1801</u>	286
<u>memoryTransform</u>	290
<u>-add {<memory> <parameter_name> <parameter_value> [<parameter_name><parameter_value>]}</u>	290
<u>-rm {<memory> [<parameter_name>]}</u>	293
<u>Examples</u>	293
<u>memWritethru</u>	295
<u>-add {<memory_instance_name> ON OFF}</u>	295
<u>-rm <memory_instance_name></u>	295
<u>-clear</u>	295
<u>moduleFV</u>	296
<u>-add [<LIB> <cellName> enable disable]</u>	296
<u>-rm [<LIB> <cellName>]</u>	296
<u>moduleStub</u>	297
<u>-add {<LIB> <cellName>}</u>	297
<u>-rm {<LIB> <cellName>}</u>	297
<u>multiplexCable</u>	298
<u>-add <cable_label></u>	298
<u>multiSampledTerminal</u>	300
<u>-add <terminal_name></u>	300
<u>-rm <terminal_name></u>	300
<u>netWeakDrive</u>	301
<u>-add</u>	301
<u><name></u>	301
<u><weakDriveValue></u>	301
<u>-rm</u>	302
<u>-clear</u>	302
<u>noBreakPath</u>	303
<u>-add {<startnet> <endnet>}</u>	303
<u>-rm {<startnet> <endnet>}</u>	303
<u>noOverlapClocks</u>	304
<u>-add {<expression>... } ...</u>	304
<u>-rm {<expression> ... } ...</u>	304

UXE Command Reference Manual

<u>Examples</u>	304
<u>orMultiDrv</u>	305
<u>-add {<signal>}</u>	305
<u>-rm {<signal>}</u>	305
<u>Examples</u>	305
<u>precompileOption</u>	306
<u>-add pullUpOnly</u>	308
<u>-add pullUp</u>	308
<u>-add pullDown</u>	308
<u>-add primaryIOPullup</u>	309
<u>-add primaryIOPulldown</u>	309
<u>-add primaryIOExternalDrive</u>	309
<u>-add retainState</u>	309
<u>-add noTimeCheck</u>	309
<u>-add full</u>	309
<u>-add tieSourceless</u>	309
<u>-add [tieSourcelessRandom tieSourcelessRandom <seed>]</u>	310
<u>-add tieSourcelessUnsafe</u>	310
<u>-add tieSourcelessHigh</u>	310
<u>-add tieSourcelessLow</u>	310
<u>-add removeSourceless</u>	310
<u>-add keepSourceless</u>	311
<u>-add {removeLoadless <N>}</u>	311
<u>-add failOnMultiDrv</u>	311
<u>-add failOnUnresolvedOOMR</u>	312
<u>-add dontKeepMem</u>	312
<u>-add keepAllMem</u>	312
<u>-add keepMem</u>	312
<u>-add syncData</u>	313
<u>-add asyncReset</u>	313
<u>-add syncReset</u>	313
<u>-add {cdc <CDC_file_name>}</u>	313
<u>-add debug</u>	313
<u>-add allowUnresolvedOOMR</u>	314
<u>-add dontLogOOMR</u>	314
<u>-add logAllOOMR</u>	314

UXE Command Reference Manual

<u>-add ignoreEmptyCells</u>	314
<u>-add keepEmptyCells</u>	315
<u>-add keepStubInputs</u>	315
<u>-add keepStubOutputs</u>	315
<u>-add ignorePowerDrvConflict</u>	315
<u>-add ignoreMultiDrv</u>	315
<u>-add disconnectTristateDrivers</u>	316
<u>-add andMultiDrv</u>	316
<u>-add orMultiDrv</u>	316
<u>-add dumpShadows</u>	317
<u>-add noLoopBreaks</u>	317
<u>-add nolsolationBreaks</u>	317
<u>-add keepBreaks</u>	317
<u>-add {loopsDepth <number>}</u>	318
<u>-add dontMinimizeDataBreaks</u>	318
<u>-add minimizeBreaks</u>	319
<u>-add keepState</u>	319
<u>-add {logAllLoops <number>}</u>	319
<u>-add moreExclusive</u>	319
<u>-add clockDataLoops</u>	320
<u>-add noReset1X</u>	320
<u>-add breakHalfCycleAll</u>	320
<u>-add breakHalfCycle</u>	320
<u>-add setPassiveShadows</u>	320
<u>-add recognizeClockDivider</u>	320
<u>-add cake_like_1x</u>	321
<u>-add cake_like_2x</u>	321
<u>-add moreClockSources</u>	321
<u>-add allowDelayClkNets</u>	321
<u>-add 1xLessClockPaths</u>	321
<u>-add verify</u>	322
<u>-add latchResetAsFF</u>	322
<u>-add dontMinimizeBreaks</u>	322
<u>-add clockLoops30</u>	322
<u>-add clockLoops101</u>	322
<u>-add combinLoops101</u>	323

UXE Command Reference Manual

<u>add dataLoops101</u>	323
<u>add resetLoops101</u>	323
<u>add {duplicateLoops <number>}</u>	323
<u>add safeLoops</u>	323
<u>add {seed <number>}</u>	325
<u>add randomizeMemories</u>	325
<u>add dontRandomizeMemories</u>	325
<u>add readTerminalAssign</u>	325
<u>add instrumentKeepNets</u>	326
<u>add noPowerIntent</u>	327
<u>add domainInterfaceDef [1.0 2.0 2.1 UPF 1801_2009 1801_2013]</u>	327
<u>add allowNoBreakViolations</u>	327
<u>add VSSwitchable</u>	327
<u>add breakTerminalTiming</u>	328
<u>add lessProbes</u>	328
<u>add allowClockConflicts</u>	328
<u>add instrumentEdges</u>	329
<u>add createAllBoundaries</u>	329
<u>add dontMergeCommon</u>	329
<u>add {1XrestrictPath <path>}</u>	329
<u>How to investigate precompile optimization</u>	330
<u>Clocking Behavior</u>	333
<u>Behavior of Latches and flip-flops</u>	333
<u>Options Affecting Loop-breaking</u>	333
<u>primaryClockSource</u>	335
<u>-add <primary_clock_source_name></u>	335
<u>-rm <primary_clock_source_name></u>	335
<u>probe</u>	336
<u>-add <signal></u>	337
<u>-add <net_name></u>	337
<u>-addinst <instance_name> [-iregexp -iglob <inst_pattern>] [-depth <n>]</u>	337
<u>-addio <instance_name></u>	338
<u>-addcone [-depth <n>] <signal></u>	338
<u>-addpath [-depth <n>] <net1> <net2></u>	338
<u>-glob -regexp [<pattern>]</u>	338
<u>-addNreport <net_name></u>	339

UXE Command Reference Manual

<u>clear</u>	339
<u>-rm <signal>... <net_name> ...</u>	339
<u>-rm {-glob -regexp <pattern>}</u>	339
<u>Examples</u>	339
<u>shadowsData</u>	340
<u>-add {<net> <keyword>}</u>	340
<u>-rm <net></u>	340
<u>stateRetentionControl</u>	342
<u>-add {<stateRetentionRuleName> low high}</u>	342
<u>-rm {<stateRetentionRuleName> low high}</u>	342
<u>symmetricConfiguration</u>	343
<u>-add {host <host>} [{boards <boards>}]</u>	343
<u>-add {file <file>} [{boards <boards>}]</u>	343
<u>-rm {host <host>}</u>	344
<u>-rm {file <file>}</u>	344
<u>-clear</u>	344
<u>terminalAssign</u>	345
<u>-add {<interface_name> [<interface_type>] <pin> <terminal_name>}</u>	345
<u><interface_type></u>	346
<u><connector></u>	346
<u><pin></u>	347
<u>TERM NET</u>	348
<u><terminal_name></u>	348
<u>-rm {<interface_name> [<interface_type>] <pin>}</u>	348
<u>-dump <filename></u>	348
<u>-clear</u>	348
<u>Examples</u>	349
<u>terminalOutputSync</u>	350
<u>-add {<terminal> <group_number>}</u>	350
<u><terminal></u>	350
<u><group_number></u>	350
<u>-clear</u>	350
<u>Examples</u>	351
<u>terminalTiming</u>	352
<u>-add {<terminal_name> <constraint>}</u>	352
<u>-add {<terminal_name> <constraint_out> <constraint_in> <constraint_en>}</u>	353

UXE Command Reference Manual

<u>-add {TARGET_TRIGGER SIGNAL <constraint>}</u>	353
<u>-add {TARGET_SUSPEND SIGNAL <constraint>}</u>	353
<u>-add {ALL_OTHER_OUTPUTS <constraint>}</u>	353
<u>-add {<terminal_name> <constraint> -pulse <pulse_info>}</u>	353
<u>-add {<terminal_name> -edgeInfo <initial_value> <hold_time> <edge_list>}</u>	354
<u>-rm {<terminal_name>}</u>	354
<u>{<terminal_name>}</u>	355
<u>-dump <file_name></u>	355
<u>Examples</u>	355
<u>Design with clock outputs (1)</u>	355
<u>Design with clock outputs (2)</u>	356
<u>Pulse outputs</u>	356
<u>Multipulse output</u>	356
<u>Avoiding output-to-input delay</u>	357
<u>Avoiding output-to-input delay for bidirectional signals</u>	358
<u>Design with a clock input asynchronous with FCLK</u>	359
<u>terminalWeakDrive</u>	362
<u>-add {<terminal_name> <weakDriveValue>}</u>	362
<u><terminal_name></u>	362
<u><weakDriveValue></u>	362
<u>-rm {<terminal_name>}</u>	363
<u>-clear</u>	363
<u>tieNet</u>	364
<u>-add {<net> 0 1}</u>	364
<u>-addNreport <net></u>	364
<u>-rm {<net>}</u>	364
<u>Examples</u>	364
<u>ttcClear</u>	365
<u>userData</u>	366
<u>-load <file></u>	366
<u>-scope <instance_path></u>	366
<u>-load -noQualify <file></u>	367
<u>-dump [<type> all] <file></u>	368
<u>Examples</u>	368
<u>-get [-max <n>] [<type> [option]]</u>	369

9

<u>IP Commands</u>	371
<u>caCell</u>	372
<u>-add {<lib> <cell>}</u>	372
<u>-rm {<lib> <cell>}</u>	372
<u>-add {<lib> <cell-name> <IP-ID>}</u>	372
<u>Examples</u>	372
<u>ipbAssign</u>	374
<u>-add {<IP_label> <IPP_number>}</u>	374
<u>-rm {<IP_label> [<IPP_number>]}</u>	374
<u>Examples</u>	374
<u>iplInstAssign</u>	375
<u>-add <inst_name> <IP_ID></u>	375
<u>[<IP_label>]</u>	375
<u>[<IP_cell_ID>]</u>	375
<u>Examples</u>	375

10

<u>Commands and System Tasks for SA Mode</u>	377
<u>vhan</u>	378
<u>-32 -64</u>	380
<u>-just or -cIC</u>	381
<u>-skip</u>	383
<u>-its</u>	384
<u>-q Q</u>	385
<u>-work -w</u>	386
<u>+civb</u>	387
<u>-hlH[elp]</u>	388
<u>-fl-F</u>	389
<u>-87l-93l-200x</u>	390
<u>-200x</u>	391
<u>-verbose</u>	393
<u>-slS</u>	394
<u>-libmaprc</u>	395

UXE Command Reference Manual

<u>L</u>	396
<u>version</u>	397
<u>I</u>	398
<u>mti</u>	399
<u>ncsim</u>	400
<u>nocasestaticerror</u>	401
<u>psl</u>	402
<u>assert</u>	403
<u>assert_vlog</u>	404
<u>assert_vhdl</u>	405
<u>noassert</u>	406
<u>profile</u>	407
<u>append_log</u>	408
<u>ignore_pragma</u>	409
<u>+rtlCommentPragma</u>	410
<u>vlan</u>	411
<u>h</u>	414
<u>work</u>	415
<u>fIF</u>	416
<u>+v2000</u>	418
<u>libmaprc</u>	419
<u>I</u>	420
<u>q</u>	421
<u>u</u>	422
<u>v</u>	423
<u>W</u>	424
<u>w</u>	425
<u>y</u>	426
<u>+define</u>	427
<u>+incdir</u>	428
<u>+libext</u>	429
<u>+liborder</u>	430
<u>+librescan</u>	431
<u>+libverbose</u>	432
<u>+convertUdp</u>	433
<u>+noHdliceCompatUdpMapping</u>	434

UXE Command Reference Manual

<u>-Z</u>	435
<u>+max_error_count</u>	436
<u>+maxdelays</u>	437
<u>+mindelays</u>	438
<u>+nowarn</u>	439
<u>-vtimescale</u>	440
<u>-netlist</u>	441
<u>+protect</u>	442
<u>-P</u>	443
<u>-p</u>	444
<u>+rtlIgnTask</u>	445
<u>+tfReadOnly</u>	446
<u>+primchange</u>	447
<u>+loadpli1</u>	448
<u>+loadvpi</u>	449
<u>+specblk</u>	450
<u>-its</u>	451
<u>-ncsim</u>	452
<u>+timescale</u>	453
<u>-default_ext</u>	454
<u>+showSvPkgAccess</u>	455
<u>-sysv_ext</u>	456
<u>-vlog95_ext</u>	457
<u>-vlog_ext</u>	458
<u>-vlogext</u>	459
<u>ixclkgen</u>	460
<u>-input <file></u>	461
<u>-output <file></u>	461
<u>-controlled <clock_sources></u>	461
<u>-default controlled uncontrolled</u>	461
<u>-help</u>	461
<u>-hierarchy <string></u>	461
<u>-use_ixc_clkgen</u>	461
<u>-module <name></u>	462
<u>-rtc</u>	462
<u>-uncontrolled <clock_sources></u>	462

UXE Command Reference Manual

<u>-timescale <arg></u>	462
<u>ixcom</u>	464
<u>-clean</u>	468
<u>-incdir</u>	469
<u>-relativeXCDIR</u>	470
<u>-m</u>	471
<u>-o</u>	472
<u>+delVhdlEntity</u>	473
<u>+no_strnInst</u>	474
<u>-top</u>	475
<u>-L</u>	477
<u>+bc</u>	478
<u>+bcc</u>	479
<u>+bcAsyncEvent</u>	480
<u>+bcDebug</u>	481
<u>-32 -ixcg32 -64</u>	482
<u>+asis</u>	483
<u>-relax</u>	484
<u>-featureFile</u>	485
<u>+tbcnba</u>	487
<u>+dpiprofsamplecnt</u>	488
<u>-externalNetlists</u>	489
<u>-ignoreInitialFinal</u>	493
<u>-keepInitialFinal</u>	495
<u>-conformal</u>	497
<u>+margdpi</u>	498
<u>+margdpisize</u>	499
<u>+margdirect</u>	500
<u>+xmrTfInlineOn</u>	501
<u>+xmrTfInlineOff</u>	502
<u>+mapInit</u>	503
<u>+no_mapInit</u>	504
<u>+moduleStub</u>	505
<u>+genIfcSideFile</u>	506
<u>-lint</u>	507
<u>+maxPIPO</u>	508

UXE Command Reference Manual

<u>+ignoreNCVerCheck</u>	509
<u>+uvmDut+<arg></u>	510
<u>-incLibFileCompilationUnit</u>	511
<u>-generic</u>	512
<u>-lower</u>	513
<u>+upper</u>	514
<u>+pd3legacy</u>	515
<u>-append_log</u>	516
<u>-h[elp]</u>	517
<u>-work</u>	518
<u>-rtlchk</u>	519
<u>-enableEmbeddedSW</u>	520
<u>+no_rtlchk</u>	521
<u>-ncsim</u>	522
<u>-nccompile</u>	523
<u>-covdut</u>	524
<u>-covfile</u>	525
<u>+covGenFeatureFile</u>	526
<u>+ccovModelSummary</u>	528
<u>+ccovProfile</u>	529
<u>-covAddResetLogic</u>	530
<u>+fcovCoverCnt+<N></u>	531
<u>-fl-F</u>	532
<u>+bcov[+<design_unit_or_file_name>]</u>	534
<u>+bcovExclude+<design_unit_or_file_name></u>	535
<u>+tcov[+<design_unit_or_file_name>]</u>	536
<u>+tcovExclude+<design_unit_or_file_name></u>	537
<u>+tcovTypes+<type></u>	538
<u>+tcovMDA[+<exponent>]</u>	539
<u>-l</u>	540
<u>+delay</u>	541
<u>+rtllgnBlkDelay</u>	542
<u>+salgnDanglingClock</u>	543
<u>+max_error_count</u>	544
<u>+nowarn</u>	545
<u>-msgSummary</u>	546

UXE Command Reference Manual

<u>+no_tran</u>	547
<u>+tran_relax</u>	548
<u>-P</u>	549
<u>-p</u>	550
<u>+tfReadOnly</u>	551
<u>-ua</u>	552
<u>+1xua</u>	553
<u>+dut</u>	554
<u>+dutignore</u>	555
<u>+salgnXmrWrite</u>	556
<u>+rtlIgnPragma</u>	557
<u>+showPragma</u>	558
<u>+rtlKeepDelays</u>	560
<u>+rtlmemsize</u>	561
<u>+rtlvarwidth</u>	563
<u>-q</u>	564
<u>-u</u>	565
<u>-vaelabargs</u>	566
<u>-vavhdlargs</u>	567
<u>-vavlogargs</u>	568
<u>-ncvlogargs</u>	569
<u>-ncvhdlargs</u>	570
<u>-ncelabargs</u>	571
<u>-loadsc</u>	572
<u>+systemc_args</u>	573
<u>+tfconfig</u>	574
<u>+gfifo_lbsize+size</u>	575
<u>+gfifoDisp</u>	576
<u>+staticFd</u>	577
<u>-rtlNameForGenerate</u>	578
<u>-xeCompile</u>	579
<u>-xecompileargs</u>	581
<u>-verbose</u>	582
<u>-V</u>	583
<u>-Z</u>	584
<u>-y</u>	585

UXE Command Reference Manual

<u>+define</u>	586
<u>+incdir</u>	587
<u>+noProcessAttrib</u>	588
<u>+libext</u>	589
<u>+libborder</u>	590
<u>+librescan</u>	591
<u>+libverbose</u>	592
<u>-Z</u>	593
<u>+maxdelays</u>	594
<u>+mindelays</u>	595
<u>+rtlHonorCasePragma</u>	596
<u>+protect</u>	598
<u>+rtllgnTask</u>	599
<u>+loadpli</u>	600
<u>+loadvpi</u>	601
<u>+vhdlmcx</u>	602
<u>+enableSvPkg</u>	603
<u>+hm</u>	604
<u>+hmpi</u>	605
<u>+hwdp</u>	606
<u>+iscdisp</u>	607
<u>+margtbc</u>	608
<u>+newhm</u>	609
<u>+saSimpleLatch</u>	610
<u>+saVerbose</u>	611
<u>+synloopsize</u>	612
<u>-target</u>	613
<u>+targetTop</u>	615
<u>+tb_export</u>	616
<u>+tb_import</u>	617
<u>+tb_import systf</u>	618
<u>-vhdl_time_precision</u>	619
<u>+xcDesignTop</u>	620
<u>+dump_bidir</u>	621
<u>+xcbidir</u>	622
<u>-trans_rtl</u>	623

UXE Command Reference Manual

<u>+prefixXref</u>	624
<u>+xcEmbeddedTb</u>	625
<u>-ixclkgen</u>	626
<u>+hwRandom</u>	627
<u>-localDiskComp</u>	628
<u>+localDiskComp</u>	629
<u>+bufdisp</u>	630
<u>+iscDelay</u>	631
<u>-timescale</u>	632
<u>+hwfrc</u>	633
<u>+sv</u>	634
<u>+v1995</u>	635
<u>+v2000</u>	636
<u>-strictLRM</u>	637
<u>+rtlCommentPragma</u>	638
<u>synth_prefix</u>	640
<u>-rm_synth_prefix</u>	641
<u>-defparam</u>	642
<u>-ixerror</u>	643
<u>-ixfatal</u>	644
<u>Synthesis Policy Checker Options</u>	645
<u>+rtlKeepDelays</u>	646
<u>+rtlmemsize</u>	647
<u>+rtlvarwidth</u>	648
<u>+no_rtlxmr</u>	649
<u>System Tasks and Procedures</u>	650
<u>Supported Verilog System Tasks & VHDL Procedures</u>	651
<u>\$xc_cmd();</u>	652
<u>\$export_deposit();</u>	653
<u>\$export_event();</u>	654
<u>\$initmem();</u>	655
<u>\$export_frcrel(); export_frcrel();</u>	656
<u>\$export_read(); export_read();</u>	658
<u>Supported IEEE-Standard Non-CLI System Tasks</u>	660
<u>Supported IEEE-Standard Compiler Directives</u>	661
<u>Verilog Primitives and VHDL Procedures</u>	662

UXE Command Reference Manual

<u>ixc_pulse(out, in):</u>	663
<u>axis_tbcall():</u>	664
<u>SVA Options to IXCOM</u>	667
<u>+sva+</u>	668
<u>+no_sva</u>	669
<u>+assertCover</u>	670
<u>+assertCoverCnt</u>	671
<u>+assertCheckedCntr+<N></u>	672
<u>+assertFinishedCntr+<N></u>	673
<u>+assertFailedCntr+<N></u>	674
<u>+assertStatus</u>	675
<u>+assertFailCounter+hw</u>	676
<u>-coverage [block toggle functional all]</u>	677
<u>-covMakeNamesUnique</u>	678
<u>+fcov_uxe_db</u>	679
<u>PSL Options to IXCOM</u>	680
<u>+no_psl</u>	681

11

<u>Compile-time Commands in ICE Mode</u>	683
<u>checkCPFfile</u>	684
<u><cpfFileName></u>	684
<u>check_power_intent</u>	685
<u><1801Filename></u>	685
<u>-version</u>	685
<u>compile</u>	686
<u>-max_steps <number></u>	686
<u>-min_steps <number></u>	686
<u>-compactDCC</u>	687
<u>-crit_paths <number></u>	687
<u>-criticalPathOnly [<number>]</u>	687
<u>-names_crit</u>	687
<u>-seed random last lastpass <number></u>	688
<u>-etOnly</u>	688
<u>Examples</u>	688

UXE Command Reference Manual

<u>compileFind</u>	689
<u>-all</u>	690
<u>-best</u>	690
<u>-first</u>	691
<u>-lsf</u>	691
<u>-mem</u>	692
<u>-normal</u>	693
<u>-opt</u>	693
<u>-max_trials <number></u>	694
<u>-max_trials each <number></u>	694
<u>-max_steps <number></u>	694
<u>-min_steps <number></u>	694
<u>-seed <number></u>	695
<u>-logfile <logfile></u>	695
<u>-compactDCC</u>	695
<u>-crit_paths <number></u>	696
<u>-names_crit</u>	696
<u>-max_util <number></u>	696
<u>-min_util <number></u>	696
<u>-dec_util <number></u>	697
<u>-oE {<values>}</u>	697
<u>-pE {<values>}</u>	697
<u>-etOnly</u>	697
<u>-et3pass2</u>	697
<u>-no_export</u>	698
<u>-1pass</u>	698
<u>-2pass</u>	698
<u>-keep_all</u>	698
<u>-rerun_best</u>	699
<u>-verify</u>	699
<u>-timeout <number></u>	699
<u>-link <filename></u>	699
<u>-file <file></u>	700
<u>Examples</u>	702
<u>design</u>	705
<u><lib></u>	705

UXE Command Reference Manual

<u><cell></u>	705
<u>-close</u>	705
<u>-rm</u>	705
<u>Examples</u>	706
<u>disconnectDriverPin</u>	707
<u>-add <hier_driver_pin></u>	707
<u>-rm <hier_driver_pin></u>	707
<u>-clear</u>	707
<u>getCriticalPathDesignNetNames</u>	708
<u>icePrepare</u>	709
<u>precompile</u>	710
<u>Examples</u>	710
<u>readCPFfile</u>	712
<u><cpfFileName></u>	712
<u>-checkSyntaxOnly</u>	713
<u>-checkOnly</u>	713
<u>-noLinter</u>	713
<u>exitOnError</u>	713
<u>-clear</u>	713
<u>-help</u>	713
<u>read_power_intent</u>	714
<u><1801Filename></u>	714
<u>-force</u>	714
<u>-version</u>	714
<u>-defaultAppliesToOutputs [1.0 2.0 2.1 UPF 1801_2009 1801_2013]</u>	715
<u>clean_power_intent</u>	716
<u>referenceMap</u>	717
<u>shrinkDB</u>	718

12

<u>Common Power Format Commands</u>	719
<u>assert_illegal_domain_configurations</u>	721
<u>-name <string></u>	721
<u>-domain_conditions <domain_condition_list></u>	721
<u>create_isolation_rule</u>	722

UXE Command Reference Manual

<u>-name <string></u>	722
<u>-isolation_condition <expression> -no_condition</u>	723
<u>{-pins <pin_list> -from <power_domain_list> -to <power_domain_list>}...</u>	723
<u>-exclude <pin_list></u>	724
<u>-isolation_output {high low hold clamp_high clamp_low}</u>	725
<u>-secondary_domain <power_domain></u>	725
<u>Related Commands</u>	725
<u>create_mode</u>	726
<u>-name <string></u>	726
<u>-condition <expression></u>	726
<u>-illegal</u>	726
<u>Related Commands</u>	726
<u>create_mode_transition</u>	727
<u>-name <string></u>	727
<u>-from <power_mode></u>	727
<u>-to <power mode></u>	727
<u>create_nominal_condition</u>	728
<u>-name <string></u>	728
<u>-voltage {<voltage> <voltage_list>}</u>	728
<u>-ground_voltage {<voltage> <voltage_list>}</u>	728
<u>-state {on off standby}</u>	728
<u>Example</u>	728
<u>Related Commands</u>	729
<u>create_power_domain</u>	730
<u>-name <power_domain></u>	730
<u>-instances <instance_list></u>	730
<u>-boundary_ports <pin_list></u>	731
<u>-default</u>	732
<u>-shutoff_condition <expression></u>	732
<u>-external_controlled_shutoff</u>	732
<u>-default_isolation_condition <expression></u>	732
<u>-default_restore_edge <expression></u>	732
<u>-default_save_edge <expression></u>	732
<u>-power_down_states {high low random inverted}</u>	733
<u>-power_up_states {high low random inverted}</u>	733
<u>-active_state_conditions <active_state_condition_list></u>	734

UXE Command Reference Manual

<u>base_domains <domain_list></u>	735
<u>Examples</u>	735
<u>Related Commands</u>	735
<u>create power mode</u>	737
<u>-name <string></u>	737
<u>-domain_conditions <domain_condition_list></u>	737
<u>-default</u>	738
<u>Related Commands</u>	738
<u>Examples</u>	738
<u>create state retention rule</u>	740
<u>-name <string></u>	741
<u>-domain <power_domain></u>	741
<u>-instances <instance_list></u>	741
<u>-exclude <instance_list></u>	742
<u>-restore_edge <expression></u>	742
<u>-save_edge <expression></u>	742
<u>-restore_precondition <expression></u>	743
<u>-save_precondition <expression></u>	743
<u>-secondary_domain <domain></u>	743
<u>Related Commands</u>	744
<u>Example</u>	744
<u>end design</u>	745
<u><power_design_name></u>	745
<u>Related Commands</u>	745
<u>end macro model</u>	746
<u><macro_cell_name></u>	746
<u><macro_model_name></u>	746
<u>Related Commands</u>	746
<u>find_design_objects</u>	747
<u><pattern></u>	747
<u>-pattern_type {name cell module}</u>	747
<u>-scope <hierarchical_instance_list></u>	748
<u>-object {inst port pin net}</u>	748
<u>-direction {in out inout}</u>	748
<u>-leaf_only -non_leaf_only</u>	748
<u>-hierarchical</u>	749

UXE Command Reference Manual

<u>exact</u>	749
<u>regexp</u>	749
<u>ignore case</u>	749
<u>get parameter</u>	750
<u><parameter name></u>	750
<u>include</u>	751
<u><CPF filename></u>	751
<u>set cpf version</u>	752
<u><value></u>	752
<u>set design</u>	753
<u>module <module list></u>	753
<u>ports <port list></u>	753
<u>input_ports <port list></u>	754
<u>output_ports <port list></u>	754
<u>inout_ports <port list></u>	754
<u>parameters <parameter value list></u>	754
<u>testbench</u>	754
<u>Related Information</u>	755
<u>set hierarchy separator</u>	756
<u><character></u>	756
<u>set instance</u>	757
<u><instance></u>	757
<u>design <power design name></u>	757
<u>model <macro cell></u>	758
<u>port mapping <port mapping list></u>	758
<u>domain_mapping <domain_mapping list></u>	759
<u>parameter_mapping <parameter_mapping list></u>	759
<u>Related Commands</u>	759
<u>set macro model</u>	760
<u><macro model name></u>	760
<u>cells <cell list></u>	761
<u>Related Commands</u>	761

13

<u>Run-time Commands</u>	763
<u>Run-time Commands Set</u>	763
<u>assertion</u>	780
<u>-check</u>	781
<u>-counter <counter_list> <assertion.pathname></u>	782
<u>-get</u>	782
<u>-off</u>	783
<u>-list</u>	785
<u>-on</u>	786
<u>-summary</u>	786
<u>Common Command Options</u>	788
<u>bm</u>	792
<u>-add <bookmark_name> [<time specification>] [-replace]</u>	792
<u>-list [-glob -regexp <pattern> [-nameOnly]]</u>	793
<u>-rm *</u>	793
<u>-rm <bookmark_name></u>	793
<u>-save <file_name></u>	793
<u>-import <file_name> [-replace]</u>	793
<u><bookmark_name></u>	793
<u>clockConfig</u>	794
<u>-sample <value></u>	795
<u>-dly <clockname> <value></u>	795
<u>-clk <clockname> <value> [<unit>]</u>	795
<u>-list</u>	796
<u>-restore <filename></u>	796
<u>-save <filename></u>	796
<u>-actual</u>	796
<u>-frequency [<value> [<unit>]]</u>	797
<u>-realclk <clockname> [<value> [<unit>]]</u>	797
<u>-stopMode [controlled *controlled both none *none]</u>	798
<u>Examples</u>	799
<u>configPM</u>	801
<u>-la</u>	802
<u>-stb</u>	802

UXE Command Reference Manual

<u>staticTarget</u>	802
<u>dynamicTarget</u>	802
<u>-vd</u>	803
<u>Examples</u>	803
<u>convertRadix</u>	804
<u><fromRadix></u>	804
<u><toRadix></u>	804
<u><value></u>	804
<u>Examples</u>	805
<u>database</u>	806
<u>[-open] <database_name></u>	806
<u>-event</u>	807
<u>-gzip</u>	807
<u>continuousupload</u>	807
<u>-phy</u>	808
<u>-close <database_name></u>	808
<u>-show [<database_name>]</u>	809
<u>-upload</u>	809
<u>-clear</u>	809
<u>-tracewindow -start <time> -end <time></u>	809
<u>-listphyfile [<phy_fileName> ...]</u>	810
<u>-prepareOffline</u>	810
<u>-disable <database_name></u>	810
<u>-enable <database_name></u>	810
<u>-captureMode [controlled uncontrolled]</u>	811
<u>debug</u>	813
<u><dbPath></u>	813
<u>-session <session_name></u>	814
<u>-close</u>	814
<u>deposit</u>	815
<u><name></u>	816
<u><value></u>	816
<u>-release</u>	817
<u>-allff {0 1 random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]</u>	817
<u>-file <textfile.dat></u>	818

UXE Command Reference Manual

<u>download</u>	819
[-skiphddc all -skiphddc <cable_logical_name>...]	819
<u>Examples</u>	819
<u>dpa</u>	820
<u>-phyfile <phy_filename></u>	822
<u>-listphyfile [<phy_fileName> ...]</u>	822
<u>-tracewindow [-start <time> {-end <time>} -size <time>]}</u>	823
<u>-setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>] <instance_names></u>	824
<u>-setupToggleCount -listinst [-verbose]</u>	824
<u>-setupToggleCount -rminst <instance_names></u>	824
<u>-setupToggleCount -rminst -all</u>	824
<u>-getToggleCount [-append] [-weighted] [-file <name>]</u>	825
<u>-list [-max <n>]</u>	825
<u>-addinst [-excludelibrary <.lib names>] [-allnets] -depth <n> <instance> ...</u>	825
<u>-addinstoutputpin <instance> [-gatecell <gatecell_filename>]</u>	825
<u>-addinstpin <instance></u>	826
<u>-addcone -depth <n> <signal> ...</u>	826
<u>-addpath -depth <n> {<net1> <net2>}</u>	826
<u>-readNetNames <rc dpa info file name> [<instanceName>]</u>	826
<u>-rm <signal>...</u>	826
<u>-rm [-regexp -glob -pattern <pattern>]*</u>	826
<u>-outfile -tcf <filename> [-instance <instanceName>] [-masterTCF]</u>	
<u>[-slaveTCF] [-noregularTCF] [-segment <n>% <cycleNumber>]</u>	
<u>[{-start <time> -end <time>}]</u>	827
<u>-upload</u>	829
<u>-saif -load <saif_file_name> [-module <module_name_pattern>]</u>	829
<u>-saif -listmodule [-lib]</u>	829
<u>-saif -listlib</u>	829
<u>-saif -listfile</u>	829
<u>-saif -rmmodule <module_name_pattern> [<lib_name>]</u>	829
<u>-saif -listduplicatemodule</u>	829
<u>-saif -rmlib <lib_name></u>	830
<u>-saif -rmfile <file_name></u>	830
<u>-saif -addscope <instance_name></u>	830
<u>-saif -rmscope <instance_name></u>	830
<u>-saif -listscope</u>	830

UXE Command Reference Manual

<u>-outfile -saif <saif file name> [-instance <instanceName>]</u>	830
<u>[-segment <cycleNumber> <percent>] [{-start <time> -end <time>}]</u>	830
<u>-genPowerProfile [-refine]</u>	831
<u>-setupRC -show</u>	831
<u>-setupRC -reset</u>	832
<u>-setupRC { -gateFlow -rtlFlow }</u>	832
<u>-setupRC -servers { <machine list> }</u>	832
<u>-setupRC -libPath { <path name list> }</u>	832
<u>-setupRC -libList { <lib name list> }</u>	832
<u>-setupRC -designPath { <path name list> }</u>	833
<u>-setupRC -designFile { <file name list> }</u>	833
<u>-setupRC -topDesign <top module name></u>	833
<u>-setupRC -instance <instance name></u>	833
<u>-setupRC -depth <N></u>	834
<u>-setupPowerProfile -show</u>	834
<u>-setupPowerProfile -reset</u>	834
<u>-setupPowerProfile -database <name></u>	834
<u>-setupPowerProfile -tcfDir <name></u>	834
<u>-setupPowerProfile -segment {<n>% <cycleNumber>}</u>	835
<u>-sparse sampling [<N>]</u>	835
<u>drtl</u>	836
<u>-definemodule <DRTL module name> [-verilog -vhdl -sv] <file list> [-topmodule <topmodule name>]</u>	836
<u>-addinst [-overwrite] <DRTL module name> <DRTL instance name> (<port name1>(<signal name1>), <port name2>(<signal name2>),...)</u>	837
<u>-addinst [-overwrite] <DRTL module name> <DRTL instance name> (<signal name1>, <signal name2>,...)</u>	838
<u>-list [-code] [DRTL module name pattern]</u>	838
<u>-listinst [DRTL instance name pattern] [-modulename <DRTL module name>]</u>	838
<u>-rm <DRTL module name pattern></u>	838
<u>-rminst <DRTL instance name pattern> [-modulename <DRTL module name>]</u>	839
<u>-getallinst [DRTL instance name pattern] [-modulename <DRTL module name>]</u>	839
<u>-getallmodule [DRTL module name pattern]</u>	839
<u>-enableSysTask -disableSysTask <DRTL instance name pattern></u>	839
<u>-systemTasksDefault [enable disable 0 1]</u>	840
<u>-save [-overwrite] <file name> [DRTL module name pattern list]</u>	840

UXE Command Reference Manual

<u>-load [-overwrite -ignore] <file name> [DRTL module name pattern list]</u>	840
<u>force</u>	842
<u><name></u>	843
<u><value></u>	843
<u>-allff {0 1 random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]</u>	843
<u>-file -prep <textfile></u>	844
<u>-file <textfile.dat></u>	845
<u>genPowerProfile</u>	846
<u>-databaseName <db name></u>	846
<u>-honorFrequency</u>	846
<u>-honorTimeScale</u>	846
<u>-reportFileNamePattern <fileNamePattern></u>	847
<u>-reportFiles <powerReportFileName></u>	847
<u>-powerAwareInfo <powerAwareInfoFile></u>	847
<u>-segmentLogFile <segment log file from TCF generation></u>	848
<u>-tcfLogFiles <logFileName></u>	848
<u>-help</u>	848
<u>getCableStatus</u>	849
<u>-detail</u>	849
<u>-list</u>	850
<u>Examples</u>	852
<u>getCycleNum</u>	854
<u>Examples</u>	854
<u>getHostStatus</u>	855
<u>Examples</u>	855
<u>getPMStatus</u>	856
<u>Examples</u>	857
<u>getTime</u>	859
<u>-simulator</u>	859
<u><timeUnit></u>	859
<u>Examples</u>	859
<u>hdlConfig</u>	861
<u>hdlConfig for Initializing the System Tasks</u>	861
<u><file></u>	861
<u>-cycleTime=<cycle_time></u>	862

UXE Command Reference Manual

<u>top=<instance></u>	862
<u>-displayDebug</u>	862
<u>+<plusargs></u>	862
<u>Example 1</u>	862
<u>hdlConfig for Enabling/Disabling the Compile Options</u>	863
<u>-disable {<insPath> -top -all}</u>	863
<u>-enable {<insPath> -top -all}</u>	863
<u>-disableBuffer</u>	864
<u>-enableBuffer</u>	864
<u>-timeOut <cycleNum></u>	864
<u>hdlRunLogfile</u>	865
<u><file></u>	865
<u>[-append]</u>	865
<u>host</u>	866
<u><hostname> [-path <design_path>] [-reserveFirst -noReserve]</u>	866
<u>-timeout <timeoutValue></u>	867
<u>-key <reservationKey></u>	868
<u>-keyfile <reservationFilePath></u>	868
<u>-firstFit</u>	868
<u>-keepAlive</u>	868
<u>-testHwFirst</u>	869
<u>-dynp</u>	869
<u>-check <hostname></u>	869
<u>-checkHDVer <hostname></u>	870
<u>-checkSuspendStatus</u>	870
<u>-kill</u>	870
<u>-location</u>	870
<u>-offline {<data_file_name> <*.phy_file_name>}</u>	870
<u>-release</u>	870
<u>-enableTIF</u>	871
<u>-disableTIF</u>	871
<u>-statusTIF</u>	871
<u>Examples</u>	871
<u>infiniTrace</u>	872
<u>-prepare <session_name> [-append]</u>	873
<u>-prepare <session_name> [-overwrite]</u>	874

UXE Command Reference Manual

<u>-on -off</u>	874
<u>-rm <name></u>	874
<u>-observe <name></u>	874
<u>-goto [-continue] <time specification></u>	874
<u>-stateSaveFrequency <value></u>	876
<u>-getStateSavePoints</u>	877
<u>-getCurrentMode</u>	877
<u>-getCurrentMode [-text]</u>	877
<u>-getSessionName</u>	877
<u>-isCaptureEnabled</u>	878
<u>-bookmark -add <book_mark_name></u>	878
<u>-bookmark -rm <book_mark_name></u>	878
<u>-bookmark -list</u>	878
<u>-bookmark -goto <book_mark_name></u>	878
<u>-close</u>	878
<u>-close [-force]</u>	879
<u>-list [-info]</u>	879
<u>-intervals</u>	880
<u>irun</u>	881
<u>+ixccWorkDir+<workdir></u>	881
<u>+ixccTest+<testname></u>	881
<u>+ixccOverwrite</u>	881
<u>lp (Low-power Run-time Command)</u>	882
<u>Options of Run-Time lp Command for CPF and IEEE 1801</u>	884
<u>Options of Run-Time lp Command for CPF</u>	886
<u>Options of Run-Time lp Command for IEEE 1801</u>	889
<u>memory (in SA Mode)</u>	902
<u>-dump {<file format> <memory name> -file <file name>}</u>	903
<u>-load {[<file format>]} <memory name> -file <file name></u>	906
<u>-list</u>	908
<u>-set {-all <memory name>...}</u>	908
<u>-reset {-all <memory name>...}</u>	908
<u>-setvalue {-all <list of instances>...} [-start <addr1>] [-end <addr2>]</u>	908
<u>-setvalue -value {<value> all0 all1} [-mask <value>]</u>	909
<u>-setvalue {-all <list of instances>...} [-start <addr1>] [-end <addr2>]</u>	909
<u>-getFileOffset <file name></u>	909

UXE Command Reference Manual

<u>sparse mem info [<mem name>]</u>	910
<u>memory (in ICE Mode)</u>	911
<u>-dump {<file format> <memory name> -file <file name>}</u>	913
<u>-load {[<file format>] <memory name> -file <file name> [-retainValue] [-retainValueOffset]}</u>	918
<u>-list</u>	922
<u>-info {-all <memory name>...} [-help]</u>	923
<u>-set {-all <memory name>...}</u>	923
<u>-reset {-all <memory name>...}</u>	923
<u>-setvalue {-all <list of instances>...} [-start <addr1>] [-end <addr2>]</u>	923
<u>-setvalue -value {<value> all0 all1} [-mask <value>]</u>	924
<u>-setvalue {-all <list of instances>...} [-start <addr1>] [-end <addr2>]</u>	924
<u>-getFileOffset <file name></u>	925
<u>foreground -fg</u>	925
<u>-sparse_mem_info [-v] <mem name></u>	925
<u>-sparse_mem_info -usedpages [-v] <mem name></u>	925
<u>-sparse_mem_info -statistic [-v] [-byusage]</u>	926
<u>-sparse_mem_info -overflow</u>	926
<u>Global Options Commands</u>	926
<u>Global Options Stack Commands</u>	927
<u>Examples</u>	929
<u>Example of Batch Mode Operation</u>	930
<u>File Formats</u>	931
<u>ncsim</u>	937
<u>[-set] {<logfile name> -default}</u>	937
<u>append <logfile name></u>	937
<u>overwrite <logfile name></u>	938
<u>show [-default]</u>	938
<u>probe</u>	939
<u>-create</u>	940
<u>-fast</u>	940
<u>-all</u>	941
<u>-allnets</u>	941
<u>-depth {<n> all to_cells}</u>	941
<u>-name <probe name></u>	941
<u>-ports</u>	942

UXE Command Reference Manual

<u>waveform</u>	942
<u><object> ... <scope_name></u>	942
<u>-errorok</u>	942
<u>-assertion</u>	942
<u>-state</u>	943
<u>-signals</u>	943
<u>-exclude {<object> <scope_name>}</u>	943
<u>-excludelibrary <.llib names></u>	943
<u>-delete <probe_name> [<probe_name>...]</u>	943
<u>-list [<probe_name>...] [-fast]</u>	944
<u>-save [<filename>] [-fast]</u>	944
<u>-show [<probe_name>...][-fast] [-v]</u>	944
<u>-addcone -depth <n> <signal> ...</u>	945
<u>-addpath -depth <n> <net1> <net2></u>	945
<u>-addIllegalDomainConfiguration {<illegalDomainConfiguration> *}</u>	945
<u>-addMode {<modeName> *}</u>	945
<u>-addModeTransition {<modeTransitionName> *}</u>	946
<u>-addPowerDomain {<powerDomainName> *}</u>	946
<u>-addPowerMode {<powerModeName> *}</u>	946
<u>-addIsolation {<isolationRuleName> *}</u>	946
<u>-addRetention {<stateRetentionRule> *}</u>	947
<u>-addpowerstate [<namePattern>]</u>	947
<u>-addpowerdomain [<namePattern>]</u>	948
<u>-addisolation [<namePattern>]</u>	948
<u>-addretention [<namePattern>]</u>	949
<u>-addsupplyport [<namePattern>]</u>	950
<u>-addsupplynet [<namePattern>]</u>	951
<u>-addsupplyset [<namePattern>]</u>	951
<u>-addswitch [<namePattern>]</u>	952
<u>-addpst [<namePattern>]</u>	953
<u>-addportstate [<namePattern>]</u>	953
<u>-addpststate [<namePattern>]</u>	954
<u>-power</u>	954
<u>release</u>	955
<u>-keepvalue</u>	955
<u><signal> ... <symbol> ...</u>	955

UXE Command Reference Manual

<u>-allff [-instance <instance_name>]</u>	955
<u>reset</u>	956
<u>Examples</u>	956
<u>resetTime</u>	957
<u>restart</u>	958
<u>[-path <path>]</u>	958
<u><save_name></u>	959
<u>-savedb</u>	959
<u>-show</u>	959
<u>restart (in VVM)</u>	960
<u>resultWaveform</u>	961
<u>Examples</u>	961
<u>resume</u>	962
<u>Example</u>	962
<u>run</u>	963
<u>-emulation</u>	964
<u><amount_to_run></u>	964
<u><time_unit></u>	965
<u>-wait</u>	965
<u>-nowait</u>	965
<u>-continue</u>	965
<u><cycles></u>	966
<u>-swap</u>	966
<u>-ignorestop</u>	966
<u>Examples</u>	966
<u>save</u>	967
<u>[-path <path>]</u>	967
<u>[-simulation] <snapshot_name></u>	967
<u><save_name></u>	968
<u>-compress [{Low Medium High None}]</u>	968
<u>-overwrite</u>	969
<u>{-commands -environment} [<filename>]</u>	969
<u>-uservars <filename></u>	969
<u>Examples</u>	969
<u>scope</u>	971
<u>[-set] <scope_name></u>	971

UXE Command Reference Manual

<u>-up</u>	971
<u>-describe [<scope_name>]</u>	971
<u>-show</u>	972
<u>sdl</u>	973
<u>sdl -autoProbe [no yes]</u>	976
<u>sdl -addProbes [-fast] <sdl_file>...</u>	977
<u>sdl -disable -save <filename></u>	977
<u>sdl -disable -list</u>	977
<u>sdl -disable [-now]</u>	977
<u>sdl -disable [-now] [<instance_names_list> -none]</u>	978
<u>sdl -disable [-now] -label <label-pattern> <label-pattern>...</u>	978
<u>sdl -disable -label -list</u>	980
<u>sdl -disable -stream <stream-pattern> <stream-pattern>...</u>	980
<u>sdl -disable -stream -list</u>	980
<u>sdl -display -console [0 1]</u>	980
<u>sdl -display -endofline 0 1</u>	981
<u>sdl -display -file [<file_name>]</u>	981
<u>sdl -display -overflow [error warning ignore]</u>	981
<u>sdl -display -translate 0 1</u>	982
<u>sdl -display -id <ID> [-file <file_name>]</u>	982
<u>sdl -display -id <ID> -disable -enable</u>	983
<u>sdl -display -loglevel <loglevel></u>	983
<u>sdl -display -deflevel <deflevel></u>	984
<u>sdl -dumpKeepNets [-exclude] <sdl_file> <script_file></u>	984
<u>sdl -enable -list</u>	984
<u>sdl -enable [-now]</u>	984
<u>sdl -enable [-now] [<instance_names_list> -none]</u>	985
<u>sdl -enable [-now] -label <label-pattern> <label-pattern>...</u>	985
<u>sdl -enable -label -list</u>	986
<u>sdl -enable -stream <stream-pattern> <stream-pattern>...</u>	986
<u>sdl -enable -stream -list</u>	987
<u>sdl -expression [<expression>]</u>	987
<u>sdl -execOvf [error warning ignore]</u>	987
<u>sdl -getActiveInstances</u>	988
<u>sdl -getAssertions</u>	988
<u>sdl -getFiles</u>	988

UXE Command Reference Manual

<u>sdl -getInstances</u>	988
<u>sdl -getLabels [-detail]</u>	988
<u>sdl -getNets</u>	989
<u>sdl -getStatus [-instance <instance name>] <parameter></u>	989
<u>sdl -getTriggers</u>	989
<u>sdl -haltOnTrigger [0 1]</u>	990
<u>sdl -isActive</u>	990
<u>sdl -isDispOvf [-clear]</u>	991
<u>sdl -isExecOvf [-clear]</u>	992
<u>sdl -isTrigger [-instance <instance name>] [-label <label>]</u>	992
<u>sdl -ldcnt1 -ldcnt2 [-instance <instance>] <value></u>	993
<u>sdl -load [<sdl file name>]</u>	993
<u>sdl -log -stopStream -startStream -upload [<size>] -reset</u>	994
<u>sdl -nacq <expression></u>	995
<u>sdl -ncbreakpoints -errorok [yes no]</u>	995
<u>sdl -ncbreakpoints -import [<filename>]</u>	996
<u>sdl -ncbreakpoints -exact [yes no]</u>	996
<u>sdl -onlyReport [yes no]</u>	997
<u>sdl -pp <output file> [<sdl file name>]</u>	998
<u>sdl -report [-file <file name>]</u>	998
<u>sdl -restoreState <file></u>	998
<u>sdl -saveState <file></u>	999
<u>sdl -scope [<scope>]</u>	999
<u>sdl -setFile [<sdl filename>]</u>	1000
<u>sdl -simTime yes no</u>	1000
<u>sdl -stop</u>	1000
<u>sdl -traceClear run dump never</u>	1000
<u>sdl -traceDump [-location] [-lineOnly] [-short] [-index c dc t dt] [-instance <instance name>] [-text -fsdb -sst2] [-time <time unit>] <file name></u>	1001
<u>sdl -traceOn [-acqFilter no yes] [-postTrigger no yes] [<action list> ALWAYS DEFAULT]</u>	1002
<u>sdl -tx {-sst2 -text -ida} <filename> -close -show</u>	1003
<u>sdl -tx -illegal [error warning ignore reportwarning clearwarning]</u>	1004
<u>sdl -tx -statistics txid scope stream type</u>	1005
<u>sdl -verify [<filename>]</u>	1005
<u>sdl -when [-time] [<time unit>]</u>	1005

UXE Command Reference Manual

<u>Examples</u>	1008
<u>stop</u>	1011
<u>emulation</u>	1011
<u>Examples</u>	1011
<u>suspend</u>	1013
<u>Example</u>	1013
<u>symbol</u>	1014
<u>-add <symbol> <sig> ...</u>	1014
<u>-get <symbol></u>	1014
<u>-rm <symbol>...</u>	1015
<u>Examples</u>	1015
<u>tca</u>	1016
<u>-setFile <file_name></u>	1016
<u>-getAllInstanceNames [<fileID> ...] [-maxDisplay <number>]</u>	1016
<u>-getValues <[fileID] inst> [<[fileID] inst> ...] -start <start_time> [-end <end_time>] [-type toggle high] [-count ave max]</u>	1017
<u>-getTimeRange [<fileID> ...]</u>	1017
<u>-getTopNInst <number> [-start <start_time>] [-end <end_time>]</u>	
<u>[-type toggle high] [-count ave max] [<fileID> ...]</u>	1018
<u>-getTopNPeak <number> [-sensitivity <N>] [-start <start_time>]</u>	
<u>[-end <end_time>] [-type toggle high] [<[fileID] inst>...]</u>	1018
<u>-getTopNSlope <top_N number> [-start <start_time>] [-end <end_time>]</u>	
<u>[-type toggle high] [-step m] [<[fileID] inst>...]</u>	1019
<u>-export <waveform_name> [<fileID> inst> [<[fileID] inst> ...]</u>	
<u>[-format <text sst2 fsdb>] [-start <time>] [-end <time>]</u>	
<u>[-filter {<filtername> <filter_arguments>}] <list_of_instance_names></u>	1020
<u>-merge <source_ppfdata_list> -to <one_destination_ppfdata></u>	1022
<u>-file -open <filename></u>	1022
<u>-file -close <fileID></u>	1022
<u>-file -filename <fileID></u>	1022
<u>-file -fileid <filename></u>	1022
<u>-file -list</u>	1022
<u>-help</u>	1022
<u>userData</u>	1023
<u>-get [-max <n>] [<type> [option]]</u>	1023
<u>-dump <file></u>	1023
<u>Examples</u>	1023

UXE Command Reference Manual

<u>value</u>	1025
<u>-file <filename></u>	1025
<u>-short</u>	1026
<u>-verbose</u>	1026
<u><format></u>	1026
<u><object names></u>	1026
<u>vector</u>	1028
<u><stim_file></u>	1028
<u><cmd_file></u>	1028
<u><aux_file></u>	1029
<u><vbf_stim_file></u>	1029
<u>-compileOnly <file.cbc></u>	1029
<u>Examples</u>	1029
<u>vvm</u>	1030
<u>-prepare</u>	1030
<u>[-overwrite]</u>	1030
<u><session_name></u>	1030
<u>[-memoryReplay [<memory_instances>]]</u>	1030
<u>-open <session_name></u>	1031
<u>-close</u>	1031
<u>-list [<session_name> -allsession] [-info]</u>	1031
<u>[-intervals]</u>	1032
<u>waitForBusy</u>	1033
<u><timeout></u>	1033
<u>Examples</u>	1033
<u>xeset</u>	1035
<u>appendTrace</u>	1038
<u>autoResumeRun</u>	1038
<u>autoUpload</u>	1039
<u>autoExit</u>	1040
<u>bpValue</u>	1040
<u>database</u>	1041
<u>dbHwType</u>	1041
<u>dbPath</u>	1041
<u>depositNCsimMode</u>	1042
<u>design</u>	1043

UXE Command Reference Manual

<u>designName [<name>]</u>	1043
<u>endOfVector</u>	1044
<u>fsdbFileSizeLimit</u>	1044
<u>fvMaxThread [<number>]</u>	1045
<u>fvPreprocess</u>	1046
<u>fvSafeMode [0 1]</u>	1046
<u>gui</u>	1047
<u>-gui autoWave</u>	1047
<u>-gui dbAutoWave</u>	1047
<u>-gui dbAutoUpload</u>	1048
<u>-gui dbMaxListLen</u>	1049
<u>-gui dbShowGenerated</u>	1049
<u>-gui dbShowPrimitiveTerm</u>	1050
<u>-gui overwriteMemFile</u>	1050
<u>-gui svSaveWaveScript [0 1]</u>	1051
<u>-gui svShowGenerated</u>	1051
<u>-gui svUploadOnProbe</u>	1052
<u>-gui userNet1</u>	1052
<u>-gui userNet2</u>	1053
<u>-gui pollingRate</u>	1053
<u>host</u>	1054
<u>idleExit</u>	1054
<u>ignoreProbeFailureInForce [value]</u>	1055
<u>keepHostAlive</u>	1055
<u>keepHostAliveTimeout</u>	1057
<u>linkFiles</u>	1058
<u>maxMismatches</u>	1059
<u>msgMaxLen</u>	1060
<u>mtHost</u>	1060
<u>postTriggerSamples</u>	1061
<u>probeCompiled [0 1]</u>	1062
<u>probePrimary</u>	1063
<u>progress [off gui display stdout file file+ gui, display gui, stdout gui, file gui, file+ stdout, file stdout, file+ gui, stdout, file gui, stdout, file+]</u>	1064
<u>reserveKey</u>	1065
<u>reserveTimeout <timeoutValue></u>	1066

UXE Command Reference Manual

<u>retryDownload</u>	1066
<u>retryInterval</u>	1066
<u>sdlAutoNcBreakpoints</u>	1067
<u>sdlIncludePath</u>	1067
<u>signalRadix</u>	1068
<u>stopDelay</u>	1069
<u>suspendEnable</u>	1070
<u>timeRangeCheck</u>	1071
<u>timeUnit</u>	1072
<u>traceMemMax</u>	1073
<u>traceMemSize</u>	1074
<u>triggerPos</u>	1075
<u>uploadSdlTrace [0 1]</u>	1077
<u>vdCreateWaves [0 1]</u>	1078
<u>vdLegacyCompare [0 1]</u>	1078
<u>vector</u>	1079
<u>version</u>	1080
<u>xogui</u>	1081
<u>-create <name> [-type <type>] [-location <location>] [-pack <options>]</u>		
<u>[<Tk widget options>]</u>	1081
<u>-destroy <name></u>	1082
<u>-configure <name> <options></u>	1082
<u>-apply <name> <arguments></u>	1082
<u>-save <file_name></u>	1083

14

<u>SA Run-time and Debug Commands</u>	1085
<u>xc Command Options</u>	1086
<u>xc asrtFailMsg</u>	1088
<u>xc checkxoff</u>	1089
<u>xc checkxon</u>	1090
<u>xc chkmem</u>	1091
<u>xc chkreg</u>	1092
<u>xc chkregx</u>	1093
<u>xc ckpt</u>	1094

UXE Command Reference Manual

<u>xc coverage dump</u>	1095
<u>xc coverage functional -reset</u>	1096
<u>xc coverage code -reset</u>	1097
<u>xc coverage off</u>	1098
<u>xc coverage reset</u>	1099
<u>xc coverage setup</u>	1100
<u>xc deposit</u>	1101
<u>xc ecmset maxBpCycle</u>	1102
<u>xc ecmset maxSyncCycle</u>	1103
<u>xc ecmset maxGfifoSyncCycle</u>	1104
<u>xc emaskclr</u>	1105
<u>xc emaskoff</u>	1106
<u>xc emaskon</u>	1107
<u>xc export_frcrel</u>	1108
<u>xc fclkPerEval</u>	1109
<u>xc free</u>	1110
<u>xc g fifoDispTimeformat</u>	1111
<u>xc help</u>	1113
<u>xc ixe sync</u>	1114
<i>ixe sync [-on -off]</i>	1114
<u>xc matchmemon</u>	1115
<u>xc matchmemoff</u>	1116
<u>xc matchregon</u>	1117
<u>xc matchregoff</u>	1118
<u>xc memory</u>	1119
<u>xc memory -dump %<file_format> <memory_name> -file <file_name> [-start <n>] [-end <n>] [-z]</u>	1120
<u>xc memory -load [%<file_format>] <memory_name> -file <file_name> [-start <n>] [-end <n>] [-z]</u>	1121
<u>xc memory -list</u>	1123
<u>xc memory -set [<memory> -all]</u>	1123
<u>xc memory -reset [<memory> -all]</u>	1124
<u>xc memory -setvalue [<memory> -all] [-start n] [-end n] -value [<value> all0 all1]</u>	1124
<u>xc off</u>	1125
<u>xc on</u>	1126

UXE Command Reference Manual

<u>autorun</u>	1126
<u>bmatch</u>	1126
<u>initok</u>	1126
<u>match</u>	1126
<u>nbrun <delay></u>	1127
<u>piok</u>	1127
<u>run</u>	1127
<u>tbrun</u>	1127
<u>wait_init</u>	1127
<u>wait_init_stop</u>	1128
<u>xt0</u>	1128
<u>xt1</u>	1128
<u>zt0</u>	1129
<u>zt1</u>	1129
<u>xc pitraceoff</u>	1130
<u>xc pitraceon</u>	1131
<u>level</u>	1131
<u>xc poclk</u>	1132
<u>xc poclk add</u>	1133
<u>hierarchical_signal_name</u>	1133
<u>xc poclk rm</u>	1135
<u>xc potraceoff</u>	1136
<u>xc potraceon</u>	1137
<u>level</u>	1137
<u>xc profoff</u>	1138
<u>xc profon</u>	1139
<u>xc showcpi</u>	1140
<u>xc showcpo</u>	1141
<u>xc showfrc</u>	1142
<u>xc signal</u>	1143
<u>force <signal_name> <value></u>	1143
<u>release <signal_name></u>	1143
<u>set <signal_name> <value></u>	1143
<u>get <signal_name></u>	1143
<u>xc sigTrace</u>	1144
<u>-add <signal>...</u>	1145

UXE Command Reference Manual

<u>addinst [-depth <n>] <instance> ...</u>	1145
<u>addinstpin <instance> ...</u>	1145
<u>addcone [-depth <n>] <signal> ...</u>	1145
<u>addpath [-depth <n>] <net1> <net2></u>	1145
<u>-rm <signal>... *</u>	1145
<u>import <filename></u>	1145
<u>save [-qel] <filename></u>	1146
<u>upload</u>	1146
<u>clear</u>	1146
<u>-regexp <pattern> -glob <pattern></u>	1146
<u>-outfile <file_format> <basename></u>	1146
<u>assertion -add -rm <assertion_name></u>	1147
<u>assertion -addfanin -rmfanin <assertion_name></u>	1147
<u>assertion -addInst -rmlInst <inst_name></u>	1147
<u>assertion -addinstfanin -rminstfanin <inst_name></u>	1147
<u>assertion -addall</u>	1147
<u>assertion -rm *</u>	1147
<u>prepareoffline</u>	1147
<u>xc status</u>	1148
<u>xc stop</u>	1149
<u>xc swapInMem</u>	1150
<u>0</u>	1150
<u>1</u>	1150
<u>xc verbose</u>	1151
<u>level</u>	1151
<u>xc wait</u>	1152
<u>tries</u>	1152
<u>status</u>	1152
<u>xc xreg</u>	1153
<u>-save</u>	1153
<u>-scope <file_scope></u>	1153
<u>-rand <seed></u>	1153
<u>-all0 -all1</u>	1154
<u>-verbose</u>	1154
<u>xrun</u>	1155
<u>ixcc</u>	1156

<u>-inputDir</u>	1156
<u>-input</u>	1156
<u>-inputAll</u>	1157
<u>-outputDir</u>	1157
<u>-output</u>	1157
<u>-format <format_name></u>	1157
<u>-summary</u>	1157
<u>-listInst</u>	1157
<u>-matchInst</u>	1158
<u>-help</u>	1158
 15	
<u>Emulating with XEL</u>	1159
<u>Using the Scripts</u>	1160
<u>Running a Script in Batch Mode (non-interactive)</u>	1160
<u>Sourcing a Script File</u>	1160
<u>Compiler and Debugger Initialization Scripts</u>	1160
<u>Creating Tcl Scripts</u>	1161
<u>Debugging Tcl Scripts</u>	1162
<u>Installation of the Tcl Dev Kit (TDK):</u>	1162
<u>Attaching the TDK debugger to an xeDebug process - basic procedure</u>	1162
<u>Automatic attachment of TDK debugger at xeDebug initialization</u>	1163
<u>Tcl Script Examples</u>	1164
<u>Compile Scripts</u>	1164
<u>Importing a Design</u>	1166
<u>Synthesis For Emulation</u>	1167
<u>Emulating in Vector Debug Mode</u>	1167
<u>Emulating in Logic Analyzer Mode</u>	1168
<u>Emulating in STB Mode</u>	1168
<u>Assigning IP Instances</u>	1169
 16	
<u>Deprecated Commands and Variables</u>	1171
<u>attr</u>	1175
<u><dbObj></u>	1175

UXE Command Reference Manual

<u>-all</u>	1175
<u>-get</u>	1176
<u><attrType></u>	1176
<u>Library Objects</u>	1176
<u>Cell Objects</u>	1177
<u>Instance Objects</u>	1177
<u>Net Objects</u>	1178
<u>Terminal Objects</u>	1178
<u>Terminal Instance Objects</u>	1179
<u>Examples</u>	1179
<u>cell</u>	1180
<u><cellobj></u>	1180
<u><cellName></u>	1180
<u><libobj></u>	1180
<u>closelib</u>	1181
<u><libobj></u>	1181
<u>Examples</u>	1181
<u>compileFindAll</u>	1182
<u>-max_trials <max_trials></u>	1183
<u>-seed <start_seed></u>	1183
<u>-max_steps <max_steps></u>	1183
<u>-min_steps <min_steps></u>	1183
<u>-crit_paths <crit_paths></u>	1183
<u>-etOnly</u>	1184
<u>-2pass</u>	1184
<u>-logfile <logfile></u>	1184
<u>compileFindAllMemUtil</u>	1185
<u>-max_trials <max_trials></u>	1186
<u>-max_util <max_util></u>	1186
<u>-min_util <min_util></u>	1186
<u>-dec_util <dec_util></u>	1186
<u>-trials_per_util <trials_per_util></u>	1186
<u>-seed <start_seed></u>	1187
<u>-max_steps <max_steps></u>	1187
<u>-min_steps <min_steps></u>	1187
<u>-crit_paths <crit_paths></u>	1187

UXE Command Reference Manual

<u>-etOnly</u>	1188
<u>-2pass</u>	1188
<u>-logfile <logfile></u>	1188
<u>-minMem</u>	1188
<u>compileFindBest</u>	1189
<u>-max_trials <max_trials></u>	1189
<u>-max_steps <max_steps></u>	1189
<u>-min_steps <min_steps></u>	1189
<u>-crit_paths <crit_paths></u>	1190
<u>-minMem</u>	1190
<u>-etOnly</u>	1190
<u>-2pass</u>	1190
<u>compileFindBestOptimization</u>	1192
<u>-max_trials_each <max_trials_each></u>	1193
<u>-oE {<values>}</u>	1193
<u>-pE {<values>}</u>	1193
<u>-seed <start_seed></u>	1194
<u>-max_steps <max_steps></u>	1194
<u>-min_steps <min_steps></u>	1194
<u>-crit_paths <crit_paths></u>	1194
<u>-etOnly</u>	1195
<u>-2pass</u>	1195
<u>-logfile <logfile></u>	1195
<u>-minMem</u>	1195
<u>compileFindFirst</u>	1196
<u>-max_trials <max_trials></u>	1196
<u>-max_steps <max_steps></u>	1196
<u>-min_steps <min_steps></u>	1196
<u>-crit_paths <crit_paths></u>	1197
<u>-minMem</u>	1197
<u>-etOnly</u>	1197
<u>-2pass</u>	1197
<u>cosimAcs</u>	1198
<u>-add {<net> <edge>}</u>	1198
<u>-rm {<net>}</u>	1198
<u>-clear</u>	1198

UXE Command Reference Manual

<u>createUserSession</u>	1199
<u><dir_name></u>	1199
<u>criticalAccess</u>	1200
<u>-add {<signal_name> [R RW W]}</u>	1200
<u>-rm <list_of_signals></u>	1200
<u>dontAcqOnEvent</u>	1201
<u><expression></u>	1201
<u>eventSignal</u>	1202
<u>group</u>	1203
<u>-add <group> <sig>...</u>	1203
<u>-rm <group></u>	1203
<u>-list <group></u>	1203
<u>-get <group></u>	1203
<u>-set <group> <value></u>	1203
<u>-force <group> <value></u>	1204
<u>-release <group></u>	1204
<u>inst</u>	1205
<u><instobj></u>	1205
<u><inst></u>	1205
<u><cellobj></u>	1205
<u>lib</u>	1206
<u><libobj></u>	1206
<u><lib></u>	1206
<u><viewtype></u>	1206
<u>net</u>	1207
<u><netobj></u>	1207
<u><net></u>	1207
<u><cellobj></u>	1207
<u>parallelCompileFindBest</u>	1208
<u>-max_trials <max_trials></u>	1208
<u>-seed <start seed></u>	1208
<u>-max_steps <max_steps></u>	1209
<u>-min_steps <min_steps></u>	1209
<u>-crit_paths <crit_paths></u>	1209
<u>-etOnly</u>	1209
<u>-first</u>	1210

UXE Command Reference Manual

<u>-1pass</u>	1210
<u>-keep_all</u>	1210
<u>-rerun_best</u>	1210
<u>-no_export</u>	1210
<u>parallelCompileFindBestOptimization</u>	1211
<u>-max_trials_each <max_trials_each></u>	1211
<u>-oE {<values>}</u>	1212
<u>-pE {<values>}</u>	1212
<u>-seed <start_seed></u>	1212
<u>-max_steps <max_steps></u>	1212
<u>-min_steps <min_steps></u>	1212
<u>-crit_paths <crit_paths></u>	1213
<u>-etOnly</u>	1213
<u>-first</u>	1213
<u>-1pass</u>	1213
<u>-keep_all</u>	1213
<u>-rerun_best</u>	1214
<u>-no_export</u>	1214
<u>-logfile <logfile></u>	1214
<u>primaryClockSource</u>	1215
<u>-add <primary_clock_source_name></u>	1215
<u>-rm <primary_clock_source_name></u>	1215
<u>qtCounter1</u>	1216
<u>qtCounter2</u>	1217
<u>qtTriggerByState</u>	1218
<u>qtTriggerFile</u>	1219
<u>-force</u>	1220
<u>resetCycleNum</u>	1221
<u>restoreState</u>	1222
<u><save_name></u>	1222
<u>runtimeCLOCKconf</u>	1223
<u>savelib</u>	1224
<u><libobj></u>	1224
<u>saveState</u>	1225
<u><save_name></u>	1225
<u>bin</u>	1225

UXE Command Reference Manual

<u>text</u>	1225
<u>-environment</u> <save_name>	1226
<u>signal</u>	1227
<u>-get</u> <signal>	1227
<u>-get</u> <symbol>	1227
<u>-get</u> <bus>[<x>:<y>]	1227
<u>-alias</u> <signal>	1228
<u>-set</u> <signal> 0 1 <u>-set</u> <symbol> <value>	1228
<u>-force</u> <signal> 0 1 <u>-force</u> <symbol> <value>	1228
<u>-allff</u> { <u>-set</u> <u>-force</u> } {0 1 random [-seed <number>]} [-instance <instance_name>]	1228
<u>-allff -release</u> [-instance <instance_name>]	1229
<u>-isForced</u> <signal>	1229
<u>-listForce</u>	1229
<u>-release</u> <signal> ... <u>-release</u> <symbol>	1229
<u>sigTrace</u>	1230
<u>-add</u> <signal>...	1231
<u>-addinst</u> [-allnets] -depth <n> <instance>	1231
<u>-addinst</u> [-iregexp -iglob -ipattern <inst_pattern>] <u>-addinst</u> [-nregexp -nglob -npattern <net_pattern>] <instance> [-nregexp -nglob -npattern <net_pattern>] <top_inst_name>	1231
<u>-addio</u> <instance>	1232
<u>-addcone</u> -depth <n> <signal>	1232
<u>-addpath</u> -depth <n> <net1> <net2>	1232
<u>-rm</u> <signal>...	1232
<u>-rm</u> [-regexp -glob -pattern <pattern>]*	1232
<u>-list</u> [-regexp -glob -pattern <pattern>]	1232
<u>-save</u> [-qel] <filename>	1233
<u>-import</u> <filename>	1233
<u>-clear</u>	1233
<u>-upload</u>	1233
<u>-prepareoffline</u>	1233
<u>-outfile</u> <file_format> <basename>	1234
<u>-must</u>	1234
<u>-assertion</u> -add -rm <assertion_name>	1234
<u>-assertion</u> -addfanin -rmfanin <assertion_name>	1234

UXE Command Reference Manual

<u>assertion -addAll -rmAll</u>	1234
<u>assertion -addInst -rmInst <inst_name></u>	1234
<u>captureMode [controlled uncontrolled]</u>	1235
<u>addinstpin <instance></u>	1235
<u>addPowerDomain <powerDomainName></u>	1235
<u>addRetention <stateRetentionRule></u>	1235
<u>addIsolation <isolationRuleName></u>	1235
<u>addPowerModeTransition</u>	1235
<u>setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>] <instance_names></u>	1236
<u>setupToggleCount -rminst <instance_names></u>	1236
<u>setupToggleCount -rminst -all</u>	1236
<u>setupToggleCount -listinst [-verbose]</u>	1236
<u>getToggleCount [-append] [-weighted] [-cycle <n>] [-file <name>]</u>	1236
<u>addinstff <instanceName> [-asicLibrary <.llib name> ... <.llib name>] [-hardmacroLibrary <.llib name> <.llib name>]</u>	1237
<u>addinstReg <instanceName> [-asicLibs <.libName><.libName>] [-hardmacroLibs <.libName><.libName>]</u>	1237
<u>-outfile -tcf <filename> [-instance <instanceName>] [-masterTCF] [-slaveTCF][-noregularTCF] [-segment <n>% <cycleNumber>]</u>	1237
<u>genPowerProfile -segmentLogFile <segmentLogFile> -reportFileNamePattern <FileNamePattern></u>	1238
<u>stimulusWaveform</u>	1239
<u>-display</u>	1239
<u>-quit</u>	1239
<u>-save <file></u>	1239
<u>-help</u>	1239
<u>stopDelay</u>	1240
<u>0</u>	1240
<u>1</u>	1240
<u>term</u>	1241
<u><termobj></u>	1241
<u><term></u>	1241
<u><cellobj></u>	1241
<u>termInst</u>	1242
<u><terminstobj></u>	1242
<u><terminst></u>	1242
<u><instobj></u>	1242

UXE Command Reference Manual

<u>Examples</u>	1242
<u>traceWaveform</u>	1244
<u>-display <readback_file> 0</u>	1244
<u>-restore <rc_file></u>	1244
<u>-update</u>	1244
<u>-quit</u>	1244
<u>-parseSymbols</u>	1245
<u>-save <file></u>	1245
<u>trEvent</u>	1246
<u>-add <event> = <expression></u>	1246
<u>-rm <event> ...</u>	1246
<u>-get <event> <symbol></u>	1246
<u>-save <filename></u>	1247
<u>-add <event> <value> <symbol> ...</u>	1247
<u>-edit <event> <symbol> <value></u>	1247
<u>verifyTrigFile</u>	1248
<u>xc uaPollLimit</u>	1249
<u>Index</u>	1251

UXE Command Reference Manual

Introduction

This command reference manual describes, in detail, the commands used to verify designs using Cadence Palladium XP verification computing platform in In-Circuit Emulation (ICE) and Simulation Acceleration (SA) modes. It is an integral part of the UXE documentation set, which includes:

- *Palladium XP Planning and Installation Guide*, which describes the major hardware and software components as well as the power, cooling, and IT requirements of Palladium XP emulators that are of interest to people responsible for planning, installing, or maintaining the system
- *Palladium Target System Developer's Guide*, which explains how to set up a target system for use with a Palladium XP emulator
- *UXE User Guide*, which describes the comprehensive use flow including information about compiling designs using either UXE compiler, that is, `xeCompile` (in ICE mode) or **Incisive Xaccelerator COMpiler**, that is, `IXCOM` (in SA mode), and running and debugging designs using the `xeDebug` or `irun` tool

Command Specification

Commands for UXE are categorized based on the following possible compile and run use flows offered on Palladium XP emulator:

- Compile the design for ICE mode using `xeCompile` and run using `xeDebug`
- Compile the design for SA mode using `IXCOM` tools and run using `xeDebug` (Simulation Acceleration)
- Compile the design for SA mode using `IXCOM` tools and run using `irun` (Software Simulation)
- Compile and run the design for SA mode using `irun` (Simulation Acceleration using `irun`)

Compile the design for ICE mode using xeCompile and run using xeDebug

- Invoke the xeCompile tool by executing the [xeCompile](#) command on the UNIX command prompt. This initializes the xeCompile tool, changes the UNIX command prompt to `XEC>`, and loads the set of compile XEL commands discussed in the following chapters:
 - ❑ [Chapter 5, “Import Commands”](#) that covers the commands used for accessing the Palladium XP design database
 - ❑ [Chapter 6, “HDL Compiler Commands”](#) that covers the commands used for RTL mapping
 - ❑ [Chapter 8, “User Data Commands”](#) that covers the commands used for accessing user data
 - ❑ [Chapter 11, “Compile-time Commands in ICE Mode”](#) that covers the commands used for compiling designs
- Invoke the debug tool by executing the [xeDebug](#) command. This initializes the xeDebug tool, changes the UNIX command prompt to `XE>`, and loads the set of debug commands discussed in [Chapter 13, “Run-time Commands.”](#)

For procedural information about this use flow, refer to the following chapters of the *UXE User Guide*:

- ❑ *Compiling Designs for In-Circuit Emulation with xeCompile*
- ❑ *Debugging Designs Using xeDebug Tool*

Compile the design for SA mode using IXCOM tools and run using xeDebug (Simulation Acceleration)

- Use IXCOM tools to analyze and compile designs by executing the [vhan](#), [vlan](#), and [ixcom](#) commands discussed in [Chapter 10, “Commands and System Tasks for SA Mode.”](#)
- Invoke the debug tool by executing the [xeDebug](#) command. This initializes the xeDebug tool, changes the UNIX command prompt to `XE>`, and loads the set of debug commands supported for designs compiled in SA mode. Refer to the [xeDebug Command Set for SA Mode](#) section for a list of supported commands.

For procedural information about this use flow, refer to the following chapters of the *UXE User Guide*:

- ❑ *Compiling Designs for Simulation Acceleration Using IXCOM*
- ❑ *Debugging Designs Using xeDebug Tool*

Compile the design for SA mode using IXCOM tools and run using *iRun* (Software Simulation)

- Use IXCOM tools to analyze and compile designs by executing the vhan, vlan, and ixcom commands discussed in [Chapter 10, “Commands and System Tasks for SA Mode.”](#)
- Refer to the [**iRun Command Set for SA Mode**](#) section for a list of supported and unsupported commands when using *iRun* and accelerating the design on the Palladium XP emulator.

For procedural information about this use flow, refer to the following chapters of the *UXE User Guide*:

- *Compiling Designs for Simulation Acceleration Using IXCOM*
- *Compiling and Running Simulation Acceleration Designs with iRun*

Compile and run the design for SA mode using *iRun* (Simulation Acceleration using *iRun*)

Use *iRun*, the front-end tool for the IES software simulator, to invoke IXCOM tools to compile a design for simulation acceleration and simulate the hardware-acceleratable snapshot using a single or a set of commands. Simulate the design using either ncsim or xeDebug.

Refer to [Chapter 10, “Commands and System Tasks for SA Mode”](#) for the supported command options for IXCOM tools (vhan, vlan, and ixcom).

Refer to the [**iRun Command Set for SA Mode**](#) section for a list of supported and unsupported commands when using *iRun* and accelerating the design on the Palladium XP emulator. For description of the listed commands, refer to *iRun User Guide* of the INCISIVE documentation set.

For procedural information about this use flow, refer to the *Compiling and Running Simulation Acceleration Designs with iRun* chapter of the *UXE User Guide*.

Introducing XEL

The commands used with the Palladium XP emulators are based on XEL, the basic scripting command language designed for easy access to emulation resources, and to accelerate the design verification process.

UXE Command Reference Manual

Introduction

XEL is based on the Tool Command Language (Tcl). XEL incorporates the popular language syntax and the programming functionality of Tcl, while including extensions that provide emulation-specific capabilities.



Explanation on Tcl is beyond the scope of this manual. For information about the Tcl features, such as following, refer to a book on Tcl programming or the references listed in the [Tcl References](#) section:

- Variable substitution
- Command substitution
- Double quotes "" vs. curly braces {}
- Control flow, such as if and for
- Command syntax and description

The XEL commands are executed from the XEL shell that launches the XEL interpreter. The set of commands populated in the XEL shell depends on the use flow that you adopt to compile and debug the design. Refer to the [Command Specification](#) section for information about the supported use flows.

The graphical user interfaces (GUI) supported by UXE are built on top of XEL, that is, individual forms and menu commands in the menus internally invoke the prepackaged XEL scripts or individual XEL commands.

A selected set of executed XEL commands, or menu selections executed by user menus are stored in a XEL log file, in the order that they were performed. The set of logged commands can be modified, and logging can be turn on or off.

Viewing online help files can assist in executing individual XEL commands, in using and customizing prepackaged script files, or in creating and executing XEL scripts.

Using XEL Shell

The XEL shell is initialized when you execute the command to launch either the compile tool or debug tool.

When the XEL shell is being initialized, the `.xerc` script file, which is located in the home directory, is executed. If the current directory from which the UXE software was invoked also has a file with the name `.xerc`, this file is executed next. If both the directories include the

UXE Command Reference Manual

Introduction

.xerc file, both files will be executed, starting from the home directory (see [Chapter 15, "Emulating with XEL"](#)).

When the Palladium XP GUI is invoked, the .xerc script file is executed before the initialization steps and opening of the main compile or debug tool window.

When you save the XEL commands in an XEL script file and runs it directly from the Unix prompt, the .xerc script file is executed before the initialization steps, and before executing the XEL script file (batch mode) or displaying the XEL prompt (interactive mode).

You can use the .xerc file to define the Tcl procedures and variables for setting up the XEL environment, or to set other XEL options, such as turning on or off the feature to log specific commands.

The following is an example of a .xerc file:

```
puts "Executing Fred's XEL customization script"
# Display all report messages in the main GUI.
reportMsg on
# Instruct Palladium not to log the cd command into
# the log file, because it clutters the xe.msg
# file when using the GUI.
unlogCommand cd
proc my_run {} {
    run 1000
    puts "current emulation time is [getTime ns]"
}
```

The .xerc file cannot request to execute any XEL command whose execution depends on availability of resources such as an allocated emulator or an opened design database. This practically includes most of the XEL commands. On the other hand, the .xerc file may contain definitions of Tcl procedures that contain any XEL commands. For automatic execution of XEL commands after such resources become available during run time, use a .rtxerc file. (The .rtxerc has no effect during a compile session).

If a .rtxerc file exists either in the home directory or in the directory from which the UXE software was invoked, this file will be executed during run time after successful configuration of the emulator (on a successful conclusion of the configPM command). If both directories include a .rtxerc file, both files will be executed, starting from the home directory.



Caution

Do not manually edit any system-generated files, such as the User Data file. Always use relevant commands to change data in your design database, especially in user data. Manually editing files can corrupt the file or add data to the file in wrong sequence.

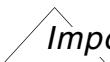
Using XEL Scripts

You can create script files that combine several XEL commands to automate each process. Samples of XEL script files can be found in the `<install_dir>/share/uxe/gift` directory.

Conventions Used in Command Syntax

In all command syntax examples in this manual:

- [brackets] enclose one or more optional parameters. You can execute the command without the optional parameters, but you must include any parameters that are not enclosed in brackets. Do not type the brackets.
- {curly braces} enclose a set of parameters. The set might contain a variable number of parameters, or might be repeatable in sequence. Unlike other syntax conventions, you type the braces literally as a delimiter for the set of parameters.
- `<angle brackets>` enclose variable names representing a value that you must specify. For example, instead of `<library>`, you must enter the actual name of your library. Do **not** type the angle brackets.
- bars | separate each choice in a multiple-choice parameter within a set of brackets or braces. For example, {verilog | vhdl} means you must use verilog or vhdl as a parameter of the command. [low | medium | high] indicates that you can use low, medium or high, or none of these parameters. Do not type the bar separating the parameters.
- ellipsis (...) indicate that you can repeat the parameter immediately preceding the ellipsis. For example, `<file>...` indicates that you can enter one or more file names. A set of parameters enclosed in braces and followed by an ellipsis indicates that you can repeat the entire set of parameters in sequence. Do not type the ellipsis.



Brackets [] are reserved characters in Tcl, used to enclose commands. To use brackets as regular characters in XEL commands, you need to precede each bracket with an escape character, or enclose the entire string in curly braces { }. For example, if you use the `probe` XEL command to specify a probe in the format `<prefix>[<number>]`, XEL reports an error.

For example:

```
probe -add P[2] => invalid command name "2"
```

However,

```
probe -add P\[2\] => this format works  
probe -add {P[2]} => this format also works
```

Double quotes "" and curly braces {} are intrinsic Tcl semantics that work exactly the same on all XEL commands. Either double quotes or curly braces can be used to present a string as a single argument to a Tcl statement. The difference between the two is that double quotes allow Tcl variable substitution inside them, while curly braces do not.

For example, if there is a Tcl variable by the name `var` that has the value `xyz`, either `$var` or `${var}` will be substituted with the string `xyz`. The difference is that `${var}123` will be substituted with the string `xyz123`, while `$var123` is not going to work because Tcl will look for a variable with the name `var123`.

Tcl References

All XEL commands described in this manual, and the Tcl and Tk version 8.5.9 commands are supported.

For more information about the Tool Command Language, see the following resources:

- *Tcl and the TK Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company, Reading, Mass., 1994. (Note that this book does not cover the most recent versions of Tcl and Tk.)
- *Practical Programming in Tcl and Tk*, Brent B. Welch, Prentice Hall PTR, Upper Saddle River, N.J., 4th Edition 2003
- *Tcl/Tk Programmer's Reference*, Chris Nelson, McGraw Hill Professional Publishing, 1999
- *Tcl/Tk for Real Programmers*, Clif FlyntMorgan Kaufmann Publishers, 1998.
- *Graphical Applications with Tcl & Tk*, Eric F. Johnson, M&T Books, New York, N.Y., 2nd Edition, 1999
- <http://www.tcl.tk>, a web site with links to many sites on Tcl from which you can also download manuals or related software

Note: Cadence does not maintain, and cannot guarantee the continued availability of, any web site listed on this page.

UXE Command Reference Manual

Introduction

UXE Command List

This chapter discusses the commands that are supported when compiling or running the design in either ICE mode or SA mode.

`xeCompile` (ICE mode), `IXCOM` (SA mode), and `iRun` (SA mode) are the compile-time tools available for compiling the designs.

`xeDebug` (ICE mode and SA mode) and `iRun` (SA mode only) are the run-time tools available for debugging the designs.

Each tool provides a different set of commands and command options that enable you to compile, run, and debug designs. You cannot execute a command relevant for one tool in another tool. For example, a command relevant for the `xeCompile` tool cannot be executed in the `xeDebug` tool or vice versa.

This chapter alphabetically lists all the commands and command options based on their availability within a specific tool. The list also provides links to the chapters of this manual that describe the corresponding command or command option.

For a list of all available commands in `xeCompile` or `xeDebug`, specify the following command in the tool being used:

```
help *
```

For a list of all available commands in the `IXCOM` tools, execute any of the following commands based on the tool being used:

```
ixcom all  
vlan all  
vhan all
```

The following sections list the supported command-sets:

- [Programs and Utilities](#)
- [Command Set for SA Mode](#)
 - [Compile-time Command Set for SA Mode](#)

- [xeDebug Command Set for SA Mode](#)
- [irun Command Set for SA Mode](#)
- [Command Set for ICE Mode](#)
 - [xeCompile Command Set](#)
 - [xeDebug Command Set for ICE Mode](#)
- [Common XEL Commands](#)
- [vvmDebug Command Set](#)

Programs and Utilities

This section lists the programs and utilities available at compile and run time. These programs and utilities are tools that are invoked from the UNIX shell.

Compiler Programs and Utilities

The following table lists the compiler programs and utilities, and refers to the location in this manual where more information about these tools can be found.

Table 2-1 Alphabetical List of Compiler Programs and Utilities

Command Name	Page	See Chapter
<u>ipBuilder</u>	<u>103</u>	<u>Chapter 3, “Programs and Utilities”</u>
<u>ModelChecker</u>	<u>104</u>	<u>Chapter 3, “Programs and Utilities”</u>
<u>primCount</u>	<u>110</u>	<u>Chapter 3, “Programs and Utilities”</u>
<u>xeCompile</u>	<u>132</u>	<u>Chapter 3, “Programs and Utilities”</u>

Run-time Programs and Utilities

The following table lists the run-time programs and utilities, and refers to the location in this manual where more information about these tools can be found.

Table 2-2 Alphabetical List of Run-time Programs and Utilities

Command Name	Page	See Chapter
<u>test_server</u>	<u>111</u>	<u>Chapter 3, “Programs and Utilities”</u>
<u>vCompare</u>	<u>121</u>	<u>Chapter 3, “Programs and Utilities”</u>
<u>verigen</u>	<u>123</u>	<u>Chapter 3, “Programs and Utilities”</u>
<u>xeDebug</u>	<u>135</u>	<u>Chapter 3, “Programs and Utilities”</u>

Command Set for SA Mode

Compile-time Command Set for SA Mode

The following commands and their options discussed in [Chapter 10, “Commands and System Tasks for SA Mode”](#) are available for compiling designs in the SA mode:

- [vhan](#) on page 378
- [vlan](#) on page 411
- [ixcom](#) on page 464

xeDebug Command Set for SA Mode

This section lists the XEL commands available in the xeDebug tool when the design is compiled in SA mode, and refers to the location in this manual where more information about the command can be found. This set of commands includes the adapted NC-Sim commands and some emulation-specific commands.

Table 2-3 Alphabetical List of Run-time Commands

Command Name	Page	See Chapter
<u>assertion</u>	<u>780</u>	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual
UXE Command List

Table 2-3 Alphabetical List of Run-time Commands, *continued*

Command Name	Page	See Chapter
<u>convertRadix</u>	<u>804</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>database</u>	<u>806</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>debug</u>	<u>813</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>deposit</u>	<u>815</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>dpa</u>	<u>820</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>force</u>	<u>842</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>genPowerProfile</u>	<u>846</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getCableStatus</u>	<u>849</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getCycleNum</u>	<u>854</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getHostStatus</u>	<u>855</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getPMStatus</u>	<u>856</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getTime</u>	<u>859</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>hdlConfig</u>	<u>861</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>host</u>	<u>866</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>infiniTrace</u>	<u>872</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>irun</u>	<u>881</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>lp (Low-power Run-time Command)</u>	<u>882</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>memory (in ICE Mode)</u>	<u>911</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>ncsim</u>	<u>937</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>probe</u>	<u>939</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>release</u>	<u>955</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>reset</u>	<u>956</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>restart</u>	<u>958</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>run</u>	<u>963</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>save</u>	<u>967</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>scope</u>	<u>971</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>sdl</u>	<u>973</u>	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-3 Alphabetical List of Run-time Commands, *continued*

Command Name	Page	See Chapter
<u>stop</u>	<u>1011</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>symbol</u>	<u>1014</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>tca</u>	<u>1016</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>userData</u>	<u>1023</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>value</u>	<u>1025</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>xeset</u>	<u>1035</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>xegui</u>	<u>1081</u>	<u>Chapter 13, “Run-time Commands”</u>

xc Commands

In Simulation Acceleration mode, to access the design running on the hardware, the following set of commands is also provided:

Table 2-4 Alphabetical List of xc Commands

Command Name	Page	See Chapter
<u>xc checkxoff</u>	<u>1089</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc checkxon</u>	<u>1090</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc chkmem</u>	<u>1091</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc chkreg</u>	<u>1092</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc chkregx</u>	<u>1093</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc coverage_dump</u>	<u>1095</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc coverage_off</u>	<u>1098</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc coverage_reset</u>	<u>1099</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc coverage_setup</u>	<u>1100</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc deposit</u>	<u>1101</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc emaskclr</u>	<u>1105</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc emaskoff</u>	<u>1106</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc emaskon</u>	<u>1107</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-4 Alphabetical List of xc Commands, *continued*

Command Name	Page	See Chapter
<u>xc free</u>	<u>1110</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc help</u>	<u>1113</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc ixe sync</u>	<u>1114</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc matchmemon</u>	<u>1115</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc matchmemoff</u>	<u>1116</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc matchregon</u>	<u>1117</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc matchregoff</u>	<u>1118</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc memory</u>	<u>1119</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc off</u>	<u>1125</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc on</u>	<u>1126</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc pitraceoff</u>	<u>1130</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc pitraceon</u>	<u>1131</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc poclk</u>	<u>1132</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc poclk add</u>	<u>1133</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc poclk rm</u>	<u>1135</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc potraceoff</u>	<u>1136</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc potraceon</u>	<u>1137</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc profoff</u>	<u>1138</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc profon</u>	<u>1139</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc showcpi</u>	<u>1140</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc showcpo</u>	<u>1141</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc showfrc</u>	<u>1142</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc signal</u>	<u>1143</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc sigTrace</u>	<u>1144</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc status</u>	<u>1148</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc stop</u>	<u>1149</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc swapInMem</u>	<u>1150</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>

Table 2-4 Alphabetical List of xc Commands, continued

Command Name	Page	See Chapter
<u>xc verbose</u>	<u>1151</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc wait</u>	<u>1152</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xc xreg</u>	<u>1153</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>
<u>xrun</u>	<u>1155</u>	<u>Chapter 14, “SA Run-time and Debug Commands”</u>

Task-based Categorization of xc Commands

The following sections categorize the `xc` commands based on the task that can be performed using them:

- [Execution mode control options](#)
- [Emulator locking control options](#)
- [Hot swap \(Software to Hardware\) options](#)
- [Match mode options](#)
- [Emulator interface control options](#)
- [Emulator interface event trace control options](#)

Use `xc help` to get the complete list of `xc` commands.

Execution mode control options

```
xc on -run  
xc on -tbrun  
xc on -autorun  
xc on -match  
xc on -bmatch  
xc stop  
xc off  
xc status  
xc free
```

Emulator locking control options

```
xc wait [<tries>] [status]
```

UXE Command Reference Manual

UXE Command List

```
xc free  
xc on -wait_init
```

Hot swap (Software to Hardware) options

```
xc on -xt0  
xc on -xt1  
xc on -zt0  
xc on -zt1  
xc on -initok  
xc on -piok  
xc checkxon  
xc checkxoff
```

Match mode options

```
xc matchmemoff  
xc matchmemon  
xc matchregoff  
xc matchregon  
xc chkreg  
xc chkmem
```

Emulator interface control options

```
xc poclk [add | rm]
```

Emulator interface event trace control options

```
xc pitraceon  
xc pitraceoff  
xc potraceoff  
xc verbose [level]
```

For the description of `xc` commands, refer to [Chapter 14, “SA Run-time and Debug Commands.”](#)

NC-Sim Escaped Commands

Some of the native NC-Sim commands can be specified using the `ncsim` prefix, such as `ncsim assertion` and `ncsim probe`.

***irun* Command Set for SA Mode**

This section lists the supported commands when the design is compiled in SA mode using IXCOM, and *irun* is used for running and debugging the design.

xc Commands

The following `xc` commands can be used at the time of running and debugging the design with *irun*:

Table 2-5 Alphabetical List of xc Commands

Command Name	Page	See Chapter
<code>xc deposit</code>	1101	Chapter 14, “SA Run-time and Debug Commands”
<code>xc export_frcrel</code>	1108	Chapter 14, “SA Run-time and Debug Commands”
<code>xc memory</code>	1119	Chapter 14, “SA Run-time and Debug Commands”
<code>xc on -wait_init</code>	1127	Chapter 14, “SA Run-time and Debug Commands”
<code>xc signal</code>	1143	Chapter 14, “SA Run-time and Debug Commands”
<code>xrun</code>	1155	Chapter 14, “SA Run-time and Debug Commands”

NC-Sim Commands

Following NC-Sim commands do not access the DUT in the hardware correctly:

- `memory`
- `force`
- `release`
- `deposit`
- `probe`
- `value`

To work with these commands, ensure the following:

- For using the `memory` command specify the following:

```
xc memory
```

- ❑ For using NC-Sim commands `force` and `release` on DUT signals in the hardware, follow either of these steps:
 - modify the HDL to include the system task `$export_frcrel` on the target signals
 - dynamically `export_frcrel` on the target signals at run time by specifying:
`xc export_frcrel <signal>`
- ❑ For using `deposit` command, specify the following at run time:
`xc deposit`

Alternatively, specify the following:

```
xc signal -force <signal> <value>
xc signal -release <signal>
xc signal -set <signal>
```

However, if these commands are compiled with `$xc export_frcrel` system task in the HDL, or if the signals are first exported specifying `$xc export_frcrel` in `xeDebug`, the above mentioned commands are supported in that case. For more information on `$xc export_frcrel`, refer to the [System Tasks and Procedures](#) section of [Chapter 10, “Commands and System Tasks for SA Mode.”](#)

Unsupported Commands in *iRun*

- Probe Assertions
`probe -create -assertion -transaction`
- Trace/Attempt-based counting
`assert_count_attempts`
- Control SVA Processing
`assertion -strict on|off`
- 'Strong Semantics
`assert_report_incompletes`
- Log Assertion Output
`assertion -logging [-state <state> {<state_list>}] [-redirect <filename>] [-append] [-on|off] [-depth <levels>|to_cells|all] [<assertion_locator>]`
- Control Summary Report Style

```
assertion -style [-multiline|-oneline]  
[-unit|-statement]
```

Command Set for ICE Mode

xeCompile Command Set

This section lists the XEL commands available in the xeCompile tool, and refers to the location in this manual where more information about the command can be found.

Table 2-6 Alphabetical List of xeCompile Commands

Command Name	Page	See Chapter
breakNetHalfCycle	212	Chapter 8, “User Data Commands”
breakPin	213	Chapter 8, “User Data Commands”
breakPinHalfCycle	216	Chapter 8, “User Data Commands”
cableConnection	217	Chapter 8, “User Data Commands”
caCell	372	Chapter 9, “IP Commands”
clockAssign	221	Chapter 8, “User Data Commands”
clockDelay	224	Chapter 8, “User Data Commands”
clockFrequency	225	Chapter 8, “User Data Commands”
clockOption	227	Chapter 8, “User Data Commands”
clockSource	231	Chapter 8, “User Data Commands”
compile	686	Chapter 11, “Compile-time Commands in ICE Mode”
compileFind	689	Chapter 11, “Compile-time Commands in ICE Mode”
compilerOption	233	Chapter 8, “User Data Commands”
create_isolation_rule	722	Chapter 12, “Common Power Format Commands”
create_mode	726	Chapter 12, “Common Power Format Commands”

UXE Command Reference Manual
UXE Command List

Table 2-6 Alphabetical List of xeCompile Commands, *continued*

Command Name	Page	See Chapter
<code>create mode transition</code>	727	Chapter 12, “Common Power Format Commands”
<code>create_nominal_condition</code>	728	Chapter 12, “Common Power Format Commands”
<code>create_power_domain</code>	730	Chapter 12, “Common Power Format Commands”
<code>create_power_mode</code>	737	Chapter 12, “Common Power Format Commands”
<code>create_state_retention_rule</code>	740	Chapter 12, “Common Power Format Commands”
<code>db</code>	198	Chapter 7, “Database Commands”
<code>delayAsynClockInputs</code>	256	Chapter 8, “User Data Commands”
<code>delayBox</code>	258	Chapter 8, “User Data Commands”
<code>delayBoxInput</code>	261	Chapter 8, “User Data Commands”
<code>design</code>	705	Chapter 11, “Compile-time Commands in ICE Mode”
<code>designImport</code>	156	Chapter 5, “Import Commands”
<code>disconnectDriverPin</code>	707	Chapter 11, “Compile-time Commands in ICE Mode”
<code>emulatorConfiguration</code>	264	Chapter 8, “User Data Commands”
<code>end_design</code>	745	Chapter 12, “Common Power Format Commands”
<code>end_macro_model</code>	746	Chapter 12, “Common Power Format Commands”
<code>getCriticalPathDesignNetNames</code>	708	Chapter 11, “Compile-time Commands in ICE Mode”
<code>globalNet</code>	268	Chapter 8, “User Data Commands”
<code>hierAssign</code>	269	Chapter 8, “User Data Commands”
<code>hdlDefineLib</code>	172	Chapter 6, “HDL Compiler Commands”
<code>hdlImport</code>	174	Chapter 6, “HDL Compiler Commands”

UXE Command Reference Manual

UXE Command List

Table 2-6 Alphabetical List of xeCompile Commands, continued

Command Name	Page	See Chapter
<u>hdlInputFile</u>	<u>179</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hdlLang</u>	<u>182</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hdlLogFile</u>	<u>183</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hdlOutputFile</u>	<u>184</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hdlSkipModule</u>	<u>187</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hdlSynthesize</u>	<u>189</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hdlWorkPath</u>	<u>196</u>	<u>Chapter 6, “HDL Compiler Commands”</u>
<u>hwInfo</u>	<u>270</u>	<u>Chapter 8, “User Data Commands”</u>
<u>icePrepare</u>	<u>709</u>	<u>Chapter 11, “Compile-time Commands in ICE Mode”</u>
<u>identify_power_domain</u>	<u>275</u>	<u>Chapter 8, “User Data Commands”</u>
<u>impDumpConfig</u>	<u>158</u>	<u>Chapter 5, “Import Commands”</u>
<u>impLoadConfig</u>	<u>159</u>	<u>Chapter 5, “Import Commands”</u>
<u>importOption</u>	<u>160</u>	<u>Chapter 5, “Import Commands”</u>
<u>include</u>	<u>751</u>	<u>Chapter 12, “Common Power Format Commands”</u>
<u>instanceFV</u>	<u>276</u>	<u>Chapter 8, “User Data Commands”</u>
<u>instanceStub</u>	<u>278</u>	<u>Chapter 8, “User Data Commands”</u>
<u>ipbAssign</u>	<u>374</u>	<u>Chapter 9, “IP Commands”</u>
<u>iplInstAssign</u>	<u>375</u>	<u>Chapter 9, “IP Commands”</u>
<u>keepNet</u>	<u>282</u>	<u>Chapter 8, “User Data Commands”</u>
<u>linkRefLib</u>	<u>165</u>	<u>Chapter 5, “Import Commands”</u>
<u>lp (Low-power Compile-time Command)</u>	<u>284</u>	<u>Chapter 8, “User Data Commands”</u>
<u>memoryTransform</u>	<u>290</u>	<u>Chapter 8, “User Data Commands”</u>
<u>memWritethru</u>	<u>295</u>	<u>Chapter 8, “User Data Commands”</u>
<u>moduleFV</u>	<u>296</u>	<u>Chapter 8, “User Data Commands”</u>
<u>moduleNet</u>	<u>202</u>	<u>Chapter 7, “Database Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-6 Alphabetical List of xeCompile Commands, *continued*

Command Name	Page	See Chapter
<u>moduleStub</u>	<u>297</u>	<u>Chapter 8, "User Data Commands"</u>
<u>multiplexCable</u>	<u>298</u>	<u>Chapter 8, "User Data Commands"</u>
<u>multiSampledTerminal</u>	<u>300</u>	<u>Chapter 8, "User Data Commands"</u>
<u>netlistFile</u>	<u>167</u>	<u>Chapter 5, "Import Commands"</u>
<u>netWeakDrive</u>	<u>301</u>	<u>Chapter 8, "User Data Commands"</u>
<u>noOverlapClocks</u>	<u>304</u>	<u>Chapter 8, "User Data Commands"</u>
<u>precompile</u>	<u>710</u>	<u>Chapter 11, "Compile-time Commands in ICE Mode"</u>
<u>precompileOption</u>	<u>306</u>	<u>Chapter 8, "User Data Commands"</u>
<u>probe</u>	<u>336</u>	<u>Chapter 8, "User Data Commands"</u>
<u>readCPFfile</u>	<u>712</u>	<u>Chapter 11, "Compile-time Commands in ICE Mode"</u>
<u>referenceMap</u>	<u>717</u>	<u>Chapter 11, "Compile-time Commands in ICE Mode"</u>
<u>refLib</u>	<u>169</u>	<u>Chapter 5, "Import Commands"</u>
<u>set_cpf_version</u>	<u>752</u>	<u>Chapter 12, "Common Power Format Commands"</u>
<u>set_design</u>	<u>753</u>	<u>Chapter 12, "Common Power Format Commands"</u>
<u>set_hierarchy_separator</u>	<u>756</u>	<u>Chapter 12, "Common Power Format Commands"</u>
<u>set_instance</u>	<u>757</u>	<u>Chapter 12, "Common Power Format Commands"</u>
<u>set_macro_model</u>	<u>760</u>	<u>Chapter 12, "Common Power Format Commands"</u>
<u>shadowsData</u>	<u>340</u>	<u>Chapter 8, "User Data Commands"</u>
<u>shrinkDB</u>	<u>718</u>	<u>Chapter 11, "Compile-time Commands in ICE Mode"</u>
<u>stateRetentionControl</u>	<u>342</u>	<u>Chapter 8, "User Data Commands"</u>
<u>symmetricConfiguration</u>	<u>343</u>	<u>Chapter 8, "User Data Commands"</u>

UXE Command Reference Manual

UXE Command List

Table 2-6 Alphabetical List of xeCompile Commands, *continued*

Command Name	Page	See Chapter
terminalAssign	345	Chapter 8, “User Data Commands”
terminalTiming	352	Chapter 8, “User Data Commands”
terminalWeakDrive	362	Chapter 8, “User Data Commands”
tieNet	364	Chapter 8, “User Data Commands”
ttcClear	365	Chapter 8, “User Data Commands”
userData	366	Chapter 8, “User Data Commands”

xeDebug Command Set for ICE Mode

This section lists the XEL commands available in the xeDebug tool when the design is compiled in ICE mode, and refers to the location in this manual where more information about the command can be found. This set of commands includes the adapted NC-Sim commands and the emulation-specific commands.

Table 2-7 Alphabetical List of Run-time Commands

Command Name	Page	See Chapter
clockConfig	794	Chapter 13, “Run-time Commands”
configPM	801	Chapter 13, “Run-time Commands”
convertRadix	804	Chapter 13, “Run-time Commands”
database	806	Chapter 13, “Run-time Commands”
debug	813	Chapter 13, “Run-time Commands”
deposit	815	Chapter 13, “Run-time Commands”
download	819	Chapter 13, “Run-time Commands”
dpa	820	Chapter 13, “Run-time Commands”
force	842	Chapter 13, “Run-time Commands”
genPowerProfile	846	Chapter 13, “Run-time Commands”
getCableStatus	849	Chapter 13, “Run-time Commands”
getCycleNum	854	Chapter 13, “Run-time Commands”

UXE Command Reference Manual

UXE Command List

Table 2-7 Alphabetical List of Run-time Commands, *continued*

Command Name	Page	See Chapter
getHostStatus	855	Chapter 13, “Run-time Commands”
getPMStatus	856	Chapter 13, “Run-time Commands”
getTime	859	Chapter 13, “Run-time Commands”
hdlConfig	861	Chapter 13, “Run-time Commands”
hdlRunLogFile	865	Chapter 13, “Run-time Commands”
host	866	Chapter 13, “Run-time Commands”
infiniTrace	872	Chapter 13, “Run-time Commands”
lp (Low-power Run-time Command)	882	Chapter 13, “Run-time Commands”
memory (in ICE Mode)	911	Chapter 13, “Run-time Commands”
probe	939	Chapter 13, “Run-time Commands”
release	955	Chapter 13, “Run-time Commands”
reset	956	Chapter 13, “Run-time Commands”
resetTime	957	Chapter 13, “Run-time Commands”
restart	958	Chapter 13, “Run-time Commands”
resultWaveform	961	Chapter 13, “Run-time Commands”
resume	962	Chapter 13, “Run-time Commands”
run	963	Chapter 13, “Run-time Commands”
save	967	Chapter 13, “Run-time Commands”
scope	971	Chapter 13, “Run-time Commands”
sdl	973	Chapter 13, “Run-time Commands”
stop	1011	Chapter 13, “Run-time Commands”
suspend	1013	Chapter 13, “Run-time Commands”
symbol	1014	Chapter 13, “Run-time Commands”
tca	1016	Chapter 13, “Run-time Commands”
userData	1023	Chapter 13, “Run-time Commands”
value	1025	Chapter 13, “Run-time Commands”
vector	1028	Chapter 13, “Run-time Commands”

UXE Command Reference Manual

UXE Command List

Table 2-7 Alphabetical List of Run-time Commands, *continued*

Command Name	Page	See Chapter
waitWhileBusy	1033	Chapter 13, “Run-time Commands”
xeset	1035	Chapter 13, “Run-time Commands”
xogui	1081	Chapter 13, “Run-time Commands”

XEL Run-time Parameters

This section lists the XEL run-time parameters in the xeDebug tool, and refers to the location in this manual where you can find more information about the run-time parameter.

Some of these run-time parameters are set automatically, and some can be set manually using the [xeset](#) command described in the [Chapter 13, “Run-time Commands.”](#) Additionally, you can view the current value of any XEL run-time parameter.

Table 2-8 Alphabetical List of XEL Run-time Parameters

Parameter Name	Page	See Chapter
database	1041	Chapter 13, “Run-time Commands”
dbHwType	1041	Chapter 13, “Run-time Commands”
dbPath	1041	Chapter 13, “Run-time Commands”
depositNCsimMode	1042	Chapter 13, “Run-time Commands”
design	1043	Chapter 13, “Run-time Commands”
endOfVector	1044	Chapter 13, “Run-time Commands”
host	1054	Chapter 13, “Run-time Commands”
linkFiles	1058	Chapter 13, “Run-time Commands”
keepHostAlive	1055	Chapter 13, “Run-time Commands”
keepHostAliveTimeout	1057	Chapter 13, “Run-time Commands”
maxMismatches	1059	Chapter 13, “Run-time Commands”
msgMaxLen	1060	Chapter 13, “Run-time Commands”
mtHost	1060	Chapter 13, “Run-time Commands”
postTriggerSamples	1061	Chapter 13, “Run-time Commands”

UXE Command Reference Manual

UXE Command List

Table 2-8 Alphabetical List of XEL Run-time Parameters

Parameter Name	Page	See Chapter
<u>probePrimary</u>	<u>1063</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>retryDownload</u>	<u>1066</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>retryInterval</u>	<u>1066</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>sdlIncludePath</u>	<u>1067</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>signalRadix</u>	<u>1068</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>stopDelay</u>	<u>1069</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>suspendEnable</u>	<u>1070</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>-gui svUploadOnProbe</u>	<u>1052</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>timeUnit</u>	<u>1072</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>traceMemMax</u>	<u>1073</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>traceMemSize</u>	<u>1074</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>triggerPos</u>	<u>1075</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>vector</u>	<u>1079</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>version</u>	<u>1080</u>	<u>Chapter 13, “Run-time Commands”</u>

Common XEL Commands

This section lists the XEL commands that are available in both xeCompile and xeDebug tools, and refers to the location in this manual where more information about the command can be found.

Table 2-9 Alphabetical List of Common Commands

Command Name	Page	See Chapter
<u>exit</u>	<u>148</u>	<u>Chapter 4, “General-purpose Commands”</u>
<u>help</u>	<u>149</u>	<u>Chapter 4, “General-purpose Commands”</u>
<u>msgControl</u>	<u>150</u>	<u>Chapter 4, “General-purpose Commands”</u>
<u>quit</u>	<u>151</u>	<u>Chapter 4, “General-purpose Commands”</u>
<u>redirect</u>	<u>152</u>	<u>Chapter 4, “General-purpose Commands”</u>

vvmDebug Command Set

The following table provides information about the support for the XEL commands with the vvmDebug tool. The table lists all the commands that are partially supported and not supported with vvmDebug. All the commands that do not appear in the following table are fully supported with vvmDebug.

Table 2-10 Alphabetical List of Run-time Commands

Command Name	Supported / Not Supported	Page	See Chapter
<u>assertion</u>	Not Supported	<u>780</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>clockConfig</u>	Not Supported	<u>794</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>configPM</u>	Not Supported	<u>801</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>database</u>	Partially Supported Not Supported: ■ <u>-listphyfile</u> <u>[<phy fileName> ...]</u> ■ <u>-prepareOffline</u> ■ <u>-captureMode [controlled uncontrolled]</u> ■ <u>-continuousupload</u> ■ <u>-phy</u>	<u>806</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>deposit</u>	Not Supported	<u>815</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>download</u>	Not Supported	<u>819</u>	

UXE Command Reference Manual
UXE Command List

Table 2-10 Alphabetical List of Run-time Commands, *continued*

Command Name	Supported / Not Supported	Page	See Chapter
<u>dpa</u>	Partially Supported Not Supported Options: <ul style="list-style-type: none">■ <u>-phyfile <phy filename></u>■ <u>-listphyfile [<phy fileName> ...]</u>■ <u>-sparse sampling [<N>]</u>	<u>820</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>force</u>	Not Supported	<u>842</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getCableStatus</u>	Not Supported	<u>849</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getHostStatus</u>	Not Supported	<u>855</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>getPMStatus</u>	Not Supported	<u>856</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>hdlConfig</u>	Not Supported	<u>861</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>hdlRunLogFile</u>	Not Supported	<u>865</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>host</u>	Not Supported	<u>866</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>infiniTrace</u>	Not Supported	<u>872</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>iRun</u>	Not Supported	<u>881</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>lp (Low-power Run-time Command)</u>	Not Supported	<u>882</u>	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-10 Alphabetical List of Run-time Commands, *continued*

Command Name	Supported / Not Supported	Page	See Chapter
<u>memory (in SA Mode)</u>	Partially Supported	<u>911</u>	<u>Chapter 13, “Run-time Commands”</u>
<u>memory (in ICE Mode)</u>	Not Supported Options: <ul style="list-style-type: none"> ■ <u>-load {[<file format>] <memory name> -file <file name> [-retainValue] [-retainValueOffset]}</u> ■ <u>-set {-all <memory name>...}</u> ■ <u>-reset {-all <memory name>...}</u> ■ <u>-setvalue {-all <list of instances>...} [-start <addr1>] [-end <addr2>]</u> ■ <u>-getFileOffset <file name></u> ■ <u>-sparse mem info [-v] <mem name></u> ■ <u>-sparse mem info -usedpages [-v] <mem name></u> ■ <u>-sparse mem info -statistic [-v] [-byusage]</u> ■ <u>-sparse mem info -overflow</u> 		
<u>ncsim</u>	Not Supported	<u>937</u>	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-10 Alphabetical List of Run-time Commands, *continued*

Command Name	Supported / Not Supported	Page	See Chapter
<u>probe</u>	Partially Supported: Not Supported: Note: In vvmDebug, the probe command options for CPF and IEEE 1801 are not supported.	939	<u>Chapter 13, “Run-time Commands”</u>
<u>release</u>	Not Supported	955	<u>Chapter 13, “Run-time Commands”</u>
<u>reset</u>	Not Supported	956	<u>Chapter 13, “Run-time Commands”</u>
<u>resetTime</u>	Not Supported	957	<u>Chapter 13, “Run-time Commands”</u>
<u>resultWaveform</u>	Not Supported	961	<u>Chapter 13, “Run-time Commands”</u>
<u>resume</u>	Not Supported	962	<u>Chapter 13, “Run-time Commands”</u>
<u>run</u>	Partially Supported Not Supported: ■ <u>-wait</u> ■ <u>-nowait</u>	963	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-10 Alphabetical List of Run-time Commands, *continued*

Command Name	Supported / Not Supported	Page	See Chapter
<u>save</u>	Partially Supported Not Supported: <ul style="list-style-type: none">■ <u>-commands</u>■ <u>-overwrite</u>■ <u>[-simulation]</u> <u><snapshot name></u>■ <u>-uservars <filename></u> Note: In VVM, only the -environment option is supported to save the last debug environment.	967	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-10 Alphabetical List of Run-time Commands, *continued*

Command Name	Supported / Not Supported	Page	See Chapter
<u>sdl</u>	Partially Supported Not Supported: <ul style="list-style-type: none"> ■ sdl -getAssertions ■ sdl -getLabels ... ■ sdl -haltOnTrigger ... ■ sdl [-ldcnt1 -ldcnt2] ... ■ sdl -log ... ■ sdl -ncbreakpoints ... ■ sdl -onlyReport ... ■ sdl -restoreState ... ■ sdl -saveState ... ■ sdl -stop ■ sdl -traceClear ... ■ sdl -traceDump ... ■ sdl -traceOn ... ■ sdl -tx ... 	973	<u>Chapter 13, “Run-time Commands”</u>
<u>stop</u>	Partially Supported Not Supported: <u>-emulation</u>	1011	<u>Chapter 13, “Run-time Commands”</u>
<u>suspend</u>	Not Supported	1013	<u>Chapter 13, “Run-time Commands”</u>
<u>vector</u>	Not Supported	1028	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual

UXE Command List

Table 2-10 Alphabetical List of Run-time Commands, *continued*

Command Name	Supported / Not Supported	Page	See Chapter
<u>xeset</u>	Partially Supported Not Supported: <u>appendTrace</u>	1035	<u>Chapter 13, “Run-time Commands”</u>

UXE Command Reference Manual
UXE Command List

Programs and Utilities

This chapter describes programs and utilities that can be invoked from the UNIX command line prompt.

The following programs and utilities are discussed in this chapter:

- “[generateDYNPwaveform](#)” on page 102
- “[ipBuilder](#)” on page 103
- “[ModelChecker](#)” on page 104
- “[gateCount](#)” on page 106
- “[libTo1801](#)” on page 107
- “[primCount](#)” on page 110
- “[test_server](#)” on page 111
- “[vCompare](#)” on page 121
- “[verigen](#)” on page 123
- “[vvmDebug](#)” on page 126
- “[fvCompute](#)” on page 130
- “[xeCompile](#)” on page 132
- “[xeDebug](#)” on page 135
- “[xeclient](#)” on page 141
- “[xeGenReport](#)” on page 142

generateDYNPwaveform

Generates the FSDB and SST2 waveforms in the DYNP (Dynamic Probes) offline mode. You must run this command in the design directory, and specify the name of the *.phy file and the output waveform file name. This command opens an offline session to generate the waveform. All successfully added probes during the online session are saved to the *.phy file. Therefore, the waveform for all probes added in the online session can be generated offline.

The syntax for this command is:

```
generateDYNPwaveform [-fsdb | -sst2] <phy_name> <waveform_filename> [-session  
                        <session_number>]
```

[-fsdb | -sst2]

Specifies the waveform type, that is FSDB or SST2, to be generated during the offline session. The default format is SST2.

<phy_name>

Specifies the name of the *.phy file containing the probes added during the online session.

<waveform_filename>

Specifies the name of the waveform file to be generated as an output of the generateDYNPwaveform command.

[-session <session_number>]

Specifies the session directory to run the offline session. This parameter is optional.

ipBuilder

Generates an IP front-end mapping (.fem) file for use in IP mapping by the compiler. You are responsible for supplying the input file contents for the IP Builder.

The output data of the terminal cell to interposer pin mapping is stored in the file *<ip-id>.fem*

Run this program before running `icePrepare`.

The syntax for this program is:

`ipBuilder <input_file>`

<input_file>

Name of the file to be mapped.

ModelChecker

Invokes the ModelChecker program from the UNIX command prompt to find the faults in the compiler model before run-time failures occur due to them.

The syntax for this program is:

`ModelChecker <design_name> [options]`

Note: Using the following equivalent set of commands, the ModelChecker program can be automatically started from within the normal compile flow as well:

- `compilerOption -add {ModelChecker OFF | ON}`
- `compilerOption -add {write_pgm OFF | ON}`
- `compilerOption -add {write_fnets OFF | ON}`
- `compilerOption -add {ModelCheckerOmit "<check_type> <check_type> ..."}`

<design_name>

Name of the final .proto file without its extension.

[options]

Specify any of the following valid options:

`-write_pgm` Dumps an ASCII view of the compiler's program on the disk to the `<design>.et5pgm` file.

`-write_fnets` Dumps an ASCII mapping of post-synthesized netlist names to data store locations within the emulation processors to the `et3compile.msg` file.

UXE Command Reference Manual

Programs and Utilities

`-omit [<check_type>]` Instructs not to perform the specified types of checks. The `<check_type>` option can have either of the following values:

- LOGIC
- LATCH
- XBIO
- DBIO
- MEM
- CONNECTIVITY

Note: The intent of the `-omit [<check_type>]` option is to turn off certain checks that might be known to be passing, but take a significant duration of time to complete, such as LOGIC check type.

For detailed information about the check types, refer to the *Finding Faults using ModelChecker* section of the *Working with the Palladium XP Emulators* chapter in *UXE User Guide*.

gateCount

Generates a report showing the gate count of a specified module, and the gate counts of all the user-defined modules instantiated at any level below the specified module. The 2-input gate count for the design reported by precompile (before any optimization) is normally very close to the gate count for the design top module reported by the `gateCount` program.

Use the `designImport` command to import the netlists into the design directory. Then, invoke the `gateCount` program from the UNIX command line.

The syntax for this program is:

```
gateCount [<options>] <libName> <cellName>
```

Invoke the program with no arguments to get a list of supported options.

libTo1801

UXE provides a utility, called libTo1801, to extract low power information from liberty files and generate the corresponding IEEE 1801 commands. The commands are saved into an IEEE 1801 file, which can be read with the other 1801 files during compilation. The libTo1801 command can be executed either from the UNIX or xeCompile command prompt.

Note: The libTo1801 command is supported in both ICE and SA modes.

The libTo1801 command translates the following information from the liberty files into the IEEE 1801 commands:

- List of cell names
- The pg_pin and pg_type information within each cell
- The related_power_pin of a data pin within each cell
- The related_ground_pin of a data pin within each cell
- The power_down_function of a data pin within each cell

Note: If the pin attributes related_power_pin, related_ground_pin, power_down_function, appear in a bus group in the liberty files, the attributes are honored for the whole bus. However, the specific pin's attributes have higher priority in the bus group.

- Cell or pin related liberty file name and line number

The generated .upf file, original liberty file name, and line number appear in the read_power_intent command and precompile output messages.

- The switch_function and pg_function information
- The bus bit pin information

For example, UXE picks both Enable and A[7] if a liberty file has the following information:

```
pin(Enable) {...}
bus(A) ...
    pin(A[7]) {...}
    ...}
```

- For duplicate cell definitions in liberty files, the first cell definition is honored and the next duplicate cell definition is ignored with warning messages.

UXE Command Reference Manual

Programs and Utilities

If an attribute is defined in both the IEEE 1801 files and liberty files, the IEEE 1801 files take precedence. The precompile gives warning messages to indicate that the information from liberty files is ignored and the information from the IEEE 1801 files is taken.

The syntax of the command is:

```
libTo1801 {<libertyFilename> | <libertyDirectory>}... -o <1801Filename>
```

<libertyFilename> | <libertyDirectory>

Specifies the name of a liberty file or directory where the liberty files are located. You can specify multiple liberty files or directories. If you specify the liberty directory, all liberty files present in the directory are read to extract the low power information and you need not specify the liberty files explicitly. The `-o` option generates output into the specified file.

For example,

```
libTo1801 a.lib b.lib basicLib -o liberty.upf
```

Here, `a.lib` and `b.lib` are liberty files, `basicLib` is a directory, and `liberty.upf` is an output 1801 file.

The supported liberty low power information is extracted from `a.lib`, `b.lib`, and all liberty files under `basicLib`. The extracted information is saved in the `liberty.upf` file.

In the `read_power_intent` command, you must include the translated 1801 file manually by passing it as the first file. For example,

```
read_power_intent <1801Filename> <filename>
```

In SA mode, invoke IXCIM or `iRun -hw` with the additional `-lps_lib_mfile <filename>` option.

The `-lps_lib_mfile` option extracts power information from a liberty model library while explicitly specifying the name of the liberty model file to use.

In `-lps_lib_mfile <filename>`, the `<filename>` is different from `<libertyFileName>` specified in the `libTo1801` command. In the `libTo1801` command, `<libertyFileName>` refers to the actual liberty file name such as `a.lib`, `b.lib`, and so on.

However, in `-lps_lib_mfile <fileName>`, the `<fileName>` refers to a file, which consists of all the liberty file names. For example,

UXE Command Reference Manual

Programs and Utilities

To extract low-power information from liberty files `a.lib` and `b.lib`, create a file, for example, `libfile` with "`a.lib b.lib`" as the file content. Here, `<filename>` is `libfile`.

For more information about the `irun -lps_lib_mfile <filename>` command, refer to the *Low-Power Simulation Guide (IEEE 1801)*

primCount

Generates a report showing the number of `qtref` library primitives instantiated directly or indirectly within a specified module. The report also shows, for each user-defined module instantiated at any level below the specified module, the number of `qtref` library primitives within the given user-defined module.

Note: The number of primitives is not directly related to the 2-input gate count reported by `precompile`, nor to the number of ET primitives reported by `et3compile`. To get a report of gate counts similar to the precompile 2-input gate count, use the [gateCount](#) program.

Use the [designImport](#) command to import the netlists into the design directory. Then, invoke the `primCount` program from the UNIX command line.

The syntax for this program is:

```
primCount [<options>] <libName> <cellName>
```

Invoke the program with no arguments to get a list of supported options.

test_server

Enables you to check the host and domain availability at run time, and reserve emulator resources for downloading the designs.

Without options, executing `test_server` returns all the boards and domains of the emulation system, providing detailed information for each domain, such as the user name, the user process ID (PID), the SA host name, the cable connections, the design name, and the elapsed time since DBEngine is running. The default functionality is also supported by clicking the *Check* button in the xeDebug GUI.

The syntax for this program is:

```
test_server [<top_cell_name>] |
test_server <top_cell_name> -reserve [-key <key_value>] [-timeout <t> [<unit>] | \
    -holdtime <t> [<unit>] [-retry [<num>]] [-bp] \
    [-sahost <host>] [-keyfile <reservationFilePath>] [-firstfit] |
test_server <top_cell_name> -location [-all] [-sahost <host>] [-nosahost] [-host
    <host>] |
test_server -rmkey [<key_value>] |
test_server -hdver |
test_server -json [-output <jsonFile>]
test_server -l <num_boards> <starting_board> [<top_cell_name>] |
test_server -l <top_cell_name> |
test_server -d <b.d> |
test_server -notime |
test_server -update [<sec>] |
test_server -sainfo [-update [<sec>]] |
test_server -listUxeData |
test_server [-moreinfo] [-noclean] |
test_server -rsh <host> | -ssh <host>
test_server -help]
```

The standard output, when the `test_server` program is executed without options, is similar to the following:

Configmgr Host: emHost1	Hardware: Palladium XP XL	System Status: ONLINE
Board 0 has 4 modules	Board: ONLINE	NCC: ONLINE
HSMA Status: Operational	NCC Version: RELEASE 5.9.3.8	
SA Hosts:	emHost1(0)	

UXE Command Reference Manual

Programs and Utilities

Domain	Owner	PID	BCA	J-con	MemCard	Design	ElapTime
0	NONE	0	0	1 2	--	NONE	--
1	NONE	0	--	--	0 1 2 3	NONE	--
2	user1	emHost1:18541	--	--	--	AdderTop	00:02:10
3	NONE	0	--	--	0 2	NONE	--

Exit code: 0

<top_cell_name>

Tests the specified design prototype for download, and checks the possible configurations into which the design can fit on the hardware. A message is returned to inform you about the number of ways in which the design can fit in the current emulator hardware configuration.

The presence of a *<top_cell_name>.bp* or *<top_cell_proto>.br* file is also taken into account. If both **.bp* (board preference) and **.br* (board restriction) files are present, they will be combined, provided there is no conflict.

Note: The treatment of **.bp* and **.br* files mentioned above applies only to running the *test_server* program. If both files are present during download, the **.br* file takes precedence over the **.bp* file.

The file specified by *<top_cell_name>* must exist in the *./dbFiles* directory.

For example:

```
test_server counter  
Design counter can fit in emulator cva-sb2503 in 4 different way.
```

- If the design can fit into the emulator and be downloaded right away, the *test_server <top_cell_name>* program exits with status 0.
- If the design can fit into the emulator but the needed resources are occupied by another job, the *test_server <top_cell_name>* program exits with status 1.
- If the design cannot fit into the emulator even when all resources are free, the *test_server <top_cell_name>* program exits with status 99. You also see a message explaining why the resources not available, recompile is necessary, and so on.

For example:

```
> test_server counter  
> Design counter cannot fit in the emulator cva-eng2. Either recompile the  
design or reconfigure the emulator.
```

<top_cell_name> -reserve [-key <key_value>] [-timeout <t> [<unit>] |

-holdtime <t> [<unit>]] [-retry [<num>]] [-bp]

[-sahost <host>] [-keyfile <reservationFilePath>] [-firstfit]

Reserves the resources needed for the specified design, *<top_cell_name>*. This program also returns the status, where 0 indicates successful reservation of resources and 99 indicates failure.

Important

The first three arguments of the resource reservation program, that is, `test_server <top_cell_name> -reserve`, must be entered in the fixed order. Other options after `-reserve` can be specified in any sequence. Also, the specified *<top_cell_name>* file must exist in the `./dbFiles` directory.

Use the `-key <key_value>` option to specify a positive integer as the key value. The key value must be less than 2147483647, which is the maximum value of signed 32bit integer.

If any of the following situations occur, the `test_server` program automatically generates a random key value for the reserved resources and does not return any warning message:

- The specified key value is greater than 2147483647 or invalid.
- The `-key` option is not specified.

Note: When two users, say from two different shells or terminals, use the same key to reserve the resources, a warning message is generated.

Use either of the following two options to define the required resource reservation mode:

- `-timeout` option to reserve the resources for single-session mode
- `-holdtime` option to reserve the resources for multi-session mode

The *<t>* option defines the time limit for the single-session or multi-session mode. This option can be followed by the *<unit>* option to specify the unit of time. The valid values for the *<unit>* option are sec (second), min (minute), or hour. The default unit of time is min.

The value specified for *<t>* option must be a positive integer that is less than either 24 hours, 86400 seconds, or 1440 minutes. If both `-timeout` and `-holdtime` options are not specified, 5 min is used as the default timeout value. This means that the resources will be reserved for five minutes in single-session mode.

Specify the `-retry [<num>]` option when the required resources are not available or operational. Consequently, the `test_server` program will retry to reserve the required resources once every few seconds. The `<num>` option should be a positive integer used to specify the auto-retry number. If you do not specify the `<num>` option, the `test_server` program retries endlessly until the resource reservation succeeds, or you enter `abort` or press `Ctrl+C`.

If a `.bp` file exists in the design directory, only the starting board or domain from that `.bp` file is used for reserving the emulation host resources.

Use the `-bp` option when the `test_server` program needs to be run from a directory other than the design directory.

When the `-bp` option is omitted, the `test_server` program can only be run from the design directory. If a `.bp` file does not exist in the design directory, the information from compile time is used for reserving the emulation host resources.

The Configmgr analyzes the locations specified in the `.bp` file and determines the best fit locations and scores the locations accordingly to assign the jobs. If, however, you have created a preference list in the `.bp` file with the `PRIORITY_FIT` keyword, the preference order for placing the jobs is the order in which the domains are listed in the `.bp` file as opposed to the score based ordering. For information on how to create a preference list, refer to the *Using a .bp File for Resource Reservation* section in the UXE User Guide.

If a `.bp` file exists in the design directory, it is used for the starting domain and reserving all required domains or boards using the information from compile time. This means that if the `.bp` file specifies more than 1 domain or 1 board, the first item is used as the starting domain or board, and the remaining items in the `.bp` file are ignored.

The `-sahost <host>` option can be used to specify host of a SA card.

The `-keyfile <reservationFilePath>` option can be used to specify the location of a file for reservation. This option provides the same functionality as provided by the `-keyfile <reservationFilePath>` option of the `host` command.

Note: If the `-keyfile <reservationFilePath>` option of the `test_server` is provided, the option should also be provided to the `host` command. Both paths should be consistent.

The `-firstfit` option can be used to reserve domains using the first fit algorithm. This option provides the same functionality as provided by the `-firstfit` option of the `host` command.

<top_cell_name> -location [-all] [-sahost <host>] [-nosahost] [-host <host>]

Returns a list of domains with the best fit domain displayed at the top.

The *<top_cell_name>* option defines the top design cell that needs to be downloaded to the best fit domain.

Use the *-all* option to display the locations currently not available because of locked resources. The *-sahost <host>* option can be used to specify the host of a SA card.

Use the *-nosahost* option to ignore the SA card requirement.

Note: This command will execute correctly only if *<top_cell_name>.proto* file exists in the dbFiles directory.

For example:

```
XE> test_server counter -location
Domain List
1.8
1.10
1.12
1.14
1.6
1.0
1.4
```

In the printed result, the domain list indicates the location where the design will be placed. The domain at the top of the list, such as 1.8 in the above example, is considered to be the location where the job will be placed by the configmgr if locking was enabled.

The *-host <host>* option returns the locations to download the design on the specified remote host. This option is supported only with the *-location* option.

-rmkey [<key_value>]

Releases the resources reserved for the specified design.

When the *<key_value>* option is specified, the resources corresponding to specified key value are released from the emulator. This option can be used when resources were reserved from a directory other than the design directory (for example, using the *test_server <top_cell_name> -bp* program).

Note: Multiple key values can be specified with one `-rmkey` option.

-hdver

Prints the hardware version of the emulator.

For example:

```
>test_server -hdver  
Palladium XP XL
```

Note: If the emulator is not a Palladium-XP XL or Palladium-XP GXL system, an error is generated.

-json [-output <jsonFile>]

Displays the result in JSON format.

Use the `-output <jsonFile>` option to specify the name of the file in which the result is generated in JSON format. If this option is not specified, the result is printed on stdout.

-l <num_boards> <starting_board> [<top_cell_name>]

Shows whether the specified boards or domains are available for the design to be downloaded. In the `test_server` output, if the resource is available, the `Exit code` field returns the value as 0; else, 99 is returned.

The `<num_boards>` parameter indicates the total number of boards to check. A Palladium-XP XL system can have maximum 4 boards that are numbered from 0 to 3. A Palladium-XP GXL system can have maximum 32 boards that are numbered from 0 to 31.

The `<starting_board>` parameter indicates the starting board logical number (for example, 0).

Note that, even if one domain in the specified board is in use, the whole board is shown as unavailable. In such cases, you can use the `proto.bp` file to specify individual domains in the board.

If only `<num_boards>` and `<starting_board>` are specified, and `<top_cell_name>` is not given, the specified boards are checked for availability. The `<num_boards>` argument specifies the number of total boards to be checked for availability. Two numerical arguments are expected, or an error is generated. If any domain on a board is busy, the board is unavailable.

UXE Command Reference Manual

Programs and Utilities

The program returns a line indicating resource availability, followed by a line for each specified board, telling its status. Status can be available, not available, offline, or does not exist.

For example:

```
> test_server -l 4 2
Requested board domain resources NOT AVAILABLE. Total usable boards:3
Board 1: available
Board 2: available
Board 3: offline
Board 4: does not exist
Board 5: does not exist
```

If the *<top_cell_name>* is specified, you can use a *.bp or *.br file and specify particular domains or boards to check for resource availability. These must be ASCII files in the current design directory. If the *<num_boards>* and *<starting_board>* options are given, but the *<top_cell_proto>.bp/.br* file name is also specified as optional argument, the *<num_boards>* and *<starting_board>* entries are ignored, and the board/domain information is obtained from the specified *.bp or *.br file.

You can also specify just the *<top_cell_name>* argument, with this *-l* option. In this case, if there are some invalid entries, the `test_server` program exits with an error.

When *<num_boards>* and *<starting_board>* are also specified before the *<top_cell_name>* argument, in case of invalid entries in the *<top_cell_name>.bp/.br* file, this last argument is ignored with a warning, and the *<num_boards>* and *<starting_board>* entries are used to check for availability.

-l <top_cell_name>

Shows whether the boards or domains specified in the *<top_cell_name>.bp* file are available for the design to be downloaded. In the `test_server` output, if the resource is available, the `Exit code` field returns the value as 0; else, 99 is returned.

-d <b.d>

Returns only the specified boards and domains. The option accepts only one argument after the *-d* option. If multiple boards/domains are to be supplied, all entries must be provided in double quotes, as shown in example below:

```
> test_server -d "1.1-1.3"
```

-notime

Returns the status of all the boards and domains of the emulation system except for the elapsed time since the DBEngine process is running (that is, the *ElapTime* column).

-update [<sec>]

Returns the detailed status of the emulation system and updates it at regular intervals based on the time specified in seconds, <sec>.

If the <sec> option is not specified, the returned status is updated after every 30 seconds, by default.

-sainfo [-update [<sec>]]

Prints information about each Orion host connected to the emulator. Use the -update option to update the emulator status at every <sec> seconds. By default, the value for <sec> is 30 seconds.

For each slot of Orion host, the `test_server -sainfo` command reports the following:

- A unique identifier (1, 2, 3, 4) for each SA card (not the domain in use)
- Host name on which the SA card is installed
- NCC_port that connects to the SA Card
- Status of the SA card (Online or Offline)
- Slot number that the job occupies
- PID on the Orion host that locks the job-slot
- Username associated with each job-slot that is in use
- Elapsed time of the job

For example:

	Card	Host	NCC_port	Status	Slot	PID	User_Name	Elapsed_Time
1		host1	-	Online	-	-	-	-
2		host1	-	Offline	-	-	-	-
3		host2	-	Online	-	-	-	-
4		host3	4	Online	1	30613	user1	367
5		host4	0	Online	1	345	user2	78

UXE Command Reference Manual

Programs and Utilities

```
5      host4    1      Online   2     456    user3    908
```

In this sample output, there are four Orion hosts. Where, `host1` has two SA cards installed (refer to rows 1 and 2 in the example), `host2` to `host4` have one SA card installed. The second SA card of `host1` is offline, while others are online. There is one user using the SA card from `host3` (refer to row 4 in the example), and two users using SA card from `host4` (refer to rows 5 and 6 in the example).

-listUxeData

Lists the xeDebug PID, UXE host, NCSim PID and NCSim host for each job currently run on the emulator. The `test_server -listUxeData` command needs to be run on configmgr host.

If `test_server` is not run on configmgr host, the information will not be displayed.

-moreinfo

Returns the warnings and information messages with the detailed status of the emulation system.

For example,

```
10 ET3CONFIG | WARNING: User requested to skip building of board wire tables. MB APIs not operational.  
10 ET3_API | There is only one configuration found and it has no load module. A design proto was not loaded.  
Configmgr Host: emHost1          Hardware: Palladium XP XL    System Status: ONLINE  
Board 0 has 4 modules           Board: ONLINE             NCC: ONLINE  
HSMA Status: Operational        NCC Version: RELEASE 5.9.3.8  
SA Hosts:                      emHost1(0)  
Domain  Owner          PID  BCA  J-con  MemCard  Design  ElapTime  
      0    NONE          0    0    1 2      --      NONE    --  
      1    NONE          0    --    --      0 1 2 3      NONE    --  
      2    user1         emHost1:18541  --    --      --      AdderTop 00:02:10  
      3    NONE          0    --    --      0 2      NONE    --  
Exit code: 0
```

-noclean

Does not delete the files created by the first execution of the `test_server` program.

-rsh <host> | -ssh <host>

Enables you to run the test_server program on the specified remote host. The -ssh <host> enables you to run the test_server program after a secure login to the specified remote host. You can find out the best locations to download the designs on a remote host without logging on to the remote host. For details, refer to the *Reserving Emulation Hosts* section in the UXE User Guide.

-help

Returns the help message summary for test_server. You can also use -h, -usage, and -u to display the help information.

vCompare

Records the mismatches in the Vector Debug mode. This utility is used along with the maxMismatches run-time parameter to increase the limit when the maximum number of mismatches is reached.

The syntax for this utility is:

```
vCompare -in <stim filename> -result <res filename> -log <log filename>
[ -comp <comp filename> ]
[ -start <vecNumber> ]
[ -count <numVectors> ]
[ -interact [Y | N] ]
[ -mm <maxMismatches> ]
```

-in <stim_filename>

Specifies the stimulus file name.

-result <res_filename>

Specifies the result file name.

-log <log_filename>

Specifies the file in which failure log will be saved.

-comp <comp_filename>

Specifies the file in which the first failed cycle will be saved.

-start <vecNumber>

Specifies the starting vector number in the result file.

-count <numVectors>

Specifies the number of vectors to compare.

-interact [Y | N]

Specifies whether or not the vectors should interact.

-mm <maxMismatches>

Specifies that the comparison will stop when number of cycles with mismatches reaches the specified number.

verigen

Generates a file containing information about the hierarchical location and connections for an instance or net in the emulation design database. The Novas tools use the instance and signal names in this Verilog file to display the schematics and waveforms for a specific instance or signal in the emulation design.

This is a UNIX executable that can also be used as an XEL command.

The syntax for this utility is:

```
verigen <libName> <cellName>
      [-inst <path1> <path2> <path3> ...]
      [-net <path1> <path2> <path3> ...]
      [-net -f <nets_file>]
      [-path <net1> <net2>]
      [-path -f <pair_file>]
      [-levels <num>]
      [-forward <num>]
      [-back <num>]
      [-stop [atStorage | atStorageData]]
      [-rename <rename_string>]
      [-out <output>] [-bus] [-force] [-help]
```

Note: The **-net** and **-path** options generate the following four output files:

<out_file>.v: Stores the Verilog output logic cone.
<out_file>.inst: Stores the traced instances for CPF.
<out_file>.pathinst: Stores the traced instances output.
<out_file>.pathnet: Stores the traced primitive output nets.

-inst <path1> <path2> <path3> ...

Lists the instances that need to be dumped.

-net <path1> <path2> <path3> ...

Lists the nets that need to be traced.

-net -f <nets_file>

Specifies the <nets_file> containing one net name per line.

-path <net1> <net2>

Specifies to trace paths from *<net1>* back to *<net2>*.

-path -f <pair_file>

Specifies the *<pair_file>* containing one net per line. Note that in the *<pair_file>* file every two lines form a pair, where the odd number of lines is *<net1>* and the even number of lines is *<net2>*.

-levels <num>

Specifies the maximum number of levels for the **-path** option to search.

-forward <num>

Specifies the forward tracing levels for the **-net** option. The default value is 10.

-back <num>

Specifies the backward tracing levels for the **-net** option. The default value is 10.

-stop [atStorage | atStorageData]

Specifies to stop tracing for DFF or data port.

-rename <rename_string>

Renames the instance name for verilog restriction.

-out <output>

Specifies the output file name.

-bus

Specifies to group the bus bits.

-force

Forces to generate a new .v file.

-help

Lists the syntax usage details.

Examples

- To generate and display a Verilog-format file for the PA13_U1 instance in the SII047B design, execute the following:

From the UNIX/Linux shell:

```
% verigen SII047B TOP -inst "PA13_U1"
```

From XEL command-line interface:

```
XE> exec verigen SII047B TOP -inst "PA13_U1"
```

- To trace the net named C in the instance named PA13, and trace the net backward five levels of design, execute the following:

From the UNIX/Linux shell:

```
% verigen SII047B TOP -net "PA13_C" -b 5
```

From XEL command-line interface:

```
XE> exec verigen SII047B TOP -net "PA13_C" -b 5
```

- To take the instance named PA13.U1, and output the results to the file named PA13.U1.v, execute the following:

```
% verigen SII047B TOP -inst PA13.U1 -bus -out PA13.u1.v
```

- To take the net named PA13.C, and output the results to the file named b2.v, execute the following:

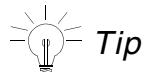
```
% verigen SII047B TOP -net PA13.C -back 2 -out b2.v
```

vvmDebug

Invokes the Virtual Verification Machine (VVM) debugger tool and loads the set of XEL commands associated with the debug process.

Note: After the vvmDebug tool initialization is complete, the UNIX command prompt changes to `XEvvvm>`.

When executed without a parameter, the vvmDebug command invokes the vvmDebug tool in an interactive command line mode.



For commands and syntax details of the XEL commands associated with the run-time and debug process, refer to [Chapter 13, “Run-time Commands,”](#) and [Chapter 14, “SA Run-time and Debug Commands.”](#)

The syntax for this command is:

```
vvmDebug [-gui|-noecho] [-shm|-fsdb|-vcd] [-init | -input <script_file> |  
@<command>] ... [-log <log_filename>] [-append_log] [-append_key] [-  
checksyntax]  
vvmDebug [-shm|-fsdb] [<script> [<script_option> ...] [probelist] [rcfile]]  
vvmDebug [-clean|-cleanall]  
vvmDebug [-version]  
vvmDebug [-help]
```

-gui

Opens the vvmDebug tool in the GUI mode. If this option is not specified, the vvmDebug tool runs in the command-line mode.

-noecho

Turns off the command echo when sourcing a Tcl script.

A normal Tcl shell does not echo commands when sourcing a file. However, in script mode, the vvmDebug tool echos the commands and redirected standard input for first level commands. You can also use the shell environment setting `setenv vXE NOECHO 1` to turn off the command echo.

-shm

Specifies that SimVision tools will be used as the design and waveform browser. The `-shm` option can also be specified as `-sst2`.

This is the default mode. Therefore, if the `vvmDebug` command is executed without specifying any of the supported options, the debugger is started in the SHM mode, by default.



You cannot change the mode once the vvmDebug tool has started.

-fsdb

Specifies that Debussy (Novas) tools will be used as the design and waveform browser. The `-fsdb` option can also be specified as `-td`.



You cannot change the mode once the vvmDebug tool has started.

-vcd

Enables support for VCD. For detailed information, refer to the *Debug Database* section in the *Overview* chapter of *UXE User Guide*.

-init | -input <script_file> | @<command>

Specifies an XEL or Tcl script file or the text of a command that is executed at the beginning of the session. A command can be a single command or a sequence of XEL or Tcl commands separated by a semi-colon (;).

For example,

```
vvmDebug -input "@debug . ; xc run xt0 zt0; run 100 ; exit"
```

- The `-init` and `-input` options are synonyms.
- You can enter the interactive mode only after the execution of the specified script file or command terminates.

- To execute more than one script file before entering the interactive mode or before executing the *<script>*, specify the **-init** or **-input** option for each script file. For example,

```
vvmDebug -gui -init f1.tcl -init f2.tcl
```

-log <log_filename>

Specifies the file to contain the log information.

-append_log

Appends the log to the existing log file.

-append_key

Appends the keystrokes to the existing key file.

-checksyntax

Checks the syntax of the commands specified in the input script.

Executing the **-checksyntax** option with the **vvmDebug** command enables you to quickly find and fix any syntactical mistakes.

Note: Short forms that uniquely identify the **-checksyntax** option, such as **-ch**, **-che**, or **-checks** can also be used.

<script> [<script_option> ...][probelist][rcfile]

Runs the specified XEL script. In GUI mode, after the script is run, the GUI remains open for further user-actions.

Note: Do not use **.** to source a script. Simply, type **vvmDebug <script>** on the command prompt. For example, **vvmDebug input**.

Extra arguments besides the predefined options within the specified XEL script can also be executed on the command line. In such cases, the first argument is treated as the name of the script to be executed after the **vvmDebug** initialization is done and the tool is ready to interact with you. All other options are arguments to be passed to the specified script.

For example,

```
vvmDebug -shm -noecho myScript.xel arg1 arg2 arg3
```

If multiple scripts need to be sourced, the first script must use all arguments as script names and source them one by one.

In addition, path to any of the following files can also be specified:

- File containing list of probes, that is, <*probelist*>
- Debussy .rc file containing the probe list for waveform display (that is, <*rcfile*>)

-clean

Removes from the design directory the compile database and log files generated at compile time.

-cleanall

Removes from the design directory all databases and log files generated at compile time and during previous debug sessions.

-version

Prints the software release version string. It does not start a debug session.

-help

Prints the command syntax and options.

fvCompute

Executes the preprocessing step necessary for waveform generation. Waveform generation consists of the following steps:

1. Uploading the binary data into the * .phy subdirectory
2. Preprocessing the binary data with the fvCompute program
3. Generating the waveform database in *.shm or *.fsdb format

All of the above steps are performed when the database -upload command is executed. However, a faster method is to execute only the first step while running the design on the emulator, and perform the other two steps while running the design in offline mode.

For this, split the waveform generation task as follows:

1. Execute the first step by using the database -prepareoffline command.
2. Execute the second step (manually) by issuing the fvCompute command on the UNIX command prompt.

Then, when you run the design using the xeDebug tool in offline mode, the final waveform-generation step will be very fast. This is because when you issue the database -upload command, and the fvCompute command has already been manually executed, only the waveform is generated. However, if the fvCompute command has not been previously executed, the fvCompute command is executed automatically, and both the preprocessing and waveform-generation steps are performed.

Note: If you execute the fvCompute command manually, the entire time window that has been captured in the *.phy database is preprocessed. When the fvCompute program is run automatically, only the time window that was specified with the following command, is preprocessed:

```
database -tracewindow -start <time> -end <time>
```

For longer time window, using more workstations will speed up the process. So, you can run the fvCompute program, in parallel, on multiple hosts. Use the xeset mt_host or the setenv CDN_MT_HOST command to specify additional workstations.

The syntax for this program is:

`fvCompute <phy file> [-nice <n>]`

UXE Command Reference Manual

Programs and Utilities

fvCompute <phy file> [-nice <n>]

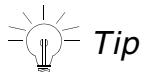
Do preprocessing of the .phy file so that the offline session runs faster. The -nice option is used to specify a nice value so that the fvCompute program runs at a lower priority.

xeCompile

Invokes the Palladium XP compiler tool and loads the set of XEL commands associated with the compile process.

Note: After the xeCompile tool initialization is complete, the UNIX command prompt changes to `XEC>`.

When executed without a parameter, the `xeCompile` program invokes the xeCompile tool in an interactive command line mode.



Tip

For commands and syntax details of the XEL commands associated with the compile process, refer to [Chapter 4, “General-purpose Commands,”](#) [Chapter 5, “Import Commands,”](#) [Chapter 6, “HDL Compiler Commands,”](#) [Chapter 8, “User Data Commands,”](#) [Chapter 9, “IP Commands,”](#) and [Chapter 11, “Compile-time Commands in ICE Mode.”](#)

The syntax for this program is:

```
xeCompile [-init <script_file> | @<command>] ... [-checksyntax] [<script>  
    [<script_option> ...]]  
  
xeCompile -version  
xeCompile -clean  
xeCompile -cleanall  
xeCompile -help
```

-init <script_file> | @<command>

Specifies an XEL or Tcl script file or the text of a command. A command can be a single command or a sequence of XEL or Tcl commands separated by a semi-colon (;).

-checksyntax

Checks the syntax of the commands specified in the input script.

Executing the `-checksyntax` option with the `xeCompile` command enables you to quickly find and fix any syntactical mistakes.

Note: Short forms that uniquely identify the `-checksyntax` option, such as `-ch`, `-che`, or `-checks` can also be used.

<script> [<script_option> ...]

Compiles the design in batch mode using the specified *<script>* Tcl script.

-version

Prints the software release version string. It does not start a compile session.

-clean

Removes the following files created during a previous xeCompile session:

- .design
- .simvision
- .xeBitMode
- .xeProfile.<username>
- .xedebugenv.gui
- .xeimp.dil
- CLOCK_SOURCES
- PDB
- QTDB
- cellList
- dbFiles
- tmp
- xe.msg
- xeCompile.key
- xeCompile.log

-cleanall

Removes the following files created during a previous xeCompile session:

UXE Command Reference Manual

Programs and Utilities

- .design
- .simvision
- .xeBitMode
- .xeProfile.<username>
- .xedebugenv.gui
- .xeimp.dil
- CLOCK_SOURCES
- DBEngine.msg
- PDB
- QTDB
- cellList
- dbFiles
- session.log
- tmp
- xe.msg
- xeCompile.key
- xeCompile.log
- xeDebug.key
- xeDebug.log
- xedebugenv.tcl

-help

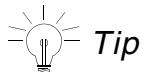
Prints the command syntax and options.

xeDebug

Invokes the Palladium XP debugger tool and loads the set of XEL commands associated with the run-time / debug process.

Note: After the xeDebug tool initialization is complete, the UNIX command prompt changes to `XE>`.

When executed without a parameter, the `xeDebug` command invokes the xeDebug tool in an interactive command line mode.



For commands and syntax details of the XEL commands associated with the run-time and debug process, refer to [Chapter 4, “General-purpose Commands,”](#) [Chapter 13, “Run-time Commands,”](#) and [Chapter 14, “SA Run-time and Debug Commands.”](#)

The syntax for this command is:

```
xeDebug [-gui | -noecho] [-shm | -fsdb | -vcd] [-init | -input <script_file> |  
@<command>] ...  
[-host <host_name>] [-dynp] [-log <log_filename>] [-append_log] [-  
append_key]  
[-checksyntax] [--ncsim <ncsim_option> ... [--]] [<script>  
<script_option> ...]  
xeDebug [-shm | -fsdb | -vcd] -wave <phyfile> [<probescript> | <probelist> |  
<rcfile>]  
xeDebug [-shm | -fsdb | -vcd] -offline <session> <phyfile> [<probescript> |  
<probelist> | <rcfile>]  
xeDebug -simlogsize <value> [-simincfile <value>]  
xeDebug [-clean | -cleanall]  
xeDebug [-version]  
xeDebug [-help]
```

-gui

Opens the xeDebug tool in the GUI mode. If this option is not specified, the xeDebug tool runs in the command-line mode.

-noecho

Turns off the command echo when sourcing a Tcl script.

A normal Tcl shell does not echo commands when sourcing a file. However, in script mode, the xeDebug tool echos the commands and redirected standard input for first level commands. This non-standard behavior of the xeDebug tool can be turned off using either the xeDebug -noecho command or the setenv XE_NOECHO 1 shell environment setting.

-shm

Specifies that SimVision tools will be used as the design and waveform browser. The **-shm** option can also be specified as **-sst2**.

This is the default mode. Therefore, if the xeDebug command is executed without specifying any of the supported options, the debugger is started in the SHM mode, by default.



Caution

You cannot change the mode once the xeDebug tool has started.

-fsdb

Specifies that Debussy (Novas) tools will be used as the design and waveform browser. The **-fsdb** option can also be specified as **-td**.



Caution

You cannot change the mode once the xeDebug tool has started.

-vcd

Enables support for VCD. For detailed information, refer to the *Debug Database* section in the *Overview* chapter of *UXE User Guide*.

-init | -input <script_file> | @<command>

Specifies an XEL or Tcl script file or the text of a command that is executed at the beginning of the session. A command can be a single command or a sequence of XEL or Tcl commands separated by a semi-colon (;).

For example,

```
xeDebug -input "@debug . ; host . ; xc run xt0 zt0; run 100 ; exit"  
xeDebug -host . -input "@xc run xt0 zt0; run 100 ; exit"
```

Note:

- The `-init` and `-input` options are synonyms.
- Both `irun` and NC-Sim accept the `-input` option with `@<command>`.
- You can enter the interactive mode only after the execution of the specified script file or command terminates.
- To execute more than one script file before entering the interactive mode or before executing the `<script>`, specify the `-init` or `-input` option for each script file. For example,

```
xeDebug -gui -init f1.tcl -init f2.tcl
```

-host <host_name>

Specifies the host name for the emulator. By default, the current machine is identified as the host machine.

-dynp

Changes the vision mode of a design compiled for FV mode to DYNP mode at run time without recompiling the design.

-log <log_filename>

Specifies the file to contain the log information.

-append_log

Appends the log to the existing log file.

-append_key

Appends the keystrokes to the existing key file.

-checksyntax

Checks the syntax of the commands specified in the input script.

Executing the `-checksyntax` option with the `xeDebug` command enables you to quickly find and fix any syntactical mistakes.

Note: Short forms that uniquely identify the `-checksyntax` option, such as `-ch`, `-che`, or `-checks` can also be used.

--ncsim <ncsim_option> ... [-]

Specifies the options that need to be passed to ncsim when using the SA mode.

Note: If `--ncsim <ncsim_options>` is followed by any other `xeDebug` command option, such as `<script> [<script_option> ...]`, specifying `--` after `<ncsim_options>` is mandatory to mark the end of the `--ncsim` command option. For example,

```
xeDebug --ncsim +GPU_500M +CPU_1G -- myscript.xel
```

<script> [<script_option> ...]

Runs the specified XEL script. In GUI mode, after the script is run, the GUI remains open for further user-actions.

Note: Do not use `.` to source a script. Simply, type `xeDebug <script>` on the command prompt. For example, `xeDebug input`.

Extra arguments besides the predefined options within the specified XEL script can also be executed on the command line. In such cases, the first argument is treated as the name of the script to be executed after the `xeDebug` initialization is done and the tool is ready to interact with the you. All other options are arguments to be passed to the specified script.

For example,

```
xeDebug -shm -noecho myScript.xel arg1 arg2 arg3
```

If multiple scripts need to be sourced, the first script must use all arguments as script names and source them one by one.

-wave <phyfile> [<probescript> | <probelist> | <rcfile>]

Opens an offline debug session in GUI mode using the specified offline `.phy` trace file. The offline debug session is assigned an automatically-generated user session name of the format `.offline<count>.session`.

In addition to the offline .phy trace file, if required, path to any of the following files can also be specified:

- File containing the script to add probes (that is, <probescript>)
- File containing list of probes (that is, <probelist>)
- Debussy .rc file containing the probe list for waveform display (that is, <rcfile>)

For example,

```
xeDebug -fsdb -wave user1.session/trace1.phy user1.session/my.rc
```

When you exit an offline debug session started using the `-wave` option, your session directory is cleaned, leaving only the `xe.msg` and waveform files.

-offline <session> <phyfile> [<probescript> | <probelist> | <rcfile>]

Opens an offline debug session in the GUI mode using your specified session name and the path to the offline .phy trace file.

In addition to the offline .phy trace file, if required, path to any of the following files can also be specified:

- File containing the script to add probes, (that is, <probescript>)
- File containing list of probes, (that is, <probelist>)
- Debussy .rc file containing the probe list for waveform display, that is, <rcfile>

For example,

```
xeDebug -offline user2 user1.session/trace1.phy user1.session/my.rc
```

When you exit an offline debug session started using the `-offline` option, the session directory is not cleaned automatically.

-simlogsize <value> [-simincfile <value>]

Specifies the size limit of the `xe.msg` log file. <value> specifies the maximum size in MB. The `-simincfile` option generates incremental log files, if the original log file exceeds the size specified with the `-simlogsize` option. The argument to the `-simincfile` option is an integer that limits the number of incremental log files. The incremental files are named as `xe-n.msg`. For example, if the original log file is called `xe.msg`, the incremental log files will be created as follows:

```
xe-1.msg
```

UXE Command Reference Manual

Programs and Utilities

```
xe-2.msg  
xe-3.msg  
...  
xe-n.msg
```

Example

```
xeDebug -simlogsize 10 -simincfile 5
```

If the `xe.msg` file exceeds 10 MB, `xeDebug` will create a new file, `xe-1.msg`.

Similarly, if `xe-1.msg` file exceeds 10 MB, `xeDebug` will create a new file, `xe-2.msg` and so on. At the most, five log files can be created as specified by the `-simincfile` option.

-clean

Removes from the design directory the compile database and log files generated at compile time.

-cleanall

Removes from the design directory all databases and log files generated at compile time and during previous debug sessions.

-version

Prints the software release version string. It does not start a debug session.

-help

Prints the command syntax and options.

xeclient

Enables you to connect from a terminal to a tool running on another terminal to run the XE commands remotely. The `xeclient` command can connect to the XE server of `xeCompile`, `xeDebug`, or `vvmDebug`.

The syntax for this command is:

```
xeclient [<hostname>:]<port> <command>
```

<hostname>

Connects to the specified host. The host name is following by a colon (:). Specifying the host name is optional.

<port>

Connects to the specified port of the server. You can also specify “-” as the port. If the port is specified as “-”, the default port for the session is used. The port is read from the current directory, that is, `xeclient` must be started from the same directory where `xeDebug` is running.

<command>

Executes the command on the specified host. The command can be also be enclosed in double quotes “ ”. You can also issue multiple commands separated by semicolon “;”.

xeGenReport

Prints the resource utilization report to show how the emulators are being used, such as, how often the machines are occupied, who is using the machine, which design is using the machine, and so on.

The xeGenReport utility generates the following usage information for each machine:

- The percentage of time that it was used, not used, and offline on a monthly basis or over a specified time period.
- The percentage of time that the emulator is used on a weekly basis, with separate day-time, night-time, weekday, and weekend usage information.
- The percentage of time that the emulator is used for each day.
- The percentage of time that each emulator board is used on a monthly basis, with separate day-time, night-time usage and board offline information.
- The percentage of time each emulator board is used in weekly basis.
- The percentage of time each emulator board is used in weekday only.
- The percentage of time each emulator board is used in weekend only.
- The percentage of time used by each user in weekly basis.
- The percentage of time used by each design in weekly basis.
- Measurements of over usage by number of design download failure due to in-used error for each day.
- The number of jobs downloaded each day.

The configmgr generates usage raw data every month in the /et3mach/log directory on the host workstation. You need to manually run the xeGenReport utility to read the raw data (*.joblog files) and generate the display data or utilization chart files in HTML format. Then, you can view the emulator utilization charts on a web browser (such as the latest versions of Firefox, Chrome, or Internet Explorer) with internet access to Google Chart.

The xeGenReport utility can only be run by authorized users. The Palladium XP software administrator needs to create a file <UXE_INSTALL_DIR>/tools.lnx86/.xegenreport that has a list of users who are authorized to run xeGenReport.

It is necessary to run this utility on a workstation that has permission to access the /et3mach/log directory on the workstation host that configmgr is running.

UXE Command Reference Manual

Programs and Utilities

Note: There is no license requirement to use Google Chart API functions. Your information or data is not transmitted to Google.

The syntax of this command is:

```
xeGenReport -host <configmgr_host1> [<configmgr_host2> ... <configmgr_hostn>]
    -period <year>.<start_month> [<year>.<end_month>]
    [-output <directory_name>]
    [-maxUser <number>]
    [-maxDesign <number>]
    [-daytimeHour <startHour> <endHour>]
    [-alias <file>]
    [-help]
```

-host <configmgr_host1> [<configmgr_host2> ... <configmgr_hostn>]

Using the `-host` option, specify the hosts for which the machine utilization report is required.

Note that `rsh` access is required to the `/et3mach` directory on the host workstation.

-period <year>.<start_month> [<year>.<end_month>]

Here, `<year>` must be between 2000-2099. The `<start_month>` and `<end_month>` to be reported must be between 1 and 12. The `<start_year>.<start_month>` must be specified. If `<end_year>.<end_month>` is not specified, only the `<start_year>.<start_month>` will be reported.

[-output <directory_name>]

Specifies the name of the directory in which the HTML files will be generated. If no output directory is specified, the HTML files will be generated in the current directory.

[-maxUser <number>]

Specifies the maximum number of user names to be displayed on the usage charts (per emulator and per month). Every month is calculated separately, and shows different users. By default, all users will be displayed.

The `-maxUser` option is primarily used to reduce the number of users on the chart. To calculate the number of users to be displayed on the chart, for example, in month 1, suppose there are six users, A, B, C, D, E, F who used 30%, 20%, 20%, 15% 10%, 5%. If no option is specified, every user will be displayed on the chart. If `-maxUser 3` is specified, only A, B, C will be on the chart, but the numbers will stay the same, 30%, 20%, 20%.

[-maxDesign <number>**]**

Specifies the maximum number of designs (projects) to be displayed on the usage charts (per emulator and per month). Every month is calculated separately, and shows different designs. By default, all designs will be displayed.

The **-maxDesign** option is primarily used to reduce the number of designs on the chart. To calculate the number of designs to be displayed on the chart, for example, in month 1, suppose there are six designs, A, B, C, D, E, F who used 30%, 20%, 20%, 15% 10%, 5%. If no option is specified, every design will be displayed on the chart. If **-maxDesign 3** is specified, only A, B, C will be on the chart, but the numbers will stay the same, 30%, 20%, 20%.

[-daytimeHour <startHour> <endHour>**]**

Specifies the starting and ending time (in hour) to be considered as day-time. **<startHour>** and **<endHour>** must be between 0 and 24, and **<startHour>** must be smaller than **<endHour>**. By default, 8 AM to 8 PM (**-daytimeHour 8 20**) is used.

[-alias <file>**]**

Specifies a file with an alias name for each of the emulator host, user, or design name. This enables custom emulator host, user, and design names to be displayed on the usage charts instead of the original names. The file contains the original host, user, or design names and the corresponding alias names in pair, and each original-alias name pair is printed on a separate line. Each line of the file also contains one of the three tags, HOST, USER, and DESIGN.

For example:

```
HOST <host1> <alias name1>
HOST <host2> <alias name2>
...
HOST <hostN> <alias nameN>
USER <user1> <alias user1>
USER <user2> <alias user2>
...
...
USER <userN> <alias userN>
DESIGN <design1> <alias design1>
DESIGN <design2> <alias design2>
...
```

UXE Command Reference Manual

Programs and Utilities

[-help]

Prints the help information for the usage of this utility.

UXE Command Reference Manual
Programs and Utilities

General-purpose Commands

This chapter describes commands that perform general-purpose tasks, which are not directly related to compiling and emulating your circuit design.

The following commands are discussed in this chapter.

- “[exit](#)” on page 148
- “[help](#)” on page 149
- “[msgControl](#)” on page 150
- “[quit](#)” on page 151
- “[redirect](#)” on page 152

exit

Closes the current Palladium XP XEL, Compiler, or Debugger session. This command is identical to the quit command.

This command can return an exit code on the operating system. If code is not specified, 0 is used as exit code.

Use this command at any time. No other commands need to precede it, and it is not a prerequisite for any other command.

The syntax for this command is:

`exit [code]`

[code]

It is an integer number that is returned as exit code on the operating system.

Example

`exit`

help

Returns online help about a specific XEL command, or about using online help.

You can use this command at any time. No other commands need to precede it, and it is not a prerequisite for any other command. This command returns the online help text about the specified XEL command.

The syntax for this command is:

```
help [-gui] [<command>]
```

-gui

Invokes the Cadence Help window that enables you to search and view the entire set of UXE documentation for the current release.

<command>

This is an optional argument that specifies the name of the XEL command for which you want to view the online help.

If you do not specify this argument, this command returns information about how to get the online help.

Specifying an asterisk (*) lists all the commands for which help is available.

Examples

```
help
```

Returns a message about how to get a list of topics.

```
help *
```

Returns a list of topics for which the online help is available.

```
help Syntax
```

Returns a message about syntax conventions used in all online help messages, such as use of <angles> and [brackets].

```
help design
```

Returns information about how to use the design command.

msgControl

Controls the echoing of messages to the console.

The syntax for this command is:

```
msgControl -<message_type> [on | off]
```

<message_type>

Specifies the type of messages that must be echoed to the console. The following values are supported:

info	Controls the printing of information messages.
error	Controls the printing of error messages.
progress	Controls the printing of progress messages.
report	Controls the printing of report messages.
warn	Controls the printing of warning messages.
<message_id>	Controls the printing of the messages having the specified ID. The message ID must be a valid integer

on | off

Specifies whether the echoing of the specified message type or ID must be switched on or off. If this option is not specified, the current settings for the corresponding message type or ID is printed.

Examples

To print the current settings for error messages, use the following command:

```
msgControl -error
```

To switch-off the echoing of error messages, use the following command:

```
msgControl -error off
```

To switch-off the echoing of the messages with the ID number 123, use the following command:

```
msgControl -123 off
```

quit

Closes the current Palladium XEL, Compiler, or Debugger session. This command is identical to the [exit](#) command.

Use this command at any time. No commands need to precede it, and it is not a prerequisite for any other command.

The syntax for this command is:

```
quit [code]
```

The `quit` command is identical to the `exit` command.

[code]

It is an integer number that is returned as exit code on the operating system.

Example

```
quit
```

redirect

Redirects the standard output (stdout) and/or errors (stderr) to a specified file.

The syntax for this command is:

```
redirect {<command>} <redirection_operator> <filename>
```

<command>

Specifies the XEL command that needs to be evaluated and its result redirected.

<redirection_operator>

Defines the required type of output redirection. Following are the valid operators:

Redirection Operator	Indicates
>	Redirects the standard output of the specified command to the named file. If the specified file already exists, the redirected standard output will overwrite the file; else, a new file with the specified name will get created to store the redirected output.
>>	Redirects the standard output of the specified command and appends it to the end of the named file. If the named file does not exist, a new file with the specified name will get created.
2>	Redirects the standard errors of the specified command and overwrites the named file. Here, 2 refers to the file number assigned to the standard error file.
2>>	Redirects the standard errors of the specified command and appends them to the end of the named file.
>&	Redirects both standard output and standard errors of the specified command, and overwrites the named file.
>>&	Redirects both standard output and standard errors of the specified command, and appends them to the end of the named file.

UXE Command Reference Manual

General-purpose Commands

<filename>

Specifies the name of the file in which the redirected standard output and/or standard error will be stored.

UXE Command Reference Manual

General-purpose Commands

Import Commands

This chapter describes XEL commands that imports your design netlists into your emulation design database.

The following XEL commands are discussed in this chapter.

- “[designImport](#)” on page 156
- “[impDumpConfig](#)” on page 158
- “[impLoadConfig](#)” on page 159
- “[importOption](#)” on page 160
- “[linkRefLib](#)” on page 165
- “[netlistFile](#)” on page 167
- “[refLib](#)” on page 169

designImport

Imports netlists, as specified in the preceding [netlistFile](#) commands, into the design database. This command can take input from a file or from any settings in [importOption](#).

Before using the `designImport` command, execute the [impLoadConfig](#) command to reload any configuration files that were created using the [impDumpConfig](#) command. Also, use [netlistFile](#) to select netlists to import.

This command returns any errors that `designImport` reported, and whether importing completed successfully or failed.

The syntax for this command is:

```
designImport [-noautomem]  
            [-keepExistingLinks]  
            [<file>]  
            [-noparallel]  
            [-useLSF]
```

-noautomem

(Optional) Turns off automatic generation of memory cells. Without this parameter, importing automatically generates memory cells from memory descriptions in the netlists that you are importing.

-keepExistingLinks

Retains all existing library links in your emulation design database. Without this parameter, incrementally importing your design automatically updates links for the `qtref.llib` and `generic.llib` files.

<file>

Specifies the name of the design import rule file created by [impDumpConfig](#).

-noparallel

Disables the parallel import mode.

Note: By default, the `designImport XEL` command invokes the netlist import process in *parallel import* mode.

-useLSF

Enables multi-host parallel import with LSF. However, to use this command option, LSF must be present and appropriately configured.

Examples

```
...
refLib vlis_vsc470 lsi_lca100k qtref generic
impDumpConfig myconfig
designImport myconfig
importOption {mode incr} {library counter}
refLib lsi_lca100k qtref generic
netlistFile {netlists c.verilog}
designImport
```

impDumpConfig

Saves import options defined using importOption into a specified file.

Before using impDumpConfig, execute importOption, netlistFile, and refLib commands.

The syntax for this command is:

`impDumpConfig <file>`

<file>

Specifies the name of the output file that saves the selected import options. This command automatically appends the .dil suffix to the specified file name.

Examples

```
refLib vlis_vsc470 lsi_lca100k qtref generic  
impDumpConfig myconfig
```

impLoadConfig

Reloads all import options from a file. You created this file when you used the [impDumpConfig](#) command.

Before using the impLoadConfig command, execute [impDumpConfig](#) to create the file.

The syntax for this command is:

```
impLoadConfig <file>
```

<file>

Specifies the name of the output file to be loaded. This file contains the saved import options that you defined using the [importOption](#) command. You do not need to specify the .dil suffix.

Examples

```
refLib vlis_vsc470 lsi_lca100k qtref generic  
impDumpConfig myconfig  
impLoadConfig myconfig
```

importOption

Specifies all input options for importing the design netlists selected in the [netlistFile](#) command. If you do not use this command, importing uses default option settings.

You can use this optional command at any time. No other commands need to precede it, and it is not a prerequisite for any other command. However, it should be used before [designImport](#); your option settings do not take effect until the next time you execute [designImport](#).

Because import options are associated with netlists, `importOption` must precede the `netlistFile` command for the netlist that the option is to be applied to. Options remain in effect until superseded or until a `designImport` command is executed. They do not have to be repeated for each netlist.

The syntax for this command is:

```
importOption [{mode incr | full}]
[{compatibility sl | ns | ci | cs | cd | ir}]
[{:library <lib>}]
[{:libType design | asic | generic | ref}]
[{:memFile <file>}]
[{:includes <dir>}]
[{:special <arg>}]
[{:DEFINE <variable>[=<value>] ...}]
[{:DEFINE <variable>[=<value>] ...}]
[{:search on | off}]
[{:protection on | off}]
```

{mode incr | full}

(Optional) Specifies whether to incrementally or fully import the netlists selected in the [netlistFile](#) command.

Incremental (`incr`) mode imports only the selected netlists that you created or edited since you last imported your design.

Full mode (`full`) imports all selected netlists, and overwrites all cells in your design the next time you execute [designImport](#), even if one or more netlists have not changed since you last imported them.

Previously-imported netlists that you do not specify for importing (using [netlistFile](#)) remain unchanged in your design database, whether you use `incr` or `full` mode.

If you do not specify this parameter, importing uses incremental mode by default.

{compatibility s1 | ns | ci | cs | cd | ir}

You can specify none, or any combination of the first four compatibility parameters (`s1 | ns | ci | cs`). However, the last option, `cd`, overwrites the `cs` and `ci` parameter settings.

`s1` (single library) imports all netlists (selected using the `netlistFile` command) either into the design library specified in the library parameter or the default `XE_design_top` library. The import ignores any library names specified in Attribute files, on import, if `s1` is used.

Without the `s1` parameter, import stores the netlist in the design library specified in the netlist.

`ns` (no search) defers searching the current library. Instead, it searches the libraries specified in `refLib`, in the order that you selected the libraries, until it finds a definition for the cell. If no definition is found in the search libraries, the current library is searched. You may include the current library in the list of the libraries to search.

Without the `ns` parameter, import first checks the current library for cell definitions, before searching the libraries listed in `refLib`. This option ensures that the import uses a specific and consistent set of cell definitions.

`cs` (case sensitivity) forces `designImport` to consider case when evaluating all names in netlists. In importing with this option, the case of identifiers is preserved in the `.llib` on a per netlist basis.

Without the `cs` parameter, all names in netlists are case-insensitive by default. For example, the Netlist Importer normally changes name and Name to NAME.

`ci` (case insensitivity) forces `designImport` to ignore case when evaluating all names in Verilog netlists and attribute files.

Note: If a netlist is imported with the `ci` option, all identifiers are stored in `.llib` as `UPPERCASE`.

Without the `ci` parameter, all names in Verilog netlists and attribute files are case-sensitive by default. For example, you can use separate name tokens called `name`, `Name`, and `NAME`.

`cd` (case default) sets the case sensitivity for the selected netlist(s), to the default for the format of each netlist. This setting overrides any previous case setting (either `cs`, `ci`, or both). That is, it turns off all currently active case sensitivity settings.

`ir` (ignore renames) causes the netlist reader to ignore rename statements.

{library <lib>}

(Optional) Specifies the name of the design library in which the netlists selected in `netlistFile` should be imported. If the specified `<lib>` does not exist, import creates the library.

If you do not specify this library, importing uses any library specified in your netlist, or the `XE_design_top` library name by default. If any specified library does not exist, import creates the library.

{libType design | asic | generic | ref}

(Optional) Specifies the type of library to be created and used when importing the design: an ASIC library (containing master cell definitions), the Quickturn generic or reference library (containing definitions of primitive cells that the verification system can process), or a design library (containing your imported netlists).

{memFile <file>}

(Optional) When import creates a memory cell definition from a pattern, it writes the name of the cell and its library into this file. This file is used by the memory compiler to identify those cells that need to be compiled. By default, this filename is derived automatically.

{includes <dir>}

(Optional) Specifies the directory to be searched for files referenced in `includes` statements in your Verilog design. The following are examples of `includes` statements in Verilog design files:

```
'includes "parts/count.v"  
'includes "ref.v"
```

{special <arg>}

(Optional) Specifies the special arguments to be provided to the `designImport` command. The argument `<arg>` can have the following values:

- `NO_TIMESTAMP`: Ignores netlist time stamps. This argument is used in the library build process.

UXE Command Reference Manual

Import Commands

- **CONTINUE_ON_FATAL_**: Ignores fatal errors in the netlist and continues to run the design till completion and creates database files. This might result in corrupt database files.
- **DELETE_LIBS_ON_FULL** or **DELETE_LIBS_ON_OVERWRITE**: Forces `designImport` to delete the design and definition libraries if you set the mode parameter value to full.
- **NO_PWRNET_STORE**: Specifies that the PWRNET attributes will not be saved in the created libraries.
- **NO_PWRNET_LOAD**: Specifies that the PWRNET attributes will not be inherited from `qtref` and other search libraries. If you do not use this option, nets with names designated as power or ground in `qtref` and other search libraries are automatically tied to power or ground.

{**DEFINE** {<variable>}[=<value>] ...}}

(Optional) Defines an `importOption` variable to a specified value.

{**search on | off**}

Searches for the specified definition based on name. The default setting is `on`.

If the `search` setting is `on`, first, the design library (the one that is being created or filled) is searched. Then, the reference libraries are searched, in the listed order.

If the `search` setting is `off`, the design library is searched after all the reference libraries have been searched.

When a definition with the name matching the search term is found, the search ends. If the search exhausts all possibilities without finding a match, the pattern files and saved patterns are searched. If these also fail, the import operation fails with an error message.

{**protection on | off**}

(Optional) Turns the source-protected netlist capability on or off. If a protected netlist is read with source protection turned on:

- the internal net and instance names within its protected modules are encrypted.
- the internals of these modules cannot be viewed using the design browser nor the schematic viewer.

UXE Command Reference Manual

Import Commands

- waveforms cannot be viewed for nets within instances of these modules.

The import program automatically turns the protection option on or off, as needed, for each netlist. The protection option exists only to enable backward compatibility with existing scripts.

Examples

```
importOption {mode incr} {library counter}
```

Imports only new or edited design netlists (incremental mode) into the counter design library.

linkRefLib

Creates symbolic links for libraries in your emulation design database: that is, the directory from which you started running the UXE software, which is included in the installed UXE software to a library of the same name in your emulation design database.

Use this command at any time. No other command needs to precede it, and it is not a prerequisite for any other command.

The syntax for this command is:

```
linkRefLib [-keepExistingLinks]  
[<lib>]
```

-keepExistingLinks

Retains all existing links in your design database. Only specified new links are established, and existing links are not changed. Without this parameter, linking reference libraries automatically updates the specified links (lib1.llib, lib2.llib, and so on), plus qtref.llib and generic.llib files.

<lib>

(Optional) Specifies the name of reference library.

<lib> can be either an absolute or relative path.

A relative path points to a reference library in the *<install_dir>/qtlib* directory, where *<install_dir>* is the directory in which you installed the UXE software.

This creates a symbolic link from the *<install_dir>/qtlib/<lib>.llib* library file to the *<design_dir>/QTDB/ <lib>.llib* library file, where *<design_dir>* is your emulation design directory (the directory from which you started running the UXE software).

An absolute path links your own reference library from a directory outside of the current emulation design directory into the QT Database (QTDB). For example:

```
linkRefLib /engineering/libraries/lib1
```

Without parameters, this command links generic.llib and qtref.llib from the *<install_dir>/qtlib* to *<design_dir>/QTDB*.

UXE Command Reference Manual

Import Commands

Examples

```
% cd /myhome/mydesign  
% xeCompile  
XE> linkRefLib qtref
```

Creates a symbolic link from the *<install_dir>/qtlib/qtref.llib* library file, to the */myhome/mydesign/QTDB/qtref.llib* library file.

netlistFile

Selects a netlist file to import when using designImport. Netlist files can be of a specific format or attribute files.

At least one `netlistFile` command must be issued before issuing the `designImport` command. You can use multiple `netlistFile` commands, but the `netlistFile` commands must be followed by a `designImport` command. The `netlistFile` command does not need to be preceded by any other commands.

In addition to specifying the netlist file to be imported, you can specify a file containing the list of modules that should be explicitly excluded from the import. Any modules listed in this *exclude* file are excluded by the Verilog parser, and neither the module's definition nor its instances are imported into the database.

All file types except the *exclude* files, add information to the database. *Exclude* files do not add any information to the database, and should not be used unless accompanied by actual netlist files.

The syntax for this command is:

```
netlistFile {verilog <netlist>} |  
           {netlists <netlist>} |  
           {attribute <file>} |  
           {exclude <filename>}
```

verilog <netlist>

Imports netlists in Verilog format.

netlists <netlist>

Imports netlists and uses the netlist filename extension to determine the format of the netlist.

attribute <file>

(Optional) Specifies the name of an attribute file to be imported. An attribute file defines properties of cells in a custom library, such as power and ground declarations, pin properties, and global properties for reset lines or fast clock nets.

exclude <filename>

(Optional) Specifies a module or list of modules that should not be imported. The importer regards these specified modules as empty modules. The *<filename>* represents a file listing modules to be excluded. The contents of this file should be given in the following syntax:

```
VERSION X
  < modulename>
  . .
  <modulename>
```

Examples

```
netlistFile {netlists c.verilog d.verilog}
```

Identifies two netlists, named **c.verilog** and **d.verilog** respectively, for subsequent **designImport**.

refLib

Selects the order of libraries to search for cell definitions. The `designImport` command uses the cell definitions in these libraries to correctly interpret cells in the imported netlists.

Use this command prior to specific `netlistFile` requiring cell definition from it. No other command needs to precede it. If you use this command, your library selections do not take effect until the next time you execute `designImport`.

If one or more cells are not defined in any of the specified ASIC libraries, importing reports an error.

The syntax for this command is:

```
refLib [-keepExistingLinks] <lib> ...
```

-keepExistingLinks

`refLib` passes this parameter when it calls `linkRefLib`. Without it, `linkRefLib` automatically updates all specified links, plus `qtref.llib` and `generic.llib` files.

<lib> ...

One or more ASIC library names.

Examples

```
refLib vlis_vsc470 lsi_lca100k qtref genericictega
```

Starts searching in the `vlis_vsc470` ASIC library for definitions of all cells in the design that you are importing. If that library does not define one or more cells in your design, importing searches `lsi_lca100k`, `qtref`, and `generic` libraries, in that order, until it finds a definition of the cell.

UXE Command Reference Manual

Import Commands

HDL Compiler Commands

This chapter describes XEL commands used to import and synthesize RTL Verilog or VHDL design files to create a gate-level netlist for the design prior to emulation.

The following XEL commands are discussed in this chapter:

- “[hdlDefineLib](#)” on page 172
- “[hdllImport](#)” on page 174
- “[hdllInputFile](#)” on page 179
- “[hdlLang](#)” on page 182
- “[hdlLogFile](#)” on page 183
- “[hdlOutputFile](#)” on page 184
- “[hdlSkipModule](#)” on page 187
- “[hdlSynthesize](#)” on page 189
- “[hdlWorkPath](#)” on page 196

Note: The commands in this chapter cannot be run in the Debugger.

hdlDefineLib

Defines the directory path that a logic library name represents. The logic library is a symbolic name for a directory, usually a subdirectory, of your emulation design directory.

Use this command for both VHDL and Verilog. By default, HDL Compiler assigns the `lib` library name to the `lib` subdirectory in your emulation design directory.

Note: HDL Compiler includes four built-in logic libraries: IEEE, STD, QTREF, and Cadence. Do not redefine any of these libraries or their content.

You can use this command at any time. No other commands need to precede it, and it is usually not a prerequisite for any other command. If your VHDL source code refers to libraries other than `work`, this command must precede `hdlImport`.

Without parameters, this command lists all logic library names that were defined in all preceding `hdlDefineLib` commands, or reports that no logic libraries are currently selected for assignment to a directory.

The syntax for this command is:

```
hdlDefineLib [-add <lib> <dir> | -rm <lib> | -clr]
```

-add <lib> <dir>

Assigns the specified logic library `<lib>` to the selected directory `<dir>`, and adds it to the list of existing logic library aliases. Each `hdlDefineLib` command assigns one logic library name to one directory.

-rm <lib>

Removes the `<lib>` logic library name from the list of existing logic library names. The `<lib>` library name is not assigned to a directory.

-clr

Clears all logic library names and directory paths from the list of logic library names to assign to directories.

Examples

```
hdlDefineLib -add mylib /project1/mylib
```

UXE Command Reference Manual

HDL Compiler Commands

Assigns the logic library name `mylib` to the `/project1/mylib` directory.

You can use a full directory path, as in this example, to import modules that several designs will share.

```
hdlDefineLib -rm projectlib
```

Removes the VHDL logic library named `projectlib`, from the list of VHDL logic library names to define (that is, to assign to a directory).

hdlImport

Imports the HDL files selected in all preceding hdlInputFile commands. These files are imported into the working library specified in hdlWorkPath, or the default working library (work) if you did not specify another library.

hdlInputFile must precede this command to select the files to import.

If the source code you are importing refers to a library other than work, IEEE, STD, QTREF, and Cadence, hdlDefineLib must also precede this command.

This command reports any syntax errors that hdlImport found in the imported HDL design files, and whether importing succeeded or failed.

The syntax for this command is:

```
hdlImport [-full] [-u] [-2001] [-v2008] [-l <lib>] [-sv] [-atb] [-ignorePragma+<module_name> [+...]]
```

-full

Imports all selected HDL design files, even if you have not edited them since last import.

Using this option makes importing designs take longer to complete.

Without this parameter, this command checks the file name and time stamp for all design files that you are importing. This command then imports each selected design file only if you did not previously import it, or if you edited it since you last imported your design.

This command does not affect any previously imported design files that you do not select for importing again.

-u

Makes Verilog names case insensitive.

-2001

Turn on Verilog 2001 support.

For more detail, refer to the IEEE Standard Verilog Hardware Description Language, IEEE Std 1364-2001. The bracketed numbers given with the features listed below will point you to the relevant sections of the IEEE reference.

Features currently supported are:

- signed arithmetic [4.1.6; 4.5]
- comma-separated sensitivity list [9.7.4]
- ‘ifndef’, ‘elsif’ and ‘undef’ directives [19.4; 19.3.2]
- named parameter association [12.2.2.2]
- combinational sensitivity list (that is, @*) [9.7.5]
- reg declaration initial value assignments [6.2.1]
- default_nettpe none (disabling implicit net declarations) [19.2]
- list_of_port_declaration style (that is, ANSI-C style module declarations) [12.3.4]
- task/function_port_list style (that is, ANSI-C style declarations) [10.2.1; 10.3.1]
- reentrant task and recursive function (automatic) [10.2.1; 10.3.1]
- automatic width extension beyond 32 bits [2.5.1]
- implicit nets with continuous assignments [3.5; 6.1.2]
- local parameter [3.11]
- power operator [4.1.5]
- line compiler directive [19.7]
- attribute [2.8] (Only `full_case` and `parallel_case` attributes are supported. Other attributes are read but ignored.)
- system timing checks [15]

Note: The Verilog -2001 option now includes the system timing checks in a restricted manner. All system timing checks are recognized but their functionality is ignored.

- multidimensional array of regs [3.2.2; 3.10.2; 4.2.2]
- indexed vector part select [4.2.1]
- sized/typed parameter [3.11.1; 12.2]
- for-generate [12.1.3.1; 12.1.3.2]

- if-generate [12.1.3.3]
- case-generate [12.1.3.4]
- constant functions [10.3.5]
- multidimensional array of nets [3.10.1]
- bit/part-select of array element [4.2.2]

-v2008

Turns on the support for VHDL 2008 features. The following features are currently supported by the HDL compiler according to the *IEEE Standard 1076-2008 VHDL Language Reference Manual*:

- Unary logical operators
- Keyword `ALL` in sensitivity list of processes
- Boolean vector
- Matching relational operators
- Matching case statement
- Modulus or remainder operators on physical types

-l <lib>

Use `<lib>` library to search for undefined modules. The `-l <lib>` option specifies the name of a Verilog, VHDL, or ASIC source library referenced by the design. Each `-l <lib>` option can specify only one library, but you can use multiple `-l <lib>` options to specify multiple libraries within a single `hdlImport` command.

-sv

Enables the recognition of the SystemVerilog keywords.

Refer to the *Supported SystemVerilog Features* section in the *Modeling Guide for Verilog Designs for ICE Mode* chapter of the *UXE User Guide* for a list of supported SystemVerilog features from the IEEE Standard 1800-2005.

-atb

For a list of supported constructs with the *-atb* option refer to the *Synthesizing Designs with Acceleratable Testbench (ATB) Constructs* section in the *Compiling Designs for In-Circuit Emulation with xeCompile* chapter of the *UXE User Guide*.

Supports the following Acceleratable Testbench (ATB) constructs:

- Initial blocks
- Verilog-2001 initial values
- VHDL initial values
- \$test\$plusargs and \$value\$plusargs in any synthesizable expression, or any initial or always block
- force and release statements in initial or always blocks
- \$finish and \$stop in any initial or always block
- \$display and \$write in any initial or always block
- \$time as an argument in \$display or \$write
- \$writememb and \$writememh
- \$qel: A special system task to invoke XEL or Tcl commands inside Verilog code. \$qel takes the arguments in a format similar to that of \$display. But, it executes the string as a command as it would in XEL prompt, instead of displaying the resulting string. For example, you can dump memory in binary format to a location specified by a variable:

```
reg [7:0] mem [65535:0] ;
reg [15:0] end_loc;
always @(posedge clk)
if (enable)
$qel ("memory -dump %%pd_raw2 %mem -file myfile -end %d", end_loc);
```

Here %% means escaped % as in \$display format, %m creates the full instance path from the top, and %d gets the run-time value of a variable.

Note that the ATB mode is supported only through the command-line interface.

-ignorePragma+<module_name>[+...]

Specifies that the mentioned module should be excluded when honoring a synthesis directive. For example:

```
hdlImport
```

UXE Command Reference Manual

HDL Compiler Commands

```
-ignorePragma+\  
<module1>+<module2>
```

Examples

```
hdlInputFile f1.v f2.v f3.v  
hdlImport -l lib1
```

Imports the f1.v f2.v, and f3.v Verilog HDL files. The f3.v file includes instances of cells from the lib1 ASIC library.

```
hdlInputFile unit1.vhd unit2.vhd  
hdlImport
```

Imports the unit1.vhd and unit2.vhd VHDL files.

```
hdlInputFile add1.v add2.v  
hdlImport -full
```

Imports the add1.v and add2.v Verilog HDL files, even if they are unchanged since you last imported them.

hdlInputFile

Specifies the VHDL or Verilog design files to import into your HDL Compiler database. You can use multiple `hdlInputFile` commands. Each `hdlInputFile` command adds to or deletes from the list of design files to import.

Before using `hdlImport`, execute one or more `hdlInputFile` commands with the `-add` parameter to specify the files to import.

Without parameters, lists all HDL design files selected for importing in all preceding `hdlInputFile` commands, or reports that no files are currently selected for importing.

If the design files are already imported, you can use `hdlInputFile` without parameters. This action lists all files selected for importing the last time you executed `hdlImport`.

The syntax for this command is:

```
hdlInputFile [-add <file>] [-errormax <arg>] [-rm <file>... | -clr]  
[-synth_prefix <pragma_name>]  
[-rm_synth_prefix <pragma_name>]
```

The HDL Compiler also supports the following Verilog-XL compile options with the `hdlInputFile` command:

```
[-f <commandFile>] |  
[-v <filename>] [-y <directory>] |  
[+define+<var1>+..] |  
[+libext+<ext1>+..] |  
[+inmdir+<path1>+..] |  
[-u]
```

-add <file>

Adds the specified file names to the list of HDL design files to import.

-errormax <arg>

Specifies the maximum number of errors processed.

-rm <file>...

Removes the specified file names from the list of HDL design files to import.

-clr

Clears all file names from the list of HDL design files to import.

You can, for example, use this parameter in the first `hdlInputFile` command. This action clears the file list from the last time you imported the design, so you can define new files to import.

-synth_prefix <pragma_name>

Appends additional pragmas for the synthesis directive to the default pragma list. The default pragmas are:

- `pragma`
- `synopsys`
- `quickturn`

-rm_synth_prefix <pragma_name>

Deletes the specified pragma for the synthesis directives from the default pragma list. Note that this option does not remove the `quickturn` pragma.

-f <commandFile>

Specifies a file for command arguments (Verilog XL compile option).

-v <filename>

Specifies a library file.

-y <directory>

Specifies a library directory.

+define+<var1>+..

Defines macros.

+libext+<ext1>+..

Specifies library directory file extensions.

+inmdir+<path1>+..

Specifies paths for searching include files.

-u

Makes the analyzer case-insensitive.

Examples

```
hdlInputFile -add f3.v  
hdlInputFile -add f1.v f2.v
```

Imports the `f3.v` file, then imports `f1.v` and `f2.v` Verilog files.

```
hdlInputFile -add unit1.vhd unit2.vhd
```

Imports the `unit.vhd` and `unit2.vhd` VHDL files.

```
hdlInputFile -rm unit2.vhd
```

Removes `unit2.vhd` from the list of HDL design files to import.

```
hdlInputFile
```

Lists all HDL design files currently selected for importing in all preceding `hdlInputFile` commands.

hdlLang

Selects whether the HDL design files that you are importing are in Verilog or VHDL language.

If you do not use `hdlLang`, `hdlImport` assumes the default language as Verilog.

When you use `hdlLang`, it must precede `hdlInputFile`, which specifies the names of the HDL design files to import.

If you give the `hdlLang` command without any value, the current HDL language setting is returned, which can be either `verilog` or `vhdl`.

If you have not set an `hdlLang` value earlier, and give the `hdlLang` command, the default value is returned, which is `verilog`.

The syntax for this command is:

```
hdlLang [verilog | vhdl]
```

verilog

Specifies that the HDL design files are in Verilog language (default).

vhdl

Specifies that the HDL design files are in VHDL language.

Examples

```
hdlLang verilog
```

```
hdlLang vhdl
```

hdlLogFile

Specifies the name of the log file that `hdlImport` or `hdlSynthesize` uses. This file records actions performed while importing HDL design files and synthesizing them into a gate-level netlist. You can specify a single log file for the entire import and synthesis process, or one file for importing the design and another for synthesizing the design.

If you do not use `hdlLogFile`, the `hdlImport` and `hdlSynthesize` commands use the default log file name, `hdlIce.log`.

If you use `hdlLogFile`, it must precede `hdlImport` or `hdlSynthesize` for which you are selecting a log file.

The `hdlLogFile` command returns the name of the current log file, as set in the most recent `hdlLogFile` command with parameters.

If you have not used `hdlLogFile` with parameters, it returns the default log file name, `hdlIce.log`.

The syntax for this command is:

```
hdlLogFile [<log_file>]
```

<log_file>

Specifies the name of the HDL Compiler log file to be created.

Examples

```
hdlLogFile my_synth.log
```

Creates a log file named `my_synth.log` in the emulation design directory.

```
hdlLogFile /project1/synthesis.log
```

Creates a log file named `synthesis.log` in the `project1` directory, which is not part of the emulation design directory.

hdlOutputFile

Writes the output of HDL synthesis to a file in Verilog or VHDL format.

The output file can be used for other design tools as well as the UXE software. For example, you can create an output file for use in simulating the gate-level netlist that [hdlSynthesize](#) generates.

This command converts all names in the VHDL design into uppercase in the Verilog netlists.

For a VHDL design in which a component with generic values is instantiated, this command writes a component with a unique name in the following format:

<componentName>_<firstGenericValue>_<secondGenericValue>_...

For example, if you have the following code:

```
module test (input din, output dout);  
parameter g1;  
parameter g2;
```

This code might have a version of module `test` with `g1=1` and `g2=1`. In this case, a component with a unique name `test_1_1` will be written to the synthesized output.

The same action is performed for a design having a Verilog module with parameters.

If you use `hdlOutputFile`, it must precede `hdlSynthesize`, which specifies the name of the design to synthesize into a gate-level netlist.

The `hdlOutputFile` command returns the name of the output file created in the most recent `hdlOutputFile` command with parameters.

If you have not used `hdlOutputFile` with parameters, no output file is generated.

The syntax for this command is:

```
hdlOutputFile [-add -f [verilog | vhdl | lec | debussy | qel] <file>] |  
[-rm <file>] |  
[-clr]
```

-add -f [verilog | vhdl | lec | debussy | qel] <file>

Adds the specified file to the list of synthesis output files to create.

<file> can include a suffix to indicate whether the file to add is in Verilog (.v suffix) or VHDL (.vhdl suffix) format. The suffix ensures that saving a file in one format does not overwrite a file with the same root file name in another format.

-f verilog <file> creates a synthesis output file in Verilog format.

-f vhdl <file> creates a synthesis output file in VHDL format.

-f lec <file> creates a Conformal script that can be invoked by using the following command:

```
lec -dofile <file> -nogui
```

-f debussy <file> automatically generates the Debussy vericom or vhdlcom script, as required.

To generate the script for Debussy tools, use the following command:

```
hdlOutputFile -add -f debussy <file>
```

This command creates the following two text files and a shell script file:

■ Text files:

- <file>.libDebussy: This file contains all the logical library and physical library mapping information.
- <file>: This file lists the design files according to the dependency order, and indicates the library to which these files should be imported. It also indicates whether you need to invoke the VHDL compiler or the Verilog compiler.

■ Shell script file:

- debussyRTLImport.sh: This shell script file is passed the list of design files (<file> generated above) as follows:

```
debussyRTLImport.sh <file>
```

The above command performs the Debussy database compilation using the required VHDL or Verilog compiler, and generates the turbo.f file. It physically maps the Debussy libraries to the same location as the library for the HDL Compiler.

For example, suppose the hdlDefine command is as follows:

```
hdlDefine lib abc ./work/abc
```

Then, the Debussy library map is specified as abc = ./work/abc and the physical location is ./work/abc.lib++.

Use the verdi -f turbo.f & command to view the Debussy schematic database that has been generated.

-f qel <file> instructs HDL Compiler to create a file with the name specified as *<file>* containing the XEL commands for ATB (Accelerated TestBench) features to be invoked by the `hdlConfig` command at run time. The `-f qel` option must not be used with `hdlSynthesize -all`.

The XEL files created by separate `hdlSynthesize` sessions cannot be concatenated. All the ATB features must be synthesized by a single `hdlSynthesize` session in order to be effective. `-f qel` output cannot be used with QTDB writer or parallel synthesis. `-f qel` cannot be used for Verilog modules with both ATB features and IP protection.

-rm <file>

Removes *<file>* from the list of output files to create.

-clr

Clears all file names from the list of output files to create.

Examples

```
hdlOutputFile -add -f verilog design_output.v
```

Creates a Verilog file named `design_output.v`.

```
hdlOutputFile -add -f vhdl syn_netlist.vhd
```

Creates a VHDL file named `syn_netlist.vhd`.

```
hdlOutputFile -rm my_output.vhd
```

Removes `my_output.vhd` from the list of VHDL files to create.

```
hdlOutputFile -clr
```

Clears the list of output files from all preceding `hdlOutputFile` commands.

hdlSkipModule

Declares imported Verilog and VHDL design modules that you do not want to synthesize.

If you use this command, it must precede the [hdlSynthesize](#) command, which generates a gate-level netlist from the imported HDL design.

The syntax for this command is:

```
hdlSkipModule [-add [<library>.]<module>] |  
[-rm [<library>.]<module>] |  
[-clr]
```

-add [<library>.]<module>

Adds the Verilog or VHDL design modules, specified in the <module> parameter, to the list of design modules that you do not want to synthesize.

If the <library> parameter is not specified, the module name is searched only in the current working library.

-rm [<library>.]<module>

Removes the Verilog or VHDL design modules specified in the <module> parameter, from the list of design modules that you do not want to synthesize. That is, synthesis includes the specified modules with the rest of the design.

If the <library> parameter is not specified, the module name is searched only in the current working library.

-clr

Clears the list of design modules that you do not want to synthesize. That is, synthesis includes all modules of the design in the gate-level netlist for emulation.

Examples

```
hdlSkipModule -add module1 module2
```

Adds the module1 and module2 Verilog design modules to the list of modules that you do not want to synthesize.

```
hdlSkipModule -rm unit1
```

UXE Command Reference Manual

HDL Compiler Commands

Removes the `unit1` VHDL module from the list of modules that you do not want to synthesize. That is, `hdlSynthesize` will synthesize `unit1` with the rest of the design.

hdlSynthesize

Synthesizes a gate-level netlist from the imported HDL design. Unless the `-full` option was used in the `hdlImport` command, this command automatically performs content-based incremental synthesis. This means that the HDL Compiler synthesizes a module again only if its contents are different than the previously synthesized design. This command returns any errors that synthesis reported, and whether synthesis completed successfully or failed.

The `hdlInputFile` and `hdlImport` commands must precede this command, in that order. If the optional `hdlOutputFile` command is used, it must also precede the `hdlSynthesize` command.

The syntax for this command is:

```
hdlSynthesize {-all | -only <module_name> | <top_module>}
               [-alu | -alu<arg>]
               [-dontStopOnError]
               [-memory | -memory<N>]
               [-keepVhdlCase]
               [-vhdlToLowercase]
               [-noNamePreservation]
               [-fanoutNetOnly]
               [-keepAllFlipFlop]
               [-parallel | -parallel<number> | -parallelLSF<number>]
               [-append]
               [-portNaming <pattern>]
               [-D<generic>=<value>]
               [-rename <module_name>]
               [-displayBuffer [+depth=<num>] [+timeOut=<num>] [+port=<num>]]
               [-siliconSynthesisCompatible]
```

-all

Synthesizes all modules in the work library. Use when compiling the cells of a library that does not have a top module.

Note: Avoid using this option with the `hdlOutputFile -f gel` command for ATB, or the output file will not be functional.

-only <module_name>

Synthesizes only the specified module.

<top_module>

Synthesizes the specified *<top_module>* and all modules in the design hierarchy below it. This command does not synthesize any modules at the same level as, or at a higher level than, the specified *<top_module>*.

To synthesize the entire design, specify the actual top module. To synthesize a part of the design, specify the highest level module of that part.

-alu

Switches on ALU mapping with the default ALU threshold of 32.

-alu<arg>

Switches on ALU mapping with the ALU threshold as represented by *<arg>*.

-dontStopOnError

Specifies that synthesis must not be stopped even when the first module with an error is found.

-memory

Synthesizes memories as hard macros.

-memory<N>

Specifies that all memories smaller than *<N>* bits (where
size_of_the_memory = *width* * *depth*) should be mapped to gates and flip-flops.

For example:

```
hdlSynthesize -memory256 TOP
```

Note that there is no space between **-memory** and 256. The **-memory256** option indicates that any memory smaller than 256 bits (*width* x *depth*) should be mapped to gates and flip-flops.

In certain cases, larger memories might also be mapped to gates and flip-flops, but definitely all memories smaller than 256 bits will be mapped. If the `-memory` option is given without any number, the default size is taken as 32.

Depending on design requirements, you might improve emulation speed or capacity by using `-memory<N>`, with `<N>` being larger or smaller than the default size.

-keepVhdlCase

When converting from VHDL to Verilog, keeps the case of names as much as possible. Otherwise, VHDL names are made all-uppercase in the Verilog output.

-vhdlToLowercase

Converts VHDL symbols into all lowercase.

-noNamePreservation

Switches off the name preservation feature to reduce gate count.

-fanoutNetOnly

Specifies that the names of the nets that do not have any fanout to the output ports are not preserved in order to reduce the gate count.

-keepAllFlipFlop

Keeps all flip-flops without removing functionally equivalent ones and those with constant inputs.

-keepRtlSymbol

Enables HDL Compiler to keep original VHDL RTL symbols, such as “.”, whenever possible. This option is useful when you want to use the RTL name to debug the design in *iRun* or the Browser. By default, the option is turned off. For example, the following command keeps the RTL name wherever possible:

```
hdlSynthesize -keepRtlSymbol <top_module>
```

Maps VHDL RTL signal name `a.b` to the netlist entry, `\a.b`. Without this modifier, the signal name would otherwise be converted to `a_b` in the netlist.

-parallel

Invokes distributed synthesis over hosts listed in the `parallel.host` file.

Note: When you specify this option with the `hdlSynthesize` command, do not use the `-all` option.

-parallel<number>

Starts the specified number of HDL Compiler synthesis processes on the current machine. Here, `<number>` is a required argument that should be greater than zero (0).

When this option is used, the following are not supported:

- Use of the `parallel.host` file
- Use of the `-all` option with the `hdlSynthesize` command

-parallelLSF<number>

Distributes the specified number of HDL Compiler synthesis processes using LSF. Here, `<number>` is a required argument that should be greater than zero (0).

Note: When you specify this option with the `hdlSynthesize` command, do not use the `-all` option.

-append

Appends new output to an existing output file, without overwriting the existing content.

-portNaming <pattern>

Configures the port names in the netlist for vectored ports.

The purpose of this option is to change the port names to interface with the netlists generated by tools which do not support `%s [%d]` port names.

-D<generic>=<value>

Assigns or replaces the default value of VHDL generics or Verilog parameters of the top module or at any level of the hierarchy.

Note: The `-D<generic>=<value>` option does not support sized Verilog parameters.

-rename <module_name>

Replaces the top module name in the HDL output files with the specified `<module_name>`.

You can specify different module names for each subsystem. When you do not want to figure out each individual module name for each subsystem, specifying `-rename 0` helps to automatically replace the top module name as if the subsystem is instantiated under a hierarchy.

For example, if the top module of a subsystem is defined as following:

```
module modA (...);  
parameter n1 = 2;  
parameter n2 = 2;  
parameter n3 = 2;
```

By default, the following `hdlSynthesize` command uses the original module name `modA` as the top module in the netlist:

```
hdlSynthesize -Dn1=0 -Dn2=2 -Dn3=3 modA
```

To replace the top module name `modA` with different names for each subsystem, specify the `{-all | -only <module_name> | <top_module> [-rename <module_name>]}` option as following:

```
hdlSynthesize -Dn1=0 -Dn2=2 -Dn3=3 -rename modA_0_2_3 modA  
hdlSynthesize -Dn1=5 -Dn2=200 -Dn3=30 -rename modA_5_200_30 modA
```

To automatically replace the top module name with different names for each subsystem, instead of the two commands illustrated above, specify the `-rename 0` option as shown below:

```
hdlSynthesize -Dn1=0 -Dn2=2 -Dn3=3 -rename 0 modA  
hdlSynthesize -Dn1=5 -Dn2=200 -Dn3=30 -rename 0 modA
```

-displayBuffer[+depth=<num>] [+timeOut=<num>] [+port=<num>]

Enables the memory buffer to improve performance when there are frequent occurrences of \$display. The three optional parameters that are provided with -displayBuffer option are:

- **+depth=<num>**: Specifies the depth of the memory for \$display buffer. The default value is 8192 (8K). Increase the depth to improve the performance if \$display still occurs frequently. Decrease the depth if there is a capacity limit. The maximum value of depth can be $2^{**}30$.
- **+timeOut=<num>**: Specifies the maximum number of emulation fast clock cycles per \$display, which can stay in the buffer. The default value is $2^{**}28$, or approximately 256 million cycles. If the fast clock is 500KHz, it means that a \$display message can be delayed for up to 128 seconds. Without such a limit, when a message does not occur, you will not know whether it is buffered, or whether something is wrong to prevent it from occurring. A smaller value also helps to reduce the width of the memory buffer. Increase the value to improve performance and decrease the value to reduce memory width. The range of the values is from $2^{**}20$ to $2^{**}30$. An error message will be reported during hdlSynthesize if an out-of-range value is given.
- **+port=<num>**: Specifies the number of ports of the memory for \$display buffer. The default value is 1. If the number of simultaneous \$display exceeds the number of ports, the messages are returned immediately without buffering. Increase the ports to reduce performance overhead caused by simultaneous \$display. Increasing the ports might increase emulation capacity or step count.

To identify the optimal number of ports and depth, turn on -displayDebug option for hdlConfig command at run time. Then, search for **depth=<num>** and **port=<num>** in additional debugging messages which begin with @<cycleNum>. Here, <cycleNum> is the value returned by `getCycleNum XEL` command.

-siliconSynthesisCompatible

Specifies that the design is compatible with the common silicon synthesis tools instead of simulation standards for selected constructs.

Examples

```
hdlSynthesize -only module1
```

Synthesizes only the module1 Verilog module in the design.

```
hdlSynthesize unit1
```

UXE Command Reference Manual

HDL Compiler Commands

Synthesizes the `unit1` VHDL module, and all levels of design hierarchy below it, in the design that you imported.

```
hdlSynthesize -memory -keepVhdlCase top
```

Synthesizes the design hierarchy under `top_module`, converts arrays into emulation hard macros when appropriate, and preserves the case of VHDL names.

```
hdlSynthesize top -D:sub0:feedback=30
```

```
hdlSynthesize top -Dtop.sub0.feedback=30
```

```
hdlSynthesize top -D:qrbs_glue_1:genQRBSV(4):QRBS:feedback=555
```

Overwrites the default VHDL generic or Verilog parameter `feedback` of the specified instance. The hierarchical path to the generic could be specified in VHDL style (:) or verilog style (.).

hdlWorkPath

Selects the working library into which you are importing the design. The working library is a directory, such as a subdirectory of your emulation design directory, in which you store some or all of your imported design files.

If you do not use `hdlWorkPath`, `hdlImport` imports your design files into a working library named `work`, representing a subdirectory named `work` in your emulation design directory.

If you use `hdlWorkPath` for a Verilog or VHDL design, it must precede `hdlImport` and `hdlInputFile`. `hdlImport` imports files, as specified in one or more `hdlInputFile` commands into the working library.

This command, when used without parameters, returns the name of the working library as set in the most recent `hdlWorkPath` command.

The syntax for this command is:

```
hdlWorkPath <dir>
```

<dir>

Directory for the working library into which you are importing your design.

Examples

```
hdlDefineLib mylib /myhome/design1/mylib  
hdlWorkPath /myhome/design1/mylib
```

Imports the VHDL design files into the `mylib` working library, which is a symbolic name for the `/myhome/design1/mylib` physical directory.

```
hdlWorkPath /myhome/design1/lib1
```

Imports the Verilog design files into the `lib1` working library in the `/myhome/design1` directory.

Database Commands

This chapter describes XEL commands by which you can make queries about the gate-level user design after the `designImport` command has run.

The following XEL commands are covered in this chapter.

- “[db](#)” on page 198
- “[moduleNet](#)” on page 202
- “[getDatabaseInfo](#)” on page 204

The following commands, described in the section [Deprecated Commands and Variables](#), provide an alternative way to make queries about the gate-level user design:

- [attr](#)
- [cell](#)
- [closelib](#)
- [inst](#)
- [lib](#)
- [net](#)
- [savelib](#)
- [term](#)
- [termInst](#)

db

Enables you to traverse and examine the design hierarchy. The db command is available in both xeCompile and xeDebug. It can be used interactively at the XEC> or XE> prompt, or in an XEL script.

In xeCompile, issue the design command before issuing the db command. In xeDebug, issue one of the commands host <hostname>, host -offline <filename>, or host -opendb before issuing the db command.

The syntax for this command is:

```
db cd <instance>
db pwd
db ls [-insts] [-nets] [-all] [<glob_pattern>...]
db find -name <pattern> [-scope <scope>] [-type net | term | inst | module] [-style
    glob | regex] -command
    [-case 0 | 1] [-levels <n>] [-max <n>] [-showgen 0 | 1]
```

cd <instance>

Changes the current hierarchy position. This position is known to, and used by, only the db command. <instance> can have several forms, as explained in [Examples](#).

pwd

Returns the current hierarchy position.

ls [-insts] [-nets] [-all] [<glob_pattern>...]

Returns instances, nets, or both at the current hierarchy position. The default is -all, which returns instances and nets as a Tcl list of two lists. The arguments -nets or -insts return only a single list. If <glob_pattern> is given, the result includes only instances and nets whose names match the specified glob pattern.

find

Searches the hierarchy at and below the current hierarchy position, or at and below a specified <scope>, for instances, terminals, or nets whose names match a specified pattern.

Alternatively, you can search for instances of modules whose names match a pattern. The result is a Tcl list of paths relative to the hierarchy top.

-name <pattern>

Specifies the name or pattern to search for. If the `-style` argument is given, then `<pattern>` is a glob or regex pattern; otherwise it is a name to be matched exactly. If the `-type` argument is given as `net`, `term`, or `inst`, only paths matching the specified pattern are returned. If the `-type` argument is given as `module`, paths of instances of modules whose names match the specified pattern are returned.

-scope <scope>

Specifies the hierarchy position where the search should start. If `-scope` is not specified, the current hierarchy position is used. If `-scope` is specified, `<scope>` must be either `/` (the hierarchy top), or a name (such as `u1`) or path (such as `u1.u2`) relative to the top.

-type net | term | inst | module

Returns paths of nets, terminals, instances or modules. The default is `net`.

-style glob | regex

Specifies the type (glob or regular expression) of the pattern given with the `-name` argument. If `-style` is not specified, the `-name` argument takes a name to be matched exactly.

-command

Searches for and returns the objects matching the specified search criteria. This option works as a Tcl script. You can use this option to print the objects to the screen, save all objects to a file, or find and call the `myproc` procedure with one or multiple arguments.

Examples

The following command prints all object names to the screen:

```
db find -name * -type net -style glob -command puts
```

The following code saves all objects to a file named `file.txt`:

UXE Command Reference Manual

Database Commands

```
set f [open "file.txt" "w"]
db find -name * -type net -style glob -command puts $f
close $f
```

The following command calls `myproc` with one argument for every found object:

```
db find -name * -type net -style glob -command myproc
```

The following command calls `myproc` with four parameters for every found object:

```
db find -name * -type net -style glob -command "myproc 1 2 3"
```

-case 0 | 1

Specifies case sensitivity. The default is 1 (case sensitive).

-levels <n>

Specifies the number of hierarchy levels to search. The default search is unlimited.

-max <n>

Specifies the maximum number of paths to return. To abort a search, press CTRL+C.

-showgen 0 | 1

Specifies whether or not to show generated objects. Generated objects are not part of the original design, but are added by the compiler. The default is 1 (show generated objects).

Examples

```
db cd /
```

Goes to the top of hierarchy.

```
db cd /a.b
```

Goes to position `a.b` relative to the top.

```
db cd a.b
```

Goes to position `a.b` relative to the current position.

```
db cd ..
```

Goes up one level from the current position.

UXE Command Reference Manual

Database Commands

```
db ls xyz*
```

Returns instances and nets at the current hierarchy position whose names start with "xyz".

```
db find -name .*reset.* -style regex
```

Returns nets at and below the current hierarchy position whose names contain "reset".

```
db find -name Q_FDP0 -type module
```

Returns the paths of all instances of module Q_FDP0 below the current position.

```
set mypaths [db find -name Q_FDP0 -type module]
set f [open myfile w]
foreach p $mypaths {puts $f $p}
close $f
```

Same as previous example, but write the paths to file `myfile`, one path per line.

moduleNet

Returns a list of pathnames of nets based on the specified search parameters (that is, library, module, and net name).

The wildcard characters * and ? can be used when specifying the search parameters. For a bus like D[15:0], names such as the following can be used:

- D (representing the entire bus)
- D[15:0] or D[3:5] (a range of bits)
- D[3] (an individual bit)

The syntax for this command is:

```
moduleNet [<lib>] <mod> <net>
```

<lib>

Specifies the name of the library that needs to be searched. If the <lib> argument is omitted, all libraries are searched.

<mod>

Specifies the name of a module. All instances of this module in the design will be searched.

<net>

Specifies the name of the net that needs to be searched.

Examples

```
moduleNet PULSE_* D
```

The above command returns the list of hierarchical pathnames of the net or bus D, in instances of modules whose name matches PULSE_*.

If u1 and u2 are instances of module PULSE_2, and D[1:0] is a bus internal to module PULSE_2, the command returns a list including the names u1.D[0], u1.D[1], u2.D[0], and u2.D[1]. (The command returns the highest-level hierarchical name for each net. Therefore, if D is a terminal of PULSE_2, the names returned will be at a level of hierarchy above u1 and u2.)

UXE Command Reference Manual

Database Commands

The XEL code given below prints the same list to file `myfile`, and also adds all the nets as `breakNets`. To improve the speed of command execution, instead of calling `breakNet` once for each net, a single `breakNet` command is given. The `eval` command is needed since without `eval`, the `breakNet -add` command tries to interpret the list as a single net name.

```
# get the list of hierarchical pathnames
set mynets [moduleNet PULSE_* D]

# print the names to file myfile
rm -f myfile
foreach n $mynets {redirect {puts "$n"} >> myfile }

# add the nets as breakNets
eval breakNet -add $mynets
```

getDatabaseInfo

Returns the design database information. The `getDatabaseInfo` command is available in `xeCompile`. It can be used interactively at the `XEC>` prompt, or in an XEL script.

The syntax for this command is:

```
getDatabaseInfo
```

Sample Output

```
Design Lib: LIB
Design Top Cell: cpu_test
Platform: PXP-II GXL
Mode: ICE
HW:
Host: cva-xeon114
Boards: 0.3
IES SW Path: /grid/avs/install/incisiv/15.2/latest
UXE SW Path: /lan/cva/home_cae/support/dpatel/LINKREL/uxe1410.fix
DBPath: /lan/cva/home_mnt/home/support/dpatel/testcases/RISC_PROCESSOR/ICE.1410
Number of domains: 1
Capacity: 6664
Utilization: 0.1%
Frequency: 3906
Trace Depth: 73688160
Vision Mode: FV
Clocking Mode: CAKE 1X
```

User Data Commands

This chapter describes XEL commands that modify, create, or save user data.



Caution

Use commands described in this chapter to change user data. Do not attempt to manually edit the User Data file. Using a UNIX editor could corrupt the file or add data to the file in the wrong sequence.

IP-related User Data commands, used in connection with the commands in this chapter, can be found in [Chapter 9, “IP Commands.”](#)

The following XEL commands are discussed in this chapter:

- [“aluTransform”](#) on page 208
- [“andMultiDrv”](#) on page 209
- [“breakNet”](#) on page 210
- [“breakNetHalfCycle”](#) on page 212
- [“breakPin”](#) on page 213
- [“breakPinHalfCycle”](#) on page 216
- [“cableConnection”](#) on page 217
- [“clockAssign”](#) on page 221
- [“clockDelay”](#) on page 224
- [“clockFrequency”](#) on page 225
- [“clockOption”](#) on page 227
- [“clockSource”](#) on page 231
- [“compilerOption”](#) on page 233

UXE Command Reference Manual

User Data Commands

- “[create assertion control](#)” on page 254
- “[customIsolationCell](#)” on page 255
- “[delayAsynClockInputs](#)” on page 256
- “[delayBox](#)” on page 258
- “[delayBoxInput](#)” on page 261
- “[emptyCell](#)” on page 263
- “[emulatorConfiguration](#)” on page 264
- “[globalNet](#)” on page 268
- “[hierAssign](#)” on page 269
- “[hwInfo](#)” on page 270
- “[identify power domain](#)” on page 275
- “[instanceFV](#)” on page 276
- “[instanceStub](#)” on page 278
- “[keepLoadlessInstance](#)” on page 279
- “[keepLoadlessModule](#)” on page 280
- “[keepLoadlessNet](#)” on page 281
- “[keepNet](#)” on page 282
- “[lp \(Low-power Compile-time Command\)](#)” on page 284
- “[memoryTransform](#)” on page 290
- “[memWritethru](#)” on page 295
- “[moduleFV](#)” on page 296
- “[moduleStub](#)” on page 297
- “[multiplexCable](#)” on page 298
- “[multiSampledTerminal](#)” on page 300
- “[netWeakDrive](#)” on page 301
- “[noBreakPath](#)” on page 303
- “[noOverlapClocks](#)” on page 304

UXE Command Reference Manual

User Data Commands

- “[orMultiDrv](#)” on page 305
- “[precompileOption](#)” on page 306
- “[primaryClockSource](#)” on page 335
- “[probe](#)” on page 336
- “[shadowsData](#)” on page 340
- “[stateRetentionControl](#)” on page 342
- “[symmetricConfiguration](#)” on page 343
- “[terminalAssign](#)” on page 345
- “[terminalOutputSync](#)” on page 350
- “[terminalTiming](#)” on page 352
- “[terminalWeakDrive](#)” on page 362
- “[tieNet](#)” on page 364
- “[ttcClear](#)” on page 365
- “[userData](#)” on page 366

aluTransform

Enables control over the ALU-to-gates transformations.

The syntax for this command is:

```
aluTransform -add {* alu2gates OFF}  
aluTransform -add {* alu2gates ON}  
aluTransform -rm {* alu2gates}  
aluTransform -add {* aluMaxUtilization <number>}  
aluTransform -rm {* aluMaxUtilization}
```

Note: The asterisk * in these commands is a placeholder. This command structure is similar to the structure of the `memoryTransform` command.

-add {* alu2gates OFF}

Specifies not to perform ALU-to-gates transformations.

-add {* alu2gates ON}

Specifies to perform ALU-to-gates transformations.

-rm {* alu2gates}

Restores the default behavior for ALU-to-gates transformations.

-add {*} aluMaxUtilization <number>}

Modifies the default maximum utilization percentage (100). Here, *<number>* is a percentage. Minimum value for *<number>* is 0 (no transformation is performed), and maximum value is 100.

-rm {*} aluMaxUtilization

Restores the default value for `aluMaxUtilization`.

andMultiDrv

Enables multiple drivers on a single net, and makes the value of the net equal to the AND of all the values driven.

A tristate driver affects the net's value only when the tristate driver is enabled. Supply0 and supply1 declarations are treated as drivers.

The syntax for this command is:

```
andMultiDrv -add {<signal>}
```

-add {<signal>}

Specifies the name of the signal on which multiple drivers are to be allowed.

Examples

```
andMultiDrv -add {sig_s1}
```

The above command enables multiple drivers on the sig_s1 signal and makes the signal value equal to the AND of all the driven values.

breakNet

Delays fanouts of a specified net. The `breakNet` command inserts a delay either for a functional purpose, or to improve emulation speed by breaking a long combinational path, or to manually break a combinational loop.



Improper use of this command can result in incorrect or broken design functionality.

This command applies to the Logic Analyzer, Synthesized Testbench, and Vector Debug modes.

Run this command after import and before precompile.

Using the `breakNet` command on a net, delays all the fanouts of the net within the design. However, it does not delay the net itself, nor does it delay any direct connection of the net to a target system. If you need to delay an output to a target system, use `breakNet` or `breakPin` commands to delay the inputs of the driver of the output net.

For nearly all flip-flops and latches in Palladium XP Reference Library, the QN output is generated internally as a fanout of the Q output. A `breakNet` on the Q output of such a flip-flop or latch causes the fanouts of QN to be delayed as well.

The syntax for this command is:

```
breakNet -add <net_name>  
breakNet -add {<net_name> <number_of_units>}  
breakNet -rm <net_name>
```

-add <net_name>

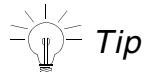
Delays fanouts of the specified net by one unit. This does not include fanouts directly to an external target system.

The amount of delay added by `breakNet -add {<net_name> N}` is determined as follows:

If CAKE oversampling is 2 or higher, and neither the `breakHalfCycle` option of the `precompile` command, nor multi-sampled IOs are used, the delay is N FCLK cycles. If the `breakHalfCycle` option is used, regardless of CAKE oversampling, the delay is $N/2$ FCLK

cycles. In all other cases, the delay can vary for different fanouts. However, the delay on each fanout is at least $N/2$ FCLK cycles, and no more than N FCLK cycles.

The `breakNet -add <net_name>` command is equivalent to the `breakNet -add {<net_name> 1}` command.



Tip
To improve emulation speed by breaking a long path, insert a full (not half) FCLK cycle delay. When `breakHalfCycle` option is used, the `breakNet -add {<net_name> 2}` command inserts a full FCLK cycle delay.

-add {<net_name> <number_of_units>}

Delays fanouts of the specified net by the specified number of units. This does not include fanouts directly to an external target system.

-rm <net_name>

Removes any previously specified delay for the net.

Examples

```
breakNet -add u1.u2.net25
```

Delays the fanouts of `u1.u2.net25` net by one unit, except a fanout to an external target.

```
breakNet -add {u1.u2.net25 2}
```

Delays the fanouts of net `u1.u2.net25` net by two units, except a fanout to an external target.

breakNetHalfCycle

The `breakNetHalfCycle` command has the same function as the `breakNet` command with the exception that when used in the CAKE 1X mode, the delay from `breakNetHalfCycle` is always one-half FCLK cycle (one-half cycle of the fastest design clock). For details, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

The syntax for this command is:

```
breakNetHalfCycle -add <net_name>
breakNetHalfCycle -add {<net_name>}
breakNetHalfCycle -rm <net_name>
```

breakPin

Inserts a delay element on an instance's I/O pins. This command can be used to break asynchronous feedback loops and/or critical paths without editing a design.

The critical path report is available only after full compilation, and is used to prepare a new round of compilation (with no need to re-import the design) in which new breaks to eliminate false/multi-cycle paths are manually specified.

The ud engine qualifies the hierarchical pin name(s). If a glob expression is used, the ud database stores only individual pin names, not glob expressions.

Run this command after import and before precompile.

To ensure that the `breakPin` command works properly, ensure that the hierarchical name of an instance pin should be specified as:

- `<hierarchical-instance-name>.<pin-name>`, or
- `<hierarchical-instance-name>_<pin-name>`

The related XEL command is `breakNet`.



Improper use of this command can result in incorrect or broken design functionality.

This command applies to LA, STB, and VD modes. Using the `breakPin` command on a pin that drives a net, delays the applicable fanouts of that net within the design. However, it does not delay the net itself, nor does it delay any direct connection of this net to a target system. If you need to delay an output to a target system, use `breakNet` or `breakPin` commands to delay the inputs of the driver of the output net.

The syntax for this command is:

```
breakPin -add { <hierarchical_pin_name> [ IFO | EFO | AFO ] }
```

-add

Adds a delay to the specified pin.

<hierarchical_pin_name>

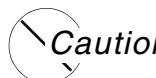
<*hierarchical-pin-name*> is the full hierarchical name of an instance pin. It has the format:

hierarchical-instance-name__pin-name

Or

hierarchical-instance-name.pin-name

pin-name can be a glob expression; in particular, * can be used to specify all the pins of the instance.



You should specify the hierarchical name of an instance pin in either of the following two formats:

- <*hierarchical-pin-name*>.<*pin-name*>

For example:

breakPin -add {U.A}

- <*hierarchical-pin-name*>_<*pin-name*>

For example:

breakPin -add {U__A}

IFO | EFO | AFO

Specifies the applicable fanouts of the pin to be delayed.

- IFO (Internal Fanout) specifies that only the fanouts of the pin(s) internal to the instance are to be delayed.
- EFO (External Fanout) specified that only the fanouts of the pin(s) external to the instance are to be delayed.
- AFO (All Fanout) specifies that all fanouts of the pin—either internal or external to the instance—are to be delayed.

If no fanouts are specified, the system uses the default setting. The default is determined by the true direction of the pin (that is, not based on its declared direction at the design source files, which might be wrong). If the pin is a true input, the default is IFO. If the pin is a true

UXE Command Reference Manual

User Data Commands

output, the default is EFO (in this case internal fanouts would not be delayed). If the pin is a true bidirectional (that is, connected to a tristate net, there is at least one driver of the net inside the instance, as well as one fanout), the default is AFO.

breakPinHalfCycle

The `breakPinHalfCycle` command has the same function as the `breakPin` command with the exception that when used in the CAKE 1X mode, the delay from `breakPinHalfCycle` is always one-half FCLK cycle (one-half cycle of the fastest design clock). For details, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

The syntax for this command is:

```
breakPinHalfCycle -add { <hierarchical_pin_name> [ IFO | EFO | AFO ] }
```

cableConnection

Specifies the cable connections between the emulator and the target system. This command is required for In-Circuit (ICE) operations and must be used before the icePrepare command.

Without any parameters, the `cableConnection` command returns a list of the existing cable connections.

The cable connections might also be verified by inspecting the user data available at the `<design>/QTDB/ud/userdata.cableConnections` path. Here, `<design>` refers to the emulation design directory from where the session was started.

The syntax for this command is:

```
cableConnection -add {<cable_label> <conn_loc> [I/O_technology]}  
cableConnection -add {<cable_label> LTA_A | LTA_B <conn_loc> [I/O_technology]}  
cableConnection -add {<cable_label> LTA_C | LTA_D <conn_loc>}  
cableConnection -rm <cable_label>  
cableConnection -dump [<filename>]  
cableConnection {-D2Y | -Y2D <cable_label>}  
cableConnection -clear
```

-add {<cable_label> <conn_loc> [I/O_technology]}

Adds an HDDC connection to target system. The specific components of this option are discussed below.

<cable_label>

Composes a user-specified interface name and the interface type in the following format:

`<interface_name>. <interface_type>`

The `<interface_type>` corresponds to a cable type. Following are the valid interface types:

- **HDDC:** It is the hardware connector used on the target-side of the BCA. HDDC is also known as VHDM and ADC (Active Data Cable).
- **EXC:** It refers to the two 8-pin headers on the LTA (Legacy Target Adapter). The LTA converts the HDDC connection into two 100 pin SCSI connectors and two 8-pin headers.

UXE Command Reference Manual

User Data Commands

- **MTIB-88:** It can be plugged into the LTA. The MTIB-88 converts the 100 pin SCSI connector into three ribbon cables.
- **TIB-192:** It converts the VHDM connector into seven ribbon cables. The TIB-192 does not use the LTA. The IP chassis uses the TIB-192.
- **DATA_CABLE:** It refers to the legacy Woven cable. The DATA_CABLE uses an LTA and is also known as TCC (Target Connection Cable).

<conn_loc>

Specifies the location of the emulator connector to which the cable connects in the `<board_number>_J<LFH_number>` format. The `<board_number>` ranges from 0 to (maximum number of boards per system - 1). The `<LFH_number>` ranges from 1 to 24 because each logic board supports 24 target connectors.

<I/O_technology>

Specifies any one of the following supported I/O technologies:

- LVCMOS15_x (where x could be 8, 12, 16, or series)
- LVCMOS18_x (where x could be 8, 12, 16, or series)
- LVCMOS25_x (where x could be 8, 12, 16, 24, or series)
- LVCMOS33_x (where x could be 8, 12, 16, 24, or series)

If the I/O technology is not specified, by default LVCMOS33_8 is used. The I/O technology also indicates the drive strength of the cable. For example, 15 stands for 1.5V and 18 stands for 1.8V.

For detailed information about the possible values for the `<I/O_technology>` option, refer to the *I/O Technology* section of the *High Density Data Cable* chapter, *Palladium Target System Developer's Guide*.

-add {<cable_label> LTA_A | LTA_B <conn_loc> [I/O_technology]}

Adds an HDDC cable using an LTA connection. Using this format turns it into one or two target cables of type MTIB-88 and/or TCC (interface type name: DATA_CABLE). Ensure that the LTA_A and LTA_B connectors have the same I/O_technology.

-add {<cable_label> LTA_C | LTA_D <conn_loc>}

Specifies the connector location on the LTA connection. There are 16 pins on the LTA that are available as individual target interface pins. These are separated into two groups, with 8 pins per group. Since a group also corresponds to a connector, the connector location on the LTA is identified by either LTA_C or LTA_D. The interface type for these pins is called EXC. Of these 16 pins, eight are always ground. Therefore, only four pins per connector can be used.

-rm <cable_label>

Removes the specified <cable_label> from the list of cables connecting the emulator to the target system.

-dump [<filename>]

Saves the specified cable connection settings, in XEL format, to a specified file or the stdout (screen).

-D2Y | -Y2D <cable_label>

A target interface cable (with MUX repeater) can use standard (D configuration) cable setup, which enables up to 88 target (data) signals, and up to 20 control signals. If you are using less than 88 target signals, use extra signals as additional control signals as long as the total of all signals on the target cable (called the A cable) does not exceed 108.

If target connectors require more than 108 total signals, or more than 54 independent enable signals, add a second DC1002 interface cable (called the B cable), and use a DC1002 Y-Adapter to connect the A and B cables together (Y configuration).

In a Y configuration, use the **-D2Y** parameter of this command to turn the standard D cable into the A cable, but also use a separate **-add** parameter to add the B cable. You can also use the **-Y2D** parameter to drop the B cable and turn the A cable back into a standard D cable.

-clear

Removes all the previously defined cable connections from the database.

Examples

- To add a TIB-192 connection, use:

UXE Command Reference Manual

User Data Commands

```
cableConnection -add {myCable3.TIB-192 2_J4 LVCMOS15_series}
```

- To add an LTA connection, use:

```
cableConnection -add {mycable1.MTIB-88 LTA_A 2_J4 LVCMOS33_16}  
cableConnection -add {mycable2.MTIB-88 LTA_B 2_J4 LVCMOS33_16}
```

The above commands add a cable connection supporting two MTIB-88 adaptor connections, one on the A connector of the Legacy Target Adapter (LTA) and one on the B LTA connector.

- To add an EXC connection, use:

```
cableConnection add {myIndividualPins.EXC LTA_C 2_J8}
```

- To remove all cable connections defined, use:

```
cableConnection -clear
```

clockAssign

Specifies how the behavior of a signal (for example *<signal1>*) is derived from the behavior of another signal (for example *<signal2>*) irrespective of how *<signal1>* is originally driven in the design. Both of the signals must be signals in the design but it is not necessary that they be clock signals.

The syntax for this command is:

```
clockAssign -add {<signal1>[~] <signal2> [<N>]}  
clockAssign -rm <signal1>
```

-add {<signal1>[~] <signal2> [<N>]}

Specifies that the behavior of *<signal1>* should be derived from the behavior of *<signal2>*, regardless of how *<signal1>* is driven in the design.

<N> and ~ are optional parameters.

The value of *<N>*, must either be 1, or be a positive even number (2, 4, 6, ...).

If the value of *<N>* is not specified or specified as 1:

- *<signal1>* behaves as *<signal2>*, if ~ is not specified.
- *<signal1>* behaves as the negation of *<signal2>*, if ~ is specified.

If *<N>* is a positive even number (2, 4, 6, ...), *<signal1>* is the output of a “divider” circuit that generates a signal changing *<N>* times less often than *<signal2>*. In this case:

- *<signal1>* changes on the rising edge of *<signal2>*, if ~ is not specified.
- *<signal1>* changes on the falling edge of *<signal2>*, if ~ is specified.

-rm <signal1>

Removes the signal assignment specified for the behavior of *<signal1>*.

Examples

Example 1

```
clockAssign -add {sig_s1 sig_s2 1}
```

In this example:

- A value of 1 has been specified for N
- \sim has not been specified

Therefore, `sig_s1` behaves as `sig_s2`.

Note: If no value is specified for N , the result is the same, that is, `sig_s1` behaves as `sig_s2`.

Example 2

```
clockAssign -add {sig_s1 ~ sig_s2}
```

In this example:

- No value has been specified for N
- \sim has been specified

Therefore, `sig_s1` behaves as the negation of `sig_s2`.

Example 3

```
clockAssign -add {sig_s1 sig_s2 4}
```

In this example:

- A value of 4 has been specified for N
- \sim has not been specified

Therefore, `sig_s1` is the output of a “divider” circuit that generates a signal changing four times less often than `sig_s2`. For example, if `sig_s2` changes at every 1 ns, `sig_s1` changes at every 4 ns.

The divider output changes on the rising edge of `sig_s2`.

Example 4

```
clockAssign -add {sig_s1 ~ sig_s2 8}
```

In this example:

- A value of 8 has been specified for N
- \sim has been specified

UXE Command Reference Manual

User Data Commands

Therefore, `sig_s1` is the output of a “divider” circuit that generates a signal changing eight times less often than `sig_s2`. For example, if `sig_s2` changes at every 1 ns, `sig_s1` changes at every 8 ns.

The divider output changes on the falling edge of `sig_s2`.

Example 5

```
clockAssign -rm sig_s1
```

Removes the signal assignment specified for the behavior of `sig_s1`.

clockDelay

Adds, removes, or modifies user clock data. This command cannot be used when design is compiled for 1X mode.

Also, this command cannot be used at run time. At run time, use the [clockConfig](#) command instead.

The syntax for this command is:

```
clockDelay [-add | -rm] {<clockname> <nDly>}
```

<clockname>

A text string corresponding to signal name syntax.

<nDly>

A whole number greater or equal to zero. *<nDly>* is interpreted as delay in FCLK cycles.

clockFrequency

Specifies the frequency of any specific clock source. This command specifies the relative frequencies of different clocks, which are generated automatically using CAKE or SCM technology.

Note:

- Only the relative frequencies, not the absolute frequencies, are significant for this command.
- A frequency of zero has a special meaning. When inserting clock generation logic, the compiler considers a clock source with frequency zero to be an ordinary net and does not insert clock generation logic. However, during loopbreaking, the net is considered to be a clock net. For more information, refer to the *Loop-breaking* appendix in the *UXE User Guide*.
- If you declare a clock source net with non-zero frequency, you can force it to have a constant value at run time by forcing the net to 0 or 1.

The `clockFrequency` command cannot be used with CAKE clocks at run time. For run-time use with CAKE, use the `clockConfig` command instead.

This command should be used before the `precompile` command to display the existing clock sources with frequency information. If no clock source is set, nothing is returned.

Frequencies specified without units are considered to be in MHz.

The defined frequencies are internally converted to whole numbers of kHz. Also note that the fractions of kHz are dropped with a warning. Thus, 123456 Hz and 123789 Hz both mean the same frequency, that is, 123 kHz.

The frequency (in kHz) of the fastest clock multiplied by the oversampling ratio must be less than 2^{31} (that is, about 2.1 billion).

Once the frequencies have been converted to whole numbers, the frequency ratio between each clock and the fastest clock is implemented exactly according to these whole numbers, irrespective of the number of clocks and their frequencies.

For example, consider there are three clocks of frequencies 1000003, 1000002, and 456789. With CAKE 2, during the first 1000003 FCLK cycles run after downloading, there are 1000003 phases of the fastest clock, 1000002 phases of the second fastest clock, and 456789 phases of the third fastest clock, and then the pattern repeats. With CAKE 1, during the first 1000003 FCLK cycles run after downloading, there are 1000003 cycles of the fastest clock, 1000002

UXE Command Reference Manual

User Data Commands

cycles of the second fastest clock, and 456789 cycles of the third fastest clock, and then the pattern repeats.

With SCM, for each 1000003 phases of the fastest clock, there are 1000002 phases of the second fastest clock and 456789 phases of the third fastest clock.

If you specify `clockSource` without `clockFrequency`, the net would be a non-instrumented clock. This means that this net is driven normally from the netlist, but it is considered a clock for loop-breaking purposes (paths from this clock should not be broken).

Note: The `clockSource` command is not required if you have specified the `clockFrequency` command.

The syntax for this command is:

```
clockFrequency -add {<clock_source_name> <frequency>}
```

```
clockFrequency -rm <clock_source_name>
```

-add {`<clock_source_name>` `<frequency>`}

Adds a frequency for a specified clock source.

-rm `<clock_source_name>`

Removes the frequency for a clock source, but not the clock source.

clockOption

Specifies the technology that will be used for automatic clock generation.

The `clockOption` command is available only at compile time. To change clocking options at run time, use the `clockConfig` command.

In the ICE mode, issue this command after the `design` command and before the `precompile` command.

In the SA mode, if you use `ixclkgen` utility to generate clocks, include the `clockOption -add {technology CAKE <number>}` command in the input file for `ixclkgen`. The `ixclkgen` utility supports CAKE technology, not SCM.

The syntax for this command is:

```
clockOption -add | -rm {<option_Name> <value>}  
clockOption -add {technology [CAKE <number> | SCM]}  
clockOption -add {activeEdge both | positive | negative}  
clockOption -add {ignoreOutput ON | OFF}
```

-add {technology [CAKE <number> | SCM]}

Specify whether CAKE or SCM technology should be used for automatic clock generation. CAKE and SCM technologies differ in the edge relationships that are created between different clocks. See the following examples for details. For CAKE technology, the number specifies the oversampling ratio, which is defined as the number of FCLK cycles per cycle of the fastest design clock. By default it is set to 2. The value of `<number>` must be either 0.5 or a positive whole number.

In both technologies (CAKE and SCM) the initial value and the precise initial behavior of a clock is unspecified. The times at which the edges occur might not match the times of occurrences of the edges of similar clocks that were specified using simple RTL code.

-add {activeEdge both | positive | negative}

This option is not required, and has been made obsolete.

-add {ignoreOutput ON | OFF}

When `ignoreOutput` is set to `OFF`, precompile refuses to implement `1X` mode if any primary output can change on the non-active edge of the fastest design clock, unless the Multisampled I/O feature is enabled.

When `ignoreOutput` is set to `ON`, if a primary output signal can change on the non-active edge and Multisampled I/O feature is not enabled, `1X` mode is implemented. However, the output to the target will change only once per FCLK, and will have the value computed for the signal in the active phase.

Use the [compilerOption](#) command to enable/disable the Multisampled I/O feature.

Default is `OFF`.

Examples of Clock Generation

Following examples illustrate the generation of clocks through CAKE and SCM technologies.

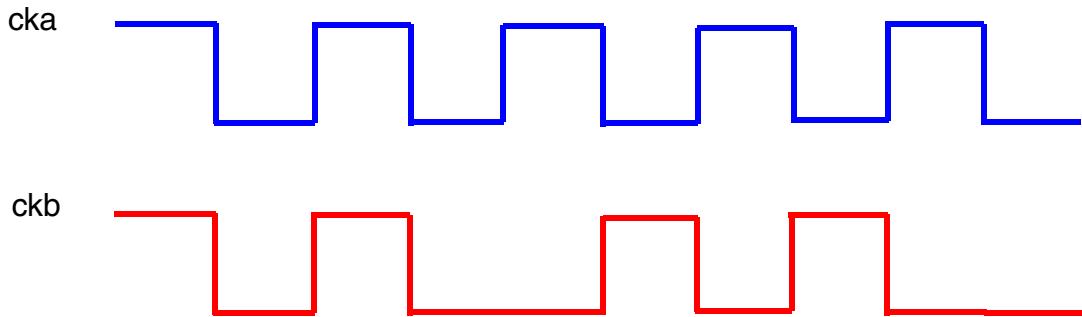
CAKE 2X Mode:

CAKE stands for Clock Averaging with Koincident Edges. In CAKE technology, every edge of a slower clock coincides with an edge of the fastest clock. The period of a slower clock might not be constant and the duty cycle might not be 50%. However, the ratio of average frequencies (`slow_clock_frequency/fastest_clock_frequency`) is exactly as specified.

```
clockOption -add {technology CAKE 2}
clockFrequency -add { cka 5}
clockFrequency -add { ckb 4}
```

This example generates two clocks using CAKE technology, with oversampling ratio 2. (CAKE with oversampling ratio 2 is sometimes called CAKE 2X mode). Since the oversampling ratio is 2, there are two FCLKs per cycle of the fastest clock `cka`. The relative frequencies of clocks `cka` and `ckb` are 5:4. For each 5 phases of `cka`, there are 4 phases of `ckb`.

The waveforms are as follows:



SCM Mode:

SCM stands for Simulation Control Module. With SCM, for a set of clock sources of specified frequencies, the sequence of edges of all clocks is the same as the sequence for clocks of the specified frequencies, each having 50% duty cycle, for which the initial rising edges coincide. In cases where SCM generates slower clocks whose edges do not coincide with edges of the fastest clock, extra cycles of the emulator's FCLK are needed compared to CAKE technology. So, time to run a given test is longer. Cadence recommends using CAKE technology rather than SCM, because CAKE gives better performance. Cadence is not aware of any customer designs that need SCM technology for correct function.

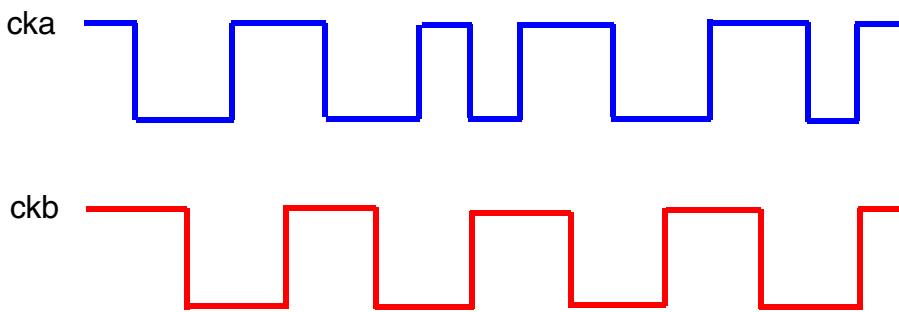
```
clockOption -add {technology SCM}
clockFrequency -add { cka 5}
clockFrequency -add { ckb 4}
```

This example generates two clocks using SCM technology. The relative frequencies of clocks `cka` and `ckb` are 5:4. For each 5 phases of `cka`, there are 4 phases of `ckb`.

UXE Command Reference Manual

User Data Commands

The waveforms are as follows:



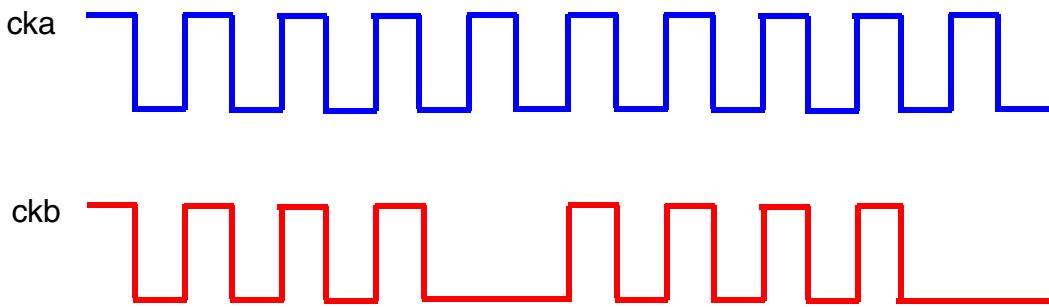
CAKE 1X Mode:

In CAKE 1X mode, there are only half as many FCLKs per design clock cycle as in 2X mode. If the FCLK frequency remains the same, the emulation speed in 1X mode is double the speed in 2X mode. In CAKE 1X mode, every posedge of a slower clock coincides with a posedge of the fastest clock.

```
clockOption -add {technology CAKE 1}
clockOption -add {ignoreOutput On}
clockFrequency -add { cka 5}
clockFrequency -add { ckb 4}
```

This example generates two clocks using CAKE technology, with oversampling ratio 1. As the oversampling ratio is 1, there is one FCLK per cycle of the fastest clock `cka`. The relative frequencies of clocks `cka` and `ckb` are 5:4. For each 5 cycles of `cka`, there are 4 cycles of `ckb`.

The waveforms are as follows:



clockSource

Selects which nets in your design are the sources of clock signals.

Before using the `clockSource` command, execute [design](#) to specify the design library and cell name. This command should be issued before precompile.

Without parameters, the `clockSource` command returns the current clock sources. If no clock source is set, no value is returned.

You can specify any net in the design as the clock source. It need not be a top-level input; any hierarchical signal path can be used.

When a clock source is driven using the CAKE or SCM technology, any design logic driving the net is automatically disconnected. If the signal is a primary input or inout signal, it is automatically changed to a primary output.

Nets specified as clock sources must exist in the design before the `clockSource` command is given. You can use the `iScopedNets` program to add a net to the design so that it can be specified as a clock source. For details about the `iScopedNets` program, refer to the *Connecting Nets Using iScopedNets Program* section in the *Compiling Designs for In-Circuit Emulation with xeCompile* chapter of *UXE User Guide*.

Adding such a net could be useful if the net will be connected during precompile because of a `userGlobalNet` property. Note that when using the CAKE 1X mode, you might want to add a dummy clock that is faster than the fastest real design clock (See [Example 3](#) below).

If you specify `clockSource` without `clockFrequency`, the net would be a non-instrumented clock. This means that this net is driven normally from the netlist, but it is considered a clock for loop-breaking purposes (paths from this clock should not be broken).

Note: The `clockSource` command is not required if you have specified the `clockFrequency` command.

The syntax for this command is:

`clockSource -add <net_name>...`

`clockSource -rm <net_name>...`

-add <net_name>...

Defines `<net_name>` as a clock source for your design.

-rm <net_name>...

Removes *<net_name>* as a clock source for your design.

Example1

```
clockSource -add my_net1
```

Selects the net `my_net1` as the clock source.

Example 2

```
clockSource -rm my_net1
```

Removes the net `my_net1` as the clock source for your design.

Example 3

The following example adds the `my_clock` net to the design top cell, and then specifies it as a clock source.

```
iScopedNets -VERSION3 my_lib my_top - my_clock . h  
clockSource -add my_clock
```

For a complete list of options of the `iScopedNets` command, execute the `iScopedNets` command without any arguments.

compilerOption

Sets options that determine how to compile your design for emulation on the Palladium XP system. This is a required command for the Palladium XP system and must be executed before running the [precompile](#) and [icePrepare](#) command sequence.

Use of InfiniTrace mode does not require any specific settings. InfiniTrace mode is supported for DYNP, and FV. It is supported with static targets in ICE mode.

FullVision in Palladium XP is switched on or off at compile time by using the `visionMode` parameter. This setting must be applied prior to running the `precompile` command. FullVision is the default setting for all modes.

In UXE, memory utilization and transformation are handled through the [memoryTransform](#) command. For more information on the use of this command, refer to the *Compiling and Running Designs with Memories* chapter of *UXE User Guide*.

The syntax for this command is:

```
compilerOption -add | -rm {<parameter> <value>}...  
compilerOption -add | -rm {allWiresParityCheck ON | OFF}  
compilerOption -add {mode vd | ice}  
compilerOption -add {ioDelayUnit pd2tif | pd2step | uatif | step | ns}  
compilerOption -add {visionMode DYNP | FV}  
compilerOption -add {enableMultiSampledIO 0 | 1 | 2}  
compilerOption -add {delayBidirIO OFF | ON}  
compilerOption -add {infiniTrace OFF | ON}  
compilerOption -add {memoryWritethru OFF | ON}  
compilerOption -add {noTailgate OFF | ON}  
compilerOption -add {sdllInstances <num_instances>}  
compilerOption -add {sdlTraceDepth <sdl_trace_depth>}  
compilerOption -add {sdlDisplayDepth <sdl_display_depth>}  
compilerOption -add {sdlDisplayWidth <sdl_display_width>}  
compilerOption -add {sdlGlobalDisplay <sdl_global_display>}  
compilerOption -add {traceDepth <value>}  
compilerOption -add {traceDepthDYNP <value>}  
compilerOption -add {numCapturedNetsPerDomain <n>}  
compilerOption -add {numExtraCapturedNetsPerDomain <n>}
```

UXE Command Reference Manual

User Data Commands

```
compilerOption -add {compilerEffort LOW | HIGH}
compilerOption -add {optimizationEffort LOW | HIGH}
compilerOption -add {optimizationAlgo CLASSIC | MODERN}
compilerOption -add {placementEffort LOW | HIGH}
compilerOption -add {powerdb <powerdb_directory>}
compilerOption -add {min_dcc_percentage <n>}
compilerOption -add {max_dcc_percentage <n>}
compilerOption -add {limfanout <n>}
compilerOption -add {retiming OFF | ON}
compilerOption -add {stepMode 256 | 512 | auto}
compilerOption -add {enhancedTerminalTiming ON | OFF}
compilerOption -add {symmetricPreserveCables ON | OFF}
compilerOption -add {symmetricPreserveMemCards ON | OFF}
compilerOption -add {wireDuplication <value>}
compilerOption -add {ModelChecker OFF | ON}
compilerOption -add {write_pgm OFF | ON}
compilerOption -add {write_fnets OFF | ON}
compilerOption -add {ModelCheckerOmit "<check_type> <check_type> ... "}
compilerOption -add {cpfLogBufferDepth <n>}
compilerOption -add {1801LogBufferDepth <n>}
compilerOption -add {1801LogType { [powerDomain] [supplySet] [isolation] [retention] [powerState] [stateTransition] [portState] [pstState] [illegalIsoState] [illegalRetState] } }
compilerOption -add {1801CoverageWidth [0-8]}
compilerOption -add {XPllCapacityFeatures <N>}
compilerOption -add {symmetricXPllonly ON}
compilerOption -add {maxStep <number>}
compilerOption -add {minStep <number>}
compilerOption -add {DPATopNInst N}
compilerOption -add {subDPATopNInst <fileName>}
compilerOption -add {dumpDPATopNInst N}
compilerOption -rm {<compiler_option>}...
```

-add | -rm {<parameter> <value>}...

Adds or replaces the specified compiler option.

-add | -rm {allWiresParityCheck ON | OFF}

This option enables parity checking for all wires when the design is compiled in more than 512 steps or the stepMode value. (See [-add {stepMode 256 | 512 | auto}](#)). Thus, when the stepMode setting is 256, parity checking will be enabled if the design compiles in more than 256 steps.

If the design compiles in less than 512 steps, all wires are always parity checked. Specifying the allWiresParityCheck option has no effect. However, if the design compiles in more than 512 steps, enabling the option increases the step count. The step count is rounded up to the next multiple of 512. So, if the design originally compiled in 800 steps, after enabling this option, the design will compile in 1024 steps.

You need to trade off this impact on performance with the possibility of unreported run-time hardware errors, which can cause DUT or testcase failures.

-add {mode vd | ice}

Selects one of the following compile modes:

- **vd**: Vector Debug
- **ice**: In-Circuit Emulation for running in STB mode (targetless or static-target environment), or Logic Analyzer mode (dynamic target).

The default mode is **vd**, Vector Debug.

-add {ioDelayUnit pd2tif | pd2step | uatif | step | ns}

Sets the units for the timing given with the `terminalTiming`, `delayBox`, and `delayBoxInput` commands. Therefore, the `ioDelayUnit` setting should be defined before these commands.

Following are the valid options for the `ioDelayUnit` settings:

- **pd2tif**: Refers to a real number in `ns` (nanoseconds) for Palladium III hardware that is calculated using the following formula:

$$n \text{ pd2tif} = n * (4 \text{ steps/pd2tif}) * (5.3\text{ns /step})$$

- **pd2step:** Refers to a real number in ns (nanoseconds) that is calculated using the following formula:
$$n \text{ pd2step} = n * (5.3\text{ns} / \text{pd2step})$$
- **uatif:** Refers to a real number in ns (nanoseconds) for Palladium XP hardware that is calculated using the following formula:
$$n \text{ uatif} = n * (8 \text{ steps/uatif}) * (2\text{ns} / \text{step})$$
- **step:** Refers to 2ns for Palladium XP hardware and 5.3ns for Palladium III hardware. When any error occurs, the `ioDelayUnit` setting defaults to `step`.
For example, 40 `uatifs` correspond to 640ns, which is calculated as follows:
$$(40 * 8\text{steps/uatif} * 2\text{ns/step}) \text{ ns}$$
- **ns:** Refers to nanoseconds. This I/O delay unit has the advantage of being based on absolute time, thus ensuring consistent functioning across changing architectures.

To provide backward compatibility for the existing compile scripts, software does not generate an error on finding a mismatch between the specified hardware and `ioDelayUnit`. If the `compilerOption` does not match the specified hardware, such as `pd2tif` is specified with Palladium III hardware or `uatif` with Palladium XP hardware, the software converts the units internally. For example, 40 `pd2tifs` are interpreted as 53 `uatifs`. Therefore, after changing the hardware in the compile scripts, each unit for the given timing need not be changed manually. The software automatically recalculates the units based on the interacting hardware.

After setting the `ioDelayUnit`, the changes in the `terminalTiming` units can be verified from the `.ctl` file; whereas, for `delayBox` and `delayBoxInput` units, verify the changes in the `.iodelays` file.

In the `terminalTiming -edgeinfo` command, the `<edge_list>` option specifies the number of steps, regardless of the `ioDelayUnit` setting. To keep the same timing when migrating a design from Palladium to Palladium XP, increase the step numbers by a factor of approximately 2.65.

-add {visionMode DYNP | FV}

FV indicates FullVision. The FullVision mode is the default mode. At compile time, these modes are supported through the `compilerOption` command, which is controlled by the switch options listed under the `compilerOption` command.

By default, FV is coupled with the InfiniTrace mode, which is supported with ICE mode.

DYNP option corresponds to selecting the dynamic probe mode. Subsequent compilerOption commands generate a database only with compiled dynamic probes.

In ICE and Vector Debug modes, dynamic probes automatically includes the top-level i/o signals. You need not specify these signals as probes to be able to view their values at run time.

-add {enableMultiSampledIO 0 | 1 | 2}

Determines whether target I/O signals are multisampled.

- If this option is set to 0, which is the default setting, no multisampling of target I/O is done.
- If this option is set to 1 or 2, an input from the target is multisampled if that input is declared with the multiSampledTerminal command.
- When the value is set to 1, the compiler determines whether each output should be multisampled based on how the signal is generated by the design.
- When the value is set to 2, an output is multisampled only if the compiler determines that this output should be multisampled, and this output is declared with the multiSampledTerminal command.
- The input component of a bidirectional I/O signal and the output and enable components (if the value is set to 2) are multisampled only if the signal is declared with the multiSampledTerminal command.

This is considered as an expert-user feature. For details, refer to Using Multisampling.

-add {delayBidirIO OFF | ON}

Specifies whether the software should automatically insert a delay on the output-to-input path of every bidirectional primary I/O signal that is not assigned to a target cable. The default setting is on. If the setting is switched to on, a delay is inserted. The delay is always one FCLK, even if the -breakHalfCycle option has been used.

-add {infiniTrace OFF | ON}

Specifies whether to enable InfiniTrace or not. The default value is on. This command can be used with any vision mode.

Usually, it is not required to turn off InfiniTrace because the InfiniTrace overhead is minimal. However, in rare cases, some designs can have a large amount of primary input. In such

cases, InfiniTrace overhead can be significant, and it can cause compilation to fail. To avoid such situations, you can use this command to turn off InfiniTrace.

-add {memoryWritethru OFF | ON}

Indicates if all memories need to be implemented as write-through or not. This implementation applies to all the memories except for the memory instances whose write-through or non-write-through implementation is specified individually using the [memWritethru](#) command.

The default value is **ON**, which means that all memories are implemented as write-through. If **memoryWritethru** is **OFF**, all user memory instances are implemented as non-write-through. It might reduce the step count.

The options values, **ON** and **OFF**, are case-insensitive.

-add {noTailgate OFF | ON}

Specifies whether the software should ignore all target terminal assignments. The default setting is **off**. If the setting is switched to **on**, terminal assignments are ignored, and the design is compiled as though the user data did not include any `cableConnection` or `terminalAssign` specifications.

-add {sdllInstances <num_instances>}

Specifies the number of SDL instances that the SDL instrumentation needs to support.

<num_instances> is the number of SDL instances that should be instrumented. At run time, *<num_instances>* will be the maximum number of SDL instances available for use.

If *<num_instances>* is not specified, a default of one SDL instance will be used to reduce capacity overhead. Maximum 64 SDL instances can be specified.

-add {sdlTraceDepth <sdl_trace_depth>}

Specifies the maximum number of samples that the SDL trace can collect without overwriting itself. If not specified, the default size is 16k samples.

<sdl_trace_depth> is a positive decimal number indicating the size, in samples, of the SDL trace buffer. The specified value can be appended with **k** or **M**. For example, **32k** means **32 x 1024**.

The specified number is rounded up to the nearest power of 2. Values below 1k are rounded up to 1k; and values above 16M are truncated to 16M.

Increasing the SDL trace depth adversely affects the emulation memory capacity at a ratio of 20bytes per sample per SDL instance. For example, setting the `sdlTraceDepth` option to 1M and the `sdlInstances` option to 5, takes up 100MB of emulation memory, calculated as following:

$$1M \times 5 \times 20 = 100MB$$

-add {sdlDisplayDepth <sdl_display_depth>}

Specifies the depth of the Memory buffer which collects the DISPLAY requests from SDL. Each SDL evaluation cycle in which one or more DISPLAY statements are active will take one word in the DISPLAY buffer.

<sdl_display_depth> is a positive decimal number indicating the maximal number of words to be allocated for the SDL display buffer. The value is rounded up to the nearest power of 2. Values below 1k are rounded up (to 1k). Values above 16M are truncated to 16M. If unspecified, the default is set to 16k.

The specified value can be appended with k or M (for example, 32k means 32×1024).

Increasing the display buffer depth comes on the expense of emulation memory at a ratio of:

$$D * (G/8 + W/8 + 8*N)$$

Where,

- D is the depth as specified by `-add {sdlDisplayDepth <sdl_display_depth>}`
- G is the value specified by `-add {sdlGlobalDisplay <sdl_global_display>}`
- W is the value specified by `-add {sdlDisplayWidth <sdl_display_width>}`
- N is the value specified by `-add {sdlInstances <num_instances>}`

For example, the display buffer for a display depth of 4k, 64 global DISPLAY statements, a display width of 1024, and 3 SDL instances, will consume 640kB of emulated memory:

$$(64/8 + 1024/8 + 8*3) * 4096 = 655,360 \text{ (equivalent to 640kB)}$$

-add {sdlDisplayWidth <sdl_display_width>}

Specifies the maximal number of different scalar arguments from all DISPLAY statements in the SDL program.

<*sdl_display_width*> is a positive decimal number. This number is rounded up to the nearest multiple of 64. If unspecified, its default value is 128.

Increasing this value comes on the expense of emulation memory. The effect is explained in the [-add {sdlDisplayDepth <sdl_display_depth>}](#) compiler option description above.

A vector argument with *N* bits is equivalent to *N* scalar arguments.

The %t escape sequence in the format specifier in the DISPLAY statement is equivalent to a vector with 64 bits under ICOM flow and 48 bits under ICE flow.

If the same signal appears as an argument in more than one DISPLAY statement in the SDL program, that signal is counted only once for the purpose of calculating the required buffer width.

-add {sdlGlobalDisplay <sdl_global_display>}

Specifies the maximal number of global DISPLAY statement (DISPLAY statements that are not defined inside STATE statements) in SDL that will be allowed at run time.

<*sdl_global_display*> is a positive decimal number. This number is rounded up to the nearest multiple of 64. If unspecified, its default value is 64.

Increasing this value comes on the expense of emulation memory. The effect is explained in the [-add {sdlDisplayDepth <sdl_display_depth>}](#) compiler option description above.

-add {traceDepth <value>}

Specifies the logic analyzer trace depth that the compiler is requested to implement in all vision modes. The <*value*> parameter can be any value which is less than the maximum trace depth.

The default value is *auto*. When the value is set to *auto*, the compiler attempts to set the trace depth to the largest value that can be implemented without the risk of compilation failure. Note that, if you set the value too high, it could cause a slower emulation speed or the compile might fail. If the compile fails, you can:

- Try multiple compiles by using, for example, the `compileFind` command.

- Increase the emulator resources, that is, number of domains or boards.
- Use a smaller *<value>* parameter, or specify the value as `auto`.

The trace depth in FV is `auto` by default. You can increase the FV trace depth to 43M cycles for Palladium XP and 75M cycles for Palladium XP II. However, using this option might increase the emulator resources required for a successful compile.

Note: The Palladium XP XL and Palladium XP GXL systems have 4GB DCC, whereas the Palladium XP II system has 8GB DCC. If 8GB DCC is installed in a Palladium XP system, it will be downgraded to 4GB DCC.

-add {traceDepthDYNP <value>}

Specifies the logic analyzer trace depth that the compiler is requested to implement when using FV vision mode database as DYNP vision mode at run time. The *<value>* parameter can be any value which is less than the maximum trace depth.

The default value is `auto`. When the value is set to `auto`, the compiler attempts to set the trace depth to the largest value that can be implemented without the risk of compilation failure. Note that, if you set the value too high, it could cause a slower emulation speed or the compile might fail. If the compile fails, you can:

- Try multiple compiles by using, for example, the `compileFind` command.
- Increase the emulator resources, that is, number of domains or boards.
- Use a smaller *<value>* parameter, or specify the value as `auto`.

The trace depth in DYNP vision mode is `auto` by default. You can increase the trace depth to 43M cycles for Palladium XP and 75M cycles for Palladium XP II. However, using this option might increase the emulator resources required for a successful compile.

Note: The Palladium XP XL and Palladium XP GXL systems have 4GB DCC, whereas the Palladium XP II system has 8GB DCC. If 8GB DCC is installed in a Palladium XP system, it will be downgraded to 4GB DCC.

-add {numCapturedNetsPerDomain <n>}

Specifies the number of probe channels available per domain. The number *<n>* should be a multiple of 768, but not greater than 307200. If *<n>* is not a multiple of 768, it is rounded downward to a multiple of 768. The waveform trace depth is approximately 32G/*<n>*, so specifying a larger *<n>* gives a smaller trace depth.

When compiling with `visionMode DYNP`, specify `<n>` so that sufficient probe channels are available to probe the desired signals. Each signal probed at compile time uses one probe channel; a signal probed only at run time uses an average of about two probe channels. The disadvantage of probing at compile time is that probe channels are committed to specific signals and cannot be changed at run time.

-add {numExtraCapturedNetsPerDomain <n>}

Reserves the probe channels for run time probes on the top of the probes required for FullVision. The number `<n>` should be a multiple of 768, but not greater than 307200. If `<n>` is not a multiple of 768, it is rounded downward to a multiple of 768.

Note: Specifying a larger `<n>` might give a smaller trace depth.

This option is supported only with FV mode. When compiling with `visionMode FV`, specify `<n>` so that extra probe channels are available to probe the desired signals.

-add {compilerEffort LOW | HIGH}

Specifies the compilation quality as `LOW` or `HIGH`. The default option is `HIGH`, which means that compiler produces the best quality results. If `compilerEffort` is set to `LOW`, the compilation process is faster. In this case, the compilation result is functionally same as with `compilerEffort` set to `HIGH`, however, the emulation might run slower.

The option values, `LOW` and `HIGH`, are case-insensitive.

The `compilerEffort` option provides simple control of compilation effort, with only two values (`LOW` and `HIGH`). You can exert more detailed control using the detailed-control options `optimizationEffort` and `placementEffort`. If you specify a value for either of these options, the `compilerEffort` setting (if any) is ignored.

-add {optimizationEffort LOW | HIGH}

Specifies optimization effort as `LOW` or `HIGH`. An integer value between 0 and 10 can also be specified instead of providing values `LOW` and `HIGH`. The value 0 gives lowest effort (same as `LOW`) and 10 gives highest effort (same as `HIGH`). Higher effort usually improves emulation speed, but the compile process takes longer. In some cases, high optimization effort might slightly increase the emulation capacity required by a design.

The option values, `LOW` and `HIGH`, are case-insensitive.

-add {optimizationAlgo CLASSIC | MODERN}

Determines the version of compiler libraries to be used for optimization. The default setting is MODERN. However, occasionally, you can switch the setting to the CLASSIC option for a better compilation result.

Note that this option is independent from the optimizationEffort and compileEffort options of the compilerOption command.

-add {placementEffort LOW | HIGH}

Specifies placement effort as LOW or HIGH. Higher effort usually improves emulation speed, but the compile process takes longer.

The option values, LOW and HIGH, are case-insensitive.

-add {powerdb <powerdb_directory>}

Specifies the path to the directory that stores the *powerdb.index* file, which contains information about the power consumption of the ASIC cells used by the design. The power information about each ASIC cell is obtained from the power database and a weight is assigned to each net based on the power information. This helps in collecting realistic toggle count data at run time to perform dynamic power analysis.

-add {min_dcc_percentage <n>}

This option is allowed for backward compatibility, but is ignored by the HDL Compiler.

-add {max_dcc_percentage <n>}

This option is allowed for backward compatibility, but is ignored by the HDL Compiler.

-add {limfanout <n>}

Specifies that nets with fanout (number of loads) higher than <n> should be ignored by the et3compile partitioning algorithm. The default value is 75.

If et3compile fails with either of the following two messages, setting <n> to a non-default value, such as 30, might get better results:

UXE Command Reference Manual

User Data Commands

■ INSUFFICIENT RESOURCES

This message is recorded in the `et3compile.msg` file.

■ ERROR (db2util-1103): There are not enough resources to allocate probes for FullVision.

This message is recorded in the `xe.msg` file.

Before using the `limfanout` option, try the following:

■ Execute either of the following command options:

- `compilerEffort low`
- `optimizationEffort low`
- `placementEffort low`

■ Try a different `visionMode`. Usually, DYNP has the best capacity.

■ If `et3compile` is failing in probe routing, use the `compilerOption min_dcc_percentage 100` command, or use a `visionMode` other than FV.

■ Sometimes, executing the `compilerOption infiniTrace off` command improves the capacity.

To evaluate the effect of the `limfanout` option, perform several compiles with each option setting, and observe the either of the following:

■ Percent of compiles that succeed with each setting.

■ Peak number of inputs and outputs at the chip and domain levels, and the peak nets per cable, which are printed in the `LOADA` section of the `tmp/et3pass2.*.msg` file like the following:

00 ET4LOADA Type	Peak	Avg / Peak
<hr/>		
00 ET4LOADA Outputs per ed:	ed[0, 8]	Outputs=13058/20436
00 ET4LOADA Outputs per ec:	ec[2, 66]	Outputs=2594/4861
00 ET4LOADA Inputs per ed:	ed[0, 4]	Inputs= 14413/30216
00 ET4LOADA Inputs per ec:	ec[0, 44]	Inputs= 3136/6170
00 ET4LOADA Peak Nets per Cable =	3497	

The maximum recommended values are design-dependent, but approximate values are:

- Peak inputs (or outputs) per chip: 20000
- Peak inputs (or outputs) per domain: 60000

- Peak nets per cable: 15000

-add {retiming OFF | ON}

Specifies whether or not design retiming should be enabled. Design retiming increases the design emulation speed. This technique is based on the fact that the emulation processors are almost never assigned with an even amount of work. While some processors are busy with calculations, the other processors might be idle, waiting for their inputs to become ready. The idle processors can be used to pre-calculate the signal values for the next emulation cycle, thus shortening the subsequent cycles.

With `retiming OFF`, no design retiming is attempted. The default value for the `retiming` option is `ON`.

-add {stepMode 256 | 512 | auto}

Specifies the stepMode value to be used at compile time.

Here, `auto` is the default value and lets the compiler choose the appropriate stepMode between 256 and 512. When the setting is `auto` (default), the compiler tries to determine the best stepMode. However, you can force it to set the stepMode as 256 or 512 as per your requirements.

If you specify 256 or 512, the stepMode value is set to 256 or 512, respectively.

If you set the stepMode as 256, the emulation processors run in a different mode that increases their connectivity. This can either improve or hurt step count.

Suppose the stepMode is set to `auto`, and the compiler chooses the stepMode value as 256. If the design, in fact, requires more steps, the advantage of setting the value to 256 is lost, and stepMode value of 512 is likely to result in a smaller step count.

Set the stepMode value manually in the following situations:

- The compiler automatically sets stepMode as 256, but the design compiles in more than 256 steps or the compilation fails. Then, manually setting the stepMode to 512 might result in a better FCLK.
- The compiler automatically sets stepMode as 512, and the design compiles within 256 or just slightly more steps than 256. Then, manually setting the stepMode to 256 might result in a better FCLK.

-add {enhancedTerminalTiming ON | OFF}

Specifies whether or not to use internal negative delays for improved terminal timings. The default value for `enhancedTerminalTiming` option is `ON`.

-add {symmetricPreserveCables ON | OFF}

Instructs the compiler to expand only subsets that have cables in the same offsets (and match in length) as the base configuration subset. This option is set to `ON` by default.

For details about the process of expanding the subset, refer to the *Compiling for Multiple Configurations (Symmetric Configuration)* section of the *Working with the Palladium XP Emulators* chapter in the *UXE User Guide*.

-add {symmetricPreserveMemCards ON | OFF}

Instructs the compiler to expand only subsets that have memory cards in the same offsets as the base configuration subset. This option is set to `ON` by default. If a particular subset does not match the memory cards of the base configuration subset, it is not eligible and the unusable resources associated with this subset are not removed from the compile. So, the design will not be relocated to that subset.

To generate additional subsets for relocation or to enable relocation to domains with bad memory card instances, set the `symmetricPreserveMemCards` to `OFF`.

If you try to force relocation on the boards with a bad memory card instance and have `symmetricPreserveMemCards` set to `ON`, then you might notice the following error message:

```
ERROR (legacy-43984): 20 DATABASE | 267: PDXETA0267E Resources required by the
job are not available from the current hardware configuration; see
tmp/S<SessionID>_joblog.<timestamp>.cfgmgr for details. Try
recompiling the design using the current Physical Configuration. But first
check tmp/S02979_joblog.11-13-04.cfgmgr for details.
ERROR (legacy-28872): Design cannot be configured on the current emulator
configuration. Check that the design configuration (xcva_top.et3config) is a
proper subset of the emulator configuration(/et3mach/rtp-palladium-
01.et3config).
```

For more information about processing the subsets, refer to the *Compiling for Multiple Configurations (Symmetric Configuration)* section of the *Working with the Palladium XP Emulators* chapter in the *UXE User Guide*.

-add {wireDuplication <value>}

Instructs the compiler to schedule two transmissions through different wires, instead of one, for a signal that requires a wire transmission between chips or boards. The *<value>* argument of the `wireDuplication` option can be specified as either of the following:

Value	Instructs the compiler to
C	Duplicate the cable wires.
B	Duplicate the board wires.
M	Duplicate the module wires.
P	Consider the entire pair of transmission as unusable if one of the wires is defective; otherwise, enables you to use the pair.
ALL	Duplicate the cable, board, and module wires.
OFF	Does not permit duplication.
ON	Duplicate the most reasonable set of cable and board wires.

Note: By default, the *<value>* is set to OFF. The values C, B, M, and P can be used in any combinations. For example,

- ❑ To duplicate board and module wires, specify:
`compilerOption -add {wireDuplication BM}`
- ❑ To duplicate cable, board, and module wires, specify either of the following:
`compilerOption -add {wireDuplication CBM}`
`compilerOption -add {wireDuplication ALL}`
- ❑ To duplicate the most reasonable set of cable and board wires, specify either of the following:
`compilerOption -add {wireDuplication CB}`
`compilerOption -add {wireDuplication ON}`

-add {ModelChecker OFF | ON}

Starts the ModelChecker program from the normal compile flow if set to ON. When this compiler option is set to OFF, the ModelChecker program stops to function.

The ModelChecker program helps to find the faults in the compiler model before run-time failures occur due to them.

UXE Command Reference Manual

User Data Commands

Note: The ModelChecker program can be alternatively started from the UNIX command prompt by giving the following command:

ModelChecker <design_name> [options]

If the design was compiled with the `wireDuplication` compiler option as enabled, the ModelChecker program performs an additional pass to check the integrity of the compiled model after wire switching.

-add {write_pgm OFF | ON}

Instructs the ModelChecker program to dump an ASCII view of the compiler's program on the disk to the `<design>.et5pgm` file.

-add {write_fnets OFF | ON}

Instructs the ModelChecker program to dump an ASCII mapping of post-synthesized netlist names to data store locations within the emulation processors to the `et3compile.msg` file.

-add {ModelCheckerOmit "<check_type> <check_type> ..."}

Instructs the ModelChecker program to not perform the specified types of checks. The `<check_type>` option can have the following values:

- LOGIC
- LATCH
- XBIO
- DBIO
- MEM
- ALU
- CONNECTIVITY

Note: When multiple `<check_type>` options are given, make sure they are enclosed within double quotes. Also, every time a new `ModelCheckerOmit` compiler option is specified, it overrides the values specified by the previous `ModelCheckerOmit` compiler option specifications.

For detailed information about the check types, refer to the *Finding Faults using ModelChecker* section of the *Working with the Palladium XP Emulators* chapter in *UXE User Guide*.

-add {cpfLogBufferDepth <n>}

Specifies the depth of memory buffer to capture CPF events. The specified depth is rounded off to a number of power of 2. The default depth is 16K. For the <n> parameter, you can specify only an integer, or an integer followed by M or K for megabyte or kilobyte, respectively. The M and K are case-insensitive.

Note:

- If you set a very high value for the depth, the memory buffer will use too many hardware resources and cause the compilation to fail. The compiler prints out a message to show the size of memory buffer and specify the reason in case the compilation failed.
- If you specify the depth as 0, the feature of capturing the CPF events is disabled.

-add {1801LogBufferDepth <n>}

Specifies the depth of the memory buffer to capture 1801 events.

The specified depth is rounded up to a number of power of 2. You can specify an integer, or an integer followed by M or K. The M and K are case insensitive. The default depth is 16K.

If you specify a depth which is too big, then the memory buffer will use too many hardware resource and cause compilation to fail. In case the compilation fails, you can look at the displayed message that shows the size of the memory buffer.

-add {1801LogType { [powerDomain] [supplySet] [isolation] [retention] [powerState] [stateTransition] [portState] [pstState] [illegalIsoState] [illegalRetState] } }

Specifies a set of 1801 event types that should be captured in the memory buffer.

By default all event types are captured.

-add {1801CoverageWidth [0-8]}

Specifies the width of the coverage counters to be used to generate the coverage report for 1801 low power objects. The default width is 8 bits. You can decrease the bits to save the hardware capacity.

-add {XP II capacityFeatures <N>}

Disables or enables certain emulator capacity enhancements targeted towards Palladium XP II. This might be necessary because these enhancements might cause some Palladium XP scripts to fail on Palladium XP II. This option helps in isolating and investigating such failures, and ensures that the netlist that is generated is compatible with Palladium XP.

Here, <N> can be any value $0 \leq N \leq 7$.

According to the value specified for <N>, the Palladium XP II capacity-enhancement features, namely partitioner enhancements, common sub-expressions, and combining flip-flops are enabled or disabled.

The feature is enabled if the corresponding bit is 1. If the bit is 0, the feature is disabled.

- bit 0 (value 1) controls partitioner enhancements
- bit 1 (value 2) controls common sub-expressions
- bit 2 (value 4) controls combining flip-flops.
- The default is 7 (all features are enabled).

-add {symmetricXP II only ON}

Ensures that if the hardware includes a mix of Palladium XP and Palladium XP II boards, only the Palladium XP II boards are included in the symmetric compilation, and the resultant compilation database is a Palladium XP II database, which can be run only on Palladium XP II boards. Note that if the base or symmetric configurations contain a mix of Palladium XP and Palladium XP II boards, by default, the compilation database is generated for Palladium XP.

If the `symmetricXP II only` option is used, and no Palladium XP II boards are found that meet the base configuration criteria, compilation fails.

-add {maxStep <number>}

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* argument must be a multiple of 2. The range for maxStep is 1-4096.

Note: The `maxStep` is a User Data option for the `compile` and `compileFind` commands, whereas, the `-max_steps` is a command-line option for the `compile` and `compileFind` commands. If both `maxStep` and `-max_steps` are present, then `-max_steps` is given priority.

-add {minStep <number>}

The `minStep` option is usually a complement to the `maxStep` option. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both `minStep` and `maxStep` are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified with the `minStep` option should not be greater than the number specified with the `maxStep` option.

The *<number>* specified in `minStep` must be a multiple of 2. The range for `minStep` is 1-4096.

Note: The `minStep` is a User Data option for the `compile` and `compileFind` commands, whereas, the `-min_steps` is a command-line option for the `compile` and `compileFind` commands. If both `minStep` and `-min_steps` are present, `-min_steps` is given priority.

-add {DPATopNInst N}

Adds top *N* pre-selected biggest instances of the whole design. The instances are sorted based on the number of nets in each instance. The default value of *N* is 10000.

The `compilerOption -add {DPATopNInst N}` command affects the `dpa -setupToggleCount -addinst` command, which can add and count into `ppfdata` file only the instances among the pre-selected top *N* instances. When you want to change this parameter, you must rerun `dbgPrepare` to make the change effective.

-add {subDPATopNInst <fileName>}

Adds top *N* instances under a specified instance, which might not be the root instance, into the pre-selected instances list.

For example,

```
compilerOption -add {subDPATopNInst ./subN}
```

If the "subN" file includes the following:

```
100 dut.cnt_u0  
150 dut.cnt_u1
```

Then, `dbgPrepare` will try to add top 100 instances under `dut.cnt_u0` and top 150 instances under `dut.cnt_u1` into the pre-selected instances list.

This option does not conflict with the `-add {DPATopNInst N}` option. When you want to change this parameter, you must rerun `dbgPrepare` to make the change effective.

-add {dumpDPATopNInst N}

Saves the list of pre-selected top `N` instance names with the number of nets in each instance. The dumped data is saved as `./dbFiles/topNInst.dump`. The `compilerOption -add {dumpDPATopNInst 0}` command will save all pre-selected instances.

The instances are saved as follows (sorting by the number of nets):

```
topNInst(1): synNetNum=3507, instName=.  
topNInst(2): synNetNum=3284, instName=dut  
topNInst(3): synNetNum=638, instName=dut.cnt_u3  
topNInst(4): synNetNum=638, instName=dut.cnt_u2  
topNInst(5): synNetNum=638, instName=dut.cnt_u1  
...  
...
```

-rm {<compiler_option>}...

Removes the values set using the specified compiler option.

For example,

- Specifying the following command removes the types of checks specified for omission using the `-add {ModelCheckerOmit "<check_type> <check_type> ..."}` compiler option.

```
compilerOption -rm ModelCheckerOmit
```

- Specifying the following command reverts the retiming settings to the default value, which is ON:

```
compilerOption -rm retiming
```

Using Multisampling

This is an expert-user feature, and should only be used by those who have expertise with the design in question, including how the design is compiled and evaluated on the emulator.

Multisampling effects a speedup by mechanisms other than a simple increase in frequency of evaluation. Instead, portions of the design are modified such that up to twice as much evaluation is compressed into one cycle. This modification can take as much as two times the capacity that the unmodified circuitry would take.

For more information, refer to the *Compiling Designs for In-Circuit Emulation with xeCompile* chapter of the *UXE User Guide*.

create_assertion_control

Enables you to specify alternate assertion behavior when domains are powered down in an IEEE 1801 environment in the IXCOM mode.

The syntax for this command is:

```
create_assertion_control -name <string>
    {-assertions <assertion_list> | -domains <power_domain_list>}
    [-assertion_control expression | -supply_set]
    [-type reset]
```

-name <string>

Specifies the name of the assertion control.

-assertions <assertion_list>

Selects only the assertions whose names are included in the assertion list.

-domains <power_domain_list>

Selects all assertions that appear in any hierarchical instance associated with one of the specified power domains.

-assertion_control expression

Specifies the condition when the selected assertions should be powered down.

-supply_set

The `-supply_set` option will cause the selected assertions to be powered down when the supply set is in a corrupted state.

-type reset

Clears all of the state information and evaluates the assertions from the reset state. This is the only currently supported behavior when `create_assertion_control` is used. The behavior is identical for both PSL and SVA assertions.

customIsolationCell

Specifies that loopbreaking should propagate a (clk) marking, if present, from D to Q of any latch inside a cell specified by the user. This command might be useful when a latch is used to isolate a clock from its downstream logic. By default, (clk) markings are propagated only through combinational gates.

The syntax for this command is:

```
customIsolationCell -add {<libraryName> <cellName>}  
customIsolationCell -rm {<libraryName> <cellName>}
```

-add {<libraryName> <cellName>}

Specifies that for any latch, which occurs as a leaf-level instance within an instance of the cell specified by <libraryName> <cellName>, loopbreaking should propagate a (clk) marking, if present, from D to Q of the latch.

In both <libraryName> and <cellName>, the wildcard characters ? and * can be used. The ? represents any single character, and * represents any sequence of zero or more characters. <cellName> must be the name of a user-defined cell (not a qtref primitive).

-rm {<libraryName> <cellName>}

Removes the cell specified by <libraryName> <cellName> from the set of custom isolation cells.

delayAsynClockInputs

Schedules the specified clock inputs later than the non-clock inputs. This command should be used for designs that have one or more clock inputs coming from outside the emulator, which are asynchronous with FCLK, when there are also data inputs coming from outside the emulator that change in response to these clocks. In such cases, these clock inputs must be scheduled later than the non-clock inputs.

The syntax for this command is:

```
delayAsynClockInputs -add {<sig1> <sig2> ... [<N>]}  
delayAsynClockInputs -rm {<sig1> <sig2>...}
```

-add {<sig1> <sig2> ... [<N>]}

Schedules the clock inputs with respect to the non-clock inputs. *<sig1> <sig2>...* represents the list of all clock inputs from the target to the Palladium XP system. *<N>* is the minimum number of steps later that these clocks should be sampled, relative to the latest non-clock input (including input components of bidirectional signals) from the target.

-rm {<sig1> <sig2>...}

Removes the specified clock signals from the existing delay specification.

Usage

```
delayAsynClockInputs
```

Lists the signals specified in the previous `delayAsynClockInputs` statement.

```
delayAsynClockInputs -add {<sig1> <sig2> ... <N>}
```

Schedules the clock signals *<sig1>* and *<sig2> ...* at least *<N>* steps after the last scheduled non-clock input signals.

```
delayAsynClockInputs -add {<sig3> <sig4> ...}
```

Schedules the clock signals *<sig3>* and *<sig4> ...* with the same delay as the clock signals defined in the previous `delayAsynClockInputs`.

```
delayAsynClockInputs -rm {<sig5> <sig6> ...}
```

Removes the clock signals *<sig5>* and *<sig6>* from the previously specified `delayAsynClockInputs` signals.

UXE Command Reference Manual

User Data Commands

Examples

```
delayAsynClockInputs -add {ck1 ck2 20}
```

Schedules both `ck1` and `ck2` at least 20 steps after the scheduling of the last non-clock input.

delayBox

Defines a delay box (used in connection with [terminalTiming](#)) with the output signal named `<output_name>`, delay given by `<delay>`, and the specified inputs. If the delay box already exists, the command modifies the delay, and will add inputs from the specified inputs list to the existing list of inputs (that is, it will not replace the existing inputs list).

Consider the `delayBox` output signal `<output_name>` to be an imaginary signal that the compiler will schedule at least `delay` steps (or TIFs) after the latest step at which any of the `delayBox` inputs are scheduled.

Run this command before [icePrepare](#).

The syntax for this command is:

```
delayBox -add {<output_name> <delay> <list_of_input_names>} ...
delayBox -dump <filename>
delayBox -rm {<output_name>} ...
delayBox <output_name>
```

-add {<output_name> <delay> <list_of_input_names>}

Adds a delay box with the specified `<output_name>`, `<delay>`, and `<list_of_input_names>`. Multiple delay boxes can be added using a single `-add` statement.

<output_name>

Specifies the output signal if given with the `-add` option.

Specifying the `<output_name>` option directly with the `delayBox` command (that is, `delayBox <output_name>`), returns the definition of the specified delay box.

<delay>

Specifies delay in terms of emulation steps or TIFs. This must be a positive integer number. If the `compilerOption ioDelayUnit` has not been specified, the delay value is interpreted in steps.

<list_of_input_names>

Adds inputs to the existing list. An input is either a terminal name or a delay box name. You can also specify a delay box input in the form:

- *<terminal-name>*: xeInput
- *<terminal-name>*: xeOutput
- *<terminal-name>*: xeEnable

The *<terminal-name>* must correspond to a bidirectional or tristate terminal.

-dump <filename>

Saves the specified `delayBox` commands into the specified file. If the file is not specified, the `delayBox` commands will be printed on `stdout` (that is, the screen).

-rm {<output_name>}

Removes the delay box with the matching output signal, that is, the specified *<output_name>*. Multiple delay boxes can be removed using a single `-rm` statement.

Special delayBoxes for multisampled I/O

For designs using multisampled I/O, special `delayBox` output names are recognized.

If a `delayBox` *<output_name>* is `START_SHADOWS`, the *<list_of_input_names>* must all be `delayBox` output names. If you define a `START_SHADOWS` `delayBox`, the shadow signals of any multisampled terminals will be scheduled at or after the step when `START_SHADOWS` is scheduled.

If a `delayBox` *<output_name>* is `EMUshadow@<net>` where *<net>* is a multisampled terminal, the `delayBox` is handled in the following special way: regardless of the `terminalTiming` commands, the shadow of *<net>* will automatically be constrained to be scheduled no earlier than the `delayBox` `EMUshadow@<net>`.

Examples

```
delayBox -add {C_DEL 10 TI D_DEL} {D2 20 T3}...
```

Adds the definitions of the specified delay boxes.

UXE Command Reference Manual

User Data Commands

```
delayBox {C_DEL}
```

Returns the definition of the specified delay box.

```
delayBox -add {delayBox1 20 A}  
terminalTiming -add {B delayBox1}
```

If A and B are top-level input or output signals of the design, these commands constrain signal B to be scheduled at least 20 steps later than signal A.

```
delayBox -add {delayBox1 20 D[0]:xeOutput D[0]:xeEnable}  
terminalTiming -add {D[0] * delayBox1 *}
```

If D[0] is a bidirectional top-level I/O of the design, these commands constrain the input component of D[0] to be scheduled at least 20 steps later than the latest of the output and enable components.

```
delayBox -add {longafterclock 32 clk}  
delayBox -add {START_SHADOWS 0 longafterclock}
```

In a design where the only multisampled terminal is a clock output, the compiler will usually by default schedule the shadow as soon as possible after the non-shadow. This results in a narrow clock pulse which might be missed by some target systems. The above two commands ensure that the shadow signal of clk is scheduled at least 32 steps after the non-shadow signal.

delayBoxInput

Specifies input signals to a delay box. The command adds inputs to or removes inputs from the existing input list. (It does not replace the existing input list).

Note: The specification of inputs can be performed entirely with the `delayBox` command, making this command optional. The `delayBoxInput` command is most useful for removing specific inputs. For example, assume that a delay box already exists with multiple inputs. However, you want to remove just a few of the inputs without deleting the entire delay box and creating it again from the beginning. Then, you can specify the inputs in multiple commands, with no need to use a single command.

Run this command before [icePrepare](#).

It can be issued before or after the `delayBox` command. If it is specified before the `delayBox` command, a `delayBox` command is still required for a complete specification of the box.

The syntax for this command is:

```
delayBoxInput -add {<output_name> <delay> <list_of_input_names>} ...
delayBoxInput -rm <output_name> [<list_of_input_names>] ...
```

-add {<output_name> <delay> <list_of_input_names>}

Adds inputs to a delay box input list. An input can be either a terminal or the output of another delay box. The required set of parameters include `<output_name>`, `<delay>`, and `<list_of_input_names>`. Multiple delay boxes can be added using a single `-add` statement.

<output_name>

Specifies the name of the delay box output.

<delay>

Specifies delay in terms of emulation steps or TIFs. This must be a positive integer number. If the `compilerOption ioDelayUnit` has not been specified, the delay value is interpreted in steps.

<list_of_input_names>

Adds inputs to the existing list.

-rm <output_name> [<list_of_input_names>]

Removes inputs from a delay box input list.

Examples

```
delayBoxInput -add {after_clocks 20 clk1 clk2 clk3}
```

Adds inputs to the specified delay box(es).

```
delayBoxInput -rm {after_clocks}...
```

Clears the input list of the specified delay box(es).

```
delayBoxInput -rm {after_clocks clk1 clk2}
```

Clears specific inputs from the inputs list of the specified delay box.

emptyCell

Enables `precompile` to handle empty cells in a design, without causing an error. Empty cells are the cells that contain neither sub-cell instances nor feedthrough connections. By default, the presence of empty cells in the design causes `precompile` to fail.

This command provides the same functionality as provided by the `keepEmptyCells` option of the `precompileOption` command. All the instances of the empty cell and their terminals are kept in the design database, so that any command (such as `clockSource`) on these terminals can be honored.

The syntax for this command is:

```
emptyCell -add {<libraryname> <cellname>}  
emptyCell -rm {<libraryname> <cellname>}
```

-add {<libraryname> <cellname>}

Enables the specified the empty cell, `<cellname>`, that belongs to the specified library, `<libraryname>`, to remain in the design without causing an error during `precompile`.

-rm {<libraryname> <cellname>}

Removes from the design the empty cell of the specified library.

emulatorConfiguration

Defines the hardware resources that will be used during the design compilation process. If used along with the [symmetricConfiguration](#) command, it defines a base configuration subset for symmetric compilation feature described in the *Compiling for Multiple Configurations (Symmetric Configuration)* section of the *Working with the Palladium XP Emulators* chapter in *UXE User Guide*.

Execute the `emulatorConfiguration` command after the [design](#) command.

Also, note that this command must be executed before the [precompile](#) command. Otherwise, the new emulator configuration will not be recognized during the compilation process and might cause inconsistency during the run time. If the `emulatorConfiguration` command is executed after the `precompile` command, it is necessary to re-run the `precompile` command.

This command generates a configuration file named `<top_cell>.et3config` in the `./dbFiles` directory, where the `<top_cell>` is the top cell name specified in the `design` command. The `<top_cell>.et3config` configuration file is used during the back-end compilation and run-time debug session.

Without parameters, this command lists the current hardware configuration, such as the hostname, filename, and board information. To specify a new emulator configuration re-invoke the `emulatorConfiguration` command with new parameter values; the new parameter values override the old parameter values.

The syntax for this command is:

```
emulatorConfiguration -add {host <hostname>}  
emulatorConfiguration -add {host <hostname>} {boards <sub_board>}  
emulatorConfiguration -add {file <filename>}  
emulatorConfiguration -add {file <filename>} {boards <sub_board>}  
emulatorConfiguration -add {boards <sub_board>}  
emulatorConfiguration -help
```

-add {host <hostname>}

Adds the specified hardware configuration information to the design database. The `{host <hostname>}` argument specifies the network name of the emulator's host workstation.

Note: A dot (.) can be used instead of `<hostname>` to specify the current machine as the emulation host.

The compiler locates the emulator configuration file (*<hostname>.et3config*) using the following steps:

1. The emulator configuration file is first searched on the local host at the following path:

et3mach/<hostname>.et3config

2. If the configuration file does not exist or is not readable on the local host, the network host is searched at the following path:

/net/<hostname>/et3mach/<hostname>.et3config

3. If the configuration file does not exist or is not readable on the network host, the remote network host is searched at the following path:

<hostname>:/et3mach/<hostname>.et3config

Note: Setting the `XE_SECURE_CONNECTION` environment variable before the software startup helps to establish a secure connection with the remote network host.

4. Lastly, if the configuration file does not exist or is not readable on the remote network host, the current working directory is searched at the following path:

./<hostname>.et3config

-add {host <hostname>} {boards <sub_board>}

Specifies the subset *<sub_board>* of the emulator host machine that the design will use. The `boards` option selects the specific subset of boards/domains from the entire emulator host machine, specified as a number or a string.

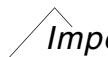
The *<sub_board>* is specified in the following string format with no space between the characters:

<boardNum>[.<domainNum>]+...<boardNum>[.<domainNum>]

Here,

- *<boardNum>* indicates a number in the range of 0 to *n-1*, where “*n*” is the maximum number of boards in the host or file. The maximum number of boards differ based on the type of host as following:
 - A Palladium XP XL system has maximum 3 boards that are numbered from 0 to 2.
 - A Palladium XP GXL system has maximum 20 boards that are numbered from 0 to 19.
- *<domainNum>* indicates a number in the range of 0 to *n-1*, where “*n*” is the maximum number of domains on a board. The maximum number of domains differ based on the type of host as following:

- A Palladium XP XL system has maximum 4 boards that are numbered from 0 to 3.
- A Palladium XP GXL system has maximum 16 boards that are numbered from 0 to 15.

 **Important**

For board number strings, each number must increase from left to right, such as (0.1+2.0+5.3+6). Strings such as 4+5.1+1.3 are not allowed. For the same board, the domain numbers should also be placed in an increasing order, such as (0.0+0.3+0.5).

The boards option does not apply to the virtual configuration file. This is because virtual configuration supports the entire model but not a subset of the model.

-add {file <filename>}

Specifies the file that contains the emulator's hardware configuration. The design must read the specified file for configuration details. The *<filename>* file must be a valid emulation configuration file. Else, the software will exit with an error message.

-add {file <filename>} {boards <sub_board>}

Specifies the subset *<sub_board>* of the configuration specified in the *<filename>* emulation configuration file that the design will use.

The boards option does not apply to the virtual configuration file because virtual configuration supports the entire model, not a subset of the model.

-add {boards <sub_board>}

Specifies the subset *<sub_board>* of the configuration that the design will use. This option can be used either after the `emulatorConfiguration -add {host <hostname>}` or `emulatorConfiguration -add {file <filename>}` command.

-help

Returns the command syntax.

Examples

- To use the hardware configuration of the host named `ostrich`, specify:
`emulatorConfiguration -add {host ostrich}`
- To use the configuration file that is available within the `/net/ostrich/sand/et3mach` directory on the host `ostrich`, specify:
`emulatorConfiguration -add {host ostrich /net/ostrich/sand/et3mach}`
- To use the hardware configuration specified in the emulator configuration file named `foo.et3cfg`, which is available in the current working directory, specify:
`emulatorConfiguration -add {file ./foo.et3cfg}`
- To compile the design using board domain 0 on board 1 and domain 2 on board 1 of the emulator host named `cuckoo`, specify:
`emulatorConfiguration -add {host cuckoo} {boards 0.0+1.2}`

globalNet

Connects a signal to the top-level terminal as a global net.

The syntax for this command is:

`globalNet -add {<net> <term>}`

`globalNet -rm {<net>}`

`globalNet -clear`

-add {<net> <term>}

Adds a global net specified by the hierarchical net name *<net>* and top-level terminal *<term>*. The *<term>* must be unique, and must not be an existing top-level terminal.

-rm {<net>}

Removes the specified global net.

-clear

Removes all entries created with the `globalNet -add` command.

hierAssign

Inserts buffers to separate part of a net at a specified hierarchical location from the rest of the net. Then, at the specified hierarchical location, all drivers of the net are removed, and the nets are connected either to power or ground, or to another net.

For information on how the buffers are inserted, refer to the *Connecting Hierarchical Nets* section in the *Compiling Designs for In-Circuit Emulation with xeCompile* chapter of the *UXE User Guide*.

The syntax for this command is:

```
hierAssign -add {<net> <value>} ...  
hierAssign -add {<net> <source_net>} ...  
hierAssign -rm <net> ...
```

-add {<net> <value>} ...

Ties the specified nets to either power or ground. The *<value>* must be either 0 or 1. If the value is 0, the selected net is tied to ground. If the value is 1, the selected net is tied to VCC (power). This command is similar to the `tieNet` command, however, the `hierAssign -add {<net> <value>}` command inserts the buffers and due to the inserted buffers, only part of the original net is affected.

-add {<net> <source_net>} ...

Connects the nets to the specified source nets. This command is similar to the `clockAssign` command, however, the `hierAssign -add {<net> <source_net>}` command inserts the buffers and due to the inserted buffers, only part of the original net is affected.

-rm <net> ...

Removes the settings done through the previous `hierAssign` commands.

hwInfo

Returns information related to all the boards, domains, memories, connector, and Orion cards on the emulator.

Note: To display the correct emulator configuration, use the `emulatorConfiguration -add {host <hostname>}` command prior to the `hwInfo` command.

Without any parameters, the `hwInfo` command returns the hardware configuration of the emulator that is connected to the host machine. If the `emulatorConfiguration -add {host <hostname>}` command has not been executed before the `hwInfo` command, executing the `hwInfo` command without any parameters will return the value `{unknown}`.

The syntax for this command is:

```
hwInfo -host <hostname>
hwInfo -file <filename>
hwInfo -design
hwInfo -orion
hwInfo -buffer
hwInfo -cable
hwInfo -connector
hwInfo -memory
hwInfo -help
```

-host <hostname>

Returns the configuration of the emulator that is connected to the specified host machine. This information includes the modules in each board, HDDC connections in each module, Orion card information, ICTI information, and IP-ID number (if applicable).

-file <filename>

Returns the information from the specified configuration file. This information includes the modules in each board, HDDC connections in each module, Orion card information, and IP-ID number (if applicable).

-design

Returns the configuration of the emulator on which the current design is being run. The `-design` option must be used after successfully running the `emulatorConfiguration` command. Else, an error message is returned.

-orion

Returns the detailed Orion card information.

-buffer

Returns the information related to the buffer cage assembly, buffer card, and HDDC cable connections.

-cable

Returns the board-to-board cable information.

-connector

Returns information related to all the board-to-buffer card connections and board-to-board connections based on the connectors in each board.

-memory

Returns the external memory card information.

-help

Returns a summary of the command usage.

Examples

Example 1

To know the configuration information of emulator connected to the host machine named *cuckoo*, use the following command:

hwInfo -host cuckoo

Following information is returned:

```
Emulator type: UA DESIGN
Total 2 boards
Board 0: physical ID 0 (3 domains)
```

Domain	board_slotNo	HDDC_conn	IPID
0	0_1	J1	J2
2			
3			

```
Board 1: physical ID 1 (3 domains)
```

Domain	board_slotNo	HDDC_conn	IPID
0			
1	1_1	J1	J2
2			

```
{HWType UA_design_system} {BoardNum 2} {DomainNum 6} {CableNum 3}
{MemoryNum 2} {OrionCardNum 0}
```

Example 2

To know the board-to-board cable connection information of the host machine named *cuckoo*, use the following command:

hwInfo -host cuckoo -cable

In addition to the emulator configuration (as illustrated in [Example 1](#)), the following information is returned:

UXE Command Reference Manual

User Data Commands

```
Detailed Cable connection information (total 3 cables):
```

```
Boards 0:
```

board1	module1	connector1	board2	module2	connector1	cableType
0	0	J1	1	0	J1	CABLE_154
0	0	J3	1	1	J3	CABLE_106
0	2	J1	1	2	J1	CABLE_154

```
Board 0 connects to 1 boards by 3 cables.
```

```
    Board 0 connects to Board 1 by 3 cables.
```

```
Boards 1: physical ID 1 (3 domains)
```

board1	module1	connector1	board2	module2	connector1	cableType
1	0	J1	0	0	J1	CABLE_154
1	1	J3	0	0	J3	CABLE_106
1	2	J1	0	2	J1	CABLE_154

```
Board 1 connects to 1 boards by 3 cables.
```

```
    Board 1 connects to Board 0 by 3 cables.
```

```
{HWType UA_design_system} {BoardNum 2} {DomainNum 6} {CableNum 3}  
{MemoryNum 2} {OrionCardNum 0}
```

Example 3

To display the orion card information of the host machine named *cuckoo*, use the following command:

```
hwInfo -host cuckoo -orion
```

In addition to the emulator configuration (as illustrated in [Example 1](#)), the following information is returned:

```
Detailed Orion card information:
```

```
OrionCardHost Board      nccPortNo  OrionDeviceNo  OrionPortNo
```

UXE Command Reference Manual

User Data Commands

```
{HWType UA_design_system} {BoardNum 2} {DomainNum 8} {CableNum 4}  
{MemoryNum 2} {OrionCardNum 0}
```

Example 4

To display the external memory card information for the current design, use the following command:

```
hwInfo -design -memory
```

In addition to the emulator configuration (as illustrated in [Example 1](#)), the following information is returned:

```
Detailed external memory information <total 2 external memory cards>:
```

Board	Module	ChipOffset
0	0	0
0	0	3
0	0	6
0	0	9
0	0	1
0	0	4
0	0	7
0	0	10

```
{HWType UA_enterprise_system} {BoardNum 1} {DomainNum 1} {CableNum 0}  
{MemoryNum 2} {OrionCardNum 0}
```

identify_power_domain

Identifies the power domain information in the netlist and specifies the power control net in the power shutoff condition.

The syntax for this command is:

```
identify_power_domain {-name <powerDomainName>} {-modules <moduleNameList> ...} \
    [-shutoff condition <expression>]
    [-clear]
```

-name <powerDomainName>

Specifies the power domain name, which can be a string comprising of letters, digits, and underscore.

-modules <moduleNameList> ...

Specifies a list of module names.

Note: Wildcards are not allowed. Also, library name is not required because the module is assumed to be in the library used by Palladium XP.

-shutoff_condition <expression>

Specifies either the power control net name or the inverter of the power control net. The net names used with the -shutoff_condition option are the signals connected to the V_{DD} port of the instances.

-clear

Removes the identified power domains.

instanceFV

Enables or disables specific instances for selective FullVision (FV).

The syntax for this command is:

```
instanceFV -add [<instance_name> enable | disable]  
instanceFV -rm [<instance_name>]
```

-add [<instance_name> enable | disable]

Adds an instance for selective FV. By default, FV is enabled on the entire design if nothing is specified with the `-add [<instance_name> enable | disable]` option.

-rm [<instance_name>]

Removes the specified instance from the `instanceFV` list.

Example

Consider the following two cases:

Case A:

```
instanceFV -add "dut enable"
```

Case B:

```
instanceFV -add "dut enable"  
instanceFV -add "wrap disable"
```

Note that cases A and B return different results.

In case A, the precompiler returns the following information message:

INFO (multclk-1379): Hierarchical nodes, which are not marked via `instanceFV` or `moduleFV` commands, **will not be subject to full vision**.

While in case B, the precompiler returns the following information message:

INFO (multclk-1379): Hierarchical nodes, which are not marked via `instanceFV` or `moduleFV` commands, **will be subject to full vision**.

The information messages returned in both cases reflect whether or not the hierarchical nodes will be subject to FullVision.

UXE Command Reference Manual

User Data Commands

In case A, the user wants to specify one or more instances for debugging by issuing the `enable` command assuming that the rest of the design will not be instrumented. However, when the user applies the `disable` command, it is assumed that the unspecified instances must be instrumented because otherwise it will be a simple DYNP case.

Therefore, when the user specifies both `enable` and `disable` commands, as illustrated in case B, it is assumed that the unspecified instances must be instrumented.

instanceStub

This command specifies the full hierarchical names of instances which are to be deleted in a module.

Note: Using the `-keepStubInputs` and `-keepStubOutputs` options of the `precompileOption` command causes the inputs and outputs of the stubbed instances to be treated as `keepNets`.

The syntax for this command is:

```
instanceStub -add {a.b.c.d.}
```

-add {a.b.c.d.}

Deletes the modules by specifying the hierarchical path to specific instances. Here, `a.b.c.d.` refers to the instance name. `*` and `?` wildcards can be used to specify patterns in the instance names.

keepLoadlessInstance

Use the `keepLoadlessInstance` command to prevent an instance from being optimized away by `precompileOption -add removeLoadless` command.

The syntax for this command is:

`keepLoadlessInstance -add {a.b.c.d}`

`keepLoadlessInstance -rm {a.b.c.d}`

-add {a.b.c.d}

Adds the instance `a.b.c.d` to the `keepLoadlessInstance` list. An asterisk (*) and ? wildcards can be used to specify patterns in the instance names.

-rm {a.b.c.d}

Removes the instance `a.b.c.d` from the `keepLoadlessInstance` list.

keepLoadlessModule

Use the `keepLoadlessModule` command to prevent all instances of a specified module from being optimized away by `precompileOption -add removeLoadless` command.

The syntax for this command is:

`keepLoadlessModule -add {<LIB> <cellName>}`

`keepLoadlessModule -rm {<LIB> <cellName>}`

-add {<LIB> <cellName>}

Adds the specified module, `<cellName>`, in library `<LIB>`, to the `keepLoadlessModule` list. An asterisk (*) and ? wildcards can be used to specify patterns in `<cellName>` and `<LIB>`.

-rm {<LIB> <cellName>}

Removes the specified module, `<cellName>`, in library `<LIB>`, from the `keepLoadlessModule` list.

keepLoadlessNet

Use the `keepLoadlessNet` command to ensure visibility of a specified net which would otherwise be removed by `precompileOption -add removeLoadless` command. Specifying a net as `keepLoadlessNet` is less expensive than specifying the net as `keepNet`.

The syntax for this command is:

`keepLoadlessNet -add <net>...`

`keepLoadlessNet -rm <net>...`

-add <net>...

Adds the net specified by `<net>` to the `keepLoadlessNet` list. An asterisk (*) can be used as a wildcard character in the last component of a hierarchical name.

-rm <net>...

Removes the net specified by `<net>` from the `keepLoadlessNet` list.

keepNet

Use the keepNet command to guarantee that a design signal can be set, forced, and released at run time.

Note: Primary I/O signals are automatically considered as keepNets, even if you do not specify them. Adding numerous keep nets might increase the amount of hardware required to model the design which can slow down emulation speed.

Before using this command, execute design to specify the design library and cell name for your design.

This is an optional command. Run it at your discretion after design and before precompile.

Without parameters, lists all nets you specified as keep nets.

The syntax for this command is:

```
keepNet -add <net>...
keepNet -add [-errorok] {net1} {net2} ... {netN}
keepNet -rm <net>...
keepNet -addReport <net_name>
```

-add <net>...

Adds the net specified by *<net>* to the keep net list. An asterisk (*) can be used as a wild card character in the last component of a hierarchical name.

-add [-errorok] {net1} {net2} ... {netN}

Specifies the list of keep nets that are reported if not found. The rest of the nets apart from the missing nets, are still added as keep nets. If the *-errorok* option is not specified and if any of the specified nets do not exist, the `keepNet -add` command fails, no nets are added, and a warning message is issued about all the non-existing nets. If the *-errorok* option is specified, the `keepNet -add` command adds all the existing nets, issues a warning message about the non-existent nets, but it does not fail.

-rm <net>...

Removes the net specified by *<net>* from the keep net list.

UXE Command Reference Manual

User Data Commands

Use the command `eval keepNet -rm [<keepNet>]` to easily remove all designated preserved nets.

-addNreport <net_name>

Adds non-duplicated qualified nets to the user data and returns the names of these nets.

`<net_name>` can be any net, but only non-duplicated qualified nets will be added and returned.

-addNreport <net_name>

Adds non-duplicated qualified nets to the user data and returns the names of these nets.

`<net_name>` can be any net, but only non-duplicated qualified nets will be added and returned.

Examples

```
keepNet -add my_net1 my_net2 my_net3
```

Adds `my_net1`, `my_net2`, and `my_net3` to the keep net list.

```
keepNet -rm my_net2 my_net3
```

Removes `my_net2` and `my_net3` from the keep net list.

```
keepNet -add u1.u2.a*
```

Within the `u1.u2` instance, adds all the nets having names that start with `a`.

lp (Low-power Compile-time Command)

Note: The `lp` command discussed below is used at the compile time. The [lp \(Low-power Run-time Command\)](#) run-time command is discussed in [Chapter 13, “Run-time Commands.”](#)

Displays the information added for the specified `lp` command in the CPF or IEEE 1801 file.

The syntax for this command is:

The following command options are for CPF:

```
lp [assert_illegal_domain_configurations | create_isolation_rule | create_mode | create_mode_transition |  
create_nominal_condition | create_power_domain | create_power_mode | create_state_retention_rule |  
set_cpf_version | set_design | set_hierarchy_separator | set_instance | set_macro_model]
```

The following command options are for IEEE 1801:

```
lp -check <1801_command> [<namePattern>] |
```

The `<1801_command>` can be one of the following IEEE 1801 commands:

```
{add_port_state | add_pst_state | associate_supply_set | connect_logic_net |  
connect_supply_net | connect_supply_set | create_hdl2upf_vct |  
create_logic_net | create_logic_port | create_power_domain |  
create_power_switch | create_pst | create_supply_net | create_supply_port |  
create_supply_set | create_upf2hdl_vct | set_design_attributes |  
set_design_top | set_domain_supply_net | set_isolation | set_isolation_control |  
set_partial_on_translation | set_port_attributes | set_retention |  
set_retention_control | set_simstate_behavior | set_related_supply_net |  
upf_version}
```

```
lp -check 1801 [<objectType>] |
```

For IEEE 1801, the `<objectType>` can be one of the following:

```
{domains | logicPorts | logicNets | supplyPorts | supplyNets | supplySets |  
switches | rootSupplies | equivalents | hdl2upf | upf2hdl | isolations |  
retentions | supplyPortStates | pst | pstStates}
```

```
lp -check 1801 statistic |
```

```
lp -check files |
```

```
lp -check find_objects {scope <scope> -pattern <search_pattern>  
[-object_type [model | inst | port | net]] [-direction [in | out | inout]]  
[-transitive [TRUE | FALSE]] [-regexp | -exact {}] [-ignore_case {}]  
[-non_leaf | -leaf_only {}]}
```

```
lp -check ls1801
```

Options of Compile-Time Ip Command for CPF

[assert_illegal_domain_configurations | create_isolation_rule | create_mode | create_mode_transition | create_nominal_condition | create_power_domain | create_power_mode | create_state_retention_rule | set_cpf_version | set_design | set_hierarchy_separator | set_instance | set_macro_model]

Specifies the `lp` command for which the information needs to be displayed. The following `lp` commands are supported as valid values for this option:

assert_illegal_domain_configurations	Displays the configuration of domain conditions and power mode control group conditions that are illegal.
create_isolation_rule	Displays the top DUT- and block-level isolation rules.
create_mode	Displays the top DUT modes.
create_mode_transition	Displays the top DUT mode transitions.
create_nominal_condition	Displays the top DUT-level nominal condition sets. Block-level nominal conditions are not displayed.
create_power_domain	Displays the top DUT- and block-level power domains, and the mapped power domains.
create_power_mode	Displays the top DUT-level power modes. Block-level nominal conditions are not displayed.
create_state_retention_rule	Displays the top DUT- and block-level state retention rules.
set_cpf_version	Displays the CPF version.
set_design	Displays the top DUT-level CPF design model and all of the design model names under it.
set_hierarchy_separator	Displays the hierarchy separators for the CPF commands.
set_instance	Displays all the instances that have macro or design model being applied to.
set_macro_model	Displays all the macro model names under top DUT-level CPF macro model.

For command description and syntax of the above-listed commands, refer to [Chapter 12, "Common Power Format Commands."](#)

Options of Compile-Time Ip Command for IEEE 1801

-check <1801_command> [<namePattern>]

Displays static information of the specified *<1801_command>* 1801 object or of the matched *<namePattern>* content if it matches the corresponding 1801 object hierarchical names or hierarchical member names.

The *<1801_command>* can be one of the following:

```
add_port_state  
add_pst_state  
associate_supply_set  
connect_logic_net  
connect_supply_net  
connect_supply_set  
create_hdl2upf_vct  
create_logic_net  
create_logic_port  
create_power_domain  
create_power_switch  
create_pst  
create_supply_net  
create_supply_port  
create_supply_set  
create_upf2hdl_vct  
set_design_attributes  
set_design_top  
set_domain_supply_net  
set_isolation  
set_isolation_control  
set_partial_on_translation  
set_port_attributes  
set_retention  
set_retention_control  
set_simstate_behavior  
set_related_supply_net  
upf_version
```

UXE Command Reference Manual

User Data Commands

For the list of supported IEEE 1801 commands and options in UXE, refer to *Using IEEE 1801 to Specify Low-Power Intent on Palladium XP in UXE User Guide*.

The `<namePattern>` can include wildcards * and ?.

The name list in the `-elements`, `-exclude_elements`, or `-pins` options of the 1801 commands is not bit blasted. Instead, the bits are grouped as a range. For example, `-elements {dut.a[1], dut.a[2], ..., dut.a[30]}` is shown as `-elements {dut.a[1:30]}`.

The 1801 object hierarchical names are `<scope>. <1801ObjectName>`. The isolation member names are `<scope>. <domain>. <isolation>`. The retention member names are `<scope>. <domain>. <retention>`. The pst state member names are `<scope>. <pst>. <pstState>`.

For example:

```
XEC> lp -check set_isolation testbench.DUT_MEMORY.ISO_MEMORY
      isolation {
        -name ISO_MEMORY
        -scope testbench -fileName scripts/dut_new.upf -lineno 30
        -domain testbench.DUT_MEMORY
        -diff_supply_only false
        -use_equivalence true
        -applies_to output
        -location automatic
        -clamp_value {0}
        -isolation_signal {dut.iso}
        -isolation_sense {high}
        -isolation_supply_set {testbench.SS_MAIN}
        -transitive true
        +version 2.0
      }
```

-check 1801 [<objectType>]

Displays complete list or the specified `<objectType>` list of power intent.

The `<objectType>` can be one of the following:

- domains
- logicPorts
- logicNets
- supplyPorts

UXE Command Reference Manual

User Data Commands

```
supplyNets
supplySets
switches
rootSupplies
equivalents
hdl2upf
upf2hdl
isолations | retentions
supplyPortStates
pst
pstStates
```

The names in the list are *<scope>. <objectSimpleName>*. For *<objectType>*, retention, isolation and pst state member names are given in the list too. The retention and isolation member names in the list are *<scope>. <domain>. <objectSimpleName>*. The pst state member names are *<scope>. <pst>. <pstStateSimpleName>*.

-check 1801 statistic

Provides statistic information for all IEEE 1801 commands. The output of this command lists the number of times each 1801 command is read.

For example:

```
XEC> lp -check 1801 statistic
power intent {
    create_power_domain 11
    create_logic_port 0
    create_logic_net 0
    connect_logic_net 0
    create_supply_port 4
    create_supply_net 7
    create_supply_set 6
    connect_supply_net 4
    connect_supply_set 0
    create_power_switch 5
    create_hdl2upf_vct 0
    create_vct2hdl_vct 0
    set_isolation 4
    set_retention 10
    add_port_state 4
    create_pst 2
```

UXE Command Reference Manual

User Data Commands

```
add_pst_state 6
add_power_state 0
describe_state_transition 0
set_repeater 0
set_design_top 1
set_port_attributes 0
set_design_attributes 0
map_isolation_cell (not applicable) 0
map_retention_cell (not applicable) 0
map_power_switch (not applicable) 0
use_interface_cell (not applicable) 0
set_simstate_behavior 0
upf_version 1
set_repeater 0
}
```

-check files

Displays the 1801 file names.

-check find_objects

Displays the result of the IEEE 1801 `find_objects` command at compile time.

The `lp -check find_objects` command enables you to check how the IEEE 1801 `find_objects` command works in `xeCompile`. The syntax of the command is slightly different from the IEEE 1801 `find_objects` command.

The syntax of the command is as follows:

```
lp -check find_objects {scope <scope> -pattern <search_pattern>
                         [-object_type [model | inst | port | net]]
                         [-direction [in | out | inout]]
                         [-transitive [TRUE | FALSE]]
                         [-regexp | -exact {}]
                         [-ignore_case {}]
                         [-non_leaf | -leaf_only {}]}
```

-check ls1801

Returns 1 if 1801 files are read in. Returns 0 if no 1801 information is present in the design.

memoryTransform

Enables or prohibits memory transformations on one or all user memories. This command should be executed before the compile command.

Without parameters, the `memoryTransform` command lists the current setting of the memory transform user data.

The syntax for this command is:

```
memoryTransform -add {<memory> <parameter_name> <parameter_value> [<parameter_name>
```

```
    <parameter_value>]}
```

```
memoryTransform -rm {<memory> [<parameter_name>]}
```

**-add {<memory> <parameter_name> <parameter_value>
[<parameter_name> <parameter_value>]}**

Applies the specified transform parameters and values to the specified memory instance.

The `<memory>` argument identifies the specific memory instance to which the transform parameters apply. Use an asterisk (*) if the transform parameters must be applied to all memories in the design.

The valid transform parameters and their values are as following:

<parameter_name>	<parameter_value>	Description
<code><COMPACT></code>	<code><on off></code>	Forces/prohibits memory compaction.
<code><MERGE></code>	<code><on off></code>	Enables/prohibits memory merging.
<code><SPLIT></code>	<code><on off></code>	Encourages/prohibits split memory transformation.
<code><SHALLOW></code>	<code><on off></code>	Forces/prohibits shallow memory to gate transformation.
<code><SQUEEZE></code>	<code><on off></code>	Enables/prohibits memory squeezing.
<code><ALL></code>	<code><on off></code>	Enables/prohibits all memory transformations.

UXE Command Reference Manual

User Data Commands

<parameter_name>	<parameter_value>	Description
<SPARSE>	<n> <p>	<p>Specifies that a memory instance with name <<i>memory</i>> should be treated as a sparse memory that can use at the most 2^n physical pages, and each page should contain 2^p words.</p> <p>The <<i>n</i>> parameter value is used to calculate the number of physical pages using the following formula:</p> $\text{Number of physical pages} = 2^n$ <p>The <<i>p</i>> parameter value is used to calculate the size of each page using the following formula:</p> $\text{Number of words per page} = 2^p$

Note:

- In the above description, meaning of the following terms is:
 - "Force" - the compiler will do the transformation if possible.
 - "Encourage" - the compiler will consider the user's preference for this transformation, but will not do it if it believes the choice is harmful.
 - "Allow" - the compiler will do the transformation if it believes the transformation is helpful.
- By default, only <COMPACT>, <SPLIT>, <MERGE>, <SQUEEZE>, and <SHALLOW> type of memory transformations are allowed on any memory.
- The <SPARSE> memory transformations happen only if it is set explicitly.

UXE Command Reference Manual

User Data Commands

Following parameters are applied only if * (asterisk) is used in the *<memory>* argument to specify all memories in the design:

<parameter_name>	<parameter_value>	Description
<i><shallowMemoryMaxUtilization></i>	<i><number></i>	Specifies the maximum utilization factor of the shallow memory. Here, <i><number></i> is an integer between 0 and 100. The default <i><number></i> is 80. For some designs, the user can improve the compile performance by specifying a <i><number></i> other than the default setting. In some cases setting a <i><number></i> lower than the default value improves the compile performance.
<i><shallowMemoryMaxAddrBits></i>	<i><number></i>	Specifies that the compile should not transform memory to gates if address bits are greater than the specified number. Here, <i><number></i> is an integer between 1 and 20. The default <i><number></i> is 4.
<i><memoryUtilization></i>	<i><number></i>	Forces memory compaction. <i><number></i> is a non-negative integer representing the utilization factor, given in percent of available DRAM size. If <i><number></i> is set to 100, the default behavior is not changed. This option only applies to memory compaction transformation.
<i><memoryMaxIgnoreDepth></i>	<i><number></i>	Specifies that the compiler should not transform a memory instance if its depth is equal to or less than $2^{<number>}$. Here, <i><number></i> should be equal to or more than 6. This option only applies to memory compaction transformation.
<i><memoryMaxIgnoreRemainder></i>	<i><number></i>	Specifies that the compiler should not transform a memory instance if its remainder is equal to or less than <i><number></i> . This option only applies to memory compaction transformation.

UXE Command Reference Manual

User Data Commands

<parameter_name>	<parameter_value>	Description
<memoryMinTransformDepth>	<number>	Transforms any memory instance if its depth is equal to or more than $2^{<\text{number}>}$. Here, <number> should be equal to or more than 6. This option only applies to memory compaction transformation.
<memoryMinTransformRemainder>	<number>	Transforms any memory instance if its remainder is equal to or more than <number>. This option only applies to memory compaction transformation.

-rm {<memory> [<parameter_name>]}

Removes the transform parameters applied on the specified memory instance. If the <parameter_name> argument is also specified, only the particular transform parameter is not applied on the identified memory instance.

Examples

- To force compaction and prohibit merging on memory mema, specify:

```
memoryTransform -add {mema COMPACT on MERGE off}
```

OR

```
memoryTransform -add {mema COMPACT on}  
memoryTransform -add {mema MERGE off}
```

- To prohibit all memory transformations on memory mema, specify:

```
memoryTransform -add {mema ALL off}
```

- To prohibit the shallow memory transformations on all memories in the design, specify:

```
memoryTransform -add {* SHALLOW off}
```

- To prohibit memory compaction on memory mema, specify:

```
memoryTransform -add {mema COMPACT off}
```

- To set memory mema to sparse memory with n=9, p=5, specify:

```
memoryTransform -add {mema SPARSE 9 5}
```

- To prohibit memory compaction, squeezing, and merging transformations on memory mema, specify:

```
memoryTransform -add {mema COMPACT off SQUEEZE off MERGE off}
```

UXE Command Reference Manual

User Data Commands

OR

```
memoryTransform -add {mem1 COMPACT off}  
memoryTransform -add {mem1 SQUEEZE off}  
memoryTransform -add {mem1 MERGE off}
```

- To remove the memory squeezing transformations on memory mem1, specify:

```
memoryTransform -rm {mem1 SQUEEZE}
```
- To remove all the memoryTransform settings on memory mem1, specify:

```
memoryTransform -rm {mem1}
```
- To set the memoryMinTransformDepth to 10 for all memories in the design, specify:

```
memoryTransform -add {* memoryMinTransformDepth 10}
```

memWritethru

Specifies the individual memory instances to be implemented as write-through or not.

The designation made by this option, whether the memory instance is write-through or not, overwrites the designation made by `memoryWritethru` option of the `compilerOption` command. For example, you set the `memoryWritethru` option of the `compilerOption` command to `ON`, and simultaneously, specify an individual memory instance as non-write-through using the `memWritethru` command. Then, that individual memory instance will not be implemented as write-through; whereas, the rest of the memories will be implemented as write-through.

The syntax for this command is:

```
memWritethru -add {<memory_instance_name> ON | OFF}  
memWritethru -rm <memory_instance_name>  
memWritethru -clear
```

-add {<memory_instance_name> ON | OFF}

Indicates if the specified individual memory instance needs to be implemented as write-through or not.

`ON` means that the instance is write-through and `OFF` means it is non write-through. The values, `ON` and `OFF`, are case-insensitive.

`<memory_instance_name>` must be valid design instance.

-rm <memory_instance_name>

Deletes an existing `memWritethru` record for the specified `<memory_instance_name>`.

-clear

Removes all existing `memWritethru` records.

moduleFV

Enables or disables specific modules for selective FullVision (FV) at run time.

The syntax for this command is:

```
moduleFV -add [<LIB> <cellName> enable | disable]  
moduleFV -rm [<LIB> <cellName>]
```

-add [<LIB> <cellName> enable | disable]

Adds a module for selective FV at run time. By default, FV is enabled on the entire design if nothing is specified with the `-add [<LIB> <cellName>]` option.

Note: The wildcard character asterisk (*) can be used when specifying the library name, that is, `<LIB>`.

When the design contains only one library, the user can use the following command syntax:

```
moduleFV -add {* <cellName> <enable|disable>}
```

For example,

```
moduleFV -add {* dut disable}
```

-rm [<LIB> <cellName>]

Removes the specified module from the `moduleFV` list.

moduleStub

Deletes all the instances of a module.

Note: Using the `-keepStubInputs` and `-keepStubOutputs` options of the `precompileOption` command causes the inputs and outputs of the stubbed modules to be treated as `keepNets`.

The syntax for this command is:

```
moduleStub -add {<LIB> <cellName>}
```

```
moduleStub -rm {<LIB> <cellName>}
```

-add {<LIB> <cellName>}

Deletes all the instances of the specified module, `<cellName>`, that belongs to the specified library, `<LIB>`.

Note: * and ? wildcards can be used inside library names and cell names to specify patterns.

-rm {<LIB> <cellName>}

Removes the instances of the specified module, `<cellName>`, belonging to the specified library, `<LIB>` that were previously marked for deletion.

multiplexCable

Enables pin multiplexing on the specified cable. For a multiplexed cable, the terminalAssign command can assign multiple top-level I/O signals of the design to the same physical pin. The signals are time-multiplexed, and terminalTiming constraints should be used to control the timing. Separate timing constraints can be given for the multiple signals assigned to a pin.

The syntax for this command is:

```
multiplexCable -add <cable_label>
```

-add <cable_label>

Adds the specified cable as a multiplexed cable.

Example

The following commands define a cable with time-multiplexed I/O:

```
compilerOption -add {ioDelayUnit step}
cableConnection -add ethernet2
multiplexCable -add ethernet2
terminalAssign -add {ethernet2 P6 1 ck0}
terminalAssign -add {ethernet2 P6 1 ck1}
terminalAssign -add {ethernet2 P6 3 out0}
terminalAssign -add {ethernet2 P6 3 out1}
delayBox -add {after_begin 8}
delayBox -add {after_ck0 8 ck0}
delayBox -add {after_out0 8 out0}
delayBox -add {after_ck1 8 ck1}
terminalTiming -add {ck0 after_begin}
terminalTiming -add {out0 after_ck0}
terminalTiming -add {ck1 after_out0}
terminalTiming -add {out1 after_ck1}
```

The signals ck0, ck1, out0, and out1 are unidirectional top-level outputs of the design. Signals ck0 and ck1 are assigned to one HDDC pin, and signals out0 and out1 are assigned to another pin. Signals ck0, ck1 might be tied in the design to 1 and 0 respectively.

The timing constraints ensure the following:

- The signals are scheduled in the order – ck0, out0, ck1, out1.

UXE Command Reference Manual

User Data Commands

- Signal `ck0` is scheduled at least 8 steps after the beginning of the FCLK cycle.
- There is a separation of at least 8 steps between each pair of signals in the sequence.

multiSampledTerminal

Adds or removes a top-level I/O terminal as one of the connections to a clock domain to be sped up.

Note: Speeding up of clock domains is intended for expert users. It requires deep knowledge of the user's design, and how that design is compiled and evaluated on the emulator.

Refer to the [compilerOption](#) command for more information on multisampling.

Specify this option before using `compilerOption -add {enableMultiSampledIO 1 | 0}` to turn the connection on or off.

The syntax for this command is:

```
multiSampledTerminal -add <terminal_name>  
multiSampledTerminal -rm <terminal_name>
```

-add <terminal_name>

Adds a terminal as a clock domain connection.

-rm <terminal_name>

Removes the terminal's clock domain connection.

netWeakDrive

Enables control over resolving the value of a tristate terminal when no active driver on the emulator drives the terminal.

Execute this command after design import and before precompile.

The syntax for this command is:

```
netWeakDrive -add {<name> <weakDriveValue>} ...
netWeakDrive -rm {<name> <weakDriveValue>}
netWeakDrive -clear
```

-add

Adds a top terminal weak driver.

<name>

A top terminal name. This can be the name of a CA/IP terminal, for example: i1:i2:i3:t3.

<weakDriveValue>

The *<weakDriveValue>* can be one of the following:

- pullUp

The *pullUp* value causes the tristate terminal to get the value of 1, unless the target drives it to 0. (The target should not have a pull down on the signal.)

- pullDown

The *pullDown* value causes the tristate terminal to get the value 0, unless the target drives it to 1. (The target should not have a pull down on the signal.)

- externalDriver

The *externalDriver* setting means that the target is expected to drive the terminal, either by a strong driver or a weak pullup/pulldown device.

UXE Command Reference Manual

User Data Commands

-rm

Removes the top terminal weak driver.

-clear

Deletes all the netWeakDrive data.

noBreakPath

The syntax of the command is:

`noBreakPath -add {<startnet> <endnet>}`

`noBreakPath -rm {<startnet> <endnet>}`

-add {<startnet> <endnet>}

Directs loopbreaking not to insert any break on a path from `<startnet>` to `<endnet>`, nor any break on a fanout of `<endnet>`. Forbidden paths include combinational paths, flip-flop and latch clock-to-Q, reset-to-Q, set-to-Q, and latch D-to-Q. If forbidden breaks are required to break a loop, precompile fails unless the `allowNoBreakViolations` option of the `precompileOption` command is specified.

-rm {<startnet> <endnet>}

Removes the `{<startnet> <endnet>}` pair from the list of `noBreakPath` pairs.

noOverlapClocks

Specifies one or more sets of non-overlapping clocks. In a latch-based design that uses a non-overlapping clock methodology, you can sometimes improve emulation speed by providing information about non-overlapping clocks. The information is used by precompile's loop-breaking algorithm. For further explanation, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

The syntax for this command is:

```
noOverlapClocks -add {<expression>...} ...
noOverlapClocks -rm {<expression> ... } ...
```

-add {<expression>... } ...

Adds one or more sets of non-overlapping clocks. Each set of non-overlapping clocks is enclosed in curly braces, {}. Each *<expression>* must be of the form *signal* or *~ signal*. The ~ symbol represents negation. A space is required between ~ and *<signal>*. The compiler assumes that no two expressions in a set will ever simultaneously have the value 1.

-rm {<expression> ... } ...

Removes one or more sets of non-overlapping clocks.

Examples

The following command indicates that signals A and B will never simultaneously have the value 1.

```
noOverlapClocks -add {A B}
```

The following command indicates that signals A and B never simultaneously have value 1, and no two of the four expressions C, D, ~E, ~U1__F ever simultaneously have the value 1.

```
noOverlapClocks -add {A B} {C D ~ E ~ U1__F}
```

The following commands are equivalent to the previous command. These two commands indicate that signals A and B never simultaneously have value 1, and no two of the four expressions C, D, ~E, ~U1__F ever simultaneously have the value 1.

```
noOverlapClocks -add {A B}
noOverlapClocks -add {C D ~ E ~ U1__F}
```

orMultiDrv

Enables multiple drivers on a single net, and makes the value of the net equal to the OR of all the values driven.

A tristate driver affects the value of the net only when the tristate driver is enabled. `supply0` and `supply1` declarations are treated as drivers.

The syntax for this command is:

`orMultiDrv -add {<signal>}`

`orMultiDrv -rm {<signal>}`

-add {<signal>}

Specifies the name of the signal on which multiple drivers are to be allowed.

-rm {<signal>}

Specifies the name of the signal on which multiple drivers are *not* to be allowed.

Examples

`orMultiDrv -add {sig_s1}`

The above command enables multiple drivers on the `sig_s1` signal and makes the signal value equal to the OR of all the driven values.

precompileOption

This command specifies options for the precompile process. The options are stored as part of the user data of the design, and are later used automatically when you execute the XEL [precompile](#) command without options, or when the XEL [compile](#) command runs precompile.

The syntax for this command is:

```
precompileOption -add pullUpOnly | -add pullUp | -add pullDown
precompileOption -add primaryIOPullup | -add primaryIOPulldown | \
-add primaryIOExternalDrive
precompileOption -add retainState
precompileOption -add noTimeCheck | -add full
precompileOption -add tieSourceless | -add [tieSourcelessRandom | tieSourcelessRandom <seed>] | \
-add tieSourcelessUnsafe | -add tieSourcelessHigh | -add tieSourcelessLow | \
-add removeSourceless | -add keepSourceless
precompileOption -add {removeLoadless <N>}
precompileOption -add failOnMultiDrv
precompileOption -add failOnUnresolvedOOMR
precompileOption -add dontKeepMem | -add keepAllMem | -add keepMem
precompileOption -add syncData
precompileOption -add asyncReset
precompileOption -add syncReset
precompileOption -add {cdc <CDC file name>}
precompileOption -add debug
precompileOption -add allowUnresolvedOOMR
precompileOption -add dontLogOOMR
precompileOption -add logAIIOMR
precompileOption -add ignoreEmptyCells
precompileOption -add keepEmptyCells
precompileOption -add keepStubInputs
precompileOption -add keepStubOutputs
precompileOption -add ignorePowerDrvConflict
precompileOption -add ignoreMultiDrv
precompileOption -add disconnectTristateDrivers
```

UXE Command Reference Manual

User Data Commands

```
precompileOption -add andMultiDrv
precompileOption -add orMultiDrv
precompileOption -add dumpShadows
precompileOption -add noLoopBreaks
precompileOption -add nolsolationBreaks
precompileOption -add keepBreaks
precompileOption -add {loopsDepth <number>}
precompileOption -add dontMinimizeDataBreaks
precompileOption -add minimizeBreaks
precompileOption -add keepState
precompileOption -add {logAllLoops <number>}
precompileOption -add moreExclusive
precompileOption -add clockDataLoops
precompileOption -add noReset1X
precompileOption -add breakHalfCycleAll
precompileOption -add breakHalfCycle
precompileOption -add setPassiveShadows
precompileOption -add recognizeClockDivider
precompileOption -add cake like 1x
precompileOption -add cake like 2x
precompileOption -add moreClockSources
precompileOption -add allowDelayClkNets
precompileOption -add 1xLessClockPaths
precompileOption -add verify
precompileOption -add latchResetAsFF
precompileOption -add dontMinimizeBreaks
precompileOption -add clockLoops30
precompileOption -add clockLoops101
precompileOption -add combinLoops101
precompileOption -add dataLoops101
precompileOption -add resetLoops101
precompileOption -add {duplicateLoops <number>}
```

UXE Command Reference Manual

User Data Commands

```
precompileOption -add safeLoops
precompileOption -add {seed <number>}
precompileOption -add randomizeMemories
precompileOption -add dontRandomizeMemories
precompileOption -add readTerminalAssign
precompileOption -add instrumentKeepNets
precompileOption -add noPowerIntent
precompileOption -add domainInterfaceDef [1.0 | 2.0 | 2.1 | UPF | 1801_2009 | 1801_2013]
precompileOption -add allowNoBreakViolations
precompileOption -add VSSswitchable
precompileOption -add breakTerminalTiming
precompileOption -add lessProbes
precompileOption -add allowClockConflicts
precompileOption -add instrumentEdges
precompileOption -add createAllBoundaries
precompileOption -add dontMergeCommon
precompileOption -add {1XrestrictPath <path>}
```

-add pullUpOnly

Treats all internal tristate nets as pullup type. It overrides all explicit pulldowns and retainstates.

-add pullUp

Treats each internal tristate net as pullup type, unless the net has a pulldown or retainstate element in the netlist. The default setting is pullup.

-add pullDown

Treats each internal tristate net as pulldown type, unless the net has a pullup or retainstate element in the netlist.

-add primaryIOPullup

Applies a weak pullup to each bidirectional or tristate primary IO signal that has neither a pulldown in the netlist nor any `terminalWeakDrive` user data. The default setting for primary I/O signals is `primaryIOPullup`.

-add primaryIOPulldown

Applies a weak pulldown to each bidirectional or tristate primary IO signal that has neither a pullup in the netlist nor any `terminalWeakDrive` user data.

-add primaryIOExternalDrive

Specifies that neither a pullup nor a pulldown should be applied to a bidirectional or tristate primary IO signal that does not have any pullup nor pulldown in the netlist and does not have any `terminalWeakDrive` user data. When the emulator is not driving the signal, it is expected to be driven by an external target, either actively or by a weak pullup or pulldown.

-add retainState

Treats each internal tristate net as retain-state type, unless the net has a pullup or pulldown element in the netlist.

-add noTimeCheck

Forces FULL mode and reloads the QTDB hierarchical database. Do not compare the `timeStamp` between the QTDB hierarchical database and DADB.

-add full

Forces FULL mode and reloads the QTDB hierarchical database. In addition, reruns the feedback loop breaking.

Note: This command option is similar to `-noTimeCheck`.

-add tieSourceless

Ties sourceless user nets to a “safe” constant:

- The net is tied to PWR if the net drives an AND gate
- The net is tied to GND if the net drives an OR gate.
- In case of ambiguity, the net is tied to GND.

-add [tieSourcelessRandom | tieSourcelessRandom <seed>]

Initializes the sourceless nets to either 0 or 1 randomly. If the seed is not specified, a random seed is picked automatically.

-add tieSourcelessUnsafe

Initializes the sourceless nets to the opposite constant as opposed to the `tieSourceless` option.

The `tieSourceless` option initializes to "safe" constants, that is, if a sourceless net drives the AND gates only, the net is initialized to 1; if a sourceless net drives the OR gates only, the net is initialized to 0; in case of uncertainty, the net is initialized to 0.

The `tieSourcelessUnsafe` option works in contrast to the `tieSourceless` option. If a sourceless net drives the AND gates only, the net is initialized to 0; if a sourceless net drives the OR gates only, the net is initialized to 1; in case of uncertainty, the net is initialized to 1.

-add tieSourcelessHigh

Ties sourceless user nets to power.

-add tieSourcelessLow

Ties sourceless user nets to ground. This is the default setting.

-add removeSourceless

If `-removeSourceless` is specified, the Compiler disconnects sourceless cones in the design as if they are "weak" logic that does not affect anything.

-add keepSourceless

If the `keepSourceless` option is specified, the Compiler treats all sourceless nets as keepnets, and, therefore, forcible.

Note: Constants, for example supply [01], are not sourceless nets.

-add {removeLoadless <N>}

Directs precompile to remove loadless logic including gates, flip-flops, latches, and memories. The removal of loadless logic reduces the emulator resources needed by a design. The `<N>` argument is optional. If specified, the `<N>` argument controls whether or not loadless memories are removed. You can specify the following values for `<N>`:

- 0: all memories are preserved (no memories are removed).
- 1: write-only memories are preserved, but all other loadless memories are removed.
- 2: all loadless logic is removed, including memories.

Note: If `<N>` is not specified, the default setting is the same as `<N> = 2`.

Logic is considered loadless if there is no path from it to a primary output, external output, or `keepNet`. Examples of external outputs include outputs to a component adapter instance, and signals monitored via OOMRs in a testbench.

Execution of the `-add removeLoadless` command option restricts the user from performing debug operations, such as viewing waveforms, dumping memory contents, and forcing signals, for the removed loadless logic and nets. To preserve specific nets, declare them as `keepNets`.

If both `-add debug` and `-add removeLoadless` options of the `precompileOption` command are used, a report file `tmp/loadless_ff.rpt` is generated. It lists the output nets of flip-flops and latches removed due to the `-add removeLoadless` command option.

Note: Removed memories are not listed. However, the removal of memories can be verified at run time using the [memory \(in ICE Mode\)](#) command.

-add failOnMultiDrv

In SA mode, the `failOnMultiDrv` option causes precompile to fail if a net has multiple non-tristate drivers, or has both tristate and non-tristate drivers. In ICE mode, by default, such multiple drivers cause precompile to fail. In SA mode, by default, multiple drivers are combined automatically; precompile issues a warning but does not fail.

-add failOnUnresolvedOOMR

Causes precompile to fail if an unresolved OOMR reference is detected.

-add dontKeepMem

Causes precompile not to automatically treat any bits of HDL_MEMORY arrays as keepNets. For more information, refer to -add keepMem.

-add keepAllMem

Causes precompile to automatically treat all bits of HDL_MEMORY arrays as keepNets. For more information, refer to -add keepMem.

-add keepMem

Causes precompile to automatically treat as keepNets, those bits of HDL_MEMORY arrays that are not driven by buffers, inverters, or combinational gates.

Usually in RTL, a multi-dimensional array (of dimension two or higher) represents a memory that can store values. Sometimes, however, bits of an array depend combinationally on other signals, and thus cannot store values. When HDL Compiler synthesizes a multi-dimensional array without using a hard macro (MPR/MPW) memory, it marks the array using a HDL_MEMORY pragma in the gate-level netlist, similar to the following:

```
// pragma CVASTRPROP MODULE HDLICE HDL_MEMORY_DECL_m9 "A 1 15 0 63 0"
```

The dontKeepMem, keepMem, and keepAllMem options are mutually exclusive. For all other modes, dontKeepMem is treated as default. With keepMem, only those bits of the HDL_MEMORY arrays that are either undriven or driven by latches or flip-flops, are automatically treated as keepNets. With dontKeepMem, except for those bits that are explicitly specified as keepNets in the userData, none of the bits are treated as keepNets.

The **benefit** of treating the HDL_MEMORY array bits as keepNets is that the bits treated as keepNets can be loaded from a file (for example, using a \$readmemh operation or XEL memory -load command). However, if the bits that are driven by combinational gates (including buffers or inverters) are loaded, the loaded values will remain only transiently before they are overwritten by the driving gates.

The **disadvantage** of treating HDL_MEMORY array bits as keepNets is a possible severe degradation in step count. Step-count degradation is only likely when keepAllMem is in effect. The degradation happens when multiple array bits are driven by the same (or

equivalent) combinational logic. HDL Compiler synthesizes a long chain of buffers driving these equivalent bits, and precompile automatically treats the bits as `keepNets` so that the chain of buffers is not optimized. This causes step count degradation.

-add syncData

Models the data inputs of latches synchronously. Default is `async`.

-add asyncReset

Models the set/reset inputs of flip-flops and latches asynchronously (without delay).

Default behavior without `-asyncReset` or `-syncReset` is with active edge applied asynchronously, and passive edge applied synchronously (delayed one clock cycle).

The `-asyncReset` and `-syncReset` options are mutually exclusive.

-add syncReset

Models the set/reset inputs of flip-flops and latches synchronously: delay one fast clock (FCLK) cycle.

Behavior without `-asyncReset` or `-syncReset` is with active edge applied asynchronously, and passive edge applied synchronously (delayed one clock cycle).

The `-asyncReset` and the `-syncReset` options are mutually exclusive.

-add {cdc <CDC_file_name>}

Specifies the Clock Domain Crossing (CDC) file that describes the cross-domain synchronizers. For detailed information about the format of a CDC file, refer to *Files and File Formats* in the *UXE User Guide*.

-add debug

Causes precompile to log information about the optimizations that it performs. When a design uses lesser emulator capacity than expected, use a `clean.request` file containing the `-random` command to find the reason for the low usage. For more information, refer to “[How to investigate precompile optimization](#)” on page 330.

When running with the `-add debug` option, precompile creates a file `tmp/clean.rpt` with two kinds of lines. Here are examples of each kind:

- `net u1.out2n` is disabled in favor of `[-] u1.out2`.

Precompile removes buffers and inverters and their output nets, connecting loads of the output nets to the input nets instead, with an inversion bit turned on, when necessary.

The example above indicates that the net `u1.out2n` has been removed, and its load has been instead connected to `u1.out2`, with inversion bit set. A + character indicates that the inversion bit is not set.

- `net u1.out1` is tied LOW.

The example above indicates that precompile's optimization algorithm could prove that the net `u1.out1` would be always low. So, it removed the driver of the net, and tied the net low.

-add allowUnresolvedOOMR

Enables the precompile process to continue even when unresolved OOMRs are found. By default, precompile fails on finding unresolved OOMRs.

-add dontLogOOMR

Disables creation of logs for all OOMRs during precompile. The logs are not created at all for your defined OOMRs and the OOMRs generated by the IXCOM compiler. Use the `dontLogOOMR` option to improve the precompile performance.

-add logAllOOMR

By default, precompile does not log instrumentation OOMRs (that is, OOMRs generated by the IXCOM compiler). The `logAllOOMR` option enables precompile to generate logs for all successful OOMRs including the instrumentation OOMRs in `OOMR.msg`. These logs help in debugging.

-add ignoreEmptyCells

Specifies that the compiler should ignore empty cells (that is, cells containing neither sub-cell instances nor feedthrough connections). When this command option is used, all instances of empty cells and their terminals are deleted from the design database. As a result, any command (such as `clockSource`) on these deleted terminals is not honored.

-add keepEmptyCells

Enables `precompile` to handle designs that contain empty cells, without deleting the instances of empty cells and their terminals from the design database. By default, empty cells cause `precompile` to fail.

-add keepStubInputs

Causes `precompile` to treat inputs of the modules and instances stubbed using the `moduleStub` and `instanceStub` commands as `keepNets`.

-add keepStubOutputs

Causes `precompile` to treat outputs of the modules and instances stubbed using the `moduleStub` and `instanceStub` commands as `keepNets`.

-add ignorePowerDrvConflict

Ignores errors due to a conflict arising in situations when a net that is assigned to power or ground also has a driver that is not power or ground. The default is to fail with an error when this conflict arises. With the `-ignorePowerDrvConflict` option, `precompile` continues with a warning and the power or ground assignment is given priority.

-add ignoreMultiDrv

Ignores errors due to multiple drivers. Disconnects all drivers except one (chosen arbitrarily), then proceeds. The default is to fail if multiple drivers are encountered.

Note: The `precompile` process will report an error and fail if multiple drivers are detected. You should review the reason for regular nets being driven by more than one driver. If you want, you can use the `-ignoreMultiDrv` switch to continue, but note that the compiler result will be unpredictable because only one driver will remain on the net, and you cannot predict which one. To achieve a predictable result, instead of using `ignoreMultiDrv`, use one of the following command options with `precompileOptions`:

- [-add disconnectTristateDrivers](#)
- [-add andMultiDrv](#)
- [-add orMultiDrv](#)

Alternatively, use one of the following XEL commands to control the treatment of particular nets:

- [disconnectDriverPin](#)
- [andMultiDrv](#)
- [orMultiDrv](#)

-add disconnectTristateDrivers

Causes tristate drivers to be automatically disconnected (with a warning) from any net that also has regular drivers. By default, if a net has both tristate and regular drivers, precompile fails.

Examples of tristate drivers are: bufif1, Q_BUZP, and Q_RBUZP.

Examples of regular drivers are: combinational gates, flip-flops, and buffers.

Note: If the disconnectTristateDrivers option is specified along with orMultiDrv, andMultiDrv, or ignoreMultiDrv, the disconnectTristateDrivers operation is applied first, so that the tristate drivers do not affect the value of a net that also has regular drivers.

-add andMultiDrv

Enables multiple drivers on a single net, and makes the value of the net equal to the AND of all the values driven. If this option is not used and a net has multiple drivers, precompile fails, unless all the drivers are tristate drivers.

A tristate driver affects the net's value only when the tristate driver is enabled. supply0 and supply1 declarations are treated as drivers.

-add orMultiDrv

Enables multiple drivers on a single net, and makes the value of the net equal to the OR of all the values driven. If this option is not used and a net has multiple drivers, precompile fails, unless all the drivers are tristate drivers.

A tristate driver affects the net's value only when the tristate driver is enabled. Supply0 and supply1 declarations are treated as drivers.

-add dumpShadows

This switch is for tracking capacity overhead in 1X mode only. It is not applicable to non-1X modes. It dumps the primitives driving nets that have shadows. Other portions of the netlist are not dumped. The primitive are dumped to the file `<design_dir>/tmp/precompile.dumpShadows` in the same format as a standard dump.

Note: The `dumpShadows` option is deprecated. In most situations, it is easier to create a `shadows.request` file, run `precompile`, and analyze the resulting report.

-add noLoopBreaks

Forces `precompile` to fail if a loop is detected.

The loop-breaking algorithm works as follows:

First, exclusivity-point breaks are added. Exclusivity-point breaks cannot affect design behavior or waveforms, and are called "safe". Then, the algorithm looks for loops in the design that has the safe breaks inserted. If the `noLoopBreaks` option has been specified and a loop is found, `precompile` fails. Otherwise, an "unsafe" break is added, and the process (look for loops; if any found, add an unsafe break) repeats until no loops remain.

-add nolsolationBreaks

Hides isolation logic from loop-breaking algorithms. Use this command option when the following three situations occur:

- The CPF file has isolation rules.
- A functional failure is caused by loop breaks.
- Loop-breaking functions correctly without isolations.

-add keepBreaks

With this option, `precompile` inserts non-exclusivity-point breaks from the file `PDB/*.brk` file, if it exists, before performing loop-breaking analysis. For information on the `*.brk` file, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

When reading the `*.brk` file, `-add keepBreaks` option ignores all exclusivity point delays and DLY cell instances. A non-DLY cell instance is ignored with a warning if its instance name or type fails to match the instance name and type of an instance in the design. Currently, user delays in the file are inserted, but they might be ignored in a future release.

If you suspect that loop-breaking changed in a certain release, and caused the design not to work, sometimes this can be proved by using `-add keepBreaks` option with the `*.brk` file created by the older, working release. However, using `-add keepBreaks` option with the `*.brk` file from a different release might give confusing results, if the `*.brk` file includes any breaks on instances whose names changed due to the change in release. (Such a name change could happen for gate instances added by precompile itself, for example during tristate net conversion.)

In `1X` mode, breaks can be half- or full-FCLK. If you use `shadowsData` to reduce the number of shadows, a side-effect can be to change some loop breaks from half- to full-FCLK, and the change in loop breaks can cause the design to fail. To investigate such a problem, use `-add keepBreaks` option with the `*.brk` file from the working database. `-add keepBreaks` option preserves any half-FCLK breaks in `*.brk` as half-FCLK breaks. However, if `*.brk` file contains a full-FCLK break on a net with a shadow, precompile inserts a half-FCLK break. If `-add breakHalfCycleAll` option is used, all breaks are taken as half-FCLK.

Precompile overwrites the `PDB/*.brk` file with a new file showing the breaks actually inserted.

-add {loopsDepth <number>}

By default, `precompile` minimizes the number of breakpoints inserted to break asynchronous loops. However, to reduce the time required for `precompile`, when a certain analysis depth is reached, `precompile` switches to a faster algorithm that does not minimize breakpoints. This algorithm usually inserts more breakpoints, which might require more oversampling (FCLKs per design clock) to make the design run correctly. By default, the maximum analysis depth is 128. You can set it to a higher or lower number by specifying the value of `n` with the `-loopsDepth` option.

-add dontMinimizeDataBreaks

Adds extra break points during compilation. By default, `precompile` minimizes the number of inserted breakpoints in a design that has asynchronous loops going through latch D pins. Usually, minimizing breakpoints gives the best chance that the design can run correctly without extra oversampling (FCLKs per design clock), and is, therefore, preferred.

However, in some designs, using the `-dontMinimizeDataBreaks` option with the `precompileOption` command can have the following advantages:

- The `precompile` process might take less time.

- The extra break points usually inserted by the `-dontMinimizeDataBreaks` option might break long paths, thereby, reducing the compilation step count and increasing the emulation speed.
- Sometimes, contrary to expectation, the `-dontMinimizeDataBreaks` option reduces the oversampling needed for correct functionality.

-add minimizeBreaks

Removes the limit on time spent removing redundant loop breaks. In rare cases, this option might improve the loop-breaking result at the expense of increasing precompilation time.

-add keepState

Disables flip-flop optimization.

-add {logAllLoops <number>}

Sets limit for dumping data/set/reset loops. The default is 0. To log all loops, use a number greater or equal to the number of delays reported in the `PDB/*.brk` file. The log information is stored in the `PDB/<lib>. <cell>.loops` file.

Precompile limits the size of the `*.loops` file to 128 MB and, by default, stops recording loops when this limit is reached. However, if you use the `-logAllLoops` option (with any `<number>` more than 0), this limit is ignored and precompile will keep writing the file. Therefore, `-logAllLoops 1` might be needed just to observe all combinational and clock loops regardless of the size of `*.loops` file.

Each loop is reported a sequence of gates between {} characters. For each gate, an @ character shows the input pin on the gate which the loop goes through. An = character shows where the loop was broken.

For more information about this option, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

-add moreExclusive

Widens the search for gate-input exclusivity points. In rare cases, this option might find more exclusivity points and thus improve the loop-breaking result, at the expense of increasing compilation time.

-add clockDataLoops

This option is obsolete and is ignored.

-add noReset1X

With this option, a shadow on flip-flop or latch set or reset does not cause a shadow on the flip-flop or latch output. This can save capacity in 1X mode compilation for some designs. However, with this option, if a set or reset signal becomes active on the falling edge of the primary clock, the flip-flops and latches are not set or reset until the next rising edge of the primary clock.

-add breakHalfCycleAll

Sets user-added and loop-breaking delays consistently to one-half FCLK.

Note: This option could have a large capacity cost in some designs, especially when used in CAKE 2x or higher modes.

-add breakHalfCycle

Sets most user-added and loop-breaking delays to one-half FCLK. This option might have a large capacity cost in some designs, especially when used in CAKE 2x or higher modes. The capacity cost is less than or equal to that of -add breakHalfCycleAll.

This option is similar to -add breakHalfCycleAll, except that with -add breakHalfCycle option, delays in parts of the design unreachable from CAKE clock sources are a full FCLK, rather than a half FCLK.

-add setPassiveShadows

In 1X mode, disables the optimizations made in UXE and causes extra logic duplication like the Incisive XE 5.x behavior.

-add recognizeClockDivider

Enables loop-breaking, when tracing forward from clock sources to label (clk) nets, to trace from clock to Q of simple clock dividers. A flip-flop is recognized as a clock divider if:

- the output net is not a bit of a bus

- the D input is the inversion of the output
- the flip-flop set, reset and enable inputs (if any) are tied inactive

Note: Without this option, tracing is possible only through pure combinational gates.

-add cake_like_1x

Causes precompile to generate CAKE 1x style clock waveforms even when CAKE 2x is specified. For an explanation of 1x and 2x clock waveform styles, see the description of the [clockOption](#) command.

When the `cake_like_1x` option is used, the restrictions to the [clockConfig](#) command that normally apply in the CAKE 1x mode also apply in the CAKE 2x mode.

-add cake_like_2x

For clocks that are not an exact multiple of the FCLK, the rising edges coincide with both the rising and falling edges of the fastest clock. This option is applicable only if `clockOption -add {technology CAKE 1}` is in effect; otherwise, it is ignored.

-add moreClockSources

Causes any nets that are connected directly or through buffers and inverters to a flip-flop or latch clock pin, to be treated as extra clock sources during loop-breaking (when tracing forward from clock sources to label (clk) nets), if the net would not otherwise be labeled (clk). The “More clock sources” section of the `tmp/* .MultiClock` file lists the extra clock sources.

-add allowDelayClkNets

Causes loop-breaking to use heuristics. In most designs, the `-add allowDelayClkNets` option increases the likelihood that loop-breaking will delay fanouts of (clk) nets, causing incorrect emulation behavior. So, use this option only if it is needed.

-add 1xLessClockPaths

Causes 1X mode shadows calculation to make a possibly-incorrect assumption. The possibly-incorrect assumption is that a clock path (a path taken by a signal transition from a clock source to a flip-flop or latch clock pin) never goes from D to Q of a latch. The `-add`

`1xLessClockPaths` option might cause incorrect emulation behavior in designs which violate its assumption. So, use this option only if it is needed.

-add verify

Enables a feature that detects, at run time, clocks that were delayed by loop-breaking. For more information, refer to *Loop-breaking* appendix of the *UXE User Guide*.

-add latchResetAsFF

With this option, the response of latches to set or reset is similar to that of flip-flops. By default, if a latch's set or reset goes inactive at the same time as the latch enable goes inactive, the latch gets the previous value of its D input. With `-add latchResetAsFF` option, in this scenario the latch remains in the set or reset condition.

The `-add latchResetAsFF` option might be used in conjunction with the `-add noReset1X` option. By default, a shadow on latch set or reset propagates to Q, even if `-add noReset1x` is used. With `-add latchResetAsFF` option, a shadow on latch set or reset does not propagate to Q.

-add dontMinimizeBreaks

Disables the removal of redundant loop breaks. Loop-breaking adds breaks one by one. By default, if a break makes a previous break unnecessary (redundant), precompile removes the redundant break. The `-add dontMinimizeBreaks` option specifies that the redundant breaks are not removed.

-add clockLoops30

Causes loop-breaking to use heuristics. By default, loop-breaking prefers to break at a non-(clk) input of a gate whose output is (clk). With the `-add clockLoops30` option, loop-breaking gives higher priority to breaking as far back as possible in a chain of gates. The `-add clockLoops30` option might also cancel later enhancements that attempt to avoid delaying (clk) nets.

-add clockLoops101

This is a rarely used option that affects loop-breaking. For details on this option, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

-add combinLoops101

This is a rarely used option that affects loop-breaking. For details on this option, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

-add dataLoops101

This is a rarely used option that affects loop-breaking. For details on this option, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

-add resetLoops101

This is a rarely used option that affects loop-breaking. For details on this option, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

-add {duplicateLoops <number>}

Controls the logic duplication for feedback loops. Logic duplication effectively eliminates a loopbreak, that is, design functions, as if this loopbreak was not set. For detailed information on handling loopbreaks, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

The *<number>* argument specifies the overhead limit, measured as percent to the design size. The default value of *<number>* is 20, which means limiting the overhead to 20% of the design size.

Specifying *<number>* as 0 switches off logic duplication completely.

-add safeLoops

Compiling with this option eliminates, in most cases, any harmful effect of loop-breaking delays on design function.

The drawbacks of the option are as follows:

- The design runs at least two times slower than normal.
- FullVision (FV) mode is not supported with `safeLoops`. You must use `visionMode DYNP`.

To use `safeLoops`, do the following:

UXE Command Reference Manual

User Data Commands

1. Compile the design using `precompileOption -add safeLoops` and `compilerOption -add {visionMode DYNP}`. Use ICE mode with CAKE oversampling ratio set to 2 or higher or SA mode flow without `+1xua`.
2. Run the design as usual. For most designs, `safeLoops` eliminates any harmful effects of loop-breaking on design function, and the design runs correctly. However, for a few designs, the run might get stuck, so that (in SA mode) time does not advance or (in ICE mode) design clocks do not toggle. The reason has been explained below. If the run gets stuck, kill the running `xeDebug` or `irun` and proceed to step 3.
3. If the run gets stuck in step 2, do the following:

Run the design again. After `download` and `configPM` (in ICE mode) or after you execute `run -swap` (in SA mode), do the following:

```
force CVA_UA_SAFE_LOOPS_SELECT 1  
force CVA_UA_SAFE_LOOPS_PERIOD[7:0] 'd<n>
```

where `<n>` is a number between 0 and 255. Usually, `<n>=1` is sufficient to eliminate the effects of loop-breaking. But in rare cases, a large number, such as 2 might be required. If you use `<n>=0`, the run is the same as if the design had been compiled without `safeLoops`.

Palladium's Implementation of `safeLoops`

When a design compiled with `safeLoops` is run, each FCLK cycle in a non-`safeLoops` run is replaced by a number of special cycles, followed by a normal FCLK cycle. During special cycles, values propagate across loop-breaking delays, `breakNet` and `breakPin` delays, and across all the delays internal to flip-flops and latches except the D input delay of a flip-flop. Values do not propagate across `Q_FDP0B` delays nor across the D input delay of a flip-flop. MPR/MPW memories are written only in normal cycles.

If `CVA_UA_SAFE_LOOPS_SELECT` is not forced to 1, Palladium automatically runs special cycles until another special cycle would produce no change. (If a logic loop with its break causes oscillation, special cycles continue to run forever, and run gets stuck.)

If `CVA_UA_SAFE_LOOPS_SELECT` is forced to 1 and `CVA_UA_SAFE_LOOPS_PERIOD[7:0]` is forced to `<n>`, then `<n>` special cycles are run. Forcing `CVA_UA_SAFE_LOOPS_PERIOD` only has an effect if `CVA_UA_SAFE_LOOPS_SELECT` is forced to 1.

For an ICE mode design, or an IXCOM design using the `-captureMode uncontrolled` option of the `database` command, the waveform shows both special and normal cycles. For an IXCOM design using the (default) `-captureMode controlled` option, the waveform shows both special and normal cycles if the `-event` option is used and `Expand Sequence Time` is used in SimVision; otherwise only the last cycle in each time bin is shown, which is

usually the last special cycle before a normal cycle. SDL does not distinguish between special and normal cycles.

If you compile `safeloops` and run without forcing `CVA_UA_SAFE_LOOPS_SELECT`, and the design does not get stuck, then any effect of `loopbreaks` or `breakNet` / `breakPin` delays on design function (as viewed at the end of normal cycles) is eliminated.

-add {seed <number>}

Specifies to use the user-defined seed `<number>` at the time of compilation. The seed `<number>` should be a non-negative integer.

When compiling designs with CPF information, using the `-add seed <number>` option is useful. This is because though you can specify random values for CPF command options like `-power_down_state` and `-power_up_state`, CPF standard does not allow specification of random seed values.

-add randomizeMemories

Enables the feature to handle CPF or IEEE 1801 memories like *real* memories based on the power state defined using the `-power_up_states {high | low | random | inverted}` and `-power_down_states {high | low | random | inverted}` options of the `create_power_domain` command.

-add dontRandomizeMemories

Disables the feature to handle CPF memories based on the power state defined using the `-power_up_states` and `-power_down_states` options of the `create_power_domain` command.

Note: By default, the feature that causes the memories to mimic the behavior of *real* memories when powered up and/or down is enabled. You need to explicitly give the `precompilOption -add dontRandomizeMemories` command to disable the default behavior.

-add readTerminalAssign

Specifies that precompile should exclude unassigned primary inputs when building instrumentation for FV and InfiniTrace vision modes.

When compiling designs with InfiniTrace enabled, the implementation needs a deep memory with width equal to the number of primary inputs (including bidirectionals) having target terminal assignments. However, by default, precompile does not consider terminal assignments and creates the deep memory with width equal to the total number of primary inputs. If the design has many primary inputs without terminal assignments, the emulator capacity or step count can be improved by giving any `terminalAssign` commands for the design, and then running `precompileOption` command with the `-readTerminalAssign` option.

-add instrumentKeepNets

Enhances the instrumentation of keepNets to enable forcing multiple signals at the same FCLK cycle, even when FCLK is running.

By default, keepNets instrumentation is switched off.

Compiling with the `instrumentKeepNets` option has the following advantages:

- If you force a bus while FCLK is running, by default different bits might be forced at different times differing by thousands of cycles. With the `instrumentKeepNets` option, all bits are forced at the same FCLK.
- For a design compiled in 1X mode (CAKE 1 or `ixcom +1xua`), if you force a net that has a shadow while FCLK is running, by default the net and shadow might be forced at different times differing by thousands of cycles.

In the waveform, instead of having a single transition, the signal will oscillate for thousands of cycles, and SDL will see this oscillation. With the `instrumentKeepNets` option, net and shadow are forced at the same FCLK and the signal has only a single transition.

Compiling with the `instrumentKeepNets` option has the disadvantage that the capacity and emulation speed costs for each keepNet are higher than the default costs. You might need to increase the probe capacity when adding instrumentation for keepNets. If you are using DYNP, see compilerOption `-add {numCapturedNetsPerDomain <n>}`. Or, if you are using FullVision mode, see compilerOption `-add {numExtraCapturedNetsPerDomain <n>}`.

In general, you should declare as keepNets only signals that you might need to force or deposit at run time.

-add noPowerIntent

Causes precompile to ignore the CPF or IEEE 1801 power intent and not to add any power intent related instrumentation logic or control signals into the netlist.

-add domainInterfaceDef [1.0 | 2.0 | 2.1 | UPF | 1801_2009 | 1801_2013]

Selects domain interface semantics based on the specified IEEE 1801 version.

UPF, 1801_2009, 1801_2013 are same as 1.0, 2.0, 2.1, respectively.

The default IEEE 1801 version is based on the `upf_version` command specified in the 1801 files. If the `upf_version` command is not in the 1801 files, the 2.0 (IEEE 1801-2009) version is used as default.

-add allowNoBreakViolations

Enables precompile to insert a loopbreak that violates `noBreakPath` user data constraints, when there is no other way to break a loop. By default, precompile fails if a loop cannot be broken without violating `noBreakPath` constraints.

-add VSSswitchable

Affects interpretation of IEEE 1801 simstate in UXE. By default, the simstate of a power domain or supply set is determined based on the state of its power net alone, that is, `NORMAL` simstate is assumed when power net is `FULL_ON`. The state of the ground net is ignored when determining simstate, that is, ground supply is assumed to be available always.

This approach is different from IEEE 1801 standard. The reason for the difference is that users who use Liberty cells assume the `VDD && !VSS` condition, which means `VDD==FULL_ON && VSS==OFF`, for the `NORMAL` simstate. On the other hand, from IEEE 1801 perspective, `NORMAL` simstate means `VDD==FULL_ON && VSS==FULL_ON`.

Because it might not be clear upfront whether a user intends to follow IEEE 1801 or Liberty, UXE ignores `VSS` by default.

The `VSSswitchable` option instructs compile flow to follow IEEE 1801, that is, equation `VDD==FULL_ON && VSS==FULL_ON` will determine `NORMAL` simstate.

-add breakTerminalTiming

Specifies that for loop-breaking, terminalTiming constraints should be considered.

In rare cases, the XEL `compile` command might fail with a message reporting loops. This happens because by default, for loop-breaking terminalTiming constraints are not considered. To avoid such failures, give any `delayBox` and `terminalTiming` commands for the design (except `terminalTiming -edgeInfo` commands), and then run the `precompileOption` command with the `-breakTerminalTiming` option.

-add lessProbes

The `lessProbes` option might be used to reduce the number of memory ports, in particular, when the number of memory ports exceeds the 512 limit.

Using the `lessProbes` option with the `precompileOption` command has the following advantages:

- Improves the step count of the compilation
- Avoids setting probes on the outputs of loadless MPR ports. Such ports might be optimized by the ET compiler.

After the precompile, the number of memory ports are added to the `system_top_et5mpart` file. You can verify the number of ports after the precompile as follows:

```
grep Stream dbFiles/system_top_et5mpart
17      1    488 node0.cpu.ape0.pd_ape_mon.ccaStream
17      1    488 node0.cpu.ape0.pd_ape_mon.ccbStream
17      1    488 node0.cpu.ape1.pd_ape_mon.ccaStream
17      1    488 node0.cpu.ape1.pd_ape_mon.ccbStream
17      1    488 node0.cpu.ape2.pd_ape_mon.ccaStream
17      1    488 node0.cpu.ape2.pd_ape_mon.ccbStream
17      1    488   node0.cpu.ape3.pd_ape_mon.ccaStream
```

-add allowClockConflicts

Enables conflicting `clockFrequency` and `clockAssign` userdata commands. In ICE mode, if a compile script includes more than one `clockFrequency` or `clockAssign` command for a net, it is called a ‘clock conflict’. By default, clock conflicts fail the precompile process. If the `allowClockConflicts` option is specified and a clock conflict occurs, one of the conflicting commands is honored and all others are ignored with a warning.

-add instrumentEdges

Enables the instrumentation at compile-time so that post run-time data can be collected to optimize the precompile. Use the following syntax to activate the instrumentation:

```
precompileOption -add instrumentEdges
```

Refer to *Activating Compile-time Instrumentation* section in *Compiling Designs for In-Circuit Emulation with xeCompile* chapter of the *UXE User Guide* for details.

-add createAllBoundaries

Displays isolation values of CPF on the boundary pins in the waveform.

By default, the isolation values of CPF are displayed as x in the waveform when the power domain of the driver of a signal is in the power off state.

-add dontMergeCommon

Disables the optimization that the precompile does to replace two or more gates (or storage elements) that have common inputs, by a single gate (or storage element). Whether or not the `dontMergeCommon` option is specified, the `et3compile` program always merges certain common expressions.

-add {1XrestrictPath <path>}

For optimizing compilation, some restrictions are necessary. For these restrictions, restriction data needs to be collected. After restriction data has been collected, optimized compilation can be performed. To activate the optimizations, use the following syntax prior to re-running precompile:

```
precompileOption -add {1XrestrictPath <path>}
```

Here, `<path>` specifies the previous run-time session directory where restriction data was collected. The `<path>` can be either an absolute or a relative path to the compilation directory.

Refer to *Compiling with Optimization* section in *Compiling Designs for In-Circuit Emulation with xeCompile* chapter of the *UXE User Guide* for details.

How to investigate precompile optimization

The back-end compiler (`et3compile`) issues a message telling the percent of the emulator's logic primitives used by a design. For example,

```
INFO (qt2dadb-1088): 29502 primitives used out of 1572864  
available (1.9% utilization)
```

Sometimes a design uses an unexpectedly small fraction of the emulator's primitives; a result that is "too good to be true". The reason for the low utilization is usually that a mistake in the user's design enables precompile to optimize out much of the design. For example, if a clock net is undriven or tied low, precompile might remove flip-flops clocked by that clock, and tie their output nets low.

To find the reason for unexpected optimization, use one of the following methods:

- Look for important clocks or reset nets reported in the `tmp/drc.rpt` file as having no source.
- Re-run precompile with a `clean.request` file containing the `-random` command (described below).

Precompile generates the `tmp/drc.rpt` file. For each undriven net, the `drc.rpt` file includes a message like

```
Warning #100003: Net ck_a0_und has no source.
```

Unfortunately, a design might have hundreds of sourceless nets, and it might be difficult to know which sourceless nets are important. In such a case, the best way to find the reason for unexpected optimization is to use a `clean.request` file containing the `-random` command, as follows:

1. Create file `clean.request` in the design directory, containing just one line such as the following:

```
-random 100 12345
```

In the above line, 12345 is a random seed (a positive integer number which you choose) and 100 is the number of randomly-chosen optimized-out flip-flop or latch output nets you want precompile to analyze.

2. Re-run precompile.

When the `-random` command is given in the `clean.request` file, precompile creates a list of all flip-flop or latch output nets that are tied high or low during optimization. It selects 100 of these nets at random, and for each selected net it prints in file `tmp/clean.dump` a trace-back report showing why the net was tied high or low. The format of the trace-back report is described in the sub-section below. At the end of the file, a summary such as the following is printed:

UXE Command Reference Manual

User Data Commands

```
{ Summary: the following nets are tied or undriven in the design  
and cause nets in clean.request to be tied by precompile. Each  
net is preceded by the number of clean.request nets it causes to  
be tied.  
 93 my_clk1  
  7 my_clk2  
  3 my_reset_net  
}
```

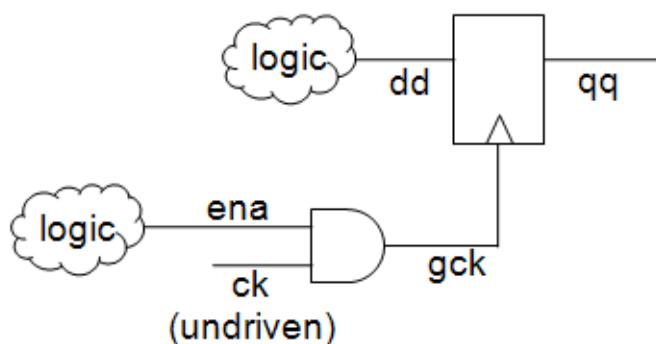
Each net $<C>$ listed in the summary is either undriven or tied in the user's design. Each net is preceded by a number telling how many of the 100 randomly-chosen nets were tied because net $<C>$ is undriven or tied. The nets with the largest numbers in the summary are likely to be the reason for the unexpected optimization.

You can use precompileOption -add debug to obtain a list of all nets that are tied high or low by precompile's constant propagation optimization.

Tracing the Optimization of Specific Nets

When the `clean.request` file contains the `-random` command, the `tmp/clean.dump` file includes trace-backs showing the reason each randomly selected net was optimized. You can also get trace-backs for other nets that were tied by optimization, by listing these nets in the `clean.request` file, one net per line.

For example, suppose the design has the logic shown below. Net `ck` is undriven, and is ANDed with net `ena` to generate net `gck`, which clocks a flip-flop whose output is `qq`.



If you run precompile with the `-add debug` option, the `clean.rpt` file includes the following lines:

```
net ck is tied LOW.  
net gck is tied LOW.  
net qq is tied LOW.
```

UXE Command Reference Manual

User Data Commands

To obtain a report showing why net `qq` was tied low, create a one-line file named `clean.request` in the design directory, with the following line:

```
qq
```

Re-run precompile. The resulting `tmp/clean.dump` file will include a trace like the following:

```
---
FDRSE u1
Q -> qq gnd START_TRACE
D <- dd
CK <- gck gnd TRACE
CE <-
R <-
S <-
---
A2 u2
O -> gck gnd
I1 <- ck gnd END_TRACE
I0 <- ena
```

The trace shows that `qq` is a flipflop (FDRSE) output. The flipflop's `Q` output is labeled `gnd` because its clock net `gck` is labeled `gnd`. Net `gck` is the output of an A2 (two-input AND) gate. Its output `gck` is `gnd` because one of its inputs (net `ck`) is `gnd`. The net `ck` is labeled `END_TRACE` because it is undriven in the netlist, so no further tracing is possible.

Nets in a trace report might have the following labels:

- `gnd`: the design logic is such that this net always has value 0. Therefore, precompile ties the net low.
- `pwr`: the design logic is such that this net always has value 1. Therefore, precompile ties the net high.
- `START_TRACE`: starting the trace of this net.
- `TRACE`: this input caused the output to be tied, and the input net will be traced further below
- `SKIP`: this input caused the output to be tied, but it will not be traced further because another input is traced instead.
- `ALREADY_TRADED`: this input caused the output to be tied, but the input will not be traced further below because it has been traced previously in the report.
- `END_TRACE`: this input caused the output to be tied, but the input is itself undriven or tied in the netlist, so no further tracing is possible.

Clocking Behavior

If `clockOption -add {technology CAKE 1}` is in effect, the following clocking default behavior applies (otherwise, it is ignored):

- If a clock has a period that is not an exact multiple (2X, 3X, 4X, and so on.) of the fastest clock's period, by default, all rising edges of this clock will coincide with rising edges of the fastest clock. This minimizes the capacity cost of 1X mode when the periods of slower clocks are not exact multiples of the fastest clock's period, but might cause the waveforms of such clocks to be less regular.

You can override this condition by using the `cake_like_2x` option, in which case, this clock will have rising edges coinciding with both rising and falling edges of the fastest clock.

Behavior of Latches and flip-flops

The default behavior of the flip-flops and latches is as follows:

- For flip-flops, when reset goes active, the output goes low in the same FCLK cycle. When reset goes inactive, if the clock goes high in the same FCLK cycle, the output stays low even if data input D was high.
- For latches, when reset goes active, the output goes low in the same FCLK cycle.

When the reset goes inactive, one of the following conditions can occur:

- If the enable goes active in the same FCLK cycle, the output gets the value that the data input D has in that cycle.
- If the enable goes inactive in the same FCLK cycle, the output stays low even if the data input D was high.

The `-latchResetAsFF` switch overrides this behavior.

Options Affecting Loop-breaking

The following options of the `precompileOption` command control loop-breaking. For certain designs, the emulation speed might improve if these options are used. For more information, refer to the *Loop-breaking* appendix of the *UXE User Guide*.

- `syncData`
- `asyncReset`
- `syncReset`

UXE Command Reference Manual

User Data Commands

- ❑ noLoopBreaks
- ❑ keepBreaks
- ❑ loopsDepth number
- ❑ dontMinimizeDataBreaks
- ❑ minimizeBreaks
- ❑ logAllLoops number
- ❑ moreExclusive
- ❑ clockDataLoops
- ❑ breakHalfCycle
- ❑ clockLoops101
- ❑ combinLoops101
- ❑ resetLoops101
- ❑ dataLoops101
- ❑ breakTerminalTiming
- ❑ cake_like_1x
- ❑ breakHalfCycleAll
- ❑ recognizeClockDivider
- ❑ moreClockSources
- ❑ allowDelayClkNets
- ❑ verify
- ❑ dontMinimizeBreaks
- ❑ clockLoops30
- ❑ breakHalfCycle
- ❑ minimizeBreaks
- ❑ moreExclusive

primaryClockSource

Specifies the name of the fastest clock. Clock sources are specified by net name.

Note: This command is only needed if you will be compiling in 1X mode.

Execute this command before running precompile.

This command returns the existing primary clock. Nothing is returned if there is no existing primary clock source.

Note: The `primaryClockSource` command enables you to specify the fastest CAKE clock without needing to check its frequency. If the frequency of the specified clock source is not the highest, the precompiler automatically raises the frequency as required.

The syntax for this command is:

```
primaryClockSource -add <primary_clock_source_name>  
primaryClockSource -rm <primary_clock_source_name>
```

-add <primary_clock_source_name>

Specifies fast clock.

-rm <primary_clock_source_name>

Removes clock as a primary clock, but retains the clock source.

probe

Note: The `probe` command discussed below is used at the compile time. The `probe` run-time command is discussed in [Chapter 13, “Run-time Commands.”](#)

Assigns probes to specified nets in the design, when compiling with `visionMode DYNP`. Primary I/O of the design are automatically probed. Compiles with `visionMode FV` ignore probes added using the `probe -add` command.

When using `visionMode DYNP`, you can probe any signal at compile time or at run time.

The disadvantage of probing at compile time is that probe resources are committed to the probed signals, and cannot be freed at run time to probe other signals.

Advantages of probing at compile time are:

- A compile-time probe uses only one probe channel, while a run-time probe uses an average of about two probe channels.
- If compilation succeeds, compile-time probes are guaranteed to be implemented. For probes added only at run time, probing might fail due to insufficient probe channels, or in rare cases, due to probe routing failure.

For both compile-time and run-time probes, you can ensure that a given number of probe channels are available using the compile-time command:

```
compilerOption -add {numCapturedNetsPerDomain <n>}
```

For more information, see the description of
[compilerOption -add {numCapturedNetsPerDomain <n>}](#).

Before using this command, execute the `design` command to specify the design library and cell name for the design.

This is an optional command. Run it at your discretion after the `design` command and before running precompile.

Note: The single clock source driving the Palladium hardware is implicit and cannot be probed.

This command enables wildcard characters (*) to be used in the `-iglob <inst_pattern>` and `-nglob <net_pattern>` fields. There is no default value for these fields; they must be specified explicitly.

Without parameters, this command lists all nets to which you assigned probes.

The syntax for this command is:

UXE Command Reference Manual

User Data Commands

```
probe -add <signal> | -add <net_name> |
      -addinst <instance_name> [-iregexp | -iglob <inst_pattern>] [-depth <n>] \
                                [-nregexp | -nglob <net_pattern>] <top_inst_name>
probe -addio <instance_name> ... |
      -addcone [-depth <n>] <signal> ... |
      -addpath [-depth <n>] <net1> <net2> [-glob | -regexp [<pattern>]]
probe -addNreport <net_name> ...
probe -clear
probe -rm <signal>... <net_name> ...
probe -rm {-glob | -regexp <pattern>}
```

-add <signal>

Adds a probe signal.

-add <net_name>

Adds a probe to the net.

A list of nets can be specified (for example `-add net1 net2 net3`).

Bus notation is supported. For example, if you want to probe bus signal `mem_address[1], ..., mem_address[8]`, you can enter the command as follows:

```
probe -add mem_address[1:8]
```

-addinst <instance_name> [-iregexp | -iglob <inst_pattern>] [-depth <n>]

[-nregexp | -nglob <net_pattern>] <top_inst_name>

Adds all nets and ports in the database hierarchy of the given instances to the trace signal list. If the `<instance_name>` is “.”, this specifies the top cell.

The `<instance_name>` serves as the root of the trace, which traces down to the sub-instances with names that match the `<pattern_expression>` specified with the `-iregexp` or `-iglob` argument. All of the nets and ports under the root instance, plus those matching sub-instances will be checked, if their name matches the specified `<pattern_expression>`. All of those matching nets and ports will be added as probe nets.

The nets and ports include the supply nets and supply ports. Similar to IES, the SST2 waveform displays the state and voltage for the supply nets and supply ports.

A maximum depth can be specified to restrict the search in the database, where *n* specifies the maximum levels of hierarchy. The default is one hierarchy level. A depth of zero means the whole hierarchy of an instance.

This option supports wildcards in names so that one command can generate multiple nets.

- `-iglob` specifies UNIX-style matching for instance names
- `-nglob` specifies UNIX-style match for net names

The `-ic` parameter is relevant only for `-iglob` and `-nglob`. Its use specifies no case sensitivity.

The `-depth <n>` argument specifies the levels for tracing back; its default value is 10 levels.

`-addio <instance_name>`

Adds all I/O nets for the given instances to the trace signal list.

`-addcone [-depth <n>] <signal>`

Adds all signals in the given nets' cone of logic to the trace signal list.

A maximum depth can be specified. The default is 5. A depth of zero means that the traversal of the logic cone stops at the first storage element output.

`-addpath [-depth <n>] <net1> <net2>`

Traces back from `<net1>` to `<net2>` and adds all nets on the path.

The `-depth <n>` argument specifies the levels for tracing back; its default value is 10 levels.

`-glob | -regexp [<pattern>]`

Returns a list of all nets in the trace upload list whose names match `<pattern>` using glob or regexp matching rules.

Glob expressions are the same as used for qualifying file names in a UNIX shell. For a description of glob patterns, refer to any Tcl/Tk manual for the string match command.

-addNreport <net_name>

Adds non-duplicated qualified nets to the user data and returns the names of these nets.

<net_name> can be any net, but only non-duplicated qualified nets will be added and returned.

-clear

Supported only in STB mode. Removes all probes.

-rm <signal>... <net_name> ...

Removes one or more probes from the net.

Note: Use the eval probe -rm [probe] command to easily remove all probes.

-rm {-glob | -regexp <pattern>}

Removes all nets in the trace upload list whose names match <pattern> using glob or regexp matching rules.

Examples

```
probe -add my_net1 my_net2 my_net3
```

The above command adds probes to the my_net1, my_net2, and my_net3 nets.

```
probe -rm my_net2 my_net3
```

The above command removes probes from the my_net2 and my_net3 nets.

```
probe -addinst -iglob *MyInst* -nglob *myNet -depth 3 ABC
```

In the above example, ABC is the instance name to start from and search downward to sublevel instances. The -iglob is for UNIX-style matching for instance names, and -nglob is for UNIX-style match for net names. These can be replaced by regular expression matches using -iregexp and/or -nregexp. If no glob or regexp options are specified, all sublevel nets will be selected. *MyInst* for -iglob matches (selects) any instance name containing the substring MyInst. In any selected instance, *MyNet matches any net name containing the substring MyNet at the end of the net name. The command will add all selected nets in all the selected instances.

shadowsData

Specifies the behavior of a particular clock, reset, or other design net in a 1X-mode compile. Specifying this behavior when compiling in 1X mode can reduce the number of shadow nets created, which can in turn improve capacity and emulation speed.

This command must be executed before the precompile command.

The syntax for this command is:

`shadowsData -add {<net> <keyword>}`

`shadowsData -rm <net>`

-add {<net> <keyword>}

Specifies the behavior of the specified *<net>*. The following *<keywords>* are supported:

- p=gnd
- p=pwr
- pos
- neg
- none
- both

For a detailed explanation of these keywords, refer to the *Modeling Clocks* chapter of the *UXE User Guide*.

-rm <net>

Removes previously specified shadowsData for the specified *<net>*.

Example

XEC> shadowsData -add {u1.u2.ck pos}

Specifies shadowsData keyword pos for net u1.u2.ck

XEC> shadowsData -rm u1.u2.ck

UXE Command Reference Manual

User Data Commands

Removes any shadowsData information for net `u1.u2.ck` specified in the previous `shadowsData` command. Removing the shadowsData information and recompiling restores the normal behavior for net `u1.u2.ck`.

stateRetentionControl

Identifies the flip-flop type that will be applied to the specified state retention rule. The supported flip-flop types are:

- Clock Low Retention Flip-Flop (CLRFF)
- Clock High Retention Flip-Flop (CHRFF)
- Clock Free Retention Flip-Flop (CFRFF)

By default, if this command is not executed, CFRFF is used. For implementing a behavior other than the default, issue this command before precompile during the compilation process.

The syntax for this command is:

```
stateRetentionControl -add {<stateRetentionRuleName> low | high}
```

```
stateRetentionControl -rm {<stateRetentionRuleName> low | high}
```

-add {<stateRetentionRuleName> low | high}

Specifies the *<stateRetentionRuleName>* for which the identified flip-flop type is to be implemented.

When the *<stateRetentionRuleName>* argument is set to:

low	Indicates that CLRFF retention cells will be used in the instances of the specified state retention rule.
high	Indicates that CHRFF retention cells will be used in the instances of the specified state retention rule.

-rm {<stateRetentionRuleName> low | high}

Specifies the *<stateRetentionRuleName>* for which the identified flip-flop type is **not** to be implemented.

symmetricConfiguration

Enables you to manage (that is create, append, modify, and clear) the list of hardware configurations targeted for symmetric compile. If no configuration is specified, the symmetric compilation function is disabled. If any configuration is in the list, symmetric compilation is enabled.

The base configuration subset for symmetric compile is derived from the [emulatorConfiguration](#) command. The base configuration subset is expanded to all available subsets in the list of configurations managed using the [symmetricConfiguration](#) command.

Also, ensure that the [design](#) command has been executed before the [symmetricConfiguration](#) command; else, the library will not be opened and an error message is returned.

The syntax for this command is:

```
symmetricConfiguration -add {host <host>} [{boards <boards>}]  
symmetricConfiguration -add {file <file>} [{boards <boards>}]  
symmetricConfiguration -rm {host <host>}  
symmetricConfiguration -rm {file <file>}  
symmetricConfiguration -clear
```

-add {host <host>} [{boards <boards>}]

Adds the specified *<host>* to the list of configurations targeted for symmetric compile.

Here,

- *<host>* option specifies the name of an emulator.
- *<boards>* is an optional list of domains. If this option is specified, the physical configuration on the specified *<host>* is limited to only the set of domains specified in the *<boards>*. If the option is not specified, the whole configuration on the specified *<host>* is targeted for symmetric compile.

-add {file <file>} [{boards <boards>}]

Adds the specified configuration *<file>* to the list of configurations targeted for symmetric compile.

Here,

- *<file>* option specifies the path to the configuration file.
- *<boards>* is an optional list of domains. If this option is specified, the physical configuration defined in the specified *<file>* is limited to only the set of domains specified in the *<boards>*. If the option is not specified, the whole configuration in the specified *<file>* is targeted for symmetric compile.

-rm {host <host>}

Removes the specified *<host>* from the list of configurations targeted for symmetric compile.

-rm {file <file>}

Removes the specified *<file>* from the list of configurations targeted for symmetric compile.

-clear

Removes all the hosts and configuration files from the list. Executing this option also disables the symmetric compile feature.

Example

The following command establishes the base configuration subset as two domains:

```
emulatorConfiguration -add {host neon17} {boards 0.0+0.1}
```

Executing the following set of `symmetricConfiguration` commands expands the base configuration subset to all available subsets within the current host (`neon17`) and to the two subsets on the host named `optra15`, that is, `(0.0+0.1)` and `(0.2+0.3)`:

```
symmetricConfiguration -clear  
symmetricConfiguration -add {host optra15} {boards 0.0+0.1+0.2+0.3}  
symmetricConfiguration -add {host neon17}
```

terminalAssign

Specifies the interface pins used for the target connection. The `terminalAssign` command assigns a primary (top level) terminal to a connector and pin on the target interface cable.

This command should be used before the [icePrepare](#) command.

The `terminalAssign` command can be used before or after the [precompile](#) command. However, if the `precompileOption -add readTerminalAssign` option is used, the terminal assignments must be specified before the `precompile` command. Otherwise, if the compile script uses global nets to extend the nets inside the design up to the top level, those will not be available until the `precompile` command is re-run.

Without parameters, this command returns a list of all terminal assignments specified in previous `terminalAssign` commands.

The syntax for this command is:

```
terminalAssign -add {<interface_name> [<interface_type>] <pin> <terminal_name>} |  
    -add {<interface_name> <connector> <pin> <terminal_name>} |  
    -add {<interface_name> EXC <pin> <terminal_name>} |  
    -add {<interface_name> MTIB-88 <connector> <pin> TERM | NET \  
          <terminal_name>} |  
    -add {<interface_name> DATA_CABLE <pin> TERM | NET \  
          <terminal_name>}  
  
terminalAssign -rm {<interface_name> [<interface_type>] <pin>} |  
    -rm {<interface_name> <connector> <pin>} |  
    -rm {<interface_name> EXC <pin>} |  
    -rm {<interface_name> MTIB-88 <connector> <pin>} |  
    -rm {<interface_name> DATA_CABLE <pin>}  
  
terminalAssign -dump <filename>  
terminalAssign -clear
```

-add {<interface_name> [<interface_type>] <pin> <terminal_name>}

Assigns the selected target `<connector>` and `<pin>` from the `<interface_name>` interface to the specified `<terminal_name>` terminal in the design. The `<interface_name>` and `<interface_type>` should be the same as specified in the `cableConnection` command.

<interface_type>

Following are the valid interface types and the corresponding parameters that need to be specified for appropriate terminal assignment / removal:

Supported <interface_type>	Parameters Required with the terminalAssign -add Command Option	Parameters Required with the terminalAssign -rm Command Option
HDDC	<interface_name>, <u><pin></u> , <u><terminal_name></u>	<interface_name>, <u><pin></u>
TIB-192	<interface_name>, <u><connector></u> , <u><pin></u> , <u><terminal_name></u>	<interface_name>, <u><connector></u> , <u><pin></u>
MTIB-88	<interface_name>, <u><interface_type></u> , <u><connector></u> , <u><pin></u> , TERM NET, <u><terminal_name></u>	<interface_name>, <u><interface_type></u> , <u><connector></u> , <u><pin></u>
DATA_CABLE	<interface_name>, <u><interface_type></u> , <u><pin></u> , TERM NET, <u><terminal_name></u>	<interface_name>, <u><interface_type></u> , <u><pin></u>
EXC	<interface_name>, <u><interface_type></u> , <u><pin></u> , <u><terminal_name></u>	<interface_name>, <u><interface_type></u> , <u><pin></u>

For detailed description of each interface type, refer to the <cable_label> option of the cableConnection command.

<connector>

This option is required only for the following interface types:

- MTIB-88: The connector must have a value of P2, P3, or P4.
- TIB-192: The connector must have a value of P2, P3, P4, P5, P6, P7, or P8.

UXE Command Reference Manual
User Data Commands

<pin>

Specifies the pin identification on the target side of the connector.

Interface Type	Valid Pin Values (Between)	Invalid Pin Values
HDDC	<A-F><1-35>	<A-F>9, <A-F>18, and <A-F>27
MTIB-88	P2 and P3 connectors: 1-59, odd numbers only P4 connector: 5-59, odd numbers only	Even numbers between 1-59 Even numbers between 5-59
TIB-192	P2 connector: 1-59, odd numbers only	Even numbers between 1-59
	P3 connector: 1-59, odd numbers only	Even numbers between 1-59
	P4 connector: 5-59, odd numbers only	Even numbers between 5-59
	P5 connector: 27-59, odd numbers only	Even numbers between 27-59, and 33
	P6 connector: 1-59, odd numbers only	Even numbers between 1-59
	P7 connector: 1-59, odd numbers only	Even numbers between 1-59
	P8 connector: 5-59, odd numbers only	Even numbers between 5-59
DATA_CABLE	1-98	7, 27, 37, 45, 49, 50, 51, 57, 77, and 95
EXC	1-15, odd numbers only	Even numbers between 1-15

Note: A TIB-192 type interface acts like two MTIB-88 interfaces. For TIB-192 interface type, the connectors are grouped in two sets, (P2, P3, P4) and (P6, P7, P8). The connectors of each TIB-192 interface set are equivalent to the corresponding connectors of the MTIB-88 interface, that is, (P2 ~ P6 ~ MTIB-88's P2; P3 ~ P7 ~ MTIB-88's P3; P4 ~ P8 ~ MTIB-88's P4).

TERM | NET

This option is required only when target interface type DATA_CABLE or MTIB-88 is being used. NET is used for component adapter terminals. TERM is used for all other terminals.

<terminal_name>

Specifies the name of the top level terminal, component adapter I/O, or scoped net. When the design delimiter is underscore (_), the component adapter I/O must be specified by pin names using a colon (:) as a hierarchical delimiter.

To enable the external target system to detect the occurrence of a trigger, the internal trigger instrumentation signal should be assigned to a target interface pin. In this case, it must use TARGET_TRIGGER_SIGNAL as the <terminal_name>.

To enable the external target system to detect the suspension of design clocks, the internal suspend instrumentation signal should be assigned to a target interface pin. In this case, it must use TARGET_SUSPEND_SIGNAL as the <terminal_name>.

-rm {<interface_name> [<interface_type>] <pin>}

Removes the current connector and pin assignment from the specified <interface_name>. Based on the selected interface type, the set of parameters required with the terminalAssign -rm command differ. For details, refer to <interface_type>.

Note: The <interface_name> and <interface_type> should be the same as specified in the cableConnection command.

-dump <filename>

Saves all existing terminal assignments as selected in previous terminalAssign commands as XEL commands into the specified file.

-clear

Removes all the existing terminal assignments.

Examples

- To assign net1 to pin A1 of the interface_type HDDC with interface_name mycable1, use:

```
terminalAssign -add {myCable1 A1 net1}
```

- To assign net2 to pin 1 of connector P2 of the interface_type TIB-192 with interface_name mycable2, use:

```
terminalAssign -add {myCable2 P2 1 net2}
```

- To assign net3 to pin 17 of connector P2 of the interface_type MTIB-88(legacy) with interface_name mycable3, use:

```
terminalAssign -add {myCable3 MTIB-88 P2 17 TERM net3}
```

- To assign net4 to pin 17 of the interface_type DATA_CABLE(legacy) with interface_name mycable4, use:

```
terminalAssign -add {myCable4 DATA_CABLE 17 TERM net4}
```

- To assign net5 to pin 1 of the interface_type EXC (LTA_C/LTA_D) with interface_name mycable5, use:

```
terminalAssign -add {myCable5 EXC 1 net5}
```

- To assign more than one net (netA, netB, and netC) to the same target cable pin 1 of connector P2, that has the interface_type TIB-192 and interface_name myCable6, use:

```
terminalAssign -add {myCable6 P2 1 netA}
```

```
terminalAssign -add {myCable6 P2 1 netB}
```

```
terminalAssign -add {myCable6 P2 1 netC}
```

- To remove all terminalAssign commands, use:

```
terminalAssign -clear
```

terminalOutputSync

Specifies a group of primary output signals that are to be synchronized with each other. There can be multiple terminal output groups. A group that contains only one terminal has no effect on the compile process.

The last `terminalOutputSync` command overrides the previous settings. For example, if a terminal is assigned to one group, it might be re-assigned to another group.

Run this command before or after `precompile`.

Without any parameters, this command returns the current list of groups and terminals in each group.

The syntax for this command is:

```
terminalOutputSync -add {<terminal> <group_number>}  
terminalOutputSync -clear
```

-add {<terminal> <group_number>}

Adds the specified `<terminal>` signal to the specific `<group_number>`.

<terminal>

Specifies the primary output signal to be synchronized with all other primary output signals in the group indicated by `<group_number>`. The signal `<terminal>` has to have a valid terminal assignment.

<group_number>

Specifies the number of the group with which the signals have to be associated so that all signals of that group can be synchronized together. The group number must be a positive integer. Otherwise, an error is generated and the command is ignored.

Each statement adds only one signal. So, if you have `n` signals to be synchronized, you will need `n` statements with the same group number.

-clear

Deletes all group settings.

Examples

To create a group that includes two signals named `my_term_1` and `my_term_2`, and has `<group_number>` equal to 1, specify:

```
terminalOutputSync -add {my_term_1 1}  
terminalOutputSync -add {my_term_2 1}
```

When the above commands are executed, the `my_term_1` and `my_term_2` signals will be synchronized as a group. Note that the placement of these commands in the compile script does not specify the exact step where the group is to be scheduled. It only indicates that the output of both these signals will be synchronized at the same step.

terminalTiming

For a top-level I/O terminal which is connected to a target system, the `terminalTiming` command specifies a timing constraint for the terminal, or specifies the behavior of the terminal as a pulse or multi-pulse output.

Without parameters, this command returns the current timing constraints for all terminals. Specifying only `terminalTiming <terminalName>` returns the current constraint for the specified terminal.

The syntax for this command is:

```
terminalTiming -add {<terminal_name> <constraint>}  
terminalTiming -add {<terminal_name> <constraint_out> <constraint_in> <constraint_en>}  
terminalTiming -add {TARGET_TRIGGER_SIGNAL <constraint>}  
terminalTiming -add {TARGET_SUSPEND_SIGNAL <constraint>}  
terminalTiming -add {ALL_OTHER_OUTPUTS <constraint>}  
terminalTiming -add {<terminal_name> <constraint> -pulse <pulse_info>}  
terminalTiming -add {<terminal_name> -edgeInfo <initial_value> <hold_time> <edge_list>}  
terminalTiming -rm {<terminal_name>}  
terminalTiming {<terminal_name>}  
terminalTiming -dump <file_name>
```

-add {<terminal_name> <constraint>}

Specifies a timing constraint for a unidirectional terminal. The `<constraint>` can be one of the following:

- An `<integer>` representing an emulation step number or TIF number. If the compilerOption `ioDelayUnit` has not been specified, the delay value is interpreted in steps.
- The name of another terminal.
- The delay box output signal name.
- An * (asterisk) character meaning “no constraint”.

A `terminalTiming` command (except `terminalTiming -edgeInfo`) specifies the earliest step number at which the compiler is allowed to schedule an input or output terminal.

The specified terminal will be scheduled at or after the step number, TIF number, or signal specified by the constraint. After compiling, the file `dbFiles/<top>.ttime` shows the TIF number at which each terminal is scheduled. To convert the TIF number to step number, multiply by 8. If the terminal is an input, the value is sampled at the interface to the target at the scheduled TIF. If the terminal is an output, the value changes at the interface to the target at the scheduled TIF.

**-add {<terminal_name> <constraint_out> <constraint_in>
<constraint_en>}**

Specifies timing constraints for the components of a bidirectional signal. Up to three constraints can be specified. The first constraint is for the output component, the second for the input component, and the third for the enable component.

-add {TARGET_TRIGGER_SIGNAL <constraint>}

Specifies timing constraints for the internal trigger instrumentation signal assigned to the target interface.

-add {TARGET_SUSPEND_SIGNAL <constraint>}

Specifies timing constraints for the internal suspend instrumentation signal assigned to the target interface.

-add {ALL_OTHER_OUTPUTS <constraint>}

In this option, `<constraint>` must be a delay box output signal name.

This option specifies that all unidirectional outputs, and output and enable components of bidirectional terminals, other than those used in the `delayBox` commands, must be scheduled at or after the step at which the given delay box output is scheduled.

-add {<terminal_name> <constraint> -pulse <pulse_info>}

Specifies the behavior of a given unidirectional output terminal as a pulse output.

A pulse output can be positive or negative. A positive pulse output starts out low in each FCLK cycle. At a certain step (the same in every FCLK cycle), it is updated to the value driven by

the design. Then, at a later step, it goes low again. A negative pulse is the same, except that it has a high value instead of a low value at the beginning and end of the FCLK cycle.

The waveform display for a pulse output signal shows the value driven by the design (a single value in each FCLK). However, the signal sent to the target shows the pulse behavior described above.

- *<terminal_name>* is the name of an output terminal of the design.
- *<constraint>* is one of the four forms of constraint described earlier. The pulse begins at or after the step number or signal specified by the constraint.
- *<pulse_info>* is one of the following:
 - +*<N>*: Here, *<N>* is a positive integer number. This option specifies a positive pulse with the width of at least *<N>* steps.
 - -*<N>*: Here, *<N>* is a positive integer number. This option specifies a negative pulse with the width of at least *<N>* steps.

-add {<terminal_name> -edgeInfo <initial_value> <hold_time> <edge_list>}

Specifies the behavior of a given unidirectional output terminal as a multipulse output. The waveform display for a multipulse output signal shows the value driven by the design. However, the signal sent to the target system is independent of the value driven by the design, and depends only on the *-edgeInfo* parameters. The signal sent to the target can change any number of times per FCLK (up to once per step) and show the same behavior in every FCLK cycle.

- *<terminal_name>* is the name of an output terminal of the design.
- *<initial_value>* is the value of the signal at the beginning of the FCLK cycle, before the first edge. It must be specified as 0 or 1.
- *<hold_time>* is the minimum number of steps allowed between the last edge and the end of the FCLK cycle. The compiler will extend the FCLK cycle, if necessary, to meet this requirement. This parameter must be a positive number.
- *<edge_list>* is a list of step numbers at which (rising or falling) edges will occur. The list must be monotonically increasing and the length of the list must be even.

-rm {<terminal_name>}

Removes the terminal timing information for the specified terminal.

{<terminal_name>}

Returns the terminal timing information for the specified terminal.

-dump <file_name>

Dumps the terminal timing information for all signals as the `terminalTiming` commands into the specified file. If the file is not specified, the `terminalTiming` commands will be printed on stdout (that is, the screen).

Additional Information

A `terminalTiming` command is only meaningful for a terminal which is also assigned to a target cable pin using the `terminalAssign` command.

Usually a terminal has the same name as the signal (net) that connects to it. If not, the name of the terminal, not the net, must be used in this command.

A `terminalTiming -add` command overrides any previous `terminalTiming` command for the same terminal, replacing any existing specification with a new one.

For Palladium XP hardware platform, the total number of pulse and multipulse outputs on a specified High Density Data Cable (HDDC) cannot exceed 4.

Examples

Design with clock outputs (1)

The most common in-circuit setup is one in which the emulator supplies one or more clocks to the target system. Often, a design has unidirectional outputs and bidirectional signals that change on rising edge of one of these clocks, and these signals are sampled by the target on the rising edge of the same clock. Normally, for correct operation, at the clock rising edge, the target must sample the “previous” values of the output signals (before they change).

In such a case, you are responsible to use `delayBox` and `terminalTiming` commands to ensure that non-clock outputs change at a later step in the FCLK cycle than clock outputs. This ensures the target will sample the non-clock outputs before they change. A margin of 4 steps in case of Palladium XP should be sufficient unless the target introduces an unusual skew.

In this small example the design is assumed to have two output clocks. `aclk` is a unidirectional output, and `bclk` is a tristate output. `out1`, `out2` are assumed to be all the

UXE Command Reference Manual

User Data Commands

unidirectional non-clock outputs of the design, and bi1, bi2 are all the bidirectional non-clock signals. The following XEL commands ensure that non-clock outputs are scheduled at least 20 steps after the clock outputs.

```
delayBox -add {afterclocks 20 aclk bclk:xeEnable bclk:xeOutput}
terminalTiming -add {out1 afterclocks}
terminalTiming -add {out2 afterclocks}
terminalTiming -add {bi1 afterclocks * afterclocks}
terminalTiming -add {bi2 afterclocks * afterclocks}
```

The delayBox command defines a delayBox output signal called afterclocks, which is scheduled at least 20 steps after the latest signal of aclk, bclk:xeEnable and bclk:xeOutput. The terminalTiming commands ensure that the output signals (and output and enable components of the bidirectional signals) are scheduled at or after the step when afterclocks is scheduled.

Design with clock outputs (2)

This example uses the same small design as in the previous example. The following XEL commands use the ALL_OTHER_OUTPUTS keyword to achieve the same result as the terminalTiming commands of the previous example.

```
delayBox -add {afterclocks 20 aclk bclk:xeEnable bclk:xeOutput}
terminalTiming -add {ALL_OTHER_OUTPUTS afterclocks}
```

Pulse outputs

If A and B are unidirectional outputs, the following commands specify A as a positive pulse output, and B as a negative pulse output. The compiler will schedule both pulses to start at or after step 100, and to have a width at least 30 steps.

```
terminalTiming -add {A 100 -pulse +30}
terminalTiming -add {B 100 -pulse -30}
```

Multipulse output

If A is a unidirectional output, the following command specifies it as a multipulse output. The initial value of the signal (at the beginning of the FCLK cycle) is 0. The signal has a rising edge at step 100, falling edge at step 120, rising edge at step 140, and falling edge at step 160. The final value (0) will be held until the beginning of the next FCLK cycle. Since the last edge is at step 160 and the specified hold-time is 40, the compiler must schedule the FCLK cycle to be at least 200 steps long.

```
terminalTiming -add {A -edgeInfo 0 40 100 120 140 160}
```

Avoiding output-to-input delay

When a design connects to a target system, a transition of a design output can cause transition of a design input. This can happen in two ways:

1. A design output transition (usually a clock output rising edge) causes the target to change the value of a design input.
2. When the design starts or stops driving a bidirectional signal, or changes the value it is driving on the bidirectional signal, the transition of the bidirectional signal's output or enable component causes a transition of the input component.

The compiler generally tries to schedule design inputs as early in the FCLK cycle as it can. Since signals within the design depend on the inputs, scheduling inputs early gives an optimal step count. When not constrained otherwise, the compiler usually schedules design inputs early in the FCLK cycle, and design outputs later in the FCLK cycle. When inputs and outputs are scheduled this way, and a transition of a design output causes a transition of a design input, the design will see a one-FCLK delay between the output transition and the resulting input transition. The reason for the delay is that if the output transition occurs at a late step of FCLK cycle N, the emulator has already sampled the value of the input signal at an earlier step of cycle N, and so the emulator does not see the transition on the input until cycle N+1. This one-FCLK delay is called “output-to-input delay”.

For most designs, the logic on both sides (that is, design and target) of a target interface form a synchronous system clocked by a single clock, which is an output of the emulated design. If the period of this design clock is one FCLK (as might be the case for a design compiled in CAKE 1X mode), usually a one-FCLK output-to-input delay from a clock output of the emulator to a design input will cause the design to work incorrectly. The failure happens because with such delay, target state transitions caused by clock rising edge at cycle N are not seen by the design until the clock rising edge at cycle N+2.

If the period of the design clock used by the target interface is two FCLKs (as might be the case for a design compiled in CAKE 2X mode), usually the design will tolerate a one-FCLK delay from clock output to design input. However, such delay might not always be tolerated. For example, it would not be tolerated if the target system state transition caused by a clock rising edge must be seen by flip-flops in the design that are clocked on the next falling edge.

Constraining a design input signal to be scheduled late in the FCLK cycle can cause an increase in the step count with which the design is compiled (therefore, decreasing the emulation speed), if this signal is on the design's critical path. You can use the post-scheduling critical path report produced by the `getCriticalPathDesignNetNames` command to know if a particular signal is on the critical path.

If you want to avoid a one-FCLK delay from a clock output to a design input, you must constrain the design input to be scheduled at a later step in the FCLK cycle than the clock

output is scheduled. You must add an additional margin to enable the clock output transition to reach the target, the target to react, and the input transition to reach the emulator before the step at which the input is sampled.

For example, suppose the design has unidirectional clock output `clk`, unidirectional input `in1`, unidirectional output `out1`, and bidirectional signal `bi1`. The following commands constrain output `out1` and the output and enable components of bidirectional `bi1` to be scheduled at least 12 steps after clock output `clk`:

```
delayBox -add {after_clk 12 clk}
delayBox -add {long_after_clk 30 clk}
terminalTiming -add {out1 after_clk}
terminalTiming -add {in1 long_after_clk}
terminalTiming -add {bi1 after_clk long_after_clk after_clk}
```

The above commands also constrain input `in1` and the input component of bidirectional `bi1` to be scheduled at least 30 steps after the clock output `clk`.

If the output `clk` is a tristate output, the `delayBox` commands must be changed to the following:

```
delayBox -add {after_clk 12 clk:xeOutput clk:xeEnable}
delayBox -add {long_after_clk 30 clk:xeOutput clk:xeEnable}
```

If the output `clk` is implemented as a multipulse output with rising edge at step 0, the two `delayBox` commands should be replaced by the following:

```
terminalTiming -add {clk -edgeInfo 0 25 0 100}
delayBox -add {after_clk 12}
delayBox -add {long_after_clk 30}
```

In this example, the `delayBoxes` `after_clk` and `long_after_clk` have no inputs, and the `delayBox` outputs are scheduled at or after steps 12 and 30, respectively.

Avoiding output-to-input delay for bidirectional signals

A target interface running in 1X mode almost never tolerates a one-FCLK delay from the clock output to the inputs. However, target interfaces running in 1X mode often do tolerate a one-FCLK delay from output and enable components of a bidirectional signal to the input component of the same signal. Two scenarios where such delay might not be tolerated are:

- There is a path from a flip-flop output in the design, out to a bidirectional primary I/O signal, and then back from the same bidirectional signal to another flip-flop in the design, and the design function requires a transition on this path to propagate within one FCLK cycle. In this scenario, if the input component of the bidirectional signal is scheduled earlier than the output component, there will be a one-FCLK delay from output

component to input component, and the requirement of the path to propagate in one cycle will not be met.

- Bus-turnaround: The design drives a value on a bidirectional signal (or bus) in a certain FCLK N, and needs to receive a value driven by the target system in the very next FCLK cycle N+1. In this scenario, if the input component of the bidirectional is scheduled earlier than the enable component, at cycle N+1, the design does not stop driving until some step later than the step at which the emulator samples the input component. Thus, in cycle N+1, the emulator will fail to receive the value driven by the target, since the emulator is still driving the signal in the early part of cycle N+1.

Even when a design's function tolerates one-FCLK delay from the output and enable components of a bidirectional signal to the input component, this delay can be confusing to a user viewing waveforms of the primary I/O signals. Suppose the design is driving bidirectional signal B, and the driven value changes at FCLK cycle N. The target will see the signal change at cycle N, but the waveform of B (which shows the input component) will not change until cycle N+1. This is especially confusing when waveforms of unidirectional outputs and bidirectional primary I/O signals are viewed together. The waveforms of the bidirectional signals are delayed, but those of the unidirectional signals are not.

If you want to avoid a one-FCLK delay from the output and enable components of a bidirectional signal to the input component, you must constrain the input component to be scheduled at a later step in the FCLK cycle than the output and enable components are scheduled.

The following example is the same as the first example of the previous section, except that it additionally constrains the input component of bidirectional signal `bi1` to be scheduled at least 30 steps later than the output and enable components.

```
delayBox -add {after_clk 12 clk}
delayBox -add {long_after_clk 30 clk}
delayBox -add {bi1_constr1 30 bi1:xeOutput bi1:xeEnable}
delayBox -add {bi1_constr2 0 bi1_constr1 long_after_clk}
terminalTiming -add {out1 after_clk}
terminalTiming -add {in1 long_after_clk}
terminalTiming -add {bi1 after_clk bi1_constr2 after_clk}
```

Design with a clock input asynchronous with FCLK

In an in-circuit setup, the target can generate one or more clocks from sources external to the emulator. These clocks can be asynchronous with each other and with FCLK. The clocks can be inputs to the design, connected through target interface cables along with other design I/O.

UXE Command Reference Manual

User Data Commands

If your target supplies clock inputs to the design, asynchronous with FCLK, take care of the following:

- You must ensure that each clock has a frequency (at least slightly) less than half that of FCLK. This ensures the emulator samples the clock at least once during each high and low phase of each clock.
- If the target supplies non-clock inputs to the design that change on the rising edge of a specific clock asynchronous with FCLK, and the design uses the rising edge of this clock to sample the inputs, correct operation requires that the design sample the “previous” values of the input signals (before they change) at the clock rising edge. In such a case, you must use `delayBox` and `terminalTiming` commands to ensure that the emulator samples the clock input later in the FCLK cycle than the nonclock inputs.

Two different types of sampling are being discussed here:

- The design samples its inputs on rising edge of the clock
- The emulator samples each input at a certain step.

By constraining the emulator to sample the clock input at a later step than the nonclock inputs, you ensure that a specific rising edge of the clock is seen by the emulator in the same FCLK cycle (or one FCLK cycle earlier) than the corresponding nonclock input transitions are seen. This, in turn, ensures that design flip-flops clocked by this clock will store the “previous” value of the nonclock inputs (before they change). In Palladium XP, when a flip-flop’s clock and data inputs change at the same FCLK cycle, the flip-flop stores the data input from the previous FCLK cycle.

- If the design samples an input on both edges of a given clock which is asynchronous with FCLK, this clock must have frequency less than one-fourth that of FCLK.
- If two externally generated clocks are synchronous with each other, and some of their edges must be seen by the design in the same FCLK cycle, FCLK must be synchronous with these clocks. Otherwise, coincident edges of two clocks could be sampled by the emulator in different FCLK cycles. In such a case, you have the following options: (1) generate the clocks using external logic (for example, an FPGA) that runs off a multipulse output of the emulator, or (2) input only one of the clocks to the emulator, and derive the other clock from the first, inside the design.

In the following example we assume the design has two input clocks, `aclk` and `bclk`, asynchronous with FCLK. `aclk` is a unidirectional input, and `bclk` is bidirectional, but during actual operation, it is always an input signal. We assume `in1` and `in2` are nonclock inputs, and `bi1` and `bi2` are bidirectional signals, which change on rising edge of one of these clocks, and that the design samples these signals using flip-flops clocked by rising edge of the same clock. The following XEL commands ensure that the clock inputs are sampled at

UXE Command Reference Manual

User Data Commands

least 20 steps later in the FCLK cycle than the nonclock inputs (and input components of the bidirectional signals).

```
delayBox -add {afterinputs 20 in1 in2 bi1:xeInput bi2:xeInput}
terminalTiming -add {aclk afterinputs}
terminalTiming -add {bclk * afterinputs *}
```

The `delayBox` command defines a `delayBox` output signal `afterinputs`, which is scheduled at least 20 steps after the latest signals of `in1`, `in2`, `bi1:xeInput` and `bi2:xeInput`. The `terminalTiming` commands ensure that the clock inputs (`aclk` and the input component of `bclk`) are sampled at the same or later step than when `afterinputs` is scheduled.

terminalWeakDrive

Directs the compiler to provide a pullup, pulldown, or neither on a top-level tristate or bidirectional terminal that has a terminal assignment. The pullup or pulldown determines the value of the terminal when it is otherwise undriven. The `terminalWeakDrive` command should be used only for terminals with terminal assignments.

For tristate or bidirectional terminals with terminal assignments, if no pullup or pulldown is connected in the netlist, the compiler by default connects a pullup. You can override the default for all terminals using the `primaryIOPulldown` or `primaryIOExternalDrive` options of the `precompileOption` command. You can (with higher priority) override the default for an individual terminal using the `terminalWeakDrive` command.

Without any parameters, this command returns a list of the currently assigned drive values.

The syntax for this command is:

```
terminalWeakDrive -add {<terminal_name> <weakDriveValue>}...  
terminalWeakDrive -rm {<terminal_name>}  
terminalWeakDrive -clear
```

-add {<terminal_name> <weakDriveValue>}

Adds a top terminal tristate driver. Multiple `{<terminal_name> <weakDriveValue>}` pairs can be assigned in one `-add` statement.

<terminal_name>

A top terminal name. This can be the name of a CA/IP terminal, for example: `i1:i2:i3:t3`.

<weakDriveValue>

The `<weakDriveValue>` can be any one of the following:

- `pullUp`
- `pullDown`
- `retainState`
- `externalDriver`

UXE Command Reference Manual

User Data Commands

The values `retainState` and `externalDriver` are synonyms, and they instruct the compiler not to connect any pullup or pulldown to the given terminal. The `retainState` value does NOT mean that the compiler provides retain-state functionality on the terminal.

Note: Specifying a value other than the ones listed above, including the shortest unique strings, generates an error.

-rm {<terminal_name>}

Removes the top terminal tristate driver.

-clear

Deletes the entire terminalWeakDrive data.

tieNet

Ties the selected nets to either power or ground.

If you execute this command without parameters, it returns a list of the currently-specified tieNet signals. For example:

```
{A 0} {B 1}
```

The syntax for this command is:

```
tieNet -add {<net> 0 | 1} ...
```

```
tieNet -addNreport <net>
```

```
tieNet -rm {<net>} ...
```

-add {<net> 0 | 1}

Adds the specified net(s) to ground or power. If the value 0 is assigned, the selected net is tied to GROUND. If the value 1 is assigned, the selected net is tied to VCC (power).

Multiple nets can be assigned in one -add statement.

-addNreport <net>

Adds non-duplicated qualified nets to the user data and ignore the other nets.

-rm {<net>}

Removes the specified net(s). In one -add statement, multiple nets can be specified for deletion.

Examples

```
tieNet -add {my_net1 1} {my_net2 0}
```

Ties the my_net1 net to VCC, and the my_net2 net to GROUND.

```
tieNet -rm {my_net1} {my_net2}
```

Unties the my_net1 and my_net2 nets from power or ground.

ttcClear

Clears all information related to terminal timing. This includes all data entered with the terminalTiming, delayBox, and delayBoxInput commands.

The syntax for this command is:

```
ttcClear
```

userData

Note: The `userData` command discussed below is used at the compile time. The `userData` run-time command is discussed in [Chapter 13, “Run-time Commands.”](#)

Loads user data into the design database, or dumps user data to either standard output or a file.

Before using this command, execute `design` to specify the design library and cell name. The command must be executed before `precompile`.

User data can be dumped any time after the data has been defined.

This is an optional command. Run it at your discretion after `design` to load and define your user data.

The syntax for this command is:

```
userData -load <file> [-scope <instance_path>]  
userData -load -noQualify <file>  
userData -dump [<type> | all] <file>  
userData -get [-max <n>] [<type> [option]]
```

-load <file>

Loads user data from the specified file.

-scope <instance_path>

Prepends the specified `<instance_path>` to the instances, pins, and nets while loading the following types of user data:

- andMultiDrv
- breakNet
- breakNetHalfCycle
- breakPin
- breakPinHalfCycle
- clockAssign

UXE Command Reference Manual

User Data Commands

- globalNet
- instanceStub
- ipInstAssign
- keepNet
- memoryTransform
- memWritethru
- orMultiDrv
- probe
- shadowsData
- tieNet

Note: All other user's data will ignore the `-scope` option.

-load -noQualify <file>

Enables skipping of data qualification to improve performance.

Keep the following points in mind when using the `-noQualify` option:

- Only correct userData should be loaded. If incorrect data is loaded, the precompile and compile steps will not be performed correctly and a crash is possible.
- Each bit of a Vector must be specified individually for the `-noQualify` load. When doing a normal load, the entire vector is accepted and converted to individual bits by loading of the userData. However with the `-noQualify` option, the vector will not be segregated and will, therefore, be considered as incorrect data.



Caution

Complete vectors are also dumped as complete vectors. These complete vectors are not reloaded correctly when the -noQualify option is used at the time of reloading.

The recommended use model when loading the userData is:

1. Load the data and enable the loader to verify the data (that is, do not use the `-noQualify` option).
2. Dump the valid userData to a file (by using the `-dump` option).

3. Use the `-noQualify` option when reloading the `userData`.

-dump [<type> | all] <file>

Dumps user data from the current XEL session to the specified file. This command does not dump user data from separate XEL sessions that are running at the same time. For example, if you run this command from a XEL command shell, it does not dump user data created in the Palladium XP Compiler main window.

If you do not specify a file name, this parameter dumps user data to standard output.

The `<type>` variable specifies a type of user data. It is the name of the XEL command used to establish the user data type.

For example. a type might be a keepNet or tieNet.

The `all` keyword selects all data types.

The `<file>` variable loads data from, or dumps data to a specified file. If you do not include this parameter with the `-dump` parameter, this command dumps the user data to the standard output (`stdout`).

Examples

- Consider that you have a `userData` file named `myUserDataFile` file containing the following content:

```
keepNet
  data[1]
  data[2]
  data[3]
  counter[1]
  counter[2]
end
```

Load the source data from the `myUserDataFile` file to the standard output using the following command:

```
userData -load myUserDataFile
```

Consequently, the following source data will be loaded in the `keepNet` `userData` category:

```
data[1]
data[2]
```

```
data[3]
counter[1]
counter[2]
```

- Load source data from the `myUserDataFile` file to the standard output and prepend `m0` to all nets using the following command:

```
userData -load myUserDataFile -scope m0
```

Consequently, the following source data will be loaded in the `keepNet` `userData` category:

```
m0.data[1]
m0.data[2]
m0.data[3]
m0.counter[1]
m0.counter[2]
```

- Dump all user data to the standard output using the following command:

```
userData -dump all
```

- Dump all user data to the `myUserDataFile` file using the following command:

```
userData -dump all myUserDataFile
```

-get [-max <n>] [<type> [option]]

Gets the user data for the specified type from the design and prints it to stdout. This command works only in the ICE mode.

- The `-max <n>` option specifies the maximum limit of user data to get. The default value of `<n>` is to return up to maximum 20 lines for every data type. When `<n>` is specified as 0, all data is returned.
- The `<type>` option specifies the type of user data. It is the name of the XEL command used to establish the user data type.
- When [option] is also specified, the value for only the specified option is returned.

For example, to get the user data specific to the `compilerOption` XEL command, specify the following:

```
userData -get compilerOption
```

The command will return the user data, as follows:

```
{XPIIcapacityFeatures 7} {constrainTBAbuffersLocation off} {cpfLogBufferDepth
16K} {enableMultiSampledIO 0} {infiniTrace on} {ixcomHotSwap off}
{memoryRetiming off} {mode ice} {numExtraCapturedNetsPerDomain 0}
```

UXE Command Reference Manual

User Data Commands

```
{optimizationAlgo modern} {partitionerAlgo auto}
{pathBreakerConsiderAsyncSetReset off} {pathBreakerLatchInputDelay on}
{pathBreakerMode off} {pathBreakerOnClockFreq off} {sdlInstances 1}
{sdlTraceDepth 16K} {symmetricPreserveCables on} {symmetricPreserveMemCards
on} {visionMode FV}
```

For more information and examples, refer to the description of the `userData -get` command in the *Run-Time Commands* chapter.

IP Commands

This chapter describes XEL commands used for managing Intellectual Property (IP). IP-related commands are a specific form of User Data commands used only in cases where the user has incorporated Intellectual Property into their design. They are used in conjunction with other User Data commands (refer to [Chapter 8, “User Data Commands”](#)).

The following XEL commands are discussed in this chapter.

- [“caCell”](#) on page 372
- [“ipbAssign”](#) on page 374
- [“iplInstAssign”](#) on page 375

caCell

Specifies which cells to make into physical IP cells and/or map to hard IP.

Before using this command, designImport, design, and userData have to be executed. The cells must be specified as physical IP before the precompile step.

The syntax for this command is:

```
caCell -add {<lib> <cell>}...  
caCell -rm {<lib> <cell>}...  
caCell -add {<lib> <cell-name> <IP-ID>}
```

-add {<lib> <cell>}

Defines the specified cell in the selected library as a physical IP cell.

-rm {<lib> <cell>}

Removes the physical IP designation from the specified cell in the selected library.

-add {<lib> <cell-name> <IP-ID>}

Specifies that all instances of the cell are to be implemented by IPs of the specified IP-ID. If a cell is specified in caCell without an IP-ID, an ipInstAssign is required for each instance to be implemented by an IP. The IP-ID identifier number must be between 1 and 255. If an IP-ID is specified, all instances of that cell are assigned to IP.

Examples

```
designImport  
design  
caCell -add {mylib my_cell1} {mylib my_cell2} {mylib my_cell3}
```

Designates the my_cell1, my_cell2, and my_cell3 cells in the mylib library as physical IP.

```
caCell -add {lib1 C1 87}
```

Adds the C1 cell instance with the IP ID of 87. This cell represents all C1 design instances having an IP ID of 87.

```
caCell -rm {lib1 cell12} {lib1 cell13}
```

UXE Command Reference Manual

IP Commands

Removes the `cell2` and `cell3` cells from the list of `caCells` in the `mylib` library.

ipbAssign

The optional ipbAssign command is a command to explicitly establish a link between an IP specified in the ipInstAssign command and a specific IP location (IPP number). If no explicit assignment of the IP location is entered for a specified IP, the Palladium XP system automatically establishes an assignment.

Run this command after [ipInstAssign](#).

The syntax for this command is:

```
ipbAssign -add {<IP_label> <IPP_number>}  
ipbAssign -rm {<IP_label> [<IPP_number>]}
```

-add {<IP_label> <IPP_number>}

Associates the user-assigned IP label with the physical location of the IP. The IPP number is the number of IP port. The IPP number starts from 0. That is, it must not be less than 0.

-rm {<IP_label> [<IPP_number>]}

Removes the connection between the IP and its location.

Examples

```
ipbAssign -add {myIPP 0}
```

This assigns myIP to IP port 0.

```
ipInstAssign -add {INST1 205 xyz}  
ipbAssign -add {xyz IPP 0}
```

This associates instance INST1 to IP 205 at location IP port 0.

ipInstAssign

Used in conjunction with caCell to further specify and constrain IP usage. An IP can be assumed to be the entire logic output of a single IP board (IPB). The ipInstAssign command explicitly assigns an instance in the design to an IP, but without committing the IP to a specific IPB J-connector pin.

Before using this command, the instance's cell must be assigned as an IP cell by means of the caCell command.

Note: Exercise care in manually assigning cell instances, as the syntax must be precisely specified for correct functionality.

Run this command after caCell and before icePrepare.

The syntax for this command is:

```
ipInstAssign -add <inst_name> <IP_ID> [<IP_label>] [<IP_cell_ID>]
```

-add <inst_name> <IP_ID>

Adds a specific instance to an IP. Use of this command overrides the *<inst_name>* and *<IP_ID>* specified in the corresponding caCell command.

[<IP_label>]

Optional. The user-specified *<IP_label>* constrains the software to map the instance to a specified IP. If no IP label is provided, the software assumes that the instance can be assigned to any available IP resource. If not specifying an IP label, but specifying an *<IP_cell_ID>*, enter an asterisk (*) for this parameter.

[<IP_cell_ID>]

Optional. The *<IP_cell_ID>* constrains the software to map the instance to a specified copy of the cell on the IP. This is useful if the IP contains more than one copy of the same cell.

Examples

```
caCell -add {LIB1 CELL1 205}
ipInstAssign -add {INST1 205 xyz}
ipbAssign -add [xyz IPB3]
```

UXE Command Reference Manual

IP Commands

This associates instance INST1 of Cell1 to IP 205 at location IPB3.

```
caCell -add {LIB1 CELL1 205}  
ipInstAssign -add {INST1 205 xyz}  
ipbAssign -add [INST1 205 xyz CELL1b]
```

This script performs the same function, but in this case IP 205 contains two copies of CELL1, called CELL1a and CELL1b. This targets INST1 to the CELL1b copy.

Commands and System Tasks for SA Mode

This chapter includes detailed information about the following options, commands, tasks, and so on used when compiling designs in SA mode:

- [vhan on page 378](#)
- [vlan on page 411](#)
- [ixclkgen on page 460](#)
- [ixcom on page 464](#)
- [Synthesis Policy Checker Options on page 645](#)
- [System Tasks and Procedures on page 650](#)
- [Supported IEEE-Standard Compiler Directives on page 661](#)
- [Verilog Primitives and VHDL Procedures on page 662](#)
- [SVA Options to IXCOM on page 667](#)
- [PSL Options to IXCOM on page 680](#)

Note: Shorter forms of the dash options of the `vhan`, `vlan`, and `ixcom` commands can be given as long as the short form uniquely identifies the option. For example, with the `vhan` command, you can specify `-w` in place of `-work`. However, with the `vlan` command specifying `-w` will indicate the option to suppress warnings and not the `-work` option that specifies the logical library to be used for analysis.

vhan

Purpose

Analyzes the VHDL sources.

Category

VHDL analyzer command

Syntax

The syntax for the vhan commands, with some of the commonly used options, is:

```
vhan [-h|H] [-f|-F <options_file>] [-87|-93] [-work <library>] [-v] [-q] [-s] [-l <logfile>] VHDL_source(s) [-mti] [-libmaprc file]
```

Note: The *VHDL_source(s)* specifies one or more VHDL sources to analyze.

Syntax Example

```
vhan -work ramlib ./sources/sram.vhd
```

Parameters

The following sections describe all the options that you can use with vhan:

- [-32 | -64](#) on page 380
- [-just or -cIC](#) on page 381
- [-skip](#) on page 383
- [-its](#) on page 451
- [-qlQ](#) on page 385
- [-work | -w](#) on page 386
- [+civb](#) on page 387
- [-hlH\[elp\]](#) on page 388
- [-fl-F](#) on page 389

UXE Command Reference Manual

Commands and System Tasks for SA Mode

- [-87I-93I-200x](#) on page 390
- [-verbose](#) on page 393
- [-sIS](#) on page 394
- [-libmaprc](#) on page 395
- [-version](#) on page 397
- [-l](#) on page 398
- [-mti](#) on page 399
- [-ncsim](#) on page 400
- [-nocasestaticerror](#) on page 401
- [-psl](#) on page 402
- [-assert](#) on page 403
- [-assert vhdl](#) on page 405
- [-noassert](#) on page 406
- [-propfile](#) on page 407
- [-append log](#) on page 408
- [-ignore pragma](#) on page 409
- [+rtlCommentPragma](#) on page 410

-32 | -64

Purpose

Enables 32-bit or 64-bit compilation

Category

VHDL analyzer option

Syntax

-32

-64

Description

The `-64` option enables 64-bit mode for 64-bit operating systems, such as Red Hat EL 4.0 running on an AMD Opteron workstation or SuSE Linux Enterprise Server 11 running on an Intel Xeon-EM64T workstation. This option is useful for HDL designs that require 4 GB or more of memory in order to compile. 64-bit compilation supports the GCC G++ 4.4.0 C and C++ compilers.

The `-32` option enables 32-bit mode compilation.

-just or -cIC

Purpose

Analyzes only the specified type of modules in the VHDL sources.

Category

VHDL analyzer option

Syntax

```
-just e|a|c|b|p  
-c|C e|a|c|b|p
```

Syntax Example

```
vhan -just ab VHDL_sources
```

Parameters

e	Analyzes only the entities
a	Analyzes only the architectures
c	Analyzes only the packages
b	Analyzes only the package bodies
p	Analyzes only the configurations

Notes

- You can specify multiple types of modules.
- Analysis should occur in this order:
 - a. Packages
 - b. Package bodies

- c. Entities
 - d. Architectures
 - e. Configurations
- If you change a package body, you need to recompile the package only if the header changes.

-skip

Purpose

Lets you skip the analysis of the specified type(s) of modules in the VHDL sources.

Category

VHDL analyzer option

Syntax

`-skip e|a|p|b|c`

Syntax Example

`-skip e`

Parameters

e	Analyzes everything except the entities
a	Analyzes everything except the architectures
p	Analyzes everything except the packages
b	Analyzes everything except the package bodies
c	Analyzes everything except the configurations

Description

Similar to the `-just` or `-c | -C` option, except the specified type(s) of modules in the source files will *not* be analyzed, and all other types *will* be analyzed. See [-just or -clC](#) on page 381 for more information.

-its

Purpose

Ignores timestamp check on dependent package(s).

Category

VHDL and Verilog analyzer / Compiler option

Syntax

`-its`

Description

Disables dependency checking. This might save recompilation times for a module, such as a package, since the dependent units do not need to be recompiled.

By default, modules on which a parent unit depends are checked to make sure they were compiled prior to the parent unit. If not, you get an out-of-date error at either analysis time or elaboration time.

Note: Please use this option carefully, since the consistency checks between dependency modules are also disabled.

-q|Q

Purpose

Goes into quiet mode

Category

VHDL analyzer option

Syntax

`-q`

Description

Goes into quiet mode, which disables any kind of output being written to `stdout`. All messages will still go to the log file.

-work | -w

Purpose

Specifies a logical library in which to compile, instead of the default, WORK.

Category

VHDL analyzer option

Syntax

`-work library`

+civb

Purpose

Enables a case insensitive search on modules.

Category

VHDL analyzer option

Syntax

+civb

-h|H[elp]

Purpose

Displays a listing of all vhan options.

Category

VHDL analyzer option

Syntax

`-h | H[elp]`

-f|-F

Purpose

Specifies a file containing additional options and/or sources.

Category

VHDL analyzer option

Syntax

`-f <options_file>`
`-F <options_file>`

Note: Do not put the `-ua` and `+dut` options in the `<options_file>`; these are ignored. Note the difference between the `-f` and `-F` options. When you use `-f <options_file>`, the options are read from a file that is relative to directory from which `vlan` or `IXCOM` was invoked. When you use `-F <options_file>`, the options are read from a file that is relative to the directory in which `<options_file>` is located.

-87|-93|-200x

Purpose

Sets the analyzer to the required *IEEE Std for VHDL* as follows:

- IEEE Std 1076-1987 for VHDL* when -87 is used
- IEEE Std 1076-1993 for VHDL* when -93 is used
- IEEE Std 1076-2008 for VHDL* when -200x is used

Category

VHDL analyzer option

Syntax

-87 | -93 | -200x

Description

The default is -93, unless you use either the -mti or -ncsim option, in which case, the default is -87. IXCOM supports all the constructs mentioned in *IEEE Std 1076-1987 for VHDL* and *IEEE Std 1076-1993 for VHDL*.

-200x

Purpose

Enables the analyzer to *IEEE Std 1076-2000 for VHDL*.

Category

VHDL analyzer option

Syntax

`-200x`

Description

Sets the analyzer to *IEEE Std 1076-2000 for VHDL*. The default is `-93`, unless you use either the `-mti` or `-ncsim` option, in which case, the default is `-87`.

When `-200x` option is used, support is provided for the following:

- ❑ Logical reduction operators (Section 9.2.2 of VHDL-LRM: IEEE Std 1076-2008)
- ❑ Multiline Comments (Section 15.9 of VHDL-LRM IEEE Std 1076-2008)

```
/* A long comment may continue to multiple lines.  
This is end of the long comment */  
X <= '1'; /* This is also a comment */
```
- ❑ Matching relational operators (Section 9.2.3. of VHDL-LRM IEEE Std 1076-2008)
- ❑ Process (ALL) construct (Section 11.3. of VHDL-LRM IEEE Std 1076-2008)
- ❑ VHDL 200x External Names (Section 8.7 of VHDL-LRM: IEEE Std 1076-2008)
- ❑ Read for output ports (Section 6.5.2 - Interface Object Declarations of VHDL-LRM: IEEE Std 1076-2008)

Following are the limitations in support for VHDL constructs with the `-200x` option:

- ❑ External names involving hierarchies from other languages like Verilog, SystemC, etc. are not supported. All the elements in the external-name hierarchy must be from VHDL.
- ❑ External names for shared variables are not supported.

UXE Command Reference Manual

Commands and System Tasks for SA Mode

- ❑ Use of external names in PSL assertions is not supported. Use of external names is not supported in DUT if it involves blocks or generate blocks.
- ❑ If the design includes generate blocks, use the `-rtlNameForGenerate` option for support of external names. External names must not be within DUT.
- ❑ Use of an external name within an external name is supported only when it is a constant.
- ❑ Aliases for external names are not supported.
- ❑ Use of VHDL 200x constructs in VHDL packages is not supported.
- ❑ Use of physical types and cyclic instantiations of a module is not supported with the `-200x` option.

-verbose

Purpose

Produces verbose messages.

Category

VHDL analyzer option

Syntax

`-verbose`

-sIS

Purpose

Sorts the specified VHDL sources in order of module dependency, then analyzes them in the correct order.

Category

VHDL analyzer option

Syntax

-sIS

Description

Sorts the specified VHDL sources in order of module dependency, then analyzes them in the correct order. This is useful if you have a set of sources, but do not know the order in which to compile them.

Note: All sources will be compiled to the library specified with the `-work` option or the default WORK library.

-libmaprc

Purpose

Specifies a file containing logical-library-to-physical-directory associations, as required by mixed HDL tools.

Category

VHDL analyzer option

Syntax

`-libmaprc file`

Usage notes

Use the libmaprc file instead of the axisrc file to specify the logical library to physical directory associations.

The use of `-axisrc` option is deprecated and supported only for backward compatibility with existing tools that make use of the axisrc file.

Instead, use the libmaprc file, which follows the same syntax as the axisrc file. For more information on the libmaprc file, refer to [-libmaprc](#) and the *Associating Logical Libraries to Physical Directories Using libmaprc File* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

Depending on whether the `-axisrc` or `-libmaprc` option is specified with the `ixcom` command, the compiler takes the following actions:

- If the `-axisrc` option is specified and the `-libmaprc` option is not specified, the compiler reads all the files specified with the `-axisrc` option in the given order.
- If the `-axisrc` and `-libmaprc` options are specified, the compiler reads the files specified with the `-libmaprc` option in the given order and ignores the `-axisrc` option with a warning.

-L

Purpose

Enables you to search libraries for modules.

Category

Compiler option

Syntax

`-L <logical_library>`

Description

When instantiating VHDL or Verilog components, lets you specify different logical libraries in which to search for modules. By default, only the parent library is searched when no logical library is specified in the VHDL instantiation.

-version

Purpose

Prints the version of the analyzer

Category

VHDL analyzer option

Syntax

`-version`

-l

Purpose

Specifies a *logfile*, instead of the default, vhan.log

Category

VHDL analyzer option

Syntax

-l *logfile*

-mti

Purpose

Improves compatibility with Mentor's ModelSim simulator

Category

VHDL analyzer option

Syntax

`-mti`

Description

Improves compatibility with Mentor's ModelSim simulator as follows:

- Enables certain non-standard VHDL semantics, such as the use of other signals in signal initialization statements.
- Sets the default VHDL standard to `-87` instead of `-93`.
- Sets the default minimum simulation time resolution to 1 ns. This can be overridden with the `-t` option.

-ncsim

Purpose

Improves compatibility with Cadence's NCSim simulator

Category

VHDL and Verilog analyzer option

Syntax

`-ncsim`

Description

Improves compatibility with Cadence's NCSim simulator as follows:

- Enables certain non-standard VHDL semantics, such as an incomplete port association list in component instantiations when elements of arrays are connected. The non-specified elements of the array will be left as OPEN.
- Sets the default VHDL standard to -87 instead of -93.

Note: Use the `-relax` option with the compiler when you use this option.

-nocasestaticerror

Purpose

Enables non-static expressions in the `case` statement.

Category

VHDL analyzer option

Syntax

`-nocasestaticerror`

-psl

Purpose

Enables parsing of PSL assertions in the DUT. Parsing of PSL assertions is *not enabled* by default.

Category

VHDL analyzer / Verilog Analyzer / Compiler option

Syntax

`-psl`

-assert

Purpose

Enables parsing of PSL assertions in the HDL files. Parsing of PSL assertions is *not enabled* by default.

Category

VHDL analyzer / Verilog Analyzer / Compiler option

Syntax

`-assert`

-assert_vlog

Purpose

Enables parsing of PSL assertions in the Verilog files. Parsing of PSL assertions is *not enabled* by default.

Category

Verilog Analyzer / Compiler option

Syntax

`-assert_vlog`

-assert_vhdl

Purpose

Enables parsing of PSL assertions in the VHDL files. Parsing of PSL assertions is *not enabled* by default.

Category

VHDL analyzer / Compiler option

Syntax

`-assert_vhdl`

-noassert

Purpose

Disables parsing of SVA and PSL assertions in the HDL files.

Category

Compiler option

Syntax

`-noassert`

-profile

Purpose

Ensures that PSL assertions specified in external files (other than the design file) are recognized and parsed. The *<file>* option refers to the external files containing PSL assertions, such as vunits.

Use of the [-assert](#) option is required to enable the recognition and parsing of PSL assertions in the design file.

Category

VHDL analyzer / Verilog Analyzer / Compiler option

Syntax

`-profile <file>`

-append_log

Purpose

Appends the log to the existing log file.

Category

VHDL analyzer option

Syntax

`-append_log`

-ignore pragma

Purpose

Ignores all the pragmas in the design prefixed with the specified *<name>*.

For example, `-ignore pragma synopsys` will not honor pragmas that have `synopsys` as their prefix, such as, `synopsys translate_off/on`. The ignored pragmas are treated as simple comments.

Category

VHDL analyzer option

Syntax

`-ignore pragma <name>`

+rtlCommentPragma

Purpose

Comments out code from a `translate_off` pragma (and related comment pragmas) to the next `translate_on` pragma.

Category

VHDL analyzer option

Syntax

`+rtlCommentPragma`

Description

Comments out code from a `translate_off` pragma (and related comment pragmas) to the next `translate_on` pragma, that is, this code will not be executed in any run-time modes (software simulation, acceleration, or emulation modes).

Note:

This option can be passed to `vlan`, `vhan`, or the compiler.

- With `vlan` or the compiler, you can specify the name of a specific *module* by using the syntax `+rtlCommentPragma [+module...]`, rather than letting it default to the entire file or files specified.
- With `vhan`, you cannot specify an individual entity; the option will always apply to the entire file or files.

vlan

Purpose

Analyzes the Verilog sources

Category

Verilog analyzer command

Syntax

```
vlan [-work library] [-f|F <options_file>] Verilog_source(s) other_options  
[-libmaprc file]
```

Description

The `vlan` commands analyze the Verilog sources.

Notes

- 64-bit compilation supports only the GCC G++ 4.4.0 C and C++ compilers.
- The `Verilog_source(s)` specifies one or more Verilog sources to analyze.
- Options relating to Simulation Acceleration modes (such as `-ua` and `+dut`) should be passed to Incisive Xaccelerator COMPiler, that is, IXCOM for elaboration, NOT to `vlan`.

The following sections describe the options available for the `vlan` commands:

- [-h](#) on page 414
- [-work](#) on page 415
- [-f|F](#) on page 416
- [+v2000](#) on page 418
- [-libmaprc](#) on page 419
- [-l](#) on page 420
- [-q](#) on page 421

- [-u](#) on page 422
- [-v](#) on page 423
- [-w](#) on page 425
- [-y](#) on page 426
- [+define](#) on page 427
- [+incdir](#) on page 428
- [+libext](#) on page 429
- [+liborder](#) on page 430
- [+librescan](#) on page 431
- [+libverbose](#) on page 432
- [+convertUdp](#) on page 433
- [-Z](#) on page 435
- [+max_error_count](#) on page 436
- [+maxdelays](#) on page 437
- [+mindelays](#) on page 438
- [+nowarn](#) on page 439
- [-vtimescale](#) on page 440
- [-netlist](#) on page 441
- [+rtlHonorCasePragma](#) on page 596
- [+rtlCommentPragma](#) on page 638
- [+rtlIgnPragma](#) on page 557
- [-P](#) on page 443
- [-p](#) on page 444
- [+rtlIgnTask](#) on page 445
- [+tfReadOnly](#) on page 446
- [+primchange](#) on page 447
- [+loadpli1](#) on page 448
- [+loadvpi](#) on page 449
- [-psl](#) on page 402

UXE Command Reference Manual

Commands and System Tasks for SA Mode

- [-propfile](#) on page 407
- [+sva+](#) on page 668
- [+no_sva](#) on page 669
- [+specblk](#) on page 450
- [-its](#) on page 451
- [-ncsim](#) on page 452
- [+timescale](#) on page 453
- [-default_ext](#) on page 454
- [+showSvPkgAccess](#) on page 455
- [-sysv_ext](#) on page 456
- [-vlog95_ext](#) on page 457
- [-vlog_ext](#) on page 458
- [-vlogext](#) on page 459
- [-incLibFileCompilationUnit](#) on page 511
- [-ixerror](#) on page 643
- [-ixfatal](#) on page 644

-h

Purpose

Displays a listing of all `vlan` options.

Category

Verilog analyzer option

Syntax

`-h`

-work

Purpose

Specifies the logical *library* (default is WORK) in which to analyze.

Category

Verilog analyzer option

Syntax

`-work <library>`

-f|F

Purpose

Reads selected options from a file, as opposed to typing them out on the command line.

Category

Verilog analyzer option

Syntax

`-f | F <options_file>`

Parameters

`<options_file>` Specifies the name of the options file. The `<options_file>` can contain:

- A list of pathnames for Verilog source files
- Any of the following analysis-time options:

`-f`
`-F`
`-v`
`-y`

- Any of the following +plusargs:

`+incdir`
`+libext`
`+libnonamehide`
`+liborder`
`+libsrescan`
`+maxdelays`
`+mindelays`
`+nowarnTFMPC`
`+pathpulse`
`+pulse_e`
`+pulse_r`
`+typdelays`

Notes

- You cannot specify C source files or object filenames in the file.
- If you use the `-f` option, you must use *absolute* path. You can also specify a path to the file, but you must still use absolute path for the included files.
- If you use the `-F` option, you must specify a path to the file, and then you can use *relative* path for the Verilog source files in the file. Nested files are not supported.
- Note the difference between the `-f` and `-F` options. When you use `-f <options_file>`, the options are read from a file that is relative to directory from which `vlan` or `IXCOM` was invoked. When you use `-F <options_file>`, the options are read from a file that is relative to the directory in which `<options_file>` is located.
- Do not put the `-ua` and `+dut` options in the `<options_file>`; these are ignored.

+v2000

Purpose

Enables the following *IEEE Std 1364-2001 for Verilog HDL* features: generate block features, genvar declaration, signed.

Category

Verilog analyzer option

Syntax

+v2000

-libmaprc

Purpose

Specifies a file containing logical-library-to-physical-directory associations.

Category

Verilog analyzer option

Syntax

`-libmaprc <file>`

Description

Specifies a file containing logical-library-to-physical-directory associations, as required by mixed HDL tools.

Usage notes

Use the libmaprc file instead of the axisrc file to specify the logical library to physical directory associations.

The use of `-axisrc` option is deprecated and supported only for backward compatibility with existing tools that make use of the axisrc file.

Instead, use the libmaprc file, which follows the same syntax as the axisrc file. For more information on the libmaprc file, refer to [-libmaprc](#) and the *Associating Logical Libraries to Physical Directories Using libmaprc File* section in the *Compiling Designs for Simulation Acceleration Using IXC* chapter of the *UXE User Guide*.

Depending on whether the `-axisrc` or `-libmaprc` option is specified with the `ixcom` command, the compiler takes the following actions:

- If the `-axisrc` option is specified and the `-libmaprc` option is not specified, the compiler reads all the files specified with the `-axisrc` option in the given order.
- If the `-axisrc` and `-libmaprc` options are specified, the compiler reads the files specified with the `-libmaprc` option in the given order and ignores the `-axisrc` option with a warning.

-l

Purpose

Specifies a *logfile*.

Category

Verilog analyzer option

Syntax

-l <*logfile*>

Description

Specifies a *logfile*, instead of `vlan.log`, which is the default.

-q

Purpose

Goes into quiet mode.

Category

Verilog analyzer option

Syntax

`-q`

Description

Goes into quiet mode, which disables any kind of output being written to `stdout`. All messages will still go to the log file.

-u

Purpose

Considers all net, register, and instance path as uppercase characters.

Category

Verilog analyzer option

Syntax

`-u`

-v

Purpose

Uses the specified Verilog *library_file* for resolving library cells and modules in the design.

Category

Verilog analyzer option

Syntax

`-v <library_file>`

Description

Specifies a Verilog library file. `vlan` will search this file for modules that are not defined in the Verilog sources specified.

Note: `vlan` will give you a warning only if there are any modules that are not defined in the Verilog source or in any `-v` or `-y` libraries. Such modules must be resolved at compile time.

-W

Purpose

Specifies the working physical directory.

Category

Compiler option

Syntax

`-W directory`

Description

Specifies the working physical directory. Use this option instead of `-work library` if you want to specify a physical directory to compile to, rather than a logical library.

-W

Purpose

Suppresses warnings.

Category

Verilog analyzer / Compiler option

Syntax

`-W`

-y

Purpose

Searches only files with the specified extensions in the specified library directory.

Category

Verilog analyzer option

Syntax

`-y directory`

Description

Searches only files with the specified extensions (`ext[ext]...`) in the specified `library_directory` when resolving library cells and modules in the design. Must be used with `+libext`.

Note: `vlan` will give you a warning only if there are any modules that are not defined in the Verilog source or in any `-v` or `-y` libraries. Such modules must be resolved at elaboration time.

+define

Purpose

Defines Verilog macros.

Category

Verilog analyzer option

Syntax

`+define+[macro]`

Syntax Example

```
'ifdef USE_DRAM  
'else  
'endif
```

Description

With the `'ifdef` and `'define` Verilog compiler directives, defines a macro or passes a value to a macro.

+includir

Purpose

Specifies one or more directory paths to be searched for the include files specified with the '`include`' compiler directive.

Category

Verilog analyzer option

Syntax

`+includir+path[+path...]`

+libext

Purpose

Searches only files with the specified extensions when resolving library cells and modules in the design.

Category

Verilog analyzer option

Syntax

`+libext+extension [+extension...]`

Note: Use with the `-y library_directory` option

+liborder

Purpose

Searches for module definitions in the next specified library.

Category

Verilog analyzer option

Syntax

+liborder

Description

With the `-y library_directory` option, searches for module definitions in the *next* library specified on the command line when an unresolved instance is found in a particular library.

+librescan

Purpose

Searches for module definitions in the first specified library.

Category

Verilog analyzer option

Syntax

+librescan

Description

With the `-y library_directory` option, searches for module definitions in the *first* (as opposed to *next*, as with `+liborder`) library specified on the command line when an unresolved instance is found in a particular library.

+libverbose

Purpose

Displays the order in which libraries are scanned for module definitions

Category

Verilog analyzer option

Syntax

+libverbose

Description

With the `-y library_directory` option, displays the order in which libraries are scanned for module definitions.

+convertUdp

Purpose

By default, the IXCOM compiler does not transform the User-Defined Primitives (UDPs) in the hardware partition. All the UDPs are passed through to HDL Compiler directly. In order to enable the transformation (that is, the old UDP flow) by IXCOM, you can use the `+convertUdp` option. It automatically generates synthesizable RTL code for all UDPs and puts the code in the `<udp_file>`.

For example:

```
ixcom dut.sv udp.v +convertUdp+udp_file +dut+dut -ua
```

Category

Compiler option

Syntax

```
+convertUdp+<udp_file>
```

+noHdliceCompatUdpMapping

Purpose

By default, IXCOM maps UDPs to synthesizable modules to match the HDL Compiler behavior. The `+noHdliceCompatUdpMapping` option disables this default behavior of IXCOM.

Category

Compiler option

Syntax

`+noHdliceCompatUdpMapping`

-Z

Purpose

Automatically replaces modules in the design with modules of the same name found in *map_file*.

Category

Verilog analyzer option

Syntax

`-Z map_file`

Description

Automatically replaces modules in the design with modules of the same name found in *map_file*, which enables you to easily replace a module without disturbing the design flow.

Note: For a mixed HDL design, use this option when compiling the *udp_file* generated by the `+convertUdp` option.

+max_error_count

Purpose

Sets the maximum error count to *n*.

Category

Verilog analyzer option

Syntax

`+max_error_count+n`

+maxdelays

Purpose

Uses the `max` value, instead of the default `typ`, in a `min:typ:max` delay expression.

Category

Verilog analyzer option

Syntax

`+maxdelays`

+mindelays

Purpose

Uses the `min` value, instead of the default `typ`, in a `min:typ:max` delay expression.

Category

Verilog analyzer option

Syntax

`+mindelays`

+nowarn

Purpose

Suppresses the display of the specified *message_type(s)* during analysis.

Category

Verilog analyzer option

Syntax

+nowarn+*message_type*[+*message_type*...]

Note: To determine a *message_type*, look in the square brackets ([]) following a message.

Example

The following argument:

+nowarn+TOO_FEW_PORTS

Suppresses this type of message:

```
Warning! in file file at line line [TOO_FEW_PORTS]  
Too few module port connections in instance (instance)
```

-vtimescale

Purpose

Sets the default timescale (initial) timescale (including unit and precision) for the files specified on the command line. Using this option implies an implicit `timescale directive at the beginning of each of the command-line files. So, if there is no explicitly written `timescale directive in that file, all modules have the timescale specified with the –vtimescale option. If there is an explicit `timescale in the middle of a file, all modules before the explicit `timescale directive will have the timescale specified with the –vtimescale option. The modules after the explicit `timescale directive will have the timescale specified with this directive in the file.

Category

Verilog analyzer option

Syntax

`-vtimescale <unit/precision> <filenames>`

Syntax Example

`-vtimescale 1ns/1ps myfile1.v myfile2.v`

-netlist

Purpose

Analyzes the netlist files separately. All source files specified in the command-line are analyzed as netlist files and the analyzed information is saved into a database in a compact format, which is different from the format used by RTL compilation. Also, the database files for netlist files are kept separate from RTL files compiled into the same logical library.

The netlist files can be analyzed into different logical libraries (similar to RTL compilation) using the `-work <library>` option. The default logical library (`WORK`) is used if the logical library is not explicitly specified.

Category

Verilog analyzer option

Syntax

```
vlan -netlist <netlist_source_files>
```

+protect

Purpose

Generates an encrypted file for the Verilog code in the region specified by the ‘`protect`’ and ‘`endprotect`’ compiler directives.

Category

Verilog analyzer source encryption option

Syntax

`+protect[.ext]`

Note: The remaining code in the file(s) is not encrypted.

-P

Purpose

Specifies a VCS format table file for PLI routines.

Category

Verilog PLI option

Syntax

`-P <file>`

Note: You must also pass this option to the compiler.

-p

Purpose

Specifies a Verilog format table file for PLI routines.

Category

Verilog PLI option

Syntax

`-p <file>`

Note: You must also pass this option to the compiler.

+rtlIgnTask

Purpose

Ignores the specified PLI or user tasks or functions when compiling for the emulator.

Category

Verilog PLI option

Syntax

```
+rtlIgnTask[+task_or_function] [+task_or_function...]
```

Description

Ignores the specified PLI or user tasks or functions (`$` sign is not required) when compiling for the emulator (`-ua` option). If these tasks are not ignored, the modules in which they exist are automatically run in software simulation mode only, resulting in slower performance. If you specify *no task_or_function(s)*, all PLI tasks and functions in the design are ignored.

+tfReadOnly

Purpose

Recognizes that the specified PLI tasks/functions only *read* their parameters

Category

Verilog analyzer option/Verilog PLI option

Syntax

```
+tfReadOnly[+task_or_function] [+task_or_function...]
```

Description

Recognizes that the specified PLI tasks/functions only *read* their parameters (they do not *write* parameters passed to them). If you do not specify any *task_or_function(s)*, *all* PLI tasks and functions in the design are considered to only read their parameters.

This option prevents Signal Analysis (SA) from assuming the PLI task or function is *writing* a signal coming from the DUT.

Example

In this testbench code, a PLI task *logF* is used to log the value of a signal inside the DUT into a file:

```
always@(dut.b1.c1.flag)
  $logF("Flag is active:%d",
        dut.b1.c1.flag)
```

To assure SA that *logF* is only *reading* the signal, use this option:

```
+tfReadOnly+logF
```

Without this option, SA assumes *logF* might write to *dut.b1.c1.flag* and warns about software-to-hardware cross-module reference writes:

```
Warning: xmr defined in DUT can't be assigned in testbench --
dut_instance_path.dut_signal_name at line in file -- Please use
force for DUT write or $axis_rccoff to guard initialization code.
```

+primchange

Purpose

Enables mapping of transistor-level primitives to user-defined modules, in map file(s) specified with the `-Z map_file option(s)`.

Category

Verilog analyzer option/Verilog PLI option

Syntax

`+primchange`

Description

Enables mapping of transistor-level primitives, such as `tran`, to user-defined modules, in map file(s) specified with the `-Z map_file` option(s). These primitives can then be compiled for Simulation Acceleration modes.

Note: Transistor-level primitives `pmos` and `nmos` are supported in Simulation Acceleration modes with some restrictions, by using both the `+tran` and `+tran_force` options with the compiler. Other transistor-level primitives are flagged as compile-time errors when compiled for Simulation Acceleration modes.

Transistor-level primitives are allowed in software simulation by using the `+tran` option with the compiler.

Example

Here is a `tran` mapping example:

```
module tran (io, io);
    inout io;
endmodule
```

+loadpli1

Purpose

Specifies the PLI1 *library_name:boot_routine(s)*.

Category

Verilog PLI option

Syntax

+loadpli1=<*library_name*>:<*boot_routine*>

+loadvpi

Purpose

Specifies the VPI *library_name:boot_routine(s)*.

Category

Verilog PLI option

Syntax

`+loadvpi=<library_name>:<boot_routine>`

+specblk

Purpose

Enables processing of `specify` blocks.

Category

Verilog Analyzer / Compiler option

Syntax

`+specblk`

-its

Refer to [-its](#) on page 384 for more information on `-its` option.

-ncsim

Refer to [-ncsim](#) on page 400 for more information on -ncsim option.

+timescale

Purpose

Specifies timescale for modules with no timescale.

Category

Verilog Analyzer option

Syntax

`+timescale+<unit/prec>`

-default_ext

Purpose

Overrides the default file extension map.

Category

Verilog Analyzer / Compiler option

Syntax

`-default_ext <file_type>`

+showSvPkgAccess

Purpose

Prints an INFO message to inform whenever a particular SystemVerilog package or a compilation unit object is accessed for the first time.

Category

Verilog Analyzer option

Syntax

+showSvPkgAccess

Example

```
Info in file dut.sv at line 6      [FOUND_COMP_UNIT_ACCESS]
  Access to a compilation unit scope object found (obj: type1, file:
  cu1.sv, line: 4)
```

Using the +showSvPkgAccess switch helps determine the possible situations where compilation unit access issues might occur.

-sysv_ext

Purpose

Adds/overrides file extension for SystemVerilog sources.

Category

Verilog Analyzer / Compiler option

Syntax

`-sysv_ext [+]<extension>`

Example

`-sysv_ext +.h`

The above command adds `.h` to the list of extensions specified (or built-in extensions).

`-sysv_ext .h`

The above command overrides previously specified list (if any) or built-in extensions defined for the language with the `.h` extension. Note that the above command does not include the `+` sign.

-vlog95_ext

Purpose

Adds/overrides extension for Verilog-95 sources.

Category

Verilog Analyzer / Compiler option

Syntax

`-vlog95_ext [+]<extension>`

Example

`-vlog95_ext +.h`

The above command adds `.h` to the list of extensions specified (or built-in extensions).

`-vlog95_ext .h`

The above command overrides previously specified list (if any) or built-in extensions defined for the language with the `.h` extension. Note that the above command does not include the `+` sign.

-vlog_ext

Purpose

Adds/overrides extension for Verilog sources.

Category

Verilog Analyzer / Compiler option

Syntax

`-vlog_ext [+]<extension>`

Example

`-vlog_ext +.h`

The above command adds `.h` to the list of extensions specified (or built-in extensions).

`-vlog_ext .h`

The above command overrides previously specified list (if any) or built-in extensions defined for the language with the `.h` extension. Note that the above command does not include the `+` sign.

-vlogext

Purpose

Adds/overrides extension for Verilog sources.

Category

Verilog Analyzer / Compiler option

Syntax

`-vlogext [+<extension>`

Example

`-vlogext +.h`

The above command adds `.h` to the list of extensions specified (or built-in extensions).

`-vlogext .h`

The above command overrides previously specified list (if any) or built-in extensions defined for the language with the `.h` extension. Note that the above command does not include the `+` sign.

ixclkgen

Enables automatic generation of the ICE clocks through a standalone utility called *ixclkgen*, which is provided in the UXE installation directory to generate clocks from user data. This utility accept the XEL user data as input, and generates a Verilog module containing the modeled clocks.

Purpose

The *ixclkgen* utility supports the ICE-clock generation and enables you to:

- Specify the clocking information using ICE user data (or command) syntax.
- Generate uncontrolled (or free running) clocks as in ICE mode.
- Automatically generate the Verilog module to model clocks for the information that you specified by using the ICE user data.

Category

Verilog Analyzer / Compiler option

Syntax

```
ixclkgen -input <file> \
           -output <file> \
           [ -controlled <clock_source>[,<clock_source> ...] \
           [ -default controlled | uncontrolled ] \
           [ -help ] \
           [ -hierarchy <string> ] \
           [ -use_ixc_clkgen ] \
           [ -module <name> ] \
           [ -rtc ] \
           [ -uncontrolled <clock_source>[,<clock_source> ...] \
           [ -timescale <arg> ]
```

For all the options listed above, you can use a shortened name by specifying a minimum number of required characters. For example: -i for -input, -o for -output, -u for -uncontrolled_clocks, -hi or -hier for -hierarchy, and so on.

Following is a description of the options of the *ixclkgen* command.

-input <file>

Specifies the file that contains the clocking user data in XEL format. See the Specifying Clocking Information section in the Modeling Clocks appendix of the UXE User Guide for a list of the user data that can be specified to *ixclkgen*. This user data is read as text, and does not have TCL interpretation support.

-output <file>

Specifies the output file name for the generated clock module.

-controlled <clock_sources>

Specifies the controlled clock sources as comma-separated names.

-default controlled | uncontrolled

Specifies the default clock-generation behaviour for the clock sources not listed explicitly using the `-controlled` or `-uncontrolled` option. The default behavior for the clock source is controlled.

-help

Displays the information about the *ixclkgen* utility usage and the options of the `ixclkgen` command.

-hierarchy <string>

Specifies the hierarchical path for the clock signals referred in the user data. You must specify the language-specific delimiter character with this option.

-use_ixc_clkgen

Specifies that `IXCclkgen` module from the library should be used to generate master clock instead of the default module.

-module <name>

Specifies the name of the Verilog module for the clock-generation module. If you do not specify this option, the *ixclkgen* utility uses the default module name, which is `_ixc_clkgen`.

-rtc

Generates the internal master clock to enable the user-clock configuration at run time using `clockConfig` command.

-uncontrolled <clock_sources>

Specifies the uncontrolled or free-running clock sources as comma-separated names. The default behavior for the clock source is controlled.

-timescale <arg>

Specifies the timescale value for the fastest user clock. Following are the values that you can specify for `<arg>`:

- 1
- 10
- 100
- ns
- ps
- fs

Examples:

To generate the clock module and compile it as a top-level side module:

```
ixclkgen -input ud.gel -output clk.sv
ixcom <source_files_and_options> clk.sv -ua +dut+\_
      ixc_clkgen -v <product_install_directory>/etc/ixcom/IXCclkgen.sv \
      {-timescale <unit> | <precision>}
```

To generate the clock module and compile it as a top-level side module using the multi-step-compilation process with separate analyze and elaboration steps:

```
ixclkgen -input ud.gel -output clk.sv -module Clk
```

UXE Command Reference Manual

Commands and System Tasks for SA Mode

```
vlan <source_files_and_options> clk.sv \
    -v <product_install_directory>/etc/ixcom/IXCclkgen.sv
ixcom [ <options> ] -top <du> -ua +dut+Clk {-timescale <unit> | <precision>}
```

To generate the clock module with uncontrolled clock sources:

```
ixclkgen -input ud.qel -output clk.sv \
    -uncontrolled clkA -uncontrolled clkB,clkC
```

To generate the clock module with default uncontrolled clocks, and one or more controlled clocks:

```
ixclkgen -input ud.qel -output clk.sv -default uncontrolled \
    -controlled clkA -controlled clkB,clkC
```

To generate the clock module with the default hierarchical path for clock-source signals:

```
ixclkgen -input ud.qel -output clk.sv -hierarchy "Tb.dut."
```

To generate the clock module with time values scaled to 10ps:

```
ixclkgen -input ud.qel -output clk.sv -timescale 10ps
```

ixcom

Purpose

Compiles all Verilog and VHDL sources.

Syntax

Here is the syntax of the help command that will list all the `ixcom` options:

```
ixcom -help all
```

Note: The `compilerOptions.qel` and `compile.qel` files in the IXCOM compilation directory can be used to include the compilation directives and to modify the compilation flow, respectively. For more information about the files refer to the *Compiling for Hardware Using IXCOM* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

Parameters

Generic Options

- | | | |
|--|--|---|
| ■ <u>-32 / -ixcg32 / -64</u> | ■ <u>-append_log</u> | ■ <u>-clean</u> |
| ■ <u>-timescale</u> | ■ <u>-fl-F</u> | ■ <u>-its</u> |
| ■ <u>-l</u> | ■ <u>-W</u> | ■ <u>-h[elp]</u> |
| ■ <u>-ncsim</u> | ■ <u>-q</u> | ■ <u>-Z</u> |
| ■ <u>-u</u> | ■ <u>-vaelabargs</u> | ■ <u>-vavhdlargs</u> |
| ■ <u>-vavlogargs</u> | ■ <u>-ncvlogargs</u> | ■ <u>-ncvhdlargs</u> |
| ■ <u>-ncelabargs</u> | ■ <u>-xeCompile</u> | ■ <u>-xecompileargs</u> |
| ■ <u>-generic</u> | ■ <u>-version</u> | ■ <u>-localDiskComp</u> |
| ■ <u>+localDiskComp</u> | ■ <u>-verbose</u> | ■ <u>+max_error_count</u> |
| ■ <u>-w</u> | ■ <u>-work</u> | ■ <u>-Z</u> |
| ■ <u>+specblk</u> | ■ <u>+define</u> | ■ <u>+incdir</u> |
| ■ <u>+convertUdp</u> | ■ <u>+noHdliceCompatUdpMapping</u> | ■ <u>+delVhdlEntity</u> |

UXE Command Reference Manual

Commands and System Tasks for SA Mode

- [+no strnInst](#)
- [+nowarn](#)
- [-o](#)
- [-top](#)
- [+libext](#)
- [-relax](#)
- [+dpiprofsamplecnt](#)
- [-keepInitialFinal](#)
- [+margdpisize](#)
- [+xmrTfInlineOff](#)
- [-lint](#)
- [+uvmDut+<arg>](#)
- [+systemc args](#)
- [+fifoDisp](#)
- [-synth prefix](#)
- [-ixerror](#)
- [-relativeIXCDIR](#)
- [+maxdelays](#)
- [-incdir](#)
- [+asis](#)
- [-libmaprc](#)
- [+pd3legacy](#)
- [-featureFile](#)
- [-externalNetlists](#)
- [-conformal](#)
- [+margdirect](#)
- [+mapInit](#)
- [+maxPIPO](#)
- [-incLibFileCompilationUnit](#)
- [+tfconfig](#)
- [+staticFd](#)
- [-rm synth prefix](#)
- [-ixfatal](#)
- [+moduleStub](#)
- [+mindelays](#)
- [-m](#)
- [+upper](#)
- [+lower](#)
- [-ncccompile](#)
- [+tbcnba](#)
- [-ignoreInitialFinal](#)
- [+margdpi](#)
- [+xmrTfInlineOn](#)
- [+no mapInit](#)
- [+ignoreNCVerCheck](#)
- [-loadsc](#)
- [+fifo lsize+size](#)
- [-rtlNameForGenerate](#)
- [-defparam](#)
- [+genIfcSideFile](#)
- [-msgSummary](#)

Assertion Options

- [-psl](#)
- [-assert vhdl](#)
- [+assertCover](#)
- [+assertFailCounter+hw](#)
- [+assertFailedCntr+<N>](#)
- [+no sva](#)
- [-assert](#)
- [-noassert](#)
- [+assertCoverCnt](#)
- [+assertCheckedCntr+<N>](#)
- [+no psl](#)
- [-assert vlog](#)
- [-propfile](#)
- [+assertStatus](#)
- [+assertFinishedCntr+<N>](#)
- [+sva+](#)

Coverage Options

- [-covdut](#)
- [+ccovModelSummary](#)
- [-covfile](#)
- [+ccovProfile](#)
- [+covGenFeatureFile](#)
- [+fcovCoverCnt+<N>](#)

UXE Command Reference Manual

Commands and System Tasks for SA Mode

- -coverage [block | toggle | functional | all]
- +fcov_uxe_db
- +tcovMDA[+<exponent>]
- -covMakeNamesUnique
- -featureFile
- -covAddResetLogic
- +tcovTypes+<type>

File Extension-based Compilation Options

- -default_ext
- -sysv_ext
- -vlog95_ext
- -vlog_ext
- +rtlCommentPragma
- +rtlHonorCasePragma
- +rtlIgnPragma
- +showPragma
- -strictLRM

Hardware Compile Options

Options relating to the emulator (such as -ua and +dut) should be passed to ixcom command for elaboration, **NOT** to vlan.

- +bc
- +bcc
- +bcAsyncEvent
- +bcDebug
- +bufdisp
- +delay
- +dump_bidir
- +xcBidir
- +dut
- -target
- +dutignore
- +hm
- +hmpi
- +hwdpi
- +hwfrc
- +iscDelay
- +hwRandom
- -ixclkgen
- +iscdisp
- +margtbc
- +newhm
- +rtlIgnBlkDelay
- +rtlIgnPragma
- +rtlIgnTask
- +rtlKeepDelays
- +rtlMemsize
- +rtlVarwidth
- +salgnXmrWrite
- +salgnDanglingClock
- +no_rtlchk
- +saVerbose
- +synloopsize
- +tb_export
- +tb_import
- +tb_import_systf
- +1xua
- +tran_relax
- +no_tran
- +vhdlmcx
- +xcDesignTop
- +targetTop
- +enableSvPkg
- -vhdl_time_precision
- -rtlchk
- -enableEmbeddedSW
- -ua
- +xcEmbeddedTb
- -trans rtl
- +prefixXref

UXE Command Reference Manual

Commands and System Tasks for SA Mode

Library File Search and Scan Options

- | | | |
|--|--|--|
| <ul style="list-style-type: none">■ <u>-V</u>■ <u>+liborder</u>■ <u>-P</u>■ <u>+loadvpi</u> | <ul style="list-style-type: none">■ <u>-Y</u>■ <u>+librescan</u>■ <u>-p</u>■ <u>+tfReadOnly</u> | <ul style="list-style-type: none">■ <u>+libext</u>■ <u>+libverbose</u>■ <u>+loadpli1</u>■ <u>-L</u> |
|--|--|--|

Verilog Design Compilation Mode Options

- | | | |
|--|---|--|
| <ul style="list-style-type: none">■ <u>+SV</u>■ <u>+v2000</u> | <ul style="list-style-type: none">■ <u>+v1995</u> | <ul style="list-style-type: none">■ <u>+rtlCommentPragma</u> |
|--|---|--|

-clean

Purpose

Deletes all IXCOM generated files and directories.

Category

Compiler option

Syntax

`-clean`

-includir

Purpose

Specifies the path for the include files.

Category

Compiler option

Syntax

`-includir <path>`

-relativeIXCDIR

Purpose

Enables relative path for `IXC_DIR` in `xc_work/hdl.var`.

Category

Compiler option

Syntax

`-relativeIXCDIR`

Description

`IXC_DIR` in `xc_work/hdl.var` specifies the full path (absolute path) for the design directory for IES. When you specify this option, instead of using absolute path, relative path is used. This enables you to move the compiled database from one location to another directory for running IES simulation.

This option might conflict with the `-svroot <root_path>` option of the `ncsim` command because the `-svroot` option prefixes the specified root path from `-sv_root` to the relative path specified for `-sv_lib`. In this case, IES modifies the IXCOM-generated path for `libixcg.so` in `hdl.var`, thereby causing path conflict.

-m

Purpose

Specifies to compile axis library.

Category

Compiler option

Syntax

`-m`

-o

Purpose

Use with -d to output to a file.

Category

Compiler option

Syntax

`-o <file>`

+delVhdlEntity

Purpose

Deletes the VHDL entity/configuration database files from prior compilation on name collision.

Category

Compiler option

Syntax

`+delVhdlEntity`

+no_strnInst

Purpose

Specifying this option to IXCOM disables the generation of `X_STR*` primitives. The strengths on the primitives and continuous assigns are as-is passed to the HDL Compiler, which ignores these primitives. This option can be used as a workaround in cases where the functionality of the strength specification on primitives is not required, and continuous assigns and the generation of the `X_STR*` primitives is leading to xeCompile errors.

Category

Compiler Option

Syntax

`+no_strnInst`

-top

Purpose

Specifies the module at the top of the design hierarchy (usually the testbench).

Category

Compiler option

Syntax

-top [*<library.>*]*<top-design-unit-identifier>*

Parameters

<i>library</i>	The logical library containing the entity, configuration, or module name
<i>top-design-unit-identifier</i>	For VHDL, the syntax of this parameter is: <i>entity</i> [<i>(architecture)</i>] <i>configuration</i> [<i>(architecture)</i>]
	For Verilog, the syntax of this parameter is: <i>module</i>

Description

Specifies the module at the top of the design hierarchy (usually the testbench).

As shown in the Parameters table above:

- For VHDL, specify the entity or configuration name. You might also specify an architecture name; otherwise, the last compiled architecture is used.
 - For Verilog, specify the module name.
 - For SystemC, specify the module name.

UXE Command Reference Manual

Commands and System Tasks for SA Mode

You can use this option more than once for multiple top-level modules. If the *library* parameter is not specified, the library specified with the `-work` option is searched; otherwise, the default library `WORK` is searched.

Using this option indicates that you are compiling a mixed language or VHDL design. This option is not used for pure Verilog designs.

-L

Purpose

Enables you to search libraries for modules.

Category

Compiler option

Syntax

`-L <logical_library>`

Description

When instantiating VHDL or Verilog components, lets you specify different logical libraries in which to search for modules. By default, only the parent library is searched when no logical library is specified in the VHDL instantiation.

+bc

Purpose

Enables behavioral compilation for a specific module in the DUT. Invoking behavioral compilation extends the RTL Verilog subset that can be synthesized.

Category

Compiler option

Syntax

`+bc+<module> [+<module...>]`

Note: Alternatively, add the `map_bc` directive in an `initial` statement within the module as shown below:

```
$ixc_ctrl("map_bc");
```

Description

When a Verilog module is not compliant with the RTL Verilog subset, it would normally not be synthesized. The `+bc` (Behavioral Compiler) option extends the RTL Verilog subset that can be synthesized. For a list of behavioral constructs that are supported in the DUT with the `+bc` option, refer to the *Handling of Behavioral Constructs During Behavioral Compilation* section of the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter in *UXE User Guide*.

+bcc

Purpose

Enables behavioral concurrent compilation for a specific module in the DUT.

Category

Compiler option

Syntax

+bcc+<module> [+<module...>]

Note: Alternatively, add the map_bcc directive in an initial statement within the module as shown below:

```
$ixc_ctrl("map_bcc");
```

+bcAsyncEvent

Purpose

Categorizes all behavioral events as asynchronous.

Category

Compiler option

Syntax

+bcAsyncEvent

+bcDebug

Purpose

Enables debug mode profiling for behavioral module.

Category

Compiler option

Syntax

+bcDebug

Description

This option is useful for fine-tuning behavioral performance. For example, instrumentation would be added to count the number of behavioral evaluation that each behavioral event incurs.

-32 | -ixcg32 | -64

Purpose

Enables 32-bit or 64-bit compilation. The default option is **-64**.

Category

Compiler option

Syntax

-32
-ixcg32
-64

Description

The **-64** option to the compiler enables 64-bit mode for 64-bit operating systems. This option is useful for HDL designs that require 4 GB or more of memory in order to compile. 64-bit compilation supports the GCC G++ 4.4.0 C and C++ compilers. With the **-64** option, IXCOM invokes all the tools like **iRun**, HDL Compiler, and xeCompile during compilation in 64-bit mode. The compiled databases for the emulator and the simulation snapshot are also generated in 64-bit mode.

When you use the **-32** option, the INCISIVE tools like **iRun**, ncelab, ncsim, and so on, are invoked in 32-bit mode, and the simulation snapshot is generated in 32-bit mode. The HDL Compiler and xeCompile tools are invoked in 64-bit mode, and the emulation database for Palladium Z1 emulator is generated in 64-bit mode.

With the the **-ixcg32** option, the INCISIVE tools like **iRun**, ncelab, ncsim, and so on, are invoked in 32-bit mode, and the simulation snapshot is generated in 32-bit mode. The difference in the usage of the **-32** and **-ixcg32** options is that with **-32**, IXCOM is invoked in 32-bit mode; with **-ixcg32**, IXCOM is invoked in 64-bit mode. In all cases, the HDL Compiler and xeCompile tools are invoked in 64-bit mode, and the emulation database for Palladium Z1 emulator is generated in 64-bit mode.

+asis

Purpose

Retains the case of VHDL identifiers in the design. This is the default behavior.

Category

Compiler option

Syntax

+asis

-relax

Purpose

Enables module binding with relaxed restrictions.

Category

Compiler option

Syntax

`-relax`

Description

When the `-relax` option is specified while binding VHDL and Verilog modules, IXCOM searches for the specified VHDL module in all the libraries specified in the `libmaprc` file. There are specific rules and order of search for design binding. For details, refer to the *Design Binding and Search Rules in IXCOM* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

-featureFile

Specifies the file that lists the features to be enabled or disabled for an instance hierarchy, module hierarchy, and/or HDL file. This option can be specified only with the `ixcom` command; it cannot be specified with the `vlan` or `vhan` command.

Category

Compiler option

Syntax

```
-featureFile <feature_file>
```

Description

In the `<feature_file>` file, you need to list the set of features that you want to enable or disable, along with the configuration options in the form of `<feature_spec_lines>`. Each line identifies the `<feature>`, the corresponding `<control_option>`, and the configuration to which this control is applied based on the `<configuration_option_opt>`.

Following is the syntax of the `<feature_spec_line>`:

```
<feature> <control_option> <configuration_option_opt>
```

- Following are the valid values for the `<feature>` parameter:

```
sva  
psl  
assert  
coverage_functional  
coverage_block  
coverage_toggle  
rtlchk
```

Note: For `rtlchk` option, only `design_unit_spec` is supported with optional depth. In addition, if a module is marked with `rtlchk off` in `<feature_file>` file, RTL checking is skipped for that module within the entire design tree.

- Following are the valid values for the `<control_option>` parameter:

```
enable  
disable
```

- Following are the valid values for the `<configuration_option_opt>` parameter:

```
instance=<instance_spec>{,<instance_spec>}  
<instance_spec> ::=  
    <hierarchical_instance_path>[+<depth>]
```

UXE Command Reference Manual

Commands and System Tasks for SA Mode

```
du=<design_unit_spec>{,<design_unit_spec>}
<design_unit_spec> ::= 
    <design_unit_name>[+<depth>]
file= <filename>{,<filename>}
```

For details about using the `-featureFile` option, refer to the *Enabling and Disabling Features at Compilation Time* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

+tbcnba

Purpose

Enables generated code to read variables that can be assigned in a non-blocking way from outside of the process.

Category

Compiler option

Syntax

+tbcnba

Description

When two `always` processes wake up at a clock edge, the correct behavior is to display the old value of the signal, that is, the value of the signal before it is updated. However, by default, when you use the `ixc tbcall_region` pragma, the updated value of the signal is displayed. To ensure that the generated code reads the value of the variables before these are updated, use the `+tbcnba` option of the `ixcom` command.

+dpiprofsamplecnt

Purpose

Changes the number of samples that the profiler should consider for the DPI calls.

Category

Compiler option

Syntax

`+dpiprofsamplecnt=<N>`

Description

Changes the number of samples that the profiler should consider for computing the average time taken by the DPI calls. By default, the value of the sample count `<N>` is set to 100.

-externalNetlists

Purpose

Helps import external netlists in the SA mode using the IXCOM compiler.

Category

Compiler option

Syntax

```
-externalNetlists <file>
```

Description

The `-externalNetlists <file>` option of the `ixcom` command enables you to import the netlist files as per the specification in the configuration file. The `<file>` is the configuration file in which you need to specify the following information:

- List of netlist files
- List of netlist design units to import (bind to RTL design hierarchy)
- List of RTL design units to export (bind to netlist design hierarchy)
- List of references to external netlist objects
- Hot-swap for the netlist hierarchy
- Netlist module definition to be kept if the definition is present in both the netlist and RTL domains

Refer to the *Importing External Netlists into IXCOM Compiler* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide* for details about the mechanism for importing external netlists into the IXCOM compiler.

To specify the netlist information in the configuration file, use the following configuration file keywords:

```
NL_FILE, NL_FILE_HWONLY, NL_FILE_SWONLY, NL_COMPILE, NL_REFLIB, NL_IMPORT,  
NL_EXPORT, NL_BIND, NL_REF, NL_HOTSWAP, NL_DEF_KEEP, NL_HDLICE_REF,  
NL_NCVLOG_OP, and NL_INFO
```

Note: This document uses uppercase names for keywords in the configuration specification file. However, these are case-insensitive keywords.

You can specify the configuration file items multiple times in the configuration file. To specify line comments, use "://" or "#" at the beginning of the line.

The syntax of the keywords is as follows:

`NL_FILE <file> [<file> ...]`

Specifies the list of Verilog netlist files.

With the `NL_FILE` keyword, you can specify the absolute or relative paths of the files. You can also use environment variables, similar to support available for the `-f <file>` option.

`NL_FILE_HWONLY <file> [<file> ...]`

Specifies the list of Verilog netlist files for use in hardware compilation only.

The `NL_FILE_HWONLY` keyword enables you to provide a set of netlist files to be used only for hardware compilation. This specification uses the same syntax as the `NL_FILE` keyword.

`NL_FILE_SWONLY <netlist file(s)>`

Specifies the list of Verilog netlist files for use in software compilation only.

The `NL_FILE_SWONLY` keyword enables you to provide a set of netlist files to be used only for software compilation. This specification uses the same syntax as the `NL_FILE` keyword.

`NL_COMPILE all | <file> [<file> ...]`

Specifies the netlist file compile options - either all of the files specified in the `NL_FILE` and `NL_FILE_HWONLY` keywords or only the specified list of files.



You must specify netlist files using `NL_FILE`, `NL_FILE_HWONLY`, or `NL_FILE_SWONLY` to enable netlist compile using `NL_COMPILE`. All other specification items are optional when using `NL_COMPILE`.

`NL_REFLIB <lib> [<lib> ...]`

Specifies the list of precompiled netlist library names.

All specification items are optional when using `NL_REFLIB`. The compiler will extract netlist files from `NL_REFLIB` specified libraries if `NL_FILE` is not specified.

UXE Command Reference Manual

Commands and System Tasks for SA Mode

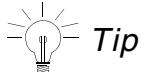
```
NL_IMPORT <design_unit>
[ <hier_path_to_design_unit_instance>
[,...]
```

Specifies the list of design units to import from netlist into RTL hierarchy.

Include at least one NL_IMPORT specification item in a configuration file if netlist compile is not enabled using NL_COMPILE or NL_REFLIB. All other specification items are optional depending on the requirements for the design.

```
NL_EXPORT <design_unit> [ <hierarchical path to design_unit instance>
[,...]
```

Specifies the list of RTL design units to export into netlist hierarchy.



NL_IMPORT and **NL_EXPORT** specification can have optional information about the hierarchical path to the <*design_unit*> instance.

```
NL_BIND <file> [ <file> ... ]
```

Specifies the list of SystemVerilog bind directive files to insert RTL module instances into netlist hierarchy.

```
NL_REF <wire_or_reg>
[ <packed_dimension> ] \
<hier_path_to_external_netlist_object>
[ <unpacked_dimension> ]
<wire_or_reg> ::= wire | reg
<packed_dimension> ::= <dimension>
<unpacked_dimension> ::= <dimension>
<dimension> ::= [ <integer_value> ] |
[ <integer_value> : <integer_value> ]
```

Specifies the hierarchical references to netlist objects.

```
NL_HOTSWAP on | off
```

Specifies the hot-swap option. The default value is off.

The NL_HOTSWAP keyword retains the last value if it has been specified multiple times. If it has not been specified, by default, it is set to off.

```
NL_DEF_KEEP ixcom | external
```

Specifies the resolve option for design units defined in both netlist and RTL domains. The default value is external.

The NL_DEF_KEEP keyword retains the last value if it has been specified multiple times. If it has not been specified, by default, it is set to external.

```
NL_HDLICE_REF init_zero | init_default
```

UXE Command Reference Manual

Commands and System Tasks for SA Mode

Specifies the state object initialization in the simulation model for the HDL-Compiler reference cells. The default value is `init_default`.

The `NL_HDLICE_REF` keyword retains the last value if it has been specified multiple times. If it has not been specified, by default, its value is set to `init_default`. This specification has no effect if the hot-swap feature is disabled.

`NL_NCVLOG_OPT <option> [<option> ...]`

Specifies the list of `ncvlog` options (if needed) to compile netlist files for hotswap.

The `NL_NCVLOG_OPT` options must be allowed `ncvlog` options. This specification has no effect if hot swap feature is not enabled.

`NL_INFO verbose`

Enables verbose information messaging for netlist compilation.

-ignoreInitialFinal

Purpose

Ignores the initial and final constructs in the entire design or the specified list of design units.

Category

Verilog Analyzer Option

Syntax

```
-ignoreInitialFinal {du=all | <du_list>}
```

Description

This option is used to ignore the initial and final constructs in the design. These constructs can be ignored in the entire design unit `du=all` option or from a specified list of design units by using the `du=<du_list>` option. The syntax and allowed values for `<du_list>` include one or more design unit names separated by a comma without any whitespace. This option requires the argument value with `du=` prefix. This option can be specified multiple times on the command line.

Note: This option is intended for ICE designs to be migrated to SA mode by using the IXCOTM compiler. In the ICE mode, usually the Verilog initial constructs (blocks or statements) are not very important in the design. By default, the HDL Compiler ignores the initial construct as a part of design analysis. The IXCOTM compiler, however, needs to honor the initial constructs as per language requirements.

While migrating ICE designs to the SA mode, the initial constructs in the design can lead to design compile error for one or more of the following reasons:

- ❑ Reference to hierarchical paths that do not exist in the design - IXCOTM compiler requires all hierarchical paths to be resolved at elaboration time.
- ❑ Undefined user tasks or functions - ICE designs might not have source or model for such tasks or functions. Therefore, you need to identify and create shell models for such situations.

 *Important*

This option must be used only for the design bring-up in above contexts. For SA mode, these options are not recommended.

These options are applied to only user HDL. These options do not apply to:

- Compiler inserted file from tool install location or release area (`ixc_time`, ...)
- Files that are specified on the command-line (using the `-v` option for example), but are actually files generated by the tool or any internal modules inserted by the compiler in the post-analysis phase (elaboration, code generation, and so on.)

Note: For most SA designs, the initial and final constructs cannot be ignored in design units that generate clock, and so on. See the `-keepInitialFinal` option to selectively keep the initial and final constructs in the design when using this option.

 *Caution*

The compiler determines the installation location using its location. The references (including copies) to files in the location outside of the current installation location for the tool is considered as user HDL. So, using the -ignoreInitial du=all option might remove the initial blocks in such HDL with incorrect simulation results.

Examples

```
ixcom -ignoreInitialFinal du=all <verilog_source_files>
```

The above command ignores the initial and final constructs in all modules.

```
vlan -ignoreInitialFinal du=all <verilog_source_files>
```

The above command ignores the initial and final constructs in all modules.

-keepInitialFinal

Purpose

Keeps initial and final constructs in the specified list of design units.

Category

Verilog Analyzer Option

Syntax

```
-keepInitialFinal du=<du_list>
```

Description

This option is used to keep the initial and final constructs in the specified list of design unit. The syntax and allowed values for *<du_list>* can include one or more design unit names separated by a comma delimiter character without any whitespace. This option requires argument value with *du=* prefix. This option can be specified multiple times in the command-line.

Note: This option must be specified with *-ignoreInitialFinal* option, and the following usage rules apply:

- ❑ The *-keepInitialFinal* option has no effect if the *-ignoreInitialFinal* is not specified in the command line.
- ❑ The design unit specified with the *-keepInitialFinal* option takes precedence over the *-ignoreInitialFinal* option specification.

See [-ignoreInitialFinal](#) for more details

Examples

```
ixcom -ignoreInitialFinal du=Dut \
      -keepInitialFinal du=Tb <verilog_source_files>
```

The above command ignores the initial and final constructs in the Dut and keeps the initial and final constructs in the Tb.

```
vlan -ignoreInitialFinal du=Dut1,Dut2 \
      -keepInitialFinal du=Tb <verilog_source_files>
```

UXE Command Reference Manual

Commands and System Tasks for SA Mode

The above command ignores the initial and final constructs in Dut1 and Dut2 and keeps the initial and final blocks in Tb.

-conformal

Purpose

Generates the LEC script for helping in equivalence checking of the IXCOM-generated RTL and HDL-Compiler-generated netlist.

Category

Compiler option

Syntax

`-conformal`

Description

Generates the LEC scripts for equivalence checking of HDL-compiler-generated netlist against IXCOM-generated RTL. This script closely matches the script that is generated by HDL Compiler. This script includes the LEC commands customized for the user design and is used to invoke LEC after the HDL Compiler has finished synthesizing the design.

+margdpi

Forces all DPI to use the marg interface.

Category

Compiler option

Syntax

+margdpi

+margdpisize

Specifies whether data in `tb_export` is sent through memory or PIO based on the width of the data.

Category

Compiler option

Syntax

`+margdpisize+<number>`

Description

The `+margdpisize+<number>` is used to control memory-based argument transfer. The default cutoff from PIO to MEM is 512 bits for any argument of the task. Use `<number>` to change this cutoff.

This limit is for any single argument and not the combined width of all arguments. The instrumentation is done per task and not per argument basis. So, you can have multiple arguments of a task each of `<number-1>` size, and the task still uses PIO transfer instead of memory. Also, if a task has any one argument width greater than `<number>`, all arguments of the task will be mapped to memory instead of PIO.

If you use the `+margdpi` option with the `+margdpisize` option, the `+margdpisize` option is ignored. All DPI is forced to use `marg` interface, irrespective of the argument size. So, the `+margdpisize` option should not be used with the `+margdpi` option.

The `+margdpisize` option does not impact any DPI task for which the `$ixc_ctrl` system task already has `tb_mem`. These tasks are mapped to memory irrespective of size specified by `+margdpisize` option.

+margdirect

Specifies the maximum size of the data to be sent using memory ports directly in `tb_import` calls.

Category

Compiler option

Syntax

`+margdirect+<N>`

Description

Using memory resources to transfer data for large-size arguments in `tb_import` calls, requires multiple memory ports to be used in the same step. The number of memory ports that need to be scheduled in a step for copying the data to the memory is proportional to the data size. So, large-size data (more than 2K bits) can lead to bottlenecks and higher step-count, or even compilation failure. To avoid these issues, IXCOM uses a constant number of memory ports over multiple steps to copy the data. However, this requires virtual logic to be added, which might in turn, incur gate-count overhead. The `+margdirect` option enables you to control the trade-off between gate-count versus usage of memory ports in the design. The argument `<N>` specifies the maximum argument size for which no virtual logic will be added by interface compiler and data will be copied to memory using concurrent memory ports in the same step.

Note: The `+margdirect` option is effective only with `tb_mem`; it has no effect with `tb_pio`.

+xmrTfInlineOn

Purpose

Enables XMR task and function inlining for the specified module(s) in DUT. If no module is specified, the `xmrTfInline` optimization is enabled for all modules in DUT.

Category

Compiler option

Syntax

```
+xmrTfInlineOn [+<module_name>...]
```

Alternatively, add the `xmrTfInlineOn` directive in an `initial` statement within each specified module as shown below:

```
$ixc_ctrl("xmrTfInlineOn");
```

Description

The `xmrTfInline` optimization attempts to inline tasks and functions in the specified modules to the module calling them. The `xmrTfInline` optimization is automatically enabled for modules with the `xmrTfInlineOn` directive of the `$ixc_ctrl` task. Use this `ixcom` command option to enable the `xmrTfInline` optimization for additional modules or all the modules in DUT.

+xmrTfInlineOff

Purpose

Disables XMR task and function inlining for all modules.

Category

Compiler option

Syntax

+xmrTfInlineOff

Description

The `xmrTfInline` optimization attempts to inline tasks and functions to the module calling them. Use this `ixcom` command option to disable the `xmrTfInline` optimization for all the modules, including those modules with the `xmrTfInlineOn` directive of the `$ixc_ctrl` task.

+mapInit

Purpose

Enables mapping of qualified clocked `initial` statement to hardware for the specified module(s) in DUT. If no module is specified, mapping is enabled for all modules in DUT.

Category

Compiler option

Syntax

`+mapInit [+<module_name>...]`

Description

When an `initial` statement in DUT is time-consuming, the design cannot be swapped to hardware until the `initial` statement finishes. The `mapInit` optimization attempts to automatically extract clocked `initial` statement to hardware.

+no_mapInit

Purpose

Enables mapping of qualified clocked `initial` statement to HW for all modules in DUT excluding the specified modules.

Category

Compiler option

Syntax

`+no_mapInit+<module_name> [+<module_name>...]`

Description

When an `initial` statement in DUT is time-consuming, the design cannot be swapped to hardware until the `initial` statement finishes. The `mapInit` optimization attempts to automatically extract clocked `initial` statement to hardware. Use this `ixcom` command option to exclude from optimization those modules that contain `initial` statement that should be executed in software or `initial` statement that does not fall in the list of constructs supported with the `-atb` option of the `hdllImport` command.

+moduleStub

Purpose

Enables stubbing out modules.

Category

Compiler option

Syntax

`+moduleStub+<module_name>`

Description

The `+moduleStub` option can be used to stub out modules in the IXCOM compilation. The modules specified with this option are stubbed out of all of the contents, that is all processes, primitives, continuous assigns, instantiations, and so on are removed. The I/Os and vars remain. The content removal is done prior to elaborating that module. An INFO message is displayed to indicate the stubbed-out module. The module is stubbed out in both hardware and software partitions.

+genIfcSideFile

Purpose

Associates the virtual interface variable written in the testbench to its appropriate definition automatically.

Category

Compiler option

Syntax

`+genIfcSideFile`

Description

The `+genIfcSideFile` option directs IXCOM to process the interfaces defined in the design and generate the mapping between the original and transformed (split) interfaces in a side file `ixc_ies_ifc` in the `AxisWork/` directory. This side file contains the information that helps bind the virtual interface with the instance of the interface that was split into a separate definition.

The side file contains the following information:

- Original Name of the interface
- Generated name of the interface
- List of parameter values (positional mapped)

The original interface definitions and generated mapping in the side file are used to bind the virtual interface to the corresponding instance of the IXCOM-generated interface.

-lint

Purpose

Enables RTL check on all the DUT modules. Use this option to see if there are any RTL rule violations.

Category

Compiler option

Syntax

`-lint`

Description

This option enables IXCOM to run RTL checks on all the DUT modules. This option only prints the result of the RTL checks and does not generate any simulation or emulation database.

+maxPIPO

Purpose

Sets the maximum limit of PIs and POs for the design to be compiled.

Category

Compiler option

Syntax

`+maxPIPO+<num>`

Description

Certain designs create a huge number of PI/POs slowing down the compilation process. To avoid this, the maximum number of PI/POs is limited, by default, to 32k. If you want to change this number, use the `+maxPIPO+<number>` option to set another limit for the PI/POs.

+ignoreNCVerCheck

Purpose

Disables the compatibility check for IES and UXE.

Category

Compiler option

Syntax

`+ignoreNCVerCheck`

Description

At run time, IXCOM checks if the IES version that you are using is compatible with the UXE software. If the INCISIVE version is found to be incompatible, an error message is issued mentioning the IES versions compatible with the UXE release that you are using.

If you are sure that the UXE and IES versions are compatible with each other, you can choose to disable this compatibility check by using the `+ignoreNCVerCheck` option with the `ixcom` command.

Note: Bypassing the check ensures that compilation continues. However, at a later compilation or run-time stage, unforeseeable errors might occur.

+uvmDut+<arg>

Purpose

Enables IXCOM compilation for UVM DUT modules.

Category

Compiler option

Syntax

`+uvmDut+<arg>`

Description

This option enables IXCOM to compile designs containing UVM. Note that IXCOM does not natively support UVM. So, even while using this option, enclose pragma, such as `IXC_EMBED_SW` within the UVM part of the design so that IXCOM skips the compilation, and switches to irun. When this option is specified, IXCOM compiles the design differently from a regular IXCOM compilation, as follows:

- In the `ncvlog` target of the `xc_work/xc_make` file:
 - For the first `systemverilog.f` entry, the `ncvlog` command is replaced by an `irun` command, which also includes the `-compile -uvm <arg>` option.
 - For the rest of the `systemverilog.f` entries, the `-compile -uvm -uvmnoautocompile <arg>` option is appended to the `irun` command. This implies that the compilation of UVM packages was done in the first `systemverilog.f` entry, and for the rest of the entries, UVM packages should not be compiled again.
- **Note:** Note that here, `<arg>` is an optional argument that is inferred from the `+uvmDut+<arg>` option. You can use it to pass any argument to the `ncvlog` compilation stage.
- The `-uvm -uvmnoautocompile` options are appended at the end of the `xc_work/irun.f` and `xc_work/irun.alt.f` files.

-incLibFileCompilationUnit

Purpose

Purpose

Enables IXCOM compilation for compilation unit scope objects contained within library files.

Category

Compiler option

Syntax

`-incLibFileCompilationUnit`

Description

Just like with SystemVerilog packages, access to *Compilation Unit Scope* objects declared in the source HDL is allowed from modules compiled from library files (using the `-v` option of the `vlan` or `ixcom` command). However, *Compilation Unit Scope* objects contained within the library files are ignored, by default, and consequently, there is no access to such *Compilation Unit Scope* objects from modules compiled from library files. To ensure that these objects are not ignored, specify the `-incLibFileCompilationUnit` option of the `ixcom` or `vlan` command.

-generic

Purpose

Overrides the value of a VHDL generic from the `ixcom` command line.

Category

Compiler option

Syntax

```
-generic "<generic_name> => <value>"
```

Description

This option can be used with any mixed-language designs for any combination of Verilog, VHDL, and SystemC hierarchies, such as Verilog->VHDL->VHDL->Verilog, VHDL->Verilog->VHDL->VHDL->Verilog->VHDL->SystemC, and so on. The generic overridden through this option, however, should belong only to a VHDL design unit.

Specify the full path for the generic as it appears in the VHDL source as `<generic_name>` to override the generic from the command line, using '.' (period) as the hierarchy separator.

The `<value>` argument specifies an appropriate value for the declared data type of the generic. This overridden value has the highest precedence as it overrides the default generic value declared in the entity, the value passed in the instance declaration or through any configuration declaration or specification. A value can be passed to a generic of an instance at any level of the design hierarchy. The default or initial value expression in the original VHDL is optional. However, if the default value is left out in the original HDL, it is mandatory to specify the value using the generic override command-line feature.

For details, refer to the *Support for Overriding VHDL Generics from the IXCOM Command Line* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

The order in which the generics are specified does not matter unless you are specifying two values for the same generic. In that case, the value specified with the last `-generic` option on the command line takes precedence, and overrides the former value. Also, a warning message is generated in verbose mode to highlight this usage of the `-generic` option.

The `-generic` command-line options can also be included in an arguments file that you can include using the `-f` option of the `ixcom` command.

+lower

Purpose

Converts VHDL identifiers into all lowercase before storing in the database.

Category

Compiler option

Syntax

+lower

+upper

Purpose

Converts VHDL identifiers into all uppercase before storing in database.

Category

Compiler option

Syntax

+upper

+pd3legacy

Purpose

Useful when migrating a design from ICE mode to SA mode. It enables the memories modeled in a style that is supported in the HDL Compiler to function the same in the SA mode.

Category

Compiler option

Syntax

+pd3legacy

-append_log

Purpose

Appends the log to the existing log file.

Category

Compiler option

Syntax

`-append_log`

-h[elp]

Purpose

Displays a listing of all compiler options relating to mixed HDL compilation.

Category

Compiler option

Syntax

`-h [elp]`

-work

Purpose

Specifies the default logical *library* when the *library* parameter is not used with the *-top* option.

Category

Compiler option

Syntax

`-work <library>`

Example

Following is an example of the `-work` option:

```
ixcom -top tb_lib.top -top exports -work worklib
```

Searches library *worklib* for module *exports* because this *-top* entry did not include a library.

Note: Use the `libmaprc` file to map the logical library to its physical library directory.

-rtlchk

Purpose

Enables control over the automatic design partitioning of the design into the hardware (RTL) side and the software (behavioral) side by IXCOM.

Category

Compiler option

Syntax

```
-rtlchk [warn|error|off]
```

Description

Following are the sub-options available to the `-rtlchk` option:

- `warn` - The non-synthesizable design constructs are reported, but compilation is not interrupted. The module containing the non-synthesizable constructs along with the hierarchy of modules under it is not pushed to the software side. The module is handled by HDL Compiler. If the `-enableEmbeddedSW` option is used with the `-rtlchk warn` option, the compiler issues a warning that if subset checking fails, the module will be pushed to the software partition.
- `error` - For any non-synthesizable design constructs, errors are generated and compilation is aborted. This is the default option.
- `off` - Disables automatic partitioning of the design. IXCOM passes all the modules under the `+dut+<mod_name>` hierarchy to the HDL compiler without conducting any RTL synthesis checks. Then, if HDL compiler detects any non-synthesizable modules, it generates an error message and aborts compilation.

-enableEmbeddedSW

Purpose

This option enables the embedded software mode in IXCOM. The non-synthesizable module will be pushed to software when it fails the RTL checks.

Category

Compiler option

Syntax

`-enableEmbeddedSW`

Description

The non-synthesizable design constructs are reported, but compilation is not interrupted. The module containing the non-synthesizable constructs along with the hierarchy of modules under it is pushed to the software side.

This option can be used only with the `-rtlchk warn` option. It has no effect when the `-rtlchk` argument is `error` or `off`.

+no_rtlchk

Purpose

Skips the automatic partitioning for a single module or set of modules of the design.

Use this option for a module or set of modules that might work correctly on the emulator, but the automatic design partitioning pushes the module to the software partition.

Note: More than one *<module_name>* arguments can be specified with the +no_rtlchk option.

Category

Compiler option

Syntax

+no_rtlchk+<*module_name*> [+<*module_name*>...]

-ncsim

Purpose

Enables the Cadence NCSim-compatible library search mechanism.

Category

Compiler option

Syntax

`-ncsim`

Description

Enables the Cadence NCSim-compatible library search mechanism. The VHDL or Verilog components that cannot be located in libraries that were specified with the `-L` option, are then searched in the libraries specified in the `libmaprc` file. The libraries are searched sequentially in the order in which the libraries are specified.

Note: Use the `-relax` option with the compiler when you use this option.

-nccompile

Purpose

Stops compilation after the *i run* compilation (parse) stage. This option enables you to compile the design using IXCOM for separate elaboration using IES.

Category

Compiler option

Syntax

`-nccompile`

Description

Use `-nccompile` to compile the design using IXCOM, for later elaboration using INCISIVE tools. Individual INCISIVE tools - *ncvlog* (or *ncvhdl*), *ncelab*, and *ncsim* are used to compile and elaborate the testbench and other non-synthesizable modules with the pre-compiled design. This option is helpful in compilation use models like IXCOM/Specman **e** (Elite) and IXCOM/TBA.

Note: The `-nccompile` option is similar to the `+xcDesignTop` option, but enables you to pre-compile the design without having to specify design hierarchy within the testbench.

-covdut

Purpose

Specifies the top-level design unit for code coverage.

Category

Compiler/Coverage option

Syntax

```
-covdut <dut_module>
```

Description

Specifies the top-level design unit for code coverage. If the design unit does not exist in the design, an error message is generated. It does not apply to functional coverage.

All of the design units instantiated within the *<dut_module>* are marked for coverage. You can specify the `-covdut` options multiple times to specify different top-level design units.

It is better to use the more powerful `-featureFile` option instead the `-covdut` option. It is not recommended that a mix of the `-featureFile` and `-covdut` options should be specified with the same `ixcom` compile command.

-covfile

Purpose

Specifies the file that lists the coverage-specific options sent directly to IES. Only for functional coverage.

Category

Compiler/Coverage option

Syntax

`-covfile <file>`

Description

Specifies the file that lists the coverage-specific options sent directly to IES. This option is recognized only for functional coverage items tracked by the simulator, not by IXCOM.

+covGenFeatureFile

Purpose

Helps identify sections of the design that can be treated as library cells and disable coverage for those sections.

Category

Compiler/Coverage option

Syntax

```
+covGenFeatureFile [+<minInsts>]x[<maxIOs>]
```

Description

When the `+covGenFeatureFile` option is specified, the IXCOM compiler creates a feature file that can be used to disable all coverage (including functional) for design units that are instantiated greater than or equal to `<minInsts>` times and have at most `<maxIOs>` port signals. The default value of `<minInsts>` is 250. The default value for `<maxIOs>` is 10.

The output file is called `covGenFeatureFile.ff` and it is placed in the design directory. Each design unit that meets the criteria is listed with each coverage type explicitly disabled. The format of this file follows the format rules of the `-featureFile` option of the `ixcom` command. Rename this file with a meaningful name and customize it as per design requirements. The resulting file can then be used as an input to IXCOM to disable coverage on the unwanted design units.

Following is a sample of a `covGenFeatureFile.ff` file:

```
# Automatic feature file covGenFeatureFile created with:  
#     - Minimum Instances = 250  
#     - Maximum IOs      = 10  
coverage_block disable du=mux2  
coverage_toggle disable du=mux2  
coverage_functional disable du=mux2
```

An error message is issued if invalid arguments are used with the `+covGenFeatureFile` option or if the `covGenFeatureFile.ff` file already exists. This is to prevent accidental deletion of the file on subsequent compiles.

UXE Command Reference Manual

Commands and System Tasks for SA Mode

A warning message is issued and the `covGenFeatureFile.ff` file is not created if no design units meet the criteria.

This file can be generated without running the entire compile flow. Add the `-target ixvcg` option of the `ixcom` command to exit the compile flow immediately after creating the `covGenFeatureFile.ff` file.

+ccovModelSummary

Purpose

Enables creation of two comma-separated value (.csv) files to help you identify the portions of the design that should be considered for code coverage. Only for toggle and block coverage.

Category

Compiler/Coverage option

Syntax

`-covfile <file>`

Description

Enables creation of two comma-separated value (.csv) files to help you identify the portions of the design that should be considered for code coverage. The files are created as follows:

- `<design_dir>/ccov_model_summary_inst.csv`
- `<design_dir>/ccov_model_summary_type.csv`

You can import these .csv files into a spreadsheet tool for browsing through the design to see how code coverage items are distributed. This is particularly useful when the available space on the emulator is limited and you need to prioritize the portions of the design to be covered.

+ccovProfile

Purpose

Enables tracking of statistics related to code-coverage processing. Only for toggle and block coverage.

Category

Compiler/Coverage option

Syntax

+ccovProfile

Description

Enables tracking of statistics related to code-coverage processing. For example the time spent in processing code coverage at compile time is reported, such as:

```
Code Coverage - End: cpu time = 1.24 sec, elapsed time = 2 sec  
heap memory size = 0.00M bytes
```

This output is included in the `ixcom.log` file.

Note: The counter used by this feature impacts other profiling counters used in the IXCOM compilation.

-covAddResetLogic

Purpose

Enables reset of toggle, block, and assertion counters at run time in hardware.

Category

Compiler/Coverage option

Syntax

`-covAddResetLogic`

Description

This option is required to get correct reset behavior when the design cannot be swapped to simulation.

Without this option, the reset can be very slow when there is a large number of coverage items. However, this option can impact the size of the design database. Therefore, use it only if you require reset functionality at run time.

The use of a hardware reset on the counters requires advancement of time before the command is executed and the reset occurs.

+fcovCoverCnt+<N>

Purpose

Specifies the minimum number of bits [default: N=8] to be used for the bin counters of covergroups in the DUT. For more information on functional coverage using SystemVerilog covergroups, refer to the *Coverage Support* chapter of the *UXE User Guide*.

Category

Compiler/Coverage option

Syntax

+fcovCoverCnt+<N>

Description

Valid values for N include 1, 4, 8, 16, and 32. When values other than these are specified, IXCOM yields a compile-time error. Once an individual counter reaches its maximum value, the counter stays fixed at that value. Note that using a value higher than 8 can impact simulation performance when creating the Unicov database.

Note: This is a minimum value. So, the actual size of the counter on the hardware might be larger if required by other coverage constructs. For example, a coverpoint that feeds into a cross, or when an `at_least` option requires more bits.

-f|-F

Purpose

Specifies an *<options_file>* containing additional options and/or source files.

Category

Compiler option

Syntax

<options_file>

Description

Specifies the name of the options file. The *<options_file>* can contain:

- A list of pathnames for Verilog source files
- Any of the following analysis-time options:

-f
-F
-v
-y

- Any of the following +plusargs:

+incdir
+libext
+libnonamehide
+liborder
+libsrescan
+maxdelays
+mindelays
+nowarnTFMPC
+pathpulse
+pulse_e
+pulse_r
+typdelays

Note:

- You cannot include C source files or object filenames in the *<options_file>*.
- There is a difference between the -f and -F options. When you use -f *<options_file>*, the options are read from a file that is relative to directory from

UXE Command Reference Manual

Commands and System Tasks for SA Mode

which IXCOM was invoked. When you use `-F <options_file>`, the options are read from a file that is relative to the directory in which `<options_file>` is located.

+bcov[+<design_unit_or_file_name>]

Enables block coverage and indicates the design units or source HDL files that are to be tracked for block coverage.

- If you do not specify any arguments, all design units in the hardware partition are targeted for block coverage. This might lead to a large capacity requirement on the emulator, and is not recommended.
- If you specify a design_unit as the argument, the block-coverage information is monitored for each instance of that design unit.
- If you specify a file_name as the argument, the block-coverage information is monitored for all design units declared in that file. Note that the file_name must be just the base file name, without any paths, and must be an exact match.

You can specify the names of multiple design units or files by delimiting each name with a '+' character with a single +bcov+ option, or by using multiple +bcov+ options.

+bcovExclude+<design_unit_or_file_name>

Excludes the specified design unit or file from block coverage collection. This option is useful in removing low-level cell libraries from coverage collection. Note that the <*file_name*> must be just the base file name, without any paths, and must be an exact match.

You can specify the names of multiple design units or files by delimiting each name with a '+' character with a single +bcovExclude+ option, or by using multiple +bcovExclude+ options.

+tcov[+<design_unit_or_file_name>]

Enables toggle coverage and specifies the design units to be tracked for toggle coverage.

- If the <*design_unit*> is not specified, the toggle-coverage information is monitored for all design units.
- If a <*design_unit*> is specified, the toggle-coverage information is monitored for each instance of that design unit.
- If you specify a <*file_name*> as the argument, the toggle-coverage information is monitored for all the design units declared in that file. Note that the <*file_name*> must be just the base file name, without any paths, and must be an exact match.
- More than one <*design_unit*> can be specified either by prefixing each <*design_unit*> name with a '+' character, or by using multiple +tcov switches with the ixcom command.

Note that enabling toggle coverage on the entire design is not recommended because of performance issues. Refer to the *Performance Considerations for Coverage Support* section in the *Coverage Support* chapter of the *UXE User Guide* for details.

+tcovExclude+<design_unit_or_file_name>

Excludes the specified *<design_unit>* or *<file_name>* from the toggle-coverage calculation. This option helps in easy exclusion of low-level cell libraries from coverage calculation.

The *<file_name>* should be the base file name, without any paths and must be an exact match. You can specify more than one *<design_unit>* or *<file_name>* by prefixing each one with a '+' character.

+tcovTypes+<type>

Enables you to specify the types of variables to be tracked:

- **p** - covers all the ports
- **v** - covers all the Verilog and VHDL variables
- **n** - covers all the Verilog nets and VHDL signals

Note: It is recommended that only a small number of modules should be covered when the **n** type of variables are used. A large number of nets being covered increases the burden on the Palladium XP compiler.

You can specify multiple types of variables together, like `+tcovTypes+vpn`. The default value is **p**. If you specify this switch more than once, all the specified types are honored.

If you specify any other value than **p**, **v**, or **n**, an error message is issued.

+tcovMDA[+<exponent>]

Purpose

Enables support for SystemVerilog MDA signals for toggle coverage where total size is less than or equal to $2^{<\text{exponent}>}$. By default, the $<\text{exponent}>$ is 14, which is 16k bits. The maximum supported value is 20, which is 1M bits.

Description

The `+tcovMDA` option enables toggle-coverage collection for SystemVerilog multi-dimensional arrays whose total size is less than or equal to $2^{<\text{exponent}>}$ number of bits, and that are fully packed. Arrays that have any unpacked portion are not supported.

By default, $<\text{exponent}>$ is 14. Specify the $<\text{exponent}>$ as a value between 0 and 20 to modify the maximum size. Values beyond 14 move into the "knee of the exponential curve", impacting compile time and emulation performance.

This option is equivalent to the IES coverage file option, `set_toggle_scoring -sv_mda`.

Example

```
reg [3:0][1:0] small; // 8 total bits, supported
reg [255:0][255:0] big; // 65,536 total bits, supported if exponent >= 16
```

Toggle coverage collection of MDAs is enabled only if the MDA signals, like other toggle signals, follow the rules specified for `+tcovTypes` or `-featureFile` options.

-l

Purpose

Specifies a compilation *logfile*, instead of `ixcom.log` (the default).

Category

Compiler option

Syntax

`-l <logfile>`

Description

Specifies a *logfile*, instead of `ixcom.log`, which is the default. Uses the specified logfile instead of the defaults for `vpp` (`vpp.log`) and `vcg` (`vcg.log`). The default for `ixcom` (`ixcom.log`) cannot be changed.

+delay

Purpose

Honors [default: ignore] delays on the continuous assigns and primitives in the software model of the DUT. All the delays in the testbench are honored by default.

Category

Compiler option

Syntax

`+delay[+<module>][+<module...>]`

Syntax Example

```
assign #3 a = b; // #3 delay simulated with +delay option  
buf #3(out, in, ctrl); // #3 delay simulated with +delay option
```

Parameters

module(s) Enables delays in only specified modules and their instances

Note:

- Affects the Verilog part of your design only
- When compiling for emulator, `+delay` is ignored for DUT scope modules mapped to emulator, because emulator supports only functional simulation. All other modules, that is, testbench scope modules or embedded software modules within the DUT, can support the `+delay` option.
- Some restrictions apply when using `+delay`:
 - Wire declaration delays are supported only if an initial value is specified for the wire.

```
wire #3 a = b; // #3 delay simulated with +delay option  
wire #3 a; // #3 delay ignored, even with +delay option
```
 - Delays on transistor primitives (tran, tranif, pmos, nmos) are not supported.

+rtlIgnBlkDelay

Purpose

Ignores blocking delays in `always` blocks to treat them as synthesizable RTL code.

Category

Compiler option

Syntax

`+rtlIgnBlkDelay`

+saIgnDanglingClock

Purpose

Downgrades to a warning any errors resulting from dangling clock input to flip-flops.

Category

Compiler option

Syntax

+saIgnDanglingClock

+max_error_count

Purpose

Displays only up to *number* errors during compilation.

Category

Compiler option

Syntax

`+max_error_count+<number>`

+nowarn

Purpose

Suppresses the display of the specified *message-type(s)* during analysis, specified by its mnemonic code.

Category

Compiler option

Syntax

`+nowarn+<mnemonic>`

-msgSummary

Purpose

Generates the message summary for each command executed during IXCOM compilation, into a separate log file, with the command name prefixed to the log-file name.

Category

Analyzer and Compiler option

Syntax

`-msgSummary`

Description

Eases analysis of messages by generating an easy-to-read message summary by each analyzer or compiler tool in a separate log file.

In the single-step compilation mode, a single log file is generated with the name `ixcom.msg.log`, for all the commands executed during the IXCOM compilation, that is `vlan`, `vhan`, and `ixcom`.

In the multi-step compilation mode, the name of the log file depends on the name of the executed command. For example, for `vlan`, it is named as `vlan.msg.log`, for `vhan`, it is named as `vhan.msg.log`, and so on.

Example

For example:

```
vhan -msgSummary generates vhan.msg.log  
vlan -msgSummary generates vlan.msg.log  
ixcom -msgSummary generates ixcom.msg.log
```

+no_tran

Purpose

Disables the default support for Verilog transistor-level primitives in the design.

Category

Compiler option

Syntax

`+no_tran`

Description

In the earlier versions, the `+tran` option was used to enable support for Verilog transistor-level primitives, including `pmos`, `nmos`, `rnmos`, `rpmos`, `cmos`, `tran`, `tranif0`, and `tranif1` in software simulation. Whereas, the `+tran_force` option was used to enable `pmos` and `nmos` mapping to the emulator within the hardware partition.

Now, by default, the hardware partition supports mapping of the transistor-level primitives `pmos` and `nmos` to the emulator, and the software partition supports all transistor-level primitives.

Therefore, the `+tran` and `+tran_force` options have been deprecated; and the `+no_tran` and `+tran_relax` options have been provided instead.

+tran_relax

Purpose

Enables specification of all transistor-level primitives within the hardware partition.

Category

Compiler option

Syntax

```
+tran_relax [<module_name>]
```

Description

By default, only `pmos` and `nmos` transistor-level primitives are supported within the hardware partition. The `rpmos`, `rnmos`, `cmos`, `tran`, `tranif0`, `tranif1.` and `vpp` transistor-level primitives are still *not* supported in the emulator. If any of these are found within the hardware partition, an error is generated.

When the `+tran_relax` option is used and IXCOM finds any of the unsupported transistor-level primitives in the hardware partition,

- An INFO message is generated indicating that the primitives might cause mismatches between the software and hardware runs.
- The transistor-level primitives are passed to the HDL compiler with the assumption that the HDL compiler will do the necessary handling during the synthesis process.

The `<module_name>` argument enables specification of all transistor-level primitives for the specified module. Without the `<module_name>` argument, the `+tran_relax` option applies to all the modules.

-P

Purpose

Specifies a VCS format table file for PLI routines.

Category

Compiler option

Syntax

`-P <filename>`

Note: You must also pass this option to `vlan` during compilation.

-p

Purpose

Specifies a Verilog format table file for PLI routines.

Category

Compiler option

Syntax

`-p <filename>`

+tfReadOnly

Purpose

Recognizes that the specified PLI tasks/functions only *read* their parameters (they do not *write* parameters passed to them).

Category

Compiler option for Verilog PLI

Syntax

```
+tfReadOnly[+<task_or_function>] [+<task_or_function...>]
```

Description

This option prevents Signal Analysis (SA) from assuming the PLI task or function is *writing* a signal coming from the DUT. If you do not specify any *task_or_function(s)*, *all* PLI tasks and functions in the design are considered to only read their parameters.

Example

In this testbench code, a PLI task *logF* is used to log the value of a signal inside the DUT to a file:

```
always@ (dut.b1.c1.flag)
    $logF("Flag is active:%d",
          dut.b1.c1.flag)
```

The following example ensures that *\$logF* is only *reading* the signal:

```
+tfReadOnly+logF
```

Without this option, SA assumes *\$logF* might write to *dut.b1.c1.flag* and warns about software-to-hardware cross-module reference writes:

```
Warning: xmr defined in DUT can't be assigned in testbench --
dut_instance_path.dut_signal_name at line in file -- Please use
force for DUT write or $axis_rccoff to guard initialization code.
```

This option prevents Signal Analysis (SA) from assuming the PLI task or function is *writing* a signal coming from the DUT and thus disables the task or function only.

-ua

Purpose

Compiles the design to the emulator for Simulation Acceleration (must be used with the `+dut` option).

Category

Compiler option

Syntax

`-ua`

Note: Use this for designs that you want to emulate or accelerate with the emulator.

+1xua

Purpose

Compiles the design to run in 1X oversampling mode. On the emulator side, 1X mode is a compilation mode that enables the fastest design clock to run at the same frequency as the emulator's FCLK.

Category

Compiler option

Syntax

+1xua

+dut

Purpose

Specifies the top-level entities or module names for the device under test (DUT)

Category

Compiler option for designs that you want to emulate or accelerate with the emulator

Syntax

```
+dut+<module> | <entity> [+<module> | <entity...>]  
+dut+<module> [+<module...>]
```

Syntax Example

```
+dut+chip_top+tx_hs64
```

Description

When compiling for the emulator and to use Simulation Acceleration modes, specifies the top-level entities or module names for the device under test (DUT), which is typically the part of the design outside the testbench, or the *chip*.

This option is required to run the performance profiler in any simulation mode.

+dutignore

Purpose

Specifies the top-level entity or module name to be ignored when running in Simulation Acceleration mode.

Category

Compiler option for designs that you want to emulate or accelerate with the emulator

Syntax

```
+dutignore+<module|entity>[+<module|entity>]...  
+dutignore+<module>[+<module>]...
```

Description

Specifies the top-level entity or module name to be ignored when running in Simulation Acceleration mode. The ignored entities or modules are simulated only when you are in software simulation mode.

The ignored entities or modules are *not* simulated at all while you are running in Simulation Acceleration mode. Furthermore, no additional emulator I/Os are created for the ignored entity or module instances.

This option is typically used to improve simulation performance by ignoring entities or modules that might not be required during Simulation Acceleration modes, for example, monitor or checker entities or modules that look for error conditions during simulation. By swapping back to software simulation mode, you cause these modules to be simulated.

+saIgnXmrWrite

Purpose

Downgrades to a warning all errors resulting from cross-module reference (XMR) writes to Device Under Test (DUT) nets or registers from the testbench while you are running in Simulation Acceleration modes.

Category

Compiler option for designs that you want to emulate or accelerate with the emulator

Syntax

```
+saIgnXmrWrite
```

Description

If you do *not* use `+saIgnXmrWrite`, and if you have the following in your testbench:

```
top.dut.data = 32'h0;
```

You will get the following error:

```
Error: XMR defined in DUT cannot be assigned in testbench
```

If you *do* use `+saIgnXmrWrite` and get a warning, after you have compiled, you should simulate the design in software simulation mode until the desired writes have occurred, before swapping to Simulation Acceleration modes. If you swap too early, the writes will not occur as expected.

Note: This option can affect your whole design, but it applies only if you have some Verilog in your design.

+rtlIgnPragma

Purpose

Ignores --translate_off and --translate_on and related compiler directives (such as synthesis_off).

Category

Verilog analyzer source encryption option

Compile-time Synthesis Policy Checker option

Syntax

```
+rtlIgnPragma [+<entity>] [+<entity...>]  
rtlIgnPragma  [+<module>] [+<module...>]
```

Description

This option applies only to the specified entity(s) and module(s); otherwise, if no modules or entities are specified, it applies to all entities and modules. This means code within translate_off blocks is executed in all run-time modes (software simulation, Simulation Acceleration). If no entity(s) module(s) are specified, these compiler directives are ignored throughout the entire design.

By default, code within translate_off blocks is executed during software simulation mode, but not during Simulation Acceleration modes. +rtlIgnPragma is used if the design has a lot of simulation code needed for correct RTL functional simulation inside translate_off blocks.

Note: If you specify the +rtlIgnPragma option only with `vlan`, only the Verilog part of your design will be affected. If you specify the +rtlIgnPragma option only with the compiler, only the VHDL part of your design will be affected. If you specify it with both `vlan` and the compiler, all of your design (Verilog and VHDL) will be affected.

+showPragma

Purpose

Displays informative messages when certain pragmas are encountered.

Category

Compile-time option

Syntax

+showPragma

Description

Displays informative messages when the following pragmas are encountered:

```
// synopsys full_case  
// cadence full_case  
// quickturn full_case  
// pragma full_case  
  
// synopsys parallel_case  
// cadence parallel_case  
// quickturn parallel_case  
// pragma parallel_case  
  
// quickturn no_hardmacro  
// quickturn use_hardmacro  
// quickturn use_hardmacro_pack  
// quickturn keep_net  
// quickturn external_ref  
  
// synopsys translate_off  
// cadence translate_off  
// quickturn translate_off  
// pragma translate_off  
  
// synopsys translate_on
```

UXE Command Reference Manual

Commands and System Tasks for SA Mode

```
// cadence translate_on  
// quickturn translate_on  
// pragma translate_on  
  
// synopsys synthesis_off  
// cadence synthesis_off  
// pragma synthesis_off  
  
// synopsys synthesis_on  
// cadence synthesis_on  
// pragma synthesis_on  
  
// pragma cvastrprop  
// pragma cvaintprop  
  
/* sparse */  
  
// ixc rtl_on  
// ixc rtl_off  
// ixc sw_only_on  
// ixc sw_only_off  
// ixc tbcall_region  
// ixc bc_unroll_loop  
// ixc hw_action_block  
// ixc sw_action_block  
// ixc bc_step_loop  
  
// axis rtl_on  
// axis rtl_off  
// axis sw_only_on  
// axis sw_only_off  
// axis bc_unroll_loop  
// axis hw_action_block  
// axis sw_action_block  
// axis bc_step_loop
```

+rtlKeepDelays

Purpose

Honors delays by forcing the module to software simulation.

From IXCOM help page: Specifies max value for the Verilog intra-assignment delay to keep [default: ignore] during RTL mapping.

Category

Compile-time Synthesis Policy Checker option

Syntax

`+rtlKeepDelays+<delay_value>`

Description

Delays greater than *delay_value* are honored; delays equal to or less than *delay_value* are ignored to simulate in the emulator. `+rtlKeepDelays+0` keeps all intra-assignment delays.

Note: Affects Verilog modules only

+rtlmemsize

Purpose

Controls the size of two-dimensional memory objects, which the Palladium XP HDL Compiler maps to the emulator.

Any memory object, the size of which is less than the size specified by the *[num_of_bits]* argument, is mapped to the flip-flops. The default value of *[num_of_bits]* argument is 64. However, if the size of memory object exceeds the size specified by the *[num_of_bits]* argument, the memory object is not mapped to the emulator.

Category

Compile-time Synthesis Policy Checker option

Syntax

`+rtlmemsize[+<num_of_bits>]`

Description

The `+rtlmemsize+num` option of the `ixcom` command specifies the number that the Palladium XP HDL Compiler will use during synthesis to determine whether or not a particular variable should be mapped to Palladium XP memory.

Note: This option applies to both Verilog and VHDL memories.

The *[num_of_bits]* argument enables you to specify the size of the memory that needs to be mapped to the flip-flops by the Palladium XP HDL Compiler for all the modules. For example:

```
ixcom +rtlmemsize+1000
```

The above command sets the size the size of the memory that needs to be mapped to 1000 bits.

Memories that are less than 64 bits (or *num_of_bits*) are synthesized as gates and memories larger are synthesized as memory primitives or macros.

This option is intended for small memories only, typically 2,048 bits or less, because logic capacity is used in the hardware, rather than the available memory devices. Furthermore, setting *num_of_bits* to a larger number might significantly increase compile times. For

UXE Command Reference Manual

Commands and System Tasks for SA Mode

larger memories, or in order to use the available memory devices, model the memory around the memory primitive.

+rtlvarwidth

Purpose

Sets a limit on the size of signals and variables that will be implemented in the emulator

Category

Compile-time Synthesis Policy Checker option

Syntax

`+rtlvarwidth+<num_of_bits>`

Description

A module is moved to the software simulator if it has one or more signals/variables that require more than *num_of_bits* to be encoded in hardware. The default limit is 2,048 bits.

This switch is intended to detect large signals and variables that might need to be handled differently.

Note: Affects the VHDL part of your design only.

-q

Purpose

Goes into quiet mode.

Category

Compiler option

Syntax

`-q`

Description

Goes into quiet mode, which disables any kind of output being written to `stdout`. All messages will still go to the log file.

-u

Purpose

Considers all net, register, and instance paths as uppercase characters.

Category

Compiler option

Syntax

`-u`

-vaelabargs

Purpose

Specifies the arguments and options to be passed to *vaelab* utility at the time of HDL compilation.

Category

Compiler option

Syntax

```
-vaelabargs [vaelab_options]
```

-vavhdlargs

Purpose

Specifies the arguments and options to be passed to *vavhdl* utility at the time of HDL compilation.

Category

Compiler option

Syntax

`-vavhdlargs [vavhdl_options]`

-vavlogargs

Purpose

Specifies the arguments and options to be passed to *vavlog* utility at the time of HDL compilation.

Category

Compiler option

Syntax

```
-vavlogargs [vavlog_options]
```

-ncvlogargs

Purpose

Specifies the arguments and options to be passed to *ncvlog* utility at the time of compilation.

Category

Compiler option

Syntax

`-ncvlogargs <string>`

Description

This option enables you to specify IES tool (*ncvlog*) options on the IXCOM command line. The *<string>* argument value is passed to the IES tool, which is invoked as a part of IXCOM compilation.

Note: IXCOM does not perform any semantic checks on the string value. Also, the string value specified is passed to all invocations (in multi-step compile) of the tool and it is functionally equivalent to using the `./uxc_hdl.var` file in the IXCOM compilation. With `iRun -hw, null` argument value to this option is not supported.

-ncvhdlargs

Purpose

Specifies the arguments and options to be passed to *ncvhdl* utility at the time of compilation.

Category

Compiler option

Syntax

`-ncvhdlargs <string>`

Description

This option enables you to specify IES tool (*ncvhdl*) options on the IXCOM command line. The *<string>* argument value is passed to the IES tool, which is invoked as a part of IXCOM compilation.

Note: IXCOM does not perform any semantic checks on the string value. Also, the string value specified is passed to all invocations (in multi-step compile) of the tool and it is functionally equivalent to using the `./uxc_hdl.var` file in the IXCOM compilation. With `iRun -hw, null` argument value to this option is not supported.

-ncelabargs

Purpose

Specifies the arguments and options to be passed to *ncelab* utility at the time of compilation.

Category

Compiler option

Syntax

`-ncelabargs <string>`

Description

This option enables you to specify IES tool (*ncelab*) options on the IXCOM command line. The *<string>* argument value is passed to the IES tool, which is invoked as a part of IXCOM compilation.

IXCOM does not perform any semantic checks on the string value. Also, the string value specified is passed to all invocations (in multi-step compile) of the tool and it is functionally equivalent to using the `./uxc_hdl.var` file in the IXCOM compilation. With `irun -hw`, `null` argument value to this option is not supported.

-loadsc

Purpose

Specifies the shared libraries that should be searched for SystemC modules if no Verilog modules are found.

Category

Compiler option

Syntax

`-loadsc <library_name>`

Description

Searches for a SystemC module if no Verilog module is found, in the shared libraries specified with the `-loadsc` option. You can specify multiple library names by repeating the `-loadsc` option on the same `ixcom` command line.

+systemc_args

Purpose

Passes the specified arguments to the `sc_main` function.

Category

Compiler option

Syntax

`+systemc_args+<args>`

Description

Passes the arguments to the `sc_main` function.

+tfconfig

Purpose

Specifies the interface-compilation mapping.

Category

Compiler option

Syntax

`+tfconfig+<side_file_name>`

Description

The `+tfconfig+<side_file_name>` option specifies the interface-compilation mapping using a side file, in which each line specifies a mapping control in the following format:

`<module_name> <task_func_name> mapping_ctrl [mapping_options] *`

The `<module_name>` can be a specific SystemVerilog module, interface, package name or the `*` character, which means for all modules with the given task and function name.

The `<task_func_name>` can either be specific task or function name or have a prefix^{*} to match all tasks and functions with the given prefix. However, you cannot have both wildcard `module_name (*)` and wildcard `task_func_name (prefix*)` at the same time.

+fifo_lbsize+size

Purpose

Specifies the size of the local buffer used for the GFIFO call instances.

Category

Compiler option

Syntax

`+fifo_lbsize+size:<size>`

Description

For each GFIFO call instance, a local buffer of size is created to buffer call packets before collection into the GFIFO. When multiple call packets are generated at the same time, local buffer can hold the call packets and keep the RTL clock running. When local buffers are full, behavioral cycles will kick in to suspend RTL clocks to prevent new packet generation. The default local buffer size is 4. Using a deeper local buffer will cost more emulator gates, but will tolerate traffic congestion for better performance. The choice of local buffer size should be based on the GFIFO traffic pattern to trade-off between performance and gate count. If the gate count is limited, or the design has large number of GFIFO call instances, set the local buffer size to 1. If the gate-count utilization is low, but traffic congestion causes high behavioral cycles, set the local buffer size to 8.

+gfifoDisp

Purpose

Maps the \$display-family task calls in a special fast non-blocking mode using memory buffers.

Category

Compiler option

Syntax

+gfifoDisp[+<module_name>]*

Description

Automatically maps the following functions using GFIFO:

- ❑ \$display, \$displayh, \$displayo, \$displayb
- ❑ \$fdisplay, \$fdisplayh, \$fdisplayo, \$fdisplayb
- ❑ \$fwrite, \$fwriteh, \$fwriteo, \$fwriteb
- ❑ \$write, \$writeh, \$writeo, \$writeb

This +gfifoDisp[+<module_name>]* option maps the \$fopen, \$fclose, \$fflush functions to tb_import. This option preserves the order of \$display statements.



Note that \$error, \$warning, \$info, \$fatal **are not supported with the +gfifoDisp option.**

+staticFd

Purpose

Keeps file descriptors in software, globally (or for the specified set of modules) to reduce the hardware-resource requirement and improve I/O performance.

Category

Compiler option

Syntax

`+staticFd[+<module_name>]*`

Description

In order to support the most general use mode, the integer-file descriptor is stored in hardware and send to software at every call. However, in those cases, when the file descriptor is created by `$fopen` in the `initial` statement and stays as a constant to the end of simulation, the file descriptor can stay in software. The `+staticFd[+<module_name>]*` option keeps file descriptors in software, globally (or for the given set of modules) to reduce the hardware-resource requirement and improve I/O performance. If you have a large number of `$fdisplay*` statements, the gate-count saving can be significant. If the design has 10,000 `$fdisplay`, using the `+staticFd` option can save 3.2MG with a minimum local buffer size (`+gfifo_lbsize+1`).

-rtlNameForGenerate

Purpose

Enables run-time access of design hierarchies involving for-generate blocks and instance arrays.

Category

Compiler option

Syntax

`-rtlNameForGenerate`

Description

By default, during elaboration, IXCOM unrolls the for-generate blocks and instance arrays written in the original design files. This transformation changes the hierarchical address of the design element present inside the for-generate block because IXCOM modifies the name of the block by appending underscores in place of the square brackets. So, IXCOM generates `<module_name>_0` instead of `<module_name>[0]` for Verilog, and `<<module_name>_0` instead of `<module_name>(0)` for VHDL.

At run time, the design elements cannot be accessed with their original-RTL names because of this change in the name. To avoid these access issues, use the `-rtlNameForGenerate` option.

With the `-rtlNameforgenerate` option, local OOMRs to variables declared within for-generate of complex datatypes, such as struct and union, and declared within a package or compilation unit scope, are not supported.

-xeCompile

Purpose

Specifies the user data command files to be used during the backend compilation process, xeCompile.

Category

Compiler option

Syntax

`-xeCompile <arg>`

Description

This option enables you to specify xeCompile user data command files on the IXCOM command line. The user data command files specified in `<arg>` argument value are processed during xeCompile process, which is invoked as a part of IXCOM compilation. The allowed argument values are:

- `compile=<file>`
Specifies the compile-time commands and options to be passed to xeCompile process
- `import=<file>`
Specifies the design import commands to be passed to the xeCompile process
- `postImport=<file>`
Specifies the post-Import commands to be passed to the xeCompile process
- `compilerOptions=<file>`
Specifies the compile-time options to be passed to xeCompile process
- `postCompile=<file>`
Specifies the commands and options to be executed after compilation

UXE Command Reference Manual

Commands and System Tasks for SA Mode

For the xeCompile process, the user data command files specified using the above option have higher precedence over user data files (`./compile.qel` and `./compilerOptions.qel`) in the compile directory.

Example

```
ixcom -xecompile compilerOptions=compilerOptions.QEL
```

-xcompileargs

Purpose

Specifies the user data command files to be used during the backend compilation process, xeCompile, to map the design to hardware.

Category

Compiler option

Syntax

`-xcompileargs <string>`

-verbose

Purpose

Enables verbose messages for design elaboration.

Category

Compiler option

Syntax

`-verbose`

-v

Purpose

Uses the specified Verilog *library_file* for resolving library cells and modules in the design.

Category

Compiler option

Syntax

`-v <library_file>`

Description

Specifies a Verilog library file. The compiler will search this file for modules that are not defined in the Verilog sources specified.

-z

Purpose

Uses the specified *map_directory* that contains compiled alternate module descriptions.

Category

Library and memory mapping compiler option

Syntax

`-z <map_directory>`

Note: This older method, for vpp only, is used with the preceding options (`-m` and `-w`).

-y

Purpose

Searches only files with the specified extensions in the specified library directory.

Category

Compiler option

Syntax

`-y <directory>`

Description

Searches only files with the specified extensions (`ext[ext]...`) in the specified `library_directory` when resolving library cells and modules in the design. Must be used with `+libext`.

+define

Purpose

Defines Verilog macros.

Category

Compiler option

Syntax

+define+[macro]

Syntax Example

```
'ifdef USE_DRAM  
'else  
'endif
```

Description

With the `'ifdef` and `'define` Verilog compiler directives, defines a macro or passes a value to a macro.

IXCOM Built-in Macros:

The following macros are predefined in IXCOM.

- ❑ IXCOM_UXE

IXCOM_UXE is a predefined or built-in macro that IXCOM always evaluates to be true.

- ❑ AXIS
- ❑ IXC_REL_DATE

For IXC_REL_DATE, the value is 20150101 for UXE 14.1 QSR 1 release.

+includir

Purpose

Specifies one or more directory paths to be searched for the include files specified with the '`include`' compiler directive.

Category

Compiler option

Syntax

`+includir+<path> [+<path...>]`

+noProcessAttrib

Disables the user attribute support in VHDL modules.

Purpose

This option disables user-defined attribute declaration and specification in VHDL modules.

Category

Compiler option

Syntax

`+noProcessAttrib+<entity>...`

In case no entity name is specified, the attribute support is disabled for the entire design.

Note: The user attributes might not work correctly when using VITAL (VHDL Initiative Towards ASIC Libraries). You might use `+noProcessAttrib` to ignore attributes.

+libext

Purpose

Searches only files with the specified extensions when resolving library cells and modules in the design.

Category

Compiler option

Syntax

`+libext+<extension>[+<extension...>]`

Note: Use with the `-y library_directory` option

+liborder

Purpose

Searches for module definitions in the next specified library.

Category

Compiler option

Syntax

+liborder

Description

With the `-y library_directory` option, searches for module definitions in the *next* library specified on the command line when an unresolved instance is found in a particular library.

+librescan

Purpose

Searches for module definitions in the first specified library.

Category

Compiler option

Syntax

+librescan

Description

With the `-y library_directory` option, searches for module definitions in the *first* (as opposed to *next*, as with `+liborder`) library specified on the command line when an unresolved instance is found in a particular library.

+libverbose

Purpose

Displays the order in which libraries are scanned for module definitions

Category

Compiler option

Syntax

+libverbose

Description

With the *-y library_directory* option, displays the order in which libraries are scanned for module definitions.

-Z

Purpose

Automatically replaces modules in the design with modules of the same name found in *map_file*.

Category

Compiler option

Syntax

`-Z <map_file>`

Description

Automatically replaces modules in the design with modules of the same name found in *map_file*, which enables you to easily replace a module without disturbing the design flow.

Note: This option replaces the older method of specifying module mapping using `-m`.

+maxdelays

Purpose

Uses the `max` value, instead of the default `typ`, in a `min:typ:max` delay expression.

Category

Compiler option

Syntax

`+maxdelays`

+mindelays

Purpose

Uses the `min` value, instead of the default `typ`, in a `min:typ:max` delay expression.

Category

Compiler option

Syntax

`+mindelays`

+rtlHonorCasePragma

Purpose

Causes Synopsys *case statement pragmas* (`parallel_case` and `full_case`) to be honored when compiling the design for simulation acceleration mode. Logic generated for the case statements in the emulator will match that generated by synthesis tools.

Category

Verilog analyzer option

Compiler option

Synthesis Policy Checker option

Syntax

```
+rtlHonorCasePragma
```

Description

By default, case statement pragmas are ignored to comply with IEEE-standard Verilog simulation semantics because these pragmas might cause the synthesis tool to generate different logic, resulting in possible mismatches between RTL and gate-level simulation.

Use this option with caution; it can cause simulation mismatches because the software simulation mode is not affected by this option.

Category

Compiler option

Syntax

```
+rtlHonorCasePragma
```

Description

Causes Synopsys case statement pragmas (`parallel_case` and `full_case`) to be honored when compiling the design for Simulation Acceleration mode. Logic generated for the case statements in the emulator will match that generated by synthesis tools.

Note: Since software simulation mode is not affected by this option, mismatches might result between Simulation Acceleration modes and software simulation mode.

+protect

Purpose

Generates an encrypted file for the Verilog code in the region specified by the `'protect` and `'endprotect` compiler directives.

Category

Compiler option

Syntax

`+protect[.ext]`

Note: The remaining code in the file(s) is not encrypted.

+rtlIgnTask

Purpose

Ignores the specified PLI or user tasks or functions when compiling for the emulator.

Category

Compiler option

Syntax

```
+rtlIgnTask[+<task_or_function>] [+<task_or_function...>]
```

Description

Ignores the specified PLI or user tasks or functions (\$ sign is not required) when compiling for the emulator (-ua option). If these tasks are not ignored, the modules in which they exist are automatically run in software simulation mode only, resulting in slower performance. If you specify *no task_or_function(s)*, all PLI tasks and functions in the design are ignored.

+loadpli1

Purpose

Specifies the PLI1 *library_name:boot_routine(s)*.

Category

Compiler option

Syntax

+loadpli1=<*library_name:boot_routine(s)*>

+loadvpi

Purpose

Specifies the VPI *library_name:boot_routine(s)*.

Category

Compiler option

Syntax

`+loadvpi=<library_name:boot_routine(s)>`

+vhdlmcx

Purpose

Defines the minimum size of VHDL one-dimensional array variable that should be synthesized into hardware memory primitive. The default minimum size is 64 bits.

Category

Compiler option

Syntax

+vhdlmcx[+<modle_name> [+size]]

+enableSvPkg

Purpose

Enables access to SystemVerilog Package objects in the hardware partition.

Category

Compiler option

Syntax

+enableSvPkg

Description

By default, in the IXCOM flow, access to SystemVerilog Package objects is not allowed in the hardware partition, and modules that have such access are partitioned to the software partition by the auto-partitioner. Use the `+enableSvPkg` option to access to the SystemVerilog Package objects in the hardware partition.

+hm

Purpose

Includes the hard module.

Category

Compiler option

Syntax

+hm+<module>

+hmpi

Purpose

Enables TBA transformations for hardware emulation.

Category

Compiler option

Syntax

+hmpi

+hwdpi

Purpose

Automatically makes all export DPI tasks/functions declared in the DUT partition to run in hardware.

Category

Compiler option

Syntax

`+hwdpi [+<mod>]`

+iscdisp

Purpose

Converts the \$display statements to tb_import tasks using the Interface Compiler.

Category

Compiler option

Syntax

+iscdisp

+margtbc

Purpose

Maps the `tbcall` interface signals into memories instead of normal SA connections. In some cases, this can enable the design to be compiled successfully, which otherwise, might have failed with the following error message:

```
ERROR (db2util-1054): There might be too many primary outputs (DBOB's) and/or probes for the scheduler to handle.
```

Category

Compiler option

Syntax

`+margtbc`

+newhm

Purpose

Includes the new hard module that needs to be compiled.

Category

Compiler option

Syntax

+newhm+<module>

+saSimpleLatch

Purpose

Delays data propagation to next cycle. The default is same.

Category

Compiler option

Syntax

+saSimpleLatch

+saVerbose

Purpose

Enables verbose messages for the signal analysis phase.

Category

Compiler option

Syntax

+saVerbose

+synloopsize

Purpose

Specifies the number of iterations (in a looping statement) beyond which the looping statement will be considered unsynthesizable.

By default (that is, when the +synloopsize option is not specified), this number is 100. So, looping statements that have more than 100 iterations are considered unsynthesizable.

Category

Compiler option

Syntax

`+synloopsize+<positive_number>`

Example

By default, the following `while` loop is deemed unsynthesizable; that is, auto-partitioner will issue a `RTL_UNSUPPORTED_STMT` message for it, and push the module containing it to the software partition. Adding `+synloopsize+101` to the `ixcom` command line will make the `while` loop synthesizable.

```
module top(input [63:0] in1, output reg [6:0] out1);

integer i;

always @ (*) begin
    i = 101;
    out1 = 7'h0;
    while (i>0) begin
        out1 = i;
        i = i - 1;
    end
end

endmodule
```

-target

Purpose

Enables compilation of the design in stages.

Category

Compiler option

Syntax

```
-target swmodel | hdlice | xecompile
```

Description

Using the `-target` option, you can control the design compilation to stop at or rerun from a particular stage, as follows:

- ❑ `swmodel`, which enables you to generate the software model of the design for software simulation using IES.
- ❑ `hdlice`, which enables you to generate the synthesized netlist for the design for acceleration on the Palladium XP emulator.
- ❑ `xecompile`, which enables you to generate the compiled database that can be downloaded to the Palladium XP emulator.

The stage name can be specified with the minimum number of case-insensitive characters.

Examples

The following examples use `Dut` as the top-level DUT name, and `tb.sv` and `dut.sv` as the testbench and design source files, respectively.

- Software compilation stage (`swmodel` stage):
 - ❑ To analyze the source files, elaborate the design, and generate a simulation snapshot, use the following command:
`ixcom -target swmodel -ua +dut+Dut tb.sv dut.sv`
 - ❑ To rerun IXCOM compilation from `swmodel` stage, use the following command:

UXE Command Reference Manual

Commands and System Tasks for SA Mode

```
ixcom -target swmodel
```

■ Design synthesis stage (`hdlice` stage):

- To analyze the source files, elaborate the design, generate a simulation snapshot, and invoke the HDL Compiler to generate the design netlist, use the following command:

```
ixcom -target hdlice -ua +dut+Dut tb.sv dut.sv
```

- To rerun compilation from the `hdlice` stage, use the following command:

```
ixcom -target hdlice
```

■ Hardware compilation stage (`xecompile` stage):

To compile the design for hardware, use the following command:

```
ixcom -target xeCompile
```

+targetTop

Purpose

Enables the top-level design under test (DUT) for ICE target connection.

Category

Compiler option

Syntax

`+targetTop+<design_unit>`

Description

This option is used to specify the top-level module for connection to ICE target interface. With this option, the compiler makes all I/Os of the specified module available for target interface connection.

+tb_export

Purpose

Designates the specified task or function in the DUT to run in the hardware using primary IOs or memory buffer to transfer arguments and results.

Category

Compiler option

Syntax

+tb_export+ [mod.]<tf>[a]

Here, a=<:pio/mem> [+ . . .]

+tb_import

Purpose

Designates the specified task or function in the DUT to run in the software using primary IOs or memory buffer to transfer arguments and results.

Category

Compiler option

Syntax

+tb_import+ [mod.]<tf>[a]

Here, a=<:pio/mem> [+ . . .]

+tb_import_systf

Purpose

Designates the specified system task or function in the DUT to run in software. The syntax and behavior of this option is the same as the `+tb_import` option, except that system <*tf*> name specified must not include \$ prefix in it.

Category

Compiler option

Syntax

`+tb_import_systf+ [mod.]<tf>[a]`

Here, *a*=<:`pio`|`mem`>[+...]

Note: When you execute the `+tb_import_systf+ [mod].display:mem` command, the value is displayed for two states only.

-vhdl_time_precision

Purpose

Specifies the default time precision for VHDL modules.

Category

Compiler option

Syntax

`-vhdl_time_precision <precision>`

Example

`-vhdl_time_precision 1ns`

+xcDesignTop

Purpose

Specifies the path to the DUT module <du> instance in the testbench hierarchy to link the HVL compiled separately in IES and the DUT compiled with IXCOM.

Category

Compiler option

Syntax

+xcDesignTop+<hier>=<du>

+dump_bidir

Purpose

Identifies and dumps potential bidirectional signals that prevent you from using the `+xcDesignTop` option of the `ixcom` command.

Category

Compiler option

Syntax

`+dump_bidir`

+xcBidir

Purpose

Generates additional instrumentation to enable support for bidirectional signals with the `+xcDesigntop` option of the `ixcom` command. To identify the potential bidirectional signals, recompile the design with the `+dump_bidir` option of the `ixcom` command.

Category

Compiler option

Syntax

`+xcBidir`

-trans_rtl

Purpose

Simplifies RTL for ICE mode and Protium consumption.

Category

Compiler option

Syntax

`-trans_rtl -top`

Description

The `-trans_rtl` option enables the required internal transformations for simplifying RTL to serve as input for later flows, such as ICE mode or Protium. It generates the data in a separate directory named `xcice_work` to distinguish it from the `xc_work` directory that is generated during typical IXCOM compilation for SA mode.

With the `-trans_rtl` option, it is mandatory to use the `-top` option to specify the top-level module. With the `-trans_rtl` option, you can specify multiple modules as top-level modules. By default, with the `-trans_rtl` option, all the modules specified with the `-top` option are considered as DUT modules. Specify SystemVerilog and Verilog configurations with the `-top` option when using the `-trans_rtl` option.

With the `-trans_rtl` option, the `+rtlKeepAssign`, `+no_sva`, `+no_bc`, `+no_psl` options are enabled by default. Do not use the `+dut` option when using the `-trans_rtl` option. If included, the `+dut` option is ignored. The `+xcDesigntop` option is also ignored when `-trans_rtl` option is used.

If you have any OOMRs in your design with a specific prefix, you can specify the hierarchical prefix using the `+prefixXref` option.

Example

```
ixcom -top cfg -top top -trans_rtl  
ixcom -sv test.sv -trans_rtl -top top -target xecompile  
ixcom -sv test.v -top TB -target hdlice -trans_rtl +prefixXref+TB_TOP.TB=TB
```

+prefixXref

Purpose

Enables specification of hierarchical prefixes for OOMRs when mapping RTL for ICE mode by using the `-trans_rtl` option.

Category

Compiler option

Syntax

```
+prefixXref+mytb.dutinst=dutmodule
```

Description

If you have any hierarchical names to be prefixed to the OOMRs in your design while compiling with `-trans_rtl` option, you can specify the hierarchical prefix with the `+prefixXref` option. This option is specific to RTL mapping.

Example

```
ixcom -sv test.v -top TB -target hdlice -trans_rtl +prefixXref+TB_TOP.TB=TB
```

+xcEmbeddedTb

Purpose

Specifies the shell module *<module_name>* manually created for the testbench module embedded in the DUT hierarchy.

Category

Compiler option

Syntax

`+xcEmbeddedTb+<module_name>{+<module_name>}`

-ixclkgen

Purpose

Automate ICE clock generation by calling *ixclkgen* utility from IXCOM.

Category

Compiler option

Syntax

`-ixclkgen <input_filename>`

Description

This option enables you to automate ICE clock generation by using the *ixclkgen* utility. Specify the command-line arguments to be consumed by *ixclkgen* utility in the `<input_filename>` file.

Note: IXCOM marks the generated clock module for partitioning into the hardware. This is equivalent to specifying the `+dut+<du>` option with the *ixcom* command. The clock module is regarded as a top-level design unit if this module has not been instantiated in the design. If you have invoked IXCOM with an explicit list of top-level design units (by using the `-top <du>` option) and the clock module is not instantiated in the design, you must specify the clock module as one of the top-level design units to include it in the design.

For more information regarding the *ixclkgen* utility, refer to the description of the [ixclkgen](#) command and the *Clock Module Generation by ixclkgen Utility* section in the *Modeling Clocks* chapter of the *UXE User Guide*.

+hwRandom

Purpose

Enables the transformation of `$random` and `$urandom` system function calls by the IXCOM compiler.

Category

Compiler option

Syntax

`+hwRandom [+<modname>] *`

Description

If the `+hwRandom` option is specified without any module, all `$random` and `$urandom` function calls inside the DUT modules are transformed. If the `+hwRandom` option is specified with module names, `$random` and `$urandom` calls in only the specified modules are transformed.

If a call to the `$random` or `$urandom` function inside a DUT module is to be transformed by the IXCOM compiler, the call must be inside the hardware partition. So, the call must not be in a `translate_off` block and must be reachable from one of the following code blocks:

- An `always` process that is mapped to the hardware (an `always` process that is not inside a `translate_off` block or `sw_only` block), and the `always` process is not one of `always_comb`, `always_latch`, or `always @*` processes.
- a `TB_EXPORT`, `SFIFO`, or `GFIFO EXPORT` scope

The `$random` and `$urandom` calls not mentioned above are not transformed by the IXCOM compiler.

-localDiskComp

Purpose

Enables IXCOM compilation on the local disk.

Category

Compiler option

Syntax

`-localDiskComp`

Description

This option enables IXCOM to utilize the local disk (`/tmp` disk) for compilation. This speeds up IXCOM compilation if your design-working directory is in a NFS disk because NFS-disk access is slower than local-disk access.

When you use the `-localDiskComp` option, IXCOM compilation differs from a regular IXCOM compilation in the following ways:

- Instead of creating a regular `AxisWork` directory, IXCOM creates a soft link that points to the `/tmp` disk (that is, `./AxisWork --> /tmp/PID123/AxisWork`).
- IXCOM also modifies the path in `xc_work/va.lib`. The entire HDL Compiler database goes to `/tmp/PID123/xc_vawork`. A soft link is also created for the HDL Compiler output netlist (that is, `/xc_work/xc_var.ver --> /tmp/PID123/xc_var.ver`).

Note: PID123 is the IXCOM-Linux process ID. So, it has a unique path for each compilation.

- To free up space in `/tmp`, IXCOM automatically removes `AxisWork` and HDL Compiler database from `/tmp` after the compilation finished. Do not use this option if you need `AxisWork` or HDL Compiler database for debugging.

+localDiskComp

Purpose

Enables local-disk IXCOM compilation.

Description

This option works in the same way as [-localDiskComp](#). The difference is that +localDiskComp allows you to specify a path for the local-disk compilation. In addition, after the compilation, IXCOM does not remove the database residing in the local-disk directory that you have provided with the +localDiskComp option.

When you set *<path>* to allowOW, you can overwrite the existing database in the *<path>* directory. This is useful for incremental compilation.

Category

Compiler option

Syntax

+localDiskComp [+<path>=allowOW]

+bufdisp

Purpose

Enables output buffering of \$display in the hardware memory to avoid context switching from the hardware to the software.

Note: The value is displayed for two states only. This option has been deprecated. Use the +fifoDisp option instead.

Category

Compiler option

Syntax

+bufdisp[+<design_unit>]

+iscDelay

Purpose

Enables blocking time delay transformations for a specific module in the DUT. This option enables \$time transformations in the DUT after it identifies and transforms RTL blocking time delay.

Category

Compiler option

Syntax

`+iscDelay+<module> [+<module>]`

Note: Alternatively, add the `map_delays` directive in an `initial` statement within the module as shown below:

```
$ixc_ctrl("map_delays");
```

-timescale

Purpose

Specifies default timescale for Verilog modules for which no default timescale is specified. Note that this is different from the [-vtimescale](#) option of the `vlan` command because `-vtimescale` applies to each file specified with the `vlan` command, whereas `-timescale` applies to the modules.

Category

Compiler option

Syntax

`-timescale <unit/precision>`

Syntax Example

`-timescale 1ns/1ps`

+hwfrc

Purpose

Instructs IXCOM compiler to create a mapping for a `force` statement found under the DUT.

Use of the `+hwfrc` option is required for the correct handling of `force` statements in the DUT. If the `+hwfrc` option is not used, the effect of a `force` statement in the DUT might be lost when the design is swapped to the emulator.

The force/release hardware mapping also applies to the procedural `assign` statements which are similar to `force` except that these apply to `reg` objects, as shown in the following example:

```
always@(*)
begin
    assign foo = 0;
    ....
    ....
end
```

Category

Compiler option

Syntax

`+hwfrc`

+SV

Purpose

Forces SystemVerilog compilation.

Category

Verilog analyzer/Compiler option

Syntax

+sv

Description

The `+sv` option introduces SystemVerilog keywords that cannot be the names of identifiers in your design. If you have a design that uses SystemVerilog keywords as identifiers, you can specify the `+sv` option to compile all files as SystemVerilog modules.

Note: The `+sv` option applies to all the files specified on the command-line. For information on determining the design language based on the file extension, refer to the *Determining File Type by Source Filename Extension* section in the *Compiling Designs for Simulation Acceleration Using IXC* chapter of the *UXE User Guide*.

+v1995

Purpose

Disables support for new Verilog-2001 keywords.

Note: The +v1995 option applies to all the files specified on the command-line. For information on determining the design language based on the file extension, refer to the *Determining File Type by Source Filename Extension* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

Category

Compiler option

Syntax

+v1995

+v2000

Purpose

Enables the *IEEE 1364-2001* (for Verilog HDL) features: generate block features, genvar declaration, signed.

Note: The +v2000 option applies to all the files specified on the command-line. For information on determining the design language based on the file extension, refer to the *Determining File Type by Source Filename Extension* section in the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

Category

Compiler option

Syntax

+v2000

-strictLRM

Purpose

Enforces strict semantic checks as per the SystemVerilog P1800-2009 LRM. By default, IXCOM relaxes the checks in certain scenarios, and might not report all the semantic errors. Using this option enforces those checks. Recompile the design for this option to take effect.

Category

Analyzer and Compiler option

Syntax

`-strictLRM`

+rtlCommentPragma

Purpose

Comments out code from a `translate_off` pragma (and related comment pragmas) to the next `translate_on` pragma.

Category

Synthesis Policy Checker option

Syntax

```
+rtlCommentPragma [+module...]
```

Description

Comments out code from a `translate_off` pragma (and related comment pragmas) to the next `translate_on` pragma, that is, this code will not be executed in any run-time modes (software simulation, acceleration, or emulation modes).

This option is useful when you get the following SAC warning in the **ixcom.log** file:

```
SAC: Lline_number "file_name": Forcing module module to software because  
wire|reg wire_or_reg_name has fanout to a HW object but is driven from  
a translate off object.
```

This warning indicates a signal declared outside a `translate_off` block is being assigned a value within the block. The *module* is pushed to software to avoid potential mismatches between software simulation and Simulation Acceleration modes. If you are satisfied that this signal assignment will not be a problem, to override this warning, you can use the `+rtlCommentPragma` option. The affected *module* will be accelerated by the emulator, instead of running in the software simulator.

Note:

This option can be passed to `vlan`, `vhan`, or the compiler.

- With `vlan` or the compiler, you can specify the name of a specific *module*, rather than letting it default to the entire file or files specified.
- With `vhan`, you cannot specify an individual entity; the option will always apply to the entire file or files.

UXE Command Reference Manual
Commands and System Tasks for SA Mode

- This option applies only to the *module(s)* specified; otherwise, it applies to all entities and modules.

-synth_prefix

Purpose

Appends additional pragmas for the synthesis directive to the default pragma list. The default pragmas are:

- `pragma`
- `synopsys`
- `quickturn`

Syntax

`-synth_prefix <pragma_name>`

-rm_synth_prefix

Purpose

Deletes the specified pragma for the synthesis directives from the default pragma list.

The default pragmas are:

- pragma
- synopsys
- quickturn

Note that this option does not remove the `quickturn` pragma.

Syntax

`-rm_synth_prefix <pragma_name>`

-defparam

Purpose

Redefines the value of the specified Verilog parameter.

Syntax

`-defparam <arg>`

-ixerror

Purpose

Upgrades the severity of warning messages to error messages. You can increase the severity level of multiple warning messages either by specifying the `-ixerror` option multiple times, or by specifying a single `-ixerror` option and separating the [*<mnemonic_code>*] arguments with a colon. If you specify the `-ixerror` option without any [*<mnemonic_code>*], all warning messages are converted to errors.

Note: This option corresponds to the `-ncerror` option of the `irun` command. This option is not available with `vhan`.

Category

Verilog Analyzer and Compiler option

Syntax

```
-ixerror [<mnemonic_code1>:<mnemonic_code2>:...] <file1.v> <file2.v>
```

Example

```
% ixcom -ixerror PORT_WIDTH_MISMATCH source.v
% ixcom -ixerror BITSEL_OFB -ixerror PORT_WIDTH_MISMATCH \
    -ixerror UNCONN_DOTSTAR_PORT source.v
% ixcom -ixerror BITSEL_OFB:PORT_WIDTH_MISMATCH:UNCONN_DOTSTAR_PORT source.v
```

-ixfatal

Purpose

Upgrades the severity of warning or error messages to fatal errors. You can increase the severity level of multiple warning or error messages either by specifying the `-ixfatal` option multiple times, or by specifying a single `-ixfatal` option and separating the [*<mnemonic_code>*] arguments with a colon. If you specify the `-ixfatal` option without any [*<mnemonic_code>*], all error messages are converted to fatal errors, and compilation stops immediately at the occurrence of the first fatal error.

Note: This option corresponds to the `-ncfatal` option of the `irun` command. This option is not available with `vhan`.

Category

Verilog Analyzer and Compiler option

Syntax

```
-ixfatal [<mnemonic_code1>:<mnemonic_code2>:...] <file1.v> <file2.v>
```

Example

```
% ixcom -ixfatal ANALYZE_INTERNAL source.v
% ixcom -ixfatal ANALYZE_INTERNAL -ixfatal BAD_WIRE_TYPE source.v
% % ixcom -ixfatal ANALYZE_INTERNAL:BAD_WIRE_TYPE source.v
```

Synthesis Policy Checker Options

The following options relate to the Synthesis Policy Checker.

- [+rtlCommentPragma](#) on page 638
- [+rtlHonorCasePragma](#) on page 596
- [+rtlIgnPragma](#) on page 557
- [+rtlKeepDelays](#) on page 646
- [+rtlMemsize](#) on page 647
- [+rtlVarwidth](#) on page 648
- [+no_rtlxmr](#) on page 649
- [+salgnXmrWrite](#) on page 556

+rtlKeepDelays

Purpose

Honors delays by forcing the module to software simulation.

Category

Synthesis Policy Checker option for Verilog modules only.

Syntax

`+rtlKeepDelays+delay_value`

Description

Delays greater than *delay_value* are honored; delays equal to or less than *delay_value* are ignored to simulate in the emulator. `+rtlKeepDelays+0` keeps all intra-assignment delays.

+rtlmemsize

Purpose

Automatically implements, in the emulator, any memory (array of registers) whose size (address x data) is less than or equal to *num_of_bits*, provided the module is synthesizable.

Category

Synthesis Policy Checker option for Verilog and VHDL modules.

Syntax

`+rtlmemsize+num_of_bits`

Description

The `+rtlmemsize+num` option specifies the number that the Palladium XP HDL Compiler will use during synthesis to determine whether or not a particular variable should be mapped to Palladium XP memory.

Note: This option applies to both Verilog and VHDL memories.

Modules with larger memories are moved to the software simulator (`RTL_UNSUPPORTED_DECL` message in `ixcom.log`). Without this option, memories 64 bits or smaller are automatically implemented in the emulator, and larger are moved to software.

This option is intended for small memories only, typically 2000 bits or less, because logic capacity is used in the hardware, rather than the available memory devices. Furthermore, setting *num_of_bits* to a larger number might significantly increase compile times.

For larger memories, or in order to use the available memory devices, model the memory around the memory primitive.

+rtlvarwidth

Purpose

Sets a limit on the size of signals and variables that will be implemented in the emulator.

Category

Synthesis Policy Checker option for the VHDL parts of your design only.

Syntax

`+rtlvarwidth+num_of_bits`

Description

A module is moved to the software simulator if it has one or more signals/variables that require more than *num_of_bits* to be encoded in hardware. The default limit is 2,048 bits.

This switch is intended to detect large signals and variables that might need to be handled differently.

+no_rtlxmr

Purpose

Disallow cross-module references (XMRs) in DUT modules, in compliance with the synthesis subset of Verilog constructs defined by Synopsys.

Category

Synthesis Policy Checker option for the Verilog modules in your design only.

Description

The emulator enables cross-module references to make it easier to move Verilog RTL testbench modules (which typically use XMRs to monitor design signals) into the DUT for Simulation Acceleration modes. It is not considered good design practice to use hierarchical references inside the chip or in the part of the design to be realized to actual hardware.

Note: Verilog `force` and `release` statements are not supported in the DUT, since they are not in the synthesis subset.

System Tasks and Procedures

Many IEEE-standard system tasks can be added to your Verilog code and, in most cases entered at the Command Line Interface (CLI). In addition, many equivalent VHDL procedures can be added to your VHDL code.

Note:

- Other system tasks can be entered using the \$xc_cmd Verilog system task, which lets you execute an xc command from your source code. Refer to [\\$xc_cmd\(\)](#); on page 652.
- If your design contains system tasks, you might get a syntax error when compiling with another simulator. To avoid this problem, surround such code with the IXCOM_UXE compiler directive. For example:

```
initial
begin
  'ifdef IXCOM_UXE
    $axis_readmemh("init.dat", top.design_mod.submod1.axis_mem_instance);
  'else
    $readmemh("init.dat", top.design_mod.submod1.usermem);
  'endif
end
```

This section documents the supported Verilog system tasks and VHDL procedures:

- [Supported Verilog System Tasks & VHDL Procedures](#) on page 651
- [Supported IEEE-Standard Non-CLI System Tasks](#) on page 660
- [Supported IEEE-Standard Compiler Directives](#) on page 661

Supported Verilog System Tasks & VHDL Procedures

The following Verilog system tasks and VHDL procedures were developed specifically. To use these tasks and procedures, you can:

- Add the system task directly to your Verilog code
- Add the procedure directly to your VHDL code
- Enter the system task at the CLI (Command Line Interface)

To use the VHDL procedures, you must add the following library clause to your code:

```
library axis;  
use axis.axis.all;
```

Note: Instance paths to signals or Verilog primitives are specified using period delimiters, that is:

instance.instance.signal_or_primitive

The following tasks and procedures are covered in this section:

- [\\$xc_cmd\(\)](#); on page 652
- [\\$export_deposit\(\)](#); on page 653
- [\\$export_event\(\)](#); on page 654
- [\\$export_frcrel\(\) | export_frcrel\(\)](#); on page 656
- [\\$export_read\(\) | export_read\(\)](#); on page 658

\$xc_cmd();

Purpose

Reads an `xc` command directly from your Verilog source code

Category

Verilog system task/CLI command: **\$xc_cmd();**

Parameters

An `xc` command enclosed within double quotation marks or saved as a string variable.

Note: The following commands are not supported through the `$xc_cmd` system task if you are using the xeDebug tool:

`on`, `off`, `stop`, `run`, `tbrun`, `autorun`, `nbrun`, `match`, `bmatch`

Description

Reads any `xc` command which does not swap states directly from your Verilog source code. For example, you can add `$xc_cmd("verbose 2")` in Verilog source code to enable verbose messages just like you would enter `xc verbose 2` at run-time command prompt.

Note: This task might be used *only* in modules running in software simulation mode, such as the testbench.

\$export_deposit();

Purpose

Sets a variable to a value and propagates immediately.

Category

Verilog system task/CLI command

Syntax

```
$export_deposit (<path_to_variable>);
```

Description

The `$export_deposit` system task tells the compiler to prepare the DUT to export the *variable* for `$export_deposit` at run time. Once the *variable* is exported for deposit, you can, at run time, issue `$export_deposit` from the CLI (command line interface) or from the C function call `exportDeposit()`.

Note: This has no VHDL equivalent.

\$export_event();

Purpose

Specifies the event on which the testbench and DUT need to synchronize at run time when the tbrun run-time mode is being used.

Category

Verilog system task/CLI command

Syntax

```
$export_event (<event_name>);
```

Description

The `$export_event` system task tells the compiler to synchronize the DUT with the testbench whenever the specified *export event* occurs at run time. Whenever the specified event occurs at run time, the output events generated from the DUT will be processed and propagated into the testbench. Note that the tbrun mode can run close to peak emulator speed if the testbench synchronization events happen at low frequency and low processing overhead is required.

\$initmem();

Purpose

Initializes one or more memory locations on a Verilog memory instance with a desired pattern

Category

Verilog task/CLI command

Syntax

```
$initmem(value, Verilog_ memory_instance_path[, start_address[, end_address]]);
```

Parameters

<i>value</i>	Might consist only of 0 and 1 binary or hexadecimal values.
<i>start_address</i> <i>end_address</i>	Use <i>start_address</i> and <i>end_address</i> to initialize specific locations, or omit them to initialize the entire memory.

\$export_frcrel(); | export_frcrel();

Purpose

Causes a signal to be exported from the emulator.

Category

Verilog system task/CLI command: `$export_frcrel()` ;

VHDL procedure: `export_frcrel()` ;

Syntax

```
$export_frcrel(path_to_signal[,path_to_signal ... ]);  
export_frcrel("path_to_signal");
```

Parameters

<i>path_to_signal</i>	A full hierarchical reference to a DUT signal using periods as delimiters, that is, <i>test_bench.dut.inst1.inst2.sign_name</i> .
-----------------------	---

Description

The `$export_frcrel` Verilog system task/CLI command and `export_frcrel` VHDL procedure causes a signal to be exported from the emulator, which is required if you want to force and release the signal during Simulation Acceleration modes, such as through the CLI, or from a C/C++ program. Special force logic is created for these signals in the hardware.

Exporting signals is often done in conjunction with creating a Shared DUT library.

Note: This task/command/procedure might be used only in modules running in software simulation, such as the testbench.

Example

```
initial  
begin
```

UXE Command Reference Manual

Commands and System Tasks for SA Mode

```
$export_frcrel(tb.dut.sram.chip_enable);  
$export_frcrel(tb.dut.u_proc.cmd);  
end
```

\$export_read(); | export_read();

Purpose

Causes a signal to be exported from the emulator

Category

Verilog system task/CLI command: `$export_read()` ;

VHDL procedure: `export_read()` ;

Syntax

```
$export_read(path_to_signal[,path_to_signal ... ]);  
export_read("path_to_signal");
```

Parameters

path_to_signal A full hierarchical reference to a DUT signal using periods as delimiters, that is,
test_bench.dut.inst1.inst2.sign_name.

Description

The `$export_read` Verilog system task/CLI command and `export_read` VHDL procedure causes a signal to be exported from the emulator, which is required if you want to read the signal during Simulation Acceleration modes, such as through the CLI, or in a C/C++ program.

Exporting signals is often done in conjunction with creating a Shared DUT library.

Note:

- This task/command/procedure might be used only in modules running in software simulation, such as the testbench.
- This procedure is defined in the `axis_util` package, which is compiled in the AXIS library, so specify this library/package with the `use` clause if you are going to use these procedures.

UXE Command Reference Manual

Commands and System Tasks for SA Mode

- Quotes ("") are required for the VHDL procedure, but quotes cause an error in the Verilog system task/CLI command.
- Signals bound to the `ixc_clkbind` primitive cannot be exported using `$export_read()`. If you try this, IXCOM generates an error.

Example

```
initial
begin
    $export_read(tb.dut.sram.chip_enable);
    $export_read(tb.dut.u_proc.cmd);
end
```

Supported IEEE-Standard Non-CLI System Tasks

The following IEEE-standard system tasks are fully supported, but you cannot enter them at the CLI. Please see *IEEE 1364-1995/2001* (for Verilog HDL) for a detailed description of these system tasks.

\$bitstoreal	\$realtobits
\$itor	\$rtoi
\$stime	\$time
\$realtime	\$timeformat
\$dist_uniform	\$getpattern
\$recrem	

Supported IEEE-Standard Compiler Directives

The following IEEE-standard compiler directives are supported for Verilog. Please see *IEEE 1364-1995/2001* (for Verilog HDL) for a detailed description of these directives.

'celldefine	'endcelldefine
'default_nettype	'define
'ifdef	'else
'endif	'include
'protect	'endprotect
'resetall	'timescale
'unconnected_drive	'nounconnected_drive
'undef	'uselib

Verilog Primitives and VHDL Procedures

The following sections describe the Verilog primitives and VHDL procedures:

- [ixc_pulse\(out, in\);](#) on page 663
- [axis_tbcall\(\);](#) on page 664

ixc_pulse(out, in);

Purpose

Lets you generate pulses of the width of a simulation delta cycle to trigger **axis_tbcall**.

Category

Verilog primitive

Syntax

```
ixc_pulse(out, in);
```

Description

For every change on input `in`, the output `out` pulses from 0 to 1 at the beginning of the timestep and then goes back to 0 at the end of the timestep.

In Simulation Acceleration modes, `ixc_pulse` generates a pulse that lasts until the end of the emulator evaluation.

The output of this primitive should not be gated and should be used directly to drive synchronous elements like flip-flops or synchronous memories.

Note: Verilog only

axis_tbcall();

Purpose

Enables you to access the software simulator during Simulation Acceleration modes to execute non-synthesizable code, such as C/C++ code or to perform Operating System calls, such as loading memory files, and so on.

Category

- Verilog primitive
- VHDL procedure

Syntax

Verilog: axis_tbcall(callback_signal, "callback_task");

Verilog: axis_tbcall #(1) (assert_signal, "assert_task");

VHDL: axis_tbcall(callback_signal, "callback_procedure");

Parameters

<i>callback_signal</i>	Must be a 1-bit scalar signal (wire or register) in the DUT driven by a flip-flop directly or by a chain of buffers (such as <i>buf</i> , <i>not</i> , <i>ixc_pulse</i>) driven by a flip-flop. Xtrigger output (dynamic triggers) can also be used.
<i>callback_task/ callback_procedure</i>	Called when <i>callback_signal/assert_signal</i> goes posedge at the end of the timestep. It is a local Verilog task or VHDL procedure defined in the current module scope, but can nest a task call to any other scope using a cross-module reference (XMR).
<i>assert_task/assert_procedure</i>	Must be enclosed with double quotes

Description

In Verilog, `axis_tbcall` is a built-in primitive; in VHDL, it is a procedure call. This Testbench Callback feature enables you to access the software simulator during Simulation Acceleration modes to execute non-synthesizable code, such as C/C++ code or to perform Operating System calls, such as loading memory files, and so on.

The values read by the task/procedure are end-of-timestep values, that is, after clocked registers are updated:

```
always@(posedge clk) begin
    bcount <= bcount + 1;
    d_bcount <= bcount;
end
axis_tbcall(d_bcount, "my_cb");
task my_cb; // called on a signal changing on posedge clk
begin
    $display("bcount:%b", bcount); // will display updated value!
    $display("d_bcount:%b", d_bcount); // displays value @ posedge
end
endtask
```

Do not drive the signal directly by combinatorial logic, as glitches on the signal could trigger `axis_tbcall` at unexpected times, both in the emulator and software simulator. A flip-flop can filter out the glitches. For example, the following situation would not be allowed:

```
reg[3:0] A, B;
wire signal=(A>B); // not a good axis_tbcall signal
```

But you could correct it by setting the signal under a clock:

```
always@(posedge clk)
    signal = (A>B);
```

You can also use `axis_tbcall` to report assertion violations; the syntax has an additional `#(1)` parameter to alert the compiler that the `axis_tbcall` is assertion-driven, so it will not connect to software using primary output pins (which, because of the potential volume of assertions, could slow down xc on -run significantly).

Note

- Do not reference any design variables in an assertion-driven `axis_tbcall` task. If you reference a design variable in a normal `axis_tbcall` task, the variable is read from the emulator to the software task using primary output pins. In assertion-driven `axis_tbcall`, design variables are not linked from the emulator into software. As a

UXE Command Reference Manual

Commands and System Tasks for SA Mode

result, you will see out-of-date values if you read design variables inside an assertion-driven `axis_tbcall` task.

In case you do need to report a design variable in an assertion-driven `axis_tbcall` task, you must first do `xc off` to hot-swap the emulator back to software, then access the design variable.

SVA Options to IXCOM

Here are the options to be used when you are incorporating SystemVerilog Assertions into your designs.

- [+sva+](#) on page 668
- [+no_sva](#) on page 669
- [+assertCoverCnt](#) on page 671
- [+assertCheckedCntr+<N>](#) on page 672
- [+assertFinishedCntr+<N>](#) on page 673
- [+assertFailedCntr+<N>](#) on page 674
- [+assertCover](#) on page 670
- [+assertStatus](#) on page 675
- [+assertFailCounter+hw](#) on page 676
- [-coverage \[block | toggle | functional | all\]](#) on page 677
- [+fcov_uxc_db](#) on page 679

These options are valid during the compilation step (that is, when you execute `ixcom`). For more information about compiling designs for SA mode, refer to the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

+sva+

Purpose

Enables SVA for only specific SystemVerilog modules when specified with the `+no_sva` option.

Note: SVA is enabled for all modules by default.

Category

SVA ixcom / vlan option

Syntax

```
ixcom | vlan other_options +no_sva +sva+module+module ...
```

Description

The `+sva+` option to `ixcom` or `vlan` lets you specify only certain modules for SVA when used with the `+no_sva` option.

+no_sva

Purpose

Disables SVA for:

- all SystemVerilog modules, when the `+no_sva` option is specified without any module names
- specific SystemVerilog modules, when the `+no_sva+<module>[+<module>...]` option is specified
- all modules except the ones specified with the `+sva+` option, that is,
`+no_sva +sva+<module>[+<module>...]`

Category

SVA `ixcom / vlan` option

Syntax

```
ixcom | vlan other_options +no_sva [+module [+module ...]] [+sva [+module [+module]]]
```

Description

The `+no_sva` option to `ixcom` or `vlan` lets you disable SVA for all SystemVerilog modules or for only the modules specified when used with the `+sva+` option.

+assertCover

Purpose

Enables display of finished, failed, or checked counts at run time for cover counters of `assert` and `assume` assertion statements.

Category

SVA / PSL `ixcom` option

Syntax

`ixcom other_options +assertCover`

Description

The `+assertCover` option generates assertion coverage data (checked, passed, and failed) for `ASSERT` and `ASSUME` statements. Coverage data is provided for `COVER` statements by default.

Note: The assertion summary is no longer generated at the end of simulation in IXCOM when this switch is used. Use the following run-time command to view assertion coverage summary at the end of the simulation:

```
assertion -summary
```

+assertCoverCnt

Purpose

Specifies the number of bits for coverage counters.

Category

SVA / PSL ixcom option

Syntax

```
ixcom [other_options] +assertCoverCnt+number
```

Description

The `+assertCoverCnt` option to `ixcom` specifies the number of bits (up to 32) for coverage counters: checked (satisfied all antecedents), passed, and failed.

The default is 1 bit, which provides information on whether the assertion has ever checked, passed or failed at least once. The values set for the `+assertCheckedCntr`, `+assertFinishedCntr`, and `+assertFailedCntr` override the setting made by the `+assertCoverCnt` option.

+assertCheckedCntr+<N>

Purpose

Specifies the number of bits for the width of the Checked counter.

Category

SVA / PSL ixcom option

Syntax

`ixcom [other_options] +assertCheckedCntr+number`

Description

Use this option to set the width of the Checked counter to the value <N>. The value of <N> should be in range from 1 to 32. By default, the value of the Checked counter width is 1. If you have not used the `+assertCover` option of the `ixcom` command, the `+assertCheckedCntr+<N>` option applies only to the assertions with cover directives. The value set for this option overrides the setting made by the `+assertCoverCnt` option.

+assertFinishedCntr+<N>

Purpose

Specifies the number of bits for the width of the Finished counter.

Category

SVA / PSL ixcom option

Syntax

`ixcom [other_options] +assertFinishedCntr+number`

Description

Use this option to set the width of the Finished counter to the value <N>. The value of <N> should be in the range of 1 to 32. By default, the value of the Finished counter width is 1. If you have not used the `+assertCover` option of the `ixcom` command, the `+assertFinishedCntr+<N>` option applies only to the assertions with cover directives. The value set for this option overrides the setting made by the `+assertCoverCnt` option.

+assertFailedCntr+<N>

Purpose

Specifies the number of bits for the width of the Failed counter.

Category

SVA / PSL ixcom option

Syntax

```
ixcom [other_options] +assertFailedCntr+number
```

Description

Use this option to set the width of the Failed counter to the value <N>. The value of <N> should be in the range of 1 to 32. By default, the value of the Failed counter width is 32. In the presence of the +assertFailCounter+hw option, the default width of the Failed counter is set to 1. If you have not used the +assertCover option of the ixcom command, the +assertFailedCntr+<N> option does not have any effect. The value set for this option overrides the setting made by the +assertCoverCnt option.

+assertStatus

Purpose

Generates instrumentation to enable probing and run-time control of assertions.

Category

SVA / PSL `ixcom` option

Syntax

`ixcom other_options +assertStatus`

Description

The `+assertStatus` option to `ixcom` generates pass/fail/pending signals for assertions. If this option is not specified, the following happens

- Pending signals are not generated.
- Passed and failed signals are generated only if their respective action blocks are specified in the assertion.
- Control signals are not generated, which means assertion cannot be disabled at run time.

+assertFailCounter+hw

Purpose

Ensures that the failure counters for assertions are available when running the design in the Dynamic Target mode.

Category

SVA ixcom option

Syntax

```
ixcom other_options +assertFailCounter+hw
```

Description

Use of this option along with the `+assertCover` option maps the failure counters to the emulator.

By default, the failure counters and failure action blocks for assertions are mapped to the software simulator. In the Dynamic Target mode, the design clocks and the emulator cannot be stopped to swap to the software simulator for processing assertions. Use of the `+assertFailCounter+hw` option ensures that the failure counts are available on the emulator side. The failure action blocks still reside on the software simulator.

For more information, see *Using Assertions in Dynamic Target Mode* in the *Compiling and Running Designs with Assertions* chapter of the *UXE User Guide*.

-coverage [block | toggle | functional | all]

Purpose

Enables coverage.

Category

Compiler/Coverage option

Syntax

```
ixcom other_options -coverage [block | toggle | functional | all]
```

Description

Enables the ability to view coverage information using the Incisive Metrics Center (IMC) tool. Alternatively, the shorthand -coverage [b | t | u | a] option can be used to achieve the same results.

- **block** - enables block coverage
- **toggle** - enables toggle coverage
- **functional** - enables functional coverage
- **all** - enables all types of coverage

The -coverage option to ixcom enables the dumping of coverage information into a coverage database. This includes coverage information related to SystemVerilog / PSL cover directives and SystemVerilog covergroup instances. In addition, information related to assert directives can be included by using the +assertCover option.

The coverage database generated at run time is an ICC compatible database. Users can interact with the database using Incisive coverage tools in the INCISIVE release tree. More information can be found in the INCISIVE documentation for *Incisive Metric Center*.

-covMakeNamesUnique

Purpose

Enables IXCOM to generate unique module names.

Category

Compiler/Coverage option

Syntax

```
ixcom other_options [-covMakeNamesUnique]
```

Description

Due to parameterization and other aspects that make design units unique, modules might appear multiple times in the type-based report. Each appearance represents a different compile-time configuration of that design unit. IMC does not summarize this information automatically when this occurs. To avoid errors during advanced type-based operations in IMC and vManager tools that expect the module names to be unique, you can generate unique module names in the UXE Unicov database by using the `-covMakeNamesUnique` option. With this option, the IXCOM compiler appends a unique identifier to the module name written into the UXE Unicov database, if that module name already exists in the database. If a module is named `mod1`, a unique name is generated like `mod1#(0x6326652f)`. Here, the number `0x6326652f` is unique to this kind of module `mod1`.

Refer to the *Generating Unique Module Names for Type-Based Coverage* section in the *UXE User Guide* for details.

+fcov_uxe_db

Purpose

Enables functional coverage to be generated by IXCOM compiler.

Category

Compiler/Coverage option

Syntax

```
ixcom [other_options] [-coverage functional +fcov_uxe_db]
```

Description

Enables functional coverage (assertions and covergroups) database to be generated by the UXE software instead of the IES tool. This option ensures that the functional coverage results contain the original file information. This option was used as a transition mechanism. It is no longer required.

PSL Options to IXCOM

Here are the options that you might need to use when you are incorporating PSL assertions into your designs.

- [+assertCover](#) on page 670
- [+assertCoverCnt](#) on page 671
- [+assertStatus](#) on page 675
- [-coverage \[block | toggle | functional | all\]](#) on page 677
- [+fcov_uxe_db](#) on page 679
- [+no_psl](#) on page 681
- [-psl](#) on page 402
- [-assert](#) on page 403
- [-assert_vlog](#) on page 404
- [-assert_vhdl](#) on page 405
- [-assert_vhdl](#) on page 405
- [-noassert](#) on page 406

These options are all required at the compilation step (that is, when you execute `ixcom`). For more about compiling designs for SA mode, refer to the *Compiling Designs for Simulation Acceleration Using IXCOM* chapter of the *UXE User Guide*.

+no_psl

Purpose

Disables elaboration of PSL assertions in the specified modules or the entire design.

Category

Compiler option

Syntax

`+no_psl [+<module>]`

UXE Command Reference Manual
Commands and System Tasks for SA Mode

Compile-time Commands in ICE Mode

This chapter describes the XEL commands used to compile in-circuit designs for emulation.



Caution

Do not run any commands documented in this chapter from the XEL command line in the Palladium XP Debugger utility. Doing so will cause erratic results and might even damage the design database.

The following XEL commands are covered in this chapter:

- “[checkCPFfile](#)” on page 684
- “[check_power_intent](#)” on page 685
- “[compile](#)” on page 686
- “[compileFind](#)” on page 689
- “[design](#)” on page 705
- “[disconnectDriverPin](#)” on page 707
- “[getCriticalPathDesignNetNames](#)” on page 708
- “[icePrepare](#)” on page 709
- “[precompile](#)” on page 710
- “[readCPFfile](#)” on page 712
- “[read_power_intent](#)” on page 714
- “[clean_power_intent](#)” on page 716
- “[referenceMap](#)” on page 717
- “[shrinkDB](#)” on page 718

checkCPFfile

Invokes the Conformal Low Power (CLP) integrator for checking the syntax of CPF commands included in the specified *<cpfFileName>*. For command syntax and description of the supported CPF commands, refer to [Chapter 12, “Common Power Format Commands.”](#)

It is better to run the `checkCPFfile` command before the `readCPFfile` command. This helps you in leveraging the CPF linter of the CLP integrator to check the syntax errors first.

In the ICE mode, the `checkCPFfile` command can be used for both hierarchical and flattened-CPF files.

The syntax for this command is:

`checkCPFfile <cpfFileName>`

<cpfFileName>

Specifies the name of the CPF file that needs to be checked for syntax.

check_power_intent

Invokes the Cadence 1801 linter to check the syntax of IEEE 1801 commands included in the specified *<1801Filename>*. For supported IEEE 1801 commands and command options, refer to *Using IEEE 1801 to Specify Low-Power Intent on Palladium XP in UXE User Guide*.

The syntax for this command is:

```
check_power_intent <1801Filename>  
-version
```

<1801Filename>

Specifies the name of the 1801 file that needs to be checked for syntax.

-version

Returns the 1801 linter version.

compile

Compiles the design for emulation. While the design is being compiled, messages are displayed about the status of each phase of design compilation (such as partitioning the design, and whether each phase completed successfully or failed).

Before using this command, execute the [design](#) command to specify the design library and cell name. Use [compilerOption](#) to set up special conditions, then run [precompile](#). The [precompile](#) command must have completed successfully on the design database.

The syntax for this command is:

```
compile [-max_steps <number>]  
        [-min_steps <number>]  
        [-compactDCC]  
        [-crit_paths <number>]  
        [-criticalPathOnly [<number>]]  
        [-names_crit]  
        [-seed random | last | lastpass | <number>]  
        [-etOnly]
```

-max_steps <number>

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* argument must be a multiple of 2. The range for -max_steps is 1-4096.

-min_steps <number>

The -min_steps option is usually a complement to the -max_steps option. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both -min_steps and -max_steps are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified with the -min_steps option should not be greater than the number specified with the -max_steps option.

The *<number>* specified in -min_steps must be a multiple of 2. The range for -min_steps is 1-4096.

-compactDCC

Instructs the compiler to implement probes in the smallest possible number of DCC memories.

By default, the compiler implements probes using the full set of available DCC memories. When the `-compactDCC` option is specified, the time to upload waveforms in DYNP mode can be significantly improved because fewer DCCs are used. However, the `-compactDCC` option increases the chance of probe routing failure, both at compile time, and when Dynamic Probes are added at run time.

-crit_paths <number>

Specifies number of critical paths to be reported in the `tmp/et3compile.msg` file. By default, the compiler reports one critical path.

This command option does not affect design compilation in any other way.

For information about the format of a critical path report, refer to the *Finding the Reason for a Poor Step Count* section of the *Improving Emulation Speed* chapter in *UXE User Guide*.

-criticalPathOnly [<number>]

Generates a critical path report in the `tmp/et3compile.msg` file after the design has been compiled. The report is generated in the same format as done using the `compile -crit_paths <number>` command.

Note: The `-criticalPathOnly` option cannot be specified with any other options of the `compile` command.

If the `<number>` parameter is not specified with the `-criticalPathOnly` option, the default value 100 is used.

-names_crit

Prints critical paths net names in the same manner as the [getCriticalPathDesignNetNames](#) command.

If this option is present, the `getCriticalPathDesignNetNames` command is executed on all the `tmp/*.msg` files created by the current `compile` command.

-seed random | last | lastpass | <number>

Specifies a seed value to use during compilation. The seed value influences logic partitioning and affects the success rate of compilation. The `random` keyword tells the compiler to select the seed value. The `last` keyword tells the compiler to use the seed from the most recent compile, if available. The `lastpass` keyword tells the Compiler to use the seed from the most recent successful run, if available. The `<number>` argument enables you to specify your own seed value (any non-negative integer).

The default keyword is `random`.

For the first compile of a design and all compiles subsequent to a successful compile, use the `lastpass` option. When a compile fails, you can repeat the command using different seed values in a range from 0 to 100, until a successful compile is achieved. To repeat a failed compile, simply use the `last` option. The seed used to compile a design is always logged to the `<design_dir>/tmp/xeCompile.log` file for future reference.

Re-compilation with the same seed `<number>` should produce results identical to the original compilation if the re-compilation starts from ET compiler (`compile -etOnly`).

If you see varying compilation results, for IXCOM-compiled design, you can use the following commands:

```
% xeCompile  
XEC> design  
XEC> compile -etOnly -seed <number>
```

-etOnly

Disables flow control checking. By default, the `compile` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

Examples

```
design my_lib my_cell  
compile -min_steps 150
```

Compiles the `my_cell` design to run with a minimum step count of 150.

```
design my_lib my_cell  
compile -cycles 4 -seed last
```

Compiles the `my_cell` design (located in the `my_lib` library) for emulation in four cycles and uses the seed value used in the most recent compile.

compileFind

Runs a compile for the given number of trials, records the results of each run to a log file, and reports the success rate and best result based on the criteria defined using the different command options discussed below.

By default, the `compileFind` command runs the given number of trials with fixed values of compile parameters defined by you or default values corresponding to each option. For example, if the maximum number of trials is not specified, the compile is run for 30 trials by default.

In addition, by default the `compileFind` command saves the best result for any option, with or without the multiple host option, unless you specify the `-no_export` option.

Before using this command, execute the `design` command to specify the design library and cell name. Also, execution of the `precompile` command must complete successfully on the design database.

The syntax of this command is:

```
compileFind [-first | -best | -all]
            [-lsf [-keep_all] [-rerun_best] [-link <filename>] [-timeout
            <number>]]
            [-opt [-oE {<values>}] [-pE {<values>}]]
            [-mem [-max_util <number>] [-min_util <number>] [-dec_util <number>]]
            [-max_trials <number> | -max_trials_each <number>]
            [-seed <number>]
            [-max_steps <number>]
            [-min_steps <number>]
            [-crit_paths <number>]
            [-etOnly]
            [-et3pass2]
            [-1pass | -2pass]
            [-verify]
            [-logfile <logfile>]
            [-names_crit]
            [-no_export]
            [-compactDCC]
            [-file <file>]
            [-normal]
```

Note:

- ❑ Unambiguous shortcuts are allowed, for example, `-2` instead of `-2pass`, or `-k` instead of `-keep_all`. However, ambiguous shortcuts are not allowed, such as `-min` that can be either `-min_steps` or `-min_util`. Also, a shortcut that is currently unambiguous can become ambiguous in the future.

- ❑ At most, one of the options `-first`, `-best`, and `-all` is allowed. At most one of the options of `-file`, `-opt`, and `-mem` is allowed. If incompatible options are executed with the `compileFind` command, or an option has an unacceptable value, the command fails with the corresponding error message.

-all

Runs the given number of trials, collects and prints compile statistics such as success rate, best step count, and average step count. The best result is the one with the minimal step count that results in fastest emulation speed.

This command option enables identification of the optimal hardware configuration to be recommended for initial purchase or an upgrade.

For example, if you have a 12M ASIC gate design that is initially configured for a machine, PXP5, run 100 trials with starting seed 1 (that is, seeds 1...100) and record the results. You can then increase the hardware configuration (using the Virtual Hardware Configuration feature) to PXP6, and repeat the evaluation. The results of such trials can be plotted into a table such as the following:

Table 11-1 Identifying a Suitable Hardware Configuration

Hardware Configuration	Cost	Success Rate	Emulation Speed			
			Best	Median	Average	Worst

This information lets you choose the hardware configuration with the optimum price/performance point to order.

-best

Runs a compile for the given number of trials and chooses the compile with the best emulation speed (that is, successful compile with minimal step count). The scheduler uses the early exit strategy. If while scheduling at the current trial it becomes clear that the step count cannot be better than the one already achieved, the current trial will be immediately aborted.

By default, the `-best` option is used if none of the `-first`, `-best`, or `-all` options are specified.

Note: If the `-lsf` option is also specified with the `-best` option, then early exit strategy is not used.

-first

Runs a compile for the given number of trials and stops after the first successful compile.

Note: Using this option with the `-lsf` option terminates the remaining trials as soon as one of the trials succeeds. Consequently, the `compileFind {-lsf -first}` command executes similar to the `compileFind -first` command rather than the `compileFind -best` command.

-lsf

Performs the required number of back-end compiles (trials) with different seeds on different machines in parallel using LSF and chooses the best trial (the one with the minimum step count).

By default, all trials are performed sequentially, on the same host machine. The `-lsf` option forces the `compileFind` command to run the trials on multiple hosts using LSF.

Note:

- ❑ If the `-lsf` option is specified with the `-first` option, as soon as one of the trials succeeds, the other trials are terminated.
- ❑ If the `-lsf` option is specified but LSF is not in the path, the command fails with the following error message:

```
Cannot find executable 'bsub' necessary for '-lsf' option. Check your path.
```
- ❑ When the `-lsf` option is specified, there is no difference between the `-all` and `-best` options of the `compileFind` command.
- ❑ When the `compileFind` command is specified with both `-opt` and `-lsf` options, the required number of back-end compiles (trials) are performed with different seeds and different combinations of optimization effort and placement effort on different machines in parallel using LSF and the best trial (the one with the minimum step count) is chosen. This speeds-up the iterative compiles because being executed sequentially they would take time for big designs. It is a fast parallel alternative to the `compileFind -opt` command.
- ❑ The `-lsf` option is incompatible with the `-mem` option.

- ❑ The `compileFind -lsf` command does not kill all LSF jobs associated with the command when the `xeCompile` process is killed. However, it generates a file named `killAllLSFjobs` in the current directory.

To kill the LSF jobs, give the following command on the UNIX command line prompt:

```
source ./killAllLSFjobs
```

-mem

Runs a compile for the given number of trials for each value of memory utilization defined by the you, starting from a maximum memory utilization value (`max_util`), with given decrements (`-dec_util`), down to a minimum value (`min_util`). The number of compile runs can be defined either explicitly or implicitly using the switches provided with this command.

For each trial run, a record will be generated in the log file. Each record will contain the memory utilization value, the number of the trial, the seed value, the compilation result (pass/fail), and the step count. In addition, the success rate, the average step count, and the gate count for each memory utilization value will also be recorded.

There will also be a summary report similar to that for the `compileFind -all` command.

For each trial compile run `<N>` (where $N=1,2,3\dots$) and each memory utilization value `<U>`, there will be an output file `tmp/et3pass2.util<U>. <N>.msg`. For example, for third trial compile run for memory utilization value 80, the file generated will be `tmp/et3pass2.util80.3.msg`.

This command option enables you to explore the effects of changing the memory utilization on the compile success rate, capacity, and step count.

To run the `compileFind -mem` command with some compiler options related to memory compaction, you must previously run either of the following commands with the required compilerOption command options:

- compile
- compileFind -all
- compileFind -best
- compileFind -first

Note: The `compileFind -mem` command can be very time consuming.

-normal

Ensures that the `compileFind` command for Palladium XP II, compiles the design only for Palladium XP II, and not Palladium XP. By default, to identify the best step count for Palladium XPII, the `compileFind` command for Palladium XP II, might in certain situations, compile the design for both for Palladium XP and XPII. This is done to guarantee that the step count for XPII is no greater than for XP. Compile times can take longer as a result. You can avoid extra compilation by specifying the `-normal` option. For more information, see the *Changes in compileFind with XP II Features* section in the *UXE User Guide*.

-opt

Runs a compile for the given number of trials for each pair of values of `optimizationEffort` and `placementEffort` defined by you. This command enables you to explore the effects of changing the optimization parameters on the compile success rate, capacity, and step count.

For each trial run, a record will be generated in a log file containing the following information:

- Date and time
- `optimizationEffort` value
- `placementEffort` value
- Number of the trial
- Seed value
- Compilation result (that is, `pass` or `fail`)
- Step count

The log file will also include the following additional information for each pair of `optimizationEffort` and `placementEffort` values:

- Success rate
- Average step count
- Gate count

There will also be a summary report similar to that generated for the `compileFind -all` command.

For each trial compile run $<N>$ (where $N=1, 2, 3\dots$) for each pair of optimizationEffort and placementEffort values, $<OE>$ and $<PE>$, there will be an output file named the following:

`tmp/et3pass2.oe<OE>.pe<PE>.<N>.msg`

For example, for third trial compile run for optimizationEffort value ULTRA and placementEffort value LOW, the file generated will be named
`tmp/et3pass2.oeULTRA.peLOW.3.msg`.



The `compileFind -opt` command can be very time consuming.

-max_trials <number>

Specifies the number of trial compiles to run.

This number must be a positive integer. The default value is 30.

Note: The `-max_trials` option is incompatible with the `-opt` and `-mem` options; use the `-max_trials_each` option instead.

-max_trials_each <number>

Specifies the number of trial compiles to run for each pair of optimizationEffort and placementEffort values when used with the `-opt` option. Specifies the number of trial compiles to run for each value of memory utilization when used with the `-mem` option.

This number must be a positive integer. The default value is 5.

-max_steps <number>

Specifies the maximum number of steps that can be accepted by the compiler. The $<number>$ argument must be a multiple of 2. The range for `-max_steps` is 1-4096. By default, this option is not defined.

-min_steps <number>

Sets the minimum number of steps for the compiler scheduler. The `-min_steps` option is usually a complement to the `-max_steps` option. By default, this option is not defined.

Note: If a trial results in step counts less than the given minimum step count, it is automatically increased to the given number.

In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both `-min_steps` and `-max_steps` are set to the same number, the design runs at this exact step count (if it can be scheduled at this step count). The number specified with the `-min_steps` option should not be greater than the number specified with the `-max_steps` option.

The number specified with the `-min_steps` option must be a multiple of 2. The range for `-min_steps` is 1-4096.

-seed <number>

Specifies the seed number for the first trial that is incremented by one for every trial. This number must be a positive integer.

By default, this option is not defined. As a result, each trial has an arbitrary seed based on the current date and time. If the `-seed` option is specified, trials run with seeds `<number>`, `<number+1>`, `<number+2>`, ..., where `number` is greater than 0.

-logfile <logfile>

Specifies the name of log file in which the summary statistics should be recorded. By default, the summary statistics is written into the `./tmp/compile.log` file.

-compactDCC

Instructs the compiler to implement probes in the smallest possible number of DCC memories.

By default, the compiler implements probes using the full set of available DCC memories. When the `-compactDCC` option is specified, the time to upload waveforms in DYNP mode can be significantly improved because fewer DCCs are used. However, the `-compactDCC` option increases the chance of probe routing failure, both at compile time, and when Dynamic Probes are added at run time.

-crit_paths <number>

Specifies the number of critical paths to be reported for each successful trial compile. By default, the compiler reports one critical path. The critical path information for the *<i>*th successful trial is stored in file `./tmp/et3pass2.<i>.msg`, where *<i>* is the number of successful trial compiles.

For example, by default, the `compileFind -all` command runs 30 compile iterations. In this case, for the first compile result, the critical path information will be stored in a file called `./tmp/et3pass2.1.msg`. Similarly, for the 29th compile result, the critical path information will be stored in the file `./tmp/et3pass2.29.msg` and for the 30th compile result, the information will be stored in the `./tmp/et3pass2.30.msg` file.

This switch does not affect design compilation in any other way.

-names_crit

Prints critical paths net names in the same manner as the [getCriticalPathDesignNetNames](#) command.

If this option is present, the `getCriticalPathDesignNetNames` command is executed on all the `tmp/*.msg` files created by the current `compileFind` command.

-max_util <number>

Specifies the maximum value of memory utilization. This number must be a positive integer. The default value is 100.

Note: The `-max_util` option can be specified only with the `-mem` option of the `compileFind` command.

-min_util <number>

Specifies the minimum value of memory utilization. This number must be a non-negative integer. The default value is 50.

Note: The `-min_util` option can be specified only with the `-mem` option of the `compileFind` command. In addition, the value of `-min_util` **must not** exceed the value of `-max_util`.

-dec_util <number>

Specifies the number by which decrements should be made in the memory utilization value during the trial compile runs. This number must be a positive integer. The default value is 10.

Note: The `-dec_util` option can be specified only with the `-mem` option of the `compileFind` command.

-oE {<values>}

Specifies the list of values for `optimizationEffort`. Possible values are any subset of {LOW MEDIUM HIGH ULTRA CLASSIC}. The default list is {LOW HIGH}. Any shortcut is acceptable and characters are case-insensitive. For example, H HI HIGH h hig are all equivalent.

Note: The `-oE` option can be specified only with the `compileFind -opt` command.

-pE {<values>}

Specifies the list of values for `placementEffort`. Possible values are any subset of {LOW HIGH}. The default list is {LOW HIGH}. Any shortcut is acceptable and characters are case-insensitive.

Note: The `-pE` option can be specified only with the `compileFind -opt` command.

-etOnly

Disables flow control checking. By default, the `compileFind` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

Note: By default, this option is not enabled.

-et3pass2

Skips `et3compile` pass 1 and runs trials starting with `et3compile` pass 2. Use this option when you have run a set of trials using `compileFind`, and you want to run additional trials. To use the `-et3pass2` option, a set of pass 1 output files (`dbFiles/et3pass1.proto*`) must have been generated during a previous (even if unsuccessful) compilation. This option is

incompatible with the `-opt`, `-mem`, and `-file` options. Like `-etOnly`, the `-et3pass2` option disables flow-control checking.

-no_export

Restricts saving of the final prototype files at the end of trials. This option might be useful for experiments, such as collecting compile-related statistics.

Note: By default, this option is not enabled.

-1pass

Sets the *one-pass* mode that forces execution of each trial in one pass.

By default, back-end compile is split into two parts. The first part (from the beginning to `mpart`) is *seed-independent* and executed only once, in the design directory, without LSF. At the end the intermediate proto file is saved. Then the second, seed-dependent part (from `upart` to the end) is executed with LSF.

Note: This option is ignored when both `-opt` and `-lsf` options are present. In the other cases, the *one-pass* mode can be used for debugging.

-2pass

Sets the *two-pass* mode that forces execution of each trial in two passes.

On the first pass, the *seed-independent* part of the back-end compile is executed. As it is seed-independent, it is executed only once. On the second pass, the remaining back-end compile that is *seed-dependent* (such as placement, scheduling, and so on.) is executed.

Note: This option is ignored when both `-opt` and `-lsf` options are present. In the other cases, the *two-pass* mode is the default.

Use of this option makes the compile process to run faster. However, if compile is run again using the same options and seed, the compile speed can be different.

-keep_all

Blocks the removal of the temporary subdirectories and files.

This option can be used only when the `-lsf` option is present. It enables you to preserve all databases created on the multiple hosts and can be useful for debugging.

Note: By default, this option is not enabled.

-rerun_best

Restricts saving of the final prototype files at the end of trials. Instead, the best trial is reran in the design directory.

Note: The `-rerun_best` option can be used only when the `-lsf` option is present. By default, this option is not enabled.

Using this option slows down the compile but might be useful if the disc space is limited because final prototype files need not be saved.

-verify

Restricts the execution of the `compileFind` command such that only its syntactic control is performed.

-timeout <number>

Instructs the compiler to kill any unfinished parallel compile (referred to as trial) if it was *quiet* longer than the specified *<number>* of minutes. The *nth* trial is considered to be *quiet* longer if it started, but did not update its log file (`ITER_<n>/tmp/e*msg`) for the specified *<number>* of minutes. By default, *<number>* equals 60, that is one hour.

For each killed trial, you will receive messages of the following format:

WARNING: Trial *<n>* was killed due to timeout (no updates for \$timeout minutes).

You might increase the timeout value using '`-timeout`' option of '`compileFind`' command.

If the *<number>* argument is specified as 0, this feature is disabled.

-link <filename>

Enables you to link additional files of the given *<filename>* in the subdirectories of the working directory.

This option is useful when `-lsf` option is specified with the `compileFind` command. In this case, the trials are performed on multiple hosts in subdirectories of the working directory with names `ITER_1`, `ITER_2`, ... In each subdirectory `ITER_i`, directory `dbFiles` is created. The links from each `ITER_i` to some files in the working directory and links from each

UXE Command Reference Manual

Compile-time Commands in ICE Mode

`ITER_i/dbFiles` to some files in subdirectory `dbFiles` of the working directory are created. The list of these links is given below. If you need to link additional files, this command should be used for each of these files.

If `<filename>` exists in the working directory, in each subdirectory `ITER_i` the following link is created:

```
ln -s ../<filename> .
```

If `<filename>` exists under `dbFiles`, in subdirectory `dbFiles` of each subdirectory `ITER_i` the following link is created:

```
ln -s ../../dbFiles/<filename> .
```

The following files (if exist in the working directory) are linked by default from each `ITER_i` subdirectory:

<code>LSF_options</code>	<code>.design</code>	
<code>et3compile.cfg</code>	<code><topcell>.et3cfg</code>	<code><topcell>.ctl</code>
<code><topcell>.cmd</code>	<code><topcell>.compOptions</code>	<code><topcell>.memRules</code>
<code><topcell>.mempreset</code>		

The following files (if exist in the `dbFiles` subdirectory of the working directory) are linked by default from the `dbFiles` subdirectory of each `ITER_i` subdirectory (* is a metacharacter meaning any extension or no extension):

<code>lseedNum</code>	<code>pSeedNum</code>	<code>et3compile.cfg</code>
<code><topcell>.iodelays</code>	<code><topcell>.et3cfg</code>	<code><topcell>.ctl</code>
<code><topcell>.keepnets</code>	<code><topcell>.breaknets</code>	<code><topcell>.compOptions</code>
<code><topcell>.memRules</code>	<code><topcell>.et3sched</code>	<code><topcell>.writethru</code>
<code><topcell>.behav</code>	<code><topcell>.critnets</code>	<code><topcell>.switch</code>
<code><topcell>.qt2dadb.proto*</code>		
<code><topcell>.et3pass1.proto*</code>		

-file <file>

Enables you to run multiple trials for different compile parameters settings defined in the specified `<file>`. The syntax of the `<file>` is as following:

```
<file> ::= <section> {<section>}
<section> ::= <section_name>
           <num_trials>
           <XEL command>
           .....
           .....
           <XEL command>
           --
```

UXE Command Reference Manual

Compile-time Commands in ICE Mode

File contains a non-empty sequence of sections.

The first line of each section is a section name. This name is used in names of the `*msg` files that are created under the `tmp` directory as follows. The section names should be alphanumeric and unique. For two-pass mode, the message file related to the first pass for section with name `<secname>` has the name of the format `et3pass1.<secname>.msg`. The message file related to the i^{th} trial of the second pass (or related to i^{th} trial in one-pass mode) for section with the given name is named as `et3pass2.<secname>.<i>.msg`.

The second line contains number of trials for this section, which should be a positive number. It is followed by zero, one, or several lines, each containing an XEL command. Last line of each section consists of two – (hyphen) characters.

The XEL commands in the file should normally be User Data commands for those user data items that can be changed without a need to re-run the precompile. Following are examples of commands that meet the criteria in the file:

```
compilerOption -add {optimizationEffort <value>}
compilerOption -add {placementEffort <value>}
compilerOption -rm placementEffort
compilerOption -add {traceDepth <value>}
aluTransform -add {<anything legitimate>}
memoryTransform -add {<anything legitimate>}
memWritethru -add {<anything legitimate>}
```

Following are examples of commands that do not meet the criteria in the file (that is, require to re-run the precompile):

```
compilerOption -add {visionMode <value>}
compilerOption -add {precompileOption <value>}
```

The `-file <file>` option implies `-etonly`, even if the `-etonly` option is not given by you.

Any other XEL command can be used on your own risk. Commands such as `precompile` or `compile` are not expected by the `compileFind` command and can produce unpredictable results.

The User Data commands only modify a temporary user data needed for the execution of the given section. The original user data are restored after execution of each section. This means that after executing the `compileFind` command, the user data will be the same as before executing it.

Note: If the `xeCompile` process is killed during the execution of the `compileFind -file <file>` command, the original User Data might not be restored. In such a case, you can restore the original User Data from the `QTDB/ud_keep` directory. Restore it by removing the `QTDB/ud` directory, and rename the `QTDB/ud_keep` directory to `QTDB/ud`.

Following are the contents of a sample file that you can specify using the `-file` option:

```
S1
5
compilerOption -add {optimizationEffort LOW}
compilerOption -add {placementEffort LOW}
memoryTransform -add {* memoryUtilization 80}
--
S2
3
compilerOption -add {optimizationEffort HIGH}
compilerOption -add {placementEffort HIGH}
memoryTransform -add {* memoryUtilization 80}
--
S3
3
compilerOption -add {optimizationEffort HIGH}
compilerOption -add {placementEffort HIGH}
memoryTransform -add {* memoryUtilization 90}
--
```

Examples

- To run 30 (default) compiles with sequential seeds starting with a random seed and log results in the `./tmp/xeCompile.log` file, specify:

```
compileFind -all
```

- To run 50 compiles with sequential seeds starting with 759 and log results in the `./tmp/xeCompile.log` file, specify:

```
compileFind -all -max_trials 50 -seed 759
```

- To run 100 compiles and log results in the `./tmp/run_100.log` file, specify:

```
compileFind -all -max_trials 100 -logfile ./tmp/run_100.log
```

- To run 5 compiles for each memory utilization value (100,96,92,88,84,80,76), specify:

```
compileFind -mem -min_util 75 -dec_util 4 -max_trials_each 5
```

- To run 30 (default) compiles in total for each memory utilization value (100,96,92,88,84,80,76), specify:

```
compileFind -mem -min_util 75 -dec_util 4
```

- To try up to 30 times (default) to find the best compile, specify:

```
compileFind -best
```

UXE Command Reference Manual

Compile-time Commands in ICE Mode

- To try up to 100 times to find the best compile that takes no more than 300 steps, specify:
`compileFind -best -max_trials 100 -max_steps 300`
- To try up to 40 times to find the best compile that takes no more than 300 steps, and no less than 280 steps, specify:
`compileFind -best -max_trials 40 -max_steps 300 -min_steps 280`
- To try up to 30 times to find the best compile that takes exactly 280 steps, specify:
`compileFind -best -max_steps 280 -min_steps 280`
- To run 5 trials for each of 4 pairs of optimizationEffort values {LOW HIGH} and placementEffort values {LOW HIGH}, in total 20 trials, specify:
`compileFind -opt -max_trials_each 5`
- To run 3 trials for each of 6 pairs of optimizationEffort values {LOW MED ULTRA} and placementEffort values {LOW HIGH}, in total 18 trials, specify:
`compileFind -opt -oE {l m u} -max_trials_each 3`
- To run 2 trials for each combination of optimizationEffort values {LOW CLASSIC HIGH} and placementEffort values {LOW HIGH}, specify:
`compileFind -opt -oE {lo HI CLA} -max_trials_each 2`
- To run 5 trials (default) for each optimizationEffort values {CLASSIC HIGH} and placementEffort value HIGH using LSF, specify:
`compileFind -opt -oE {HI CLA} -pE HI -lsf`
- To try up to 100 times to find the first compile that passes and takes no more than 300 steps, specify:
`compileFind -first -max_trials 100 -max_steps 300`
- To try up to 30 times to find the first compile that passes and takes no less than 280 steps, specify:
`compileFind -first -min_steps 280`
- To try up to 50 times on different machines in parallel using LSF to find the best compile that passes and takes no less than 350 steps, specify:
`compileFind -lsf -max_trials 50 -max_steps 350`
- To try up to 40 times on different machines in parallel using LSF and terminate the process as soon as the first passing compile with maximum 150 steps and minimum 50 steps is found, specify:
`compileFind -lsf -first -max_trials 40 -max_steps 150 -min_steps 50`
- To run 6 trials on different machines in parallel using LSF for each of 4 pairs of default optimizationEffort values {LOW HIGH} and placementEffort values {LOW HIGH}, in total 24 trials, specify:
`compileFind -lsf -oE {lo HI CLA} -pE HI -max_trials 6 -max_steps 300 -min_steps 280`

UXE Command Reference Manual

Compile-time Commands in ICE Mode

```
compileFind -opt -lsf -max_trials_each 6
```

design

Specifies the name of the design library and the top-level cell to open.

Before using most XEL commands, execute `design` to specify the name of the design to which you are applying those commands. Debugger commands do not require the `design` command.

The syntax for this command is:

```
design <lib> <cell> |  
design -close |  
design -rm
```

<lib>

Name of the library, which is the database containing your design.

<cell>

Name of the top-level design cell to open.

-close

Closes currently open databases. It does not delete the emulation design databases from the emulation design directory.

-rm

Closes currently open databases, then deletes the emulation design databases, including all files and subdirectories, from the emulation design directory.



Caution

This parameter deletes the entire design database. If you use the design -rm command, you need to import and compile your entire design again.

Examples

```
design my_library my_cell  
compile
```

Designates the `my_cell` cell in the `my_library` library (database) as the current design library and cell, then compiles this design for emulation.

```
design my_library my_cell
```

Designates the `my_cell` cell in the `my_library` library (database) as the current design library and cell.

```
design my_lib my_cell  
design -rm
```

Sets the current design path to `my_cell` in the `my_lib` library, closes the design database if it is currently open, then removes the design database from the `my_cell` emulation design directory.

disconnectDriverPin

Specifies the driver pin to be disconnected from a net that has multiple drivers. When used without any argument, the `disconnectDriverPin` command returns the list of driver pins that are currently disconnected.

The syntax for this command is:

```
disconnectDriverPin -add <hier_driver_pin> |  
disconnectDriverPin -rm <hier_driver_pin> |  
disconnectDriverPin -clear
```

-add <hier_driver_pin>

Adds the specified driver pin to the list of driver pins to be disconnected.

-rm <hier_driver_pin>

Removes the specified driver pin from the list of driver pins to be disconnected.

-clear

Clears the list of driver pins to be disconnected.

Example

```
disconnectDriverPin -add Top.dsp.decoder.pin25
```

Disconnects the driver pin named `Top.dsp.decoder.pin25`.

getCriticalPathDesignNetNames

Reads the critical path reports previously generated during compilation, and translates the encrypted net names in these reports to the design net names. The output of the `getCriticalPathDesignNetNames` command is placed in the `./tmp/criticalPathReport.msg` file.

At times, some net names in the `criticalPathReport.msg` file are of the form `trNet_<number>`. These names refer to the nets that were added to the design by the compiler. In some cases, you can get meaningful names instead of `trNet` names by setting the `SMDB_SAVE_MC` environment variable to 1 before precompile. However, setting the `SDMB_SAVE_MC` environment variable can greatly increase the database size and compilation time. Set this variable only if it is needed.

Run the `getCriticalPathDesignNetNames` command after the `compile` command. Ensure that the `compile` process has finished before this command is executed.

Two syntaxes of the command are supported:

```
getCriticalPathDesignNetNames
```

This command reads the critical path reports in the `tmp/et3compile.msg` file, if it exists. Otherwise, it reads the reports in the `tmp/et3pass1.msg` and `tmp/et3pass2.*.msg` files.

```
getCriticalPathDesignNetNames <file>
```

Here, `<file>` is the name of a file that itself contains a list of filenames. The files in this list contain the critical path reports to be read by the `getCriticalPathDesignNetNames` command.

For an example and an explanation of the critical path report, refer to the *Finding the Reason for a Poor Step Count* section of the *Improving Emulation Speed* chapter in *UXE User Guide*.

icePrepare

This command processes the target-related user data and generates files used by the subsequent compile command. This command is applicable for the ICE Mode only (used only when emulating with a Target System).

The `icePrepare` command must be executed after the precompile command. Between these two commands, run the in-circuit configuration commands: `terminalAssign`, `terminalTiming`, `delayBox`, and `cableConnection`.

The syntax for this command is:

```
icePrepare
```

precompile

This command performs design rule checks, flattening, loop-breaking, and optimizing on the design. Before running this command,

- Execute the `design` command to specify the design library and cell name
- Enter all required user data

Instead of running the `precompile` command explicitly, it is recommended that you do the following:

- Execute the `design` command to specify the design library and cell name
- Specify all user data, including that for the target interface (if any)
- Use the `precompileOption` command to specify options for precompile
- Run the XEL `compile` command.

The `compile` command now automatically runs `precompile` and `icePrepare` (if they are needed and have not already been run) and then runs the final steps of compilation.

Note: The old style of explicitly running the `precompile` command with its options is no longer supported.

The syntax for this command is:

```
precompile
```

Additional Information

The files `xe.msg`, `tmp/drc.report`, and `tmp/<LIB>.TOP.Multiclock` can provide additional information if `precompile` fails due to any user errors.

Examples

To debug netlist optimizations using the `precompile -debug` command:

1. Run `precompile -debug`.
2. Check the file `tmp/clean.rpt` and identify questionable nets. Note that this file might be huge. Copy-paste these net names into the file named `clean.request` in the design directory.

UXE Command Reference Manual

Compile-time Commands in ICE Mode

3. Run `precompile -debug` again. This creates a file called `tmp/clean.dump`. This file should normally be quite small and easy to read.

readCPFfile

Reads the input CPF script files that consist of the CPF commands. For command syntax and description of the supported CPF commands, refer to [Chapter 12, “Common Power Format Commands.”](#)

Executing the `readCPFfile` command without any option returns its syntax.

The syntax for this command is:

```
readCPFfile [-checkSyntaxOnly | -checkOnly | -noLinter | exitOnError] <cpfFileName>
readCPFfile [-clear]
readCPFfile [-help]
```

The `readCPFfile -checkSyntaxOnly <cpfFileName>` and `readCPFfile [-clear]` commands can be executed any time during a compilation session.

The `readCPFfile -checkOnly <cpfFileName>` command should be executed after the design is imported; otherwise, the instance, pin, or net names in the CPF file cannot be validated.

The `readCPFfile <cpfFileName>` command should be executed after the design is imported and before the `precompile` command. Otherwise, the compiler will not add the instrumentation logic based on the CPF file content.

At any point during compilation, after execution of the `readCPFfile <cpfFileName>` command succeeds (even if this command is executed in a previous compilation session), a CPF command can be issued without any arguments to see the information that was accepted.

<cpfFileName>

Specifies the name of the CPF file that needs to be read. The `readCPFfile <cpfFileName>` command checks for both syntax and semantic errors. The command stops processing if errors are found. If no errors are found, the CPF information is translated into Palladium XP data and saved in that format.

If a design element such as instance, pin or net existing in the CPF file cannot be found in the Palladium database, the compiler issues appropriate warnings. Wrong names in an expression terminate the execution and error messages are displayed. For example, if a pin name in the `shutoff_condition` is not in the Palladium database, the `readCPFfile` command stops processing and an error message is displayed.

-checkSyntaxOnly

Checks only the syntax errors in the specified <cpffileName> file.

Note: When this command option is used, the CPF information is not saved into Palladium XP user data format.

-checkOnly

Checks all syntax and semantic errors in the specified <cpffileName> file.

Note: When this command option is used, the CPF information is not saved into Palladium XP user data format.

-noLinter

Reads and saves the power intents for core compilation and run-time debug. When you specify this option, the CPF linter of the Conformal Low Power integrator is not invoked for checking the command syntax.

exitOnError

Prints an error message if a design element is not found in the palladium database.

For example, you specify the following command in the CPF file giving the instance names as path_to_inst1, path_to_inst2, and path_to_inst3.

```
create_power_domain -name pd1 \
    -instances {path_to_inst1 path_to_inst2 path_to_inst3}
```

However, the instance path_to_inst2 does not exist in the palladium database. In this case, the readCPFFfile -exitOnError command will print an error message saying that path_to_inst2 is not found in the Palladium database.

-clear

Cleans up the entire saved CPF information.

-help

Prints the command syntax.

read_power_intent

Reads the input 1801 files that consist of the IEEE 1801 commands. For supported 1801 commands and command options, refer to *Using IEEE 1801 to Specify Low-Power Intent on Palladium XP in UXE User Guide*.

Note: If a design name with Verilog escape name is used in the IEEE 1801 commands, the dot (.) can be used as a delimiter although 1801 files only accept slash (/) as a delimiter. For example, the name string \bank[0].out[7:0] means \bank[0] is an instance name and out[7:0] is a pin name. The `read_power_intent` command will accept this kind of names with a warning message as follows:

Accept escape name '%s' by allowing '..' to be used as delimiter in segment '%s'

The syntax for this command is:

```
read_power_intent <1801Filename>
    -force
    -version
    -defaultAppliesToOutputs [1.0 | 2.0 | 2.1 | UPF | 1801_2009 | ...
    1801_2013]
```

<1801Filename>

Specifies the name of the 1801 file that needs to be read. If the design instances are encrypted, the `read_power_intent` command locates the original signals from the specified 1801 files for the corresponding encrypted design instances.

-force

Ignores the names of the instances, pins, or nets that are given in the specified 1801 file, and not found in the Palladium design database. 1801 files power intent information will be saved to the Palladium internal files.

Without the `-force` option of the `read_power_intent` command, the power intent will not be saved if any errors are found in the 1801 file.

-version

Returns the 1801 linter version.

UXE Command Reference Manual

Compile-time Commands in ICE Mode

-defaultAppliesToOutputs [1.0 | 2.0 | 2.1 | UPF | 1801_2009 | 1801_2013]

Sets the default value for the `-applies_to` option of the `set_isolation` command.

UPF, 1801_2009, 1801_2013 are same as 1.0, 2.0, 2.1, respectively.

The default IEEE 1801 version is based on the `upf_version` command specified in the 1801 files. If the `upf_version` command is not in the 1801 files, the 2.1 (IEEE 1801-2013) version will be used as default.

clean_power_intent

Cleans up the entire saved power information.

The syntax for this command is:

```
clean_power_intent
```

referenceMap

Enables routing of OOMRs through protected portions of a hierarchy. If the OOMRs include protected components, use the `referenceMap` command to ensure that the protected components are included when the design is being traversed to identify the OOMRs.

Use the `referenceMap` command before `precompile` and after `designImport`.

The syntax for this command is:

```
referenceMap
```

shrinkDB

Removes the database files that are not needed for design download or debug after a successful compile, and retains only the files that are needed at the run time.

You can give this command after executing the `compile` command to reduce the disk space required for the design database.

After executing the `shrinkDB` command, if you need to compile the design again, it is necessary to re-run the `precompile` command and then execute the `compile` command for compilation.

The syntax for this command is:

```
shrinkDB
```

Common Power Format Commands

This chapter provides information about only those general Common Power Format (CPF) commands and options that are supported in UXE. These commands are only a subset of the full Cadence CPF command set. These Tcl-based commands are used along with the XEL commands to support low power design techniques.

For information on the CPF 1.1 standard for the supported low-power verification techniques, refer to the *Common Power Format User Guide, Version 1.1* and *Common Power Format Language Reference, Version 1.1*, available on the *Cadence Online Support* website -> *Product Pages* menu -> *Product Manuals* sub-menu -> *Digital IC Design* section -> *Encounter® RTL Compiler 11.1* -> *RC 11.1* -> *Product Documentation for RC 11.1* web page.

For information on the CPF 2.0 standard, refer to *Si2 Common Power Format Specification™, Version 2.0* document.

CPF wildcards for instance, pin, and net names are supported. The asterisk (*) matches any characters, and question mark (?) matches a single character. Wild card characters should not be used as the hierarchical delimiter, but can be used in hierarchical names. Wildcards in expressions are not supported. Expressions for definitions of power conditions can only be composed of terminal names.

The following CPF commands are discussed in this chapter:

- “[assert_illegal_domain_configurations](#)” on page 721
- “[create_isolation_rule](#)” on page 722
- “[create_mode](#)” on page 726
- “[create_mode_transition](#)” on page 727
- “[create_nominal_condition](#)” on page 728
- “[create_power_domain](#)” on page 730
- “[create_power_mode](#)” on page 737

UXE Command Reference Manual

Common Power Format Commands

- “[create_state_retention_rule](#)” on page 740
- “[end_design](#)” on page 745
- “[end_macro_model](#)” on page 746
- “[find_design_objects](#)” on page 747
- “[get_parameter](#)” on page 750
- “[include](#)” on page 751
- “[set_cpf_version](#)” on page 752
- “[set_design](#)” on page 753
- “[set_hierarchy_separator](#)” on page 756
- “[set_instance](#)” on page 757
- “[set_macro_model](#)” on page 760

assert_illegal_domain_configurations

Asserts that the specified configuration of domain conditions and power mode control group conditions is illegal. The assertion is checked against only those conditions that are explicitly specified.

```
assert_illegal_domain_configurations -name <string>  
{-domain_conditions <domain_condition_list>}
```

-name <string>

Specifies the name of the mode.

Note:

- ❑ The specified mode should not be defined using the create_power_mode command.
- ❑ The *<string>* parameter should not contain wildcards or hierarchy delimiter character.

-domain_conditions <domain_condition_list>

Specifies the nominal condition of each power domain to be considered as an illegal configuration. Use the following format to specify a domain condition:

```
<domain_name>@<nominal_condition_name>
```

Note:

- ❑ You can associate each power domain with only one nominal condition per command.
- ❑ The power domains must have been previously defined using the create_power_domain command.
- ❑ The condition must have been previously defined using the create_nominal_condition command.

create_isolation_rule

Defines a rule for adding isolation cells.

This command must be executed after the [create_power_domain](#) command.

A power domain is usually connected to other power domains. When a domain is powered down, the isolation logic must be added between the power domains to prevent the propagation of unknown states from a power domain that is powered down to a power domain that remains on.

The `create_isolation_rule` command enables you to specify the net segments that must be isolated, their isolation values, and the control expression that determines when the isolation must happen. This can be achieved using the following guidelines:

- Specify all pins to be isolated with the `-pins` option.
- Select only output pins in the power domains listed with the `-from` option.
- Select only input pins in the power domains listed with the `-to` option.
- Combine options to filter the set of pins.

In UXE, pin isolation is implicitly modeled by imitating the effect of inserting the appropriate isolation cell. This virtual isolation cell is inserted outside the power domain as a driver that drives the wire connected to the port. The isolation logic drives a logic value to the power-on domain connected to the pin. Note that the hierarchical equivalent nets and pins are treated as the same net (or physically the same thing). No isolation cell can be inserted between hierarchical equivalent nets.

The syntax for this command is:

```
create_isolation_rule -name <string>
    [-isolation_condition <expression> | -no_condition]
    [-pins <pin_list> | -from <power_domain_list> | -to <power_domain_list>{...}]
    [-exclude <pin_list>]
    [-isolation_output {high | low | hold | clamp_high | clamp_low}]
    [-secondary_domain <power_domain>]
```

-name <string>

Specifies the name of the isolation rule. The `<string>` parameter should not contain wildcards or hierarchy delimiters.

-isolation_condition <expression> | -no_condition

The `-isolation_condition` option specifies the condition when the selected nets should be isolated. Nets are isolated when the condition `<expression>` evaluates to true. The condition can be a function of pins and ports.

The `-no_condition` option specifies that the isolation logic is automatically enabled when the power domains containing the drivers of the selected net segments are shutoff.

If neither `-isolation_condition` nor `-no_condition` option is specified, the isolation rule is considered incomplete. In this case, only net segments connected to the primary input ports of the current design or macro model can be selected.

To complete an incomplete isolation rule, the `-default_isolation_condition` option specified with the `create_power_domain` command for the driving power domain(s) of the selected net segments is used as the isolation condition.

If the `-default_isolation_condition` option for the driving power domain of the selected net segments was also not specified, the incomplete isolation rule will be ignored with a warning message.

{-pins <pin_list> | -from <power_domain_list> | -to <power_domain_list>}...

The `-pins` option selects nets connected to, driven by, or driving the pins specified in the `<pin_list>` argument. You can specify ports and instance pins. This option must be used along with the `-from` and/or `-to` options. An error message will be generated if the `-pins` option is executed without the `-from` and/or `-to` option. The pins should be input pins or output pins.

Note: UXE does not support isolation of nets connected to bidirectional pins. A warning message is returned if a bidirectional pin appears in the `-pins` option, and the pin is ignored.

The `-from` option specifies to apply the rule to the nets driven by logic in the specified (from) domain and with at least one driving logic in another power domain. The power domain must have been previously defined using the `create_power_domain` command.

The `-to` option specifies to apply the rule to the nets that are connected to logic belonging to the specified (to) domain and that are driven by a signal from another power domain. The power domain must have been previously defined using the `create_power_domain` command.

UXE supports the following six combinations of defining isolation rules using the `-from`, `-to`, and `-pins` options:

- Specifying only the `-from` option selects all nets driven by logic in the specified *from* domains and nets that are driving logic in other power domains.
- Specifying only the `-to` option selects all nets driving logic in the specified *to* domains and nets that are driven by logic from another power domain.
- If a combination of `-to` and `-from` options is used, the isolation rule applies to the nets that only drive logic in the specified *to* domains and that are driven by logic in the specified *from* domains.
- If a combination of `-from` and `-pins` options is used, the isolation rule applies to the nets that drive or connect to the specified pins, and also meet the requirements of the `-from` option.
- If a combination of `-to` and `-pins` options is used, the isolation rule applies to the nets that are driven by or connected to the specified pins, and also meet the requirements of the `-to` option.
- If a combination of `-from`, `-to`, and `-pins` options is used, the isolation rule applies to the nets that
 - Are driven by or connected to the specified pins
 - Only drive logic in the specified *to* domains
 - Are driven by logic in the specified *from* domains

-exclude <pin_list>

Specifies to exclude the selected net segments from isolation that are connected to, driven by, or driving the specified pins. The `<pin_list>` argument can include ports and instance pins.

Following type of nets are excluded:

- a floating net
- an undriven net
- a net driven by a tie high or tie low signal
- a net that has its leaf level driver and loads in the same power domain

Note: The isolation rule will be ignored if all nets are excluded.

-isolation_output {high | low | hold | clamp_high | clamp_low}

Controls the output value of the isolation gates when the isolation condition is true.

In UXE, the isolation output can have either of the following values:

- `low` - Forces the output to 0. This is the default value.
- `high` - Forces the output to 1.
- `hold` - Forces the output to the same logic value it had right before the isolation condition is activated.
- `clamp_high` - Forces the output to be clamped to a high value if the driving domain is ground switched.
- `clamp_low` - Forces the output to be clamped to a low value if the driving domain is a power switched.

When the isolation control expression is `false`, no isolation occurs and the port signal value is propagated. When the isolation control expression is `true`, isolation is enabled, and the signal value propagated from the port is either 0 (for `low`) or 1 (for `high`). For `hold` isolation, the value of the isolated output is latched when the isolation control is enabled.

-secondary_domain <power_domain>

Specifies the domain from which the isolation logic gets the power supply. The isolation logic is considered powered on as long as the secondary domain is powered on.

Related Commands

[create_power_domain](#)

create_mode

Defines a mode of the design. Multiple modes can be active simultaneously at a point in time. In addition, a mode definition does not require explicit specification of the states of all power domains. Therefore, this is a more general specification than power modes.

The syntax for this command is:

```
create_mode -name <string>
-condition <expression>
[-illegal]
```

-name <string>

Specifies the name of the mode. The *<string>* parameter should not contain wildcards and hierarchy delimiters.

Note: The name of a mode must be unique among other modes and power modes within the same design scope.

-condition <expression>

Specifies the condition when the design is in the specified mode. Use a Boolean or arithmetic expression of the design pins and ports to describe a condition.

Note: Only logical operators ! (unary logical negation), && (logical AND), and | | (logical OR) are supported with operands as a port or pin. However, bitwise XOR operator (^) is not supported.

-illegal

Identifies the mode as illegal. By default, a mode is legal.

Related Commands

[create_power_mode](#)

create_mode_transition

Describes how the transition between two modes is controlled, and the time it takes for each power domain to complete the transition. The transition can be between power modes that were defined using the [create_power_mode](#) command.

The syntax for this command is:

```
create_mode_transition -name <string>
                      {-from <power_mode> -to <power_mode>}
```



On Palladium XP, the create_mode_transition command supports only the three options given above. Without the -start_condition option, the CPF linter will give an error message that must be ignored.

-name <string>

Specifies the name of the power-mode transition. The *<string>* parameter should not contain wildcards and hierarchy delimiters.

-from <power_mode>

Specifies the power mode from which to transit.

-to <power_mode>

Specifies the power mode to which to transit.

create_nominal_condition

Creates a nominal operating condition with the specified voltage. Power domains with a nominal voltage of 0 are switched off.

This command must be executed after the [set_design](#) command.

The syntax for this command is:

```
create_nominal_condition -name <string>
                          -voltage {<voltage> | <voltage_list>}
                          [-ground_voltage {<voltage> | <voltage_list>}]
                          [-state {on | off | standby}]
```

-name <string>

Specifies the name of the nominal operating condition. The *<string>* parameter should not contain wildcards and hierarchy delimiters.

-voltage {<voltage> | <voltage_list>}

Specifies either a single voltage or a voltage list for the power net in the domain that uses this nominal condition. The voltage must be specified in volt (V).

-ground_voltage {<voltage> | <voltage_list>}

Specifies either a single voltage or a voltage list for the ground net in the domain that uses this condition. The voltage must be specified in volt (V).

-state {on | off | standby}

Specifies the state of a power domain when it uses the specified nominal condition. By default, the state is considered on if the voltage specified for the **-voltage** option is non-zero.

If the specified voltage is 0, the **-state** option is ignored. If the **-state** option is not **off**, but the voltage is 0, the **-state** option is ignored with a warning message.

Example

```
create_nominal_condition -name high -voltage 1.2
```

UXE Command Reference Manual

Common Power Format Commands

```
create_nominal_condition -name low -voltage 1.0
create_nominal_condition -name off -voltage 0
create_power_mode -name M1 -domain_conditions {PD1@high PD2@high PD3@low}
create_power_mode -name M2 -domain_conditions {PD1@high PD2@low PD3@off}
create_power_mode -name M3 -domain_conditions {PD1@low PD2@off PD3@off}
```

Related Commands

[create_power_mode](#)

create_power_domain

Creates a power domain and specifies the instances, boundary ports, and pins that belong to the specified power domain.

A power domain is a collection of instances, pins, and ports that share the same power distribution network and that either can be simultaneously switched on or off, or can not be switched.

Power domains are scope-specific.

The `create_power_domain` command must be used after the `set_design` command. Also, it can be used multiple times. If a conflict occurs, the first one is selected and the later ones with the same power domain name within the same scope are ignored.

The syntax for this command is:

```
create_power_domain [-name <power_domain>
                     [-instances <instance_list>]
                     [-boundary_ports <pin_list>]
                     [-default]
                     [-shutoff_condition <expression>]
                     [-external_controlled_shutoff]
                     [-default_isolation_condition <expression>]
                     [-default_restore_edge <expression> | -default_save_edge <expression> |
                      -default_restore_edge <expression> -default_save_edge <expression>]
                     [-power_down_states {high | low | random | inverted}]
                     [-power_up_states {high | low | random | inverted}]
                     [-active_state_conditions <active_state_condition_list>]
                     [-base_domains <domain_list>]]
```

-name <power_domain>

Specifies the name of the power domain to which the power information applies.

Note: The specified string should not contain wildcards and hierarchy delimiter character.

-instances <instance_list>

Specifies the names of all instances that belong to the specified power domain.

Instances defined in a power domain can be either module instances or Verilog regs.

A `create_power_domain` command that defines instances of a macro cell as a power domain must be specified between the `set_design` and `end_design` commands.

A `create_power_domain` command that defines a Verilog reg as a power domain must be specified between the `set_macro_model` and `end_macro_model` commands. An error will be generated if it is specified between the `set_design` and `end_design` commands.

Note: Only full register names can be specified as a power domain instance. Specifying a reg bit-select or part-select as a power domain instance is an error.

When a wildcard is used to specify instances, reg names are not picked up if they are specified between the `set_design` and `end_design` commands. However, both reg names and other instances names are picked up if they are specified between the `set_macro_model` and `end_macro_model` commands.

To specify a sub-instance in default power domain while its super instance is in another power domain, the `-instances <instance_list>` option must be specified together with the `-default` option.

For example, assume there is an instance `A` and sub-instance `A.b`. Instance `A` is in power domain `PD2` and `A.b` is in power domain `PD1`. Also, `PD1` is the default power domain. Then, specify the following commands between the `set_design` and `end_design` commands:

```
create_power_domain -name PD2 -instances {A}  
create_power_domain -name PD1 -default -instances {A.b}
```

An instance and its sub-instance can be defined in separate power domains. In this case, the hierarchically equivalent output pins will be in sub-instance domain, while hierarchically equivalent input pins will be in their drivers' domain.

-boundary_ports <pin_list>

Specifies the list of inputs and outputs that are considered part of this domain.

For inputs and outputs of the top-level design, specify the ports. For inputs and outputs of IP instances, specify a list of the instance pins that are part of the domain.

Note: In case of a macro model, a boundary port can be associated with multiple power domains.

If the `-instances` option is specified together with the `-boundary_ports` option, it indicates that for any connection between a specified port and any instance inside the power domain, no special interface logic for power management is required.

-default

Identifies the specified domain as the default power domain. All instances of the design that were not associated with a specific power domain belong to the default power domain.

-shutoff_condition <expression>

Specifies the condition when a power domain is shutoff. The condition is a Boolean function of pins and ports. If this option and the `-external_controlled_shutoff` option are omitted, the power domain is considered to be always on.

-external_controlled_shutoff

Specifies that the power domain being defined is shutoff by an off-chip power switch controlled by a signal external to the current design.

-default_isolation_condition <expression>

Specifies the default isolation condition for those isolation rules that have no isolation condition and apply to net segments with leaf driver in the current power domain.

-default_restore_edge <expression>

Specifies the default edge-sensitive condition when the states of the sequential elements need to be restored. This option applies to all state retention rules that are created for sequential elements in the current power domain. If no state retention rules were created for the current power domain or a list of sequential elements in it, this option is ignored.

If the condition is edge-sensitive, the states are restored when the expression changes from `false` to `true`.

The `<expression>` argument can be a function of pins and ports.

-default_save_edge <expression>

Specifies the default edge-sensitive condition when the states of the sequential elements need to be saved. This applies to all state retention rules which are created for sequential instances in the current power domain. If no state retention rules were created for the current power domain, this option is ignored.

If the condition is edge_sensitive, the states are saved when the expression changed from false to true.

The *<expression>* argument can be a function of pins and ports.

-power_down_states {high | low | random | inverted}

Specifies the output values for the sequential logic in the corresponding power domain after the domain is in power-off state. This command is supported only if sequential elements are modeled as flip-flops or latches during compilation. Sequential elements modeled as memory instances are supported only in STB mode through run-time control.

The `-power_down_states` command option can have either of the following values:

Value	Indicates that after the power domain is powered off
high	All output values of the sequential elements in the domain are set to 1.
low	All output values of the sequential elements in the domain are set to 0.
random	All output values of the sequential elements in the domain are randomly set to 0 or 1.
inverted	All output values of the sequential elements in the domain are set to the inversion of the values before the power domain is powered off.

If the `-power_down_states` option is not specified, by default the `low` value will be used.

-power_up_states {high | low | random | inverted}

Specifies the output values of the non-state-retention sequential logic in the corresponding power domain after the domain is in power-up state. This command is supported only if sequential elements are modeled as flip-flops or latches during compilation. Sequential elements modeled as memory instances are supported only in STB mode through run-time control.

The `-power_up_states` command option can have either of the following values:

Value	Indicates that after the power domain is powered up
high	All output values of the non-state-retention sequential elements in the domain are set to 1.
low	All output values of the non-state-retention sequential elements in the domain are set to 0.
random	All output values of the non-state-retention sequential elements in the domain are randomly set to 0 or 1.
inverted	All output values of the non-state-retention sequential elements in the domain are randomly set to the inversion of the values before the power domain is powered off.

If the `-power_up_states` option is not specified, by default the `random` value will be used.

-active_state_conditions <active_state_condition_list>

Specifies the Boolean condition for each nominal condition or state at which the power domain is considered on, that is, active.

When a condition changes to true, the power domain starts the transition to the state specified by the nominal condition.

The `<active_state_condition_list>` argument lists all nominal conditions that the specified power domain might be in, except the power-off nominal condition. Use the following format to specify an active state condition:

`<nominal_condition_name>@<expression>`

The `<nominal_condition_name>` must have been specified using a `create_nominal_condition` command.

The `<expression>` is a regular CPF expression that must be enclosed within braces.



None of the conditions can overlap and the domain must be within one of the active states or be shutoff.

The `-active_state_conditions` option is required in DVFS, which means the power domain can operate on different supply voltages and its operation is controlled by a set of signals. If this option is used in PSO, it means that a power domain can be in standby or on nominal condition.

-base_domains <domain_list>

Specifies a list of base domains that supply external power to the primary domain through some power switch network.

When all base power domains are switched off, the power domain will be switched off irrespective of the shutoff conditions.

If one of the base domains is on or in standby state and the shutoff condition is false, the power domain is considered on or in standby state.

Examples

The following command associates a list of instances with power domain PD2.

```
create_power_domain -name PD2 -instances {A*C I_ARM1 PAD1}
```

In the example below, assume the design uses the following hierarchy:

```
Top
    INST1
        INST2
```

The following two sets of CPF commands are equivalent when applied to this design:

```
create_power_domain -name PD1 -instances INST1
create_power_domain -name PD2 -instances INST1.INST2
```

```
create_power_domain -name PD2 -instances INST1.INST2
create_power_domain -name PD1 -instances INST1
```

This illustrates that the order in which you specify the target domains is irrelevant. The result is that instance INST1 belongs to power domain PD1 and instance INST2 belongs to power domain PD2.

Related Commands

[create_isolation_rule](#)

[create_nominal_condition](#)

UXE Command Reference Manual

Common Power Format Commands

create state retention rule

create_power_mode

Defines the power mode. A power mode is a static state of a design in which each power domain operates on a specific nominal condition.

In the nominal conditions, some power conditions can be aliased to each other, for example:

```
create_power_mode -name m1 -domain_conditions {PD1@high PD2@low}
create_power_mode -name m2 -domain_conditions {PD1@high PD2@lhigh}
```

This creates equivalent power modes m1 and m2. When UXE shows power mode waveforms, only the first power mode name, m1, is used.

The `create_power_mode` command must be executed after the [create_nominal_condition](#) and [create_power_domain](#) commands.

The syntax for this command is:

```
create_power_mode -name <string>
                  {-domain_conditions <domain_condition_list>}
                  [-default]
```

-name <string>

Specifies the name of the mode.

Note: The specified *<string>* should not contain wildcards or hierarchy delimiter character.

-domain_conditions <domain_condition_list>

Specifies the nominal condition of each power domain to be considered in the specified power mode. A domain condition should be specified in the following format:

```
<domain_name>@<nominal_condition_name>
```

Before creating a power mode, the power domains and nominal condition must have been defined using the `create_power_domain` command and the `create_nominal_condition` command, respectively.

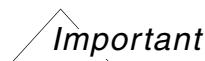
Note: Each power domain can be associated with only one nominal condition for a given power mode.

A domain is considered in a power off state in the specified mode, if the following conditions are satisfied:

- It is associated with a nominal condition whose state is off.
- It is not specified in the list of domain conditions.

-default

Labels the specified mode as the default mode, which corresponds to the initial state of the design.



If any power mode has been created, it is mandatory to define one (and only one) power mode as the default mode.

Related Commands

[create_nominal_condition](#)

[create_power_domain](#)

Examples

The following example applies to a design with three power domains, designated as PD1 and PD2. The design can operate in two power modes, M1 and M2.

Following table shows the voltages for each power domain in each of the power modes:

Power Domain			
Power Mode	PD1	PD2	PD3
M1	1.2V	1.2V	1.0V
M2	1.2V	1.0V	0.0V

```
create_nominal_condition -name high -voltage 1.2
create_nominal_condition -name low -voltage 1.0
create_power_mode -name M1 -domain_conditions {PD1@high PD2@high PD3@low}
create_power_mode -name M2 -domain_conditions {PD1@high PD2@low}
```

UXE Command Reference Manual

Common Power Format Commands

Note: Since power domain PD3 is switched off in power mode M2, it can be omitted from the domain conditions when defining power mode M2.

create_state_retention_rule

Describes the sequential logic save and restore behavior.

When a power domain is powered down, the states of certain sequential elements, such as flip-flops, latches, and memories in the power domain must be saved and retained for the entire shutoff period. When the power domain is powered up once again, the saved states must be written back into the registers. To ensure that a powered-down domain resumes back normal operation after the power up, special state retention cells can *replace* these sequential elements.

The `create_state_retention_rule` command defines the rule for replacing either all or selected registers in the specified power domain or instances with state retention registers.

In UXE, state retention is implicitly modeled by mimicking the effect of replacing appropriate state retention cell. The *replace* is done by adding additional instrumental logic in the sequential elements.

This command must be executed after the `create_power_domain` command.

The table below shows how to select the save or restore condition for a specific power domain, depending on how it is defined in the `create_power_domain` or `create_state_retention_rule` command. The abbreviations used in the table are as follows:

- The underscore (_) means whether the condition is present or not
- DS signifies the default save-edge condition
- DR signifies the default restore-edge condition
- S signifies the save-edge condition
- R signifies restore-edge condition

Table 12-1 Criteria for selecting save/restore condition for a power domain

default save-edge in <code>create_power_domain</code>	—	DS	—
default restore-edge in <code>create_power_domain</code>	—	DR	—
save-edge in <code>create_state_retention_rule</code>	S	—	DR

UXE Command Reference Manual

Common Power Format Commands

restore-edge in	R		R	
create_state_retention_rule				
final save-edge condition	S	DS	^R	^DR
final restore-edge condition	R	DR	R	DR

The syntax for this command is:

```
create_state_retention_rule -name <string>
    {-domain <power_domain> | -instances <instance_list>}
    [-exclude <instance_list>]
    [-restore_edge <expression> | -save_edge <expression> |
     -restore_edge <expression> -save_edge <expression>]
    [-restore_precondition <expression>]
    [-save_precondition <expression>]
    [-secondary_domain <domain>]
```

-name <string>

Specifies the name of the state retention rule.

-domain <power_domain>

Specifies to apply the rule to all sequential instances in the current domain. The sequential instances that are specified in the `-exclude <instance_list>` option are ignored.

The rule is ignored if the specified power domain is always on or if the domain has no sequential elements.

The power domain must have been previously defined using the `create_power_domain` command.

-instances <instance_list>

Specifies the instances that need to be replaced with the state retention cells.

An instance can be a leaf or hierarchical instance name in a gate-level netlist; or a register variable or hierarchical instance in RTL.

If the name of a hierarchical instance is specified, all registers or non-state-retention sequential elements in that instance and its children will be replaced.

Note: The specified instances can belong to multiple power domains. If they belong to different power domains, the same save and restore conditions will be applied to all of them.

-exclude <instance_list>

Specifies to exclude the specified list of instances from the list of selected instances that must be replaced with state retention elements.

An instance can be a leaf or hierarchical instance name in a gate-level netlist; or a register variable or hierarchical instance in RTL.

The basic steps to exclude instances are:

1. Collect and arrange the instances using either the `-domain` or `-instances` option.
2. Then exclude the instances from the selected instance list.

If the name of a hierarchical instance is specified, all registers or non-state-retention sequential elements in that instance and its children will be excluded.

-restore_edge <expression>

Specifies that the states are restored when the expression changes from `false` to `true`.

The `<expression>` argument can be a Boolean function of pins and ports.

If this option is specified with the `create_state_retention_rule` command and the `-default_restore_edge` option has also been specified with the `create_power_domain` command, the condition specified with the `create_state_retention_rule` command takes precedence.

If this option is omitted but the `-save_edge` option is specified, the registers restore the saved values when the power is turned on.

If both `-restore_edge` and `-save_edge` options are omitted, but the `-default_restore_edge` option is specified with the `create_power_domain` command for the corresponding power domain(s), the conditions specified with the `create_power_domain` command will be used.

-save_edge <expression>

Specifies that the states are saved when the expression *changes* from `false` to `true` and the power is on.

The *<expression>* argument can be a Boolean function of pins and ports.

If this option is specified with the `create_state_retention_rule` and the `-default_save_edge` option is also specified with the `create_power_domain` command, the condition specified with the `create_state_retention_rule` command takes precedence.

If this option is omitted but the `-restore_edge` option is specified, the states are saved when the expression changes from `true` to `false` and the power is on.

If both `-restore_edge` and `-save_edge` options are omitted, but the `-default_save_edge` option has been specified with the `create_power_domain` command for the corresponding power domain(s), the conditions specified with the `create_power_domain` command will be used.

`-restore_precondition <expression>`

Specifies an additional condition that must be met for the restore operation to be successful.

If this option is not specified, the restore operation is performed when the `restore_edge` condition evaluates to true.

The *<expression>* parameter can be a Boolean function of pins, ports and nets.

`-save_precondition <expression>`

Specifies an additional condition that must be met for the save operation to be successful.

If this option is not specified, the save operation is performed when the `save_edge` condition evaluates to true.

The *<expression>* parameter can be a Boolean function of pins, ports and nets.

`-secondary_domain <domain>`

Specifies the name of the power domain that provides the continuous power when the retention logic is in retention mode.

During implementation, the primary power and ground nets of this domain must be connected to the non-switchable power and ground pins of the state retention cells.

Related Commands

[create_power_domain](#)

Example

In the following example, module `IPBlock` is instantiated as instance `IPInst` in design `Top`. Instance `IPInst` is part of a switchable power domain `X`.

`IPblock.cpf` is the CPF file for module `IPBlock`:

```
set_design IPBlock
create_state_retention_rule -name sr1 -instances *
end_design
```

Consider the following 2 CPF files for the design `Top`:

Case 1: Top.cpf

```
set_design Top
create_power_domain -name X -instance IPInst -shutoff_condition switch_en \
-restore_edge restore_en
source IPBlock.cpf
```

In this case, all instances of `IPInst` will be converted to state-retention flops using the restore condition specified for domain `X` at the top-level design.

Case 2: Top.cpf

```
set_design Top
create_power_domain -name X -instance IPInst -shutoff_condition switch_en \
source IPBlock.cpf
```

Here, the state retention rule created for `IPBlock` is ignored because the `-restore_edge` option is neither defined with the `create_state_retention_rule` command nor with the `create_power_domain` command for domain `X`.

end_design

Specifies the end of the current scope. It is used along with the `set_design` command to group a number of CPF commands that apply to the current scope.

Note: If either `set_design` or `end_design` is used without the matching parameter, no error message is issued. Therefore, the pairings of these commands should be matched carefully such that the specified module or power design name must match the name specified in the immediately preceding `set_design` command.

The syntax for this command is:

`end_design [<power_design_name>]`

<power_design_name>

Specifies the name of the power design used with the `set_design` command.

Related Commands

[set_design](#)

end_macro_model

Specifies the end of the current macro model. It is used along with the `set_macro_model` command to group a number of CPF commands that apply to the described macro cell.

Note: The pairings of the `set_macro_cell` and `end_macro_cell` commands should be matched carefully. The macro cell or model name specified with the `end_macro_cell` command must match the name specified in the immediately preceding `set_macro_cell` command.

The syntax for this command is:

```
end_macro_model [<macro_cell_name> | <macro_model_name>]
```

<macro_cell_name>

Specifies the name of the macro cell used with the `set_macro_cell` command.

<macro_model_name>

Specifies the name of the macro model used with the `set_macro_cell` command.

Related Commands

[set_macro_model](#)

find_design_objects

Searches for and returns design objects that match the specified search criteria in the specified scope. This provides a general way to select design objects for use in CPF commands. All objects are returned with respect to the current scope.

The syntax for this command is:

```
find_design_objects <pattern>
    [-pattern_type {name | cell | module}]
    [-scope <hierarchical_instance_list>]
    [-object {inst | port | pin | net}]
    [-direction {in | out | inout}]
    [-leaf_only | -non_leaf_only]
    [-hierarchical] [-exact | -regexp]
    [-ignore_case]
```

<pattern>

Specifies the search string. By default, it is a Tcl global expression.

-pattern_type {name | cell | module}

Specifies the type of name to be matched by the pattern string. This option supports either of the following pattern types:

name	Returns objects whose names match the pattern. This pattern type is allowed for any object type.
------	--

Note: This is the default pattern type.

cell	Returns objects that are instances of library cells whose names match the pattern. This pattern type is only allowed if the -object option is specified with the inst value or if the -object option is omitted.
------	---

module	Returns objects that are instances of modules whose names match the pattern. This pattern type is only allowed if the -object option is specified with the inst value or if the -object option is omitted.
--------	---

-scope <hierarchical_instance_list>

Specifies the scope or scopes from which the search must start. You can only reference the current scope and its children.

You can specify a list of hierarchical instances, but you should not use wildcard characters or regular expressions.

When this parameter is not specified, the search begins in the current scope. An error is generated if the specified hierarchical instances cannot be found within the current scope.

-object {inst | port | pin | net}

Specifies the type of design objects to be returned.

inst Returns instances.

Note: This is the default object type.

port Returns ports of the modules in the current scope. When you specify this object type, the `-scope` and `-hierarchical` options are ignored.

pin Returns pins.

net Returns nets.

-direction {in | out | inout}

Returns only pins or ports with the specified direction. This option is applied only if the `-object` option is specified with `pin` or `port` argument. It is ignored when used with other object types.

When you omit this option and you specify the `-object` option with either `pin` or `port` argument, all matching pins or ports will be returned, irrespective of the direction.

-leaf_only | -non_leaf_only

Specifies that only instances without children (`-leaf_only`), or with children (`-non_leaf_only`) should be returned.

This option is applied only if the `-object` option is specified with the `inst` argument or if the `-object` option is omitted. When used with an object type other than `inst`, either of these options is ignored.

By default, all matched `inst` objects are returned.

-hierarchical

Specifies to perform a recursive search within the specified scopes and all children of those scopes.

By default, the search only applies to the specified scopes, not including its children.

-exact

Returns only objects whose names exactly match the pattern value. No wildcard expansion is performed.

-regexp

Indicates that the specified pattern is a regular expression.

-ignore_case

Specifies to perform a case-insensitive search for both the pattern and the string specified by the `-scope` option.

By default, the search pattern is case sensitive.

get_parameter

Returns the value of a predefined parameter in the current design.

An error message is generated if the parameter is not defined for the current design.

The syntax for this command is:

`get_parameter <parameter_name>`

<parameter_name>

Specifies a parameter name.

The parameter must have been defined using the `-parameters` option of the `set_design` command for the current design.

include

Includes a CPF file or a Tcl file within a CPF file.



Use of include and source commands in the same CPF file is not recommended. Use either of the two commands consistently; otherwise, the file references might be incorrect.

The syntax for this command is:

```
include <CPF_filename>
```

<CPF_filename>

Specifies the path of the CPF file to be included.

The path to the file can contain a period (.) to refer to the current directory of the CPF file being read and ".." to refer to the parent directory of the current directory of the CPF file being read.

Note: This command differs from the `source` command in Tcl syntax where a period (.) always refers to the directory of the top level CPF file being read and ".." refers to the parent directory of the top level CPF file being read.

set_cpf_version

Specifies the CPF version.

This is an optional command, which if given, should be the first CPF command in the CPF file.

The syntax for this command is:

`set_cpf_version [value]`

<value>

Specifies the `<value>` in the format `<ReleaseNumber>.<RevisionNumber>`.

The supported values are 1.1 and 2.0. The default is 1.1.

set_design

Represents the beginning of a scope and specifies the name of the module to which the power information applies.

The first appearance of the `set_design` command defines the top cell. Thus, do not use the `-ports` argument with the first `set_design` command because it is the main entry to the CPF file.

The `set_design` command must be used along with the `end_design` command to define the beginning and end of a scope. If the design includes both testbench and DUV, but CPF includes only DUV, then use the DUV module name as CPF `set_design` module, and use the DUV instance name as the scope for the CPF objects. For example, if the design top cell is `testbench`, DUV cell is `DUT`, and instance name is `dut`, the following CPF file will reference `en` to `dut.en`, `A` to `dut.A`, and `B` to `dut.B`:

```
set_design DUT
create_power_domain -name PD_1 -default
create_power_domain -name PD_2 -shutoff_condition en -instances {A,B}
end_design
```

The syntax for this command is:

```
set_design [-module <module_list>]
           [-ports <port_list>] [-input_ports <port_list>]
           [-output_ports <port_list>] [-inout_ports <port_list>]
           [-parameters <parameter_value_list>]
           [-testbench]
```

-module <module_list>

Specifies the name of the module to which the power design information applies.

The power design can only be applied to instances of a logic module whose name exactly matches one of the specified names.

When this option is not specified, the power design can be applied to instances of any logic module.

The module names should not contain wildcards.

-ports <port_list>

Specifies the list of virtual ports in the specified module.

Virtual ports do not exist in the RTL code, but could be needed for control signals in the low power logic, such as isolation logic and state-retention logic.

If the `-port` option is given and the `-honor_boundary_port_domain` option is omitted, a warning will be displayed and the `-port` option will be ignored.

-input_ports <port_list>

Specifies a list of virtual input ports in a module to which this power design applies.

-output_ports <port_list>

Specifies a list of virtual output ports in a module to which this power design applies.

-inout_ports <port_list>

Specifies a list of virtual inout ports in a module to which this power design applies.

-parameters <parameter_value_list>

Specifies a list of parameters with their default values. Use the following format for each parameter:

{*<parameter_name>* *<default_value>*}

The default value can be a number or a string.

-testbench

Specifies that the power design model applies to a testbench module.

Use this option to create a testbench-level CPF to drive power domains for designs under test, without the need to create power domains or modes in the testbench level CPF.

Note: When specified, the ports of the DUV (design under verification) module must be treated as the leaf level drivers or loads when considering the isolation or level shifter rules in the CPF for the DUV module.

Related Information

[end_design](#)

set_hierarchy_separator

Sets the hierarchy separator for the CPF commands. This separator only affects the CPF commands.

This command must be executed after the [set_design](#) command, but before any `create_` CPF command of the same scope.

The syntax for this command is:

```
set_hierarchy_separator [<character>]
```

<character>

Designates the hierarchy separator for CPF commands. The accepted characters are period (.), slash (/), and colon (:). The default separator is a period (.).

set_instance

Changes the scope to the specified instance or links a previously defined CPF power design or macro model to the specified instance. The scope is used for name resolution, and affects all design objects and expressions in the CPF design-related constraints. All CPF objects referred to in the library cell-related CPF commands are scope insensitive.

This command must be executed after the [set_design](#) command.

When the command is executed without an argument, it returns the current scope. If the current scope is the top design, the hierarchy separator is returned.

If the command is specified with an instance name only, it will change the design scope for the purpose of searching design objects referenced in the commands after it.

If the command is specified with an instance name, and without the `-design` or `-model` options, but with some other options, it must be followed by a [set_design](#) or [set_macro_model](#) command. Otherwise, the `set_design` or `set_macro_model` command will fail. Taken together, `set_instance` and `set_design` commands serve as scope settings.

The syntax for this command is:

```
set_instance [<instance>
              [-design <power_design_name> | -model <macro_cell>]
              [-port_mapping <port_mapping_list>]
              [-domain_mapping <domain_mapping_list>]
              [-parameter_mapping <parameter_mapping_list>]]
```

<instance>

Specifies a valid instance name in the current scope.

If the name is not followed by either the `-design` or `-model` option, the `set_instance` command changes the scope to the specified instance.

-design <power_design_name>

Specifies to link a previously defined power design to the specified instance. The scope is not changed when this option is used.

A bound power design is associated with either of the following:

- an instance through the `set_instance` command

- the top level module of a design

To bind a power design to the top level module of a design,

- you can specify the power design name at the command level when reading or elaborating the CPF
- a tool can select the only power design that is not bound to any instance

It is an error to bind multiple power designs to the same instance.

-model <macro_cell>

Specifies to use a previously loaded CPF description for the specified macro CPF module.

The CPF description must precede the current `set_instance` command and be contained between a `set_macro_model <macro_cell>` command and the next `end_macro_model` command. In this case, the scope does not change.

-port_mapping <port_mapping_list>

Defines the connection between a pin of the specified instance and a pin or port visible in the current scope.

The instance pin must be a pin corresponding to the virtual port defined in the `set_design` command or a real pin from the cell or module definition.

It is an error if the specified connection conflicts with any existing netlist connection.

Use the following format to specify a port mapping:

```
{<block_instance_pin> <current_scope_driving_pin>}  
{<macro_port> <parent_level_driver>}
```



Any rule created in the block-level CPF file that references a virtual port declared using the `-inout_ports`, `-input_ports`, `-output_ports`, or `-ports` option of the `set_design` command will be ignored if its mapping is not specified using the `set_instance -port_mapping <port_mapping_list>` command.

For example, considering the following two CPF files, `test1.cpf` and `test2.cpf`:

```
#test1.cpf
```

```
set_design Top
...
set_instance inst_A -port_mapping {P a.c.}
source test2.cpf
set_instance inst_B -port_mapping {P b.c}
source test2.cpf
...
end_design

#test2.cpf
set_design IP_Top -ports P
...
end_design
```

In this example, the virtual port `P` is mapped to `a.c` or `b.c` by the commands in the `test2.cpf` file.

-domain_mapping <domain_mapping_list>

Specifies the mapping of the domains in the current scope to the domains for the specified design or macro model.

Use the following format to specify a domain mapping:

```
{<domain_in_child_scope> <domain_in_parent_level_scope>}
```

-parameter_mapping <parameter_mapping_list>

Specifies the mapping of the parameters specified in the `set_design` command to the local values.

Note: It is an error if the parameter is not specified in the `set_design` command.

Use the following format to specify a parameter mapping:

```
{<parameter_name> <local_value>}
```

Related Commands

[set_design](#)

[set_macro_model](#)

set_macro_model

Indicates the start of the CPF content that describes the power behavior of a custom IP.

Only the following CPF commands are supported in a macro CPF module:

- `set_macro_model <macro_model_name> -cells`
- `create_nominal_condition -name -voltage -ground_voltage -state`
- `create_power_domain -name -default -instances -boundary_ports
-shutoff_condition -power_up_states
-power_domain_states -active_state_conditon
-default_isolation_condition
-default_restore_edge_condition
-default_save_edge_condition -base_domains`
- `create_isolation_rule -name -from -to -pins -isolation_output_condition
-exclude -secondary_power_domain`
- `create_state_retention_rule -name -domain -instances -exclude
-required -restore_edge -save_edge
-restore_precondition -save_precondition
-retention_precondition -secondary_domain`
- `create_power_mode -name -domain_conditions`
- `end_macro_model <macro_cell> <macro_model_name>`

Commands issued from inside a macro model are not intended to drive the implementation of the macro, but rather they are intended to describe the behavior of the macro model. A macro model specification should explicitly associate each non power and ground port to a power domain.



Within a macro model definition, you should not have another CPF model.

The syntax for this command is:

```
set_macro_model {<macro_model_name>}  
[-cells <cell_list>]
```

<macro_model_name>

Specifies the name of the macro cell or model for which the CPF description follows.

-cells <cell_list>

Lists the names of the cells to which the macro model applies. The model can only be applied to instances of a cell whose name exactly matches one of the specified names.

When this option is not specified, the macro model can be applied to instances of any library cell.

This option enables you to create one macro model representing a family of macro cells. This model can be reused for all instances of these cells.

The cell names should not contain wildcards.

Related Commands

[end_macro_model](#)

UXE Command Reference Manual

Common Power Format Commands

Run-time Commands

This chapter describes the XEL commands that you can use to run design verification on the circuit design.

The compile-time commands are discussed in [Chapter 8, “User Data Commands.”](#)



You can use different directories for the compiled database and for running the Debugger tool so that the Debugger files that are created during emulation, are separate from the design database files.

You can run only one debug session from a single directory. However, you can run multiple debug sessions from different debug directories using the same design directory.

The format of specifying the hierarchical path to an object as an option to a run-time command, such as the probe command, varies in the ICE and SA modes.

In the ICE mode, the hierarchical path should be specified as illustrated in the following example:

```
tb.i_dut_top.ch[0].gen_module.i_sub_domain .dout
```

In the SA mode, the same hierarchical path should be specified as illustrated in the following example:

```
tb.i_dut_top.ch_0.gen_module.i_sub_domain .dout
```

In the above examples, note the text in bold for the differences in the hierarchical paths for the ICE and SA modes.

Run-time Commands Set

The following table lists all run-time commands. The table also lists the restrictions for using these commands in xeDebug offline sessions and vvmDebug.

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>assertion</u>	Not Supported	Not Supported
<u>bm</u>	Supported	Supported
<u>clockConfig</u>	Not Supported	Not Supported
<u>configPM</u>	Not Supported	Not Supported
<u>convertRadix</u>	Supported	Supported
<u>database [-open] <database_name> [-event][-gzip][-continuousupload][-phy]</u>	Partially Supported The following options are not supported: -continuousupload -phy	Partially Supported The following options are not supported: -continuousupload -phy
<u>database -close <database_name></u>	Supported	Supported
<u>database -show [<database_name>]</u>	Supported	Supported
<u>database -upload</u>	Supported	Supported
<u>database -clear</u>	Supported	Supported
<u>database -tracewindow -start <time> -end <time></u>	Supported	Supported
<u>database -listphyfile [<phy fileName> ...]</u>	Supported	Not Supported
<u>database -prepareOffline</u>	Not Supported	Not Supported
<u>database -disable <database_name></u>	Not Supported	Not Supported
<u>database -enable <database_name></u>	Not Supported	Not Supported
<u>database -captureMode [controlled uncontrolled]</u>	Not Supported	Not Supported
<u>debug</u>	Supported	Supported
<u>deposit</u>	Not Supported	Not Supported
<u>download</u>	Not Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
dpa -phyfile <phy_filename>	Supported	Not Supported
dpa -listphyfile [<phy_fileName> ...]	Supported	Not Supported
dpa -tracewindow [-start <time> {-end <time> -size <time>}]	Supported	Supported
dpa -setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>] <instance_names>	Supported	Supported
dpa -setupToggleCount -listinst [-verbose]	Supported	Supported
dpa -setupToggleCount -rminst <instance_names>	Supported	Supported
dpa -setupToggleCount -rminst -all	Supported	Supported
dpa -getToggleCount [-append] [-weighted] [-file <name>]	Supported	Supported
dpa -list [-max <n>]	Supported	Supported
dpa -addinst [-excludelibrary <.llib names>] [-allnets] -depth <n> <instance> ...	Supported	Supported
dpa -addinstoutputpin <instance> [-gatecell <gatecell filename>]	Supported	Supported
dpa -addinstpin <instance>	Supported	Supported
dpa -addcone -depth <n> <signal> ...	Supported	Supported
dpa -addpath -depth <n> {<net1> <net2>}	Supported	Supported
dpa -readNetNames <rc dpa info file name> [<instanceName>]	Supported	Supported
dpa -rm <signal>...	Supported	Supported
dpa -rm [-regexp -glob -pattern <pattern>]*	Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
dpa <u>-outfile -tcf <filename> [-instance <instanceName>] [-masterTCF] [-slaveTCF] [-noregularTCF] [-segment <n>% <cycleNumber>] [{-start <time>} {-end <time>}]</u>	Supported	Supported
dpa <u>-upload</u>	Supported	Supported
dpa <u>-saif -load <saif_file_name> [-module <module_name_pattern>]</u>	Supported	Supported
dpa <u>-saif -listmodule [-lib]</u>	Supported	Supported
dpa <u>-saif -listlib</u>	Supported	Supported
dpa <u>-saif -listfile</u>	Supported	Supported
dpa <u>-saif -rmmodule <module_name_pattern> [<lib_name>]</u>	Supported	Supported
dpa <u>-saif -listduplicateModule</u>	Supported	Supported
dpa <u>-saif -rmlib <lib_name></u>	Supported	Supported
dpa <u>-saif -rmfile <file_name></u>	Supported	Supported
dpa <u>-saif -addscope <instance_name></u>	Supported	Supported
dpa <u>-saif -rmscope <instance_name></u>	Supported	Supported
dpa <u>-saif -listscope</u>	Supported	Supported
dpa <u>-outfile -saif <saif_file_name> [-instance <instanceName>] [-segment <cycleNumber> <percent>] [{-start <time>} {-end <time>}]</u>	Supported	Supported
dpa <u>-genPowerProfile [-refine]</u>	Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
dpa <u>-setupRC -show</u> dpa <u>-setupRC -reset</u> dpa <u>-setupRC { -gateFlow -rtlFlow }</u> dpa <u>-setupRC -servers { <machine_list> }</u> dpa <u>-setupRC -libPath { <path_name_list> }</u> dpa <u>-setupRC -libList { <lib_name_list> }</u> dpa <u>-setupRC -designPath { <path_name_list> }</u> dpa <u>-setupRC -designFile { <file_name_list> }</u> dpa <u>-setupRC -topDesign <top_module_name></u> dpa <u>-setupRC -instance <instance_name></u> dpa <u>-setupRC -depth <N></u>	Supported	Supported
dpa <u>-setupPowerProfile -show</u> dpa <u>-setupPowerProfile -reset</u> dpa <u>-setupPowerProfile -database <name></u> dpa <u>-setupPowerProfile -tcfDir <name></u> dpa <u>-setupPowerProfile -segment {<n>% <cycleNumber>}</u>	Supported	Supported
dpa <u>-sparse_sampling [<N>]</u>	Not Supported	Not Supported
drtl	Not Supported	Not Supported
force	Not Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<code>genPowerProfile</code>	Supported	Supported
<code>getCableStatus</code>	Not Supported	Not Supported
<code>getCycleNum</code>	Not Supported	Supported
<code>getHostStatus</code>	Not Supported	Not Supported
<code>getPMStatus</code>	Not Supported	Not Supported
<code>getTime</code>	Not Supported	Supported
<code>hdlConfig</code>	Not Supported	Not Supported
<code>hdlRunLogFile</code>	Not Supported	Not Supported
<code>host</code>	Not Supported	Not Supported
<code>infiniTrace -prepare <session_name> [-append]</code>	Not Supported	Not Supported
<code>infiniTrace -on -off</code>	Not Supported	Not Supported
<code>infiniTrace -rm <name></code>	Not Supported	Not Supported
<code>infiniTrace -observe <name></code>	Not Supported	Not Supported
<code>infiniTrace -goto [-continuel] <time specification></code>	Not Supported	Not Supported
<code>infiniTrace -stateSaveFrequency <value></code>	Not Supported	Not Supported
<code>infiniTrace -getStateSavePoints</code>	Not Supported	Not Supported
<code>infiniTrace -getCurrentMode</code>	Not Supported	Not Supported
<code>infiniTrace -getCurrentMode [-text]</code>		
<code>infiniTrace -getSessionName</code>		
<code>infiniTrace -isCaptureEnabled</code>	Not Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
infiniTrace <u>-bookmark -add</u> <u><book_mark_name></u>	Not Supported	Not Supported
infiniTrace <u>-bookmark -rm</u> <u><book_mark_name></u>		
infiniTrace <u>-bookmark -list</u>		
infiniTrace <u>-bookmark -goto</u> <u><book_mark_name></u>		
infiniTrace <u>-close</u>	Not Supported	Not Supported
infiniTrace <u>-close [-force]</u>	Not Supported	Not Supported
infiniTrace <u>-list [-info]</u>	Not Supported	Not Supported
infiniTrace <u>-intervals</u>	Not Supported	Not Supported
irun	Not Supported	Not Supported
lp (Low-power Run-time Command)	Not Supported	Not Supported
memory (in SA Mode)	Not Supported	Partially Supported
memory <u>-dump {<file_format></u> <u><memory_name> -file <file_name>} \</u> <u>[-append] [-start</u> <u><start_number>] [-end</u> <u><end_number>] [-skipzero] \</u> <u>[-addrRadix <radix>] [-</u> <u>dataRadix <radix>] [-z]</u>	Not Supported	Supported
memory <u>-load {[<file_format>]</u> <u><memory_name> -file <file_name>} \</u> <u>[-start <start_number>] [-</u> <u>end <end_number>] \</u> <u>[-fileoffset <offset_number></u> <u> continue]</u>	Not Supported	Not Supported
memory <u>-list</u>	Not Supported	Supported
memory <u>-set {-all </u> <u><memory_name>...}</u>	Not Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<code>memory -reset {-all <memory_name>...}</code>	Not Supported	Not Supported
<code>memory -setvalue {-all <list_of_instances>...} [-start <addr1>] [-end <addr2>] \ -value {<value> all0 all1}</code>	Not Supported	Not Supported
<code>memory -setvalue {-all <list_of_instances>...} [-start <addr1>] [-end <addr2>] \ -random <seed></code>	Not Supported	Not Supported
<code>memory -getFileOffset <file_name></code>	Not Supported	Not Supported
<code>memory -sparse_mem_info [<mem_name>]</code>	Not Supported	Not Supported
<code>memory (in ICE Mode)</code>	Not Supported	Partially Supported
<code>memory -dump {<file_format>} <memory_name> -file <file_name> \ [-append] [-start <start_number>] [-end <end_number>] [-nooverwrite] \ [-skipzero] [-addrReg <register_name>] [-clearAddrReg] \ [-full <number>] [-addrRadix <radix>] [-dataRadix <radix>] [-z] [-gz] \ [-dir <directory_name>] [-background -bg] [-screen] [-lines <n>]</code>	Not Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<code>memory -load {[<file_format>] <memory_name> -file <file_name> [-retainValue] [-retainValueOffset]} \ [-start <start_number>] [-end <end_number>] \ [-fileoffset <offset_number>} continue] [-dir <directory_name>] \ [-backgroundl-bg] [-nofill] [-nochecksize]</code>	Not Supported	Not Supported
<code>memory -list</code>	Not Supported	Supported
<code>memory -info {-all <memory_name>...} [-help]</code>	Not Supported	Supported
<code>memory -set {-all <memory_name>...}</code>	Not Supported	Not Supported
<code>memory -reset {-all <memory_name>...}</code>	Not Supported	Not Supported
<code>memory -reset {-all <memory_name>...}</code>	Not Supported	Not Supported
<code>memory -setvalue {-all <list_of_instances>...} [-start <addr1>] [-end <addr2>] \ -value {<value> all0 all1}</code> <code>memory -setvalue {-all <list_of_instances>...} [-start <addr1>] [-end <addr2>] \ -random <seed></code>	Not Supported	Not Supported
<code>memory -getFileOffset <file_name></code>	Not Supported	Not Supported
<code>memory -foreground -fg</code>	Not Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>memory -sparse_mem_info [-v] <mem_name></u>	Not Supported	Not Supported
<u>memory -sparse_mem_info -usedpages [-v] <mem_name></u>		
<u>memory -sparse_mem_info -statistic [-v] [-byusage]</u>		
<u>memory -sparse_mem_info -overflow</u>		
<u>memory -opt <global_options></u>	Not Supported	Supported
<u>memory -optrm <global_options></u>	Not Supported	Supported
<u>memory -opt</u>	Not Supported	Supported
<u>memory -optclear</u>	Not Supported	Supported
<u>memory -optstack</u>	Not Supported	Supported
<u>memory -optpush</u>	Not Supported	Supported
<u>memory -optpop</u>	Not Supported	Supported
<u>memory -optuse [<n>]</u>	Not Supported	Supported
<u>ncsim</u>	Not Supported	Not Supported
<u>probe -create {[-all][-allnets][-depth <n> all to_cells]} [-name <probe_name>] [-ports][-waveform} } \ {<object> ... <scope_name>} [-addcone -depth <n> <signal> ...] \ [-assertion [-state]] [-errorok] [-exclude {<object> <scope_name>}] [-excludelibrary <.llib names>] [-signals][-fast]</u>	Supported	Supported
<u>probe -delete <probe_name> [<probe_name>...]</u>	Supported	Supported
<u>probe -list [<probe_name>...] [-fast]</u>	Supported	Supported
<u>probe -save [<filename>] [-fast]</u>	Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
probe <u>-show [<probe_name>...][-fast]</u> [-v]	Supported	Supported
probe <u>-addcone -depth <n> <signal> ...</u>	Supported	Supported
probe <u>-addpath -depth <n> <net1></u> <u><net2></u>	Supported	Supported
probe <u>-addIllegalDomainConfiguration</u> <u>{<illegalDomainConfiguration> *}</u>	Supported	Not Supported
probe <u>-addMode {<modeName> *}</u>	Supported	Not Supported
probe <u>-addModeTransition</u> <u>{<modeTransitionName> *}</u>	Supported	Not Supported
probe <u>-addPowerDomain</u> <u>{<powerDomainName> *}</u>	Supported	Not Supported
probe <u>-addPowerMode</u> <u>{<powerModeName> *}</u>	Supported	Not Supported
probe <u>-addIsolation</u> <u>{<isolationRuleName> *}</u>	Supported	Not Supported
probe <u>-addRetention</u> <u>{<stateRetentionRule> *}</u>	Supported	Not Supported
probe <u>-addpowerdomain</u> <u>[<namePattern>]</u>	Supported	Not Supported
probe <u>-addisolation [<namePattern>]</u>	Supported	Not Supported
probe <u>-addretention [<namePattern>]</u>	Supported	Not Supported
probe <u>-addsupplyport [<namePattern>]</u>	Supported	Not Supported
probe <u>-addsupplynet [<namePattern>]</u>	Supported	Not Supported
probe <u>-addsupplyset [<namePattern>]</u>	Supported	Not Supported
probe <u>-addswitch [<namePattern>]</u>	Supported	Not Supported
probe <u>-addpst [<namePattern>]</u>	Supported	Not Supported
probe <u>-addportstate [<namePattern>]</u>	Supported	Not Supported
probe <u>-addpststate [<namePattern>]</u>	Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>release</u>	Not Supported	Not Supported
<u>reset</u>	Not Supported	Not Supported
<u>resetTime</u>	Not Supported	Not Supported
<u>restart</u>	Not Supported	Supported
<u>restart (in VVM)</u>	Not Supported	Supported
<u>resultWaveform</u>	Not Supported	Not Supported
<u>resume</u>	Not Supported	Not Supported
In the Logic Analyzer and Dynamic Target mode under IXCOM: <u>run [-emulation] [-wait -nowait] [-continue]</u>	Not Supported	Partially Supported Not Supported: -wait -nowait
In the STB mode: <u>run [-wait -nowait] [-continue]</u> <u>[<amount_to_run></u> <u>[<time_unit>]]</u>	Not Supported	Partially Supported Not Supported: -wait -nowait
In the Vector Debug mode: <u>run [<cycles>]</u>	Not Supported	Not Supported
In all SA modes, except Dynamic Target mode: <u>run [-wait -nowait] [-continue]</u> <u>[-swap]</u> <u>[-ignorestop]</u> <u>[<amount_to_run></u> <u>[<time_unit>]]</u>	Not Supported	Partially Supported Not Supported: -wait -nowait -ignorestop -swap
Saving the design state for SA mode: <u>save [-simulation] <snapshot_name> [-overwrite]</u>	Not Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
Saving the design state for ICE mode: <code>save <save_name> [-overwrite]</code>	Not Supported	Not Supported
Saving the run-time environment: <code>save {-commands -environment} [<filename>]</code>	Not Supported	Only the -environment option is supported
Saving the user-defined Tcl variable: <code>save -uservars <filename></code>	Not Supported	Not Supported
<code>scope</code>	Supported	Supported
<code>scope [-set] <scope_name></code>	Supported	Supported
<code>scope -up</code>	Supported	Supported
<code>scope -describe [<scope_name>]</code>	Supported	Supported
<code>scope -show</code>	Supported	Supported
<code>sdl -autoProbe [no yes]</code>	Not Supported	Supported
<code>sdl -addProbes [-fast] <sdl_file>...</code>	Not Supported	Supported
<code>sdl -disable -save <filename></code> <code>sdl -disable -list</code> <code>sdl -disable [-now]</code> <code>sdl -disable [-now]</code> [<instance_names_list> -none] <code>sdl -disable [-now] -label <label-pattern> <label-pattern> ...</code> <code>sdl -disable -label -list</code>	Not Supported	Supported Note: Disabling is always instant and does not require the -now option.

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>sdl -display -console [0 1]</u> <u>sdl -display -endofline 0 1</u> <u>sdl -display -file [<file_name>]</u> <u>sdl -display -overflow [error warning ignore]</u> <u>sdl -display -translate 0 1</u> <u>sdl -display -id <ID> [-file <file_name>]</u> <u>sdl -display -id <ID> -disable -enable</u> <u>sdl -display -loglevel <loglevel></u> <u>sdl -display -deflevel <deflevel></u>	Not Supported	Not Supported
<u>sdl -enable -list</u> <u>sdl -enable [-now]</u> <u>sdl -enable [-now] [<instance_names_list> -none]</u> <u>sdl -enable [-now] -label <label-pattern> <label-pattern> ...</u> <u>sdl -enable -label -list</u>	Not Supported	Supported Note: Enabling is always instant and does not require the -now option.
<u>sdl -expression [<expression>]</u>	Not Supported	Supported
<u>sdl -execOvf [error warning ignore]</u>	Not Supported	Not Supported
<u>sdl -getActiveInstances</u>	Not Supported	Supported
<u>sdl -getAssertions</u>	Not Supported	Not Supported
<u>sdl -getFiles</u>	Not Supported	Supported
<u>sdl -getInstances</u>	Not Supported	Supported
<u>sdl -getLabels [-detail]</u>	Not Supported	Not Supported
<u>sdl -getNets</u>	Not Supported	Supported
<u>sdl -getStatus [-instance <instance_name>] <parameter></u>	Not Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>sdl -getTriggers</u>	Not Supported	Supported
<u>sdl -haltOnTrigger [0 1]</u>	Not Supported	Not Supported
<u>sdl -isActive</u>	Not Supported	Supported
<u>sdl -isDispOvf [-clear]</u>	Not Supported	Supported; always returns 0, which indicates no display buffer overflow.
<u>sdl -isExecOvf [-clear]</u>	Not Supported	Supported; always returns 0, which indicates no execution overflow.
<u>sdl -isTrigger [-instance <instance_name>] [-label <label>]</u>	Not Supported	Partially Supported: Not Supported Options: ■ -instance ■ -label
<u>sdl -ldcnt1 -ldcnt2 [-instance <instance>] <value></u>	Not Supported	Not Supported
<u>sdl -load [<sdl file name>]</u>	Not Supported	Supported
<u>sdl -log -stopStream -startStream -upload [<size>] -reset</u>	Not Supported	Not Supported
<u>sdl -ncbreakpoints -errorok [yes no]</u> <u>sdl -ncbreakpoints -import [<filename>]</u> <u>sdl -ncbreakpoints -exact [yes no]</u>	Not Supported	Not Supported
<u>sdl -onlyReport [yes no]</u>	Not Supported	Not Supported
<u>sdl -pp <output file> [<sdl file name>]</u>	Not Supported	Supported
<u>sdl -report [-file <file name>]</u>	Not Supported	Supported
<u>sdl -restoreState <file></u>	Not Supported	Not Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>sdl -saveState <file></u>	Not Supported	Not Supported
<u>sdl -scope [<scope>]</u>	Not Supported	Supported
<u>sdl -setFile [<sdl_filename>]</u>	Not Supported	Supported
<u>sdl -simTime yes no</u>	Not Supported	Supported
<u>sdl -stop</u>	Not Supported	Not Supported
<u>sdl -traceClear run dump never</u>	Not Supported	Not Supported
<u>sdl -traceDump [-location] [-lineOnly] [-short] [-index c dc t dt] [-instance <instance_name>] [-text -fsdb -sst2] [-time <time_unit>] <file_name></u>	Not Supported	Not Supported
<u>sdl -traceOn [-acqFilter no yes] [-postTrigger no yes] [<action_list> ALWAYS DEFAULT]</u>	Not Supported	Not Supported
<u>sdl -tx {-sst2 -text -ida} <filename> -close -show</u> <u>sdl -tx -illegal [error warning ignore reportwarning clearwarning]</u>	Not Supported	Not Supported
<u>sdl -verify [<filename>]</u>	Not Supported	Supported
<u>sdl -when [-time] [<time_unit>]</u>	Not Supported	Supported
<u>stop</u>	Not Supported	Partially Supported
<u>stop -emulation</u>	Not Supported	Not Supported
<u>suspend</u>	Not Supported	Not Supported
<u>symbol</u>	Not Supported	Supported
<u>tca</u>	Supported	Supported
<u>userData</u>	Supported	Supported
<u>value</u>	Not Supported	Supported
<u>vector</u>	Not Supported	Not Supported
<u>vvm</u>	Not Supported	Supported

UXE Command Reference Manual

Run-time Commands

Table 13-1 Run-time Commands Set

Command	xeDebug Offline Session Restrictions	vvmDebug Restrictions
<u>waitWhileBusy</u>	Not Supported	Supported
<u>xeset</u>	Partially Supported Not Supported: -appendtrace	Partially Supported Not Supported: -appendtrace
<u>xogui</u>	Supported	Supported

assertion

Provides support for assertions on UXE platforms.

You can use the *glob-style* wildcard characters for only assertion labels and not scopes, if supported (as is done for all other commands that support assertionLocators). All the run-time assertion commands begin with the word `assertion` followed by the relevant command, for example `assertion -on` command include all the options required for re-activating assertions that had been disabled previously.

Assertion run-time commands are case-insensitive and can be abbreviated to minimum unique characters. As with other CLI commands, the invocation of these commands is echoed to the `xe.msg` log file.

The behavior of most assertion run-time commands is identical whether the DUT is currently in the emulator or simulator.

The `describe` and `scope -describe` Tcl commands support the assertion objects. These commands are supported to take full advantage of IES functionality. For detailed description of these commands and their syntax, refer to the *Cadence NC-Verilog Simulator Help* documentation.

Note: The Assertion run-time commands are not supported with `vvmDebug`.

For the unsupported assertion-related Tcl variables, refer to the [Unsupported Commands in irun](#) section of [Chapter 2, “UXE Command List.”](#)

To include the assertions in the tracelist, use the standard `probe` run-time command. The `probe` command enables you to trace both testbench and DUT assertions in SST2 (SHM) format.

Note:

At compile time in the SA mode, specifying the `+no_assertCtrl` option of the `ixcom` command prevents run-time usage of the assertion run-time commands.

Similarly, if the `+assertStatus` option is not used with the `ixcom` command, waveforms and fetching of assertion values are not supported. If these are requested when not available, the following warning is returned:

WARNING (ABART-) Assertion state values are not supported. Ensure that the `ixcom '+assertStatus'` switch is present to instrument state values.

Also, if the `+assertCover` option is not used with the `ixcom` command, the assertion summary is limited to only COVER assertions, the `assertion -counter` command is not supported, and waveforms do not include the counter data. In addition, the following warning is returned:

UXE Command Reference Manual

Run-time Commands

WARNING (ABART-) Assertion counters are not supported. Ensure that the ixcom '+assertCover' switch is present to instrument counters.

For detailed information on the ixcom command and its various options, refer to [Chapter 10, “Commands and System Tasks for SA Mode.”](#)

The syntax for this command is:

```
assertion -check
assertion -counter <counter_list> <assertion_pathname>
assertion -get [-depth <numHierLevels>] [-directive <directiveTypes>]
    <assertionLocator>
assertion -off [-onfailure [<failureLimit>]] [-depth <numHierLevels>] [-directive
    <directiveTypes>]
    <assertionLocator>
assertion -list [-depth <numHierLevels>] [-directive <directiveTypes>] [-failed] [-
    uncovered] [-signals]
    <assertionLocator>
assertion -on [-depth <numHierLevels>] [-directive <directiveTypes>]
    <assertionLocator>
assertion -summary [-final] [-byName] [-byFailure] [-show <counter_list>] [-directive
    <directiveTypes>]
    [-redirect <file_name>] [-tclResult]
```

-check

Returns a count of the number of assertions found in the design. This also includes the testbench assertions. The value is returned through the Tcl result, and not to stdout. This command enables common command files to be used for designs that might be built with or without assertions. Then, assertion commands are conditionally executed depending on whether or not an assertion exists.

If the design does not contain any assertions, an error condition or warning message is not generated; instead, the Tcl status is returned as 0. This command generates an error condition only if there are any execution errors in examining the database and design.

Example

If a design includes 23 assertions, the following command returns the output as 23:

```
puts [assertion -check]
23
```

-counter <counter_list> <assertion.pathname>

Returns the current values of the specified counters for the specified assertion. The result is returned as a space-separated list with the values in the same order as the counters arguments are listed.

The *<counter_list>* argument specifies a space-separated list of assertion counter types, which can be any of the following:

- checked
- finished
- failed



If more than one counter type needs to be specified, enclose the *counter_list* values within curly braces { }; otherwise, the curly braces can be omitted.

The *<assertion.pathname>* argument specifies the path of the required assertion.

When running designs in the Dynamic Target mode, the output from the `assertion -counter` command is from the time that the emulator counter value was accessed.

-get

Returns the current values or states of the specified assertions. The value is returned as the value of the Tcl command and not displayed on stdout. When multiple assertions are selected, the values are returned in a space-delimited Tcl list. To explicitly send the value to stdout, use the `puts` Tcl command. For example:

```
puts [assertion -get inst1.assert_always_on]
```

Multiple concurrent assertion threads are possible at simulation time and the returned value reflects only one resolved value. In such cases, the following list of states, in the given decreasing order of precedence, is used for resolving values from different threads of the same assertion:

1. disabled
2. failed
3. finished
4. active

5. inactive

This means that if two threads have assigned finished and failed as the values to the same assertion, the resolved value for that assertion will be failed. This is the same value that is written to the waveform database.

When multiple assertion values are returned, the following example illustrates how assertion names can be associated with their corresponding values:

```
set assertionNames [assertion -list DUT.inst1.*]
set assertionValues [assertion -get DUT.inst1.*]
for {set i 0} {$i < [llength $assertionNames]} {incr i}
{
    puts "Assertion [lindex $assertionNames $i] = [lindex $assertionValues $i]"
}
```

Note: In the SA mode, support of assertion values requires that the +assertStatus option of the ixcom command is used at compile time.

The assertion -get command is not supported in the Dynamic Target mode in the SA mode.

Using the same syntax as for the assertion -on and assertion -off commands, wildcard symbols, such as * can be used with the assertion -get command. Note that these are glob-style wildcarded.

The complete syntax for the assertion -get command is:

```
assertion -get [-depth <numHierLevels>] [-directive <directiveTypes>]
               <assertionLocator>
```

-off

Deactivates reporting and recording of the specified assertions or those listed within the specified scopes. Also, any pending evaluations of the specified assertion are aborted.

Assertions can be deactivated on a module-instance basis or individually.

Note: In the SA mode, support for assertion disabling is enabled only if the +assertStatus option of the ixcom command is used at compile time.

If the specified assertion is already disabled, no action is taken and a warning message is returned. This command might be used to disable all compiled assertions or specific assertions by using a *glob-style* wildcarded subset or -depth all <*topDesignInstance*>.



An assertion can be enabled again after being disabled using the `assertion -on` command. The assertion counts are continued from the point at which they were disabled. When the assertion is enabled for the first time, the counts start from zero.

Protected assertions cannot be disabled; neither by specific commands nor by wildcarding the parent scope.

Assertion state is maintained in hardware and is sensitive to the assertion clock. Therefore, even though an assertion is effectively disabled immediately after the `assertion -off` command is executed, the waveform does not reflect the Disabled state until the next assertion clock edge.

The complete syntax for this command is:

```
assertion -off [-onfailure [<failureLimit>]] [-depth <numHierLevels>] [-directive <directiveTypes>]  
                  <assertionLocator>
```

-onfailure [<failureLimit>]

Indicates that the selected assertions will be disabled on failure.

The `<failureLimit>` optional argument can be used to specify the maximum number of failures to be reported before disabling the specified assertions. This value must be a positive integer. By default, the `<failureLimit>` value is set to 1. This implies that the assertion should be disabled on the first failure.

If you use another assertion command, such as `assertion -on` or `assertion -off` after using the `assertion -onfailure [<failureLimit>]` command, the `<failureLimit>` is cleared, and will not be applicable.

Example

Disable all assertions in the top-level scope after first failure:

```
assertion -off -onfailure *
```

Disable specific assertion after 5 failures:

```
assertion -off -onfailure 5 inst1 ASSERT_ALWAYS1
```

After assertion fails and is disabled, re-enable for three more failures:

```
assertion -off -onfailure 3 inst1 ASSERT_ALWAYS1
```

```
assertion -on inst1 ASSERT_ALWAYS1
```

-list

Returns a space-separated list of assertions selected by the *<assertionLocator>* and associated options.

This command provides a means of identifying the assertions that are selected when using other assertion commands without actually acting on those assertions. Also, this command can be used to locate or verify assertions in the design. It can also be used to associate values returned by the `assertion -get` command with the corresponding assertion name.

When this command is embedded in a command file, the values are not displayed to stdout unless explicitly sent to stdout using a 'puts' command, as shown below:

```
puts [assertion -list -depth 0 top]
```

Using the same syntax as for the `assertion -on` and `assertion -off` commands, wildcard symbols, such as * can be used with the `assertion -list` command. Note that these are glob-style wildcared.

The `-failed` and `-uncovered` options of the `assertion -list` command are especially useful for reporting assertions information in the dynamic mode.

The complete syntax for the `assertion -list` command is:

```
assertion -list [-depth <numHierLevels>] [-directive <directiveTypes>] [-failed] [-uncovered] [-signals] <assertionLocator>
```

-failed

Returns a cumulative list of all the assertions that have failed since time 0 as a Tcl result list.

-uncovered

Returns a list of assertions that are still uncovered at the time of the emulator state access, as a Tcl result list.

Using both the `-failed` and `-uncovered` options returns a set of assertions that have either failed since time 0 or are still uncovered at the time of the emulator state access.

In the Dynamic Target mode, the design clocks continue to run while processing the `assertion -list` command; therefore, the assertion counters might change before the

results are reported. The output always presents the counter values at the time they were examined and not any particular simulation time.

-signals

Adds the assertion fanin signals to the probe set if any assertions are specified. This option can also be used with the [probe](#) run-time command. For detailed information, refer to the *Support for Assertion Fanin Signals* section of the *Compiling and Running Designs with Assertions* chapter of *UXE User Guide*.

-on

Re-activates the reporting and recording of the specified assertions or those listed within the specified scopes. Assertion processing re-starts from a clean state, that is, all previous assertion thread handling is cleared and the assertion processing is started from beginning.

Assertions can be re-activated individually or on a module-instance basis.

If the assertion is already enabled, no action is taken. This command might be used to enable all assertions instrumented at compile time or explicitly specify a *glob-style* wildcarded subset.

The assertion `-on` command is not supported in the Dynamic Target mode.

The complete syntax for this command is:

```
assertion -on [-depth <numHierLevels>] [-directive <directiveTypes>]  
             <assertionLocator>
```

-summary

Summarizes the assertion data from the session start to the current simulation time and provides a listing of assertion event counts (that is, checked, finished, and failed). The generated result is unaffected by the previous summaries.

If no assertions exist in the design, a warning message to that effect is displayed.

Note: For non-COVER assertions, the `ixcom +assertCover` command must be specified to include the counter instrumentation. If this option is not executed, the following message is displayed:

```
WARNING (ABART-) Assertion counters are not supported. Ensure that the ixcom  
'+assertCover' switch is present to instrument counters.
```

UXE Command Reference Manual

Run-time Commands

The complete syntax for the assertion `-summary` command is:

```
assertion -summary [-final] [-byName] [-byFailure] [-show <counter_list>] [-directive <directiveTypes>]  
[-redirect <file_name>] [-tclResult]
```

-final

Produces a summary report at the session termination. Use of this option does not produce a summary report immediately, instead an information message as following is displayed notifying that the report will be generated at the end of the simulation:

INFO: Summary report deferred until the end of simulation.



In the Dynamic Target mode, the assertion `-summary` command is not supported without the `-final` option.

If multiple assertion `-summary -final` commands are executed, only one final summary report is produced, using the settings of the last invocation. Also, on each invocation a warning message as illustrated below is displayed:

WARNING: Previously scheduled final assertion summary report will be supersede this invocation.

Once a final summary has been scheduled, there is no means to cancel it.

-byName

Specifies that the list of assertions in the summary table should be sorted by assertion name, which is the default sort order.

-byFailure

Specifies that the list of assertions in the summary table should be sorted by number of failures.

-show <counter_list>

Limits the summary report to include only the specified space-separated list of assertion counter types, which can be any of the following:

- checked

- finished
- failed

If more than one counter type needs to be specified, enclose the *<counter_list>* values within curly braces { }; otherwise, the curly braces might be omitted.

If this option is not specified, the summary report includes all types of counters.

-redirect <file_name>

Specifies an alternate log file to dump the summary report.

Assertion Summary Output

The Assertion summary report is not printed at the end of a run, by default. Earlier, this report was printed whenever the design had been compiled with the +assertCover switch. In the current version of IXCOM, when the design is compiled with the +assertCover switch (and optionally, the +assertCoverCnt+<n> switch), the summary report can be printed at run time by using the assertion -summary command.

Following is an excerpt from a sample summary report:

```
Checked Finish Failed Assertion Name
 2   1   1    top_1.Arbiter_1.vcomp_arb_inst.assert_arb.ReqA_eventually_GntA
never 0   0    top_1.Arbiter_1.vcomp_arb_inst.assert_arb.ReqB_eventually_GntB
 3   3   0    top_1.Block_B.vcomp_arb_block.assert_arb_block.request_asserted.ial_window
 1   1   0    top_1.cover_fair_for_A
 2   2   0    top_1.cover_fair_for_B
off   1   0    top_1.tr_A_to_Slave
 1   1   0    top_1.tr_B_to_Slave
Total Assertions = 7,  Failing Assertions = 1,  Unchecked Assertions = 1
Assertion summary at time 320 NS + 0
```

Common Command Options

This section describes common options for the assertion commands:

- -depth <numHierLevels>
- -directive <directiveTypes>
- <assertionLocator>
- -tclResult

-depth <numHierLevels>

Indicates that a hierarchical level specifier is to be used.

Note: When the `-depth` switch is present, the Palladium XP assertions will be selected only if the specified root hierarchical path is within the Palladium XP partition.

The `<numHierLevels>` switch specifies the number of hierarchical levels (including the parent) to be traversed in the process of locating assertions for applying this command. This value must be a positive integer to indicate the number of hierarchical levels to be included.

Specifying the following values signifies:

- 0 or all: Indicates the entire depth of the selected hierarchy.
- 1: Indicates all the assertions in the specified scope only. This is the default value if the `-depth` option is not specified.
- 2: Indicates the specified scopes plus one level of hierarchy below each scope. This argument has no effect for explicit assertion pathnames.

Specifying a negative `<numHierLevels>` argument generates the following error and the command is ignored:

ERROR: Argument -1 to assertion -get -depth option must be 'all' or a non-negative integer.

-directive <directiveTypes>

Filters the selected assertions by including only the assertions of the specified directive type.

The `<directiveTypes>` argument is a curly braces '{}' enclosed, space separated list of assertion directive types which can be any of the following values:

- assert
- assume
- cover
- restrict

For example:

```
assertion -get -directive {assume cover}
```

Otherwise, exactly one of the following might be specified:

- all

■ none

However, these arguments are not really useful. This is because the `all` argument is equivalent to omitting the `-directive` option and the `none` argument pre-empts the entire command. These arguments are provided only for IES consistency. If one of the above is used with other values in a directive list, the following error is generated:

ERROR: Assertion Directive Type '<all|none>' is not valid in list context, must be one of: assert, assume, restrict, or cover.

If only one directive type is specified, the braces might be omitted.

For example:

```
assertion -get -directive cover
```

If an incorrect `<directiveTypes>` argument is specified, the following error is generated:

ERROR: Assertion Directive Type '<invalidType>' specified, must be one of: assert, assume, restrict, or cover.

If no `<directiveTypes>` argument is specified, the following error is generated:

ERROR: Missing parameter for 'assertion -get' '-directive' option.

<assertionLocator>

Specifies a space-separated list of hierarchical paths to the component instances or specific assertion pathnames that will be traversed for this command. Glob wildcarded names are supported for assertion label names, but not instances.

Hierarchical paths use periods (.) as delimiters except for designs with VHDL tops which will use ‘:’ for the top instance name. The current design scope set using the `scope` command is used to identify the selected assertions. Scopes and absolute paths should be specified starting with the top module type name (user design top module). For VHDL-top user designs, this top path instance will be ‘:’.

A warning message is displayed if the specified scope does not contain any assertions or if the specified entry is not a design scope.

Escaped pathnames and scopes are also supported.

As a special case, the `-all` option might be used in place of any scopes or assertion pathnames to select all assertions in the design.

Note: A protected assertion is not selected either if explicitly specified or when owned in a specified scope.

-tclResult

Suppresses dumping of the assertion summary data to stdout and instead returns the data as a Tcl value. For example, the following Tcl command sets the `summary` variable to the entire summary table – possibly for easy command file parsing:

```
set summary [assertion -summary -tclResult]
```

bm

The `bm` command is used to define bookmarks for specific time points. You can use the defined bookmarks as a parameter to other commands. The `bm` command can be used in both `xeDebug` and `vvmDebug`.

For example, you create two bookmarks, `bm1` for 250 fclk and `bm2` for 500 fclk. You might use them to upload waveform from 250 fclk to 500 fclk, as follows:

```
database -upload -start bm1 -end bm2
```

The syntax for this command is:

```
bm -add <bookmark_name> [<time_specification>] [-replace]  
bm -list [-glob | -regexp <pattern> [-nameOnly]]  
bm -rm *  
bm -rm <bookmark_name>  
bm -save <file_name>  
bm -import <file_name> [-replace]  
bm <bookmark_name>
```

-add <bookmark_name> [<time_specification>] [-replace]

Defines a new bookmark with the specified name. If the time is not specified, the new bookmark is created for the current time. If a bookmark already exists with the specified name, an error message is shown. To replace an already defined bookmark, specify the `-replace` option. A bookmark name should be a legal Tcl identifier.

Note:

- ❑ A bookmark name should not include the "*" character. This character is used in the `bm -rm *` command, which removes all the pre-defined bookmarks.
- ❑ A bookmark name should not start with a "-" symbol as this symbol indicates the XEL command options.
- ❑ You should not use the following special strings as the name of a bookmark:
 - `trigger`: This is a special bookmark, which indicates the last trigger point.
 - `begin` and `end`: The strings `begin` and `end` are reserved by default for start time and end time.

- A bookmark name should not be specified like a time specification, such as, 10, 10ns, 10ps, or 10fclk.

-list [-glob | -regexp <pattern> [-nameOnly]]

Lists all the currently defined bookmarks, if no option is specified with the command. If the option `-glob <pattern>` is specified, it lists the currently defined bookmark names matching the Tcl glob pattern. If the option `-nameOnly` is specified, only the bookmark names are listed.

-rm *

Removes all the defined bookmarks.

-rm <bookmark_name>

Removes the specified bookmark. You can also remove several bookmarks simultaneously using a single command.

For example,

```
bm -rm bm1 bm2 bm3
```

where *bm1*, *bm2*, and *bm3* are pre-defined bookmarks.

-save <file_name>

Saves all the defined bookmarks in a file.

-import <file_name> [-replace]

Imports the bookmarks into the *Bookmark* tab from the specified bookmark file. If any bookmark listed in the file already exists, an error message is displayed unless you specify the `-replace` option.

<bookmark_name>

Returns the time defined by the specified bookmark.

clockConfig

Changes the CAKE or SCM clock options, including oversampling value, delay, and frequency. Clock values can be changed whenever emulation clocks are not running. To stop the emulation clocks, use the `stop` command. In LA mode, use the `stop -emulation` command.

This command applies only to SCM or CAKE clocks generated when running in STB or LA mode. Restrictions apply to some clocks when using oversampling of less than 2X.

The command can also be used when running a design in the SA mode. However, some command options can be used only when there are uncontrolled clocks. This command is not supported with `vvmDebug`.

Executing the `clockConfig` command without any options or with the `-list` option returns all current clock configuration values.

The syntax for this command is:

```
clockConfig [options]
```

The following command options can be used in both ICE and SA modes:

```
clockConfig -sample <value>
clockConfig -dly <clockname> <value>
clockConfig -clk <clockname> <value> [<unit>]
clockConfig -list
clockConfig -restore <filename>
clockConfig -save <filename>
clockConfig -actual
```

The following command options can be used in the ICE mode and only when there are uncontrolled clocks in the SA mode:

```
clockConfig -frequency [<value> [<unit>]]
clockConfig -realclk <clockname> [<value> [<unit>]]
```

The following command option can be used only in the SA mode when there are uncontrolled clocks:

```
clockConfig -stopMode [controlled | *controlled | both | none | *none]
```

-sample <value>

Changes oversampling to the specified <value>. The <value> must be a whole number greater than 1.

This option can be used only if the design was compiled with CAKE clocks with oversampling ratio of 2X or greater. For more information, refer to the [clockFrequency](#) command.

-dly <clockname> <value>

Increases delay on the specified clock net by the number of fclk cycles specified in the <value> parameter. If this command is applied more than once on the same net, the delay will increase cumulatively. The <clockname> parameter must be the name of a net defined at compile time as a clock driven by CAKE technology. The <value> parameter must be a whole number greater than or equal to zero. This option cannot be used if the design was compiled with CAKE 1X clock generation or with SCM clock generation.

-clk <clockname> <value> [<unit>]

Changes frequency on the specified clock net to the specified <value>. The specified <value> must be a positive number. It might include a decimal point.

<unit> indicates the frequency units and can be one of the following strings: hz khz mhz ghz (Case is ignored).

If <unit> is not provided, the specified <value> is interpreted as frequency in MHz.

Note: Only the relative, not the absolute, frequencies are significant.

The following restrictions apply:

In CAKE 1X mode, the clock frequencies are divided into three ranges:

- The fastest clock frequency specified at compile time
- The range of frequencies less than the fastest, but more than half of the fastest
- The range of frequencies half or less than half of the fastest frequency

A clock specified in one of these three ranges at compile time cannot be changed to a different range at run time. The frequency of the fastest clock cannot be changed because in 1X mode, the compiler optimizes capacity using the fact that this clock is the same as the emulator's FCLK.

UXE Command Reference Manual

Run-time Commands

If you need the ability to adjust the frequency of the design's fastest clock source, at compile time, you can add a dummy clock source with a slightly higher frequency than the frequency of the fastest real clock source. In 1X mode, defining a dummy clock source could have a capacity cost. For more information, refer to the *Using a Dummy Fastest Clock Source* section of the *Modeling Clocks* chapter in the *UXE User Guide*. Also refer to the [-frequency \[<value> \[<unit>\]\]](#) option.

If the CAKE oversampling is 2 or more, at run time at least one clock must have the maximum frequency (say, F) specified during compilation. No clock can have a frequency higher than F. For example, if clock A is currently the fastest clock source and has frequency F, and you want to lower its frequency, you must first raise the frequency of some other clock source (say, B) to F. If you try to lower the frequency of A first, it is not allowed, because after lowering the frequency of A, no clock has frequency equal to F.

-list

Returns a list of the current clocks including the frequency and delay values for each clock. The frequency values are returned in MHz. It also returns the current oversampling ratio for the design.

For example, in the following output:

```
2 {{ck2} 2.000 0} {ck3} 3.000 0} {{ck4} 4.000 0}
```

the number 2 in the beginning is the oversampling ratio. It is followed by the clocks with their frequencies and delays.

-restore <filename>

Restores clock frequency from the specified file. Delay values will not be restored with this option.

-save <filename>

Saves current clock frequency and delays to the specified file. The saved delay value is accumulative delay.

-actual

Returns the actual (real time) emulation frequency of FCLK in MHz. You can use this value to gauge how fast the emulator is running.

-frequency [<value> [<unit>]]

The `clockConfig -frequency` command (without `<value> <unit>`) returns the simulated frequency of FCLK (Emulator's basic fast clock).

Note: In SA mode, this option applies only if the design has uncontrolled clocks as following:

- ❑ Returns or changes the simulated frequency of the uncontrolled base clock (multiplied by the oversampling ratio).
- ❑ Affects only the simulated frequencies of the uncontrolled clocks.

If `<value>` and `<unit>` are specified, the FCLK simulated frequency is modified to the specified value. `<value>` can be a fractional number (might contain a decimal point). `<unit>` can be one of the strings: `hz` `khz` `mhz` `ghz` (Case is ignored). If `<unit>` is not specified, the default is MHz.

Modifying the FCLK simulated frequency with this command does not affect the frequency at which FCLK actually runs (as seen, for example, by an external target system). However, it will cause re-scaling of all design clock simulated frequencies according to the same ratio, and will affect the scaling of the time axis on the waveform display. Therefore, to ensure waveform data coherency, XEL automatically closes the waveform database after changing the frequency, and re-opens a new database.

-realclk <clockname> [<value> [<unit>]]

Sets or returns the actual (that is, real time) frequency of the specified clock.

Note:

- ❑ The set frequency must be lower than F/N , where F is the emulation frequency reported by the compiler, and N is the oversampling ratio. For additional restrictions, refer to the description of the `_clk <clockname> <value> [<unit>]` option.
- ❑ In SA mode, this option applies only to the uncontrolled clocks other than the uncontrolled base clock.

If the `<value>` is specified, the actual frequency is set to that value; else, the current actual frequency is returned. The specified `<value>` must be a positive number. It might include a decimal point.

`<unit>` indicates the frequency units and can be one of the following strings: `hz` `khz` `mhz` `ghz` (Case is ignored). If `<unit>` is not provided, the specified `<value>` is interpreted as frequency in MHz.

-stopMode [controlled | *controlled | both | none | *none]

Determines how design clocks and probe tracing are stopped by the various events, such as SDL triggers or NC-Sim breakpoints, that usually cause the run to end.

```
clockConfig -stopMode  
[controlled | *controlled | uncontrolled | both | none | *none]
```

The * before the controlled and none options is a part of the string that constitutes the option. For example:

```
clockConfig -stopMode *none
```

The -stopMode option can be used only with designs that have an uncontrolled clock domain.

If unspecified, the default is controlled.

This command has no effect on the behavior of the command stop when specified with the -emulation option. The stop -emulation command always stops the controlled clocks, uncontrolled clocks, and probe tracing.

In the following explanations, the term NCsim events refers to any one of the following:

- NC breakpoints
- \$stop system tasks
- Run expired by exhausting the amount of specified simulation time in the command

The term stop command refers to any one of the following:

- The stop command without any argument (typed during non-blocking run)
- The key combination CTRL+C (typed during blocking run)

controlled

SDL trigger, NCsim events, or stop command stops the controlled clocks and probe tracing.

***controlled**

NCsim events stop the controlled clocks.

SDL trigger and stop command stops the controlled clocks and probe tracing.

uncontrolled

SDL trigger stops the uncontrolled clocks and probe tracing.

NCsim events or `stop` command stops the controlled clocks, uncontrolled clocks, and probe tracing.

Note: The `uncontrolled` option also enables pausing the clocks for various run time services such as SDL EXEC, SDL DISPLAY or TX, continuous waveform upload, or any other run time service that requires pausing the clocks.

both

SDL trigger, NCsim events, or `stop` command stops the controlled clocks, uncontrolled clocks, and probe tracing.

Note: The `both` option also enables pausing the clocks for various run time services such as SDL EXEC, SDL DISPLAY or TX, continuous waveform upload, or any other run time service that requires pausing the clocks.

none

SDL trigger stops probe tracing.

NCsim events or `stop` command stops the controlled clocks and probe tracing.

***none**

SDL trigger stops probe tracing.

NCsim events stop the controlled clocks.

`stop` command stops the controlled clocks and probe tracing.

Examples

The following command changes CAKE oversampling to 4.

```
clockConfig -sample 4
```

The following command changes the delay on clock `pci_clk` to 3 cycles. When issued after download before running emulation cycles, this command causes the first transition of `pci_clk` to be delayed 3 emulation cycles. In STB mode this command, when issued,

UXE Command Reference Manual

Run-time Commands

causes a pause of 3 cycles on `pci_clk` (the next transition of `pci_clk` will occur 3 cycles later than it would have otherwise).

```
clockConfig -dly pci_clk 3
```

The following command changes the frequency of the clock net `pci_clk` to 66 MHz.

```
clockConfig -clk pci_clk 66
```

The following command lists the current clocks, including the frequency and delay for each one.

```
clockConfig -list
```

Here is a typical value returned from the above command. The design was compiled for CAKE with 2X sampling mode, and has 2 clocks, `ck2` and `ck3` with specified frequencies of 200MHz and 300 MHz:

```
2{{ck2} 200.000 0} {ck3} 300.000 0}
```

The following command returns the simulated frequency of FCLK:

```
clockConfig -frequency
```

The value returned for the same design will be the string `600.000000` which means 600 MHz. Note that this is exactly 2X the frequency of the fastest design clock `ck3` due to the 2X sampling ratio.

The following command returns the actual frequency of FCLK:

```
clockConfig -actual
```

For example, a return value of `0.600000` means that the actual frequency of FCLK is 0.6 MHz, or in other words, the design is running at 0.1% ($=0.6/600$) of the specified simulation speed.

The following command sets the stop mode to `*controlled`, which means that NCsim events, such as breakpoints will stop the simulator, but will not stop probe tracing. Probe tracing will continue until SDL trigger (or `stop` command).

```
clockConfig -stopMode *controlled
```

configPM

Enables you to switch between different execution modes.

- In the ICE mode, the `configPM` command enable you to switch between STB and LA modes.
- In the IXCOM mode, the `configPM` command enable you to switch between the Static Target and Dynamic Target mode. This is available only under the `xc on -autorun` mode.
- In the VD mode, the use of the `configPM` command verifies that the design is compiled for VD mode.

In all run modes, the use of `configPM` command is optional, and the run mode will initially be determined automatically from the type of the design database. This command is not supported with `vvmDebug`.

Before using this command, execute the [download](#) command in the ICE mode or [host](#) command in IXCOM mode.

Execute the `configPM` command to switch between the LA and STB debugging modes, or when running under IXCOM, between Dynamic Target mode and Static Target mode. The command can be executed at any time during the debug session after the design is downloaded, except when the emulator is busy (during a non-blocking run).

When the `configPM` command is specified without arguments, the current mode is returned as follows:

- LA - Logic Analyzer mode
- STB - Static Target testBench (STB) mode
- STA - Static Target mode (under IXCOM)
- DYN - Dynamic Target mode (under IXCOM/autorun)
- VD - Vector Debug mode

The returned value can be useful inside the `~/.rtxerc` initialization file to determine the type of the design database because the script inside the file is shared by all design database types.

The syntax for this command is:

```
configPM [-la | -stb | -staticTarget | -dynamicTarget | -vd]
```

-la

Switches to the Logic Analyzer (LA) mode from STB mode. The command has no effect if already running in LA mode.

LA mode is a free-running mode and the design is typically connected to a target system (an external electronic system hooked up to the emulator).

-stb

Switches to the Static Target testBench (STB) mode from LA mode. The command has no effect if already running in STB mode.

STB mode enables emulation of a design that either includes the testbench that stimulates the design or connects to an external target system. The target system must be static, which means that its clocks can be stopped and restarted without corrupting its internal state.

STB mode is the default run mode in which the system starts when using a design database that was compiled for ICE.

-staticTarget

Switches to Static Target mode (under IXCOM) from Dynamic Target mode. The command has no effect if already running in Static Target mode. Partial match is also accepted (For example, `-sta` instead of `-staticTarget`). Static Target mode is the default mode under IXCOM, so execution of the `configPM -sta` command is needed only when already running under Dynamic Target mode.

Static Target mode is the default run mode in which the system starts when using a design database that was compiled for IXCOM.

-dynamicTarget

Switches to Dynamic Target mode (under IXCOM) from Static Target mode. The command has no effect if already running in Dynamic Target mode. Partial match is also accepted (For example, `-dyn` instead of `-dynamicTarget`).

Dynamic Target mode under IXCOM is a free running mode suitable for using with a dynamic target. It is very similar to LA mode under the ICE flow, but provides in addition some features that are available only with IXCOM flow, such as support for assertions or the ability to initialize the design with the software simulator before hot swapping into the emulator.

Dynamic Target mode can be used only under the `xc on -autorun` emulation mode.

-vd

Verifies that the current run mode is VD (Vector Debug mode). Specifying this command has no other impact.

Examples

```
##### ICE Flow #####
debug .
host samurai
download
# run 100 cycles in default STB mode
sdl -load x.tdf
sdl -enable
run 100
# switch to LA mode and run
configPM -la
run -wait
# switch back to STB mode and run
configPM -stb
run

##### IXCOM Flow #####
debug .
host samurai
xc piok
# hot swap into the emulator
xc on -autorun
sdl -load x.tdf
sdl -enable
# run 100 ns in Static Target mode
run 100 ns
# switch to Dynamic Target mode and run
configPM -dyn
run -wait
# stop clocks and switch back to Static Target mode
configPM -sta
```

convertRadix

Converts a specified numerical value from one radix (binary, hexadecimal, decimal, or octal) to another radix. The command can convert values that can be represented by a maximum of 1024 bits. This command is available in all modes. Before using this command, execute the debug command. The command returns the specified numerical value after conversion to a different radix.

The syntax for this command is:

```
convertRadix <fromRadix> <toRadix> <value>
```

<fromRadix>

Specifies the original radix of the specified numerical value. The valid values for this option are:

- bin - binary
- hex - hexadecimal
- dec - decimal
- oct - octal

<toRadix>

Specifies the radix to which you want to convert the specified value. The valid values for this option are:

- bin - binary
- hex - hexadecimal
- dec - decimal
- oct - octal

<value>

Specifies the numerical value to be converted from one radix to another.

UXE Command Reference Manual

Run-time Commands

The valid values for *<fromRadix>* and *<toRadix>* are one of the strings bin, oct, dec, or hex, or any substring. Both upper case and lower case are accepted. For example, bin can be represented by any of the following strings: BIN, BI, B, bin, bi, b.

Examples

```
# Convert the binary number '1101' to its
```

```
# equivalent hexadecimal value.
```

```
XE> convertRadix b h 1101
```

```
d
```

In this example, the returned value is d, which is the hex equivalent of binary value 1101.

database

Manipulates the waveform output database. When running under IXCOM, this command has been adapted to work in both testbench partition (running in IES) and DUV partition (running in emulator hardware). In xeDebug, the waveform database format (SHM or FSDB) is specified at the UNIX command line while bringing up the tool.

Note: In Vector Debug mode, the name of the directory is automatically set according to the vector name that has been set using the [vector](#) command. Therefore, the `database` command is not required.

The syntax for this command is:

```
database [-open] <database_name> [-event] [-gzip] [-continuousupload] [-phy]
database -close <database_name>
database -show [<database_name>]
database -upload
database -clear
database -tracewindow -start <time> -end <time>
database -listphyfile [<phy_fileName> ...]
database -prepareOffline
database -disable <database_name>
database -enable <database_name>
database -captureMode [controlled | uncontrolled]
```

[-open] <database_name>

Creates a new database with the specified name. By default, the `database` command opens an SHM database, unless xeDebug was called with a `-fsdb` argument. The `-open` parameter is optional.

When running under IXCOM with both a testbench and DUV partitions, a single `database -open <database_name>` command will open a main waveform database for the testbench partition running in the simulator, and a secondary waveform database for the DUV running in the emulator.

-event

Sends every value change event for each probed object to the database, instead of sending only one event per simulation time at which multiple events occur.

When running a design that has uncontrolled clocks, the `-event` option will cause probe tracing at the fastest clock rate, such that all value changes on any signal, whether driven by controlled or uncontrolled clocks can be viewed later.

Note: For VCD format databases, this option has no effect. The simulator always dumps all events.

-gzip

Compresses the generated VCD format file and saves it with the name `<database_name>.vcd.gz`. Without the `-gzip` option, the file is saved as a regular VCD file.

Note: The `-gzip` option can be specified only when `xeDebug` was invoked with a `-vcd` argument.

-continuousupload

Starts continuous upload of a waveform database. The `-continuousupload` option automatically detects DCC overflow and pauses the design run to upload the waveform before the DCC memories overflow. So, a continuous upload is done automatically when you execute the `run` command later, or when you issue the `stop` command if running in the non-blocking mode.

The `-continuousupload` option can be abbreviated to `-cont`.

Use the `database -close` command to close the database and stop the continuous upload. If a probe does not exist, then only DCC data is uploaded without doing the FullVision computation. For best performance, remove all probes before the run, so that only DCC data is uploaded and saved to the `*.phy` file.

In ICOM mode, the `-continuousupload` option can be used across swap-in and swap-out. If you do swap-in and swap-out multiple times, a continuous and merged waveform is generated for the time window. For example, the following commands will automatically generate a continuous waveform from 0 to 4 ms:

```
database -open trace -continuousupload
probe dut.inst1
run 1ms
```

```
xc run
run 1ms
xc off
run 1ms
xc on
run 1ms
simvision -64 -snapshot <name> trace.shm
```

Note: The `-continuousupload` option is not supported with `vvmDebug`.

-phy

The `-phy` option of the `database` command causes the `-continuousupload` option to only dump the `*.phy` file, irrespective of any probes being defined or not:

```
database -open <database_name> -continuousupload [-phy]
```

Without the `-phy` option, if some probes are defined then the waveform is created; otherwise, only the `*.phy` file is created.

Note: The `-phy` option is not supported with `vvmDebug`.

-close <database_name>

Closes the currently open waveform database.

The `database -close` command has the following two usage models:

- Closes the currently open waveform database when using FSDB format waveforms.

In most debug modes, after uploading an FSDB format waveform file, the FSDB file remains open. However, some tools that process FSDB files are unable to process an already open FSDB file. Also, if you copy an open FSDB file, `<basename>.fsdb`, without copying the associated `<basename>.*` files, the copied file is considered incomplete and cannot be processed. To close the FSDB file, use either of the following methods:

 - Use the `database -close` command.
 - Open a different waveform database.
 - Exit `xeDebug` or `vvmDebug`.
- Stops upload when used with the `database -open -continuousupload` command.

UXE Command Reference Manual

Run-time Commands

The database -close command can be used in software mode. Use the database -close command with the database -open -continuousupload command to close the database and stop the continuous upload.

-show [<database_name>]

Returns information about the specified database. If the <database_name> option is not specified, information about all open databases is returned.

-upload

Uploads the trace results from the emulator into the SHM or FSDB waveform database, so that it can be used by the waveform browser. When using FullVision mode, it also creates a *.phy file, which can be used later in offline mode.

Note: The database -upload command can be used to upload all probed CPF or IEEE 1801 objects and their transitions into the SHM or FSDB waveform database.

-clear

Causes the previously captured trace data to be removed from the waveform database on the next trace data upload. The effect on the waveform display is that all previous data is cleared from the screen. Note that when running in LA mode, this command option is ignored because the traced data is cleared from the waveform database by default before each new upload.

The database -clear command clears only trace files, but not the emulation cycle count.

-tracewindow -start <time> -end <time>

Specifies a smaller time range for generating a waveform. This command option must be specified before the database -upload command. You can also specify a pre-defined bookmark name instead of the time range.

For example,

```
-tracewindow -start <bm1> -end <bm2>
```

where *bm1* and *bm2* are pre-defined bookmarks for specific time points.

-listphyfile [<phy_fileName> ...]

Returns the cycle range of all .phy files. If the <phy_fileName> option is given, the cycle range is returned only for the specified .phy file(s).

The database -listphyfile command can be executed after the debug command.

For example:

- If a trace.phy file contains cycles 0–999, the database -listphyfile command will return the following string:

```
{trace.phy interval: [0:999]}
```

- To get the cycle range for trace1.phy and trace2.phy files, execute the following command:

```
database -listphyfile trace1.phy trace2.phy
```

Note: This option is not supported with vvmDebug.

-prepareOffline

Uploads the trace results from the emulator into a *.phy file, which can be used later in offline mode, but cannot be read directly by the waveform browsers. Since this command does not process the data into SHM or FSDB format, it is faster than the -upload option. Use this option in online mode. This option is supported in the FullVision mode, but not in DYNP (Dynamic Probes) mode. This option is not supported with vvmDebug.



Prior to using the offline mode, either the database -prepareoffline command or the database -upload command must be executed at least once.

-disable <database_name>

The database -disable command causes the waveform capture to stop until it is enabled again by the database -enable <database_name> command.

Note: This command is not supported with vvmDebug.

-enable <database_name>

The database -enable command causes the waveform capture to resume if it was previously stopped by the database -disable <database_name> command.

Note: This command is not supported with vvmDebug.

-captureMode [controlled | uncontrolled]

The `-captureMode` option of the `database` command controls how waveform display behaves with a design having uncontrolled clocks:

```
database -open <name> -captureMode [controlled | uncontrolled]
```

The `-captureMode` option can only be used with the `-open` option. It is not possible to change capture mode on a database that is already open.

This option can only be specified while the design is already swapped-in to the hardware. It is applicable only for designs with an uncontrolled clock domain.

Note: This option is not supported with vvmDebug.

The default (if `-captureMode` not specified) is controlled.

If the `-captureMode` option is set to controlled, the time stamp of the software simulator is used in the waveform.

If the `-captureMode` option is set to uncontrolled, the time stamp of the software simulator is ignored, and a nominal interval is used for each waveform sample. The probed data from the NC-sim simulator and the data from the Palladium XP emulator do not merge in the waveform browser. This is because both use a different time scaling, and there is no way to align the transitions correctly along the time axis.

If the `-captureMode` option is set to uncontrolled, or the database was opened with the `-event` option, the data captured into DCC always uses the uncontrolled clock, with two samples per FCLK in 1X mode, or one sample per FCLK in 2x and higher sampling modes.

If the `-captureMode` option is set to controlled, the clock used for capturing depends on the vision mode as follows:

DYNP mode and `-captureMode` set to controlled:

Data capture into DCC during run time uses the controlled clock, meaning one sample per simulator time slot.

FV mode and `-captureMode` set to controlled:

Each simulator time slot causes one full FV frame (typically 32 samples) to be captured. The frame contains the consecutive samples taken by the uncontrolled clock. This might have adverse effect on maximal trace depth.

UXE Command Reference Manual

Run-time Commands

When you use the `-captureMode uncontrolled` option, the waveform time cannot equal simulation time because during simulation, controlled clocks might be stopped to handle `tbcall` statements, whereas uncontrolled clocks would continue to run.

The following commands use simulation time:

- `getTime`: returns simulation time
- `run <interval>`: runs an amount of simulation time

The following commands and features use waveform time, not simulation time:

- The waveform display
- The `xeDebug Current Cycle` and `Current Time`
- `xeset traceMemSize <number>ns`
- SDL Trace
- SDL Trigger Position

debug

Initializes the Debugger and specifies the directory path to the design database. This command is available in all modes.

The `debug` command is a prerequisite for all other debugger commands. This is because according to the design type, the `debug` command sets up a debug session and calls the set of relevant XEL commands into `xeDebug`. The XEL commands set called by the `debug` command differ for the SA and ICE designs.

This command is also used to switch between online and offline mode, or to switch using different emulator, without quitting the `xeDebug` session.

With a few exceptions (for example, the path to the design database), you should not expect the `debug` command to reset the program settings to their default values. Program settings might include global TCL variables as well as other settings that are maintained through XEL commands (most of them but not all, through the `xeset` command). This means that if you open a session with the `debug` command, then close it, and then open a second session, most of the settings from the first session will be carried over to the second session.

The syntax for this command is:

```
debug <dbPath> [-session <session_name>]  
debug -close
```

<dbPath>

Specifies the path to the design directory. This command creates symbolic links to the files that are located under the specified `<dbPath>`. Instead of specifying the full path, a dot (.) as an argument in the subsequent `debug` commands is also allowed.

There is a default set of files and directories for which symbolic links are created. However, an additional set of file or directory names can be specified to be linked by the `debug` command using the `linkFiles` run-time parameter through the `xeset` command.

The path specification using a dot works properly if you are working on the host workstation attached to the emulator. If you are not working on the host workstation, you must use a network wide path that is recognized also by the emulator's workstation host, such as:

```
debug ~joe/asic/design1
```

or

```
debug /net/dogbert/work/asic/design1
```

UXE Command Reference Manual

Run-time Commands

The debugger will try to convert a relative path to an absolute path, but this is not always guaranteed to be recognized by a remote host. Different UNIX shells, NFS mount points, presence of symbolic links in the path, and automount can affect the way pathnames are treated.

```
# Initialize the debugger in command line mode
% xeDebug
# Specify the path to the design database directory that is, '/mydesign/design1'.
XE> debug /mydesign/design1
```

-session <session_name>

Creates a user directory with the name as *<session_name>.session* and makes it the current directory so that all the subsequent logs, for example, *xe.msg* and waveforms are stored in the new directory. This command is typically useful if while running a batch script, different results need to be stored in different directories.

For example, if a directory named *a1.session* is created in the current design directory by using the following command, all the subsequent log files will be stored in this directory:

```
debug . -session a1
```

-close

Closes the current debug session. You must close your session before opening another session, to separate the logging of different sessions into the correct log files.

deposit

Sets the state of a signal, symbol, or bus to the specified value. The `deposit` command can be applied to a signal if it is specified as a `keepNet` during compile.

In ICE mode, you should download the design before executing this command.

In Static TestBench (STB), or Vector Debug mode; or if the emulation clocks are stopped, the value deposited for the specified signal is held for only one clock cycle. Also, the previously deposited value is overwritten during the emulation run.

In Logic Analyzer (LA) mode, `deposit` is equivalent to the `force` command followed by the `release` command. Therefore, if the `deposit` command is executed while the clocks are still running, the deposited value might be held for several thousands of clock cycles. When depositing to a bus or a group of signals, the deposit operation into each individual signal is done at a different time. Therefore, time coherency between deposited values of the different signals is not maintained.

In STB, LA and VD modes, a `deposit` command on a net that was previously forced will be either ignored or performed depending on the value of the `depositNCsimMode` run-time parameter.

The `-deposit` command is not supported with `vvmDebug`.

If the `depositNCsimMode` run-time parameter is set to 1, the `deposit` command on the net is ignored if the net was previously forced and the `deposit` command does not include the `-release` option.

If the `depositNCsimMode` run-time parameter is set to 0, the `deposit` command sets a new value to the forced net, and the state of the net is changed to released.

Note: In the ICE mode, if the `deposit` command needs to be executed on a large set of signals, the access time can be significantly improved by grouping the signals with the `symbol` command, and using that symbol in the `deposit` command, instead of separately executing the `deposit` command on each signal. When signals are grouped in such a way, a more efficient mechanism is used to access the hardware.

In the SA mode, a `deposit` command on a net that was previously forced will be ignored (unless the `deposit` command has the `-release` option). Also, note that in the SA mode the `deposit` command does not operate on the `symbol` command.

For most types of flip-flop and latch cells, setting the signal connected to the Q output of the cell also sets the internal state of the cell at the next FCLK cycle. However, setting the Q

UXE Command Reference Manual

Run-time Commands

output of the cells listed below does not set the internal state. In most cases, this is because the Q output is separated from an internal flip-flop or latch by a combinational gate.

Q_FDN31	Q_FDP0LS	Q_FDP0W2V1
Q_FDP0XPW2	Q_FDP2LS	Q_FDP31
Q_FDP31XP	Q_FDP61	Q_FDP61XP
Q_FDP61XPEN	Q_FJP61	Q_FJP61XN
Q_FJP61XNEN	Q_FSP61	Q_FTN31
Q_FTP31	Q_FTP61	Q_FTP61EP
Q_LDN0V1	Q_LDN0ZP	Q_LDN1ZP
Q_LDN2ZP	Q_LDN31	Q_LDN3S
Q_LDN3SZP	Q_LDN4V1	Q_LDP0FDMX
Q_LDP0W2	Q_LDP31	Q_LDP3S
Q_LDP4FDMX	Q_LMP0V1	Q_LMP0W2
Q_LMP0W2V1	Q_LSN01	Q_LSN013Y
Q_LSN31V1	Q_LSP01G	

The syntax for this command is:

```
deposit <name> <value> [-release]
deposit -allff {0 | 1 | random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]
deposit -file <textfile.dat>
```

<name>

Specifies the name of either a signal or a symbol. A signal name representing a vector (bus) might appear with part-select notation (for example, xyz [7 : 4]), or without, if all its bits are to be deposited.

<value>

Specifies the value to be deposited to the specified signal or symbol. When the signal or symbol represents a single bit, *<value>* is either 0 or 1. When depositing to a vector or a

symbol that represents several bits, the value of a radix might be specified in binary, octal, decimal, or hexadecimal format, prefixed with the appropriate letter. For example, '`b011`', '`o1237`', '`d089`', '`hffcd`'. If not specified, the default radix is decimal.

-release

In both SA flow and STB, LA, and VD modes, releases an active force and enables you to deposit a specified value using the `deposit` command.

Note: In the STB, LA, and VD modes, without this option and the `depositNCsimMode` run-time parameter set as 1, the `deposit` command on the forced signal is ignored.

-allff {0 | 1 | random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]

Sets the state of most flip-flops and latches in the design. This option is supported only in ICE mode and not in SA mode.

The following flip-flops and latches are not set or forced:

- Any flip-flop or latch whose output is not declared as a `keepNet`, and whose value is a constant that can be determined at compile time. (Precompile removes such flip-flops and latches, and ties their outputs.)
- Any flip-flop or latch whose output is declared as a `tieNet`.
- Any flip-flop or latch whose output is currently forced using the signal `-force` command option.
- Any instance of a delay cell (`Q_FDP0B`)
- Any flip-flop or latch that is initialized in an `initial` statement, if the `hdlImport -atb` command option is used. This exception might be removed in a future release.

If a memory is synthesized to flip-flops and latches, the resulting flip-flops and latches are set by this command.

If the `-instance` option is specified, only the flip-flops belonging to the `<instance_name>` are set or forced. The flip-flops belonging to the instance include all the nodes reachable in the hierarchical tree with root `<instance_name>`.

If the `-exclude` option is specified, all flip-flops in the design except the instance belonging to the `<instance_name>` are set or forced.

UXE Command Reference Manual

Run-time Commands

The `random [-seed <number>]` option sets or forces the state of each flip-flop or latch randomly to either 0 or 1. The `<number>` can be any valid integer. If the same seed number is used for different runs, the initialization results will be the same. If the `-seed <number>` is omitted, the initialization results will change with each run.

-file <textfile.dat>

Sets the values of multiple signals simultaneously by reading the signals from a specified binary (.dat) file, which is generated as a result of the `force -file -prep <textfile>` command.

download

Downloads the current design to the emulation hardware. This command is available if the design database was compiled for ICE or VD modes. Execute the [host](#) command before using this command. The `download` command is not supported with vvmDebug.

In IXCOM mode, this command is executed implicitly by the software during swap-in (when switching from pure simulation to SA mode).

When the `download` command is executed, the software checks whether a file named `.rtxerc` exists either in the home directory or in the directory from which the `xeDebug` executable was invoked. If yes, this file is executed after successful conclusion of the `download` command. If both directories include a `.rtxerc` file, both files are executed, starting from the home directory.

The syntax for this command is:

```
download [-skiphddc all | -skiphddc <cable_logical_name>...]
```

[-skiphddc all | -skiphddc <cable_logical_name>...]

With the `-skiphddc` option, the design is downloaded, but the download of specified cables, or all cables, is skipped. `<cable_logical_name>...` refers to the logical names of one or more cables to be skipped (for example, TesterPCI). Use the keyword `all` to skip all of the design's HDDC cables. Each `<cable_logical_name>` must be the same as the `<interface_name>` in a `cableConnection` command, [`cableConnection -add {<interface_name>.<interface_type> <conn_loc> \[<I/O_technology>\]}`](#).

Use the `-skiphddc` option to download a design compiled with target cables, to an emulator or domain that lacks some of the cables. If you skip downloading a cable, the pins of that cable do not drive the target, and the behavior of any design inputs or bidirectional signals assigned to the cable is unspecified.

Examples

```
download -skiphddc <cable_logical_name>
```

Or,

```
download -skiphddc all
```

dpa

Collects toggle count data and generates TCF/SAIF files for dynamic power analysis.

The `dpa` command is available in the Logic Analyzer, STB, and SA modes.

Without parameters, the `dpa` command returns a list of all signals currently selected for tracing. The `dpa` command is available in FV mode and not DYNP mode.

This command should be executed after opening either online or offline session. However, Cadence recommends you use the `dpa` command in an offline session to avoid keeping the emulator hardware occupied while doing the DPA operations.

The syntax for this command is:

```
dpa [options]
```

The following command options enable you to start using the DPA features:

```
dpa -phyfile <phy filename> |
dpa -listphyfile [<phy fileName> ...] |
dpa -tracewindow [-start <time> {-end <time>} | -size <time>)]
```

The following command options handle the Toggle Count data for direct use and display by the xeDebug tool:

```
dpa -setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>] <instance names> |
dpa -setupToggleCount -listinst [-verbose] |
dpa -setupToggleCount -rminst <instance names> |
dpa -setupToggleCount -rminst -all |
dpa -getToggleCount [-append] [-weighted] [-file <name>]
dpa -list [-max <n>]
```

The following command options handle TCF files (which are processed by RC) and SAIF files (which are processed by third-party tools):

```
dpa -addinst [-excludelibrary <.llib names>] [-allnets] -depth <n> <instance> ... |
dpa -addinstoutputpin <instance> [-gatecell <gatecell filename>] |
dpa -addinstpin <instance> |
dpa -addcone -depth <n> <signal> ... |
dpa -addpath -depth <n> {<net1> <net2>} |
dpa -readNetNames <rc dpa info file name> [<instanceName>] |
```

UXE Command Reference Manual

Run-time Commands

```
dpa -rm <signal>... |
dpa -rm [-regexp | -glob | -pattern <pattern>]* |
dpa -outfile -tcf <filename> [-instance <instanceName>] [-masterTCF] [-slaveTCF] [-noregulartcf] [-segment <n>% | <cycleNumber>] [{-start <time> -end <time>}] |
dpa -upload |
dpa -saif -load <saif_file_name> [-module <module_name_pattern>] |
dpa -saif -listmodule [-lib] |
dpa -saif -listlib |
dpa -saif -listfile |
dpa -saif -rmmodule <module_name_pattern> [<lib_name>] |
dpa -saif -listduplicatemodule |
dpa -saif -rmlib <lib_name> |
dpa -saif -rmfile <file_name> |
dpa -saif -addscope <instance_name> |
dpa -saif -rmscope <instance_name> |
dpa -saif -listscope |
dpa -outfile -saif <saif_file_name> [-instance <instanceName>] [-segment <cycleNumber> | <percent>] [{-start <time> -end <time>}]
```

The following command options are used to generate TCF files, power report, and power profile by invoking RTL Compiler and genPowerProfile program directly from the xeDebug tool:

```
dpa -genPowerProfile [-refine] |
dpa -setupRC -show |
dpa -setupRC -reset |
dpa -setupRC { -gateFlow | -rtlFlow } |
dpa -setupRC -servers { <machine_list> } |
dpa -setupRC -libPath { <path_name_list> } |
dpa -setupRC -libList { <lib_name_list> } |
dpa -setupRC -designPath { <path_name_list> } |
dpa -setupRC -designFile { <file_name_list> } |
dpa -setupRC -topDesign <top_module_name> |
dpa -setupRC -instance <instance_name> |
dpa -setupRC -depth <N> |
```

```
dpa -setupPowerProfile -show |  
dpa -setupPowerProfile -reset |  
dpa -setupPowerProfile -database <name> |  
dpa -setupPowerProfile -tcfDir <name> |  
dpa -setupPowerProfile -segment {<n>% | <cycleNumber>}
```

The following command option enables using of sparse sampling for DPA:

```
dpa -sparse sampling [<N>]
```

-phyfile <phy_filename>

Specifies the * .phy file that will be used for DPA operations. This command must be run before running the other DPA related commands, such as `dpa -outfile -tcf` command.

Note: This option is not supported with vvmDebug.

When the `dpa -phyfile` command is executed without arguments, it returns the name of the currently selected * .phy.

To use DPA feature, you must first specify a .phy file using the `dpa -phyfile` command. In offline mode, after opening the offline session, the .phy file for DPA is set to the same file as specified using the host `-offline <phyfile>` command. During the offline session, you can use `dpa -phyfile` command to switch to a different .phy file. If you want to use DPA in online session, they must explicitly specify a .phy file by using the `dpa -phyfile` command.



If .phy file gets changed after issuing the `dpa -phyfile` command, for example, more cycles are uploaded into the .phy file, the `dpa` command will not be able to see the new data. You need to issue the `dpa -phyfile` command again to update the .phy file information.

-listphyfile [<phy_fileName> ...]

Returns the cycle range of all .phy files. If the `<phy_fileName>` option is given, the cycle range is returned only for the specified .phy file(s).

The `dpa -listphyfile` command can be executed after the `debug` command.

For example:

- If a `trace.phy` file contains cycles 0-999, the `dpa -listphyfile` command will return the following:

```
{trace.phy interval: [0:999]}
```

- To get the cycle range for `trace1.phy` and `trace2.phy` files, execute the following command:

```
dpa -listphyfile trace1.phy trace2.phy
```

Note: This `-listphyfile` option is not supported with `vvmDebug`.

-tracewindow [-start <time> {-end <time> | -size <time>}]

Specifies a smaller time range for DPA operations. This command should be specified before executing the following two commands:

- `dpa -upload`
- `dpa -getToggleCount`

Note: The `-tracewindow` option can also be specified as `-tw`.

If the `-tracewindow` option is not executed, the DPA operation is performed for the entire time range of `*.phy`.

Important

This command can only be used in offline debug session. It helps to speed up the process of generating TCF/SAIF files by ignoring the data outside the specified window.

The trace window can be specified either by a start and end time, or by a start time and window size. You can also specify a pre-defined bookmark name instead of the time range.

For example,

```
-tracewindow start <bm1> -end <bm2>
```

where `bm1` and `bm2` are pre-defined bookmarks for specific time points.

When the `dpa -tracewindow` command is executed without arguments, the time range specified within the currently selected `*.phy` file is returned.

**-setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>]
<instance_names>**

Specifies the list of module instance names for which toggle count data is to be collected.

Instance name "." indicates top module of the design. Sub-instances are included based on the following options:

- **-top <n>:** Adds only the top <n> largest instances based on the number of nets. The default setting is top 1000.
- **-depth <n>:** Adds only those child instances that are within <n> levels below the root instance. The default depth setting is 0, which means that all children at all levels are added.
- **-min <n>:** Adds only those child instances whose size (number of nets contained) is larger than <n> percent of the specified instance. The default setting is 1 percent.

Note: If this command is not used before the `dpa -getToggleCount` command, data collection takes place for a default set of modules. This default set includes the top 1000 largest instances from all levels of hierarchy in the design, with a size more than 1% of the complete design. In most cases, this module set is adequate for a meaningful toggle count data analysis.

-setupToggleCount -listinst [-verbose]

Lists all the module instances that are currently set for toggle count data collection. If the `-verbose` option is specified, the size of each instance is also listed.

-setupToggleCount -rminst <instance_names>

Removes the specified module instances and their children from the list of modules set for toggle count data collection.

-setupToggleCount -rminst -all

Removes all the module instances and their children from the list of modules set for toggle count data collection.

-getToggleCount [-append] [-weighted] [-file <name>]

Computes toggle count or weighted toggle count. The following options are used to control the toggle count data calculation:

- **-append**: Appends the new data to the existing data file. By default, the new data overwrites the existing data in the data file.
- **-weighted**: Captures weighted toggle count data instead of normal toggle count data. By default, normal toggle count data is computed.
- **-file <name>**: Specifies the base name of the data file in which the toggle count data is written. The data file is created with the *.ppfdata* extension. If no file name is specified, the current waveform name, which is set by the *dpa -outfile* command, is used as the base name.

-list [-max <n>]

Prints all nets which are to be used during the *dpa -upload* command for creating the TCF or SAIF files. The **-max** option specifies the maximum number of nets to be printed.

-addinst [-excludelibrary <.l1lib names>] [-allnets] -depth <n> <instance> ...

Adds all the nets in the database hierarchy of the given instances to the trace signal list.

The **-allnets** option is used to add nets starting with *n#*, which are otherwise by default filtered and not added to the database hierarchy.

The **-excludelibrary** option is used to exclude all internal nets in the modules that are defined in the specified library. The name of the library is the name of the *l1lib* that you specified during compilation. You can execute the *ls QTDB* command to see the correct *l1lib* names. If you specify two or more library names you need to enclose all the library names with double quotes or curly braces.

The **-depth** option is used to specify a maximum depth to restrict the search in the database, where *<n>* specifies the maximum levels of hierarchy. By default one hierarchy level is searched. A depth of zero means the whole hierarchy of an instance is searched.

-addinstoutputpin <instance> [-gatecell <gatecell_filename>]

Adds all output pins of the cells and hardmacros within the specified *<instance>*. The cells and hardmacros are determined by a gate cell list. This gate cell list can be generated by RC.

The default gate cell list is saved in the dbFiles/gateCell file. You can specify a different file with the -gatecell option.

Note: The internal nets that are not useful for dynamic power analysis are not included.

-addinstpin <instance>

Adds all input/outputs of the specified <instance> and all instances within it.

-addcone -depth <n> <signal> ...

Adds all signals in the input cone of logic of the specified nets to the DPA trace signal list.

-addpath -depth <n> {<net1> <net2>}

Traces back from <net1> to <net2> and adds all nets on the path. The -depth <n> parameter indicates the number of levels that are traced back. The default value is 10 levels.

-readNetNames <rc_dpa_info_file_name> [<instanceName>]

Adds the nets and pins from the specified <rc_dpa_info_file_name> into the probe list. Also specify the <instanceName> argument if the required nets and pins are not under the top cell.

-rm <signal>...

Removes the specified signal(s) from the trace signal list. The -rm * command removes all signals from the trace signal list.

-rm [-regexp | -glob | -pattern <pattern>]*

Removes signals with names that match the specified pattern.

UXE Command Reference Manual

Run-time Commands

```
-outfile -tcf <filename> [-instance <instanceName>] [-masterTCF]
[-slaveTCF] [-noregulartcf] [-segment <n>% | <cycleNumber>]
[{-start <time> -end <time>}]
```

Specifies the format of the TCF file, which is used by RC to calculate the power consumption. The `-tcf <filename>` option specifies the name of the TCF file.

By default, a design-based TCF file is generated, which uses the full hierarchical name of the pins and nets. To generate a module-based TCF file, use the `-instance <instanceName>` option. A module-based TCF file uses the relative name of the pins and nets, which are valid only within the specified module instance.

The `-masterTCF` and `-slaveTCF` options are used to generate different types of TCF files. Following are the three types of TCF files that can be generated using the `dpa -outfile` command:

- Regular TCF file

This is the default format of the generated TCF file. A regular TCF file stores both the hierarchical names and toggle counts of the pins.

- Master TCF file

This file is generated when the `-masterTCF` option is used. A master TCF file stores only the hierarchical names of the pins.

- Slave TCF file

This file is generated when the `-slaveTCF` option is used. A slave TCF file stores only the toggle counts of the pins.

The regular TCF file is always generated by default, regardless of whether you use or do not use the `-masterTCF` and `-slaveTCF` options. If you do not want to generate the regular TCF file, use the `-noregulartcf` option.

Note: The master and slave TCF files use less storage space and provide faster reading and writing speed, as compared to regular TCF files. Therefore, use master or slave TCF file if you want to calculate power consumption values over multiple time periods. Use regular TCF file to calculate the power consumption value for a single time period.

The `-segment` option specifies the number of segment windows for which toggle counts are calculated. Each TCF file covers a time period, which is called a segment window. For a given time range in .phy file, you can generate one slave TCF file in one segment window, or many slave TCF files in multiple adjacent segment windows. The `<n>%` parameter specifies the percentage of the total analysis time period that is to be covered in one segment window. The

UXE Command Reference Manual

Run-time Commands

<cycleNumber> parameter specifies number of emulation cycles to be covered in one segment window.

For example, if the .phy file covers 100K period, use the following command in the offline mode to generate one TCF for 100K period:

```
dpa -outfile -tcf <filename> -segment 100%
dpa -outfile -tcf <filename> -segment 100000
```

In the same example, use the following command to generate a master TCF file and 10 slave TCF files (one slave file per 10K segment):

```
dpa -outfile -tcf <filename> -mastertcf -slavetcf -noregulartcf -segment 10%
```

The above command with the -segment option generates the following:

- A master TCF file and 10 slave TCF files.
- A master TCF log file and 10 slave TCF log files that records the clock frequency and segment activity statistical information in text format.
- A TCF segment log file that records segment start and end time information in text format.

A .phy file collects design activities during a specific time window. To narrow down the time window for the TCF generation process, use the {-start <time> -end <time>} option as following:

```
dpa -outfile -tcf <filename> -start <time> -end <time>
```

You can also specify a pre-defined bookmark name instead of the time range.

For example,

```
dpa -outfile -tcf mytcf start bm1 end bm2
```

where *bm1* and *bm2* are pre-defined bookmarks for specific time points.

Note: When the design has CPF information, more slave TCF files might be generated than what is specified using the -segment option. The UXE software will create a new segment, that is, a new slave TCF file whenever there is a change in the power domain status, for example, changed from power on to power off. This is to get more accurate results for power estimation. When a power domain is turned off, its power consumption is close to zero. For example, if you do not turn off a power domain, one slave TCF might show the power domain as powered-on for one half of the time and powered-off for the remaining half. It is very obvious that you will have much more accurate power estimation if you split a TCF file into two.

-upload

Creates the TCF or SAIF file.

-saif -load <saif_file_name> [-module <module_name_pattern>]

Loads the SAIF file. At a time, more than one SAIF file can be loaded.

A name pattern can be specified by using the `-module` option. Multiple `-module` options can be used to specify multiple patterns. If module name patterns are specified, the modules with names that do not match the specified patterns are skipped.

-saif -listmodule [-lib]

Lists all modules that have been loaded from the SAIF file.

The library forward SAIF file contains the module definitions. Each SAIF file has an associated library name. If the `-lib` option is specified, library names are also listed with the module names.

-saif -listlib

Lists all library names that have been loaded from the SAIF file.

-saif -listfile

Lists all file names that have been loaded.

-saif -rmmodule <module_name_pattern> [<lib_name>]

Removes all modules with names matching the specified `<module_name_pattern>`. Some modules might exist in multiple libraries. In that case, specify the library name, `<lib_name>`, to resolve the ambiguity.

-saif -listduplicatemodule

Lists the module names and library names for all modules that appear in multiple libraries.

-saif -rmlib <lib_name>

Removes all modules from the specified library.

-saif -rmfile <file_name>

Removes all modules that are defined in the specified SAIF file.

-saif -addscope <instance_name>

Adds to the scope the specified instance from the forward library SAIF file. The scope identifies the instances that need to be included in the backward SAIF file generated from the forward library SAIF file.

-saif -rmscope <instance_name>

Removes the specified instance from the scope.

-saif -listscope

Lists the instances that are a part of the current scope.

**-outfile -saif <saif_file_name> [-instance <instanceName>]
[-segment <cycleNumber> | <percent>] [{-start <time> -end <time>}]**

Sets the parameters for generating backward SAIF file.

The `-saif <saif_file_name>` option specifies the base name of the backward SAIF file to be generated.

By default, a design-based SAIF file is generated, which uses the full hierarchical name of the pins and nets. To generate a module-based SAIF file, use the `-instance <instanceName>` option. A module-based SAIF file uses the relative name of the pins and nets, which are valid only within the specified module instance.

The `-segment` option specifies the number of segment windows for which toggle counts are calculated. Each SAIF file covers a time period, which is called a segment window. For a given time range, one SAIF file can be generated in one segment window, or many SAIF files in multiple adjacent segment windows. The `<cycleNumber>` parameter specifies number

UXE Command Reference Manual

Run-time Commands

of cycles to be covered in one segment window. The *<percent>* parameter specifies the percentage of the total analysis time period that is to be covered in one segment window.

The design activities are collected during a specific time window. To narrow down the time window for the SAIF generation process, use the { -start <time> -end <time>} option as following:

```
dpa -outfile -saif <filename> -start <time> -end <time>
```

You can also specify a pre-defined bookmark name instead of the time range.

For example,

```
dpa -outfile -saif mysaif start bm1 end bm2
```

where *bm1* and *bm2* are pre-defined bookmarks for specific time points.

-genPowerProfile [-refine]

Generates a power profile based on the parameters set using the `-setupPowerProfile` and `-setupRC` options of the `dpa` command.

If the `-refine` option is specified, refinement is done for the power consumption data reported last time. Refinement is used to get more accurate power consumption data of a time range. The rough data is updated with more accurate data during the refinement process. This option should be explicitly specified if the existing power consumption data needs to be refined.

Note: The time range specified to be refined must be less than the previous time range. The design file and the `.phy` file must be the same as last run. To do refinement, only the `-tracewindow [-start <time> {-end <time> | -size <time>}]` and `-setupPowerProfile -segment {<n>% | <cycleNumber>}` options of the `dpa` command should be changed comparing to the last run.

To ensure the power profile waveform is in sync with the signal waveform, you need to set the same time unit as the signal waveform using the `xeset timeUnit` command before you invoke the `dpa -genPowerProfile` command. In ICE mode, if the same time unit is not set, the power profile is not aligned with the signal waveform. The default time unit is `fclk`. In SA mode, the tool gets the correct time unit automatically.

-setupRC -show

Shows the current setting of all parameters set for the `-setupRC` command.

-setupRC -reset

Resets all the RTL Compiler parameter settings to default values.

-setupRC { -gateFlow | -rtlFlow }

Specifies whether the RTL Compiler must be invoked for gate-level netlist flow or RTL design flow. By default, the RTL Compiler is invoked for gate-level netlist flow.

-setupRC -servers { <machine_list> }

Indicates whether the SuperThreading feature of RTL Compiler should be used or not. By default, this feature is not used. This is an optional parameter.

Enabling the SuperThreading feature enables RTL Compiler to compute multiple power report in parallel. To use this feature, you need to acquire a special license.

Note: Multiple machines can be specified with space as the separator.

For example, to enable SuperThreading feature on servers `srv1`, `srv2`, `srv4`, and `srv6`, execute the following command:

```
dpa -setupRC -servers {srv1 srv2 srv4 srv6}
```

Note: This option is not supported in RTL flow of power analysis.

-setupRC -libPath { <path_name_list> }

Sets the path that should be used to search technology libraries while invoking the RTL Compiler. This option should be used with the `-setupRC -libList` option. This is an optional parameter.

For example, if the technology libraries, `tutorial.lib` and `RAM.lib`, are available in the directory named `lib` within the current design directory, specify the following command:

```
dpa -setupRC -libPath {./lib} -libList {tutorial.lib RAM.lib}
```

-setupRC -libList { <lib_name_list> }

Sets the target technology libraries used by RTL Compiler.

If the `-setupRC -libPath` option is not specified, full path to the target technology libraries is required. Otherwise, specify the library name included in the previously defined `-setupRC -libPath` option. This is a mandatory parameter.

For example, to set all technology libraries available in the `lib` directory within the current design directory, specify the following command:

```
dpa -setupRC -libList {./lib/*.lib}
```

-setupRC -designPath { <path_name_list> }

Sets the path that will be used to search the design file. This option should be used along with the `-setupRC -designFile` option. This is an optional parameter.

-setupRC -designFile { <file_name_list> }

Defines the design files to be used for generating TCF files.

If the `-setupRC -gateFlow` option is set, specify the gate-level netlist file to be used. If the `-setupRC -rtlFlow` option is set, specify the RTL design files to be used.

If the `-setupRC -designPath` option is not specified, the full path to access the design files is required with respect to the current working directory. Else, specify the design file name included in the previously defined the `-setupRC -designPath` option. This is a mandatory parameter.

-setupRC -topDesign <top_module_name>

Defines the top module name that is then used by the RTL Compiler for processing. The `<top_module_name>` is an optional parameter in the ICE mode, and a mandatory parameter in the SA mode. Only the name of the top level module is needed in this command.

-setupRC -instance <instance_name>

Defines the name of the instance for which power calculations need to be done.

The specified instance name is used by the RTL Compiler to create probe nets for TCF files generation. This is a mandatory parameter.

You must specify the full hierarchical path to the instance because the RTL compiler needs the complete path to create probes.

-setupRC -depth <N>

Sets the number of hierarchy levels to descend in the power report data. This is an optional parameter. The default value is 10.

-setupPowerProfile -show

Shows the current setting of all parameters set for the `-setupPowerProfile` command.

-setupPowerProfile -reset

Resets all existing power profile settings to default values. You can use the `dpa -setupPowerProfile -reset` command to clear the existing settings.

The `dpa` commands that are given after the `dpa -setupPowerProfile -reset` command can incrementally setup power profile parameters. You can either setup the power profile parameters one by one or combine more parameters into a single `dpa` command.

-setupPowerProfile -database <name>

Specifies the name of the power profile. The generated power profile file will be a waveform file of SST2 format.

If the `<name>` parameter contains a directory name in which the power profile file should be saved, the directory is created automatically. This is a mandatory parameter.

-setupPowerProfile -tcfDir <name>

Specifies the subdirectory name in which the generated TCF files are saved. This is an optional command.

The default names for TCF files are derived from the power profile name.

For example, if the power profile name is “`test1`”, then the TCF directory name will be “`./tcf`”. If the power profile name contains a directory like “`test1/topInst`”, then the TCF directory will be “`test1/tcf`”. You can also use the `-tcfDir` option to specify the expected directory to save the TCF files.

Note: The TCF files are generated when the `dpa -genPowerProfile` command is executed to initiate the power profile computation process. There are multiple TCF files, each of which corresponds to a time segment.

-setupPowerProfile -segment {<n>% | <cycleNumber>}

Identifies the number of segment windows for which toggle counts need to be calculated.

The *<n>%* parameter specifies the percentage of the total analysis time period that is to be covered in one segment window.

The *<cycleNumber>* parameter specifies the number of emulation cycles to be covered in one segment window.



Each TCF file covers a time period, which is called a segment window. If the `-setupPowerProfile -segment` command is not specified, the whole time range in the `.phy` file will be computed as one segment window. The default value is 100%.

-sparse_sampling [<N>]

Enables you to define the sparse sampling rate to *<N>*. The DCC capture is based on frames. Each frame typically has 32 cycles. When sparse sampling is enabled, it captures one frame out of the specified *<N>* frames. This applies to both ICE and SA modes. Sparse sampling is used for Dynamic Power Analysis (DPA). The result of DPA might not be accurate with sparse sampling because the power analysis is based on $1/N$ of design activity. However, it can speed up the computation by *N* times.

N must be an integer larger or equal to 0. If *N* is specified as 0 or 1, sparse sampling is disabled. If *N* is not specified, the command returns the current sparse sampling rate.

The `dpa -sparse_sampling` command is the only `dpa` command that must be issued in an online session. This command option is not supported with `vvmDebug`.

drtl

This command provides a single interface for XEL to access the Dynamic RTL (DRTL) features. DRTL enables you to create control and monitoring programs in Verilog, VHDL, and SystemVerilog at run time. The DRTL modules can be used independently or their outputs can be referenced from SDL programs.

For details on how to use DRTL, refer to the *Controlling the Run with DRTL* chapter in the *UXE User Guide*.

The syntax for this command is:

```
drtl -definemodule <DRTL_module_name> [-verilog | -vhdl | -sv] <file_list>  
[ -topmodule <topmodule_name> ]  
  
drtl -addinst [-overwrite] <DRTL_module_name> <DRTL_instance_name>  
(.<port_name1>(<signal_name1>), .<port_name2>(<signal_name2>), ...)  
  
drtl -addinst [-overwrite] <DRTL_module_name> <DRTL_instance_name>  
(<signal_name1>, <signal_name2>, ...)  
  
drtl -list [-code] [DRTL_module_name_pattern]  
  
drtl -listinst [DRTL_instance_name_pattern] [-modulename <DRTL_module_name>]  
  
drtl -rm <DRTL_module_name_pattern>  
  
drtl -rminst <DRTL_instance_name_pattern> [-modulename <DRTL_module_name>]  
  
drtl -getallinst [DRTL_instance_name_pattern] [-modulename <DRTL_module_name>]  
  
drtl -getallmodule [DRTL_module_name_pattern]  
  
drtl -enableSysTask | -disableSysTask <DRTL_instance_name_pattern>  
  
drtl -systemTasksDefault [enable | disable | 0 | 1]  
  
drtl -save [-overwrite] <file_name> [DRTL_module_name_pattern_list]  
  
drtl -load [-overwrite | -ignore] <file_name> [DRTL_module_name_pattern_list]
```

-definemodule <DRTL_module_name> [-verilog | -vhdl | -sv] <file_list> [-topmodule <topmodule_name>]

Creates a DRTL module from the specified Verilog, VHDL, or SystemVerilog files with a specified name. You should specify the language type for each file using the `-verilog`, `-vhdl`, or `-sv` options. However, you may omit specifying the language type options by specifying the correct file extensions — that is, `.v`, `.sv`, or `.vhdl`.

Mixed RTL code in one HDL file is not supported. Therefore, in the same HDL file, you cannot use constructs in both Verilog and VHDL.

Each DRTL module must have a top-level module. The `-topmodule` option is used to specify a name for the top module. By default, name of the top-level module is `DRTL_TOP`.

Note: Escaped identifiers are not supported for `<DRTL_module_name>` and `<topmodule_name>`.

For example, the following module name is invalid and will produce an error message because it contains an escaped identifier, `*^dtest`:

```
module \*^dtest_ (input dclk,dreset,[1:0] dctrl, [1:0] da, [1:0] db, [1:0] dc,  
output reg [1:0] dd);  
...  
endmodule
```

**-addinst [-overwrite] <DRTL_module_name> <DRTL_instance_name>
(.<port_name1>(<signal_name1>), .<port_name2>(<signal_name2>),...)**

Creates an instance from a specified DRTL module and connects the input ports of the instance to the design signals. The above syntax follows the rules of Verilog port name association.

You can also use Verilog port positional association — that is, you can connect ports to signals based on the position of the ports in the Verilog module definition. For the syntax using positional association, see the next section — that is, [-addinst \[-overwrite\] <DRTL_module_name> <DRTL_instance_name> \(<signal_name1>, <signal_name2>,...\)](#).

You cannot add an existing DRTL instance name. However, using the `-overwrite` option removes the existing DRTL instance and adds a new instance with the specified arguments. The new instance definition replaces the previous definition.

Here is an example of connecting signals to ports based on port name association:

```
drtl -addinst mod1 inst1(.port1(sig1), .port2(sig2));
```

The TCL interpreter considers extra blanks as a TCL command. Therefore, the extra blanks in signal names in an instance definition are invalid. For example, the following definition is invalid because of the extra blanks in `top.a.addr [3 : 0]`. The TCL interpreter will consider `[3 : 0]` as a TCL command.

```
drtl -addinst drtlA mydrtlInst(top.clk, top.a.addr [ 3 : 0 ])
```

Using curly braces to enclose the arguments is useful to avoid conflicts with the requirements of the TCL interpreter. For example,

```
drtl -addinst drtlA {mydrtlInst(top.clk, top.a.addr [ 3 : 0 ])}
```

**-addinst [-overwrite] <DRTL_module_name> <DRTL_instance_name>
(<signal_name1>, <signal_name2>,...)**

Creates an instance from a specified DRTL module and connects the input ports of the instance to the design signals. The above syntax follows the rules of Verilog positional association. Signals are connected to module input ports using the same order in which the module's input ports were declared.

The connections are only made to input ports. To avoid confusion in counting port positions, it is recommended to arrange the ports in the DRTL source such that the output ports appear last, after all the input ports.

For the syntax using Verilog port name association, see the previous section — that is, [-addinst \[-overwrite\] <DRTL_module_name> <DRTL_instance_name>
\(.<port_name1>\(<signal_name1>\), .<port_name2>\(<signal_name2>\),...\).](#)

-list [-code] [DRTL_module_name_pattern]

Lists the modules that match the specified module name pattern. If you do not specify a pattern, all DRTL modules are listed. You can also view the source code for each DRTL module using the `-code` option. A pattern can be any of the following:

- An empty string (that is, module name is unspecified). In this case, all DRTL modules will be listed.
- One or more than one DRTL module names
- A string with wildcard characters (*) and (?) that specifies all DRTL modules or matching module names.

**-listinst [DRTL_instance_name_pattern] [-modulename
<DRTL_module_name>]**

Lists all defined DRTL instances that match the specified pattern. You can also view all DRTL instances for all defined DRTL modules using the `drtl -listinst` or `drtl -listinst *` commands. The `-modulename <DRTL_module_name>` option lists all DRTL instances for the specified module.

-rm <DRTL_module_name_pattern>

Removes the DRTL modules that match the specified pattern. All of DRTL instances derived from the removed module are also removed. You can also remove all DRTL modules and

instances at once using the `drtl -rm *` command. The command also returns a TCL list of the removed modules.

-rminst <DRTL_instance_name_pattern> [-modulename <DRTL_module_name>]

Removes the DRTL instances that match the specified pattern. You cannot remove a DRTL instance if one or more of its output ports are connected to the input port of other DRTL instances.

If you specify a module name using the `-modulename` option, only the instances derived from the specified DRTL module will be removed. The command also returns a TCL list of all removed instances.

For example,

```
XE> drtl -add drtlA dr(dut.clk, dut.cnt_u0.dat[3:0])
XE> drtl -add drtlA dr2(dut.clk, dut.cnt_u0.dat[3:0])
XE> drtl -add drtlA dr3(dut.clk, dut.cnt_u0.dat[3:0])
XE> drtl -rminst *
{drtlA dr} {drtlA dr2} {drtlA dr3}
```

-getallinst [DRTL_instance_name_pattern] [-modulename <DRTL_module_name>]

Returns a TCL list of DRTL instances that match the specified pattern. The instances are listed in the following format:

```
{DRTL_module_name1 DRTL_instance_name1} {DRTL_module_name2  
DRTL_instance_name2} ...
```

If you do not specify a pattern, all defined DRTL instances are listed. The `-modulename <DRTL_module_name>` command returns the instances for only the specified module.

-getallmodule [DRTL_module_name_pattern]

Returns a TCL list of DRTL modules that match the specified pattern. If pattern is not specified, you get a TCL list with all defined DRTL modules.

-enableSysTask | -disableSysTask <DRTL_instance_name_pattern>

Enables or disables the system tasks in the DRTL instances that match the specified pattern. By default, if an instance contains system tasks, the system tasks are disabled immediately after the instance is created. To make the system tasks effective, you need to enable them.

UXE Command Reference Manual

Run-time Commands

You can also use * as the pattern in the command line to enable or disable all system tasks simultaneously. You can view the status of each DRTL instance using the `drtl -listinst` command.

Note: The `drtl -enableSysTask | -disableSysTask` command works for only the DRTL instances with system tasks.

-systemTasksDefault [enable | disable | 0 | 1]

Changes the initial status of system tasks immediately after an instance is created. By default, when a DRTL instance with system tasks is created, the system tasks in it are all disabled.

- The `drtl -systemTasksDefaults [enable | 1]` command makes the system tasks enabled immediately after an instance is created. After executing this command, all new instances will be created with enabled system tasks.
- The `drtl -systemTasksDefault [disable | 0]` option will make the system tasks disabled immediately after an instance is created.

If the `drtl -systemTasksDefault` command is executed without any option — that is, without `enable`, `disable`, `0`, or `1`, the command returns the default settings for system tasks.

-save [-overwrite] <file_name> [DRTL_module_name_pattern_list]

Saves those modules to a specified file that match the specified module name pattern. If you do not specify a pattern, all defined DRTL modules will be saved to the specified file. You can also use the wildcard character * to save all defined modules simultaneously.

The save operation fails if the specified file already exists. In such cases, use the `-overwrite` option to overwrite an existing file. The save operation also fails if none of the DRTL modules match the pattern list.

-load [-overwrite | -ignore] <file_name> [DRTL_module_name_pattern_list]

Loads those modules from a specified file that match the specified module name pattern. If the pattern is omitted, all defined modules are loaded from the specified file, by default. Loading a DRTL module fails if none of the module names in the file match the specified pattern. Loading a DRTL module also fails if you use an existing module name.

You can use the following options to overcome the failure:

UXE Command Reference Manual

Run-time Commands

- Use the `-overwrite` option to overwrite the existing DRTL modules. This operation is equivalent to removing all DRTL instances from a module, and re-defining the module.
- Use the `-ignore` option to skip loading modules if there are already existing modules with same name.

force

Sets a given value for the specified signal, symbol, or bus and forces it to retain that value until explicitly released using the `release` command. A signal can be forced if it is specified as a `keepNet` during compile.

In certain rare cases, the synthesis process might not preserve the expected behavior of some signals when other signals are forced. To inform the synthesis process that a signal might be forced at run time, use the `keep_net` synthesis directive described in the *UXE User Guide*.

In LA mode, if the `force` command is operated on a bus while the clocks are running, the time coherency between the values of the signals in the bus is not guaranteed. In other words, the values of different bits will be forced in different cycles. Time coherency of the `force` command is guaranteed in other modes.

Note: If the `force` command needs to be executed on a large set of signals, the access time can be significantly improved by grouping the signals with the `symbol` command, and using that symbol in the `force` command, instead of separately executing the `force` command on each signal. When signals are grouped in such a way, a more efficient mechanism is used to access the hardware.

For most types of flip-flop and latch cells, forcing the signal connected to the Q output of the cell also forces the internal state of the cell at the next FCLK cycle. However, forcing the Q output of the cells listed below does not set the internal state. In most cases, this is because the Q output is separated from an internal flip-flop or latch by a combinational gate. This issue might affect the value after a force is removed.

Q_FDN31	Q_FDP0LS	Q_FDP0W2V1
Q_FDP0XPW2	Q_FDP2LS	Q_FDP31
Q_FDP31XP	Q_FDP61	Q_FDP61XP
Q_FDP61XPEN	Q_FJP61	Q_FJP61XN
Q_FJP61XNEN	Q_FSP61	Q_FTN31
Q_FTP31	Q_FTP61	Q_FTP61EP
Q_LDN0V1	Q_LDN0ZP	Q_LDN1ZP
Q_LDN2ZP	Q_LDN31	Q_LDN3S
Q_LDN3SZP	Q_LDN4V1	Q_LDP0FDMX
Q_LDP0W2	Q_LDP31	Q_LDP3S

UXE Command Reference Manual

Run-time Commands

Q_LDP4FDMX	Q_LMP0V1	Q_LMP0W2
Q_LMP0W2V1	Q_LSN01	Q_LSN013Y
Q_LSN31V1	Q_LSP01G	

Note: The `force` command is not supported with `vvmDebug`.

The syntax for this command is:

```
force <name> <value>
force -allff {0 | 1 | random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]
force -file -prep <textfile>
force -file <textfile.dat>
```

<name>

Specifies the name of either a signal or a symbol. A signal name representing a vector (bus) might appear with part-select notation (for example, `xyz [7 : 4]`), or without if all its bits are to be deposited.

<value>

Specifies the value to be forced to the specified signal or symbol. When the signal or symbol represents a single bit, `<value>` is either 0 or 1. When forcing to a vector or a symbol that represents several bits, the value might be specified in binary, octal, decimal, or hexadecimal radix, prefixed by the appropriate letter. For example, '`b011`', '`o1237`', '`d089`', '`hffcd`'. A number without a radix prefix is assumed to be decimal, for example 89 is the same as '`d89`'.

-allff {0 | 1 | random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]

Forces the output of most flip-flops and latches in the design. Refer to `deposit -allff {0 | 1 | random [-seed <number>]} [-instance <instance_name>] [-exclude <instance_name>]` command description for details.

Note: This option is supported only in the ICE mode and not in the SA mode.

-file -prep <textfile>

Does preprocessing for the specified text file and creates a binary (.dat) file containing an array of net IDs and corresponding data. The binary file is used by the `force_symbol` or `set_symbol` API internally that enable forcing of multiple signals simultaneously. The supported format for the file is as follows:

```
<signal_name1> <value>
```

For example,

```
sig1 'b1  
sig2 'd1  
sig3 0  
sig4 1
```

Supported Format

- Curly braces {} and inverted commas "" are supported to wrap the signal names. For example,

```
"top.net" 'b1  
{top.signal} 'b0
```

- Escaped names, hierarchical escaped names, and underscores "_" are also supported. The wildcard character "*" is not supported in signal names.

Restrictions

- The `-file` option is supported only while running the designs in the ICE mode. Using this option in the SA mode issues the following warning:

WARNING (legacy-52636): you are using 'force -file' in IXCOM mode, it may lead to some side effect.

- Only the scalar signals are allowed. Vectors must be handled as separate bits. For example,

```
dut.midv_2[0] 'h1  
dut.midv_2[1] 'h1  
dut.midv_2[2] 'h1  
dut.midv_2[3] 'h1
```

- Ensure that no duplicate nets exist in the specified file. An error message is issued at run time if duplicate nets are found.

UXE Command Reference Manual

Run-time Commands

-file <textfile.dat>

Forces multiple signals simultaneously by reading signals from a specified binary (.dat) file, which is generated as a result of the above mentioned -file -prep <textfile> command. The -file option is supported only in the ICE mode.

genPowerProfile

Generates a power consumption profile (that is, waveforms in SHM format) that can be viewed in SimVision.

This command has two options, -honorFrequency and -honorTimeScale, that can be used to synchronize the time unit in the power profile waveform with the signal waveform.

The syntax for this command is:

```
genPowerProfile -segmentLogFile <segment_log_file_from_TCF_generation>
                -reportFileNamePattern <fileNamePattern> [-honorTimeScale]
                [-honorFrequency] [-databaseName <db_name>] |
genPowerProfile {-tcfLogFiles <logFileName> | -reportFiles <powerReportFileName>
                 -powerAwareInfo <powerAwareInfoFile> [-honorTimeScale] [-
honorFrequency]
                 -databaseName <db_name> |
genPowerProfile -help
```

-databaseName <db_name>

Specifies the name of the generated database.

-honorFrequency

Specifies whether or not the waveform uses the timescale based on clock frequency specified by the segment log file or power report file.

In the ICE mode, when the value of the timeUnit run-time parameter is set to `fclk` or `dclk`, the signal waveforms are generated in terms of fast clocks. Therefore, the -honorFrequency option should be specified with the `genPowerProfile` command.

Note: If both -honorTimeScale and -honorFrequency command options are executed, time scale is honored. If both are not executed, by default `1ns` is used.

-honorTimeScale

Specifies whether or not the waveform uses the timescale specified by the segment log file and/or power report file.

In the SA mode, the -honorTimeScale option should be specified with the `genPowerProfile` command because the signal waveforms are generated in terms of

simulation time. Similarly, specify this option in the ICE mode, when the value of the timeUnit run-time parameter is set to fs, ps, ns, us, ms, s, or sec.

-reportFileNamePattern <fileNamePattern>

Specifies a pattern or generic file name for the power report file name.

The pattern can use %d to represent integer starting from 1. For example, specifying the pattern dut/f%d.rpt names the power report files as dut/f1.rpt, dut/f2.rpt, and so on.

-reportFiles <powerReportFileName>

Specifies the names of the power report file. The * and ? wildcard characters can be used. The report files have two parts, first part is from the TCF writer, the second part is from compiler `report power` command.

Format of the first part of a report file is as following:

```
ClockFreq <double> MHZ WaveformTS <integer> s/ms/us/ms/ps/fs TopCell <cellName>
Instance <instanceName> TCFFile <tcfFileName>
Segment [<integer> <integer>] Samples <integer> Signals <integer>
togglesPerCycle <double> highsPerCycle <double> toggleRate <double> highRate
<double>
```

Format of the second part of a report file is as following:

Instance	Cells	Leakage Power	Internal Power	Net Power	Dynamic Power	Total Power
<name>	<integer>	<double>	<double>	<double>	<double>	<double>

Using the `-reportFiles <powerReportFiles>` option generates the segment timing and power consumptions for all instances in the given power report files.



The `<powerReportFiles>` option should attach TCF log file content before all the instance power consumption information.

-powerAwareInfo <powerAwareInfoFile>

Specifies the file which contains power-aware information when CPF is used. The file should be set as segment log file name.

Note: This option is mandatory only when performing CPF-aware DPA using the genPowerProfile merged file use model. For detailed information, refer to the following sections of the *Analyzing Power Consumption of the Designs* chapter in UXE User Guide:

- ❑ *Analyzing Power Consumption with CPF* section for CPF-aware DPA information
- ❑ *Generate power consumption profiles* section for merged file use model

-segmentLogFile <segment_log_file_from_TCF_generation>

Specifies the name of the TCF files segment log file generated by executing the dpa -outfile -tcf <filename> -slaveTCF -masterTCF -segment command.

Note: This log file can be generated using a TCF writer or created manually.

A segment log file is of the following format:

```
ClockFreq <double> MHZ WaveformTS <integer> s/ms/us/ms/ps/fs TopCell <cellName>
Instance <instanceName> TCFFile <tcfFileName>
Segment [<integer> <integer>] Samples <integer> Signals <integer>
togglesPerCycle <double> highsPerCycle <double> toggleRate <double> highRate
<double>
.....
Segment [<integer> <integer>] Samples <integer> Signals <integer>
togglesPerCycle <double> highsPerCycle <double> toggleRate <double> highRate
<double>
```

-tcfLogFiles <logFileName>

Specifies the names of the TCF log files. The waveforms generated based on these log files show the segment timing information, and TCF file average toggles and high probabilities.

-help

Returns the syntax of the genPowerProfile command.

Note: This command option can also be executed from the UNIX command line.

getCableStatus

Provides realtime checking on the HDDC cable and retrieves the status information for the cable running with the design.

Without any parameters, the `getCableStatus` command returns the following basic status information about the HDDC cable:

- Number of cables used
- Logical names of cables specified during compilation
- Logical locations of cables specified during compilation
- Interface Status (Interface LED status) of cables
- Target Status (Target LED status) of cables

Note: The `getCableStatus` command is not supported with `vvmDebug`.

The syntax for this command is:

```
getCableStatus [-detail | -list]
```

-detail

Retrieves detailed information about HDDC cables. It provides the following information in addition to the basic status information:

- Connector types of cables specified during compilation
- IO technology (specified during compilation) used for cables
- Measured I/O voltage (depends on connector type) for the cable programming
- Measured target voltage 0 (depends on connector type) on vsense0
- Measured target voltage 1 (depends on connector type) on vsense1
- Pin map for a contention (if any)

Due to the extensive hardware query required to retrieve the information correctly, the output is generated at a slow speed.

Note: If the emulation clock is not running, the `-detail` option is ignored.

-list

Returns the status of the cables used by the design as a list of sub-lists. There is one sub-list for each cable. The sub-list comprises of the following information:

- Cable name
- Connector location
- Status of the Interface Status and Target System Status LEDs

Using the `-list` option with the `getCableStatus` command returns the cable-related errors, if any; though warnings and information messages about the cable are not printed.

Debugging Contention on Target Cable Pins

Contention on a target cable pin means that the emulator is driving a value (1 or 0) on the pin, but the target is driving a different value.

If contention occurs immediately on downloading a design, warnings like the following are issued:

```
WARNING (legacy-47469) : MY_CABLE1 6_J3 GREEN(Active) YELLOW(contention)
```

```
WARNING (legacy-46457) : Tristated/no Target/Cable Error detected in design.
```

Contention might also occur later, when the design is running. In this case, the Target System Status LED glows yellow, but the software does not issue a warning.

Following are some of the reasons that might cause contention on target cable pins:

- The compile script assigns signals to the wrong cable pins.
- The cable is connected to a wrong connector on the target.
- The design drives a signal that should be an input to the design. (If the design drives a primary input, it is automatically converted to act as an output from the emulator.)
- Both the design and target drive a bidirectional signal at the same time.

To debug contention, first identify the pin(s) with the contention problem using the following command:

```
getCableStatus -detail
```

Before running the above command, ensure that the emulator's FCLK is free-running. FCLK becomes free-running after a `run` command is executed in LA mode, or after executing `run -nowait` in STB mode while triggers are disabled.

Note: This command takes a few minutes to run.

After identifying the pin with contention problems, use any of the following methods to locate the exact contention issue:

- **Using the dbFiles/*.ctl file and running the terminalAssign -dump command:**

Select a pin number that is reported to have contention, and perform the following steps:

- a. View the dbFiles/*.ctl file to locate the design signal name that is associated with the specified pin number. The contention report uses the same pin numbering scheme as used by the dbFiles/*.ctl file.
- b. Ensure that the pin direction for this pin specified in the *.ctl file, is as expected. A unidirectional output from the emulator is shown in the *.ctl file in a single line with the suffix ".O".

For example:

```
6_J6.75.O tdo * PU
```

A bidirectional pin is shown in the *.ctl file in three lines. Following is an example:

```
6_J6.75.I tdo * PU
6_J6.75.O QT@tdo:data * PU
6_J6.75.E u1.tdo_pad.oe * PU
```

- c. If the *.ctl file shows an input signal as an output, it means that, incorrectly, the signal has a driver in the design.
- d. Check the value of the enable signal in the waveform to see if a bidirectional signal is driven when it should not be.
- e. Check the target pin to which you assigned the specified signal by using the following command:

```
terminalAssign -dump my_ta.xel
```
- f. Ensure that the signal has been assigned to the correct pin.
- g. Check if the target is driving the pin inappropriately. If you are using a TIB-192 or MTIB-88 adapter, you can disconnect the target by removing the appropriate ribbon cable. Disconnecting the target should eliminate the contention.

- **Using the contention command:**

Run the contention command in the same session directory in which the get CableStatus -detail command was run. For example:

```
<installdir>/share/uxe/gift/misc/contention
```

This command prints a report indicating the design signals that have contention and the target pins to which they are assigned.

Examples

- To get the basic status information about the HDDC cable, use:

```
XE> getCableStatus
```

Following information is returned:

```
INFO : Getting cable information, this might take a while.  
INFO : Design contain 1 cable.  
INFO : Name Conn Interface Target  
INFO : my_cable_0_1 0_J1 GREEN(Active) GREEN(Active)
```

- To display the detailed status information about the HDDC cable when the emulation clock is running free, use:

```
XE> getCableStatus -detail
```

Following information is returned:

```
INFO : Getting cable information, this might take a while.  
INFO : Design contain 1 cable.  
INFO : Name Conn Interface Target Type IO Tech  
INFO : my_cable_0_1 0_J1 GREEN(Active) GREEN(Active) TIB_192 LVCMOS33_8  
INFO : - IOVolt: 3.299 VSENSE0_Volt: 3.931 VSENSE1_Volt: 0.715
```

- To display the HDDC cable status information when the TIF cable is disabled or tristated, use:

```
XE> getCableStatus
```

Following information is returned:

```
INFO : Getting cable information, this might take a while.  
INFO : Design contain 1 cable.  
INFO : Name Conn Interface Target  
WARNING (legacy-47469) : my_cable_0_1 0_J1 FLASHGREEN(Tristate) GREEN(Active)  
WARNING : Tristated/no Target/Cable Error detected in design.  
WARNING : If the cable is actively in use in the design,  
WARNING : I/O signals assigned to that cable are undriven and show flat  
waveforms,  
WARNING : even if the design drives those signals with toggling values.
```

- To display the HDDC cable status information when the I/O of the TIF cable is in contention, use:

```
XE> getCableStatus -detail
```

UXE Command Reference Manual

Run-time Commands

Following information is returned:

```
INFO : Getting cable information,,this might take a while.  
INFO : Design contain 1 cable..  
INFO : Name Conn Interface Target Type IO Tech  
WARNING : A__CABLE_0_1 0_J1 GREEN(Active) YELLOW(contention) VHDM LVCMOS33_8  
INFO : --IOVolt:3.299 TargetVolt0:3.821 TargetVolt1:0.302  
INFO : --pin [1.....32 ]:00011001 10000110 00011000 01100001  
INFO : --pin [33.....64 ]:10000110 00011000 01100001 10000110  
INFO : --pin [65.....96 ]:00011000 01100001 10000110 00000000  
INFO : --pin [97....128 ]:00000000 00011001 10000110 00011000  
INFO : --pin [129..160 ]:01100001 10000110 00011000 01100001  
INFO : --pin [161..192 ]:10000110 00011000 01100001 10000110  
WARNING : Tristated//no Target/Cable Error detected in design.  
WARNING : If the cable is actively in use in the design,,  
WARNING : I//O signals assigned to that cable are undriven and show at  
waveforms,  
WARNING : even if the design drives those signals with toggling values.
```

- To display the status of the cables used by the design and get the cables as a Tcl list, use:

```
XE> getCableStatus -list
```

Following information is returned:

INFO : Name	Conn	Interface	Target
INFO : PCIX	0_J1	GREEN(Active)	GREEN(Active)
INFO : 1J1	0_J2	GREEN(Active)	GREEN(Active)

getCycleNum

Returns the current verification cycle number.

In the STB, and LA modes, the command returns the number of Fast Clock (FCLK) cycles that have been executed since the beginning of the debug session (since the execution of the download command, or since the last execution of the resetTime command).

FCLK (also referred to as Fast Clock) is the basic emulation clock. Operations such as SDL evaluations, or probe samples are paced by this clock.

In the Vector Debug mode, the command returns the number of stimulus vectors that have been executed relative to the beginning of the stimulus file.

In the SA mode, the command returns the number of emulation cycles that have been executed relative to simulation time zero.

The syntax for this command is:

```
getCycleNum
```

Examples

```
# Initialize debugger; use /mydesign/design1 directory.  
debug /mydesign/design1  
# Download design to the 'samurai' emulator host.  
host samurai  
download  
# Configure to Vector Debug mode, specify stimulus file, and run 30 vectors.  
configPM -vd  
vector mytest.in -  
run 30  
# Show the current emulation cycle number.  
getCycleNum
```

getHostStatus

Returns the status of the emulation host specified in the `host` command. This command is available in all modes. Before using this command, execute the debug command. The status is reported in the form of one of the following strings:

- `NotSet` - Indicates that the emulation host has not been specified for the current session.
- `Set` - Indicates that the emulation host has been specified and connected successfully.
- `Locked` - Indicates that only the resources required by the current design are locked for exclusive use of the current process.
- `Downloaded` - Indicates that the design has been downloaded to the emulation host and is ready to run.
- `Released` - Indicates that the emulation host was previously connected in the current session and has since been released.

Note: The `getHostStatus` command is not supported with `vvmDebug`.

The syntax for this command is:

```
getHostStatus
```

Examples

```
# Initialize the debugger and use the /mydesign/design1 design database directory.  
debug /mydesign/design1  
# Show the current status of any host connection  
getHostStatus  
# Returned value:  
NotSet  
# Use the `samurai` emulation host.  
host samurai  
# Show the current status of the `samurai` emulation host.  
getHostStatus  
# Returned value:  
Set
```

getPMStatus

Returns information about the current status of the emulator's instrumentation.

This command can be used to track the progress of triggering and data collection while the emulator is busy. It also enables to determine when the emulator becomes available after a run command.

Note: The `getPMStatus` command is not supported with `vvmDebug`.

When executed without any arguments, the information about the state of the emulator's instrumentation is returned in the following format:

```
<currentState> <counter1> <counter2> <preTrigger> <postTrigger> <abortFlag>
    <busyFlag> <clockFlag> <triggerFlag> <UclockFlag>
```

`<currentState>` Specifies the symbolic name of the state that the SDL program is now executing.

`<counter1>` Specifies the current value that counter1 in the SDL program has reached.

`<counter2>` Specifies the current value that counter2 in the SDL program has reached.

`<preTrigger>` Specifies the number of samples that were captured into the trace buffer before triggering, in the most recent run.

`<postTrigger>` Specifies the number of samples that were captured into the trace buffer after triggering.

`<abortFlag>` The `<abortFlag>` is set to T (True) if the run (or just data collection, depending on the mode) was aborted prematurely. It is set to F (false) if the run was completed normally. The run can be terminated prematurely by several means such as the `stop` command or by pressing `CTRL+C`.

`<busyFlag>` Specifies either T (true) if the SDL or capture clocks are currently busy, or F (false) if the SDL or capture clocks are available. Busy means that a run is still in progress.

Note: In LA mode, the run is in progress as long as the data collection is in progress.

UXE Command Reference Manual

Run-time Commands

<clockFlag> Specifies either T (true) if emulation clocks are running, or F (false) if emulation clocks are stopped.

If running an IXCOM-compiled design that also contains uncontrolled clocks, this flag conveys only the status of the controlled clocks.

<triggerFlag> Specifies either T (true) if the trigger has occurred, or F (false) if triggering has not yet happened.

<UclockFlag> Specifies either T (true) if uncontrolled clocks are running, or F (false) if uncontrolled clocks are stopped.

Note: This flag is reported only for the designs that have uncontrolled clocks. If the design does not have uncontrolled clocks, this flag does not appear in the returned string.

Note: The arguments *<currentState>*, *<counter1>*, and *<counter2>* relate only to the first SDL instance. To retrieve state and counter value information for other SDL instances (when the SDL program has more than one), use the `sdl -getStatus` command or the `sdl -report` command. For details, refer to [sdl](#) command.

The syntax for this command is:

```
getPMStatus [-pretrigger | -posttrigger | -abort | -busy | -clock | -trigger |  
-uclock]
```

With any of the arguments specified above, the command returns the value only for that argument. For example, the `getPMStatus -busy` command returns a single character, either F or T.

Partial match is also accepted. For example, `-po` is accepted instead of `-posttrigger`, but `-p` is not accepted because it can match either `pretrigger` or `posttrigger`.

Examples

```
# Initialize the debugger and use the  
# '/mydesign/design1' design database directory.  
debug /mydesign/design1  
  
# Download the design to the 'samurai' host.  
host samurai  
download  
  
# Configure for Logic Analyzer mode.  
configPM -la  
  
# Run emulation and show status.  
run  
getPMStatus  
  
# Returned value:
```

UXE Command Reference Manual

Run-time Commands

```
ST1 9 17 104 30 F T T F
```

```
getPMStatus -clock
```

```
# Returned value:
```

```
T
```

getTime

Returns the current simulated time. This command is available in all emulation modes.

Executing the [reset](#) or [resetTime](#) command in STB, Vector Debug, or LA mode will reset the time counter that is used by `getTime`.

The syntax for this command is:

```
getTime [-simulator] [<timeUnit>]
```

-simulator

This option is applicable only when running a design compiled using IXCIM. When this option is specified, the command returns the simulated time from the ncsim simulator. Without this option, the command returns the time value that is consistent with the waveform display for DUT signals.

This makes a difference when using a design that has uncontrolled clocks, and when the capture mode is set to uncontrolled (see documentation for command [database -captureMode uncontrolled](#)). In such a case, the time scale on the waveform display uses a nominal time value for the basic uncontrolled clock cycle (1ns per FCLK), which is different than the time simulated by ncsim.

<timeUnit>

Specifies units by which the returned result is scaled. For example, a value of `ns` requests to return the result scaled in nanoseconds.

For a complete list of legal values for `<timeUnit>` and their meaning, refer to the [timeUnit](#) run-time parameter.

If `<timeUnit>` is not specified, the command uses the default unit that is currently set for the `timeUnit` run-time parameter.

Examples

```
XE> debug ~john/mytest/test1
XE> host pluto
XE> download
XE> configPM -stb
XE> run 100
XE> getTime
100 fclk
```

UXE Command Reference Manual

Run-time Commands

```
XE> xeset timeUnit dclk  
XE> getTime  
50 dclk  
XE> getTime ps  
500 ps
```

hdlConfig

Initializes the system at run time for system tasks processed by `hdlImport -atb` and `hdlOutputFile -f qel`. For more information on these system tasks, refer to the *UXE User Guide*.

The `hdlConfig` command is available only in the ATB mode. In addition, this command can be used to enable/disable some compile options without recompiling the design. This command is not supported with `vvmDebug`.

There are two syntax supported by this command, which have specific purpose and usage:

- hdlConfig for Initializing the System Tasks is used to initialize the system tasks
- hdlConfig for Enabling/Disabling the Compile Options is used to enable/disable compile options



The options of these syntax cannot be mixed together. Only one syntax can be used for each `hdlConfig` command.

hdlConfig for Initializing the System Tasks

The `hdlConfig` command with this syntax must be invoked before the first `run` command at the emulation time 0. It must be invoked only once in the entire run session.

This command is specified immediately after the configPM command at run time.

The syntax for this command is:

```
hdlConfig <file>
  [-cycleTime=<cycle_time>]
  [-top=<instance>]
  [-displayDebug]
  [+<plusargs>]
```

<file>

Specifies the file generated by `hdlOutputFile -f qel <file>` command during the HDL synthesis process so that the HDL Compiler can extract system task and initialization information from Verilog/VHDL files at run time.

-cycleTime=<cycle_time>

Specifies one half of cycle time of a user design's fastest clock, where *<cycle_time>* is an integer. For instance, if the simulation model of a clock source is:

```
clk = 0;  
forever begin #100 clk = ~clk; end
```

the *<cycle_time>* argument should always be 100 as *<cycle_time>* does not depend on CAKE value. Note that the clock simulation model must be removed before invoking xeCompile for emulation compilation, and the clockSource XEL command should be used instead. You must provide the *<cycle_time>* information because xeCompile is not able to synthesize delays or clock generators.

-top=<instance>

This argument is necessary only when the top module for `hdlSynthesize` is different from the top module specified by the `design` command. *<instance>* is the instance path to the top module specified for `hdlSynthesize`.

-displayDebug

Prints the cycle number each time before messages are updated. This option is useful for debugging when the messages generated in the log file `hdlRun.log` are incorrect.

+<plusargs>

This argument is used to configure the `$test$plusargs` and `$value$plusargs` system functions.

You can provide more than one *<plusargs>* in the same command line. For example:

```
hdlConfig tb_init.xel +debug_on + maxCycle=500
```

Example 1

If the two system functions, `$test$plusargs` and `$value$plusargs` appear in if expressions like:

```
if ($test$plusargs("debug_on"))  
    debug_mode = 1;  
else  
    debug_mode = 0;
```

```
if ($value$plusargs("maxCycle=%d", maxc))
else maxc = 500;
```

The `hdlConfig` command to change the arguments of the `$test$plusargs` and `$value$plusargs` system functions can be something like:

```
hdlConfig tb_init.xel +debug_on + maxCycle=500
```

hdlConfig for Enabling/Disabling the Compile Options

This syntax can be used any time after the `hdlConfig` command with [hdlConfig for Initializing the System Tasks](#) is executed for initialization. It can be used at multiple times during a run session.

The syntax for this command is:

```
hdlConfig [-disable {<insPath> | -top | -all}]
          [-enable {<insPath> | -top | -all}]
          [-disableBuffer]
          [-enableBuffer]
          [-timeOut <cycleNum>]
```

-disable {<insPath> | -top | -all}

Disables all the system tasks, such as `$display`, `$write`, `$readmemb`, `$readmemh`, `$stop`, `$finish`, `$writememb`, and `$writememh`, depending on the following parameters provided with this option:

- `<instPath>`: Disables all the system tasks in an instance pointed by `<instPath>`. For example, when a system task slows down the run-time performance, you might choose to disable it for a while to speed up the performance.
- `-top`: Disables the system tasks in the top module.
- `-all`: Disables all the system tasks in the design.

-enable {<insPath> | -top | -all}

Enables the system tasks that were disabled using the `-disable` option of the `hdlConfig` command. The system tasks are enabled based on the following parameters of the `-enable` option:

- `<instPath>`: Enables those system tasks that were disabled using the `-disable` `<instPath>` option.

UXE Command Reference Manual

Run-time Commands

- **-top:** Enables those system tasks in the top module that were disabled using the **-disable -top** option.
- **-all:** Enables all those system tasks that were disabled using the **-disable -all** option.

Note: The **-all** parameter must be used independently of **<instPath>** or **-top** options. The **-enable -all** option does not enable those system tasks that were disabled using the **-disable <instPath>** or **-disable -top** options. Similarly, **-enable <instPath>** or **-enable -top** options do not enable system tasks that were disabled using the **-disable -all** option.

-disableBuffer

Disables the compiler option **-displayBuffer** so that each message is updated immediately. When used together with the **-displayDebug** option, this option prints a cycle number for each message.

-enableBuffer

Enables the **\$display** buffer that was disabled using the **-disableBuffer** option, if the **-displayBuffer** option was applied during the compile time.

-timeOut <cycleNum>

Overwrites the cycle number specified using the **-displayBuffer+timeOut=<num>** option at the compile time. The number **<cycleNum>** must be in decimal integer format and must be larger than 2^{19} and smaller than the number specified by the **-displayBuffer+timeOut=<num>** option. For example, you can reduce the **timeOut** number for messages to be updated more quickly for easier debug, and then increase the **timeOut** number later to speed up the performance.

hdlRunLogFile

Changes the default name of the log file generated by the [hdlConfig](#) command. The `hdlConfig` command records the messages generated by system tasks like `$display`, `$write`, `$stop` and `$finish`, by default in the `hdlRun.log` file.

The `hdlRunLogFile` command is available only in the ATB mode. This command is not supported with `vvmDebug`.

The syntax for this command is:

`hdlRunLogFile [<file> [-append]]`

<file>

Specify a new log file for the messages generated by system tasks like `$display`.

[-append]

Appends the messages to an existing log file. The default is to overwrite a log file if it exists.

host

Selects the emulation host name to be used during emulation of the design, and changes / gets the current status of a specified emulation host.

The host command is not supported with vvmDebug.

Before using this command, execute the [debug](#) command. Most options of the host command (except -offline) require prior specification of the emulator host name.

```
debug .
host <hostname>
host <TIF_options>
```

The host TIF commands can be used to disable or enable the tristate drivers of the target interface. Disabling the tristate drivers of the target interface lets you connect, disconnect, or power cycle your target circuit while it is still connected to the emulator, without shutting down the emulator. The TIF options (-enableTIF, -disableTIF, and -statusTIF) are not available in Vector Debug mode.

The syntax for this command is:

```
host <hostname> [-path <design_path>] [-reserveFirst | -noReserve]
                 [-timeout <timeoutValue>] [-key <reservationKey>] [-keyfile
                 <reservationFilePath>] [-firstFit]
                 [-keepAlive] [-testHwFirst] [-dynp]

host -check <hostname>
host -checkHDVer <hostname>
host -checkSuspendStatus
host -kill
host -location
host -offline {<data_file_name> | <*.phy_file_name>}
host -release
host -enableTIF
host -disableTIF
host -statusTIF
```

<hostname> [-path <design_path>] [-reserveFirst | -noReserve]

Specifies the name of the emulation host.

Note: If the emulation host is the same workstation from which `xeDebug` is run, a dot (.) can be used instead of `<hostname>`.

Optionally, use the `-path <design_path>` option to specify the path to access the design on the host machine.

If a `.bp` file exists in the design directory, in ICE mode, the `host <hostname>` command automatically reserves the emulation host for you. In the SA mode, to reserve the emulation host before downloading the design, the `-reserveFirst` option is also required with the `<hostname>` option as shown below:

```
host <hostname> -reserveFirst
```

In ICE mode, the default behavior is to reserve the required domains prior to download. The `-noReserve` option enables you to skip the emulation host reservation and proceed directly to download.

Note:

- ❑ The emulation host is reserved for 10 minutes for one time use only.
- ❑ The emulation host reservation feature is turned on only if either of the following two `.bp` files is available in the design directory:
 - `<design_directory>/<hostname>. <design_name>.bp`
 - `<design_directory>/<design_name>.bp`
- If both these `.bp` files are available, `xeDebug` uses the `<hostname>. <design_name>.bp` file.
- ❑ If the emulation host reservation fails, the `host` command still continues to run.
- ❑ `xeDebug` cancels the reservation before the design run is terminated.
- ❑ An error message is generated if both the `-noReserve` and `-reserveFirst` options are present.

-timeout <timeoutValue>

Specifies the number of seconds for which the required emulator resources should be reserved before downloading the design. The `<timeoutValue>` is a positive integer. The default timeout value is 600 seconds. If an early domain reservation is not to be done, this option is ignored.

Note: The `xeset reserveTimeout <timeoutValue>` command executed before the `host` command provides identical functionality to the `host -timeout <timeoutValue>`

command. If both are specified, the host -timeout <timeoutValue> command takes precedence.

-key <reservationKey>

Specifies the reservation key to be used with the session. The <*reservationKey*> is a positive integer.

If a reservation already exists for the key, the existing reservation is used. If it does not exist, even if another reservation is currently available, a new reservation is made with the specified key. If no key is specified and no reservation is available, a random key is created.

-keyfile <reservationFilePath>

Specifies the location of a file for reservation.

The <*reservationFilePath*> requires that all directories exist although the leaf filename might not exist. If the file already exists, it is overwritten.

This option provides the same functionality as provided by the -keyfile <*reservationFilePath*> option of the test_server command.

Note: If the -keyfile <*reservationFilePath*> option of the test server command is provided, the option should also be provided with the host command. Both paths should be consistent.

-firstFit

Identifies the domains to be reserved using the first-fit algorithm.

This option provides the same functionality as provided by the -firstfit option of the test_server command.

-keepAlive

Enables the connection between the host and the emulator to be retained even after the design run is finished.

Note: If the -keepAlive option of the host command is used, the host <*hostname*> command must not be executed again for the same debug session.

In this mode, when xeDebug exits, the hardware resources on the emulator for the particular design remain locked for the time period specified using the [xeset keepHostAliveTimeout](#) run-time parameter.

 **Important**

Alternatively, execute either of the following commands with the value set to 2:

- `setenv XE_KEEP_HOST_ALIVE 2`

Set this environment variable before starting xeDebug.

- `xeset keepHostAlive 2`

Execute this command before downloading the design.

-testHwFirst

Tests the hardware before downloading the design.

Note: Hardware is tested before downloading only if the `-testHwFirst` option is specified. If the hardware test fails, the `host` command returns an error.

This feature is available only when either of the following two `.bp` files exists in the design directory:

- `<design_directory>/<hostname>.<design_name>.bp`
- `<design_directory>/<design_name>.bp`

If both these `.bp` files are available, xeDebug uses the `<hostname>.<design_name>.bp` file.

-dynp

Changes the vision mode to DYNP if the design was compiled for FV mode, at run time. You need not re-compile the design.

-check <hostname>

Tests the current design prototype for download on the specified host and prints either of the following two values:

- `Fit` if the design can be successfully downloaded on the specified host.

- Unfit if the design cannot be downloaded on the specified host.

This option also takes into account the `.bp` and `.br` files available in the current design directory.

-checkHDVer <hostname>

Returns the physical hardware type of the specified host.

-checkSuspendStatus

Checks the current state of the emulator that is, SUSPENDED or ONLINE and returns it on the command prompt. The state is controlled by the commands used for setting the emulator status, such as resume, stop, or suspend.

-kill

Terminates the KeepAlive session between the host and the emulator after the design run is finished, if the KeepAlive feature is already enabled. The `host -kill` command is ignored with a warning if the KeepAlive feature is not enabled.

Alternatively, execute the `xeset keepHostAlive 0` command.

-location

Returns the list of actual resources to which the design has been downloaded.

The `-location` option requires prior execution of the `download` command.

-offline {<data_file_name> | <*.phy_file_name>}

Specifies the name of the data file or `.phy` file to be used in the offline mode.

Note: The `.phy` file is used only in FullVision mode.

-release

Releases the emulation host.

-enableTIF

Enables the TIF (Target Interface) for the target cable. After the design download, by default, the HDDC cables are enabled to drive the target interface.

-disableTIF

Disables the TIF for the target cable.

-statusTIF

Returns the current TIF status for the target cable. If the HDDC cable is currently enabled, the -statusTIF option returns 1; else, 0 is returned.

Examples

```
# Initialize the debugger and use the
# '/mydesign/design1' design database directory.
debug /mydesign/design1

# Use the 'samurai' emulation host.
host samurai
host -lock
```

infiniTrace

Controls and initializes the InfiniTrace mode.

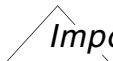
InfiniTrace enables you to continuously save the state of the design during simulation and emulation. Later, this state can be restored from any time point to observe the emulation status. This includes dumping memory content, getting the value of any nets in the design, and adding probes for uploading waveforms of any net.

InfiniTrace can be used only in the online mode; you must download a design and access an emulator to be able to use InfiniTrace. InfiniTrace is supported in STB and SA emulation modes, with or without a static target system, using either FV or DYNP probing mode. It is, however, not supported with vvmDebug.

The online debugging session operates in three modes: normal run, prepare session, and observe session. A normal run is normal emulation without using InfiniTrace. A prepare session is the mode in which the system collects data while the emulation is running. An observe session is the mode in which emulation status is restored to a specific time point by applying the data collected during prepare session.

The compiler inserts logic and memory buffers into the design to capture the design state during every emulation cycle. When the buffer is filled, emulation pauses to upload the buffer, and restarts emulation to begin capture of a new set of data in the buffer. In a prepare session, the run, pause, upload, resume running steps are repeated, causing a slowdown of the effective simulation and emulation speed. The target will also pause while data upload takes place. This is the major difference between a prepare session and normal run. Any operation that can be performed during a normal run can be performed during a prepare session.

During an observe session, emulation status is restored to a specific time point. In an observe session, you can perform any operation that “observes” the emulation status, such as dumping memory content, getting values of nets, defining probes and uploading waveform. You cannot perform operations that change emulation status and alter the path of the emulation, for example, memory write, set or force or release on nets, or running emulation. During the observe sessions, the input and output to the target is frozen, so that operations on emulator will not affect the target.



Important
The maximum trace memory size for InfiniTrace prepare and observe modes is 2147483647, which is the maximum value for a 32-bit integer.

A 40-bit counter is inserted into your design. This counter is initialized to zero at the beginning of InfiniTrace and will be increased by one at every emulation cycle. This counter indicates how many cycles have passed since the start of emulation.

UXE Command Reference Manual

Run-time Commands

It can only be used in online mode. Therefore, you must download a design and access an emulator before using InfiniTrace. Ability to access certain functions depends on the mode in use.

If the you are in an InfiniTrace session, you must exit the session before suspending the emulation.

The syntax for this command is:

```
infiniTrace -prepare <session_name> [-append]
infiniTrace -prepare <session_name> [-overwrite]
infiniTrace -on | -off
infiniTrace -rm <name>
infiniTrace -observe <name>
infiniTrace -goto [-continuel] <time specification>
infiniTrace -stateSaveFrequency <value>
infiniTrace -getStateSavePoints
infiniTrace -getCurrentMode
infiniTrace -getCurrentMode [-text]
infiniTrace -getSessionName
infiniTrace -isCaptureEnabled
infiniTrace -bookmark -add <book_mark_name>
infiniTrace -bookmark -rm <book_mark_name>
infiniTrace -bookmark -list
infiniTrace -bookmark -goto <book_mark_name>
infiniTrace -close
infiniTrace -close [-force]
infiniTrace -list [-info]
infiniTrace -intervals
```

-prepare <session_name> [-append]

Starts a prepare session and creates a directory with the specified *<session_name>*. The *<session_name>* directory is created to store the data files for the InfiniTrace session.

Note: The *-prepare <session_name>* command cannot be executed when a prepare or observe session is running.

The `-append` option can be used only in the ICE mode to add the new data to an existing InfiniTrace session.

-prepare <session_name> [-overwrite]

Starts a prepare session with the specified name and overwrites the data of the existing InfiniTrace session with the new data. Note that the name of an existing prepare session cannot be used without this option.

-on | -off

Enables or disables data capture. After the initial `infiniTrace -prepare` command, data capture is enabled by default. In an observe session, you can only go to a time point when the data capture is enabled.

-rm <name>

Removes the specified data set, including the directory and its contents. The session must be closed prior to issuing this command.

-observe <name>

Explicitly opens an observe session. If a session is already open, it is closed. If an observe session is opened in this way, the current cycle will be always smaller than ending cycle. Issuing a `run` command inside an explicit observe session, unlike a temporary observe session, generates an error condition.

In the ICE and SA modes, you must go to the end, if they are in a temporary observe session.

In the SA mode, the `-observe` option is available only with the use of the `xeDebug` tool. You cannot use this option when you use the InfiniTrace feature with the `irun` tool.

-goto [-continue] <time_specification>

Restores emulation status to the specified time point. This command applies only to an observe session.

In the ICE mode, if this command is issued within a prepare session, it is temporarily closed and an observe session with the same name is opened. Note that in the SA mode, this command cannot be issued within a prepare session. Also, in the SA mode, this option is

UXE Command Reference Manual

Run-time Commands

available only with the use of the xeDebug tool. You cannot use this option when you use the InfiniTrace feature with the irun tool.

The *<time_specification>* parameter can be either an absolute time or a relative time based on current time, and/or the keyword `trigger`, as follows:

- `t` - An absolute time specification, where `t` specifies the time
- `+t` - Advances from the current time, where `t` specifies the time
- `-t` - Decrement from the current time, where `t` specifies the time
- `trigger` - Searches forward from the current time until a trigger point is encountered or reaches the ending time point of the current observe session. The trigger condition is defined using SDL the same way as it is defined in normal run.

When `trigger` is used, you can also specify the `-continue` option. It indicates that the SDL internal state will not be reset, but will continue from the previous `infiniTrace -goto trigger`. The following statements illustrate the use of the `-continue` option with the `infiniTrace -goto trigger` command:

- `infiniTrace -goto -continue trigger`
- `infiniTrace -goto -continue trigger 1000`
- `begin` - Goes to the first cycle
- `end` - Goes to the end cycle

The time `t` can be specified with or without a time unit (possible time units are `s`, `ms`, `us`, `ns`, `ps`, and `fs`). In STB mode, the values `dclk` and `fclk` can also be specified as legal time units. For more details, refer to the `timeUnit` run-time parameter.

For example, in order to go to simulated time of 1000 ns, you can execute the following command:

```
infiniTrace -goto 1000 ns
```

If the time unit is not specified (like `infiniTrace -goto 1000`), the default is determined by the `timeUnit` run-time parameter, which can be set using the `xeset timeUnit` command.

The default initial value for the time unit is different between the STB and SA modes. In STB mode, the default time unit is `fclk`, which means that the time `t` specifies the number of FCLK cycles.

UXE Command Reference Manual

Run-time Commands

You can specify time t along with trigger. This indicates that the searching of the trigger point will stop either at the trigger or at the specified time t , whichever is reached first. For example,

```
infiniTrace -goto trigger 1000
```

If any dynamic probes need to be added (using the probe -add command), the command must be given before the `infiniTrace -goto` command to ensure that those probes are traced.

Additional `probe -add` commands can be executed anytime during the observe session and these signals will be traced after the next `-goto` command. If the time of the `-goto` command is too close to the first StateSavePoint, the trace buffer might not be as full as it can only contain the data starting at the time of that save.

-stateSaveFrequency <value>

Specifies the amount of time between automatic capture points. When the `-stateSaveFrequency` option is used without an argument, it returns the current time value.

The supported time format for ICE mode is in terms of number of emulation cycles. The time format for SA mode is in terms of units of simulation time.

For example, in ICE mode, the following command specifies the capture points interval as 2000000 cycles:

```
infiniTrace -stateSaveFrequency 2000000
```

Set the save frequency value using the `-stateSaveFrequency` option to trade-off between prepare session and observe session performances. If the value is large, prepare session performance will be better but observe session performance will be worse. By default, the system will try to optimize the save frequency value to balance the prepare session and observe session performances. The default value of save frequency is not a fixed number. It varies depending on the design and your computer's speed. You will get a value 0 to represent the default value if the `infiniTrace -stateSaveFrequency` command is used to query the current value.

Use the `infiniTrace -stateSaveFrequency 0` command to restore to default value.

The user-specified value is not always honored by the system. If it is too small, the system will use a minimum value. The minimum value varies depending on the design. The typical minimum value is several million cycles for the ICE mode. If the user-specified value is not a multiple of the minimum value, the actual value used is rounded up to the multiple of the minimum value.

In the SA mode, this option is available only with the use of the xeDebug tool. You cannot use this option when you use the InfiniTrace feature with the irun tool.

-getStateSavePoints

Returns information about the automatic saves that happen during the prepare mode. In ICE mode, this option returns the cycle number for each save point. In SA mode, it returns the unit of simulation time corresponding to each save point. Also, in the SA mode, this option is available only with the use of the xeDebug tool. You cannot use this option when you use the InfiniTrace feature with the irun tool.

-getCurrentMode

Specifies if the InfiniTrace session is in Hardware-mode InfiniTrace (HWIT).

For more information on HWIT, refer to the *UXE User Guide*.

-getCurrentMode [-text]

Returns the current status as follows:

- 0 - Indicates normal run
- 1 - Indicates inside prepare session
- 2 - Indicates inside observe session

If the `-text` option is specified, the command returns `normal_run`, `prepare`, or `observe` instead of the number 0, 1, or 2 respectively.

-getSessionName

Returns the current session name. It returns an empty string if no session has been opened.

-isCaptureEnabled

Returns a value to indicate whether data capture is disabled or enabled.

Return Value	Indicates
0	Data capture is disabled.
1	Data capture is enabled.

-bookmark -add <book_mark_name>

Marks a specific time point in an InfiniTrace session. In the *Observe* session, you can go to the pre-defined bookmarks. For more information on bookmarking an InfiniTrace session, refer to the *UXE User Guide*.

-bookmark -rm <book_mark_name>

Removes a pre-set bookmark. For more information on bookmarking an InfiniTrace session, refer to the *UXE User Guide*.

-bookmark -list

Lists the defined bookmarks. For more information on bookmarking an InfiniTrace session, refer to the *UXE User Guide*.

-bookmark -goto <book_mark_name>

Searches for the specified bookmark. For more information on bookmarking an InfiniTrace session, refer to the *UXE User Guide*.

-close

Closes a session.

The `infiniTrace -close` command has the following three usage models:

- Execute the `infiniTrace -close` command from a prepare session.

UXE Command Reference Manual

Run-time Commands

This will cause the prepare session to be closed. After that, you cannot capture new data into the session. This is most common usage of this command.

- Execute the `infiniTrace -close` command from an observe session.

This will leave the DUV status at the current cycle. The user can continue to run the emulation even after executing this command. This can be used as a way to restore a design status. It is more powerful than using either `save` or `restart` command that can only restore to a cycle which has a saved snapshot. With the `infiniTrace` command, the user can restore to any cycle within the time range of an observe session.

This usage model can be used only when a design does not have a target. When a design has a target, the user can continue to run emulation after restoring design status by ensuring that the target and the DUV are in sync.

- Execute the `infiniTrace -close` command from a temporary observe session.

To enter temporary observe mode status, an `infiniTrace -goto` command is executed while in prepare session. While in temporary observe mode, the session cannot be closed unless the current cycle is at the end of the InfiniTrace session. If the current cycle is not at the end of session, execute the `infiniTrace -goto end` command before executing the `infiniTrace -close` command. The `infiniTrace -goto end` command will bring the design status to the same point before entering the temporary observe mode. The design status will be in sync with the target and you can continue to run the emulation.

Note: The `infiniTrace -close -force` command can be used to close the temporary observe session without going to end of session. In this case, the design status will not be in sync with the target.

-close [-force]

Closes an observe session when current cycle is not at the end of session. With this option, the session is closed unconditionally.

-list [-info]

Lists the name of all available data sets that can be used for observe sessions.

If the `-info` option is given, then for each data set, the listing includes the set of intervals where the data was captured.

For example:

```
XE> infiniTrace -list
```

UXE Command Reference Manual

Run-time Commands

```
it1 it2
XE> infinitracer -list -info
{it1 intervals: [100:10100]} {it2 intervals: [20100:20028100]}
```

-intervals

Lists time intervals where data capture is enabled in current session. This option can be used in either a prepare session or an observe session.

irun

Enables you to run the simulator by specifying all input files and command-line options on a single command line, when compiling and debugging designs in the SA mode.

Note: This command is not supported with vvmDebug.

The syntax for this command is:

```
irun <command_options>
```

For example, irun help <command_options>

For code-coverage collection, the following run-time switches can be used with either `irun`, `irun -xeDebug`, or `irun -hw` command:

+ixccWorkDir+<workdir>
+ixccTest+<testname>
+ixccOverwrite

For detailed description of the `irun` commands, refer to the *irun User Guide* of the INCISIVE documentation set.

+ixccWorkDir+<workdir>

Specifies the directory to store the coverage information. If `<workdir>` name is not specified, the current directory is used for storing the information. If `<workdir>` does not exist, it is created. The default directory structure is `./uxe_cov/cope/`.

+ixccTest+<testname>

Specifies the testname file for organizing the files in the `<workdir>` to prevent conflicts. If `<testname>` is not specified, a unique testname is generated. If `<testname>` already exists, an error message is issued and simulation exits. The default test name is `uxetest`.

+ixccOverwrite

Erases the existing testname file to avoid a naming conflict.

lp (Low-power Run-time Command)

Note: The `lp` command discussed below is used at the run time. The [lp \(Low-power Compile-time Command\)](#) compile-time command is discussed in [Chapter 8, “User Data Commands.”](#)

This command displays the information added for the specified `lp` command or CPF/1801 run-time control states in the CPF or IEEE 1801 file. This command can also be used to change the CPF or IEEE 1801 run-time controls.

Note: The `lp` command is not supported in vvmDebug.

The syntax for this command is:

The following command option is for both CPF and IEEE 1801:

```
lp -getdomain <signal_name>  
lp [-setRandomSeed <seed_value>]
```

The following command options are for CPF:

```
lp [assert_illegal_domain_configurations | create_isolation_rule |  
create_mode | create_mode_transition | create_nominal_condition |  
create_power_domain | create_power_mode | create_state_retention_rule |  
set_cpf_version | set_design | set_hierarchy_separator | set_instance |  
set_macro_model |  
  
lp [-lpDisable | cpfDisable [true | false | 1 | 01] |  
  
lp powerLossXDisplay [true | false | 1 | 01] |  
  
lp [-lvalue | -powerDomain | -retention | -isolation | -powerMode | -mode |  
-illegalDomainConfiguration] <lpObjectName> |  
  
lp [-llog -start [<log_file_name>] [-append1] |  
  
lp [-llog -start -buffered [<log_file_name>] [-append1] |  
  
lp [-llog -start -lamode {earliest | latest | trigger} <log_file_name>  
-posttrigger <num>] |  
  
lp [-llog -stop] |  
  
lp [-llog -flush] |  
  
lp [-llog -info [<log_file_name>]] |  
  
lp [-llog -show [-start <time>] [-end <time>] [<log_file_name>]] [-outfile  
<out_file_name>]
```

The following command options are for IEEE 1801:

UXE Command Reference Manual

Run-time Commands

```
lp -assign_supply_state -port <portName> -net <netName> -state {on | off | partialOn} [-voltage <realNumber>] |
lp -check <1801_command> [<namePattern>] |
```

The *<1801_command>* can be one of the following IEEE 1801 commands:

```
{add_port_state | add_pst_state | associate_supply_set | connect_logic_net |
connect_supply_net | connect_supply_set | create_hdl2upf_vct |
create_logic_net | create_logic_port | create_power_domain |
create_power_switch | create_pst | create_supply_net | create_supply_port |
create_supply_set | create_upf2hdl_vct | set_design_attributes |
set_design_top | set_domain_supply_net | set_isolation | set_isolation_control |
set_partial_on_translation | set_port_attributes | set_retention |
set_retention_control | set_simstate_behavior | set_related_supply_net |
upf_version}
```

```
lp -check 1801 [<objectType>] |
```

For IEEE 1801, the *<objectType>* can be one of the following:

```
{domains | logicPorts | logicNets | supplyPorts | supplyNets | supplySets |
switches | rootSupplies | equivalents | hdl2upf | upf2hdl | isolations |
retentions | supplyPortStates | pst | pstStates | powerStates}
```

```
lp -check 1801 statistic |
```

```
lp -check files |
```

```
lp -check Is1801 |
```

```
lp -check domain [<namePattern>] |
```

```
lp -check isolation [<namePattern>] |
```

```
lp -check retention [<namePattern>] |
```

```
lp -check sdl [<eventType>] |
```

For IEEE 1801, the *<eventType>* can be one of the following:

```
{global | domains | supplySets | switches | isolations | retentions |
supplyPortStates | pstStates | powerState}
```

```
lp -check sdl
```

```
lp -illegal |
```

```
lp -illegal -level {error | warning | firstwarning | ignore} |
```

```
lp -illegal -list |
```

```
lp -illegal <name> |
```

```
lp -illegal [-enable | -disable] |
```

```
lp -illegal [-enable | -disable] <assertion name ...> |
```

```
lp -illegal -global [-enable | -disable] |
```

```
lp -illegal -describe {* | type} |
```

```
lp -log -console [0 | 1] |
```

UXE Command Reference Manual

Run-time Commands

```
lp -log -file <filename> |
lp -log -eventType { [powerDomain] [supplySet] [isolation] [retention] [powerState]
[stateTransition] [portState] [pstState] [illegalIsoState] [illegalRetState]
} |
lp -log -reset |
lp -log -scope <scope_name> |
lp -log [-shm <filename>] | [-fsdb <filename>] |
lp -log -start |
lp -log -stop |
lp -log -overflow [ignore | warning | error] |
lp -log -upload {[-num_sample <n>] | [ [-start <startTime>] [-end <endTime>] ]} |
lp -metrics [-reset|-enable|-disable|-summary [-file <outputFileName>] | [-merge
<outputFileName> <inputFileName> ...] -get [-supplyNet | -supplySet | -
powerDomain | -supplyPortState | -pst | -powerState | -stateTransition | -
switchState| -isolation | -retention] <name pattern> [-ucis] |
lp -runEnable | -runDisable |
lp -value powerstate <namePattern> |
lp -value powerdomain <namePattern> |
lp -value retention <namePattern> |
lp -value isolation <namePattern> |
lp -value supplyport <namePattern> |
lp -value supplynet <namePattern> |
lp -value supplyset <namePattern> |
lp -value pst <namePattern>
```

Options of Run-Time Ip Command for CPF and IEEE 1801

-getdomain <signal_name>

Returns the domain in which the corresponding instance of a signal is present.

For CPF, the *<signal_name>* is a single signal name. It should be a scalar name. The signal name cannot be a list of signal names, a vector name, or a vector range.

For IEEE 1801, the *<signal_name>* can be a single signal name or a list of signal names. The signal name can be a scalar name, a vector name, or a vector range.

UXE Command Reference Manual

Run-time Commands

The following are some examples for IEEE 1801:

```
XE> lp -getdomain dut.clk
'dut.clk' => domain 'PD_Def'

XE> lp -getdomain dut.clk dut.and_in dut.out_0
'dut.clk' => domain 'PD_Def'
'dut.and_in' => domain 'PD_Def'
'dut.out_0[0]' => domain 'PD_Def'
'dut.out_0[1]' => domain 'PD_Def'
'dut.out_0[2]' => domain 'PD_Def'
'dut.out_0[3]' => domain 'PD_Def'
'dut.out_0[4]' => domain 'PD_Def'
'dut.out_0[5]' => domain 'PD_Def'
'dut.out_0[6]' => domain 'PD_Def'
'dut.out_0[7]' => domain 'PD_Def'

XE> lp -getdomain dut.clk dut.and_in dut.out_0 dut.out_1[3:5]
'dut.clk' => domain 'PD_Def'
'dut.and_in' => domain 'PD_Def'
'dut.out_0[0]' => domain 'PD_Def'
'dut.out_0[1]' => domain 'PD_Def'
'dut.out_0[2]' => domain 'PD_Def'
'dut.out_0[3]' => domain 'PD_Def'
'dut.out_0[4]' => domain 'PD_Def'
'dut.out_0[5]' => domain 'PD_Def'
'dut.out_0[6]' => domain 'PD_Def'
'dut.out_0[7]' => domain 'PD_Def'
'dut.out_1[3]' => domain 'PD_Def'
'dut.out_1[4]' => domain 'PD_Def'
'dut.out_1[5]' => domain 'PD_Def'

XE> lp -getdomain dut.clk dut.bank_0.and_in dut.bank_2.out dut.bank_9.out[3:5]
'dut.clk' => domain 'PD_Def'
'dut.bank_0.and_in' => domain 'PD_bank0'
'dut.bank_2.out[0]' => domain 'PD_bank2'
'dut.bank_2.out[1]' => domain 'PD_bank2'
'dut.bank_2.out[2]' => domain 'PD_bank2'
'dut.bank_2.out[3]' => domain 'PD_bank2'
'dut.bank_2.out[4]' => domain 'PD_bank2'
'dut.bank_2.out[5]' => domain 'PD_bank2'
'dut.bank_2.out[6]' => domain 'PD_bank2'
'dut.bank_2.out[7]' => domain 'PD_bank2'
'dut.bank_9.out[3]' => domain 'PD_bank9'
'dut.bank_9.out[4]' => domain 'PD_bank9'
'dut.bank_9.out[5]' => domain 'PD_bank9'
```

[*-setRandomSeed <seed_value>*]

Specifies a random seed number at run time for random power-down or power-up states. When domains are specified in the UPF or CPF files to have random power-down or power-up states, then after each power cycle within a single debug session, the power-down or power-up values will change. The pattern of the random power-down or power-up value changes is determined by the random seed used with each power cycle; the same seed number produces the same pattern. The *lp -setRandomSeed <seed_value>* command is optional. Without this command, the default seed is used.

UXE Command Reference Manual

Run-time Commands

The value of a seed might be specified in binary, octal, decimal, or hexadecimal format, prefixed by the appropriate letter. For example, '`b011`', '`o1237`', '`d089`', '`hffcd`'. If not specified, the default seed value is decimal.

Options of Run-Time Ip Command for CPF

[`assert_illegal_domain_configurations` | `create_isolation_rule` | `create_mode` | `create_mode_transition` | `create_nominal_condition` | `create_power_domain` | `create_power_mode` | `create_state_retention_rule` | `set_cpf_version` | `set_design` | `set_hierarchy_separator` | `set_instance` | `set_macro_model`]

Specifies the CPF command for which the current setting needs to be displayed. The following CPF commands can be specified as arguments with the `lp` command:

<code>assert_illegal_domain_configurations</code>	Displays the configuration of domain conditions and power mode control group conditions that are illegal.
<code>create_isolation_rule</code>	Displays the top DUT- and block-level isolation rules.
<code>create_mode</code>	Displays the top DUT modes.
<code>create_mode_transition</code>	Displays the top DUT mode transitions.
<code>create_nominal_condition</code>	Displays the top DUT-level nominal condition sets. Block-level nominal conditions are not displayed.
<code>create_power_domain</code>	Displays the top DUT- and block-level power domains, and the mapped power domains.
<code>create_power_mode</code>	Displays the top DUT-level power modes. Block-level nominal conditions are not displayed.
<code>create_state_retention_rule</code>	Displays the top DUT- and block-level state retention rules.
<code>set_cpf_version</code>	Displays the CPF version.
<code>set_design</code>	Displays the top DUT-level CPF design model and all of the design model names under it.
<code>set_hierarchy_separator</code>	Displays the hierarchy separators for the CPF commands.

UXE Command Reference Manual

Run-time Commands

set_instance	Displays all the instances that have macro or design model being applied to.
set_macro_model	Displays all the macro model names under top DUT-level CPF macro model.

For the description and syntax of the above-listed CPF commands, refer to [Chapter 12, "Common Power Format Commands."](#)

[‐]lpDisable | cpfDisable [true | false | 1 | 0]

Disables or enables the CPF support feature at run time. By default, it is enabled.

Setting the `lpDisable` or `cpfDisable` command option to:

true	Disables the CPF support feature at run time.
false	Enables the CPF support feature at run time.
0	Disables the CPF support feature at run time.
1	Enables the CPF support feature at run time.

To display whether CPF is disabled or enabled, execute the `lpDisable` or `cpfDisable` command option without setting its value.

powerLossXDisplay [true | false | 1 | 0]

Identifies the way of depicting power loss corruption in a waveform. If the value of `powerLossXDisplay` option is set to `true` or `1`, the signals in power loss are displayed with value `X`; otherwise, the actual emulator values are displayed.

[‐]value [-powerDomain | -retention | -isolation | -powerMode | -mode | -illegalDomainConfiguration] <IpObjectName>

Gets the actual value of the specified CPF object.

Any of the following values can be specified:

- `powerDomain`
- `retention`

- isolation
- powerMode
- mode
- illegalDomainConfiguration

[-]log -start [<log_file_name> [-append]]

Starts logging the CPF events in an instant log.

Note: This command cannot be used in LA mode.

If the `lp log -start` command is specified without the optional parameters, that is `<log_file_name>` and `-append`, the messages are printed to the screen while the design is running. If a log file name is specified, the CPF events are also saved to that log file. If the `-append` option is specified, the log data is appended to the log file if it already exists.

[-]log -start -buffered [<log_file_name> [-append]]

Starts logging the CPF events in the specified buffered log file. If the `-append` option is specified, the log data is appended to the buffered log file if it already exists.

Note: This command cannot be used in LA mode.

[-]log -start -lamode {earliest | latest | trigger} <log_file_name> [-posttrigger <num>]

Starts logging the CPF events in the specified buffered log file in LA mode. This command also specifies the manner in which the CPF events should be captured, that is, either earliest, latest, or trigger. If and only if trigger mode is specified, you can specify `-posttrigger` option to enable the number of CPF log events to be captured after trigger occurs. The default value is 0.

For details about these three strategies to control the logging, refer to the *Buffered Log in LA Mode* section in the *Using CPF in UXE* chapter, *UXE User Guide*.

[-]log -stop

Stops logging of the CPF events.

When a buffered log is being used, this command will also upload the remaining data in the memory buffer.

[-]log -flush

Uploads the remaining CPF events data in memory buffer when a buffered log is used in STB and SA modes. This command should be specified after the `run` command.

If this command is issued in LA mode, it is an equivalent to the `lp log -stop` command.

[-]log -info [<log_file_name>]

Lists the basic information about a log file, including the time range and number of the captured CPF events. If the log file name is not specified, the information is listed for the currently opened log file.

Note: This command can be used in an offline session as well.

[-]log -show [-start <time>] [-end <time>] [<log_file_name>][-outfile <out_file_name>]

Decrypts the log file and prints out the logged CPF event messages. If `-outfile` name is specified, the decrypted logging messages will be saved in the specified output file.

The `-start` and `-end` options can be used to specify a time range for which to print the logged CPF event messages. If the `-start` option is not specified, the first time logged in the log file is taken as the start point. If the `-end` option is not specified, the last time logged in the log file is taken as the end point. If a log file name is not specified, the currently opened log file is taken into consideration.

Note: This command can be used in an offline session as well.

Options of Run-Time Ip Command for IEEE 1801

-assign_supply_state -port <portName> -net <netName> -state {on | off | partialOn} [-voltage <realNumber>]

Updates power state and corresponding voltage of the supply port or supply net during run time.

UXE supports the OFF, FULL_ON, PARTIAL_ON, and UNDETERMINED states of the supply port or supply net. The `set_parity_on_translation` command of IEEE 1801 can define PARTIAL_ON as OFF or FULL_ON.

The simstate has six states - NORMAL, CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_STATE_ON_CHANGE, CORRUPT_STATE_ON_ACTIVITY, and NOT_NORMAL. UXE

UXE Command Reference Manual

Run-time Commands

only supports two states - NORMAL and CORRUPT. The other four states are treated as CORRUPT.

-check <1801_command> [<namePattern>]

Displays static information of the specified *<1801_command>* 1801 object or of the matched *<namePattern>* content if it matches the corresponding IEEE 1801 object hierarchical names or hierarchical member names.

The *<1801_command>* can be one of the following:

```
add_port_state  
add_pst_state  
associate_supply_set  
connect_logic_net  
connect_supply_net  
connect_supply_set  
create_hdl2upf_vct  
create_logic_net  
create_logic_port  
create_power_domain  
create_power_switch  
create_pst  
create_supply_net  
create_supply_port  
create_supply_set  
create_upf2hdl_vct  
set_design_attributes  
set_design_top  
set_domain_supply_net  
set_isolation  
set_isolation_control  
set_partial_on_translation  
set_port_attributes  
set_retention  
set_retention_control  
set_simstate_behavior  
set_related_supply_net  
upf_version
```

For the list of supported IEEE 1801 commands and options in UXE, refer to *Using IEEE 1801 to Specify Low-Power Intent on Palladium XP in UXE User Guide*.

UXE Command Reference Manual

Run-time Commands

The *<namePattern>* can include wildcards * and ?.

In UXE, the name list in the -elements, -exclude_elements, or -pins options of the 1801 commands is not bit blasted. Instead, the bits are grouped as a range. For example, -elements {dut.a[1], dut.a[2], ..., dut.a[30]} is shown as -elements {dut.a[1:30]}.

Note: This command should be issued after the `read_power_intent` command.

The IEEE 1801 object hierarchical names are *<scope>. <1801ObjectName>*. The isolation member names are *<scope>. <domain>. <isolation>*. The retention member names are *<scope>. <domain>. <retention>*. The pst state member names are *<scope>. <pst>. <pstState>*.

-check 1801 [<objectType>]

Displays complete list or the specified *<objectType>* list of power intent.

The *<objectType>* can be one of the following:

```
domains  
logicPorts  
logicNets  
supplyPorts  
supplyNets  
supplySets  
switches  
rootSupplies  
equivalents  
hdl2upf  
upf2hdl  
isolations | retentions  
supplyPortStates  
pst  
pstStates
```

Note: This command should be issued after the `read_power_intent` command.

The names in the list are *<scope>. <objectSimpleName>*. For *<objectType>*, retention, isolation and pst state member names are given in the list too. The retention and isolation member names in the list are *<scope>. <domain>. <objectSimpleName>*. The pst state member names are *<scope>. <pst>. <pstStateSimpleName>*.

-check 1801 statistic

Provides statistic information for all IEEE 1801 commands. The output of this command lists the number of times each 1801 command is read.

-check files

Displays the 1801 file names.

-check ls1801

Returns 1 if 1801 files are read in. Returns 0 if no 1801 information is present in the design.

-check domain [<namePattern>]

Displays list of interface ports of all power domains or of the matched *<namePattern>* power domains.

The *<namePattern>* can include wildcards * and ?.

The matched domain names are *<scope>. <domain>*.

In UXE, the `lp -check domain` command includes the content of the `lp -check create_power_domain` command plus lists of +receivers, +drivers and +inouts. All bus bits in those three name lists are shown as bus range.

-check isolation [<namePattern>]

Displays list of isolated pins of all isolations or of the specified *<namePattern>* isolations.

The *<namePattern>* can include wildcards * and ?.

The matched isolation names are *<scope>. <isolation>* and *<scope>. <domain>. <isolation>*.

In UXE, the `lp -check isolation` command includes the content of the `lp -check set_isolation` command plus a list of +isoPins. All bus bits in the list of +isoPins are shown as bus range.

-check retention [<namePattern>]

Displays list of sequential output pins of all retention strategies or of the matched <*namePattern*> retention strategies.

The <*namePattern*> can include wildcards * and ?.

The matched retention names are <*scope*>. <*retention*> and <*scope*>. <*domain*>. <*retention*>.

In UXE, the lp -check retention command includes the content of the lp -check set_retention command, and lists of +retainedPins and +retainedInsts. All bus bits in the +retainedPins list are shown as bus range. The +retainedInsts list shows individual instance name.

-check sdl [<eventType>]

Displays the 1801 event names based on the given event type. The 1801 event can be used in SDL expression.

The <*eventType*> can be global, domains, supplySets, switches, isolations, retentions, supplyPortStates, or pstStates.

For example:

```
XE> lp -check sdl isolations
sdl {
    <isolation> {
        power::dut.iso0 power::dut.PD_bank0.iso0
        power::dut.iso1 power::dut.PD_bank1.iso1
        power::dut.iso2 power::dut.PD_bank2.iso2
        power::dut.iso3 power::dut.PD_bank3.iso3}
    <isolation>@isolation {
        power::dut.iso0@isolation power::dut.PD_bank0.iso0@isolation
        power::dut.iso1@isolation power::dut.PD_bank1.iso1@isolation
        power::dut.iso2@isolation power::dut.PD_bank2.iso2@isolation
        power::dut.iso3@isolation power::dut.PD_bank3.iso3@isolation}
    <isolation>@corrupted {}
    <isolation>@inputDriverCorrupt {}
    <isolation>@domainCorrupt {
        power::dut.iso0@domainCorrupt power::dut.PD_bank0.iso0@domainCorrupt
        power::dut.iso1@domainCorrupt power::dut.PD_bank1.iso1@domainCorrupt
        power::dut.iso2@domainCorrupt power::dut.PD_bank2.iso2@domainCorrupt}
```

UXE Command Reference Manual

Run-time Commands

```
power::dut.iso3@domainCorrupt power::dut.PD_bank3.iso3@domainCorrupt}
}
```

-check sdl

Displays a list of 1801 conditions at run time.

For example,

```
XE> lp -check sdl
sdl {
+global {
{anyRetentionControlSignalCorrupt 1 0}
{anyRetentionControlSignalCorruptTrigger 1 0}{anyIllegalEvent 1 0}

{runLPDisable 1 0}
<supplyNet> {{}
power::dut.VDD {OFF FULL_ON PARTIAL_ON UNDETERMINED}}}
<supplyPort> {{}
power::dut.VDD {OFF FULL_ON PARITAL_ON UNDETERMINED }}}
<supplyPortState> {{}
power::dut.VDD_sw1.sw1on {1 0}} }
<domain> {{}
power::dut.PD_Def {NORMAL CORRUPT } }
<isolation> {{}
{power::dut.iso0 power::dut.PD_bank0.iso0} {NORMAL CORRUPT ISOLATED
NOT_ISOLATED}}
<isolation>@isolation {{}
{power::dut.iso0@isolation power::dut.PD_bank0.iso0@isolation} {1 0}} }
<isolation>@domainCorrupt {{}
{power::dut.iso3@domainCorrupt power::dut.PD_bank3.iso3@domainCorrupt} {1 0}} }
<retention> {{}
{power::dut.ret_9 power::dut.PD_bank9.ret_9} {NORMAL CORRUPT RETAINED
NOT_RETAINED}}
<switch> {{}
power::dut.sw_0 {sw_ON } } {
power::dut.sw_1 {sw_ON sw_OFF } } {
power::dut.sw_2 {sw_ON sw_OFF } } {
power::dut.sw_3 {sw_ON sw_OFF sw_ERR } } {
power::dut.sw_4 {sw_ON sw_OFF sw_ERR sw_ERR2 } } }
<pst> {{}
power::dut.TABLE_1 {off on } } {
power::dut.TABLE_12 {both_off both_on one_on two_on } } }
<pstStates> {{}
{power::dut.off power::dut.TABLE_1.off} {1 0}} {
{power::dut.on power::dut.TABLE_1.on} {1 0}} {
{power::dut.both_off power::dut.TABLE_12.both_off} {1 0}} {
{power::dut.both_on power::dut.TABLE_12.both_on} {1 0}} {
{power::dut.one_on power::dut.TABLE_12.one_on} {1 0}} {
{power::dut.two_on power::dut.TABLE_12.two_on} {1 0}} }}
```

-illegal

Returns a list of all valid illegal condition checks.

-illegal -level [error | warning | firstwarning | ignore]

Returns the current level of reporting the illegal condition checks or changes the reporting level.

The **error** level stops the run if an illegal event occurs. All other levels do not stop the run.

The **warning** level reports a warning when an illegal event occurs. The default level is **warning**.

The **firstwarning** level reports a warning for the illegal event only at the first time.

The **ignore** level turns off checking for the illegal event and nothing is reported. However, the events might still be logged in the 1801 event log. All other levels, except the **ignore** level, report the illegal event.

For example,

At cycle 206, 1801 illegal <name> failed.	//In ICE mode
At time 206ns, 1801 illegal <name> failed.	//In SA mode

-illegal -list

Lists all of the illegal condition checks without displaying the state as Enabled or Disabled.

The **lp -illegal -list** command does not take wildcards.

-illegal <name>

Returns a list of matching event names followed by a 1 if the event is enabled or a 0 if the event is disabled.

The **lp -illegal <name>** command takes wildcards and always displays the state.

For example, the **lp -illegal *** command returns all of the names and the states (enabled or disabled).

-illegal [-enable | -disable]

Returns the list of the enabled or disabled illegal condition checks.

Disabled assertions are not tracked in the UXE 1801 event log.

-illegal [-enable | -disable] <assertion name ...>

Enables or disables the specific illegal condition checks.

Multiple names are accepted. Wildcards are accepted and allow you to also enable or disable by domain or strategy. For example, `lp -illegal -disable *tb.dut.pdA*` disables all assertions related to the power domain.

The `lp -illegal -disable *` command disables all of the assertions.

Disabled assertions are not tracked in the UXE 1801 event log.

-illegal -global [-enable | -disable]

Enables or disables all of the assertions.

The `lp -illegal -global` command provides an additional mask. The list of enabled or disabled conditions generated through the `-illegal -global` option and the individual `-illegal [-enable | -disable]` option are combined (that is, ANDed).

-illegal -describe {* | type}

Displays description for all of the assertions or for the named assertion type. The assertion types is assertion name minus the IEEE 1801 object and extra underscore.

For example,

```
lp -illegal LPV_ISO_ENABLED_WHEN_DOMAIN_CORRUPT -describe
```

-log -console [0 | 1]

Specifies whether to print the 1801 event messages to the console or not.

If 1 is specified, all event messages are printed into the xe.msg file and to the screen. By default, the messages are printed.

If 0 is specified, no messages are printed.

-log -file <filename>

Specifies the name of the file into which the 1801 event messages are saved.

If " " or {} is specified, the 1801 event message are not saved to a file.

-log -eventType { [powerDomain] [supplySet] [isolation] [retention] [powerState] [stateTransition] [portState] [pstState] [illegalIsoState] [illegalRetState] }

Specifies a set of 1801 event types that should be printed to the screen or saved to a specified file.

The event type must be a subset of the 1801 event type specified by the following command:

```
compilerOption -add {1801LogType {[powerDomain] [supplySet] [isolation]
                           [retention] [powerState] [stateTransition] [portState]
                           [pstState] [illegalIsoState] [illegalRetState]}}
```

If the `lp -log -eventType` command is not specified, all 1801 event types specified by the `compilerOption` command are printed to the screen or saved to a specified file.

-log -reset

Discards the event data that is already captured in the memory buffer.

-log -scope <scope_name>

Limits the scope of 1801 event message.

The `<scope_name>` is full hierarchical name of an instance.

-log [-shm <filename>] | [-fsdb <filename>]

Specifies the waveform format and the name of the file used to generate an event waveform when uploading 1801 event data from the memory buffer.

-log -start

Starts the streaming log mode.

-log -stop

Stops the streaming log mode and switches to the buffered log mode.

-log -overflow [ignore | warning | error]

Specifies how to handle buffer overflow in streaming mode in LA mode or IXCOM dynamic mode where clock cannot be stopped.

If `-overflow` is set to `ignore`, the buffer overflow is ignored.

If `warning` is specified, a warning message is displayed if a buffer overflow happens during the run.

If `error` is specified, the `run` command is stopped with an error message when buffer overflow happens.

`-log -upload {[-num_sample <n>] | [-start <startTime>] [-end <endTime>] }`

Uploads 1801 event data in the memory buffer. The `lp -log -upload` command can only be used in a buffered log mode.

If the `-num_sample` option is specified, the latest `<n>` samples in the memory buffer are printed to the screen and/or saved to a file. If `0` is specified or the option is missing, all 1801 event messages are handled. One sample means that at a time point, at least one power object's state changed.

If the `-start` or `-end` option is specified, 1801 events which occur from `startTime` to `endTime` are printed to the screen or saved to a specified file. If the `-start` option is set to `0` or is missing, the `startTime` is the timestamp of the first event in the current memory buffer. If the `-end` option is set to `0` or is missing, the `endTime` is the timestamp of the last event in the current memory buffer.

Note: The `-num_sample` option cannot be used in combination with the `-start` and `-end` options.

`-metrics [-reset|-enable|-disable|-summary [-file <outputFileName>] | [-merge <outputFileName> <inputFileName> ...] | -get [-supplyNet | -supplySet | -powerDomain | -supplyPortState | -pst | -powerState | -stateTransition | -switchState| -isolation | -retention] <name pattern> [-ucis]`

Generates the coverage data using the coverage counters. The coverage counters indicate the intervals for state transitions for various power objects. The coverage counters are enabled by default.

The `-reset` option resets the coverage counters. You can reset the coverage counters after a simulation reset or a restore.

The `-disable` option disables the coverage counters from incrementing. To enable the counters, use the `-enable` option.

The `-summary` option prints the summary of the coverage counters in a text file.

The `-file` option enables you to specify an output file to save the coverage information.

The `-merge` option enables you to specify multiple input files of coverage data from multiple runs and output one file with the summed counts. This option does not read the current coverage data.

The `-get` option returns to stdout the current coverage counter information. You can also specify the power object type to print the coverage counter information. If no object types are specified, all coverage counters are printed.

The `-ucis` option generates UXE Unicov databases in the current directory. The naming convention for the Unicov database is `<prefix>_<time>.ucm` and `<prefix>_<time>.ucd`. Here, the `<prefix>` is `uxe_1801`. The `<time>` is based on current data and time in the `YYYYMMDDHHMM` format. For example, if the time while generating the Unicov database is 2016, June 18, 18:35 PM, the `<time>` would be `201606181835`.

-runEnable | -runDisable

The `-runEnable` option enables power intent at run time. The `-runDisable` option disables power intent at run time. By default, power intent is enabled. The behavior of the `-runDisable` option is as follows:

- Disables the 1801 instrumentation logic.
- Disables all `lp` run- time commands.
- Disables 1801 objects probing, that is, `probe -add<1801 objects> | power | -pwr_mode` command.

-value powerstate <namePattern>

Displays value of the matched `<namePattern>` states of power state.

The `<namePattern>` can include wildcards * and ?.

The matched power state names are `<scope>. <powerstate>`.

-value powerdomain <namePattern>

Displays value of the matched `<namePattern>` power domain state as CORRUPT or NORMAL.

The *<namePattern>* can include wildcards * and ?.

The matched domain names are *<scope>. <domain>*.

-value retention <namePattern>

Displays value of the matched *<namePattern>* retention strategy state as ON or OFF.

The *<namePattern>* can include wildcards * and ?.

The matched retention names are *<scope>. <retention>* and
<scope>. <domain>. <retention>.

-value isolation <namePattern>

Displays value of the matched *<namePattern>* isolation strategy state as ON or OFF.

The *<namePattern>* can include wildcards * and ?.

The matched isolation names are *<scope>. <isolation>* and
<scope>. <domain>. <isolation>.

-value supplyport <namePattern>

Displays value of the matched *<namePattern>* supply port as OFF, UNDETERMINED, PARTIAL_ON, or FULL_ON for power state; and, real number or UNSPECIFIED for voltage.

The *<namePattern>* can include wildcards * and ?.

The matched supply port names are *<scope>. <supplyPort>*.

-value supplynet <namePattern>

Displays value of the matched *<namePattern>* supply net as OFF, UNDETERMINED, PARTIAL_ON, or FULL_ON for power state; and, real number or UNSPECIFIED for voltage.

The *<namePattern>* can include wildcards * and ?.

The matched supply net names are *<scope>. <supplyNet>*.

UXE Command Reference Manual

Run-time Commands

-value supplyset <namePattern>

Displays value of the matched *<namePattern>* supply set state as CORRUPT or NORMAL.

The *<namePattern>* can include wildcards * and ?.

The matched supply set names are *<scope>. <supplySet>*.

-value pst <namePattern>

Displays value of the matched *<namePattern>* pst states as ACTIVE or OFF.

The *<namePattern>* can include wildcards * and ?.

The matched pst table names are *<scope>. <pst>*.

memory (in SA Mode)

Dumps, loads, or lists all memory instances in the design. The `-dump` parameter reads data from the memory instance in the design and dumps it to the file specified. Conversely, the `-load` parameter reads data from the file specified and loads it to the memory instance in the design.

During compilation, the compiler might optimize some bits in some memories for better performance if it determines that the optimization will not affect the emulation results. When you try to load a memory with some optimized bits using the `memory -load` command, the optimized bits of the memory are ignored with a warning message. When you dump a memory with optimized bits using the `memory -dump` command, the optimized bits of the memory are assumed to have a value 0 and a warning message is printed. It is possible that if you load a memory and then dump it, the dumped value of the optimized bits can be different from the loaded value.

The syntax for this command is:

```
memory -dump {<file_format> <memory_name> -file <file_name>} \
[-append] [-start <start_number>] [-end <end_number>] [-skipzero] \
[-addrRadix <radix>] [-dataRadix <radix>] [-z]

memory -load {[<file_format>] <memory_name> -file <file_name>} \
[-start <start_number>] [-end <end_number>] \
[-fileoffset <offset_number> | continue]

memory -list

memory -set {-all | <memory_name>...}

memory -reset {-all | <memory_name>...}

memory -setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>] \
-value {<value> | all0 | all1}

memory -setvalue -value {<value> | all0 | all1} [-mask <value>]

memory -setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>] \
-random <seed>

memory -getFileOffset <file_name>

memory -sparse_mem_info [<mem_name>]
```

UXE Command Reference Manual

Run-time Commands

-dump {<file_format> <memory_name> -file <file_name>}

[**-append**] [**-start <start_number>**] [**-end <end_number>**] [**-skipzero**]

[**-addrRadix <radix>**] [**-dataRadix <radix>**] [**-z**]

Dumps the specified memory with optimized bits in the specified file format.

Argument	Type	Description
<i><file_format></i>	Required	<p>Specifies the format of the output file in which the data will be dumped. The supported file formats are:</p> <ul style="list-style-type: none">■ %pd_b: Palladium ASCII format, radix bin■ %pd_o: Palladium ASCII format, radix oct■ %pd_d: Palladium ASCII format, radix dec■ %pd_h: Palladium ASCII format, radix hex■ %pd_memtran: memtran format■ %pd_raw: raw format■ %pd_raw2: raw2 format■ %readmemb: readmemb format■ %readmemh: readmemh format■ %b: IES format, radix bin■ %o: IES format, radix oct■ %h: IES format, radix hex■ %dn: Denali format <p>Refer to File Formats to know more about the supported formats for this command.</p>
<i><memory_name></i>	Required	Reads from the memory instance specified in <i><memory_name></i> .
-file <i><file_name></i>	Required	Specifies the name of the file where the data needs to be dumped. If the file you specify contains some data, it will be overwritten.

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-append	Optional	Appends new data to an existing memory content file. This option can be used only with files of <code>raw2</code> format.
[-start <start_number>] [-end <end_number>]	Optional	The <code>-start <start_number></code> option specifies the starting memory address. The <code>-end <end_number></code> option specifies an ending memory address. If you do not specify a starting address, the read operation starts from the first address in memory. If you do not specify an ending address, the read operation ends at the last address in memory. The <code><start_number></code> and <code><end_number></code> arguments must fall within the address range of the memory instance specified. The radix for <code><start_number></code> and <code><end_number></code> follows the C/C++ convention. For example, <code>0x12</code> is a hex number, <code>12</code> is a decimal number, and <code>012</code> is an octal number.
-skipzero	Optional	When this option is specified, the memory dump will not include lines with all zero data. This option works only with the Palladium ASCII format.
-addrRadix <radix>	Optional	Dumps the memory data to the specified file of Denali memory format with address values of the specified <code><radix></code> . The <code><radix></code> can be either of the following: <ul style="list-style-type: none">■ 2 for binary■ 8 for octal■ 10 for decimal■ 16 for hexadecimal, which is the default <p>Note: This option is supported only when the <code><file_format></code> argument is set to <code>%dn</code>.</p>

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-dataRadix <i><radix></i>	Optional	Dumps the memory data to the specified file of Denali memory format with data values of the specified <i><radix></i> . The <i><radix></i> can be either of the following: <ul style="list-style-type: none">■ 2 for binary■ 8 for octal■ 10 for decimal■ 16 for hexadecimal, which is the default <p>Note: This option is supported only when the <i><file_format></i> argument is set to %dn.</p>
-z	Optional	Saves the memory data file in a compressed format, which can be uncompressed using the UNIX uncompress or zcat commands. <p>Note: This option can be used only for raw and raw2 file formats.</p> <p>This option cannot be used in combination with the -append or -fileoffset options for memory streaming using the raw2 file format.</p>

Note: In the SA mode, this option is applicable only to DUV memories.

UXE Command Reference Manual

Run-time Commands

-load {[<file_format>] <memory_name> -file <file_name>}

[-start <start_number>] [-end <end_number>]

[-fileoffset <offset_number> | continue]

Loads the specified memory into the specified file.

Argument	Type	Description
<memory_name>	Required	Loads data from a file into memory specified in <memory_name> argument.
-file <file_name>	Required	Specifies the name of the file from where the data needs to be loaded into the memory.
[-start <start_number>] [-end <end_number>]	Optional	The -start <start_number> option specifies the starting memory address. The -end <end_number> option specifies an ending memory address. If you do not specify a starting address, the load operation starts from the first address in memory. If you do not specify an ending address, the load operation ends at the last address in memory. The <start_number> and <end_number> arguments must fall within the address range of the memory instance specified. The radix for <start_number> and <end_number> follows the C/C++ convention. For example, 0x12 is a hex number, 12 is a decimal number, and 012 is an octal number.

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
<code><file_format></code>	Optional	<p>Specifies the format in which the data needs to be loaded in the memory. The format needs to be specified only when the file is in <code>%pd_raw2</code>, <code>%readmemb</code>, <code>%readmemh</code>, or <code>%dn</code> format.</p> <p>Note: The <code>memory -dump</code> command must specify the <code><file_format></code>. It is optional for the <code>memory -load</code> command if the file format can be identified from the memory content. However, if the file format cannot be derived from the file content, for example <code>%readmemb</code>, <code>%readmemh</code>, <code>%pd_raw2</code>, and <code>%dn</code> format, it is mandatory to specify <code><file_format></code> with the <code>memory -load</code> command.</p>
<code>-fileoffset <offset_number> continue</code>	Optional	<p>If you are using this option, ensure that this option is the first option specified with the <code>memory -load</code> command and is followed by the <code><memory_name></code> option.</p> <p>Other formats can identify themselves by the file content. However, you can still specify format, but that should be consistent with the file. Refer to File Formats to know about the supported file formats for this command.</p> <p>This option can only be used with <code>raw2</code> file format.</p> <p>Specifies a file offset that indicates the start point to load from. The <code><offset_number></code> can be a number that indicates the file offset in bytes. If the offset value is <code>continue</code>, the file offset will be set to the memory location where last data was loaded in previous load operation. The software remembers a offset for each file name.</p>

Note: This option is not supported with `vvmDebug`.

-list

Returns the names of all the memory instances in the DUV.

-set {-all | <memory_name>...}

Sets the content of all memory instances, or the specified memory instance to all one.

Note: In the SA mode, this command is applicable only to DUV memories. This option is not supported with vvmDebug.

-reset {-all | <memory_name>...}

Sets the content of all memory instances, or the specified memory instance to all zero.

Note: In the SA mode, this command is applicable only to DUV memories. This option is not supported with vvmDebug.

-setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>]

-value {<value> | all0 | all1}

Initializes the specified range of memory instances, or all the memory instances, to a given value.

The `-start` and `-end` options specify the range of memory addresses for which this value is set. This range of memory addresses applies to all the memory instances specified in the list, regardless of the size of these memory instances. If the `-start` and `-end` options are not specified with the command, the default start and end are taken from the beginning and end of each memory instance specified in the list.

The value can be set to 0, 1, or any other value. To set a value other than 0 and 1, ensure that the value is specified in a format that follows Verilog, VHDL, or C++ (0x hex only) convention.

Note: If the data width of the value being set is less than the memory width, MSBs of the memory word are filled with 0. If the data width is larger than the memory width, MSBs of data are ignored. The `memory -setvalue` command is not supported with vvmDebug.

-setvalue -value {<value> | all0 | all1} [-mask <value>]

Overwrites the bit position of the memory words to the specified value as indicated by the specified mask value.

The bit position of the mask value indicates whether that bit position of a memory word should be overwritten. A value 1 specifies that bit of the memory word should be written by the corresponding bit as specified by the `-value` option. A value 0 specifies that bit of the memory word should not be overwritten.

The `-mask` is not supported with the `memory -all` and `memory -load` commands

-setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>]

-random <seed>

Initializes the specified range of memory instances, or all the memory instances, to a random value.

The `-start` and `-end` options specify the range of memory addresses that are set. If these options are not specified, the default start and end are taken from the beginning and end of each memory instance specified in the list.

The `-random <seed>` option sets the memory words with random numbers, based on the integer value specified as the seed number in the command.

Note: If the same memory list, random seed, and start and end addresses are used, the `memory -setvalue` command sets the memory instances to the same content every time the command is run. The `memory -setvalue` command is not supported with `vvmDebug`.

-getFileOffset <file_name>

This command can only be used with `raw2` file format.

Returns the file offset number, in bytes, for the specified file name.

Returns a value of `-1` if the file offset has reached the end of the file.

Note: This option is not supported with `vvmDebug`.

-sparse_mem_info [<mem_name>]

Provides information for sparse memories that are specified by \$ixc_ctrl("sparse_mem") at the compile time. If no memory name is specified, the -sparse_mem_info option lists all IXCOM compiled sparse memories. If a memory name is specified, the -sparse_mem_info option shows the sparse memory's width, depth, default value, physical pages, and pages in use.

Note: This option is not supported with vvmDebug.

memory (in ICE Mode)

Dumps, loads, or lists all memory instances in the design. The `-dump` parameter reads data from the memory instance in the design and dumps it to the file specified. Conversely, the `-load` parameter reads data from the file specified and loads it to the memory instance in the design.

Before using this command in the ICE mode, execute the [configPM](#) command.



The output of the memory command can be very difficult to read if the design has a large number of memories. Below is an example of Tcl code that prints each item in a separate line:

```
set memlist [memory]
foreach item $memlist {
    puts $item
}
```

During compilation, the compiler might optimize some bits in some memories for better performance if it determines that the optimization will not affect the emulation results. When you try to load a memory with some optimized bits using the `memory -load` command, the optimized bits of the memory are ignored with a warning message. When you dump a memory with optimized bits using the `memory -dump` command, the optimized bits of the memory are assumed to have a value 0 and a warning message is printed. It is possible that if you load a memory and then dump it, the dumped value of the optimized bits can be different from the loaded value.

Executing the `memory` command without any parameters returns the names of all the memory instances in the design.

The syntax for this command is:

```
memory -dump {[<file_format>] <memory_name> -file <file_name>} \
        [-append] [-start <start_number>] [-end <end_number>] [-nooverwrite] \
        [-skipzero] [-addrReg <register_name>] [-clearAddrReg] \
        [-full <number>] [-addrRadix <radix>] [-dataRadix <radix>] [-z] [-gz] \
        [-dir <directory_name>] [-background | -bg] [-screen] [-lines <n>]

memory -load {[<file_format>] <memory_name> -file <file_name> [-retainValue] \
        [-retainValueOffset]} \
        [-start <start_number>] [-end <end_number>] \
        [-fileoffset <offset_number> | continue] [-dir <directory_name>] \
        [-background|-bg] [-nofill] [-nochecksize]

memory -list
```

UXE Command Reference Manual

Run-time Commands

```
memory -info {-all | <memory_name>...} [-help]
memory -set {-all | <memory_name>...}
memory -reset {-all | <memory_name>...}
memory -setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>] \
           -value {<value> | all0 | all1}
memory -setvalue -value {<value> | all0 | all1} [-mask <value>]
memory -setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>] \
           -random <seed>
memory -getFileOffset <file name>
memory -foreground | -fg
memory -sparse_mem_info [-v] <mem_name>
memory -sparse_mem_info -usedpages [-v] <mem_name>
memory -sparse_mem_info -statistic [-v] [-byusage]
memory -sparse_mem_info -overflow
```

Following commands are used to manage the global options:

```
memory -opt <global_options>
memory -optrm <global_options>
memory -opt
memory -optclear
```

The global options stack provides a convenient way to maintain several sets of global option settings. At the beginning, the global options stack has only one level. To create more levels and manage them, use the following commands:

```
memory -optstack
memory -optpush
memory -optpop
memory -optuse [<n>]
```

UXE Command Reference Manual

Run-time Commands

-dump {<file_format> <memory_name> -file <file_name>}

[**-append**] [**-start <start_number>**] [**-end <end_number>**] [**-nooverwrite**]
[**-skipzero**] [**-addrReg <register_name>**] [**-clearAddrReg**]
[**-full <number>**] [**-addrRadix <radix>**] [**-dataRadix <radix>**] [**-z**] [**-gz**]
[**-dir <directory_name>**] [**-background | -bg**] [**-screen**] [**-lines <n>**]

Dumps the specified memory with optimized bits in the specified file format.

Argument	Type	Description
<i><file_format></i>	Required	<p>Specifies the format of the output file in which the data will be dumped. The supported file formats are:</p> <ul style="list-style-type: none">■ %pd_b: Palladium ASCII format, radix bin■ %pd_o: Palladium ASCII format, radix oct■ %pd_d: Palladium ASCII format, radix dec■ %pd_h: Palladium ASCII format, radix hex■ %pd_memtran: memtran format■ %pd_raw: raw format■ %pd_raw2: raw2 format■ %readmemb: readmemb format■ %readmemh: readmemh format■ %b: IES format, radix bin■ %o: IES format, radix oct■ %h: IES format, radix hex■ %dn: Denali format <p>Refer to File Formats to know more about the supported formats for this command.</p>
<i><memory_name></i>	Required	Reads from the memory instance specified in <i><memory_name></i> .

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-file <i><file_name></i>	Required	Specifies the name of the file where the data needs to be dumped. If the file you specify contains some data, it will be overwritten.
-append	Optional	Appends new data to an existing memory content file. This option can be used only with files of <code>raw2</code> format.
[-start <i><start_number></i>] [-end <i><end_number></i>]	Optional	The <code>-start <start_number></code> option specifies the starting memory address. The <code>-end <end_number></code> option specifies an ending memory address. If you do not specify a starting address, the read operation starts from the first address in memory. If you do not specify an ending address, the read operation ends at the last address in memory. The <i><start_number></i> and <i><end_number></i> arguments must fall within the address range of the memory instance specified. The radix for <i><start_number></i> and <i><end_number></i> follows the C/C++ convention. For example, 0x12 is a hex number, 12 is a decimal number, and 012 is an octal number.
-nooverwrite	Optional	Aborts the memory dump operation when a dump file already exists. Without the <code>-nooverwrite</code> option, the memory dump operation overwrites the existing dump file with the same name.
-skipzero	Optional	When this option is specified, the memory dump will not include lines with all zero data. This option works only with the Palladium ASCII format.
-addrReg <i><register_name></i>	Optional	Defines the address range to dump, which is from 0 to the current value of specified register.



This option cannot be used with the -start and -end options, otherwise the memory -dump command fails with an error condition.

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-clearAddrReg	Optional	This option is used with the -addrReg option to reset the register value to 0 after dumping the memory.
-full <number>	Optional	This option is used with the -addrReg option, where <number> ranges from 0 to 100 indicating the percentage of memory. This option specifies that memory should be dumped only if the memory is <number> percent full.
-addrRadix < <i>radix</i> >	Optional	Dumps the memory data to the specified file of Denali memory format with address values of the specified < <i>radix</i> >. The < <i>radix</i> > can be either of the following: <ul style="list-style-type: none">■ 2 for binary■ 8 for octal■ 10 for decimal■ 16 for hexadecimal, which is the default <p>Note: This option is supported only when the <<i>file_format</i>> argument is set to %dn.</p>
-dataRadix < <i>radix</i> >	Optional	Dumps the memory data to the specified file of Denali memory format with data values of the specified < <i>radix</i> >. The < <i>radix</i> > can be either of the following: <ul style="list-style-type: none">■ 2 for binary■ 8 for octal■ 10 for decimal■ 16 for hexadecimal, which is the default <p>Note: This option is supported only when the <<i>file_format</i>> argument is set to %dn.</p>

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-z	Optional	<p>Saves the memory data file in a compressed format, which can be uncompressed using the UNIX <i>uncompress</i> or <i>zcat</i> commands.</p> <p>Note: This option can be used only for <i>raw</i> and <i>raw2</i> file formats.</p> <p>This option cannot be used in combination with the <i>-append</i> or <i>-fileoffset</i> options for memory streaming using the <i>raw2</i> file format.</p>
-gz	Optional	<p>Saves the memory data file in *.<i>.gz</i> format using the <i>gzip</i> file compression utility. The <i>memory -load</i> command uses the <i>gunzip</i> utility to read the *.<i>.gz</i> file.</p> <p>Note:</p> <ul style="list-style-type: none">■ This option enables you to use the <i>gzip</i> compression on SuSE 11 platform.■ This option can be used only for <i>raw</i> and <i>raw2</i> file formats.■ This option cannot be used in combination with the <i>-append</i> or <i>-fileoffset</i> options for memory streaming using the <i>raw2</i> file format.
-dir < <i>directory_name</i> >	Optional	Defines the directory name, < <i>directory_name</i> >, in which memory data files are located. If the memory data file in a <i>memory</i> command is specified with relative path, that is, not starting with "/", the file is assumed to be in this directory.

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-background -bg	Optional	<p>Specifies that the memory operation will be added in a task queue and the actual operation will be performed in background. This option is used with both <code>memory -dump</code> and <code>memory -load</code> commands.</p> <p>You can run the memory dump and load operations in batch mode for significant performance improvement. In the best case, you can expect performance gain as high as 2X.</p> <p>Use the <code>-background</code> or <code>-bg</code> option of the <code>memory</code> command to a group of <code>memory -load</code> and <code>memory -dump</code> commands to run them in batch mode in the background. The command will immediately return without waiting the memory operation to finish. These dump and load operations will be put in a task queue and will be executed in a pipeline, which can improve performance. However, you will not get any performance gain if you have only one memory load or dump operation in a batch.</p>
		<p>Note: This option is not supported in SA mode.</p>
-screen	Optional	<p>Outputs the memory content to the screen.</p> <p>Note: This option can only be used for text memory formats and is available only in ICE mode.</p>
-lines <n>	Optional	<p>Specifies the first <n> number of lines to print to the screen. The default value is 1000.</p> <p>Note: This option can only be used for text memory formats and is available only in ICE mode.</p>

Note: In the SA mode, this option is applicable only to DUV memories.

UXE Command Reference Manual

Run-time Commands

-load {[<file_format>} <memory_name> -file <file_name> [-retainValue] [-retainValueOffset]}

[-start <start_number>] [-end <end_number>]

[-fileoffset <offset_number> | continue] [-dir <directory_name>]

[-background|-bg] [-nofill] [-nochecksize]

Loads the specified memory into the specified file.

Argument	Type	Description
<memory_name>	Required	Loads data from a file into memory specified in <memory_name> argument.
-file <file_name>	Required	Specifies the name of the file from where the data needs to be loaded into the memory.

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
[-retainValue]	Optional	<p>Retains the values for the unspecified data bits. This option is applicable to the following formats:</p> <ul style="list-style-type: none">■ %readmemh: readmemh format■ %readmemb: readmemb format■ %h: IES format, radix hex■ %o: IES format, radix oct■ %b: IES format, radix bin■ %pd_h: Palladium ASCII format, radix hex■ %pd_o: Palladium ASCII format, radix oct■ %pd_b: Palladium ASCII format, radix bin <p>For example, an 8-bit width file has the following values:</p> <p>1111 3333 2222 5555 4444 6666</p> <p>The <code>memory -load %readmemh mem -file load.file -retainValue</code> command will load the following values to the memory. The xxxx values will not be written.</p> <p>xxxx 1111 3333 2222 5555 4444 xxxx 6666</p>

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
<code>[-retainValueOff set]</code>	Optional	<p>Retains the values for the unspecified data bits and also shifts the specified bits to the left. This option applies only to the first line of the <code>%readmemh</code> file.</p> <p>For example, an 8-bit width file has the following values:</p> <p>1111 3333 2222 5555 4444 6666</p> <p>The <code>memory -load %readmemh mem -file load.file -retainValue -retainValueOffset</code> command will load the following values to the memory. The <code>xxxx</code> values will not be overwritten.</p> <p>1111 <code>xxxx</code> 3333 2222 5555 4444 <code>xxxx</code> 6666</p>
<code>[-start <start_number>] [-end <end_number>]</code>	Optional	<p>The <code>-start <start_number></code> option specifies the starting memory address. The <code>-end <end_number></code> option specifies an ending memory address. If you do not specify a starting address, the load operation starts from the first address in memory. If you do not specify an ending address, the load operation ends at the last address in memory.</p> <p>The <code><start_number></code> and <code><end_number></code> arguments must fall within the address range of the memory instance specified. The radix for <code><start_number></code> and <code><end_number></code> follows the C/C++ convention. For example, 0x12 is a hex number, 12 is a decimal number, and 012 is an octal number.</p>

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
<code><file_format></code>	Optional	<p>Specifies the format in which the data needs to be loaded in the memory. The format needs to be specified only when the file is in <code>%pd_raw2</code>, <code>%readmemb</code>, <code>%readmemh</code>, or <code>%dn</code> format.</p> <p>Note: The <code>memory -dump</code> command must specify the <code><file_format></code>. It is optional for the <code>memory -load</code> command if the file format can be identified from the memory content. However, if the file format cannot be derived from the file content, for example <code>%readmemb</code>, <code>%readmemh</code>, <code>%pd_raw2</code>, and <code>%dn</code> format, it is mandatory to specify <code><file_format></code> with the <code>memory -load</code> command.</p>
 <i>Important</i>		<p>If you are using this option, ensure that this option is the first option specified with the <code>memory -load</code> command and is followed by the <code><memory_name></code> option.</p> <p>Other formats can identify themselves by the file content. However, you can still specify format, but that should be consistent with the file. Refer to File Formats to know about the supported file formats for this command.</p>
<code>-fileoffset <offset_number> continue</code>	Optional	<p>This option can only be used with <code>raw2</code> file format.</p> <p>Specifies a file offset that indicates the start point to load from. The <code><offset_number></code> can be a number that indicates the file offset in bytes. If the offset value is <code>continue</code>, the file offset will be set to the memory location where last data was loaded in previous load operation. The software remembers a offset for each file name.</p>
<code>-dir <directory_name></code>	Optional	<p>This option is used in the same way as with <code>memory -dump</code> command. For detailed description of this option, refer to the <code>[-dir <directory_name>]</code> optional parameter of the <code>memory -dump</code> command.</p>

UXE Command Reference Manual

Run-time Commands

Argument	Type	Description
-background -bg	Optional	This option is used in the same way as with <code>memory -dump</code> command. For detailed description of this option, refer to the <code>[-background -bg]</code> optional parameter of the <code>memory -dump</code> command. Note: This option is not supported in SA mode.
-nofill	Optional	Specifies that all the unspecified addresses should be kept unchanged for the following file formats: <ul style="list-style-type: none">■ <code>%pd_h</code>■ <code>%pd_d</code>■ <code>%pd_o</code>■ <code>%pd_b</code>■ <code>%pd_raw2</code> If the <code>-nofill</code> parameter is not specified, all the unspecified addresses are assigned zero value.
-nochecksize	Optional	Suppresses the errors that are generated on execution of the <code>memory -load</code> command when either of the following is true: <ul style="list-style-type: none">■ The depth of the memory file is more than the depth of the memory instance.■ The width of the memory file is more than the width of the memory instance.

Note: This option is not supported with `vvmDebug`.

-list

Returns the names of all the memory instances in the design.

Note: In SA mode, use the `memory -list` command in place of the `memory -info -all` command.

-info {-all | <memory_name>...} [-help]

The `-info -all` option displays information about all the memory instances.

The `-info <memory_name>` option displays information about the specified memory instance. You can specify multiple memory instances.

The `-info -help` option displays help information

Note: The `-info` option is valid only for the ICE mode. In the SA mode, use the `ncsim describe` command in place of `memory -info`. In place of `memory -info -all` command, use the `memory -list` command.

For example:

```
XE> memory -info -all
Memory m0.m_mema: reg[0:31] m0.m_mema[0:1048575] depth=1048576 Type: MERGE
Memory m0.m_memb: reg[32:63] m0.m_memb[0:1048575] depth=1048576 Type: MERGE
Memory m0.m_memf: reg[31:0] m0.m_memf[0:1048575] depth=1048576 Type: MPX
Memory m0.m_memz: reg[31:0] m0.m_memz[0:1048575] depth=1048576 Type: MPX
```

-set {-all | <memory_name>...}

Sets the content of all memory instances, or the specified memory instance to all one.

Note: In the SA mode, this command is applicable only to DUV memories.

-reset {-all | <memory_name>...}

Sets the content of all memory instances, or the specified memory instance to all zero.

Note: In the SA mode, this command is applicable only to DUV memories. This option is not supported with vvmDebug.

-setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>]

-value {<value> | all0 | all1}

Initializes the specified range of memory instances, or all the memory instances, to a given value.

The `-start` and `-end` options specify the range of memory addresses for which this value is set. This range of memory addresses applies to all the memory instances specified in the

list, regardless of the size of these memory instances. If the `-start` and `-end` options are not specified with the command, the default start and end are taken from the beginning and end of each memory instance specified in the list.

The value can be set to 0, 1, or any other value. To set a value other than 0 and 1, ensure that the value is specified in a format that follows Verilog, VHDL, or C++ (0x hex only) convention.

Note: If the data width of the value being set is less than the memory width, MSBs of the memory word are filled with 0. If the data width is larger than the memory width, MSBs of data are ignored. The `memory -setvalue` command is not supported with `vvmDebug`.

-setvalue {-value <value> | all0 | all1} [-mask <value>]

Overwrites the bit position of the memory words to the specified value as indicated by the specified mask value.

The bit position of the mask value indicates whether that bit position of a memory word should be overwritten. A value 1 specifies that bit of the memory word should be written by the corresponding bit as specified by the `-value` option. A value 0 specifies that bit of the memory word should not be overwritten.

The `-mask` is not supported with the `memory -all` and `memory -load` commands.

-setvalue {-all | <list_of_instances>...} [-start <addr1>] [-end <addr2>]

-random <seed>

Initializes the specified range of memory instances, or all the memory instances, to a random value.

The `-start` and `-end` options specify the range of memory addresses that are set. If these options are not specified, the default start and end are taken from the beginning and end of each memory instance specified in the list.

The `-random <seed>` option sets the memory words with random numbers, based on the integer value specified as the seed number in the command.

Note: If the same memory list, random seed, and start and end addresses are used, the `memory -setvalue` command sets the memory instances to the same content every time the command is run. The `memory -setvalue` command is not supported with `vvmDebug`.

-getFileOffset <file_name>

This command can only be used with raw2 file format.

Returns the file offset number, in bytes, for the specified file name.

Returns a value of -1 if the file offset has reached the end of the file.

Note: This option is not supported with vvmDebug.

-foreground | -fg

Used to wait for all background tasks to finish. If there is any error in the background tasks, the memory -foreground command generates an error condition.

By default, if a memory -dump or memory -load command is issued without the -background parameter, it is executed as a foreground task. A memory command without the -background parameter will first call the memory -foreground command to wait for all background tasks to finish before starting the new foreground task.

Note: This command is not supported in SA mode. It is, however, supported with vvmDebug.

-sparse_mem_info [-v] <mem_name>

Provides information such as, page size, number of virtual pages, number of physical pages, and number of used physical pages for the specified <mem_name> sparse memory. Using the -v option returns the information in a more user-friendly format.

Note: This option is not supported with vvmDebug.

-sparse_mem_info -usedpages [-v] <mem_name>

Provides a list of page numbers for all used virtual pages, that is, virtual pages that do not have the default value for the specified <mem_name> sparse memory. Note that the list might be very long. Using the -v option returns the information in a more user-friendly format.

Note: This option is not supported with vvmDebug.

-sparse_mem_info -statistic [-v] [-byusage]

Provides statistic information for all sparse memories. The output of this command lists the percentage of used physical pages for each memory name. The list is sorted by memory name. If you specify the **-byusage** option, the list is sorted by usage.

Note: This option is not supported with vvmDebug.

-sparse_mem_info -overflow

Provides a list of overflowed sparse memory instances.

Note: This option is not supported with vvmDebug.

Global Options Commands

When some parameters of the `memory` command are common to a group of `memory` commands, you can define these parameters as global options, which will apply to all the subsequent `memory` commands. This simplifies the command line for a group of `memory` commands.

If a global option is contradicting a specific `memory` command option, the specific `memory` command option is applied.

Following are the global options for the `memory` command:

- `-append`: used with the `memory -dump` command
- `-clearAddrReg`: used with the `memory -dump` command
- `-full`: used with the `memory -dump` command
- `-skipzero`: used with the `memory -dump` command
- `-z`: used with both `memory -dump` and `memory -load` commands
- `-dir`: used with both `memory -dump` and `memory -load` commands
- `-background` or `-bg`: used with both `memory -dump` and `memory -load` commands

Following commands are used to manage the global options:

- `memory -opt <global_options>`
- `memory -optrm <global_options>`

- memory -opt
- memory -optclear

memory -opt <global_options>

Specifies the global options to be applied to the subsequent memory commands. The specified options are added to the existing set of global options. If the specified global option already exists in the current setting, it is replaced with the new value. For example, following command will add the -append and -dir /usr/me/data global options. If -dir is already set, it is replaced with the new value /usr/me/data:

```
memory -opt -append -dir /usr/me/data
```

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

memory -oprm <global_options>

Removes the specified global options. For example, the following command will remove the -z and -dir global options from current settings:

```
memory -oprm -z -dir
```

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

memory -opt

Returns a list of global options applied in the current settings. For instance, for the example used above, the `memory -opt` command will return the following:

```
-append -dir /usr/me/data
```

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

memory -optclear

Clears all the global options from the current settings.

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

Global Options Stack Commands

The global options stack provides a convenient way to maintain several sets of global option settings. Each set of global option settings is called one level of the stack. The top level of the

UXE Command Reference Manual

Run-time Commands

stack indicates the current global options. The commands specified in the [Global Options Commands](#) section are used to manage the top level of the stack.

At the beginning, the global options stack has only one level. To create more levels and manage them, use the following commands:

- [memory -optstack](#)
- [memory -optpush](#)
- [memory -optpop](#)
- [memory -optuse \[<n>\]](#)

memory -optstack

Shows the value of all levels of the global options stack. Each level is marked with a number. The top level of the stack is indicated by number 1. For example, the following command shows that there are two levels of the global options stack, in which (1) -append -skipzero is the top level of the stack and is the current setting of global options:

```
XE> memory -optstack
(1) -append -skipzero
(2) -z
```

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

memory -optpush

Pushes a new level above the top level of the stack, which is a duplication of the current top level of the stack. This is the command to create a new set of global options. After pushing, you can change global options at this new level and the new level becomes the current global options setting.

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

memory -optpop

Pops a level of global options stack. The top level is removed and the second level becomes the new top level. The number of levels on the stack is decreased by one. The `memory -optpop` will not do anything if there is only one level on the stack. This command is used to remove a set of global options.

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.

memory -optuse [<n>]

Brings the n^{th} level of the global options stack to the top level, so that it becomes the new top. The order of other levels of the stack remains the same. The default value of number $<n>$ is 2. If you do not specify a value of $<n>$, the top and the second level of the stack will swap. This command is used to select a particular set of global options as the current settings.

Note: This command is not supported in SA mode. However, it is supported with vvmDebug.



While managing the global options stack, you can only remove the top level of the stack. You cannot directly remove a particular level (other than the top level) from the stack. Therefore, if you want to remove the n^{th} level of the stack, you need to first bring the n^{th} level to the top level using the `memory -optuse <n>` command, and then remove it using the `memory -optpop` command.

Examples

```
# Initialize the debugger and use the '/mydesign/design1' design database directory
debug /mydesign/design1
# Download design to 'samurai' emulation host
host samurai
download
# Configure probes in Vector Debug mode
configPM -vd
```

After downloading the design as illustrated in the above commands, you can execute the different `memory` command options as shown in the following examples:

- List all memory instances

```
memory
```

- Load to a memory instance

```
memory -load RAM8X32 -file RAM8X32.init
```

- Read from a memory instance

```
memory -dump %pd_h ROM8X64 -file ROM8X64.content
```

- Read from memory instance M1 and write contents to Denali memory format file dump.dat. Address and data values in hexadecimal.

```
memory -dump %dn M1 -file dump.dat
```

- Read from memory instance M1 and write contents to Denali memory format file dump.dat. Address values in decimal and data values in hexadecimal.

UXE Command Reference Manual

Run-time Commands

```
memory -dump %dn M1 -file dump.dat -addrRadix 10 -dataRadix 16
```

- Read from Denali memory file load.dat and load content to memory instance M1

```
memory -load %dn M1 -file load.dat
```

- Print the first 50 lines of memory M1 in pd_h format

```
memory -dump %pd_h M1 -screen -lines 50
```

- Write value 0xabcd to memory instance M1 for all addresses

```
memory -setvalue M1 -value 0xabcd
```

- Write value 0xabcd to memory instance M1 for addresses 16 to 32

```
memory -setvalue M1 -value 0xabcd -start 16 -end 32
```

- Write value 0xabcd to memory instances M1 and M2 for all addresses

```
memory -setvalue M1 M2 -value 0xabcd
```

Example of Batch Mode Operation

Following is a sample XEL script containing commands to dump 10 memories:

```
$cat dump_cpu_memes.xel
memory -dump %pd_raw CPU.mem1 -file mem1.dump.raw -addrReg CPU.mem1_waddr[12:0]
memory -dump %pd_raw CPU.mem2 -file mem2.dump.raw -addrReg CPU.mem2_waddr[12:0]
...
memory -dump %pd_raw CPU.mem10 -file mem10.dump.raw -addrReg CPU.mem10_waddr[12:0]
```

In the above XEL script, address register are used to specify address range.

Now, the following tasks need to be performed:

- Dump all memories that are more than 90% full to the /usr/my/dump1 directory.
- Clear the address register after dumping.
- Compress the data file.

The following XEL commands are used to perform the above-stated tasks:

```
# push a new level of global options and modify global options for current dump
# need to specify -background for batch operation
memory -optpush
memory -opt -z -full 90 -clearAddrReg -dir /usr/my/dump1 -background
# run the operations in background
source dump_cpu_memes.xel
# wait background tasks done
memory -foreground
```

UXE Command Reference Manual

Run-time Commands

```
# run more cycles and decide to dump again to a different directory
run 1000000
# only need to change directory, other options will remain the same
memory -opt -dir /usr/my/dump2
# dump again
source dump_cpu_mems.xel
memory -foreground
# done with dumping, restore global options
memory -optpop
```

Logical and Physical Memory Addresses

Memories are two-dimensional arrays of DxW bits, where D is the depth and W is the width. Each row of the array is called a word. There are D words in a DxW memory and each word is W bit wide.

For example, you can define memories in Verilog as follows:

```
reg [10:0] mem1 [10:20];
reg [2:7] mem2 [100:1];
```

Since the logical addresses are consistent with the memory definitions in Verilog source code, memory words can be specified in a natural way. If a memory is defined as `mem [a1:a2]` in Verilog source code, the memory words are addressed from `a1` up to or down to `a2`. The first memory word is `mem [a1]` and the last memory word is `mem [a2]`.

Following are the formulas to translate logical addresses into physical addresses and vice-versa (given the memory definition of `mem [a1:a2]`):

$$\text{logicAddr} = \text{phyAddr} + \min(a1, a2)$$

$$\text{phyAddr} = \text{logicAddr} - \min(a1, a2)$$

Note: You must use xeCompile to compile the design if they are using logical addresses because xeCompile saves the logical address information in the emulator database. If the logical address information is not available in the emulator database for the memory you are trying to access, wrong memory portions can be accessed.

File Formats

The supported file formats are:

Palladium ASCII

This format uses physical memory addresses. A memory content file in Palladium ASCII format looks like:

```
$INSTANCE <memory_instance_name>
$RADIX BIN|HEX|OCT|DEC
$ADDRESS <start_address><end_address>
          <address><data>
          <address><data>
...
$END
```

In the syntax above, the \$INSTANCE, \$RADIX, and \$ADDRESS lines are header lines, which can be specified in any order. The \$RADIX and \$ADDRESS header lines are mandatory lines for all memory content files. The purpose of each header line is as follows.

- \$INSTANCE line specifies a hierarchical memory instance name related to the file. This line is optional.
- \$RADIX line specifies the radix of data in the file.
- \$ADDRESS line specifies the address range for which data has been specified in the file. The start address and end address should be hexadecimal number. The start address must be less or equal to the end address.

The header lines are followed by the data lines. Each data line contains a hexadecimal number as address and the data stored at that address. The radix of the data is specified by the \$RADIX header line. The addresses specified in the data lines must fall in the range specified in the \$ADDRESS header line. The addresses specified in the data lines need not be contiguous, but they must be in the ascending order. If an address that falls in the range specified by the \$ADDRESS header line, is not listed in the data section, it is assumed that the address contains all zero data. For such addresses, when memory is loaded from the file, corresponding memory words will be set to all zeros.

The \$END line indicates the end of memory content file.

memtran

This is a binary file format. It provides better load and dump performance as compared to the Palladium ASCII format. The memtran format supports data encryption that can protect the contents of the memory file. You can use encrypted file to initialize a memory. However, they can not dump out the file content.

raw/raw2

The `raw` format is a binary file format. You can write your own C/C++ program to read the raw format file or to generate raw format file.

The `memTran` utility can translate between raw format and other file formats. Refer to the *Files and File Formats* chapter in the *UXE User Guide* for more information on the `memTran` utility.

Table 13-2 shows the `raw` format of the binary memory file.

Table 13-2 Binary Memory File Format

Offset	Description	Value
0-5	signature	QTMEM\0
6	major version	1
7	minor version	0
8-11	reserved	0X12345678
12-47	reserved	0
48-51	depth of memory instance	
52-55	width of memory in number of bits	
56-59	number of bytes for each memory word. Should be calculated as $(\text{memory_width}+7)/8$	Depends on the specific memory instance.
60-63	start address	
64-67	end address	
68-71	instance name length	
72-xx	instance name	
yy-zz	memory data (left justified)	

The memory data are aligned to the byte boundary for each line of data. The bits of each line are left-justified in the bytes, that is, the most significant bit of the memory is the most significant bit of the first byte.

For example, if memory width is 50 bits, each line will occupy 7 bytes. Bit 49 of the memory is bit 7 (that is, 0x80) of the first byte. Bit 48 of the memory is bit 6 (that is, 0x40) of the first byte. Bit 0 of the memory is bit 6 (i.e. 0x40) of the 7th byte.

The `raw2` format is similar to the `raw` format. The only difference is that the `raw2` format does not contain any header. It contains pure data.

Denali

The memory loading behavior for this format strictly follows the Denali memory file format definition. For details, refer to Cadence Memory Model Portfolio User Guide available on the *Cadence Online Support* website – *Product Pages* menu – *Product Manuals* sub-menu – *Functional Verification* section – *Verification IP – VIPCAT 11.3 – Product Documentation for VIPCAT 11.3* web page.

readmemb/readmemh

The memory loading behavior for this format strictly follows the IES definition. For details, refer to *NC-Verilog Simulator Help* available in the INCISIVE documentation set.

IES

A memory image file in IES format contains:

- Mandatory directives for specifying the address format and the data format.
 - `$ADDRESSFMT <address_format>`
The `<address_format>` can be `H` (Hex), `B` (Binary), or `O` (Octal).
The memory image file must have a `$ADDRESSFMT` directive as the first non-comment line.
 - `$DATAFMT <data_format>`
The `<data_format>` can be `H` (Hex), `B` (Binary), or `O` (Octal).
- An optional directive for specifying a default value for unspecified addresses.
`$DEFAULTVALUE <default_value>`
The `<default_value>` can be `0` or `1`. If this directive is not specified, unspecified memory words will remain unchanged.
- One or more lines that specify:

- ❑ The address(es) to be filled and the data to be loaded. You can specify the address and data in hexadecimal, octal, or binary format. Refer to [Address/Data Format](#).
- ❑ The data to be loaded. You can specify the data in hexadecimal, octal, or binary format. Refer to [Data-Only Format](#).
- Comments (if required), which begin with a pound (#)sign.

There are two types of formats.

Address/Data Format

A memory image file in address/data format contains lines that specify both the addresses to be filled and data to be loaded. The format is:

```
<start_address>/<data> [<data> ...]  
<start_address>:<end_address>/<data>
```

As shown in the syntax above, three cases are supported:

- Single address/single data
 - For example, 3 / 1, which means address 3 has data 1.
- Single address/multiple data
 - For example, 3 / 1 ; 1 ; 0, which means address 3, 4, 5 have data 1, 1, 0 respectively.
- Multiple address/single data
 - For example, 3 : 5 / 1, which means address 3 to 5 have data 1.

The following is an example memory image file in address/data format.

```
# Address format  
$ADDRESSFMT H  
  
# Data format  
$DATAFMT H  
  
# Default value for unspecified addresses.  
# If default value is omitted, only the memory addresses that are  
# specified are filled. Other addresses remain unchanged.  
$DEFAULTVALUE 0  
  
# The following line loads value 0 at address 0.  
0/0
```

UXE Command Reference Manual

Run-time Commands

```
# The following line loads values a, b, c, d, and e at addresses 1 to 5.  
1/a;b;c;d;e  
# The following line loads value f at addresses a to 1f.  
a:1f/f  
# Values for address 6 to 9 are not specified, and will be set to 0, the  
# default value.
```

Data-Only Format

In this format, the lines in the memory file specify only the data. The format is:

```
data[;data ...]
```

The following is an example memory image file in data-only format.

```
# Address format  
$ADDRESSFMT B  
  
# Data format  
$DATAFMT B  
  
# Default value for unspecified addresses.  
# If default value is omitted, only the memory addresses that are  
# specified are filled. Other addresses remain unchanged.  
$DEFAULTVALUE 0000  
  
# The following line loads value 0000 at the address specified with -start.  
0000  
# The following line loads values 0000, 1111, and 0000 at the next three addresses.  
0000;1111;0000  
# Values for other addresses are not specified, and will be set to 0000, the  
# default value.
```

ncsim

Sends the *<ncsim_command>* directly to the NC-Sim simulator when debugging designs in the SA mode.

Note: The `ncsim` command is not supported with `vvmDebug`.

The syntax for this command is:

```
ncsim <ncsim_command>
```

For example, `ncsim help <ncsim_command>`

Note: In SA mode, use the `ncsim describe` command in place of the `memory -info` command.

The following run-time options can be used with the `ncsim` command to change the log file at run time in SA mode:

```
logfile [-set] {<logfile_name> | -default}  
    -append <logfile_name>  
    -overwrite <logfile_name>  
    -show [-default]
```

For detailed description of the `ncsim` commands, refer to the *Simulation Command-Line Options* document of the INCISIVE documentation set.

Some `ncsim` command options only give valid values on the software side. Even though the command appears to work when the design is swapped into hardware, it returns a stale value.

[-set] {<logfile_name> | -default}

Changes the current log file to the specified *<logfile_name>* log file during the run time. The `-default` option switches back to the default log file. The `-set` option is optional.

For example,

```
ncsim logfile -set test1.log
```

-append <logfile_name>

Appends the log to the specified log file.

UXE Command Reference Manual

Run-time Commands

-overwrite <logfile_name>

Overwrites the content of the specified log file.

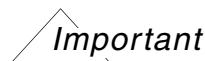
-show [-default]

Displays the name of the current log file. The `-default` option displays the name of the default log file. The `-default` option is optional.

probe

Note: The `probe` command discussed below is used at the run time. The `probe` compile-time command is discussed in [Chapter 8, “User Data Commands.”](#)

Creates or deletes probes for objects, such as signals and assertions, during run time. This command also provides the functionality to add power domain-related probes.



The `probe` command can work in both testbench partition (running in IES) and DUV partition (running in emulator hardware). The `xeDebug` command interpreter splits the probing requests between the testbench and DUV probes. IES can probe either nets or memory words. The emulator cannot support probing of memory words, so only probed nets (either as individual, or specified through instance names) are supported in the DUV. For more details on the `probe` command, refer to the [NC-Verilog Simulator Help](#) in the INCISIVE documentation set.

The `probe` command should be executed before running the design or executing the [database -upload](#) command.

Note: In `vvmDebug`, the `probe` command options for CPF and IEEE 1801 are not supported.

The syntax for this command is:

```
probe -create {[-all] [-allnets] [-depth {<n> | all | to_cells}] [-name <probe_name>] \  
    [-ports] [-waveform] } \  
    {<object> ... | <scope_name>} [-addcone -depth <n> <signal> ...] \  
    [-assertion [-state]] [-errorok] \  
    [-exclude {<object> | <scope_name>}] [-excludelibrary <.llib names>] \  
    [-signals] [-fast]  
  
probe -delete <probe_name> [<probe_name>...]  
  
probe -list [<probe_name>...] [-fast]  
  
probe -save [<filename>] [-fast]  
  
probe -show [<probe_name>...] [-fast] [-v]  
  
probe -addcone -depth <n> <signal> ...  
  
probe -addpath -depth <n> <net1> <net2>
```

The following command options are for CPF:

```
probe -addIllegalDomainConfiguration {<illegalDomainConfiguration> | *}  
probe -addMode {<modeName> | *}
```

UXE Command Reference Manual

Run-time Commands

```
probe -addModeTransition {<modeTransitionName> | *  
probe -addPowerDomain {<powerDomainName> | *  
probe -addPowerMode {<powerModeName> | *  
probe -addIsolation {<isolationRuleName> | *  
probe -addRetention {<stateRetentionRule> | *
```

The following command options are for IEEE 1801:

```
probe -addpowerstate [<namePattern>]  
probe -addpowerdomain [<namePattern>]  
probe -addisolation [<namePattern>]  
probe -addretention [<namePattern>]  
probe -addsupplyport [<namePattern>]  
probe -addsupplynet [<namePattern>]  
probe -addsupplyset [<namePattern>]  
probe -addswitch [<namePattern>]  
probe -addpst [<namePattern>]  
probe -addportstate [<namePattern>]  
probe -addpststate [<namePattern>]  
probe -power
```

-create

Creates a probe during the run time.

-fast

Creates a fast probe. The fast probe enables you to bypass fullvision computation for faster waveform generation and faster waveform trigger in VVM. The fast probe signals improve the upload time in both VVM sessions and normal runs. Based on your compile options, there is a limit to the number of fast probes that can be added at run time. For more information, see [compilerOption -add {numExtraCapturedNetsPerDomain <n>}](#).

Example

```
probe -create -fast .
```

The command listed above will probe all top level signals.

Note: The `-fast` option is used only in the online sessions and is appropriate only with FullVision mode.

-all

Adds all the objects, including assertions, in the database hierarchy of the given instances to the trace signal list.

Note: The `probe -all` command will not add the tool-generated internal nets.

-allnets

Adds the tool-generated internal nets and the design nets to the trace signal list.

-depth {<n> | all | to_cells}

Specifies the number of hierarchical levels to descend when searching for objects to probe.

By default, only the given scope is included in the probe, that is, the value of `<n>` is set to 1.

If the value of `<n>` is specified as 2, the given scope as well as its sub-scopes are included in the probe.

Specifying the depth as 0 or `all` indicates that all scopes in the hierarchy below the specified scope(s) should be included.

The `to_cells` argument is defined in IES tools as modules with ``celldefine` or VITAL entities with VITAL Level10 attribute. This is meaningful only in IES where selection will stop its hierarchical descent at these modules. In UXE, the depth specifier will not be limited when using a `to_cells` argument, but will act as setting the `-depth` option to 0 or `all`.

-name <probe_name>

Specifies a user-defined name for the probe. The `-name` sub-option applies to all the options of the `probe` command. Therefore, a subsequent `probe -delete -name <probe_name>` command operates on all the items that were specified in that `probe` command.

If a probe name is not specified, xeDebug will assign a default probe name. This means that the probe requests that xeDebug sends to IES will always have an explicit probe name whether a user-specified probe name or a system-assigned one.

-ports

Adds all input and output ports within a scope in the probe. This applies to the current debug scope (if no scope name is specified in the argument), the scope(s) specified in the argument, and the sub-scopes specified with the `-depth` option.

-waveform

Causes the objects in the probe to be added to the waveform display.

<object> ... | <scope_name>

Identifies the specific object(s) or scope(s) for creating probes.

If a scope is specified, the probes are created for the objects within it. The objects can be simple signal names or single bits or range selection of a vector signal such as `A[5]` and `A[20:10]` or unpacked structs. It can also be names of design instances, or assertions (in the SA mode). If an instance name is specified then all unpacked structs under the instance are probed. Objects can be specified with or without a fully qualified design hierarchical path (such as `x.y.z`). (Refer to the description of the `scope` command for the case where the given object is not specified with a hierarchical path.)

Note: While probing assertions, unless you specify the `-assertion` option, other signals in the specified scopes will also be selected.

Some restrictions apply while using the RTL names for probing the sub-constructs and field bits of unpacked structs, packed structs, and MDAs. For complete information and examples, refer to the *Using RTL and PDB Names for Probing SystemVerilog Design Constructs* section in the *UXE User Guide*.

-errorok

Specifies to ignore errors related to missing nets.

-assertion

Selects only the assertion objects from the specified scope(s).

-state

Captures and displays the assertion state values. This is a default argument. Therefore, its specification on the command prompt is optional. Use of this argument displays the text state waveforms. The other `-failure` and `-transaction` arguments are IES-defined switches that are not supported for DUV assertions.

-signals

Adds the assertion fanin signals for the specified assertions to the probe set. This option can also be used with the `assertion -list` run-time command. For detailed information, refer to the *Support for Assertion Fanin Signals* section of the *Compiling and Running Designs with Assertions* chapter of *UXE User Guide*.

-exclude {<object> | <scope_name>}

Excludes tracing of the specified assertions and/or those within the specified scopes.

Note: This option is supported only for software simulator probes under the SA (IXCOM) mode.

-excludelibrary <.lilib names>

Excludes all internal nets in the modules that are defined in the libraries. The name of the library is the name of the `.lilib` that you specified during compilation. You can execute the `ls QTDB` command to see the correct `.lilib` names. If you specify two or more library names you need to enclose all the library names with double quotes or curly braces.

-delete <probe_name> [<probe_name>...]

Deletes the specified probe(s) to the net. Use the `probe -delete` command to delete only whole probes and not individual signals within each probe.

Two wildcard characters, `*` and `?`, can be used in the `<probe_name>` argument. The asterisk (`*`) matches any number of characters; whereas, the question mark (`?`) matches any one character.

Note: The SHM probes can be deleted at any time. However, VCD probes can be deleted only at the time of VCD database creation.

-list [<probe_name>...] [-fast]

Returns a list of the nets existing in the specified probe, or a list of probes. If the `-fast` option is specified, the list of nets is displayed for all the existing fast probes.

Note:

- The `probe -list` command only lists probes in DUV partition running in emulator hardware. It is not applicable in the software simulation mode before the design is swapped-in to the hardware.
- The `probe -list` command does not list the CPF probes.

The wildcard characters, * and ?, can be used in the `<probe_name>` argument. The asterisk (*) matches any number of characters; whereas, the question mark (?) matches any one character. Consider the following two examples that illustrate the use of wildcard characters to get a list the nets:

■ `probe -list *`

This command line will list all the nets within the existing probes.

■ `probe -list myprobe *5`

This command line will list the nets from the probe named `myprobe` and also from other probes with a name that ends with 5.

If the `<probe_name>` argument is not specified with the `-list` option, all nets within all existing probes are displayed.

-save [<filename>] [-fast]

Creates a Tcl script that can be executed to re-create the current databases and probes. If a file name is not specified, the command only returns the script as a string. If the `-fast` option is specified, only the fast probe signals will be saved.

-show [<probe_name>...][-fast**] [**-v**]**

Displays information about the specified probe or list of probes. If the `-fast` option is specified, information about only the fast probe signals will be displayed. The `-v` option returns the number of probing resources occupied by each fast probe.

The wildcard characters, * and ?, can be used in the `<probe_name>` argument. The asterisk (*) matches any number of characters; whereas, the question mark (?) matches any one character.

If the *<probe_name>* argument is not specified with the **-show** option, information about all probes is displayed.

Example

```
probe -show x13 -v
```

Sample Output

```
x13: -fast -depth 0 c_top
      13.1hw; : -fast -depth 0 c_top
```

-addcone -depth <n> <signal> ...

Adds all signals in the given nets input cone of logic to the trace signal list. The **-depth <n>** parameter indicates the number of levels to search.

-addpath -depth <n> <net1> <net2>

Traces back from *<net1>* to *<net2>* and adds all nets on the path. The **-depth <n>** parameter indicates the number of levels that are traced back. The default value is 10 levels.

If the path cannot be found within the specified depth (level), an error is generated.

-addIllegalDomainConfiguration {<illegalDomainConfiguration> | *}

Adds the specified CPF illegal domain configuration into the probe set. If this option is specified, the status of illegal domain configuration is displayed in the waveform.

Use the wildcard * to add all CPF illegal domain configuration into the probe set.

-addMode {<modeName> | *}

Adds the specified CPF general modes into the probe set. If this option is specified, general mode-related information is displayed in the waveform.

Use the wildcard * to add all CPF general modes into the probe set.

-addModeTransition {<modeTransitionName> | *}

Adds the specified CPF mode transitions into the probe set. If this option is specified, the status of a power domain transitioning from one power mode to another power mode is shown in the waveform.

Use the wildcard * to add all CPF mode transitions into the probe set.

-addPowerDomain {<powerDomainName> | *}

Adds the specified CPF power domain into the probe set. The PSO status for the specified *<powerDomainName>* is displayed in the waveform.

Use the wildcard * to add all CPF power domain into the probe set.

Note: A power domain in a CPF macro model cannot be probed directly. The top-level mapped power domain can be probed and the mapping information is in the top-level power domain. To probe a block-level power domain, specify the power domain name as *<scope>. <powerDomainName>*.

-addPowerMode {<powerModeName> | *}

Adds the specified CPF power modes into the probe set.

Use the wildcard * to add all CPF power modes into the probe set.

Note: A power mode in a CPF macro or design model cannot be probed.

-addIsolation {<isolationRuleName> | *}

Adds the specified CPF power isolation rules into the probe set. The isolation status (on or off) of the specified *<isolationRuleName>* is displayed in the waveform.

Use the wildcard * to add all power isolation rules into the probe set.

Note: An isolation rule in a CPF macro model cannot be probed because the corresponding macro simulation model should include this isolation behavior and the UXE compiler does not add the isolation instrumentation logic into the netlist. To probe a block-level isolation rule, specify the name as *<scope>. <isolationRuleName>*.

-addRetention {<stateRetentionRule> | *}

Adds the specified CPF power state retention rules into the probe set. The save/restore and on/off status of the specified <*stateRetentionRule*> is displayed in the waveform.

Use the wildcard * to add all power state retention rules into the probe set.

Note: A state retention rule in a CPF macro model cannot be probed because the corresponding macro simulation model should include the state retention behavior and the UXE compiler does not add the state retention instrumentation logic into the netlist. Only top-level state retention rules can be probed. To probe a block-level state retention rule, specify the name as <*scope*>. <*stateRetentionRule*>.

-addpowerstate [<namePattern>]

Displays the list of all probed power state object names. If <*namePattern*> is specified, adds the matched <*namePattern*> power states into the probe set.

The <*namePattern*> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <*namePattern*>.

The matched power state names are <*scope*>. <*powerState*>.

The probe -addpowerstate <*namePattern*> command adds the matched power states into the probe set, and the database -upload command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 power states in waveform are under the power_intent_1801_objects.PowerStates hierarchical scope. The content of probed power state object depends on its specified options. Typically, the waveform displays the following items under a power state object:

```
PowerState
<scope>. <powerState>
    supply - simstate of corresponding supply set: normal, corrupt, or undetermined
    domain - simstate of corresponding domain
    simstate - normal, corrupt, or undetermined
    legal - true or false
    logic_expr - true or false
    supply_expr - true or false
```

-addpowerdomain [<namePattern>]

Displays the list of all probed power domain object names. If <*namePattern*> is specified, adds the matched <*namePattern*> power domains into the probe set.

The <*namePattern*> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <*namePattern*>.

The matched power domain names are <*scope*>. <*domain*>.

The probe -addpowerdomain <*namePattern*> command adds the matched power domains into the probe set, and the database -upload command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 power domains in waveform are under the power_intent_1801_objects.PowerDomain hierarchical scope. The content of probed power domain object depends on its specified options. Typically, the waveform displays the following items under a power domain object:

```
PowerDomain
  <scope>. <domain>
    simOnly - true, false or none
    simstate - the power domain's simstate
    interface_drivers - output interface pins/nets of the power domain
    interface_receivers - input interface pins/nets of the power domain
    isolations - all related isolations simstate
    retentions - all related retentions simstate
    switches - all related switches simstate
    primany - primary supply set simstate: normal, corrupt, or undermined
    default_retention - default retention supply set simstate
    default_isolation - default isolation supply set simstate
    <user_defined supply set> - user defined supply set simstate
    states - all related power states simstate
    interface_inouts - inout interface pins/nets of the power domain
```

-addisolation [<namePattern>]

Displays the list of all probed isolation strategy names. If <*namePattern*> is specified, adds the matched <*namePattern*> isolation strategy into the probe set.

The <*namePattern*> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <*namePattern*>.

UXE Command Reference Manual

Run-time Commands

The matched isolation names are `<scope>.<isolation>` and `<scope>.<domain>.<isolation>`. If multiple isolation strategies have the same name but different domains, specify the retention member name as `<scope>.<domain>.<isolation>`. Otherwise, only a random isolation strategy is probed.

The probe `-addisolation <namePattern>` command adds the matched isolation strategies into the probe set, and the database `-upload` command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 isolation strategies in waveform are under the `power_intent_1801_objects.Isolation` hierarchical scope. The content of probed isolation object depends on its specified options. Typically, the waveform displays the following items under an isolation object:

Isolation

```
<scope>.<isolation>
  diff_supply_only - true or false
  domain - the related domain's simstate
  emptyElements - true or false
  isolation_condition - the isolation is true or false
  isolationControlSingalIsCorrupted - the control signal is corrupted or not
  isolation_ground_net - supply state and voltage
  isolation_power_net - supply state and voltage
  simstate - the isolation's simstate: corrupt or normal
  use_equivalence - true or false
  isolationPins - isolated pins
  source - source supply set simstate
  sink - sink supply set simstate
  no_isolation - true or false
  <isolation supply set> - user-defined supply set simstate
  force_isolation - true or false
```

-addretention [<namePattern>]

Displays the list of all probed retention strategy names. If `<namePattern>` is specified, adds the matched `<namePattern>` retention strategy into the probe set.

The `<namePattern>` can include wildcards * and ?. Use a dot (.) as hierarchical separator in the `<namePattern>`.

The matched retention names are `<scope>.<retention>` and `<scope>.<domain>.<retention>`. If multiple retention strategies have the same name

but different domains, specify the retention member name as `<scope>.<domain>.<retention>`. Otherwise, only a random retention strategy is probed.

The probe `-addretention <namePattern>` command adds the matched retention strategies into the probe set, and the database `-upload` command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 retention strategies in waveform are under the `power_intent_1801_objects.Retention` hierarchical scope. The content of probed retention object depends on its specified options. Typically, the waveform displays the following items under a retention object:

Retention

```
<scope>.<retention>
    domain - corresponding domains' simstate
    restore_condition - true or false
    retentionControlSignalIsCorrupted - true or false
    retention_conditon - true or false
    retention_ground_net - supply state and voltage
    retention_power_net - supply state and voltage
    save_condition - true or false
    simState - the retention's simstate: corrupted or normal
    retainedPins - all the sequential elements' output pins if they are retained
    retention_supply_set - supply set simstate: normal, corrupt, or undermined
    retainedInstances - all the retained instances' simstate (equal to the
                        retention's simstate)
```

-addsupplyport [<namePattern>]

Displays the list of all probed supply ports. If `<namePattern>` is specified, adds the matched `<namePattern>` supply port into the probe set.

The `<namePattern>` can include wildcards * and ?. Use a dot (.) as hierarchical separator in the `<namePattern>`.

The matched supply port names are `<scope>.<supplyPort>`.

The probe `-addsupplyport <namePattern>` command adds the matched supply port strategy into the probe set, and the database `-upload` command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 supply port strategies in waveform are under the `power_intent_1801_objects.SupplyPort` hierarchical scope. The content of probed

supply port object depends on its specified options. Typically, the waveform displays the following items under a supply port object:

```
SupplyPort
<scope>.<supplyPort>
  supply state - off, undetermined, partial_on, or full_on
  supply voltage - voltage number
```

-addsupplynet [<namePattern>]

Displays the list of all probed supply nets. If <namePattern> is specified, adds the matched <namePattern> supply nets into the probe set.

The <namePattern> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <namePattern>.

The matched supply net names are <scope>.<supplyNet>.

The probe -addsupplynet <namePattern> command adds the matched supply nets into the probe set, and the database -upload command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 supply nets in waveform are under the power_intent_1801_objects.SupplyNet hierarchical scope. The content of probed supply net depends on its specified options. Typically, the waveform displays the following items under a supply net object:

```
SupplyNet
<scope>.<supplyNet>
  supply state - off, undetermined, partial_on, or full_on
  supply voltage - voltage number
  switches - related switches' information
```

-addsupplyset [<namePattern>]

Displays the list of all probed supply sets. If <namePattern> is specified, adds the matched <namePattern> supply sets into the probe set.

The <namePattern> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <namePattern>.

The matched supply set names are <scope>.<supplySet>.

The probe `-addsupplyset <namePattern>` command adds the matched supply sets into the probe set, and the database `-upload` command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 supply sets in waveform are under the `power_intent_1801_objects.SupplySet` hierarchical scope. The content of probed supply set depends on its specified options. Typically, the waveform displays the following items under a supply set object:

`SupplySet`

```
<scope>.〈supplySet〉  
power - power net/port state: off, undetermined, partial_on, full_on + voltage  
ground - ground net/port state  
states - supply set states' simstate - corrupt, normal, or none  
domains - all the related domains' simstate  
retentions - all the related retentions' simstate  
isolations - all the related isolations' simstate  
simstate - current supply set's simstate
```

-addswitch [<namePattern>]

Displays the list of all probed power switch names. If `<namePattern>` is specified, adds the matched `<namePattern>` power switches into the probe set.

The `<namePattern>` can include wildcards * and ?. Use a dot (.) as hierarchical separator in the `<namePattern>`.

The matched switch names are `<scope>.〈switch〉`.

The probe `-addswitch <namePattern>` command adds the matched power switches into the probe set, and the database `-upload` command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 power switches in waveform are under the `power_intent_1801_objects.PowerSwitch` hierarchical scope. The content of probed switches depends on its specified options. Typically, the waveform displays the following items under a power switch object:

`PowerSwitch`

```
<scope>.〈switch〉  
domain - the corresponding domain's simstate  
<port>.input - input supply: off, undetermined, partial_on, full_on + voltage  
onState - all on states: true or false  
output - output supply: off, undetermined, partial_on, full_on + voltage
```

```
supplySet - supply set simstate  
offState - all off states: true or false  
onPartialState - all onPartial states: true or false  
errorState - all error states: true or false  
ackPort - all ack port state: true or false
```

-addpst [<namePattern>]

Displays the list of all probed pst names. If <namePattern> is specified, adds the matched <namePattern> pst objects into the probe set.

The <namePattern> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <namePattern>.

The matched pst names are <scope>. <pst>.

The probe -addpst <namePattern> command adds the matched pst objects into the probe set, and the database -upload command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 pst objects in waveform are under the power_intent_1801_objects.PST hierarchical scope. The content of probed pst depends on its specified options. Typically, the waveform displays the following items under a pst object:

PST

```
<scope>. <pst>  
states - pst states: true or false (on or off)  
supplies - member supply port/net state: off, undetermined, partial_on, full_on
```

-addportstate [<namePattern>]

Displays the list of all probed port state names. If <namePattern> is specified, adds the matched <namePattern> port states into the probe set.

The <namePattern> can include wildcards * and ?. Use a dot (.) as hierarchical separator in the <namePattern>.

The matched port state names are <scope>. <portState>.

The probe -addportstate <namePattern> command adds the matched port states into the probe set, and the database -upload command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 port state objects in waveform are under the `power_intent_1801_objects.PortState` hierarchical scope. The content of probed port states depends on its specified options. Typically, the waveform displays the following items under a port state object:

```
PortState  
<scope>.<portState>  
<isInState> - true, false or none
```

-addpststate [<namePattern>]

Displays the list of all probed pst state names. If `<namePattern>` is specified, adds the matched `<namePattern>` pst states into the probe set.

The `<namePattern>` can include wildcards * and ?. Use a dot (.) as hierarchical separator in the `<namePattern>`.

The matched pst state names are `<scope>.<pstState>` and `<scope>.<pst>.<pstState>`. If multiple pst states have the same name but different pst, specify the pst state member name as `<scope>.<pst>.<pstState>`. Otherwise, only a random pst state is probed.

The probe `-addpststate <namePattern>` command adds the matched pst states into the probe set, and the database `-upload` command adds the probed IEEE 1801 objects into SimVision or Debussy waveform viewer.

All IEEE 1801 pst states in waveform are under the `power_intent_1801_objects.PST_State` hierarchical scope. The content of probed pst states depends on its specified options. Typically, the waveform displays the following items under a pst state object:

```
PST_State  
<scope>.<pstState>  
<isInState> - true, false or none
```

-power

Probes all low power simulation control conditions for corruption, isolation and retention derived from the power switch network. It also probes the nets under domain instances.

release

Releases any previous force (that was done using the `force` command) on the specified signal or symbol. A signal can be released if it is specified as a `keepNet` during compile. Before executing this command, download the design. If the `release` command is operated on a bus in LA mode while the clocks are running, time coherency between the values of the signals in the bus is not guaranteed.

Note: When operating the `release` command on a large set of signals, significantly improve the access time by grouping the signals with the `symbol` command, and using that symbol in the `release` command, instead of separately executing the `release` command on each signal. When grouped in such way, more efficient mechanisms are used to access the hardware. The `release` command is not supported with `vvmDebug`.

The syntax for this command is:

```
release [-keepvalue] <signal> ... | <symbol> ...  
release -allff [-instance <instance_name>]
```

-keepvalue

Releases the forced object, but retains the forced value. This command option is supported only in the SA mode for testbench objects and DUV running on the software side.

<signal> ... | <symbol> ...

Releases a previously executed force on the specified signal or symbol. The value of the specified signal is recalculated at the next cycle. The sourceless nets that were declared as `keepNets` will retain their values (state).

-allff [-instance <instance_name>]

Releases the flip-flop and latch outputs forced by a previous `force -allff` command.

Note: This option is supported only in the ICE mode and not in the SA mode.

reset

Resets the emulation/simulation and clears the contents of the design memory.

This command is supported in the Vector Debug and SA (IXCOM) modes. It is not supported in LA and STB modes. It is also not supported with vvmDebug.

If you need to reset the time to zero in STB mode, you can use the resetTime command.

In the Vector Debug mode, this command resets the stimulus to the beginning. Consequently, the next run command uses the first vector from the stimulus file and not the next vector after the save point. Note that before using the reset command, the vector command must be executed.

In the SA mode, this command returns the design to the simulator and resets the simulation to time zero (0). In addition, it initializes all nets, registers, and memories to their initial value by reloading the initial snapshot. The emulator is also notified and it performs its own reset sequence, including reset of the waveform database. The waveform database is reset by discarding any previously probed data such that any probed data will start again from time zero.

The syntax for this command is:

```
reset
```

Examples

```
# Initialize debugger; use '/mydesign/design1'  
# directory.  
debug /mydesign/design1  
  
# Use the 'samurai' emulation host.  
host samurai  
  
# Download the design to the 'samurai' host.  
download  
  
# Configure for Vector Debug mode.  
configPM -vd  
  
# Compile the vectors in the 'myvector.cbc' file.  
vector myvector.cbc  
  
# run only first 50 vectors  
run 50  
  
#run same first 50 vectors again  
reset  
run 50
```

resetTime

Resets the simulated time to 0. You can execute this command only after downloading the design.

This command can be used in STB mode to reset the simulation time to 0. This affects only simulation time (as reported by SDL, waveform, and so on), but it does not reset the design state.

This command is not supported when using infiniTrace (either prepare or observe modes).

The syntax for this command is:

```
resetTime
```

restart

Reloads an existing snapshot that was saved using the save [-simulation] <snapshot name> command.

This command is available in the STB and SA modes. In vvmDebug, this command is used to go to a specified time point in the design. For more information, see restart (in VVM).

The `restart` command must not be the first command if the designs running in SA mode use the following:

- DPI calls
- GFIFO or SFIFO calls
- SCE-MI pipes

For example, your design uses the above-mentioned function calls or SCE-MI pipes, and you issue the following commands, while running in SA mode:

```
debug .
host .
restart SS1
```

Here, SS1 is the NC-Sim snapshot name. The run-time software will issue the following error message:

```
SF *FATAL*: [234234] restart called before port initialization, please at least 'run -delta' before restarting port-interface design.
Connection has been closed while executing "restart SS1"
```

To workaround the problem, issue the `run -delta` command as the first CLI command.

The syntax for this command is:

```
restart [-path <path>] <save_name> [-savedb]
restart -show
```

[-path <path>]

Specifies the path to reload the saved simulation snapshot when running the designs in SA mode.

<save_name>

Restarts simulation and emulation states from the checkpoint snapshot created using the `save -simulation` command.

Note: The snapshot must not be from a different model.

The `<save_name>` argument is interpreted in the same manner as on the simulator command line with an addition that a view of the current snapshot's library and cell can be abbreviated as `:<view>`.

When running in the SA mode, both simulation and emulation states will be restored. The `restart` command requires only the NC-Sim snapshot name in the argument. The UXE software automatically determines the name of the emulator image file by deriving it from the specified NC-Sim snapshot name.

When running in ICE mode, only emulation state is restored.

-savedb

Saves the current databases. This option is valid only in the SA mode.

Without this option, the `restart` command will overwrite the waveform database starting from the restart time point.

-show

Lists the names of all snapshots that can currently be used as an argument to a `restart` command. This option is valid only in the SA mode.

restart (in VVM)

In vvmDebug, the `restart` command is used to go to a specified time point in the design. You can either specify the number of clock cycles or use a pre-defined bookmark.

The syntax of this command is:

```
restart {<cycles> | <bookmark_name>}
```

resultWaveform

Opens the Waveform Viewer and displays the results from the last `run` command. This command is available only in Vector Debug mode.

This command reads the emulation result waveform and shows the comparative results on the screen. Note that any mismatches between the expected values and actual results are highlighted in a different color.

Before using this command, execute the `debug` command to specify the design to use.

Note: This command is not supported with vvmDebug.

The syntax for this command is:

```
resultWaveform
```

Note: To view the waveforms in the FSDB flow, before running the `xeDebug` command, ensure that the Novas tools and licenses are installed and setup is completed as follows:

```
setenv NOVAS_HOME <novas_tool_installation_directory>
setenv TURBO_ETC_DIR $NOVAS_HOME/etc/access
setenv LM_LICENSE_FILE $NOVAS_HOME/license/license.dat:$LM_LICENSE_FILE
set path = ($NOVAS_HOME/bin $path)
## set NOVAS_EXE for xeDebug
setenv NOVAS_EXE $NOVAS_HOME/bin/verdi (or debussy or siloti)
```

Examples

```
XE> debug .
XE> host <emulator_host_name>
XE> download
XE> configPM -vd
XE> vector <cbc_file_name> cbc2vbf.vtran {}
    # If your cbc.in file has been generated,
    # you can set vector without translation as shown in the following line:
XE> vector <cbc_file_name> - {}
XE> run
XE> resultWaveform
```

resume

Restores the last snapshot of the `suspend` command. This command is available in the STB and SA modes. However, it is not supported with vvmDebug and VVM Prepare sessions in xeDebug. For detailed information about resume and suspend operations, refer to the *Using Suspend and Resume to Manage Emulator Resources* section of the *Working with the Palladium XP Emulators* chapter in the UXE User Guide.

You must have issued a `suspend` command before using the `resume` command. A `resume` command issued without a `suspend` command will generate an error condition. If the design is suspended remotely using the File-based Suspend-Resume operation, it cannot be resumed locally using the command prompt or xeDebug GUI. For details, refer to the *File-based Suspend-Resume in the SA Mode* section of the *Working with the Palladium XP Emulators* chapter in *UXE User Guide*.

The syntax for this command is:

```
resume
```

If the emulation clock was running before the `suspend` command was executed, after executing the `resume` command the remaining clock cycles would be completed. Any SDL trigger set before the suspend would still remain valid after the resume. Also, if suspend-resume happens in a batch or script mode, the xeDebug commands following the `resume` command would be executed as expected.

Example

```
XE> debug .
XE> host myhost
XE> download
XE> configPM -stb
XE> run -nowait          << non-blocking free run
XE> suspend              << suspend the emulation and release the emulator
XE> resume                << reload the design from the last stopping point;
                           << remaining run cycles continue
```

run

Starts design emulation clocks and/or data tracing. It is available in all debug modes.

To emulate in the Vector Debug mode, you need to first execute the [vector](#) command. In the VD and STB modes, this command starts both emulation and data tracing. In the LA and Dynamic Target modes, if the clocks are already running, this command starts only data tracing.

The run will stop on a trigger from SDL, regardless of whether any time or number of cycles is specified in the `run` command in all emulation modes. The SA, VD, and STB modes also enable you to specify the number of cycles or simulation time in addition to trigger conditions. However, note that in the SA, VD, and STB modes stopping the run means stopping the emulation clocks, while in the LA and Dynamic Target modes, stopping the run only means stopping probe tracing, but not the emulation clocks.

In all emulation modes except VD mode and pure software simulation, the `run` command can operate in two modes, blocking or non-blocking. In a blocking run, control does not return from the `run` command to the XE prompt until the run is stopped. You can stop a blocking `run` by using the **CTRL+C** key combination. In such case, the command will return the string *Canceled* and generate an error condition (which can be caught by the `catch Tcl` command).

When debugging a design that has uncontrolled clocks, and the stop mode is set to `none` or `*none` (with the command `clockConfig -stopMode none`), then an SDL trigger will stop only SDL and waveform capture, but the design will continue to run. In such case, if the run was non blocking, you can issue a new `run` command, which will restart SDL and waveform capture. This is the only case in which a new `run` command is allowed before the previous non blocking run terminated.

In case of non-blocking run, the control is immediately returned to the XE prompt, without waiting for the emulator to stop. In such a case, you can put the system back into blocking mode (and wait until the emulator clocks have stopped) by executing the [waitWhileBusy](#) command, or you can stop the run with the [stop](#) command.

During a non-blocking run, executing a **TCL** or **XEL** command suspends concurrent operations from ATB or SDL. This includes messages from ATB `$display` and SDL `DISPLAY` actions, and execution of `EXEC` actions from SDL.

For example, consider the following case:

```
run -nowait
source scr.qel
```

Execution of the SDL EXEC statement, as well as messages from SDL DISPLAY and ATB \$display will be suspended throughout the entire execution of the scr.qel script file.

In the VD mode, the run command is always blocking.

In VD mode, the run cannot be interrupted until the emulator stops. The **CTRL+C key** combination does not work in this mode.

The syntax for this command is:

In the Logic Analyzer and Dynamic Target mode under IXC0M:

```
run [-emulation] [-wait | -nowait] [-continue]
```

In the STB mode:

```
run [-wait | -nowait] [-continue] [<amount_to_run> [<time_unit>]]
```

In the Vector Debug mode:

```
run [<cycles>]
```

In all SA modes, except Dynamic Target mode:

```
run [-wait | -nowait] [-continue] [-swap] [-ignorestop] [<amount_to_run> [<time_unit>]]
```

Note: The run command options discussed here are UXE-specific extensions. In addition to these, the run command supports use of other options described in the *NC-Verilog Simulator Help* of the INCISIVE documentation set.

-emulation

When specified, starts emulation without starting probe tracing in the Logic Analyzer or Dynamic Target mode. In such case, the emulator is still considered “not busy” until another run command (without the -emulation option) is given. If not specified, it starts both emulation and probe tracing.

<amount_to_run>

Specifies the number of clock cycles or time to run, when emulating in the STB mode.

The interpretation of this value depends on the specified <time_unit>.

<time_unit>

Specifies the time unit in which *<amount_to_run>* is interpreted. If not specified, the value of the `timeUnit` run-time parameter is used. This argument can assume any value that is accepted by `timeUnit`. For a complete explanation of legal values, refer to [timeUnit](#).

Under IXCOM, it can specify only time, but not clock cycles.

-wait

Causes the `run` command to be blocking. The control is returned to the XEL command interpreter only when the `run` command ends. This option is recognized in all the run-time modes except the VD mode and pure simulation mode. Without the `-wait` and `-nowait` options, the command is run as non-blocking in the LA mode and Dynamic Target mode (for designs compiled using IXCOM), and as blocking in all other run-time modes.

Note: This option is not supported with `vvmDebug`.

-nowait

Causes the `run` command to be non-blocking. The control is immediately returned to the XEL command interpreter without waiting for the `run` to end.

This option is recognized in all the run-time modes except the VD mode and pure simulation mode. Without the `-wait` and `-nowait` options, the command is run as non-blocking in the LA mode and Dynamic Target mode (for designs compiled using IXCOM), and as blocking in all other run-time modes.

Note: This option is not supported with `vvmDebug`.

-continue

This option is available in all modes except VD mode. It preserves the internal SDL state from the previous run. Any changes since the last run to the SDL file or other run-time parameters that affect the SDL program are ignored.

In addition, in STB mode, if *<amount_to_run>* is not specified, the `run` will continue to execute the number of cycles (or time) that remained in the previous run. For example, if the previous run was for 100 cycles and the emulator stopped after 30 cycles for some reason, a `run -continue` command will attempt to run only up to 70 cycles.

The `-continue` option can be used with any combination of the options `-nowait` and `<amount_to_run>`.

In Logic Analyzer and Dynamic Target modes, the `-continue` option can be used only when the design clocks are stopped. For example, in Logic Analyzer mode only the first `run` command after switching from STB mode can use the `-continue` switch.

<cycles>

Specifies the number of vectors from the stimulus file to run, when emulating in Vector Debug mode.

-swap

Completes a swap request previously issued by executing either the `xc on` or `xc off` command without advancing time, when running a design in the SA mode. When this command option is used without the `-ignorestop` option, the swap request can be stopped by an assertion or `$stop` system task.

Note: This option is not supported with vvmDebug when running a design in SA mode.

-ignorestop

Enforces swapping regardless of interruptions requested by an assertion or `$stop` system task.

Note: This option is not supported with vvmDebug when running a design in SA mode.

Examples

```
# Initialize the debugger and use the
# '/mydesign/design1' design database directory.
debug /mydesign/design1

# Download the design to the 'samurai' host.
host samurai
download

configPM -vd

# Compile the vectors in the 'myvector.cbc' file.
vector myvector.cbc

# Start emulation.
run
```

save

Saves the current state of the emulator and simulator (if running in the SA mode), or the run-time environment. This command is not available in VD mode.

The syntax for saving the design state for SA mode is:

```
save -path <path> -simulation <snapshot_name> -overwrite
```

The syntax for saving the design state for ICE mode is:

```
save <save_name> -overwrite  
save -compress [{Low | Medium | High | None}]
```

The syntax for saving the run-time environment is:

```
save {-commands | -environment} [<filename>]
```

The syntax for saving the user-defined Tcl variable is:

```
save -uservars <filename>
```

[-path <path>]

Specifies the path to save the simulation snapshot when running the designs in SA mode. The -path option is optional. Without this option, the snapshot is saved at the run directory.

[-simulation] <snapshot_name>

Creates a snapshot of the current emulation state (and also simulation state in the SA mode). The keyword -simulation is optional. The -simulation option is supported only in SA mode. This option is not supported with vvmDebug.

Note: For a simulation state, the snapshot name must include only letters, numbers, and underscores.

In the SA mode, the process of saving the emulator image and the simulation snapshot is synchronized by NC-Sim. Therefore, a single save -simulation command performs the save function for both the design state image in both the emulator and the simulator, and UXE software automatically determines the name of the emulator image file by deriving it from the specified NC-Sim snapshot name.

<save_name>

Specifies the basename of the files where the design state information will be saved. This option is supported only in ICE mode. It is not supported with vvmDebug.

When saving the design state, the state information is saved into a few files under the specified *<save_name>* directory.

The *<save_name>* option can include path information. For example, `mysave/ss1`, where the state will be saved under the `mysave/ss1` directory.

The design state will be saved in binary format.

Note: The text format to save a design state is obsolete.

-compress [{Low | Medium | High | None}]

Compresses the saved design snapshots while running the designs in the ICE mode. This option saves the disk space and reduces the I/O time for writing the larger snapshots. You can also specify the level of compression using the following options:

- Low or L: Low reduction with high performance
- Medium or M: Medium reduction with good performance
- High or H: High reduction with low performance
- None or N: No reduction with high performance

Note: You can also specify the first letter for the compression levels, such as `L` for low compression level, `M` for medium compression and so on.

The saved snapshots are compressed as they are written. A flag is also written to the saved snapshot directory to indicate the level of compression, if any, to ensure that the correct decompression algorithm is used on restore.

By default, the design snapshots are not compressed in the ICE mode. You must specify the `-compress` option with the `save` command to compress the saved snapshots.

Note: If you do not specify the compression level (that is, `L`, `M`, `H`, or `N`) with the `-compress` option, compression is done at low level.

In the SA mode, the simulator database is compressed by default. Therefore, the `-compress` option is not supported in the SA mode and is ignored with the following warning:

WARNING (legacy-52437): The -compress feature is not supported in ixcom mode since ixcom save databases are compressed by default. It will be ignored.

-overwrite

Specifies to overwrite an existing snapshot. Without this option, saving the state with an existing name will fail.

Note: This option is not supported with vvmDebug.

{-commands | -environment} [<filename>]

Saves the run-time environment into a Tcl script file with the specified <filename>. When the saved Tcl script is sourced, it recreates the current state of the run-time environment, which includes things such as, symbols, probes, clock settings, SDL settings, user defined Tcl variables used in the SDL program, and predefined run-time parameters.

Note: The -commands and -environment options are supported in both SA and ICE modes.

If the <filename> argument is omitted, the output of the command is returned as a standard Tcl result.

-uservars <filename>

Saves the user-defined Tcl variables into a Tcl script file with the specified <filename>. The file generated can later be sourced to restore the Tcl variables.

Note: This option is not supported with vvmDebug.

Examples

The following example saves the run-time environment as a Tcl script, then restores it:

```
XE> save -environment sv_env.tcl      << environment is saved  
XE> sdl -setFile new.tdf            << environment is changed  
XE> source sv_env.tcl              << environment is restored from the file
```

The following example saves the states of the current session in STB mode:

```
XE> run 1000  
XE> getCycleNum  
1000
```

UXE Command Reference Manual

Run-time Commands

```
XE> save state0
XE> run 1000
XE> getCycleNum
2000
```

The following example saves the user-defined Tcl variables `var1` and `var2` to the Tcl script file `usave.tcl`:

```
XE> set var1 value1
XE> set var2 value2
XE> save -uservars usave.tcl
```

To restore the values for `var1` and `var2`, source the Tcl file `usave.tcl` as following:

```
XE> source usave.tcl
```

scope

Specifies the setting and usage of a debug scope. The debug scope states the design hierarchy that is to be used as default in some other commands during the debug session. It also enables some level of access to the design hierarchy information.

The debug scope affects the specification of objects in subsequent commands that make use of design object names, such as `force`, `deposit`, `memory`, `probe`, `release`, and `value`. In such commands, if the object is specified without a hierarchical path, it is searched in the debug scope. If not found there, it is searched in the top hierarchy. This is true whether the object resides in the testbench or the DUV.

If the `scope` command is executed without any arguments, it returns the current value of the debug scope.

The syntax for this command is:

```
scope [-set] <scope_name>  
scope -up  
scope -describe [<scope_name>]  
scope -show
```

[-set] <scope_name>

Sets the current debug scope to the specified scope. If `<scope_name>` is not specified, the command returns the value of the current debug scope. The `-set` option is optional.

-up

Changes the current debug scope to one level up in the hierarchy.

For example, if the current debug scope is `top.dut`, then the `scope -up` command will change the scope to `top`.

-describe [<scope_name>]

Returns a formatted description of all the objects in the debug scope or in the specified scope.

Note: In ICE mode, the `-describe` option of the `scope` command is not supported.

UXE Command Reference Manual

Run-time Commands

-show

Returns the scope information, including the current debug scope, list of instances within the debug scope, and top-level modules in the currently loaded model.

Note: In ICE mode, the `-show` option of the `scope` command is not supported.

sdl

This command provides a single interface for XEL to access the SDL-related features.

SDL controls when to stop the run and when to take probe samples. It also provides other debug facilities, such as conditionally executing Tcl commands during the run based on design state. For details on how to use SDL, refer to the *Controlling the Run with SDL* chapter in the *UXE User Guide*.

All the options of the `sdl` command are case-insensitive (can be specified in either lower or upper case), and can be abbreviated to the shortest string possible as long as a unique match exists. For example, `sdl -getf` is accepted as an abbreviation to `sdl -getFile`, but `sdl -get` is not accepted as a legal command because it does not have a unique match; it can match `sdl -getFile` or `sdl -getInstances` (and a few others).

The syntax for this command is:

```
sdl -autoProbe [no | yes]  
sdl -addProbes [-fast] <sdl_file>...  
sdl -disable -save <filename>  
sdl -disable -list  
sdl -disable [-now]  
sdl -disable [-now] [<instance_names_list> | -none]  
sdl -disable [-now] -label <label-pattern> <label-pattern> ...  
sdl -disable -label -list  
sdl -disable -stream <stream-pattern> <stream-pattern> ...  
sdl -disable -stream -list  
sdl -display -console [0 | 1]  
sdl -display -endofline 0 | 1  
sdl -display -file [<file_name>]  
sdl -display -overflow [error | warning | ignore]  
sdl -display -translate 0 | 1  
sdl -display -id <ID> [-file <file_name>]  
sdl -display -id <ID> -disable | -enable  
sdl -display -loglevel <loglevel>  
sdl -display -deflevel <deflevel>  
sdl -dumpKeepNets [-exclude] <sdl_file> <script_file>
```

UXE Command Reference Manual

Run-time Commands

sdl -enable -list
sdl -enable [-now]
sdl -enable [-now] [<instance_names_list> | -none]
sdl -enable [-now] -label <label-pattern> <label-pattern> ...
sdl -enable -label -list
sdl -enable -stream <stream-pattern> <stream-pattern> ...
sdl -enable -stream -list
sdl -expression [<expression>]
sdl -execOvf [error | warning | ignore]
sdl -getActiveInstances
sdl -getAssertions
sdl -getFiles
sdl -getInstances
sdl -getLabels [-detail]
sdl -getNets
sdl -getStatus [-instance <instance_name>] <parameter>
sdl -getTriggers
sdl -haltOnTrigger [0 | 1]
sdl -isActive
sdl -isDispOvf [-clear]
sdl -isExecOvf [-clear]
sdl -isTrigger [-instance <instance_name>] [-label <label>]
sdl -ldcnt1 | -ldcnt2 [-instance <instance>] <value>
sdl -load [<sdl_file_name>]
sdl -log -stopStream | -startStream | -upload [<size>] | -reset
sdl -nacq <expression>
sdl -ncbreakpoints -errorok [yes | no]
sdl -ncbreakpoints -import [<filename>]
sdl -ncbreakpoints -exact [yes | no]
sdl -onlyReport [yes | no]
sdl -pp <output_file> [<sdl_file_name>]
sdl -report [-file <file_name>]

UXE Command Reference Manual

Run-time Commands

```
sdl -restoreState <file>
sdl -saveState <file>
sdl -scope [<scope>]
sdl -setFile [<sdl_filename>]
sdl -simTime yes | no
sdl -stop
sdl -traceClear run | dump | never
sdl -traceDump [-location] [-lineOnly] [-short] [-index c | dc | t | dt] [-instance
    <instance_name>] [-text | -fsdb | -sst2] [-time <time_unit>] <file_name>
sdl -traceOn [-acqFilter no | yes] [-postTrigger no | yes] [<action_list> | ALWAYS
    | DEFAULT]
sdl -tx {-sst2 | -text | -ida} <filename> | -close | -show
sdl -tx -illegal [error | warning | ignore | reportwarning | clearwarning]
sdl -tx -statistics txid | scope | stream | type
sdl -verify [<filename>]
sdl -when [-time] [<time_unit>]
```

In vvmDebug, the following command options are not supported:

```
sdl -display ...
sdl -getAssertions
sdl -getLabels ...
sdl -haltOnTrigger ...
sdl [-ldcnt1 | -ldcnt2] ...
sdl -log ...
sdl -ncbreakpoints ...
sdl -onlyReport ...
sdl -restoreState ...
sdl -saveState ...
sdl -stop
sdl -traceClear ...
sdl -traceDump ...
sdl -traceOn ...
sdl -tx ...
```

In vvmDebug, the `-now` option is not required in the following commands as disabling is always instant in vvmDebug:

```
sdl -disable ...
sdl -enable ...
```

In vvmDebug, some other commands behave as follows:

- The `sdl -execOvf` command is ignored as EXEC overflow cannot happen in VVM.
- The display buffer overflow cannot happen in vvmDebug, therefore, the `sdl -isDispOvf` command always returns 0.
- The EXEC overflow cannot happen in vvmDebug, therefore, the `sdl -isExecOvf` command always returns 0.
- The `-instance` and `-label` options of the `sdl -isTrigger` command are not supported in vvmDebug.

sdl -autoProbe [no | yes]

Automatically probes all nets, assertions, and power conditions used in either the SDL program or the independent trigger expression (which is specified by sdl -expression [<expression>]).

If this option is set to `yes` and SDL is enabled, nets and assertions used in the SDL program will be probed every time a `run` command is issued. If you change the SDL program and issues another `run` command, the set of probed nets and assertions will be re-adjusted according to the new contents of the SDL program.

If a request is made to see the list of probed nets and assertions (for example, through the `probe -show` command), they will be grouped under the name `SDL_PROBED_NETS`.

This option applies only to nets and assertions that are actually being used. Nets or assertions that appear in alias definitions that are not used, are not probed.

Power conditions are added as independent probes (not grouped under `SDL_PROBED_NETS`), and they are not deleted when the SDL program is replaced with a different one.

If the option is set to `no`, this feature is turned off.

The values `0`, `false`, and `off` are accepted as synonyms for the keyword `no`; while `1`, `true`, and `on` are recognized as synonyms for the keyword `yes`.

Without the `no` or `yes` argument, the command returns the current settings, as either `0` or `1`.

Initially, the autoprobe feature is turned off.

sdl -addProbes [-fast] <sdl_file>...

Extracts all nets, assertions, and power conditions from the list of specified SDL files and adds the objects as probes.

The command does not interfere with or affect the currently active SDL program.

This command is useful (with the `-fast` option) to add fast probes for the purpose of using the waveform database later in VVM or offline with the specified SDL files. When you use this command in the VVM Prepare or online session, then waveforms and calculation of SDL expressions based on the fast probes will be done faster in the following vvmDebug or offline session. The command can be called multiple times with different SDL files, and all the objects from all the files are added as probes.

All the added probes remain in effect until you explicitly remove the probes.

sdl -disable -save <filename>

Creates a file (with specified name `<filename>`) containing a list of XEL commands that reproduce the current enabled/disabled state of all instances and labels in the SDL program. This file can be executed later by using the following Tcl command:

```
source <filename>
```

sdl -disable -list

Returns a list of the disabled SDL instances (that is, the [Disabled Instances List](#)) as a Tcl list.

sdl -disable [-now]

Disables all SDL components in the [Uncommitted Instances List](#). This means disabling all SDL instances that do not belong to the [Enabled Instances List](#), as well as all the TRIGGER, EXEC, ACQUIRE and NO_ACQUIRE statements in the Global Definition Section of the SDL program. Disabling an Instance means only disabling the instance from affecting emulation. This means disabling all TRIGGER, EXEC, ACQUIRE and NO_ACQUIRE actions inside the instance, but not other actions, such as GOTO or DECREMENT.

For more information on the various sections of a SDL program, refer to the *SDL Language Constructs* section in the *Controlling the Run with SDL* chapter of the *UXE User Guide*.

This command does not change the contents of the [Disabled Instances List](#).

This command does not affect triggering through the `-expression` option.

For details on the purpose of the `-now` option and additional information, refer to the [Additional Information on Enabling and Disabling SDL](#) section.

sdl -disable [-now] [<instance_names_list> | -none]

`<instance_names_list>` specifies a list of SDL instances (called [Disabled Instances List](#)) that will be disabled. If the Disabled Instances List already exists from a previous execution of this command, the new names specified by `<instance_names_list>` are appended to the existing list.

The Disabled Instances List might only contain instance names that exist in the currently loaded SDL program. Therefore loading a new SDL program might cause instance names that do not exist in the new SDL program to be removed from the existing list. Similarly if the command specifies an instance name that is not found in the currently loaded SDL program, the request is ignored with a warning message.

Disabling an instance means only disabling the instance from affecting emulation. This means disabling all TRIGGER, EXEC, ACQUIRE and NO_ACQUIRE actions inside the instance, but not other actions, such as GOTO or DECREMENT.

Any name from `<instance_names_list>` is also removed from the [Enabled Instances List](#) (if found there). This ensures that the contents of the Enabled and Disabled Instances Lists are mutually exclusive.

The `-none` option removes all the elements from the Disabled Instances List.

For details on the purpose of the `-now` option and additional information, refer to the [Additional Information on Enabling and Disabling SDL](#) section.

sdl -disable [-now] -label <label-pattern> <label-pattern> ...

Disables all trigger, display, and exec statements whose label matches any one of the `<label-pattern>` arguments. A single list of disabled labels is maintained and the `sdl -disable -label` command adds the specified labels to this list.

Since SDL does not restrict labels to be unique, a request to disable a label disables all the trigger and exec statements that have the specified label.

Trigger, Display, or Exec statements for which no label is specified cannot be disabled by this option.

Although the software will maintain the list of disabled labels across SDL loading, the list might only contain labels that exist in the currently loaded SDL program. Therefore, loading

a new SDL program might cause labels that do not exist in the new SDL program to be removed from the existing list. Similarly, if the command specifies a label that is not found in the currently loaded SDL program, the request is ignored with a warning message.

Here, *<label-pattern>* can be specified in one of the following formats:

- *<instance>.<label>*

Targets only labels that match *<label>*, and are located inside instances with name that match *<instance>*.

- *.<label>*

Targets only labels that match *<label>*, and are located in the global definition section, that is, outside any instance definition.

- *<label>*

Targets all labels that match *<label>*, regardless of whether they belong to an instance or not.

The strings *<label>* and/or *<instance>* can be a glob pattern (wildcard), following the same semantics as Unix shell. This means that:

- * matches any string.
- ? matches any single character.
- [...] matches a single character that appears inside the bracket.

The expansion of glob patterns is carried out by adding all the matching labels in the current SDL program, to the list of disabled labels.

For example:

```
sdl -disable -label ABC.?EF X* *.LGO
```

The above command disables:

- All Trigger, Display, and Exec statements inside instance ABC whose label has 3 characters of which the last two are EF,
- All Trigger, Display, and Exec statements (anywhere) whose label starts with the character X, and
- All Trigger, Display, and Exec statements whose label is LGO, except those that are located in the global definition section.

For details on the `-now` option and additional information, refer to the [Additional Information on Enabling and Disabling SDL](#) section.

sdl -disable -label -list

Returns a list of the disabled labels as a Tcl list.

sdl -disable -stream <stream-pattern> <stream-pattern> ...

Allows to disable specific TX streams.

The output from any TX stream whose stream name matches `<stream-pattern>` is suppressed.

The `<stream-pattern>` is a glob pattern (wildcard) that follows the same semantics as UNIX shell. This means that:

- * matches any string.
- ? matches any single character.
- [...] matches a single character that appears inside the bracket.

sdl -disable -stream -list

Returns the names of currently disabled TX streams as a Tcl list.

sdl -display -console [0 | 1]

By default, all messages that are generated by `DISPLAY` statements are printed to the console.

Specifying the value:

- 0 disables printout of `DISPLAY` messages to the console. This affects all `DISPLAY` messages,
Note: `no`, `off`, and `false` are accepted as synonyms for 0.
- 1 re-enables printout of `DISPLAY` messages to the console.
Note: `yes`, `on`, and `true` are accepted as synonyms for 1.

If neither 0 nor 1 is specified, the command returns the current settings (0 or 1).

sdl -display -endofline 0 | 1

By default, an end-of-line character is inserted automatically at the end of each message coming from a display statement just like in the case of the Verilog \$display system task.

With the `-endofline` option, specify the argument as:

- 0 to disable the automatic insertion of end-of-line character.
- 1 to ensure the addition of the end-of-line character.

Here, no, off, false are accepted as synonyms for 0, and yes, on, true are accepted as synonyms for 1.

sdl -display -file [<file_name>]

Opens for write the file specified by `<file_name>` and writes display messages into it from that point on. If a file was previously opened for writing display messages with this command, then it is first closed before opening the new file. If it is the same file name, then this command will close and re-open the file. This will cause the previous data in the file to be deleted.

The command returns the Tcl channel ID of the file. The Tcl channel ID is a string that can be used to access the file using standard Tcl commands for buffered I/O. For example, the `Tcl puts` command can be used to add text into the same file, as in the following example:

```
set disp_fid [sdl -display -file dispout.txt]
puts $disp_fid {This file will contain output from SDL DISPLAY statements}
```

To stop writing to the file and close it, use {} or "" instead of `<file_name>`.

If the argument `<file_name>` is missing altogether, the command only returns the Tcl channel ID of the file, or an empty string if no file is opened for display output.

This command does not affect printout of Display messages to the console. This command applies only to Display actions in SDL for which a F=<ID> field was either not specified at all, or specified but not associated with any output file.

sdl -display -overflow [error | warning | ignore]

Defines how the system should behave when a display buffer overflow condition happens. A display buffer overflow condition might happen in Logic Analyzer or Dynamic Target mode if

the SDL program generates a burst of display messages in a short period of time such that it fills up the display buffer before the workstation can read and process it.

A display buffer overflow can happen only in Logic Analyzer or Dynamic Target mode, because these modes do not enable to pause the design clocks.

- **error** - Generate an error message and stop the run.
- **warning** - Generate a warning message and continue the run.
- **ignore** - Ignore the overflow condition.

The default is **error**.

If **warning** is selected, a warning is generated when an overflow condition is detected first time in the run. No further warnings are issued during that run unless the overflow flag has been explicitly cleared with the command `sdl -isDispOvf -clear`.

The current setting (one of: **error**, **warning**, **ignore**) is returned if no option is specified.

sdl -display -translate 0 | 1

By default, the `%c` and `%s` format conversions of display actions write out the raw ASCII values without any translation. In some cases, this might be undesirable when the resulting characters are non-printable, especially when the message is sent to the console.

The `sdl -display -translate` command controls whether non-printable characters resulting from `%c` and `%s` format conversions are translated to the character “?”.

The value 1 will cause translation of non printable characters to “?”. The value 0 will use the raw values.

Here, `no`, `off`, `false` are accepted as synonyms for 0, and `yes`, `on`, `true` are accepted as synonyms for 1.

sdl -display -id <ID> [-file <file_name>]

Controls the destination of Display messages that contain a `F=<ID>` argument.

If a Display statement contains `F=<ID>` argument, and `<ID>` is associated with an open file, then, in addition to being printed to the console, the message from the Display statement is also sent to that file. The association of `<ID>` with a file is established by the following command:

```
sdl -display -file <file_name> -id <ID>
```

OR:

```
sdl -display -id <ID> -file <file_name>
```

<ID> is a user given string that must follow the same lexical rules as other SDL identifiers.

<file_name> is the name of the output file.

The command returns the TCL channel ID of the file. The TCL channel ID is a string that can be used to access the file using standard TCL commands for buffered I/O. For example, the TCL puts command can be used to add comments into the same file.

If <file_name> is specified as an empty argument (like " " or {}), then this removes the association between the <ID> and file. If the file is not associated with any other <ID>, then it is closed.

If -file <file_name> is missing, the command only returns the TCL channel ID of the file that is associated with <ID>, or an empty string if no opened file is associated with <ID>.

The file is opened for write by this command, unless it is already opened because it is associated with another <ID>.

A single file can be associated with more than one <ID>. To do that, call this command multiple times using same <file_name> but different values for <ID>. In such case, Display messages that were specified with different F=<ID> arguments will be sent to the same file.

If an <ID> is not associated with an open file, the destination for the Display message is the same as for Display messages that were defined without F=<ID>.

sdl -display -id <ID> -disable | -enable

Disables (or re-enables) all messages from Display statements with the specified <ID>.

sdl -display -loglevel <loglevel>

Enables or disables messages from Display statements based on L=<level> argument:

Here, <loglevel> is a non-negative integer number. Any Display statements with value of <level> below this number are not logged. The initial value for <loglevel> is 0.

sdl -display -deflevel <deflevel>

Sets the default *<level>* value for Display statements. Here, *<deflevel>* is the default level number that would be assigned for Display statements that do not have *L=<level>* argument. This must be a non-negative integer value. The initial value for *<deflevel>* is 0.

sdl -dumpKeepNets [-exclude] <sdl_file> <script_file>

Creates a new file with the name *<script_file>* and writes into that file a script with the *keepNet -add <net>* commands covering all net names used in the sdl file *<sdl_file>*. If *<script_file>* already exists, then it is overwritten.

The *-exclude* option causes the generated file to exclude all nets that are either FF outputs or were already defined as keepnets. The *-exclude* option is optional.

The *sdl -dumpKeepNets* command can be executed anytime after the *host* command in both SA and ICE modes. Downloading the design is not mandatory before executing this command.

sdl -enable -list

Returns a list of the enabled SDL instances (that is, the [Enabled Instances List](#)) as a Tcl list.

sdl -enable [-now]

Enables all SDL components in the [Uncommitted Instances List](#). This means that all instances not belonging to the [Disabled Instances List](#), as well as all the TRIGGER, EXEC, ACQUIRE, and NO_ACQUIRE statements in the Global Definition Section of the SDL program, except those that were individually disabled through the command *sdl -disable -label*, are enabled using this command.

This command does not change the contents of the [Enabled Instances List](#).

This command does not affect triggering through the *sdl -expression [<expression>]* command.

For details on the purpose of the *-now* option and additional information, refer to the [Additional Information on Enabling and Disabling SDL](#) section.

sdl -enable [-now] [<instance_names_list> | -none]

<*instance_names_list*> specifies a list of SDL instances (called Enabled Instances List) that will be enabled in the next run. If the Enabled Instances List already exists from a previous execution of this command, the new names specified by <*instance_names_list*> are appended to the existing list. If an instance name in the list is not found in the currently loaded SDL program, the request is ignored with a warning message.

Any name from <*instance_names_list*> is also removed from the Disabled Instances List (if found there). This ensures that the contents of the Enabled and Disabled Instances Lists are mutually exclusive.

The -none option removes all the elements from the Enabled Instances List.

For details on the purpose of the -now option and additional information, refer to the Additional Information on Enabling and Disabling SDL section.

sdl -enable [-now] -label <label-pattern> <label-pattern> ...

Re-enables the labeled Trigger, Display, and Exec statements that were previously disabled by the `sdl -disable -label` command. Specifying as a label pattern the name of a label that was not previously disabled has no effect. Specifying a label pattern that does not match any label in the currently loaded SDL program is ignored with a warning message.

To ensure that a Trigger, Display, or Exec statement is active during a run, it is essential that both the individual Trigger, Display, or Exec statement and the instance in which it is located are not disabled.

Here, <*label-pattern*> can be specified in one of the following formats:

- <*instance*>.<*label*>

Targets all labels that match <*label*> and are located inside an instance whose name match <*instance*>.

- .<*label*>

Targets only labels that match <*label*>, and are located in the global definition section, that is, outside any instance definition.

- <*label*>

Targets all labels that match <*label*>, regardless of whether they belong to an instance or not.

The strings `<label>` and/or `<instance>` can be a glob pattern (wildcard), following the same semantics as Unix shell. This means that:

- * matches any string.
- ? matches any single character.
- [...] matches a single character that appears inside the bracket.

The expansion of glob patterns is done by scanning the list of disabled labels.

For details on the `-now` option and additional information, refer to the [Additional Information on Enabling and Disabling SDL](#) section.

sdl -enable -label -list

Returns a list of all labels that are currently enabled.

The list reflects the current state of the SDL program that is downloaded into the hardware. For example, if a specific SDL instance is currently disabled, all the labels in that instance will not appear in the returned value.

Items in the returned list are in the format `INSTANCE . LABEL`, where `INSTANCE` is the name of the instance and `LABEL` is the name of the label.

If a label is located in the global definition section (above any state or instance statement), `.LABEL` is returned as a string.

sdl -enable -stream <stream-pattern> <stream-pattern> ...

Re-enables TX streams that were previously disabled by the `sdl disable -stream` command. Any TX stream whose stream name matches `<stream-pattern>` is re-enabled.

The `<stream-pattern>` is a glob pattern (wildcard) that follows the same semantics as UNIX shell. This means that:

- * matches any string.
- ? matches any single character.
- [...] matches a single character that appears inside the bracket.

sdl -enable -stream -list

Returns names of TX streams that are currently enabled as a Tcl list.

sdl -expression [<expression>]

Specifies an independent trigger expression that causes a trigger when true. This trigger is not affected by the -enable or -disable options, and is not considered part of the SDL program. It only serves as a simple and quick way to define a trigger condition without an SDL program.

The syntax of <expression> follows regular SDL expression syntax for global expressions.

If <expression> is an empty string (that is, an empty pair of double quotes or curly braces), the independent trigger expression is disabled.

Without the <expression> argument, the command only returns the current expression.

sdl -execOvf [error | warning | ignore]

Specifies how the system should behave when an EXEC overflow happens.

EXEC overflow might happen in any operation that supports SDL EXEC, except STB mode. This includes LA, Dynamic Target, and SA modes. SDL has an internal buffer that can hold EXEC requests from up to 4 separate FCLK cycles. If EXEC requests are issued when that buffer is full, then the new EXEC requests are lost. This condition is called EXEC overflow.

- error - Generate an error message and stop the run.
- warning - Generate a warning message and continue the run.
- ignore - Ignore overflow conditions.

The default is error.

If warning is selected, then each run command might create one warning message at the first occurrence of the overflow condition in that run. No further warnings are issued during that run unless the overflow flag has been explicitly cleared with the `sdl -isExecOvf -clear` command.

The current setting (one of: error, warning, ignore) is returned if no option is specified.

UXE Command Reference Manual

Run-time Commands

One case where `EXEC` overflow might happen is in operation modes that employ free running clocks that cannot be stopped. This includes LA mode, Dynamic Target mode, and designs that have uncontrolled clocks (under IXCOM).

Another case where `EXEC` overflow might happen is in SA mode. In SA mode, `EXEC` requests can only be serviced at the end of each simulation time slot. If the amount of cycles with `EXEC` requests that are scheduled in a single time slot exceeds the size of the `EXEC` buffer, then this will also cause an `EXEC` overflow. One way to avoid `EXEC` overflow in this operation mode is to make sure that SDL is evaluated no more than once per each simulation time stamp. This can be done with the following XEL command:

```
sdl -simTime 1
```

sdl -getActiveInstances

Returns list of SDL instances that are currently active (and therefore might affect the results). Keep in mind that even if SDL was disabled globally with the command `sdl -disable`, any instance that was explicitly enabled through the command `sdl -enable <instance>` is still going to remain active.

sdl -getAssertions

Returns a list of all assertions in the currently loaded SDL program.

sdl -getFiles

Returns a list of all the SDL source files used in the currently loaded SDL program. The list includes the main trigger file and all the files that were included directly or indirectly with the `'include` directive.

sdl -getInstances

Returns a list of names of SDL instances in the currently loaded SDL program.

If the SDL file has only STATE statements but no INSTANCE statement, or if SDL is inactive, an empty string is returned.

sdl -getLabels [-detail]

Returns a Tcl list of all labels that are defined in the currently loaded SDL program.

The `-detail` option, if specified, prefixes each label with the instance name. For example,
`instance.label`

sdl -getNets

Returns a list of all nets that are used by currently loaded SDL program.

sdl -getStatus [-instance <instance_name>] <parameter>

Returns the status value of the SDL instance specified by `<instance_name>` from the SDL instrumentation. If a specific `<instance_name>` is not specified, the first instance in the program is used by default.

The `<parameter>` specifies the type of status information to be returned. It must be one of the following keywords:

- `state`
- `counter1` or `cntr1`
- `counter2` or `cntr2`
- `all`

`state` returns the state name, while `counter1` or `counter2` returns the value of the corresponding SDL counter using decimal radix. `all` returns a Tcl list with all three parameters in the above order.

An empty string is returned if SDL is inactive.

sdl -getTriggers

Returns a list of all the triggers that fired in the last run. For each trigger fired, the following information is returned as a sub-list:

- Trigger file name
- Line number in the file
- Instance name
- Label (if defined in the trigger file)

An empty string is returned if SDL is inactive, or if there was no trigger in the last run.

Note: The list returned by the `sdl -getTriggers` command does not include any information on trigger by expression (that is, the independent trigger expressions that are specified using the `sdl -expression <expressions>` command.)

sdl -haltOnTrigger [0 | 1]

By default, when running in Logic Analyzer or Dynamic Target modes, a TRIGGER action stops only waveform tracing, but not emulation clocks. The `sdl -haltOnTrigger` command enables to override the default behavior, such that a TRIGGER action stops the emulation clocks even when running in Logic Analyzer or Dynamic Target mode. This command applies only to Logic Analyzer and Dynamic Target modes, and has no effect in other emulation modes. The `sdl -haltOnTrigger` command is disabled in designs that have uncontrolled clocks.

Specifying the value:

- 0 stops only waveform tracing.

Note: off, false, and no are accepted as synonyms for 0.

- 1 stops both design clocks and waveform tracing.

Note: on, true, and yes are accepted as synonyms for 1.

When called without the argument 0 | 1, the function returns the current settings as 0 or 1. This command can be called while a run is ongoing (when running in non-blocking mode), and it takes effect immediately when executed.

sdl -isActive

Returns the string 0 if SDL is not active, and the string 1 if SDL is active.

SDL is considered active if an SDL program is loaded into the emulator.

Some operations, such as the `sdl -enable -now` command, can only be executed when SDL is active.

Lists of disabled/enabled SDL components (instances or labels) are maintained only for an active SDL program. Once the program becomes inactive (by unloading it from the emulator), these lists are purged.

SDL can change only between active and inactive states while executing one of the following three commands:

UXE Command Reference Manual

Run-time Commands

- run
- infiniTrace -goto trigger
- sdl -load

The `run` and `infiniTrace -goto trigger` commands will compile and load the SDL program if SDL was enabled in general (with the `sdl -enable` command) or if any component of the previously active SDL program was enabled. After the SDL program is successfully compiled and loaded into the emulator, it becomes active.

The `sdl -load` command will compile and load the SDL program (and make it active) whether SDL is enabled or not.

An already active SDL program might become inactive under several conditions, but a minimum requirement for this to happen is that SDL must be entirely disabled (or SDL file name is set to an empty string) while executing one of the above three commands. Under some conditions, the software might decide to continue and keep the SDL program loaded even when disabled, and in such case the `sdl -isactive` command will return 1.

To disable all SDL components, use both of the following commands:

```
sdl -disable  
sdl -enable -none
```

To enable any of the SDL components, use either of the following commands:

```
sdl -enable  
sdl -enable <instance_names_list>
```

Note: The value returned by the `sdl -isActive` command refers only to the SDL files. It does not take into account the independent trigger expression that is defined using the `sdl -expression <expressions>` command.

sdl -isDispOvf [-clear]

Enables you to check whether an overflow condition happened, and optionally clear it.

The return value from this command is:

- 1 – Indicates that the display buffer overflow happened.
- 0 – Indicates that the display buffer overflow did not happen.

If the `-clear` option is specified, then the command clears the overflow condition if one exists. Starting a new run or switching to a different run mode (for example, from LA to STB) also clears the overflow condition.

UXE Command Reference Manual

Run-time Commands

The returned value therefore indicates whether an overflow happened in the last run, or since the last execution of `sdl -isDispOvf -clear`, whichever happened last.

The value returned by `sdl -isDispOvf` is independent of the settings of `sdl -display-overflow`.

Clearing the overflow flag (with `sdl -isDispOvf -clear`) while the run is in progress might potentially cause some display overflow conditions to happen unreported while the command is being executed.

sdl -isExecOvf [-clear]

Enables you to check whether an `EXEC` overflow condition happened, and optionally clear it.

It returns either 1 (overflow happened) or 0 (no overflow).

If the `-clear` switch is specified, then the command clears the overflow condition if one exists. Starting a new run or switching to a different run mode (for example, from LA to STB) also clears the overflow condition.

The returned value indicates whether an overflow happened in the last run, or since the last execution of `sdl -isExecOvf -clear`, whichever happened last.

The value returned by `sdl -isExecOvf` is independent of the settings of `sdl -execOvf`.

For more information on `EXEC` overflow conditions see description of [sdl -execOvf \[error | warning | ignore\]](#).

sdl -isTrigger [-instance <instance_name>] [-label <label>]

Checks if a trigger fired in the most recent run. The `-instance <instance_name>` option specifies the SDL instance and the `-label <label>` option specifies the label of a specific trigger action.

If only `-instance` option is specified, the command returns 1 if the trigger was caused by the specified instance name, and 0 otherwise.

If only `-label` option is specified, the command returns 1 if the trigger was caused by a trigger action or a global trigger with the specified label, and 0 otherwise.

If both `-instance` and `-label` options are specified, the command returns 1 if the trigger was caused by a trigger action with the specified label from the specified SDL instance, and 0 otherwise.

UXE Command Reference Manual

Run-time Commands

If no option is provided with `-isTrigger` and the last run was stopped because of a trigger from either the SDL program or the independent trigger expression (as specified by `sdl -expression [<expression>]`), the command returns 1; otherwise, 0 is returned.

sdl -ldcnt1 | -ldcnt2 [-instance <instance>] <value>

Loads the specified value into SDL counter1 or counter2 from the command line even while SDL is running.

The `-instance` option is mandatory if the SDL program has more than one instances.

Note: The execution of a `run` command without a `-continue` option resets all the SDL counters to 0. This means that any value assigned to the counters before beginning the design run will be lost. Therefore, the `-ldcnt1` and `-ldcnt2` options of the `sdl` command have an effect only in two conditions:

- The options are executed during the run.

OR

- The design is run using the `run -continue` command that preserves the SDL state.

sdl -load [<sdl_file_name>]

Compiles and loads the SDL program immediately into the emulator. If this command is not executed, the SDL program is compiled and loaded when needed, for example, upon execution of the `run` or `infiniTrace -goto` trigger commands. If an independent trigger expression was defined by `sdl -expression [<expression>]`, it is also compiled and loaded. If a run is in progress, this command will stop the run before loading the new SDL program.

After executing this command, information on the new SDL program becomes available to XEL through `sdl` command options such as `-getFiles`, `-getInstances`, `-getNets`, `-getAssertions`, `-getLabels`, and others, even if SDL is disabled.

Once the `sdl -load` command is executed, any information on SDL status from the previous run (like which trigger happened) is lost.

Also note that `sdl -load` unconditionally load the SDL program into the emulator (excluding errors), whether `sdl` is enabled or not. On the other hand if all the SDL components are disabled, a `run` command will not load the SDL program.

The `sdl -load` command provides a more comprehensive way to verify the SDL program than the `sdl -verify` command.

The `sdl -verify` command only checks for syntax and semantics errors in the SDL program. However, loading of the SDL program into the emulator might still fail at a later stage due to other reasons, such as lack of sufficient emulation resources.

If the optional argument `<sdl_file_name>` is specified, a new SDL main file with the specified name is selected. This means that the command:

```
sdl -load <sdl_file_name>
```

is equivalent to the following sequence:

```
sdl -setFile <sdl_file_name>
```

```
sdl -load
```

sdl -log -stopStream | -startStream | -upload [<size>] | -reset

Specifies how to process the data being generated by `DISPLAY` and `TX` statements in the SDL program. The `DISPLAY` and `TX` features share the same hardware buffer for data capture. They also share the same set of XEL commands for controlling the way data is captured and processed. By default, data from the `DISPLAY` and `TX` statements is continuously streamed into the specified output file(s). This behavior can be changed to a non streaming mode in which data is written to the specified output file only when you request it.

- `sdl -log -stopStream`

The `sdl -log -stopStream` command stops data streaming. It affects both display and transaction recording. The command stops the data streaming, although the data continues to be captured into the internal hardware buffer and overwrite older data without producing any errors or warnings.

The data from the buffer can be uploaded at any time with the `sdl -log -upload` command.

- `sdl -log -startStream`

The `sdl -log -startStream` command will cause all future `run` commands to stream the data to the output files. Any existing data in the internal hardware buffer remaining from previous runs will be discarded at the beginning of the next run.

- `sdl -log -upload [<size>]`

The `sdl -log -upload [<size>]` command reads the data from the hardware buffer into the workstation. It is applicable only when the data streaming was stopped earlier with the `sdl -log -stop` command.

The `<size>` limits the amount of data and specifies the number of data points to upload. It is optional. If not specified, the entire buffer is uploaded. A single data point represents any number of TX or display statements that happened in the same emulation cycle.

■ **`sdl -log -reset`**

The `sdl -log -reset` command discards the contents of the hardware buffer. It is applicable only when the data streaming was stopped earlier with the `sdl -log -stop` command. The contents of the hardware buffer are also discarded at the beginning of each run, except `run -continue`, which does not discard the previous contents of the hardware buffer.

`sdl -nacq <expression>`

Specify an independent conditional acquisition expression. Waveform acquisition is turned off in each cycle in which `<expression>` is true, regardless of any other acquisition filter specified inside the SDL program. The syntax used for `<expression>` must follow the same syntax rules for SDL expressions, except that it may not contain SDL counters.

To disable this feature use the value `{}` for `<expression>`.

`sdl -ncbreakpoints -errorok [yes | no]`

Ignores errors when importing NC-Sim breakpoints into SDL with the `sdl -ncbreakpoints -import [<filename>]` command. Setting this value affects all subsequent breakpoints import operations, including automatic import during swap-in under IXCIM (refer to [sdlAutoNcBreakpoints](#) for more detail).

- `yes`: Ignore breakpoints that generated errors during the import, and enable the import operation to complete successfully with other breakpoints that did not generate errors.
- `no`: If any breakpoint generated an error condition during import, the import operation fails and none of the breakpoints are imported to SDL.

The default value is `no`.

Note: The values `true`, `on`, and `1` are treated as substitutes of `yes`, and the values `false`, `off`, and `0` are accepted as substitutes of `no`.

If neither `yes` or `no` is specified, the command returns the current setting as either `1` or `0`.

sdl -ncbreakpoints -import [<filename>]

Imports ncsim breakpoints (that were created with the ncsim `stop` command) into SDL.

If `<filename>` is not specified, the command behaves as follows:

It translates all the breakpoints (that can be translated) into an SDL file with the name `ncbreak.tdf`, and then compiles and loads the file into the Palladium hardware if the translation is successful.

The command creates a second file `ncbreak.tcl`, containing a Tcl script with definitions that are needed for setting the initial values of the `delete` and `skip` counts. Users can source this file any time during the run if they want to reset the delete and skip counts to their original values, but otherwise they do not need this file, because the command sources the contents of this file into the Tcl interpreter. The contents of this file only affect the skip and delete counts as processed by SDL on Palladium XP, but has no effect on the NC-Sim simulator.

If `<filename>` is specified, the result of the import is written into the files `<filename>.tdf` and `<filename>.tcl` instead of `ncbreak.tdf` and `ncbreak.tcl`, but the run-time environment is not affected, which means that the Tcl variables that hold the skip and delete counter values are not modified and the SDL program in `<filename>.tdf` is not enabled or loaded into the emulator.

In either case the command returns a list of the identifiers of all the breakpoints that were imported into the SDL file.

See also description of the [sdlAutoNcBreakpoints](#) run-time parameter.

sdl -ncbreakpoints -exact [yes | no]

Enables you to control the translation semantics when importing Condition type breakpoints to SDL.

- yes: Use exact ncsim's event driven semantics when translating Condition type breakpoints to SDL. These semantics mean that the breakpoint stops the simulation when the condition is true and there is some event related to the condition. An event is generated when any of the wires in the condition toggling, and also in any value change of a vector, if any bit of that vector is used in the condition. For more details see ncsim documentation on Condition type breakpoints and also the example below.
- no: Use simple semantics when translating Condition type breakpoints to SDL. These semantics will make the breakpoint stop the run when the value of the condition makes a transition from false to true.

The default value is no.

Using the exact ncsim semantics might, in some cases, increase significantly the amount of emulation resources, and therefore take more time to load the resulting SDL program.

As an example, suppose that G is a vector with 256 bits and you define the following breakpoint:

```
stop -create -condition {#G[3] == 1}
```

If using the simple (default) semantics, the translated SDL will generate a trigger when G[3] makes a transition from 0 to 1. In such case all the other 255 bits of G are ignored.

If using the exact semantics, a trigger is generated not only when G[3] makes a transition from 0 to 1, but also when any other bit of the vector G makes any transition while G[3] is 1. For example, if G[3] is 1 and G[63] makes a transition from 1 to 0, this will also cause a trigger.

In this example, if using the exact semantics, more emulation resources are required for the purpose of detecting edges on any of the 256 bits of G. While this does not slow down emulation, it can add processing time to the import operation.

Note: The values true, on, and 1 are treated as substitutes of yes, and the values false, off, and 0 are accepted as substitutes of no.

sdl -onlyReport [yes | no]

Enables you to partially suppress the triggers defined in the Global Definition Section and Instance Definition Section (that is,, all the triggers that do not appear inside the STATE statements), while still using them to monitor for occurrence of specific conditions during the run. This ensures that the run will not be stopped because of such triggers. Triggers that are defined inside the STATE statements are not affected by this setting; so these can still be used for ending the run.

Use the yes parameter to enable the partial suppression. Use the no parameter to disable it.

When the -onlyReport option is set to yes, all triggers outside the STATE statements are restricted from stopping the design clocks or the trace capture clocks, and from ending the run. However, these triggers are still latched throughout the run.

You can use the sdl -getTriggers and sdl -report commands to check whether or not these triggers were activated in the last run. In all other commands, such as sdl -isTrigger, these triggers are not reported.

If the parameter yes or no is not specified, the -onlyReport option returns the current setting (yes or no).

Note: The values `true`, `on`, and `1` are treated as substitutes of `yes`, and the values `false`, `off`, and `0` are accepted as substitutes of `no`.

sdl -pp <output_file> [<sdl_file_name>]

Creates a file with the name as specified by `<output_file>` that shows the result of running only the preprocessor phase (that includes processing of: text macros, conditional compilation directives, and comments).

If the argument `<sdl_file_name>` is specified, then this file is used. Else, the currently selected SDL file, which was specified earlier by the `sdl -setFile <file>` command or the `sdl -load <file>` command, is used.

sdl -report [-file <file_name>]

Returns a comprehensive report that includes all the information that is provided by `-getStatus` and `-getTriggers` (and a bit more) in a user-friendly format suitable for printing.

If the option `-file` is specified, the text is written into the specified file.

sdl -restoreState <file>

Restores SDL internal state from a file that was saved earlier with the `sdl -saveState` command.

This command is typically used immediately after restoring the emulator's state, using the file that was saved at the same cycle when the emulator's state was saved.

A typical sequence to restore both the design and the SDL program is as follows:

```
restart <name>
sdl -setFile <sdl-file>
sdl -restoreState <sdl-state-file>
run -continue
```

This command restores only the SDL state from `<file>`, but not the SDL program. During the restore process, the SDL program is re-loaded from the SDL program file (as defined earlier by the `-setFile` option). The restore process is based on matching the instance and state names. Therefore, although the saved SDL program can be different from the new SDL program, all restored state names must have a match in the new SDL program and SDL instance names must match perfectly.

UXE Command Reference Manual

Run-time Commands

The `sdl -restoreState` command will honor any existing settings by switches, such as `-enable`, `-disable`, `-setFile`, and others. For example, if all SDL components were disabled prior to calling `sdl -restoreState`, the SDL program will remain disabled after the restore process.

After restoring the SDL state, use the `-continue` option in the `run` command to make SDL continue evaluation from the restored state.

The restored state takes effect only when the `run` is executed (by either a `run` or `infiniTrace -goto` trigger command), so a call to `sdl -getState` immediately after `sdl -restoreState` will still show the old state and not the one being restored.

The `sdl -restoreState` also loads the SDL file, so an `sdl -load` is not needed before the restore, and in fact might cause the `run -continue` to reject the restored SDL state.

sdl -saveState <file>

Saves SDL internal state into the specified file. The saved data includes the following for each SDL instance:

- State name
- Values of the two counters
- Status of each counter, whether it is in stopped or free running mode.

This command can be executed only after running at least one cycle in accurate trigger mode (that is, the `stopDelay` run-time parameter is set to 0).

This command is typically (but not necessarily) used immediately after saving the emulator's state.

sdl -scope [<scope>]

Specifies the SDL scope. This is the default hierarchical path that is used for resolving the design objects (net names) that appear in expressions in the SDL program. If a net name is not found in the design database, SDL searches the net under the hierarchical path that is defined by `<scope>`.

If the `<scope>` argument is not specified, the command returns the current value of the scope. This scope is independent of the debug scope that is set by the `scope` command, but follows the same scope resolution rules. Unlike the debug scope, which might change during the `run` depending on various activities, the SDL scope can be changed only by this command.

sdl -setFile [<sdl_filename>]

Specifies the name of the SDL file that should be used for the subsequent `run` command or `infiniTrace -goto` trigger command. If the argument `<sdl_filename>` is not specified, the command only returns the name of the current SDL file.

The options `-verify` `-load` `-restoreState` of the `sdl` command might also initiate compilation of the SDL program, and therefore use the value of `<sdl_filename>`.

sdl -simTime yes | no

In SA mode, selects whether to clock SDL state transitions with FCLK or by using the time stamp of the simulator.

- `yes`: SDL is clocked once on every first FCLK of the simulator's time slot.
- `no`: SDL is clocked on every FCLK in which there is any change in the design's state. This is the default mode.

The values `0`, `false`, and `off` are accepted as synonyms for the keyword `no`; while `1`, `true`, and `on` are accepted as synonyms for the keyword `yes`.

When using the `sdl -simTime yes` command, the restrictions under which the `run` command is allowed to use the `-continue` option are somewhat relaxed when running under IXCOM with uncontrolled clocks. In such case the `-continue` option is allowed also when the clocks are configured with the `clockConfig -stopMode` controlled command.

sdl -stop

Stops the run and returns control to the XE prompt. The difference between this command and a simple `stop` command is when running under IXCOM with a design that has uncontrolled clocks. If the stop mode was selected as `none` or `*none` (through the `clockConfig -stopMode none | *none` command), this command can be used to stop SDL and waveform capturing without stopping the simulator. The regular `stop` command will stop both SDL/capture and the simulator.

sdl -traceClear run | dump | never

Specifies the policy for clearing the contents of the SDL trace memory.

UXE Command Reference Manual

Run-time Commands

- **run:** The SDL trace memory is cleared at the beginning of each `run` or `infiniTrace -goto trigger` command.
- **dump:** The SDL trace memory is cleared after each successful execution of the `sdl -traceDump` command.
- **never:** The SDL trace memory is never cleared. When the buffer fills up, new samples overwrite the oldest samples.

Note that in some situations, the software might decide to clear the SDL trace memory even against the requested policy, when it determines that the SDL program changed enough to make it impossible for consistent interpretation of the data from multiple runs (for example, names of SDL states have changed).

The initial default is `run`.

sdl -traceDump [-location] [-lineOnly] [-short] [-index c | dc | t | dt] [-instance <instance_name>] [-text | -fsdb | -sst2] [-time <time_unit>] <file_name>

Dumps the contents of the SDL trace memory into the specified `<file_name>` file in either a user-readable text format or a waveform database.

By default, the SDL trace memory content is dumped in text format. The text format is more comprehensive than the waveform format. The waveform format contains only the state and counter values, whereas the text format contains all the actions along with the state and counter values.

- `-location`: Prints both the SDL file name and the line number of each action being traced (for use in text output format).
- `-lineOnly`: Similar to `-location`, but prints only the line number information without the file name (for use in text output format).
- `-short`: Does not print the command text of `EXEC` actions, and arguments of `TX` and `DISPLAY` actions.
- `-index c | dc | t | dt` specifies the way each line in the output text file should be indexed. By default, each line is indexed by the cycle number and the simulated time. If the `-index` option is used, the index information depends on the keyword that follows the `-index` option as follows:
 - `-index c` shows the cycle number

UXE Command Reference Manual

Run-time Commands

- ❑ -index dc shows the delta cycle, which is the difference in cycle number between current and previous line
- ❑ -index t shows the time
- ❑ -index dt shows the delta time, which is the difference in simulated time between current and previous line

The -index option can be used multiple times in the same command if you require more than one indexing column. For example, the following command generate both a cycle-number index and a time index, and is, therefore, equivalent to the default sdl -traceDump file.txt command:

```
sdl -traceDump -index c -index t file.txt
```

- -instance <instance_name>: Specifies the name of the SDL instance whose trace is to be dumped. If not specified, the output will contain trace dump from all the instances.
- -text: Creates the trace file in user-readable text format (default).
- -fsdb: Creates a waveform database in FSDB format. This command creates two files, a file by the name <file_name>.fsdb holds the FSDB database, and a file by the name <file_name>.rc has definitions for improved viewing, the most important one is a mapping of SDL state values to state names. This file can be read by the Debussy waveform viewer using its restore signals option.
- -sst2: Creates a waveform database in SST2 (SHM) format using the name <file_name> for the top directory of the database. The -shm option is recognized as a synonym of -sst2.
- The -location, -lineonly and -index options are relevant only when you have selected a text format.
- -time <time_unit>: Can be used only if the -text option is used with the sdl -traceDump command. It specifies the time unit to be used for formatting the time field in the output file. <time_unit> can be any string that can be assigned to the timeUnit run-time parameter (such as s, ms, us, ns, ps, fs, fclk, or dclk).

sdl -traceOn [-acqFilter no | yes] [-postTrigger no | yes] [<action_list> | ALWAYS | DEFAULT]

Specifies the cycles to be included in a SDL trace dump. This option will affect the SDL trace collected in all future run commands. It provides a way to override the default list of traced actions. The settings of this command affect all SDL instances.

The values 0, false, and off are recognized as synonyms for the keyword no, while 1, true, and on are recognized as synonyms for the keyword yes. For example:

UXE Command Reference Manual

Run-time Commands

```
sdl -traceon -acq on LOAD GOTO TRIGGER
```

- **-acqFilter no|yes** - When specified as yes, SDL trace is not taken in cycles in which probe samples are filtered out because of conditional acquisition. If specified as no, SDL tracing is done regardless of conditional acquisition. The default value for this option is no.
- **-postTrigger no|yes** - When specified as yes, SDL trace continues through the post-trigger sample phase of regular probing. If specified as no, SDL tracing stops at the trigger point. The default value for this option is no.
- **{<action_list>}**: A list of SDL actions. SDL trace is collected only when one of these actions is executed. GOTO actions are traced only if the target state is different than the current state. Possible actions are: goto, load, decrement, start, stop, trigger, display, exec, acquire, no_acquire, as well as their shortcuts.
- **ALWAYS**: Specifies that tracing should be done at every cycle.
- **DEFAULT**: Uses the default settings, which is all actions except ACQUIRE and NO_ACQUIRE should be traced.

If none of the **<action_list>**, **DEFAULT** or **ALWAYS** options are specified, the command does not modify the existing list of actions to be traced.

sdl -tx {-sst2 | -text | -ida} <filename> | -close | -show

Without the **-close** option, this command opens or re-initializes a new SST2 database into which the transaction information from the **tx** statements is written.

In ICE mode, if the **sdl -tx -sst2 <filename>** command is not specified, then by default all transaction information from the **tx** statements is written into the default SST2 waveform database, if one exists. If a default SST2 waveform database does not exist, then output from the **tx** statements in SDL will be written into a database by the name **trace_sdltx.shm**. In ICE mode, an SST2 **tx** database is written even if you are using a third-party waveform viewer.

In SA mode, if the **sdl -tx -sst2 <filename>** command is not specified and a waveform database does not exist, then output from the **tx** statements in SDL will be written into a database by the name **trace_sdltx.shm**. If the **sdl -tx -sst2 <filename>** command is not specified and a waveform database does exist, then output from the **tx** statements in SDL will be written into a database by the name **<wavedb>_sdltx.shm**, where **wavedb** is the name of the waveform database minus the extension. In SA mode, an SST2 **tx** database is written even if you are using a third-party waveform viewer.

UXE Command Reference Manual

Run-time Commands

If an SST2 database for writing transactions is already open, when the `sdl -tx -sst2 <filename>` command is issued, then the SST2 database is first closed, and a new one is opened. The `<filename>` is the name of the SST2 database. If an empty string, it only closes the currently open database.

The `sdl -tx -sst2 -close` command can also close the SST2 database. Using the `-close` option to close the database will cause all transaction recording to be routed to the default SST2 waveform database, if one exists. If you are using the waveform database, issuing this command will cause subsequent transactions to be written into the new specified database. Closing it will cause subsequent transactions to be written into the waveform database again, or the default one if the waveform database is also closed.

The `sdl -tx -sst2 -show` command returns the name of the SST2 database.

The option `-shm` can be used as a synonym for `-sst2`.

You can use the following syntax to get the transaction data in a simple text format or IDA format:

```
sdl -tx -text <filename> | -close | -show  
sdl -tx -ida <filename> | -close | -show
```

sdl -tx -illegal [error | warning | ignore | reportwarning | clearwarning]

The `error`, `warning`, and `ignore` options determine whether an illegal sequence will cause the run to stop with an error (default behavior), only generate a warning, or ignore the error, respectively. For example, an illegal sequence is to try to end a transaction before it begins.

If the `warning` option is selected, then you will be warned about the first occurrence of an illegal transaction recording request for each `tx` statement. Additional information on illegal transaction recording requests is accumulated by the debugger and can be printed with the `reportwarning` option.

- `sdl -tx -illegal reportwarning`

This command prints out additional information on illegal transaction recording requests that happened during the time covered by the report. The information includes, for each illegal request, the time stamp of the last occurrence, and number of times.

- `sdl -tx -illegal clearwarning`

This command clears the accumulated information on illegal transaction recording requests. This affects future execution of the `reportwarning` option.

sdl -tx -statistics txid | scope | stream | type

Displays the number of SDL transactions in the transaction history, and the number of all unique transaction IDs, scopes, streams, or types.

For example,

```
XE> sdl -tx -statistics txid
History contains 8 number of SDL transactions.
Number of SDL tx with id = id1: 4
Number of SDL tx with id = id2: 4
```

In this example, the total number of recorded transactions is eight. In four of the recorded transactions the transaction ID is `id1`, and in the other four transactions the transaction ID is `id2`.

sdl -verify [<filename>]

Verifies the correctness of the SDL program specified by the `sdl -setFile` command. It returns a string that can be either “Passed” or “Failed”. Note that in either case, this command will not generate an error condition.

`<filename>` is an optional argument specifying the name of an SDL file. If specified, that file is verified instead of the file that was specified by `sdl -setFile`. This enables you to verify the syntax of an SDL file without affecting the selection of the SDL file that is used for the actual run.

This command only verifies the syntax and semantics of the SDL program and the independent trigger expression. However, loading of the SDL program might still fail at a later stage due to other reasons, such as lack of sufficient emulation resources. To verify that the SDL program will be able to load, you can use the `sdl -load [<sdl_file_name>]` command, which will go through all the steps including verification and loading of the image into the emulator.

The `sdl -verify` command can be issued after the `host` command without executing a `download` command. Therefore, you can use it to verify a complex SDL program without locking any emulation resources.

sdl -when [-time] [<time_unit>]

Returns the time when the trigger happened in the last run (that is, either `run` command or `infiniTrace -goto` trigger command). If there was no trigger in the last run, the command returns an empty string.

If both the `-time` and `<time_unit>` options are not specified with the `-when` option, the returned value is the FCLK cycle number of the trigger point when using ICE flow or the time stamp when using IXCOM flow.

If `<time_unit>` is specified (with or without the `-time` option), the returned value represents the time of the trigger point using the following two components:

- a numeric value
- time unit (as specified by `<time_unit>`)

For example, `150.32 ns`.

`<time_unit>` can be any value that is accepted for the `timeUnit` run-time parameter.

If `-time` is specified without `<time_unit>`, the software assigns a time unit by itself.

Additional Information on Enabling and Disabling SDL

Each SDL instance, as well as the Global Definition Section (SDL statements located above any `instance` statement in the SDL program - for more details, refer to *SDL Program Example* in the *Controlling the Run with SDL* chapter of the *UXE User Guide*) can be enabled or disabled individually from XEL, even while the emulator is running.

The major components of the SDL program are divided into three lists as follows:

- Enabled Instances List
- Disabled Instances List
- Uncommitted Instances List

The Enabled Instances List is a list of SDL instances which were defined by the following command:

```
sdl -enable [-now] <instance_names_list>
```

The Disabled Instances List is a list of SDL instances which were defined by the following command:

```
sdl -disable [-now] <instance_names_list>
```

`<instance_names_list>` is a list of SDL instance names. For example:

```
sdl -enable INIT CHECK1 CHECK2
```

The Uncommitted Instances List contains all the other SDL components. This includes all the SDL constructs in the Global Definition Section and all the SDL instances that do not appear in either the Enabled Instances List or the Disabled Instances List.

UXE Command Reference Manual

Run-time Commands

When the SDL program is compiled and loaded into the emulator, all the SDL instances in the Enabled Instances List are enabled, while all the SDL instances in the Disabled Instances List are disabled. Note that the contents of the two lists are always mutually exclusive. All the other SDL components (in the Uncommitted Instances List) are enabled or disabled by one of the following two commands:

```
sdl -enable [-now]  
sdl -disable [-now]
```

The SDL program is compiled and loaded into the emulator when any one of the following commands is executed:

- run
- sdl -load
- infiniTrace -goto trigger
- sdl -restoreState <file>

The value returned by the `sdl` command when executed without any options is 0 if the Uncommitted Instances List is disabled, and 1 if the Uncommitted Instances List is enabled. In other words the result indicates whether `sdl -disable` or `sdl -enable` was called most recently.

Note: A value of 0 returned by the `sdl` command, does not necessarily mean that SDL is completely disabled. It only means that the SDL components that belongs to the Uncommitted Instances List are disabled.

Disabling the individual trigger or exec statements through the command:

```
sdl -disable -label <label-pattern> <label-pattern> ...
```

is independent of disabling/enabling of instances through the command:

```
sdl -disable <instance_names_list>
```

In other words, a trigger statement can be prevented from causing a trigger during run, by either disabling the individual trigger through the use of the `-label` option, or by disabling the instance in which the trigger is located (same applies for labeled exec statements).

By default, the enabling or disabling takes effect only when the SDL program is compiled and loaded into the emulator. The `-now` option can be used in combination with the `-disable` or `-enable` option for the enabling or disabling to take effect immediately, even while the emulator is running.

The `-now` option can be used only while SDL is active. For more information, refer to the [sdl -isActive](#) command.

UXE Command Reference Manual

Run-time Commands

The use of the `-now` option in commands that accept it will cause any pending enabling or disabling of SDL components to take effect immediately. This might also include any enabling or disabling from previous commands. For example, consider the following sequence of commands:

```
run -nowait
sdl -disable -label TRGX
sdl -enable -now INST1
```

In the above example, the request to disable TRGX is not executed immediately, but instead it is put in a pending state. The following command that requests to enable the `INST1` instance has the `-now` option. Therefore, it will not only enable the `INST1` instance, but it will also disable TRGX at the same time.

Examples

```
% xeDebug
XE> debug /home/george/test1
XE> host pluto
XE> download
XE> configPM -stb
XE> sdl -enable
XE> sdl -setFile trig1.tdf
XE> sdl -verify
Passed
```

The following command shows that SDL is not active because the `run` command was not executed yet.

```
XE> sdl -isActive
0
```

The following two commands request to send all messages from `DISPLAY` statements to the file `dispfile.txt` (in addition to the console), and print the current date and time of day at the top of the file:

```
XE> set fid [sdl -display -file dispfile.txt]
XE> puts $fid "[clock format [clock seconds] -format {date=%D time=%T}]"

XE> database -open trace
XE> run
25388
XE> sdl -isActive
1
XE> sdl -isTrig -label TRX
```

UXE Command Reference Manual

Run-time Commands

```
1
XE>sdl -when
25386
XE>sdl -when ns
101544 ns
XE> sdl -getStat counter2
153
XE> sdl -getTriggers
{trig1.tdf 5 {} TRX} {inc.tdf 12 INST2 LABEL4}
XE> sdl -traceOn goto load
XE> run
512
```

Create SDL trace dump in text format into a file by the name `sdlldump1.txt`. This file contains only cycles in which there was either GOTO or LOAD actions.

```
XE> sdl -traceDump sdlldump1.txt
XE> sdl -traceOn -acqOnly always
```

Show list of all SDL instances that were used in the run:

```
XE> sdl -getInstances
INST1 INST2 INST3 INST4 INST5 INST6
```

Enable instances INST1 and INST2:

```
XE> sdl -enable INST1 INST2
```

Disable SDL instance INST3

```
XE> sdl -disable INST3
XE> sdl -disable -list
INST3
```

Disable also instance INST4

```
XE> sdl -disable INST4
```

Disable everything except instances INST1 and INST2 (which were enabled earlier):

```
XE> sdl -disable
```

Show contents of Disabled Instances List and Enabled Instances List. Note that while all instances except INST1 and INST2 are disabled, the Disabled Instances List shows only instances INST3 and INST4.

```
XE> sdl -disable -list
INST3 INST4
XE> sdl -enable -list
INST1 INST2
```

UXE Command Reference Manual

Run-time Commands

Disable individual triggers with labels tr1 and tr2:

```
XE> sdl -disable -label tr1 tr2
```

Load a new SDL file and print out the names of all the nets used in any conditions:

```
XE> sdl -setFile trig2.tdf
XE> sdl -load
XE> puts "Nets used in SDL program: [sdl -getNets]"
Nets used in SDL program: clk1 reset enable
```

Request to probe all assertions and nets in the SDL program

```
XE> sdl -auto on
XE> run
700
```

Save SDL state and emulator's state, run, restore both to the same cycle, and then run again from the restored checkpoint using same SDL state. The emulator is run one cycle with `stopDelay 0` before saving the state:

```
xeset stopDelay 0
run -continue 1
xeset stopDelay 1
sdl -saveState checkpt1.sdlstate
save checkpt1
run 5.54 sec
restart checkpt1
sdl -restoreState checkpt1.sdlstate
run -continue 130 ms
```

Create FSDB waveform database (for viewing with Debussy/Verdi) with the file name `sdlldump2.fsdb`, and a signal definitions file called `sdlldump.fsdb.rc`. `sdlldump2.fsdb` shows SDL operation every cycle up to the trigger point, except cycles that were filtered out by conditional acquisition. This setting was determined earlier through the command `sdl -traceOn`.

```
XE> sdl -tracedu -fsdb sdlldump2.fsdb
```

Upload into the regular waveform database all the nets that were used in the SDL program. (as requested earlier by the command `sdl -auto on`)

```
XE> database -upload
XE> exit
```

stop

In the Logic Analyzer (LA) mode, stops tracing data but does not stop the emulator. In the STB mode and SA mode (under IXCOM), it stops tracing data and stops emulation by stopping the design clocks.

This command is typically used if the debug instrumentation does not reach the trigger point or the amount of cycles that were requested to run when running in the non-blocking mode. However, it can also be used during the STB and SA modes from within an SDL EXEC function to end a blocking run. Before using this command, execute the [run](#) command.

Note: When running in the SA mode, the `stop` command can accept additional arguments for the purpose of defining breakpoints, according to the semantics of ncsim. These should not be confused with the semantics of the `stop` command as described here.

The syntax for this command is:

```
stop [-emulation]
```

-emulation

Stops all the design clocks in run modes in which clocks are free running. This applies to LA mode, Dynamic Target mode, and IXCOM designs that have uncontrolled clocks. This is an optional parameter. This option is not supported in vvmDebug.

In the LA and Dynamic Target modes, if this parameter is omitted, only data tracing is stopped without interrupting the in-circuit emulation. This option is ignored in the STB mode.

When running under IXCOM a design that has uncontrolled clocks, this option will stop both the controlled and uncontrolled clocks. Without this option, when running with uncontrolled clocks, the effect of the `stop` command depends on the settings defined through the `clockConfig` command.

Examples

```
# Initialize debugger, use '/mydesign/design1' directory.  
debug /mydesign/design1  
  
# Download design to 'samurai' host.  
host samurai  
download  
  
# Configure for Logic Analyzer mode.  
configPM -la  
  
# run Logic Analyzer  
run
```

UXE Command Reference Manual

Run-time Commands

```
# stop Logic Analyzer  
stop
```

suspend

Suspends all the operations that are running on xeDebug and releases the emulator. A suspended design can be resumed using the [resume](#) command if the xeDebug session is still running and has not been exited. For detailed information about suspend and resume operations, refer to the *Using Suspend and Resume to Manage Emulator Resources* section in the *Working with the Palladium XP Emulators* chapter of *UXE User Guide*.

This command is available in the STB and SA modes. However, it is not supported with vvmDebug and VVM Prepare sessions.

During the suspend operation, the emulator is released and made available for other users, which means the HDDC cable is deprogrammed. If the design has a HDDC connection, you must ensure the target survival during the suspend operation.

During the suspend operation, no other XEL commands, except the `resume` command, can be executed to prevent the altering of the hardware state and ensuring synchronization between the software and hardware. If an unsupported command is executed, an error condition is generated.

A design can be suspended by using either the *Suspend* button in the xeDebug GUI, or the `suspend` command in the console in batch or interactive mode. The design can also be suspended remotely by using a Suspend file. For details, refer to the *File-based Suspend-Resume in the SA Mode* section in the *Working with the Palladium XP Emulators* chapter of *UXE User Guide*.

The emulation clock need not be running when suspending the design using the GUI or by typing the `suspend` command in the console. If it happens, the remaining clocks would get completed once the design resumes.

The syntax for this command is:

```
suspend
```

Example

```
XE> debug .
XE> host myhost
XE> download
XE> configPM -stb
XE> run -nowait          << non-blocking free run
XE> suspend              << suspend the emulation and release the emulator
XE> resume                << reload the design from the last stopping point;
                           << remaining run cycles continue
```

symbol

Creates or deletes symbols and returns symbols. Each symbol represents one or more signals.

A symbol is a convenient way to represent a bus composed of an arbitrary collection of nets. When inspecting the integer value that the symbol represents, the left-most signal name in the list is the most significant bit (MSB).

Without parameters, the `symbol` command lists the names of all currently-defined symbols. This command is available in all debug modes.

Note: In the SA mode, the `symbol` command works only on the DUV signals on the hardware-side and not the testbench signals.

If either `value`, `force`, `deposit`, or `release` command needs to be used on a large set of signals, the access time can be significantly improved by grouping the signals using the `symbol` command and thereafter using the symbol instead of each signal separately. When grouped in such a way, more efficient mechanisms can be used to access the hardware.

The syntax for this command is:

```
symbol [-add <symbol> <sig> ...]  
symbol [-get <symbol>]  
symbol [-rm <symbol>...]
```

-add <symbol> <sig> ...

Creates a symbol with the specified `<symbol>` name, representing one or more signals as specified in the `<sig>...` signal list. The first signal name in the list represents the MSB. Elements in the list might be either single bit signals or buses, but cannot be other symbols. For signals that are defined as buses (vectors), use either the name of the signal (in which case all of its bits will be included), a part-select, such as `signame[7:0]`, or a bit-select, such as `signame[5]`.

-get <symbol>

Lists all the signals associated with the specified symbol.

-rm <symbol>...

Deletes the specified symbols.

Examples

```
# Initialize the debugger and use the
# '/mydesign/design1' design database directory.
debug /mydesign/design1

# Use the 'samurai' emulation host.
host samurai

# Download the design on 'samurai'.
download

# Use STB mode when debugging the design.
configPM -stb

# Create a new symbol called s1, representing the
# 'A' and 'B' signals. A[3] is the MSB in the symbol
symbol -add s1 A[3:0] B

# List all currently-defined symbols for signals.
symbol

# List all signals associated with the s1 symbol.
symbol -get s1

# use the symbol s1 to examine the values of A and B
value s1

# Delete the s1 symbol.
symbol -rm s1
```

tca

The **tca** Tcl command can be used to analyze toggle count data.

The syntax for this command is:

```
tca -setFile <file_name>
tca -getAllInstanceNames [<fileID> ...] [-maxDisplay <number>]
tca -getValues <[fileID] inst> [<[fileID] inst> ...] -start <start_time> [-end
<end_time>] [-type toggle | high] [-count ave | max]
tca -getTimeRange [<fileID> ...]
tca -getTopNInst <number> [-start <start_time>] [-end <end_time>] [-type toggle |
high] [-count ave | max] [<fileID> ...]
tca -getTopNPeak <number> [-sensitivity <N>] [-start <start_time>] [-end
<end_time>] [-type toggle | high] [<[fileID] inst>...]
tca -getTopNSlope <top_N_number> [-start <start_time>] [-end <end_time>] [-type
toggle | high] [-step m] [<[fileID] inst>...]
tca -export <waveform_name> <[fileID] inst> [<[fileID] inst> ...] [-format <text |
sst2 | fsdb>] [-start <time>] [-end <time>] [-filter {<filtername>
<filter_arguments>}] <list_of_instance_names>
tca -merge <source_ppfdata_list> -to <one_destination_ppfdata>
tca -file -open <filename>
tca -file -close <fileID>
tca -file -filename <fileID>
tca -file -fileid <filename>
tca -file -list
tca -help
```

-setFile <file_name>

Opens the * .ppfdata file. The * .ppfdata file must be opened before using any other tca command.

This option sets the specified * .ppfdata file as the current working file.

-getAllInstanceNames [<fileID> ...] [-maxDisplay <number>]

Returns a list of all the instances in the specified file(s).

Note: The specified file(s) must have been opened using either the `tca -setFile` command or the `tca -file -open` command. If `<fileID>` option is not specified, all instances of all the opened files are returned.

The instance names are returned in the following format:

```
{fileID inst} {fileID inst}...
```

If only one file has been opened, the results are returned as following (for backward compatibility):

```
inst inst inst ...
```

The `-maxdisplay <number>` option specifies the number of maximum instances to be displayed in the toggle count database.

-getValues <[fileID] inst> [<[fileID] inst> ...] -start <start_time> [-end <end_time>] [-type toggle | high] [-count ave | max]

Returns the values for the specified list of instances.

If the `<fileID>` is specified, it must be grouped with the instance name into a list of items. A list of numbers is returned in the same order as the instance names passed as shown below:

```
10 123 235 ...
```

Note: `TCL_ERROR` is returned if any instance name specified does not exist in the data file.

The `-start` and `-end` options specify a time window in which the value is calculated. The `-start` parameter must be specified, but the `-end` parameter is optional. If `-end` is not specified, its value is assumed to be the same value as the start time.

You can also specify a pre-defined bookmark name instead of the time range.

The `-type` option specifies which data type to list, either toggle count or high count. If the type of data is not specified, the default data type is toggle count.

The `-count` option specifies whether or not the value should be calculated as an average value or a maximum value. The default is given as an average value.

-getTimeRange [<fileID> ...]

Returns the time range combined from the specified file(s). If no file is specified, the command returns the combined time range from all of the opened files as illustrated below:

```
{1 fclk} {1100 fclk}
```

UXE Command Reference Manual

Run-time Commands

-getTopNInst <number> [-start <start_time>] [-end <end_time>] [-type toggle | high] [-count ave | max] [<fileID> ...]

Returns the top set of instances based on the value of instance data. *<number>* specifies the number of instances to list. By default, all opened files are searched to return the data in the following format:

```
{fileID inst value} {fileID inst value} ...
```

If only one file has been opened, the output is returned in the following format:

```
{inst value} {inst value} ...
```

The *-start* and *-end* options are optional parameters that specify a time window for calculating the values. If no time is specified, the entire time range is used. You can also specify a pre-defined bookmark name instead of the time range.

The *-type* option specifies which data type to list, either toggle count or high count. If the type of data is not specified, the default data type is toggle count.

The *-count* option specifies whether or not the value should be calculated as an average value or a maximum value. The default is given as an average value.

If the *<fileID>* option is specified, the `tca -getTopNInst` command returns the top set of instances in the specified file.

The top N instances are picked based on the value specified in the above-mentioned options.

Using the `tca -getTopNInst` command without the optional parameters specified within the [] brackets in the syntax above, returns a list of pairs of instance names and value, which are sorted in decreasing order of the value.

-getTopNPeak <number> [-sensitivity <N>] [-start <start_time>] [-end <end_time>] [-type toggle | high] [<[fileID] inst>...]

Returns the top set of peaks among the specified instances. By default, all opened files are searched to return the data in the following format:

```
{fileID inst value {time time_unit}} ...
```

If only one file has been opened, the output is returned in the following format:

```
{inst value {time time_unit}} ...
```

<number> specifies how many peaks to display. It is possible for one instance to have more than one peak.

UXE Command Reference Manual

Run-time Commands

The `-sensitivity <N>` option specifies the sensitivity of the peak detection. `<N>` needs to be an odd number. Once the peak detection algorithm decides that one sample is peak, it checks $(N-1)/2$ samples before and after the peak sample. With a larger `N` value the algorithm checks more samples. Therefore, it smooths out more noises and finds less peaks. The default value of `N` is 33.

The `-start` and `-end` options are optional parameters that specify a time window for calculating the values. If no time is specified, the entire time range is used. You can also specify a pre-defined bookmark name instead of the time range.

The `-type` option specifies which data type to list, either toggle count or high count. If the type of data is not specified, the default data type is toggle count.

If the `<[fileID] inst>` option is specified, the `tca -getTopNPeak` command returns the top set of peaks for the specific instance.

Using the `tca -getTopNPeak` command without the optional parameters specified within the `[]` brackets in the syntax above, returns a list of triplets composed of instance names, time point of peak and peak value, which are sorted in decreasing order of peak value.

`-getTopNSlope <top_N_number> [-start <start_time>] [-end <end_time>] [-type toggle | high] [-step m] [<[fileID] inst>...]`

Returns the top `N` slopes of toggle counts among the specified instances. A slope describes how fast a signal or port toggles during a simulation run. The maximum `N` toggle counts are calculated based on the specified time range and step count.

The `<top_N_number>` option specifies the number of top `N` slopes.

The `-start` and `-end` options are optional parameters that specify a time window for calculating the values. If no time is specified, the entire time range is used.

The `-type` option specifies which data type to list, either toggle count or high count. If the type of data is not specified, the default data type is toggle count.

The `-step` option specifies the step count. The default step count is 1.

If the `<[fileID] inst>` option is specified, the `tca -getTopNSlope` command returns the top set of slopes for the specific instances in the `.ppfdata` file.

For example, consider the following toggle count data:

UXE Command Reference Manual

Run-time Commands

Time	0ns	1ns	2ns	3ns	4ns	5ns	6ns	7ns	8ns	9ns	10ns
Toggle	0	30	45	75	150	0	40	30	20	25	40

If the step count is 1, then the slope is determined as follows:

```
S1 = abs(30-0) = 30
S2 = abs(45-30) = 15
S3 = abs(75-45) = 30
S4 = abs(150-75) = 75
S5 = abs(0-150) = 150
S6 = abs(40-0) = 40
S7 = abs(30-40) = 10
S8 = abs(20-30) = 10
S9 = abs(25-20) = 5
S10 = abs(40-25) = 15
```

In this example, the maximum three slopes are 150 at 4ns, 75 at 3ns, and 40 at 6ns.

If the step count is 2, the slope is determined as follows:

```
S1 = abs((0+30)/2 - (15+30)/2)
...

```

```
-export <waveform_name> <[fileID] inst> [<[fileID] inst> ...]
[ -format <text | sst2 | fsdb> ] [-start <time>] [-end <time>]
[ -filter {<filtername> <filter_arguments>} ] <list_of_instance_names>
```

Exports the toggle count and high count information to the specified waveform database. The default format of the waveform database is SST2 and _dpa is automatically added to the name of the exported file. The information can also be exported in FSDB format. To export the information to a text file instead, use the -format text option. Consequently, the information is saved to the specified text file in the following format:

- * Comment line starts with '#'
- * The first few lines are comment lines that describe basic informations such as:
 - + name of .ppfdata file from which the data are exported
 - + date and time
 - + time range specified in the command line for export
 - + Any filter that is applied together its arguments
- * The basic information lines are followed by exported data, one instance after another.
- * There is a comment line to show the instance name before the data for that instance. For example:
Instance: top.A
- * The data are one cycle per line. There are following three numbers on each line:

UXE Command Reference Manual

Run-time Commands

```
+ cycle number  
+ toggle count  
+ high count
```

The instance names to be exported must be specified. If only one file is opened, the `fileID` can be omitted. The format of instance list can be as following:

```
inst inst ...
```

If multiple files are opened, the `fileID` must be specified. The format of instance list can be as following:

```
<fileID inst> <fileID inst> ...
```

The `-start` and `-end` options are optional parameters that specify a time window for analysis. If no value is specified for these options, the entire time range is used. You can also specify a pre-defined bookmark name instead of the time range.

The `-filter` option specifies the filter to be applied. It is recommended that you only export the instances of interest because the export operation is time consuming. The `<filtername> <filter_arguments>` arguments can have either of the following values:

■ `IntervalAverage <interval_size>`

Specify the `<interval_size>` of the time window by using the `IntervalAverage` filter. The `IntervalAverage` filter calculates and exports to the `SHM` file an average value for toggle count and high count for cycles in each interval.

Using this filter reduces the number of data samples saved in the `SHM` file and helps SimVision to run faster by splitting the whole time window into multiple intervals.

■ `RunningAverage <window_size>`

Specify the `<window_size>` of the average window by using the `RunningAverage` filter. The `RunningAverage` filter calculates an average number for each cycle from a window surrounding it and makes the waveform look smoother.

Important

The `<filtername> <filter_arguments>` arguments should be enclosed either within curly braces '{ }' or double quotes. For example, use either `-filter "IntervalAverage 10"` or `-filter {IntervalAverage 10}`

-merge <source_ppfdata_list> -to <one_destination_ppfdata>

The **-merge** option specifies the consecutive source *.ppfdata files to be merged into one destination *.ppfdata file.

Note: All the *.ppfdata files must be different from each other and all the source *.ppfdata files must come from the same design.

For example,

```
tca -merge wtc1.ppfdata wtc2.ppfdata wtc3.ppfdata -to wtc.ppfdata
```

where, wtc1.ppfdata, wtc2.ppfdata, and wtc3.ppfdata are the consecutive data files. And, wtc.ppfdata is the destination data file.

-file -open <filename>

Opens a data file with the specified name. When the *<filename>* is not specified, the file ID is returned.

-file -close <fileID>

Closes the data file with the specified file ID.

-file -filename <fileID>

Returns the file name of the specified file ID.

-file -fileid <filename>

Returns the file ID of the specified file.

-file -list

List the pairs of file ID and file name in the following format:

```
{fileID filename}
```

-help

Returns the command syntax.

userData

Note: The `userData` command discussed below is used at the run time. The `userData` compile-time command is discussed in [Chapter 8, “User Data Commands.”](#)

Returns the user data as a Tcl string. User data can be dumped any time after the data has been defined. This is an optional command.

The dumped user data can also be saved to a file by the using the `redirect` command.

The syntax for this command is:

```
userData -get [-max <n>] [<type> [option]]  
userData -dump <file>
```

-get [-max <n>] [<type> [option]]

Gets the user data for the specified type from the design and prints it to stdout.

The `-max <n>` option specifies the maximum limit of user data to get. The default value of `<n>` is to return up to maximum 20 lines for every data type. When `<n>` is specified as 0 (that is, zero), all data is returned.

The `<type>` option specifies the type of user data. It is the name of the XEL command used to establish the user data type. When `[option]` is also specified, the value for only the specified option is returned.

-dump <file>

Dumps user data from the design to the specified file. This command does not have any limit on the user data to be dumped.

Examples

- To get the entire user data and print it to stdout, specify the following:

```
userData -get
```

User data as illustrated below is returned:

```
clockPathOptions:  
  {activeEdge positive}  
  {ignoreEvent 0}  
  {ignoreOutput 0}
```

UXE Command Reference Manual

Run-time Commands

```
{technology none}
emulatorConfiguration:
    {{HOST ostrich} {BOARDS 0.0}}
compilerOption:
    {enableMultiSampledIO 0}
...
```

- To get the user data specific to the `compilerOption` XEL command, specify the following:

```
userData -get compilerOption
```

User data as illustrated below is returned:

```
{enableMultiSampledIO 0} {infiniTrace on} {mode ice} {sdlInstances 1}
{visionMode DYNP}
```

- To get only the mode for which the design was compiled, specify the following:

```
userData -get compilerOption mode
```

If the design was compiled for ICE mode, the following data is returned:

```
ice
```

- To save the result to a file, execute the following command:

```
redirect userData -get > file1
```

value

Returns the current state of the specified objects.

If the `value` command is operated on a vector while clocks are running, time coherency between the values of the signals in the vector is not guaranteed. In addition, the values of single bit nets might also be incorrect when read while clocks are running, unless the nets being read are primary inputs, state device outputs, or keepnets.

The `value` command is available in all operation modes. This command can accept all options supported by IES. For detailed description of the options supported by IES, refer to *Incisive Simulator Tcl Command Reference* of the INCISIVE documentation set.

In `vvmDebug`, the `value` command returns the current state of the specified objects only after the `run` command finishes execution.

Note: In the SA mode, after the DUV is swapped to the hardware, only `-short` and `-verbose` options of the `value` command are supported. For testbench and DUV running on the simulator (that is, software), all options of `value` command that are supported by IES can be used. In this case, the `-short` option, which is not an IES-specific option, can also be used.

In `xeDebug`, the `value` command supports both ascending and descending order for a part-select range, that is, you can also specify the range opposite to the declared order. However, in IES, the part-select range has to match the declared order. For example,

```
module test(input clk, output reg c[1:0]); // ....  
endmodule
```

In IES, if you specify the range in the `value` command in the opposite order of the declared range, the test fails at run time with the following error message:

```
ncsim> value -v %d c[0:1]  
ncsim: *E,PPRTDR: Direction of part-select does not match definition: [0:1]
```

The syntax for this command is:

```
value [-file <filename>] [-short] [-verbose] [<format>] <object_names>
```

-file <filename>

Gets the value of a signal to a file. The file contains the value of one single bit net per line. The scope resolution or RTL-to-gate translation is not done. This option is supported only for DUV that has been swapped to the hardware.

-short

Returns the value of a signal or vector without adding the default formatting or prefixes, such as '`b`', '`o`', '`d`', and '`h`'. This option is supported only for DUV that has been swapped to the hardware.

For example, specifying the `value -short` command, returns the signal value as `11` instead of '`b11`'.

-verbose

Turns on the mechanism to return detailed messages. This option is supported only for DUV that has been swapped to the hardware.

<format>

Specifies the format of the signal, symbol, or vector. The valid format values are:

- `%b` indicates binary format
- `%o` indicates octal format
- `%d` indicates decimal format
- `%h` or `%x` indicates hexadecimal format

Note: The format set using the `value` command overrides the format set using the `signalRadix` run-time parameter.

<object_names>

An object can be a signal, a vector, a bit in a vector, a range of bits in a vector, a memory address, a memory range, or a symbol.

For example,

```
test.clk  
test.in  
test.in[0]  
test.in[7:0]  
test.dut.mem[0]  
test.dut.mem[0:9]  
sym1
```

UXE Command Reference Manual

Run-time Commands

The `value` command can accept regular expressions that can be applied only to nets (not symbols).

For example,

```
value dut.data*
```

Note:

- ❑ In the SA mode, symbols are supported only after the DUV has been swapped to the hardware.
- ❑ The radix of the returned value is determined by the `signalRadix` run-time parameter. It is initialized by default to the value `bin` (binary), but you can modify it with the `xeset` command to other radix values (such as `hex`, `dec`, or `oct`).

vector

Specifies the name of the stimulus file to use in Vector Debug mode, and compiles it to VBF file (the one internally used by Vector Debug mode) if needed. This command sets the vector parameter. This command is available in Vector Debug mode only. Before using this command (without the `-compileOnly` option), execute the [configPM](#) command. The `vector` command is not supported with `vvmDebug`.

The stimulus file (`<stim_file>` or `<vbf_stim_file>`) , which is specified in the `vector` command, contains input stimulus and expected results that are used during Vector Debug mode. The file might be either a text file (in one of the input formats that the PTRAN vector-translation program supports), or directly in VBF format (which is the binary format used by the software).

If the specified file is not already in VBF format, the `vector` command internally invokes the PTRAN program as a subprocess to compile the input file into VBF format. The compilation step is skipped if a previous copy of the translated VBF file still exists in the current directory and is up to date. PTRAN can also be called directly from the Unix command line. The `vector` command can also specify the name of a PTRAN command file (`cmd_file`) that contains the instructions for PTRAN to compile the vector file.

The syntax for this command is either one of the following:

```
vector <stim_file> [<cmd_file>] [<aux_file>] |
vector <vbf_stim_file> |
vector -compileOnly <file.cbc>
```

<stim_file>

Specifies the name of the file containing the vectors to be compiled for use in emulating the design. This is the name of the source file in any of the formats accepted by PTRAN that contains the stimulus vectors.

<cmd_file>

Specifies the name of a file (called PTRAN command file) that contains instructions about how to perform the translation. The installation directory has command files for several vector formats in the `<install_dir>/etc` directory. The `<install_dir>` is the directory where the software is installed. The names of these command files have an extension of `vtran`.

To view a complete list of all available PTRAN command files in that directory for translating from various formats into VBF format, type the following on the Unix command line:

```
ls <install_dir>/etc/*2vbf.vtran
```

If the PTRAN command file is taken from the default location under the installation directory (that is, *<install_dir>/etc*), only the base file name is required and the path can be omitted. For example, instead of *<install_dir>/etc/cbc2vbf.vtran*, just use *cbc2vbf.vtran*.

If *<cmd_file>* is missing from the command, the command assumes as default a CBC input format, which is equivalent to specifying *cbc2vbf.vtran*.

<aux_file>

Some input formats might also require a second file *<aux_file>* that contains specific signal information.

<vbf_stim_file>

Specifies the name of a stimulus file that is already in VBF format. When using this option, the second argument to the `vector` command is a dash, indicating that no translation is needed.

-compileOnly <file.cbc>

Specifies the name of a stimulus file *<file.cbc>* to be compiled before the host command. You must call the `vector` command again after the `host` command to specify the compiled vector (.in) file to be used.

Examples

```
# Initialize the debugger and use the
# `/mydesign/design1' design database directory.
debug /mydesign/design1

# Compile a stimulus file file.cbc before the host command.
vector -compileOnly file.cbc

# Download design to `samurai' emulation host..
host samurai
download
configPM -vd

# use the previously compiled vector *.in file.
vector file.cbc.in -
run
```

vvm

The `vvm` command is used to create Virtual Verification Machine (VVM) sessions online, which can be used later for debugging in offline mode. The `vvm` command is also used in the `vvmDebug` tool, which is used for offline debugging.

The VVM sessions can be used only in STB mode and SA mode with the exception that VVM is not supported in VD mode, LA mode, Dynamic Target mode, and with designs that have uncontrolled clocks (under IXCOM).

The syntax for this command is:

```
vvm -prepare [-overwrite] <session_name> [-memoryReplay [<memory_instances>]]  
vvm -open <session_name>  
vvm -close  
vvm -list [<session_name> | -allsession] [-info]  
vvm [-intervals]
```

-prepare

Creates a new VVM session with the specified name. When the design is running, Palladium captures and uploads the DCC data automatically into the VVM session.

Note: This command is used only in the online sessions.

[-overwrite]

Overwrites any existing VVM session. If the `-overwrite` option is not specified and a session with the specified name already exists, an error message is issued.

<session_name>

Specifies the name of the session directory that contains data for the VVM session.

[-memoryReplay [<memory_instances>]]

This option enables memory replay in the offline session, which enables you to dump memory data from any time point in the captured session. If a list of memory names are specified, those memories can be replayed faster. By default, the VVM sessions are created without

UXE Command Reference Manual

Run-time Commands

-memoryReplay. When memory replay is enabled, memory snapshots are saved at a predefined intervals.

Example

The following command will create a VVM session directory `ghi.vvm`. The VVM session has memory replay enabled. The memories `m0.m_mema` and `m0.m_memb` are enabled for fast memory replay. The other memories in the design might be replayed, however, this might take longer time as compared to the specified memories.

```
vvm -prepare ghi -memoryReplay m0.m_mema m0.m_memb
```

-open <session_name>

Opens the specified VVM session for debugging.

Note: This command is used only in the offline sessions.

-close

Closes the VVM sessions. If this command is used in an online session, it closes the current VVM session and stops capturing. If used in an offline session, it closes the current VVM session that is being debugged.

-list [<session_name> | -allsession] [-info]

If this command is executed without any option, it lists all the VVM sessions existing in the current session directory.

You can also view detailed information for a specific session by specifying the session name with `-list` command such as `vvm -list <session_name>`.

If you do not specify any session name and specify the `-allsession` option instead, all the VVM sessions existing in all the session directories are displayed. The `-info` option shows detailed information for each session.

Examples

■ `vvm -list -allsession -info`

The above command displays a list of all available VVM sessions and the intervals when the data was captured for each session.

UXE Command Reference Manual

Run-time Commands

■ `vvm -list vvmSession12`

The above command shows detailed information for the session `vvmSession12`.

[`-intervals`]

List the time interval for the current VVM session. Currently, VVM doesn't support multiple intervals within a single session. However, a single online run can create multiple sessions, each capturing a different time range.

waitWhileBusy

Blocks out further command execution while the emulator is busy. The command returns control to the XE prompt when the emulator becomes not busy. In STB and SA modes, the emulator is regarded busy as long as emulation clocks are running. In LA and Dynamic Target modes, the emulator is regarded busy as long as capture clocks are running (that is, as long as the trace buffer is collecting data).

Typically, you execute this command after using the `run` command in non-blocking mode, but before executing `database -upload`, to verify that the `run` command has finished.

This command is available in the LA, STB, Dynamic Target, and SA modes. It is not available in the Vector Debug mode.

You can force the control to return to the XE prompt and stop the emulation clocks (in STB and SA modes) or capture clocks (in LA and Dynamic Target modes) by pressing the **CTRL+C** key combination on the keyboard.

In vvmDebug, this command is used to wait until the trigger happens.

Executing this command immediately after executing `run -nowait` is equivalent to executing a `run -wait`.

The syntax for this command is:

```
waitWhileBusy [<timeout>]
```

<timeout>

Specifies the number of seconds to wait before timing out. This is an optional parameter. If the command is used without this parameter, the wait will be for an indefinite time period.

Examples

```
# Initialize the debugger and use the
# '/mydesign/design1' design database directory.
debug /mydesign/design1

# Download design to the 'samurai' emulation host.
host samurai
download

# Set the system for LA mode.
configPM -la

# set the system to trigger according to an SDL program
sdl -setFile mySDLprogram.tdf
sdl -enable
```

UXE Command Reference Manual

Run-time Commands

```
# Start tracing data; wait until trigger happens but no more than 10 seconds.  
# The catch statement prevents the script from aborting in case of time out  
run  
catch {waitForBusy 10}  
  
# Upload traced data to workstation.  
sigTrace -upload  
  
# run again, but this time wait until trigger happens without any time out  
run  
waitForBusy
```

xeset

Sets and retrieves value of run-time parameters.

Run-time parameters are of two types, Read-Write and Read-Only. The values of Read-Write run-time parameters can be modified using the `xeset` command. However, values of Read-Only run-time parameters cannot be re-assigned by the `xeset` command. Such parameters either have pre-assigned values (for example, the `version` parameter) or their value is assigned by another command (for example, the `host` parameter).

Following syntax sets the value of a run-time parameter:

```
xeset <parameter_name> [<value>]
```

The string `<parameter_name>` is case-insensitive, which means that either lower or upper case can be used for any of the characters in the string.

It is not necessary to type the entire parameter name. Typing the first few characters that distinguish it from other parameters is enough.

For example, to set the value of the `triggerPos` parameter as 40, specify:

```
xeset triggerPos 40
```

The following command will produce the same result:

```
xeset Trig 40
```

If run-time parameter `<value>` is expected to be an integer, unless otherwise documented, the value string can be written as either decimal, hexadecimal, or octal using the same notation as in the `expr` Tcl command. If the left-most digit is one of 1-9, the number is read as decimal. If the left-most two digits are 0x, the number is read as hexadecimal. Otherwise, if the left-most digit is 0, the number is interpreted as octal. For example, the number 15 can be written as either 15 (decimal), 0xff (hexadecimal), and 0377 (octal).

If `<value>` is not specified, the `xeset` command returns the existing value of the parameter. Therefore, following is the default syntax for reading the value of the specified run-time parameter:

```
xeset <parameter_name>
```

For example, the following command prints the value of the `triggerPos` parameter:

```
puts "Trigger position = [xeset trig]"
```

The above command prints the following result:

```
Trigger position = 40
```

UXE Command Reference Manual

Run-time Commands

The syntax for this command is:

```
xeset appendTrace [0|1]
xeset autoResumeRun [0|1]
xeset autoUpload [0|1|onTrigger]
xeset autoExit [<value>]
xeset bpValue [<domains or boards>]
xeset database
xeset dbHwType
xeset dbPath
xeset depositNCSimMode [1|0]
xeset design
xeset designName [<name>]
xeset endOfVector
xeset fsdbFileSizeLimit <number>
xeset fvMaxThread [<number>]
xeset fvPreprocess [0|1]
xeset fvSafeMode [0 | 1]
xeset gui
xeset -gui autoWave [0|1]
xeset -gui dbAutoWave [0|1]
xeset -gui dbAutoUpload [0|1]
xeset -gui dbMaxListLen [<number>]
xeset -gui dbShowGenerated [0|1]
xeset -gui dbShowPrimitiveTerm [0|1]
xeset -gui overwriteMemFile [0|1]
xeset -gui svSaveWaveScript [0|1]
xeset -gui svShowGenerated [0|1]
xeset -gui svUploadOnProbe [0|1]
xeset -gui userNet1 [<net name>]
xeset -gui userNet2 [<net name>]
xeset -gui pollingRate [<value>]
xeset host
```

UXE Command Reference Manual

Run-time Commands

```
xeset idleExit [<value>]
xeset ignoreProbeFailureInForce [value]
xeset keepHostAlive [0|1|2|3|4]
xeset keepHostAliveTimeout [<value>]
xeset linkFiles [{[<pattern> ...] [-ice <pattern> ...] [-sa <pattern> ...]}]
xeset maxMismatches [<number>]
xeset msgMaxLen [<chars_per_line>]
xeset mtHost {[<host1> <host2> <host3>...] | {LSF <number_of_hosts>
<command_line>}}
xeset postTriggerSamples [<value>]
xeset probeCompiled [0 | 1]
xeset probePrimary [0|1]
xeset progress [off | gui | display | stdout | file | file+ | gui, display | gui,
stdout | gui, file | gui, file+ | stdout, file | stdout, file+ | gui, stdout,
file | gui, stdout, file+]
xeset reserveKey [<key_value>]
xeset reserveTimeout <timeoutValue>
xeset retryDownload [<N>]
xeset retryInterval [<T>]
xeset sdlAutoNcBreakpoints [0|1]
xeset sdlIncludePath {[<dirname> ...]}
xeset signalRadix [<radix>]
xeset stopDelay [0|1]
xeset suspendEnable [0|1]
xeset timeRangeCheck [0|1]
xeset timeUnit [<value>]
xeset traceMemMax
xeset traceMemSize [<size>]
xeset triggerPos [<percent>]
xeset uploadSdlTrace [0 | 1]
xeset vdCreateWaves [0 | 1]
xeset vdLegacyCompare [0 | 1]
xeset vector
```

`xeset version`

appendTrace

Specifies whether or not to append the new trace data to the existing trace data.

When this run-time parameter is specified without any value, it returns the current setting.

Note: The `-appendTrace` option is not supported with `vvmDebug`.

The syntax for setting the value of this parameter is:

`xeset appendTrace [0 | 1]`

0

Indicates that the trace data should not be appended to the existing trace data.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

1

Indicates that the trace data should be appended to the existing trace data.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

By default, the `appendTrace` run-time parameter is set to 1.

autoResumeRun

Specifies whether or not a design run that automatically uploads the trace data after a trigger happens should be resumed as a trigger-free run.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

`xeset autoResumeRun [0 | 1]`

0

Indicates that the run should not be resumed automatically after the trigger happens.

Note: The values `off`, `no`, and `false` are accepted as synonyms for `0`.

By default, the `autoResumeRun` run-time parameter is set to `0` in both ICE and SA modes.

1

Indicates that the run should be resumed automatically after the trigger happens.

Note:

- The `autoResumeRun` run-time parameter can be set to `1` only when you run the design on the emulator in the STB or SA mode.
- The values `on`, `yes`, and `true` are accepted as synonyms for `1`.

autoUpload

Specifies whether or not to enable automatic upload of the trace data at the end of a design run.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset autoUpload [0 | 1 | onTrigger]
```

0

Indicates that the trace data should not be uploaded automatically at the end of the design run.

Note: The values `off`, `no`, and `false` are accepted as synonyms for `0`.

By default, the `autoUpload` run-time parameter is set to `0`.

1

Indicates that the trace data should be uploaded automatically at the end of the design run.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for `1`.

onTrigger

Indicates that the trace data should be uploaded automatically after the trigger happens during a design run.

autoExit

Controls overall time to be spent in the xeDebug tool. The time starts when the `xeset` command is issued.

The syntax for setting the value of this parameter is:

```
xeset autoExit [<value>]
```

where `<value>` is a non-negative integer number that specifies the time in minutes after which xeDebug should automatically exit.

By default, the value is set to 0, which means no automatic exit.

bpValue

Sets or overwrites the content of the `<design>.bp` file to specify the target board or domain.

Note: Alternatively, use the `XE_BP_VALUE <sub_board>` environment variable on the UNIX command prompt before starting the debug process.

The syntax for setting the value of this parameter is:

```
xeset bpValue <sub_board>
```



The `<sub_board>` string can be enclosed either within double quotes " " or curly braces { }.

When enclosed within double quotes, the `<sub_board>` must be specified in the following string format with no space between the characters:

```
"<boardNum>[.<domainNum>] + ... <boardNum>[.<domainNum>]"
```

When enclosed within curly braces, the `<sub_board>` must be specified in the following string format with space between the each `<boardNum>.<domainNum>` set:

```
{<boardNum>.<domainNum> <boardNum>.<domainNum> ... }
```

For information about `<boardNum>` and `<domainNum>`, refer to the [emulatorConfiguration -add {host <hostname>} {boards <sub_board>}](#) command description in [Chapter 8, “User Data Commands.”](#)

Examples

```
XE> xeset bpValue "0+1.0+1.1"
```

OR

```
XE> xeset bpValue {0.0 0.1 1.1}
```

Sets the *<sub_board>* information in the .bp file.

database

Stores the name of the current design database. The name of the design database is set during the design compilation. This is a read-only parameter. Its value is set by the debug command (in a debugging session).

Examples

```
xeset database  
ccl           << Returned data
```

Returns the name of the current design database, ccl.

dbHwType

Returns the Palladium XP hardware type for which the current database has been compiled. This is a read-only parameter that returns either of the following two values:

- Palladium XP GXL
- Palladium XP XL

Examples

```
xeset dbHwType  
Palladium XP GXL
```

Returns the Palladium XP hardware type as Palladium XP GXL.

dbPath

Stores the directory path to the emulation design database. This is a read-only parameter and its value is set by the debug command.

Examples

```
debug /mydir/jpeg.test  
xeset dbPath  
/mydir/jpeg.test      << Returned data
```

Returns the name of the directory containing the current design, `/mydir/jpeg.test`, as set in the most recent `debug` command.

depositNCsimMode

Specifies whether or not the `deposit` command should behave the same in the ICE mode as in the SA mode.



This parameter only affects the `deposit` command in ICE mode. It has no effect on the SA mode.

The syntax for setting the value of this parameter is:

```
xeset depositNCsimMode [1 | 0]
```

1

Indicates that the `deposit` command behaves the same in the ICE mode as in the SA mode, that is, it will not deposit value into a forced signal.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

0

Indicates that the `deposit` command behaves different in the ICE mode from the SA mode, that is, it will deposit value into a signal whether that signal is forced or not.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, this parameter is set to 0 in the ICE mode. In the SA mode, this parameter has a value of 1, and the value cannot be modified.

Examples

```
xeset depositNCsimMode true
```

Sets the deposit work mode to NCsim consistent.

```
xeset depositNCsimMode  
1 << Returned data
```

Returns the current value of the `depositNCsimMode` run-time parameter.

design

Stores the name of the current design cell. The name of the design cell is set during design compilation. This is a read-only parameter. Its value is set by the [debug](#) command.

Examples

```
xeset design  
JPEGP << Returned data
```

Returns the name of the current design cell, JPEGP.

designName [<name>]

Enables you to change the design name before using the `download` command.

For example,

```
debug .  
xeset designName "mydesign_v1.1"  
host .  
download
```

The `test_server` will display the user-defined design name in its output. You can use the command in `.xerc` file to change design name as follows:

```
% cat .xerc  
if {[info exists env(LOCAL DESIGN NAME)]} {  
    xeset designName $env(LOCAL DESIGN NAME)  
}  
  
% setenv LOCAL DESIGN NAME yyyyyy  
% xeDebug  
XE> xeset designName  
YYYYYY  
XE>
```

endOfVector

Stores the value `true` or `false` depending on whether the last `run` command emulated all the vectors from the stimulus file in Vector Debug mode or not. If any vectors still need to run, the value of this parameter is `false`.

This is a read-only parameter and its value is set by the `run` command.

Examples

```
debug /mydir/jpeg.test
host zeus
download
configPM -vd
vector JPEGP.clk
run
xeset endOfV
true      << Returned data
```

Shows that all the vectors in the `JPEGP.clk` vector file were emulated.

fsdbFileSizeLimit

Specifies the file size limit for the database created in FSDB format. Setting this run-time parameter helps to automatically create a new FSDB database when the file size of the existing database reaches the specified limit. This saves the time and memory required to open a new FSDB database.

Each new FSDB database is assigned a name of the format
`<database_name>_<number>`, where, the `<number>` starts from 2.

For example, if the FSDB database name is `big` and it is split into six files, the output will be as shown below:

```
-rw-rw-r--  1 demo    cad1      1913355 Aug 31 12:11 big.fsdb
-rw-rw-r--  1 demo    cad1      1996022 Aug 31 12:13 big_2.fsdb
-rw-rw-r--  1 demo    cad1      2016818 Aug 31 12:14 big_3.fsdb
-rw-rw-r--  1 demo    cad1      2014343 Aug 31 12:16 big_4.fsdb
-rw-rw-r--  1 demo    cad1      1991496 Aug 31 12:18 big_5.fsdb
-rw-rw-r--  1 demo    cad1      859676  Aug 31 12:24 big_6.fsdb
```

Ideally, this run-time parameter must be set before creating a FSDB database. Even in offline mode, set it before opening the offline database.

In case, a FSDB database already exists, first close the existing database and then create a new one after setting the `fsdbFileSizeLimit` run-time parameter.

The syntax for setting the value of this parameter is:

```
xeset fsdbFileSizeLimit <number>
```

<number>

Specifies the maximum file size limit for a FSDB database. The `<number>` should be a positive integer greater than or equal to 2, which is the minimum file size. The specified file size is in megabytes.

By default, this function is turned off, that is, the value is set to 0.

Example

```
xeset fsdbFileSizeLimit 4
database -open new
=====> Above command would close existing database and open the new database
=====> Use probe command to add probe signals
database -upload
```

Sets the file size limit of the FSDB database to 4MB.

fvMaxThread [<number>]

Specifies the maximum number of CPUs on a host that can be used for DPA.

The `xeset fvMaxThread` command has the same effect as setting the `CDN_FV_MAX_THREAD` environment variable. If both are specified, the `xeset fvMaxThread` command takes precedence.

The syntax for setting the value of this parameter is:

```
xeset fvMaxThread [<number>]
```

<number>

Specifies the maximum number of CPUs on a host.

fvPreprocess

Specifies whether or not to launch the `fvCompute` process after the database `-prepareoffline` command is executed to process the `.phy` trace data file.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset fvPreprocess [0 | 1]
```

0

Indicates that the `fvCompute` process should not be launched.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, the `fvPreprocess` run-time parameter is set to 0.

1

Indicates that the `fvCompute` process should be launched after the database `-prepareoffline` command is executed.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

fvSafeMode [0 | 1]

Enables you to use FullVision in safe mode while debugging the designs in SA mode. Use this option when the waveform upload fails with verification errors. When safe mode is enabled, additional probe samples are taken such that data integrity is always maintained.

Cadence recommends not to use this command unless you encounter upload errors because this command increases the size of the waveform database and the upload time. In some cases, it is possible that the increase will be significant.

The syntax for setting the value of this parameter is:

```
xeset fvSafeMode [0 | 1]
```

0

Disables FV in safe mode. By default, the `fvSafeMode` run-time parameter is set to 0.

1

Enables FV in safe mode.

gui

Specifies whether the xeDebug tool is running in the GUI mode or not.

When xeDebug is running in the GUI mode, the `xeset gui` command returns the value as 1; else, it returns 0.

-gui autoWave

Specifies whether or not to display the waveforms automatically on upload of the trace data.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset -gui autoWave [0 | 1]
```

0

Indicates that the waveforms should not be displayed automatically.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, the `autoWave` run-time parameter is set to 0.

1

Indicates that the waveforms should be displayed automatically.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

-gui dbAutoWave

Specifies whether or not to automatically send the probes to the waveforms display (and bring up the waveform display, if needed) when probes are added to the DUT Design Browser.

It configures the *Display* check box in the *Probing and Tracing* group box of the DUV Design Browser window.

UXE Command Reference Manual

Run-time Commands

The syntax for setting the value of this parameter is:

```
xeset -gui dbAutoWave [0 | 1]
```

0

Indicates that new probes should not be sent automatically to the waveform display.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, the `dbAutoWave` run-time parameter is set to 0.

1

Indicates that new probes should be sent automatically to the waveform display.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

-gui dbAutoUpload

Specifies whether or not to automatically upload waveform data when probes are added to the DUT Design Browser.

It configures the *AutoUpload* check box in the *Probing and Tracing* group box of the DUV Design Browser window.

The syntax for setting the value of this parameter is:

```
xeset -gui dbAutoUpload [0 | 1]
```

0

Indicates that waveform data should not be uploaded automatically.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, the `dbAutoUpload` run-time parameter is set to 0.

1

Indicates that waveform data should be uploaded automatically.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

-gui dbMaxListLen

Specifies the maximum number of entries that should be listed in the design browser.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset -gui dbMaxListLen [<number>]
```

<number>

Specifies the number of entries to be listed in the design browser.

-gui dbShowGenerated

Specifies whether to show or hide in the Design Browser, the objects generated by the compiler.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset -gui dbShowGenerated [0 | 1]
```

0

Indicates that the generated objects should not be displayed in the Design Browser.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, the `dbShowGenerated` run-time parameter is set to 0.

1

Indicates that the generated objects should be displayed in the Design Browser.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

-gui dbShowPrimitiveTerm

Specifies whether to show or hide in the Design Browser, the primitive terminals such as MPR/MPW terminals, and stub modules.

By default, the users are not able to see the MPR/MPW terminals. However, if this option is enabled, the users are able to see the terminals.

Note: Adding the terminals translates their terminal names into the outside net names.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset -gui dbShowPrimitiveTerm [0 | 1]
```

0

Indicates that the primitive terminals should not be displayed in the Design Browser.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

By default, the `dbShowPrimitiveTerm` run-time parameter is set to 0.

1

Indicates that the primitive terminals should be displayed in the Design Browser.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

-gui overwriteMemFile

Controls the memory dump operations by defining whether to enable or disable overwriting of existing memory dump files.

When this run-time parameter is specified without any value, it returns the current setting.

The syntax for setting the value of this parameter is:

```
xeset -gui overwriteMemFile [0 | 1]
```

0

Indicates that the existing memory dump files should not be overwritten.

Note: The values `off`, `no`, and `false` are accepted as synonyms for `0`.

1

Indicates that the existing memory dump files should be overwritten.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for `1`.

By default, the `overwritememFile` run-time parameter is set to `1`.

-gui svSaveWaveScript [0|1]

Controls whether or not to save and restore the signals that are displayed on the waveform when debugging the design using the SimVision GUI or Verdi.

When enabled, the `probe -save <file>` command also creates the `<file>_sv.tcl` file, the `probe -import <file>` command also sources the `<file>_sv.tcl` file. The extra file only stores the waveform signals.

The syntax for setting the value of this parameter is:

```
xeset -gui svSaveWaveScript [0 | 1]
```

0

Specifies that the signals should not be saved and restored.

The default value of the `svSaveWaveScript` run-time parameter is `0`.

1

Specifies that the signals should be saved and restored.

-gui svShowGenerated

Specifies whether or not to show the generated objects in the SimVision design browser.

The syntax for setting the value of this parameter is:

```
xeset -gui svShowGenerated [0 | 1]
```

0

Specifies that the generated objects should not be displayed.

1

Specifies that the generated objects should be displayed.

-gui svUploadOnProbe

Specifies if the probes are to be uploaded onto the waveforms as soon as they are added when debugging the design using the SimVision GUI. Consequently, the execution of the database -upload command is controlled by the click of the *Probe + Display* button in the SimVision GUI.

The syntax for setting the value of this parameter is:

```
xeset -gui svUploadOnProbe [0 | 1]
```

0

Specifies that the probes will not be uploaded automatically and you need to manually upload them.

1

Specifies that the probes will be uploaded when added.

-gui userNet1

Defines the signal name to be used by the *UserNet1* button in the *Force Nets* group box of the *Control* tab for force and release operations.

This run-time parameter is not required when running a design without the GUI.

Note: The GUI enables setting of signal names for two user-specified nets, UserNet1 and UserNet2.

The syntax for setting the value of this parameter is:

```
xeset -gui userNet1 <net name>
```

<net_name>

Specifies the name of the user net.

Example

Executing the following command automatically replaces the signal name used by the UserNet1 button on the *Control* tab by the net name U1.U2.U3.my_net:

```
XE> xeset -gui userNet1 U1.U2.U3.my_net
```

-gui userNet2

Defines the signal name to be used by the *UserNet2* button in the *Force Nets* group box of the *Control* tab for force and release operations.

This run-time parameter is not required when running a design without the GUI.

Note: The GUI enables setting of signal names for two user-specified nets, UserNet1 and UserNet2.

The syntax for setting the value of this parameter is:

```
xeset -gui userNet2 <net_name>
```

<net_name>

Specifies the name of the user net.

Example

Executing the following command automatically replaces the signal name used by the UserNet2 button on the *Control* tab by the net name U2.U3.U4.my_net:

```
XE> xeset -gui userNet2 U2.U3.U4.my_net
```

-gui pollingRate

Determines the frequency of GUI updates.

The syntax for setting the value of this parameter is:

```
xeset -gui pollingRate <value>
```

<value>

Specifies the time after which the GUI should be updated.

By default, the value of `pollingRate` is set to 1, which means the goal is of one update per second.

Example

Setting the value to 60 as following means one update every 60 seconds:

```
xeset -gui pollingRate 60
```

host

Stores the name of the current emulation host. This is a read-only parameter and its value is set by the [host](#) command.

Examples

```
debug /mydir/jpeg.test
host ostrich
xeset host
ostrich      << Returned data
```

Shows that the current emulation host name is `ostrich`, as set in the most recent `host` command.

idleExit

Controls exit in the `xeDebug` tool when no activity is noticed in interactive mode, with or without GUI.

The syntax for setting the value of this parameter is:

```
xeset idleExit [<value>]
```

where `<value>` is a non-negative integer number that specifies the time in minutes. The value 0 means that no timeout is set.

By default, the value is set to 0 minutes.

ignoreProbeFailureInForce [value]

Ignores the probe failure errors while forcing and depositing the signals in ICE mode, and changes the errors into warnings.

The syntax for setting the value of this option is:

```
xeset ignoreProbeFailureInForce {0 | 1}
```

0

Disables the `ignoreProbeFailureInForce` option if the option is enabled.

1

Enables the `ignoreProbeFailureInForce` option to ignore the probe failure errors.

keepHostAlive

Enables or disables the KeepHostAlive feature of xeDebug that enables to retain connection between the host and the emulator even after the design run is finished.

The syntax for setting the value of this parameter is:

```
xeset keepHostAlive {0 | 1 | 2 | 3 | 4}
```



To use this feature in the SA mode, it is required to use value 2, 3, or 4. In ICE mode, values can be either 1, 2, 3, or 4.

Alternatively, use the `XE_KEEP_HOST_ALIVE` environment variable as following to enable or disable the KeepHostAlive feature:

```
setenv XE_KEEP_HOST_ALIVE {0 | 1 | 2 | 3 | 4}
```

Note: This environment setting must be done before starting the xeDebug session.

0

Disables the KeepHostAlive feature, if it is already enabled.

Alternatively, execute the `host -kill` command.

1

Enables the KeepHostAlive feature.

For this run-time parameter setting to take effect, ensure that between the first and each successive xeDebug session the following requirements are fulfilled:

- The design database is the same.
- The host name from where the xeDebug session is invoked is the same.
Note: The above two requirements are easily met when the same command shell or same shell script is used during each xeDebug session.
- The parent process ID (PID) of the xeDebug sessions is the same.
- The .bp or .br files during the first xeDebug session and the successive xeDebug sessions are the same. If .bp or .br files do not exist during the first xeDebug session, they should not exist during the successive xeDebug sessions also.

2

Enables the KeepHostAlive feature. This is the default value of the `keepHostAlive` run-time parameter.

For this run-time parameter setting to take effect, ensure that between the first and each successive xeDebug session the following requirements are fulfilled:

- The design database is the same.
- The username is the same.
- The .bp or .br files during the first xeDebug session and the successive xeDebug sessions are the same.

Note: If .bp or .br files have not been defined for the design, the behavior of the `xeset keepHostAlive` run-time parameter set to value 2 and 3 is the same.

Alternatively, before downloading the design, execute the following command to enable the KeepHostAlive feature:

host -keepAlive

Note: If the `-keepAlive <hostname>` option of the `host` command is used, the `host <hostname>` command must not be executed again for the same debug session.

3

Enables the KeepHostAlive feature.

For this run-time parameter setting to take effect, ensure that between the first and each successive xeDebug session the following requirements are fulfilled:

- The design database is the same.
- The username is the same.
- If `.bp` or `.br` files existed during the first xeDebug session, they need not exist during the successive xeDebug sessions. However, if they exist in the first and successive sessions, their values need not remain the same during all xeDebug sessions. Note that, `.bp` or `.br` files (if present) are taken into consideration during design download in the first xeDebug session.

Note: If `.bp` or `.br` files have not been defined for the design, the behavior of the `xeset keepHostAlive` run-time parameter set to value 2 and 3 is the same.

4

Enables the KeepHostAlive feature.

The `KeepHostAlive 4` run-time parameter indicates that successive xeDebug sessions do not need to be run by the same user.

For this run-time parameter setting to take effect, ensure that between the first and each successive xeDebug session the following requirements are fulfilled:

- The design database is the same.
- All users of the design have write permissions to the `work` directory.
- All users have set the same value of the `XE_DB_SVR_RPC_NUM` environment variable.

keepHostAliveTimeout

Specifies the timeout value (in seconds) to release the hardware resources used in previous xeDebug session with KeepHostAlive feature enabled. If no xeDebug process starts within this time, the emulation resources on the host will be released.

The syntax for setting the value of this parameter is:

```
xeset keepHostAliveTimeout [<value>]
```

<value>

Specifies the timeout value in seconds. The default timeout value is 180 seconds.

Alternatively, this timeout value can be controlled by setting the following environment variable:

```
setenv XE_KEEP_HOST_ALIVE_TIMEOUT <value>
```

If both the ways of specifying the timeout period are used with conflicting values, the `xeset keepHostAliveTimeout <value>` command is given priority.

linkFiles

Specifies which files or directories should be linked from the design directory to the debug session directory by the `debug` command. The value of this parameter is specified as a list of glob-style patterns.

When a session is created using the `debug . -session <session_name>` command, in the ICE mode, by default links for the following directories and files get created within the session directory:

- PDB
- QTDB
- cellList
- dbFiles
- .xeBitMode
- .design

In the SA mode, by default the session directory includes a link to the `xc_work` directory.



The session directory does not include links to the temp files, logs, and run-time generated files.

Use the following syntax for specifying a glob-style pattern matching on the files and directories that need to be linked to the session directory in addition to the default links:

```
xeset linkFiles {[<pattern> ...] [-ice <pattern> ...] [-sa <pattern> ...]}
```

By default, the value of the `linkFiles` run-time parameter is set to blank, that is, {}.

<pattern>

Specifies the pattern of the additional files and directories to be linked using the glob-style pattern matching.

-ice <pattern>

Specifies the pattern which is matched only in the ICE database (which covers LA, STB, and VD emulation modes). Without the `-ice` argument, the `<pattern>` parameter works for all databases.

-sa <pattern>

Specifies the search pattern for the SA database. Without the `-sa` argument, the `<pattern>` parameter works for all databases.

Examples

```
xeset linkFiles {*.mem -ice *.tdf}
```

Creates links in the session directory for all files with `.mem` and `.tdf` extensions. Files with `.mem` extension are created in all emulation modes, while files with `.tdf` extensions are created only for ICE.

```
xeset linkFiles *
```

Creates links in the session directory for all files irrespective of their extensions.

```
xeset linkFiles {}
```

Creates links in the session directory for only the default directories and files.

maxMismatches

Sets the maximum number of vector mismatches that can be reported in the output file during vector comparison in the Vector Debug mode. When the maximum number of vector mismatches is reached, the vector comparison stops.

The default value set for the maximum vector mismatches is 2048. In other words, if the default value of the `maxMismatches` parameter has not been modified, the vector comparison will automatically stop after 2048 vector mismatches are reported.

The syntax for setting the value of this parameter is:

```
xeset maxMismatches [<number>]
```

<number>

Specifies the maximum number of reported vector mismatches that can be reported before the vector comparison stops.

Example

The following example sets the maximum value of vector mismatches during vector comparison to 4096.

```
xeset maxMismatches 4096
```

msgMaxLen

The value of this parameter determines the maximum number of characters per line in messages written to log files. Set it to a positive integer number.

The syntax for setting the value of this parameter is:

```
xeset msgMaxLen [<chars_per_line>]
```

Examples

```
xeset msgMax 70
```

Sets the maximum number of characters per line to 70 characters.

mtHost

Defines how FullVision computation should be distributed among other workstations. By default, the FullVision computation happens on the emulator host. However, to achieve better performance, you can specify multiple workstations to serve as hosts for the computation. In such case, the computation is done in parallel.

This parameter accepts two possible formats. One specifies directly the list of hosts, and the other specifies that the work should be scheduled by LSF.

This parameter will override the settings done using the `CDN_MT_HOST` environment variable, if it was set up before starting the `xeDebug` tool. The new hosts that are set using this parameter take effect from the next trace upload.

The syntax for setting the value of this parameter is:

```
xeset mtHost {<host1> <host2> <host3>...}
```

or:

```
xeset mtHost {LSF <number_of_hosts> <command_line>}
```

<host1> <host2> <host3>...

Specifies a list of hosts separated by spaces. These hosts are used based on their load and CPU speed.

There is no restriction on the number of host names that can be specified.

<number_of_hosts>

Specifies the number of workstations to be used by LSF.

<command_line>

Specifies the command line for bsub.

Examples

```
xeset mtHost {myhost ostrich}
```

The above-example sets `myhost` and `ostrich` as the host names.

```
xeset mtHost "LSF 10 bsub -q dpaq"
```

The above-example uses LSF to schedule the computation.

postTriggerSamples

Defines the number of additional probe samples that will be captured in the trace buffer after the SDL trigger point, when you emulate the design in the STB, LA, or SA mode. It is not used in VD mode.

When the trace buffer fills up to the specified amount:

- In the LA mode, only capturing of probe samples is stopped.
- In the STB and SA modes, both capturing of probe samples and design clocks are stopped.

The `postTriggerSamples` parameter is linked to two other parameters: `traceMemSize` and `triggerPos`. For detailed discussion on how the three parameters are linked and affect each other, see description of parameter `triggerPos`.

Setting the `postTriggerSamples` parameter affects all subsequent `run` commands.

When the SDL program is loaded into the hardware (for example, when executing a `run` command), the value of the `postTriggerSamples` parameter is updated if the SDL program has one of the following statements:

```
postTriggerSamples = <value>;  
triggerPos = <value>;
```

At the end of a run, the actual number of post trigger samples in the trace buffer is not always guaranteed to be exactly the number that was specified using this parameter. Several possible reasons can cause a discrepancy, such as aborting the run before a trigger happens or switching the settings of `stopDelay` in the middle of the run.

The syntax for setting the value of this parameter is:

```
xeset postTriggerSamples [<value>]
```

<value>

Number of probe samples to be captured after the trigger.

Examples

```
xeset postTriggerSamples 5
```

Requests to capture 5 additional probe samples after the trigger.

probeCompiled [0 | 1]

Controls whether or not to automatically add the probes defined at compile time into the waveform database in ICE mode.

The syntax for setting the value of this parameter is:

```
xeset probeCompiled [0 | 1]
```

0

Indicates that the probes defined at compile time should not be added automatically into the waveform database.

The default value of the `probeCompiled` parameter is 0.

1

Indicates that the probes defined at compile time should be added automatically into the waveform database.

probePrimary

Specifies primary inputs and outputs in probe signals to capture trace data. This parameter applies only in the LA, STB, and Vector Debug modes.

The syntax for setting the value of this parameter is:

```
xeset probePrimary [0 | 1]
```

0

Indicates that the primary inputs and outputs should not be added in the probed signal.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

1

Indicates that the primary inputs and outputs should be added in the probed signal.

By default, this parameter is set to 1 for LA and STB modes; whereas to 0 for Vector Debug mode.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

When this run-time parameter is set without any value, it returns the current setting (that is, 0 or 1)

Examples

```
xeset probePrimary 1
```

Adds primary inputs and outputs in probe signal.

```
xeset probePrimary  
1           << Returned data
```

Returns the current value of the `probePrimary` parameter.

progress [off | gui | display | stdout | file | file+ | gui, display | gui, stdout | gui, file | gui, file+ | stdout, file | stdout, file+ | gui, stdout, file | gui, stdout, file+]

Long operations provide a progress display to indicate the status of the command until it completes. An example of a long operation is uploading data to a waveform.

The `xeset progress` command enables you to control how to display the progress for any long operation. The options include displaying the progress to a widget in the GUI, writing to a file, writing to standard out (stdout), various combinations of the above, or disabling it. The default in the GUI mode is to display the progress in a widget. By default, the progress is not displayed in the non-GUI mode.

The syntax for setting the value of this parameter is:

```
xeset progress [off | gui | display | stdout | file | file+ |  
                 gui, display | gui, stdout | gui, file | gui, file+ |  
                 stdout, file | stdout, file+ |  
                 gui, stdout, file | gui, stdout, file+]
```

You can use more than one output method at the same time and it needs to be separated by a comma. The order of the comma-separated options is interchangeable, that is, "gui, display" is the same as "display, gui".

Note: The `display` option cannot be used with the `file` or `file+` options. The `display` option is ignored when used with `stdout` option, that is "display, stdout" is interpreted as "stdout" only.

off

Does not display the progress. The default option for the non-GUI mode is `off`.

gui

Displays the progress in a Tk widget and it is applicable only in the GUI mode. The default option for the GUI mode is `gui`.

display

Displays the current progress state on the `xterm` in the GUI mode or on the console in the non-GUI mode. The line displaying the progress is temporary, it gets overwritten by the next progress update, and goes away after the command is complete. It does not get recorded in the `xe.msg` file or any other log files.

stdout

Displays the current progress state on the console in both GUI and non-GUI modes. The lines displaying the progress accumulate, that is, they do not get overwritten, and also get recorded in the `xe.msg` file and the other log files.

file

Writes a line describing the current progress state to the `tmp/progress.log` file. This line is temporary, it gets overwritten by the next progress update, and goes away after the command is complete.

file+

Writes a line describing the current progress state to the `tmp/progress.log` file. The lines accumulate, that is, they do not get overwritten, and do not go away after the command is complete.

reserveKey

Enables to manually provide the resource reservation key when downloading the design. When this run-time parameter is set, the reservation key value from the user name-specific file available in the `./dbFiles` directory is ignored.

The syntax for setting the value of this parameter is:

```
xeset reserveKey [<key_value>]
```

<key_value>

Specifies the required key value. The value should be a positive integer.

Alternatively, the key value can be specified by setting the `XE_RESERVE_KEY` environment variable as following:

```
setenv XE_RESERVE_KEY <key_value>
```

Note: If both `xeset reserveKey` run-time parameter and `XE_RESERVE_KEY` environment variable are used with conflicting key values, the key value set using the run-time parameter is given priority. When reserving resources, the `test_server` program does not use the key value specified with the `XE_RESERVE_KEY` environment variable.

reserveTimeout <timeoutValue>

Specifies the number of seconds for which the required domains are to be reserved. The `<timeoutValue>` is a positive integer.

Note: The `xeset reserveTimeout <timeoutValue>` command executed before the `host` command provides identical functionality to the `host -timeout <timeoutValue>` command. If both are specified, the `host -timeout` command takes precedence.

retryDownload

When the `download` command is executed and the emulator is locked, the debugger tool tries to download the design repetitively until a successful download is accomplished. The integer value `<N>` assigned to this parameter is the extra number of download attempts.

The retry takes place only when the emulation resources are busy by other users. If the download fails due to other reasons, for example, design relocation error, invalid configuration error, insufficient resources, or if the emulator is offline, a retry does not take place.

The syntax for setting the value of this parameter is:

```
xeset retryDownload <N>
```

The default value of `retryDownload` is 0. This means that if the design is not downloaded at the first instance due to the resources used or locked by other users, the debugger tool will not retry downloading the design.

retryInterval

When the `download` command is executed and the emulator is locked, the debugger tool tries to download the design repetitively until a successful download is accomplished. The integer value `<T>` assigned to this parameter is the time interval in seconds between successive download attempts.

The syntax for setting the value of this parameter is:

```
xeset retryInterval <T>
```

The default value of `retryInterval` is 10, that is, each retry would be made after 10 seconds.

sdlAutoNcBreakpoints

Specifies in IXCOM mode whether to automatically import NC-Sim breakpoints during swap-in (when switching from simulation to the SA mode). The value can be set to either 0 or 1.

- 0: Disable automatic breakpoints import on swap-in.
- 1: Enable automatic breakpoints import on swap-in.

Note: yes, on and true are accepted as synonyms for 1; and no, off and false are accepted as synonyms for 0.

If automatic breakpoints import is enabled, during swap-in, the system performs automatically the equivalent of the `sdl -ncbreakpoints -import` command. Any error condition during the import causes the swap-in to fail and return to simulation mode unless you previously requested to ignore import errors with the `sdl -ncbreakpoints -errorok 1` command.

In addition, if automatic breakpoints import is enabled, during swap-out (when switching back to simulation mode) the system performs automatically the equivalent of the `sdl -ncbreakpoints -restore` command.

sdlIncludePath

Specifies the list of directories to search for files when SDL file inclusion directive is used.

For detailed information on SDL file inclusion directives, refer to the *File Inclusion* section of *Controlling the Run with SDL* of the *UXE User Guide*.

The syntax for setting the value of this parameter is:

```
xeset sdlIncludePath {<dirname> ...}
```

<dirname>

Specifies the path to the directory that needs to be searched. Multiple directory paths can be specified in a string format with a space between each path.

Examples

```
xeset sdlinc {/home/test1/myfiles/sdl /home/test2/projectx}
```

Specifies the path to the `sdl` and `projectx` directories that need to be searched for SDL file inclusion.

signalRadix

Enables retrieval of the symbol/signal value in non-binary format.

Setting the value of this parameter will affect the format of the value returned from all subsequent executions of the `value` command. Note that this affects only the values returned by the `value` command.

The syntax for setting the value of this parameter is:

```
xeset signalRadix <radix>
```

<radix>

Sets the radix. The valid values for radix are:

- `bin` for setting it to binary format
- `oct` for setting it to octal format
- `dec` for setting it to decimal format
- `hex` for setting it to hexadecimal format

Examples

```
force A[3:0] 'b0101
xeset signalRadix bin
value A[3:0]
'b0101                                << Returned data
xeset signalRadix dec
value A[3:0]
'd5                                    << Returned data
```

stopDelay

Specifies the delay between the time the conditions for SDL trigger or EXEC actions happen and the actual time at which these actions are executed. This command applies only to emulation modes that enable SDL to stop the design clocks: VD, STB, and SA under IXCOM.

The value of the `stopDelay` run-time parameter can be specified as 0 or 1. Its default value is 1.

By default (when the value of `stopDelay` is 1), when requesting to break using an SDL trigger, the design clocks might stop with a delay of up to 2 FCLK cycles after the trigger point. The exact amount of delay might vary depending on the emulation mode and other factors. In STB and VD modes it is typically 2 FCLK cycles.

Under IXCOM, when running in SA mode (when the value of `stopDelay` is 1), EXEC or TRIGGER actions will cause the design to stop in either the same time slot in which the condition happened, or in the following time slot.

The same amount of delay applies also for SDL EXEC actions.

This applies to triggers generated by both the SDL program and the independent trigger expression.

When running in STB or VD mode, specifying the value as 0 instructs the instrumentation to stop the design clocks (or execute an EXEC action) in the same cycle where the trigger or EXEC happens (without any delay).

Under IXCOM, specifying the value as 0 only guarantees that the EXEC or TRIGGER are executed in the same simulation time in which the condition that generated them happened, but it does not guarantee that when the emulator stops (or the EXEC action is executed), the conditions that caused these actions are still present. The reason is that each simulation time slot might take several FCLKs to evaluate. The TRIGGER or EXEC actions can be detected during any one of these FCLKs, but they are not executed until the end of the time slot. By that time, the conditions that caused them might have already disappeared.

When specifying a value of 0, the design clocks will run at one-third of the normal speed. This is done by extending each emulation cycle (FCLK) to three times its regular period.

Keep in mind that not always overall run speed is affected much by design clock speed. For example, in VD mode, a significant amount of time is spent on loading/unloading the test vectors, while in the SA mode, a significant amount of time might be spent on software simulation. In such cases, the value assigned to `stopDelay` might have just a very small effect on overall speed.

The syntax for setting the value of this parameter is:

```
xeset stopDelay [0|1]
```

0

In the STB or VD mode, stops the emulator or execute EXEC requests immediately in the same design cycle as the SDL TRIGGER or EXEC action. Under IXCOM, the TRIGGER or EXEC actions happen in the same simulation time in which they were issued by SDL. However, clock speed in such case is one-third of the full speed.

Note: The values off, no and false are accepted as synonyms of 0.

1

In the STB or VD mode, stops the emulator or execute EXEC actions with a delay of up to two FCLK cycles after the SDL TRIGGER or EXEC action happened. Under IXCOM, this might cause the TRIGGER or EXEC action to happen with some simulation time delay. This is the default value of the stopDelay parameter.

Note: The values on, yes, and true are accepted as synonyms of 1.

Example

```
xeset stopDelay 1
# run at regular speed until a trigger happens.
# clocks might stop with delay after trigger
run
# Run again, but request to break with zero delay after trigger
stopDelay 0
run
```

suspendEnable

Enables or disables the remote suspend or resume request. The following command returns the value of this parameter:

```
xeset suspendEnable
```

You can set the default value of the suspendEnable parameter in the .xerc file or specify it at the XE prompt in the xeDebug/GUI session.

The syntax for setting the value of this parameter is:

```
xeset suspendEnable [0|1]
```

UXE Command Reference Manual

Run-time Commands

Note: The `xeset suspendEnable [0|1]` command only affects the remote user.

0

Disables the remote suspend and resume requests.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

1

Enables the remote suspend and resume requests.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for 1.

Examples

```
XE> xeset suspendEnable 1      << enable remote suspend/resume request
XE> debug .
XE> host myhost
XE> download
XE> configPM -stb
XE> run                      << blocking run
XE> suspend                   << suspend the emulation and release the emulator
XE> resume                    << reload the design from the last stopping point
XE> run                        << continue the emulation
```

timeRangeCheck

Checks the range of time.

The syntax for setting the value of this parameter is:

```
xeset timeRangeCheck [0|1]
```

0

Disables the checking of time range.

Note: The values `off`, `no`, and `false` are accepted as synonyms for 0.

1

Enables the checking of time range.

Note: The values `on`, `yes`, and `true` are accepted as synonyms for `1`.

timeUnit

Specifies the default time units that should be used by some commands when they are called without explicit time units.

The syntax for setting the value of this parameter is:

```
xeset timeUnit <value>
```

Note: In SA mode, the `xeset timeUnit` command does not change the time unit of the `run` command. However, you can use the `set time_unit <value>` command of `ncsim` to specify the time unit of the `run` command.

<value>

The `<value>` can be one of the following:

- `fs` - femtoseconds
- `ps` - picoseconds
- `ns` - nanoseconds
- `us` - microseconds
- `ms` - milliseconds
- `s` - seconds
- `sec` - seconds
- `fclk` - Fast Clock (FCLK)
- `dclk` - Design Clock

Note: The values are recognized in either lower or upper case.

In the ICE mode, after the tool initialization, `fclk` is assigned as the default value for this run-time parameter. In the SA mode, the `fclk` and `dclk` time units are not supported; therefore, `ns` is assigned as the default value.

Additional Information

Fast Clock (FCLK) is the basic emulation clock. Operations such as SDL evaluations, or probe samples are paced by this clock. For example, SDL evaluations are always done one per FCLK cycle.

When running in CAKE clocking mode, each `dclk` cycle equals N `fclk` cycles where N is the oversampling ratio as defined during compilation by the command `clockOption -add technology CAKE <value>`, or at run time by the `clockConfig -sample <value>` command. In other clocking modes, `fclk` and `dclk` are considered the same.

When running in CAKE clocking mode, the clock frequency of the Design Clock is defined as the frequency of the fastest clock that was defined at compile time (by the `clockFrequency` command). The frequency of FCLK equals to the frequency of the Design Clock multiplied by the oversampling ratio. These frequencies can be modified later by the `clockConfig` command.

Running in the CAKE clocking mode has the following effect on the waveform display:

- If the waveform upload is performed with `timeUnit` set to value `fclk` or `dclk`, the waveform time scale shows 1ns per waveform sample. (With CAKE oversampling 1 or 0.5, there are two waveform samples per FCLK; with oversampling 2 or higher, there is one waveform sample per FCLK.)
- If the waveform upload is performed with `timeUnit` set to value other than `fclk` or `dclk`, the waveform time scale is the Simulated Time Scale. For example, a CAKE clock with frequency 500Mhz has a Simulated Time Scale of 2ns.

When not using CAKE clocking mode, a default value of 1ns is assigned to each FCLK on the waveform, except when running in the SA mode, which shows the simulation time as derived from the software simulator.

Example

```
xeset timeUnit ns
```

Sets the time unit to nanoseconds.

traceMemMax

Stores the maximum trace samples depth that can be captured by the emulator for a particular emulator database. This is a read-only parameter that can be used in LA and STB modes.

Use the `traceMemMax` parameter after the `debug`, `host`, and `download` commands in LA or STB mode.

Note: The `xeset traceMemMax` command is not supported in SA mode.

Example

```
debug.  
host emulator1  
download  
puts "maximal possible trace depth is: [xeset traceMemMax]"  
maximal possible trace depth is: 74898      << printed data
```

traceMemSize

Stores the requested number of samples to upload from the trace buffer into a file in SA, LA, and STB modes. Setting this parameter affects all subsequent `run` and `database -upload` commands.

Changing this parameter might also affect the number of additional clock cycles that are counted after a trigger happens before the clocks or tracing are stopped. For details, refer to description of parameter `triggerPos`.

The syntax for setting the value of this parameter is:

```
xeset traceMemSize <size>
```

<size>

Specifies the amount of traced data to be uploaded from trace memory into the trace file. It can be specified in one of the following ways:

- Number of probe samples, using an integer followed by "fclk".
- Simulation time, using an integer or a floating point number followed by any of the units (except `fclk` or `dclk`) that are accepted by the `timeUnit` run-time parameter.
- Percentage value of the maximal possible trace memory size, using an integer or floating point number followed by the character %.

Note: `traceMemSize %` is not valid in SA mode.

If you specify `<size>` as a number with no units, `fclk`, or `%` symbol, the units will be the current setting of the `xeset timeUnit` variable.

The numeric part of the value is allowed only in decimal. Octal and hexadecimal formats are not allowed.

When using time units (including `fclk`) or percentage, the number and the time units or % might optionally be separated by a space, but in such case they both need to be enclosed inside " " or {}.

Examples

```
xeset traceMemSize 3000fclk
```

Sets the trace memory size to 3000 probe samples.

```
xeset traceMemSize 1600ns
```

Sets the trace memory size to 1600 nanoseconds.

```
xeset traceMemSize  
1600 ns           << Returned data
```

Returns the current value of the `traceMemSize` parameter, that is, 1600 ns.

```
xeset traceMemSize 100%
```

Sets the trace memory size to the maximal possible value.

```
xeset traceMemSize {50 %}
```

Sets the trace memory size to half the maximal size.

triggerPos

Defines the trigger position when the design is emulated in the STB, LA, or SA modes. It is not supported in VD mode.

The trigger position is defined as the relative location (in percentage) of the SDL trigger point in the trace buffer.

When the trace buffer fills up to the specified amount:

- In LA mode, only capturing of probe samples is stopped.
- In STB and SA modes, both capturing of probe samples and design clocks are stopped.

The actual trigger position might differ from the value set in the `triggerPos` parameter. For example, if you abort the run prematurely, or if the trigger occurs before enough samples are captured, the trigger point is not positioned to the requested percentage in the result.

Using conditional acquisition in FullVision mode might also cause the actual percentage to differ from the requested value. There could also be other reasons. For better accuracy use `postTriggerSamples` instead of `triggerPos`.

The exact interpretation of the trigger position value depends on whether `traceMemSize` was specified in number of samples (using `fclk` as units) or in simulation time units (for example, `ns`).

If `traceMemSize` was specified in number of samples (`fclk` units), `triggerPos` indicates relative position measured in number of samples. For example, if `traceMemSize` is set to `100fclk` and `triggerPos` is set to 30, after a successful trigger, the trace buffer will have 30 probe samples before the trigger point, and 70 samples after the trigger point, even in the presence of conditional acquisition.

If `traceMemSize` is specified in simulation time units, `triggerPos` indicates relative position measured in simulation time. For example, if `traceMemSize` is set to `100ns` and `triggerPos` is set to 30, after a successful trigger, the trace buffer shows trace samples from up to `30ns` before and `70ns` after the trigger point. These might not necessarily correspond to 30/70 ratio of samples due to conditional acquisition or other factors (for example, SA mode does not guarantee any fixed relationship between simulation time and number of probe samples).

This parameter is linked to two other parameters: `traceMemSize` and `postTriggerSamples`. If `TraceMemSize` is specified in number of samples (that is, using `fclk` units) then the software makes sure that the following formula is always correct:

```
postTriggerSamples = [traceMemSize * (100 - triggerPos)]/100
```

- If `postTriggerSamples` is set to a new value, the software re-adjusts the value of `triggerPos` to match the formula.
- If `triggerPos` is set to a new value, the software re-adjusts the value of `postTriggerSamples` to match the formula.
- If `traceMemSize` is set to a new value, the software re-adjusts the value of either `postTriggerSamples` or `triggerPos`, depending on which of the two was modified most recently. The software will leave the most recently modified parameter unaltered.

If `TraceMemSize` is specified in simulation time units (for example, `ns`), setting a new value to `triggerPos` will make the value of `postTriggerSamples` irrelevant, and setting a new value to `triggerPos` will make the value of `postTriggerSamples` irrelevant. If the value is irrelevant, it is returned as an empty string. For example, consider the following sequence of commands:

```
xeset traceMemSize 300us
xeset triggerPos 75
puts "post trigger samples=[xeset postTriggerSamples]"
```

UXE Command Reference Manual

Run-time Commands

```
puts "trigger position=[xeset triggerPos]"
```

The printout will look as follows:

```
post trigger samples=
trigger position=75
```

Setting the `triggerPos` parameter affects all subsequent `run` commands.

When the SDL program is loaded into the hardware (for example, when executing a `run` command), the value of the `triggerPos` parameter is updated if the SDL program has one of the following statements:

```
postTriggerSamples = <value>;
triggerPos = <value>;
```

The syntax for setting the value of this parameter is:

```
xeset triggerPos [<percent>]
```

<percent>

Sets a trigger position as a percent of the trace size set in the `traceMemSize` parameter.

Examples

```
xeset triggerPos 30
```

The above command specifies the trigger position at 30% of the complete trace. The result has 30% pre-trigger samples (or pre-trigger simulation time) and 70% post-trigger samples (or post trigger simulation time).

```
xeset triggerPos
30           << Returned data
```

The above command returns the current value of the `triggerPos` parameter, that is, 30.

uploadSdlTrace [0 | 1]

Specifies whether automatic upload of regular waveform will also cause automatic upload of SDL trace. For information on automatic upload of regular waveforms see documentation of the `xeset autoUpload` command.

The syntax for setting the value of this parameter is:

```
xeset uploadSdlTrace [0 | 1]
```

0

Indicates that the SDL trace data should not be uploaded automatically. It will be uploaded only when the `sdl -traceDump` command is executed.

The default value of the `uploadSdlTrace` run-time parameter is 0.

1

Indicates that automatic upload of regular waveform will also cause automatic upload of SDL trace.

vdCreateWaves [0 | 1]

Determines if stimulus and results waveforms need to be created. If the `vdCreateWaves` command is executed prior to the `vector` command, stimulus and expected results waveforms will be produced. If, however, the `vdCreateWaves` command is executed after the `vector` command, results waveforms will be produced for all following runs.

The syntax for setting the value of this parameter is:

```
xeset vdCreateWaves [0 | 1]
```

0

Suppresses creation of the waveform database from both stimulus and emulation results. This is the default behavior.

1

Enables creation of the waveform database. The `xeset vdCreateWaves 1` command must be executed prior to the `vector` command to enable successful generation of the SST2 and FSDB stimulus and expected results waveforms.

vdLegacyCompare [0 | 1]

Performance of Vector Debug comparison has been enhanced to use a dedicated results memory rather than uploading the waveform data. The `xeset vdLegacyCompare` command determines whether the new or the legacy Vector Debug compare method should be used.

UXE Command Reference Manual

Run-time Commands

The syntax for setting the value of this parameter is:

```
xeset vdLegacyCompare[0 | 1]
```

0

Indicates that the new VD compare method will be used. The new method is enabled by default.

1

Enables the legacy VD compare method when uploads are automatically performed and the results compared with the expected values. When set, this option directs Vector Debug handling to compare the expected and observed results using the legacy method of uploading the VBF format and comparing to the stimulus or expected VBF file.

The `xeset vdLegacyCompare` command must be executed before the [download](#) command and must not be changed during the run-time session.

vector

Stores the name of the vector file containing input stimulus and expected values to apply to the design during emulation. This read-only parameter is used only when the design is emulated in Vector Debug mode. The value of this parameter is set by the [vector](#) command.

For more information on the vector file used in Vector Debug mode, refer to the `vector` command.

Examples

```
debug /mydir/jpeg.test
host bert
configPM -vd
vector JPEGP.cbc
xeset vector
JPEGP.cbc           << Returned data
```

Returns the name of the current vector file, `JPEGP.cbc`.

version

Stores the software version. This is a read-only parameter. The value for this parameter is set after xeDebug is invoked.

Examples

```
xeDebug  
xeset version  
11.1           << Returned data
```

Returns the software version, 11.1.

xegui

Enables the users to customize the xeDebug GUI and the vvmDebug GUI by adding their own menus, tabs, buttons, and so on. For information on adding custom elements to the xeDebug GUI, refer to the *Customizing the xeDebug GUI* section of the *Using the xeDebug GUI* chapter in *UXE User Guide*.

```
xegui -create <name> [-type <type>] [-location <location>] [-pack <options>] [<Tk  
widget_options>]  
xegui -destroy <name>  
xegui -configure <name> <options>  
xegui -apply <name> <arguments>  
xegui -save <file_name>
```

-create <name> [-type <type>] [-location <location>] [-pack <options>] [<Tk widget_options>]

Creates the specified GUI element of one of the existing Tk types at the specified location in the xeDebug GUI.

Here,

- **<name>** is an arbitrary string needed for reference in the other `xegui` commands, such as `-destroy` and `-configure`.
- **<type>** is the type of the widget that you want to create. If not specified, the default value for `<type>` is `button`. The pre-defined values that can be specified are:

button	checkbutton	entry	frame
image	label	listbox	menu
radiobutton	scale	spinbox	tab text
ttk::button	ttk::checkbutton	ttk::combobox	ttk::entry
ttk::labelframe	ttk::radiobutton	ttk::scale	ttk::separator
scrollbar			

- **<location>** is the location on the xeDebug GUI, either `top` or `bottom` can be specified. The `top` refers to the location on the left side of *Preferences* button; the

UXE Command Reference Manual

Run-time Commands

`bottom` refers to the location right above the status bar. By default the `<location>` is `top`.

Any Tk options that are accepted by an appropriate Tk widget can be specified with the `-create` command, for example, `-width`, `-relief` and so on.

For example:

To create a custom tab named *mytab*, use the following command:

```
xegui -create mytab -type tab
```

To create a custom menu named *My menu* with a submenu *Force abc*, use the following command:

```
xegui -create "My menu>Force abc" -command "force abc $abc"
```

By default, the `xegui` command packs the widgets even without specifying any options. However, for special packing, use `-pack` option with the `-create` command. To place a widget in a custom tab, the value of `-pack` option must have the `-in` option. For example,

```
xegui -create f1 -type frame -pack "-in mytab -fill both"
```

-destroy <name>

Deletes the specified GUI element.

For example, the following command deletes the widget named *check1*:

```
xegui -destroy check1
```

-configure <name> <options>

Modifies the options of an existing custom widget. All the Tk options specific to the widget can be specified.

For example:

```
xegui -configure reset1 -state disabled
```

-apply <name> <arguments>

Executes any Tk widget commands specified with `-apply` option. The widget command specified with `-apply` command option are directly passed to Tk.

For example:

UXE Command Reference Manual

Run-time Commands

- The following command executes the `invoke` command of the `reset1` widget:

```
xegui -apply reset1 invoke
```

- The following command gets the complete content from the `text1` widget:

```
set txt [xegui -apply text1 get 0.0 end];
```

-save <file_name>

Saves the changes made to the xeDebug GUI.

Note: If the file name is an empty string, the command returns a script as a Tcl result that can be evaluated by using the `eval` Tcl command.

UXE Command Reference Manual

Run-time Commands

SA Run-time and Debug Commands

This chapter describes certain commands that are specifically used while running or debugging designs in SA mode.

Designs can be run in Simulation Acceleration mode by using the `xeDebug` and `irun` run-time tools.

To access the design running on the hardware, a special set of CLI commands called the `xc` command set, is provided. The `xc Command Options` commands are used to switch between various simulation modes and to determine how your design runs in the simulation acceleration mode.

In addition, you can use the `ixcc` command to generate run-time reports based on code-coverage data.

xc Command Options

The following `xc` commands can be run in both *irun* and xeDebug tools.

Note: To view the syntax of each `xc` subcommand, specify the `help` command as `help xc_<subcommand>`. For example, to display the syntax help for the `xc_checkxoff` command, execute `help xc_checkxoff`.

[xc_asrtFailMsg](#)
[xc_checkxoff](#)
[xc_checkxon](#)
[xc_chkmem](#)
[xc_chkreg](#)
[xc_chkregx](#)
[xc_ckpt](#)
[xc_coverage_dump](#)
[xc_coverage_functional -reset](#)
[xc_coverage_code -reset](#)
[xc_coverage_off](#)
[xc_coverage_reset](#)
[xc_coverage_setup](#)
[xc_ecmset maxBpCycle](#)
[xc_ecmset maxSyncCycle](#)
[xc_ecmset maxGfifoSyncCycle](#)
[xc_emaskclr](#)
[xc_emaskoff](#)
[xc_emaskon](#)
[xc_export_frcrel](#)
[xc_fclkPerEval](#)
[xc_free](#)
[xc_gfifoDispTimeformat](#)
[xc_help](#)
[xc_ixe_sync](#)
[xc_matchmemon](#)

UXE Command Reference Manual

SA Run-time and Debug Commands

xc_matchmemoff

xc_matchregon

xc_matchregoff

xc_off

xc_on

xc_pitraceoff

xc_pitraceon

xc_poclk

xc_poclk add

xc_poclk rm

xc_potraceoff

xc_potraceon

xc_proffoff

xc_profon

xc_showcpi

xc_showcpo

xc_showfrc

xc_status

xc_stop

xc_swapInMem

xc_verbose

xc_wait

xc_xreg

ixcc

Note: The following commands can be executed only in the *iRun* tool. Running these commands in the xeDebug tool issues an error message. For information about the corresponding command to be executed in the xeDebug tool, refer to the description of the command:

xc_deposit

xc_memory

xc_signal

xc_sigTrace

xrun

xc asrtFailMsg

Used to enable or disable the tool-generated assertion failure messages at run time.

The syntax of this command is:

```
xc asrtFailMsg -on | -off | -ncCompat
```

Here,

-on - Enables display of tool-generated assertion failure messages for assertions with user-written fail action blocks without severity tasks. This is the default setting.

-off – Suppresses all assertion-failure messages.

-ncCompat – Enables display of tool-generated failure messages for all assertions including user-defined explicit fail action blocks without severity task calls. This behavior matches IES behavior.

xc checkxoff

Temporarily disables the `xc checkxon` command.

Note: The `xc checkxon` and `xc checkxoff` commands can be entered interchangeably to disable or enable X checking.

The syntax for this command is:

```
xc checkxoff
```

xc checkxon

Causes a warning to be generated any time a primary input to a DUV module is driven to a value of X by the software simulator.

Verilog DUV ports are checked for X values; VHDL ports of type STD_LOGIC or STD_ULOGIC, are checked for X, W, -, or U values. Other data types are not checked. Note that X values generated by the software simulator on DUV primary inputs are passed as either 0 or 1 to the emulator hardware, depending on whether xc on -xt0 or xc on -xt1 is entered with xc on -run, xc on -match, xc on -tbrun, or xc on -autorun.

The syntax for this command is:

```
xc checkxon
```

xc chkmem

Available only in match mode. Compares DUV memories in software versus hardware at one point in simulation time, which is useful for comparing memories at specific points in time.

The syntax for this command is:

```
xc chkmem
```

xc chkreg

Available only in match mode. Compares DUV registers in software versus hardware, which is useful for comparing registers at specific time points.

The syntax for this command is:

```
xc chkreg
```

xc chkregx

Available only in simulation or match mode. Checks the registers for X or Z values at the current time.

The syntax for this command is:

```
xc chkregx
```

xc ckpt

Available only in the hardware mode. Executes all `marg_ckpt_emu2disk()` and `marg_ckpt_disk2emu()` C functions defined in `<uxe_install>/share/uxe/etc/ixcom/marg_ckpt.h`.

The syntax for this command is:

```
xc ckpt [-emu2disk <file> | -disk2emu <file>]
```

Here,

- `-emu2disk <file>`: Saves the emulator state to the disk during hardware run.
- `-disk2emu <file>`: Swaps in a saved emulator state from the disk. The `xc ckpt -disk2emu <file>` command must be issued immediately after the first `xc on -run`, `xc on -tbrun`, or `xc on -nbrun` commands.

xc coverage_dump

Generates the coverage database with the <*db_name*> name. This is equivalent to the IES coverage -dump command. The syntax for this command is:

```
xc coverage_dump <db_name>
```

xc coverage_functional -reset

Resets the coverage counters to zero, just like the [xc coverage_reset](#) command. The only difference between the two commands is that `xc coverage_functional -reset` command restricts the reset to assertion and covergroup coverage. Other counters are not reset.

The syntax for this command is:

```
xc coverage_functional -reset
```

xc coverage_code -reset

Resets the coverage counters to zero, just like the [xc coverage_reset](#) command. The only difference between the two commands is that `xc coverage_code -reset` command restricts the reset to toggle and block coverage. Other counters are not reset.

The syntax for this command is:

```
xc coverage_code -reset
```

xc coverage_off

Disables dumping of the coverage database. It is most useful for disabling the implicit generation of a coverage database at the end of simulation. This is equivalent to the IES coverage -off command. If you issue the `xc coverage_dump` command after `xc coverage_off` command, a non-fatal error message is generated.

The syntax for this command is:

```
xc coverage_off
```

xc coverage_reset

Resets all coverage counters to zero. This command resets only toggle, block, and assertion counters. Covergroups are ignored.

Note: Certain combinational blocks that are specified to always execute, are not reset. These are considered as always covered, provided the logic executes.

If the design cannot be swapped to software, use the `-covAddResetLogic` option of the `ixcom` command to inject reset logic into the design at compile-time.

The syntax for this command is:

```
xc coverage_reset
```

xc coverage_setup

This is equivalent to the IES coverage -setup command. The -dut option is not supported.

The syntax for this command is:

```
xc coverage_setup
```

xc deposit

Deposits a value to DUV signals in both hardware and software modes when *irun* run-time tool is used. This command works only for signals defined in the DUV modules.

Note: When using the xeDebug run-time tool, use the [deposit](#) command described in [Chapter 13, “Run-time Commands.”](#)



Caution

The xc deposit command is available only in the irun tool. In the xeDebug tool, to deposit values, use the deposit command instead of the xc deposit command. Executing the xc deposit command in the xeDebug tool results in a syntax error.

The syntax for this command is:

```
xc deposit <signal_name> <value>
```

xc ecmset maxBpCycle

Specifies the maximum number of behavioral cycles to capture the waveforms. This command limits the behavioral cycles and forces the emulator to stop capturing the waveforms after reaching the specified maximum behavioral cycle threshold.

The syntax for this command is:

```
xc ecmset maxBpCycle <n>
```

Here, *<n>* must be in the range of 2 to 0xffff. The default value of *<n>* is 0. 0 indicates no limit on capturing behavior cycles in the waveform. If more behavior cycles are detected in one time step, the emulator stops capturing the waveform. You can interrupt the run by pressing **Ctrl+C** or by issuing the `touch <run_dir>/tmp/uaPollAbort` command.

For more information, refer to the *Controlling Emulation Hang in SA Mode* section in the *UXE User Guide*.

You can execute the above command at the beginning of the hardware run or in the middle of emulation.

xc ecmset maxSyncCycle

Specifies the maximum number of FCLK cycles between the tbsync events while running the designs in the TBrn mode. This command helps you resolve the emulator hangs in the TBrn mode.

The syntax for this command is:

```
xc ecmset maxSyncCycle <n>
```

Here, *n* must be in the range of 2 to 0xffff. The default value of *n* is 0, that is, by default, the tbsync events are not generated based on FCLK counting.

Note: The `xc ecmset maxSyncCycle` command is not supported in the NBrn Mode.

For more information, refer to the *Controlling Emulation Hang in SA Mode* section in the *UXE User Guide*.

xc ecmset maxGfifoSyncCycle

Specifies the maximum number of FCLK cycles between a GFIFO call and a tbsync event while running the designs in the TBrun mode. This command helps you resolve the emulator hangs in the TBrun mode.

The syntax for this command is:

```
xc ecmset maxGfifoSyncCycle <n>
```

Here, *<n>* must be in the range of 2 to 0xffff. The default value of *<n>* is 0, that is, by default, the tbsync events are not generated based on FCLK counting.

Note: The `xc ecmset maxGfifoSyncCycle` command is not supported in the NBrun Mode.

For more information, refer to the *Controlling Emulation Hang in SA Mode* section in the *UXE User Guide*.

xc emaskclr

Clears the error mask set by `xc emaskon`, that is, all registers, Primary Outputs, and memories previously reported will again get checked for mismatches.

The syntax for this command is:

```
xc emaskclr
```

xc emaskoff

Unmasks the mismatch error reporting, which means all mismatches on all registers, Primary Outputs, and mapped memories in the DUV are reported.

The syntax for this command is:

```
xc emaskoff
```

xc emaskon

Reports only the first mismatch on each register, Primary Output, or mapped memory in the DUV and masks it for subsequent mismatches.

The syntax for this command is:

```
xc emaskon
```

xc export_frcrel

Exports the specified signals from the emulator when *iRun* run-time tool is used. This command creates a special force logic for the specified signals in the hardware and causes the signals to be exported from the emulator, which is required to force and release the signals in SA mode.

Note: The `xc export_frcrel` command is available only in the *iRun* tool. The `xc export_frcrel` command might not work for certain DUT signals that have been optimized by the compiler.

The syntax for this command is:

```
xc export_frcrel <signal>
```

xc fclkPerEval

Specifies the oversampling ration in SA mode. By default, designs are run with 2X sampling ratio, if you have not included the `+1xua` option while compiling the design. You can increase the oversampling ratio at run time by using this command.

The syntax of the command is:

```
xc fclkPerEval <N>
```

According to the value of `<N>`, the corresponding oversampling ratio is defined as $2^{*(N+1)} X$.

For example:1

The following command sets the oversampling ration to 4X.

```
xc fclkPerEval 1
```

The following command sets the oversampling ration to 6X.

```
xc fclkPerEval 2
```

The following command sets the oversampling ration to 8X.

```
xc fclkPerEval 3
```

You must specify the `xc fclkPerEval <N>` command immediately after a command that switches the design run to hardware mode, such as `xc on`, `xc run`, or `xc tbrun` command. For example,

```
ncsim> xc on -tbrun  
ncsim> xc fclkPerEval 1
```

If you have used the `+1xua` option at the time of compiling the design, the `xc fclkPerEval <N>` command has no effect. For using this command, you must remove the `+1xua` option, and compile the design again.

xc free

Frees, explicitly, the emulator hardware for another user. When you swap to hardware, the emulator is locked for you until simulation is terminated; it remains locked even if you just swap back to the software.

The syntax for this command is:

```
xc free
```

xc gfifoDispTimeformat

Sets and lists \$timeformat argument(s) and values, overriding any subsequent \$timeformat call from DUT. The xc gfifoDispTimeformat command sets and shows \$timeformat arguments specifically, unit, precision, suffix-string, minimum width for GFIFO display.

The +fifoDisp option of the ixcom command transforms the \$timeformat system task in DUT running in the hardware partition. However, the \$timeformat system tasks in testbench running in software partition are not supported and ignored by +fifoDisp transformation. For these software-partition tasks, use xc gfifoDispTimeformat run-time command to specify the \$timeformat system-task arguments.

The syntax for this command is:

```
xc gfifoDispTimeformat -unit <n> | -prec <n> | -suffix "<suffix_string>" | \
    -mwidth <n> | -default | -show
```

- ❑ Use -unit <n> to set units field of \$timeformat. Here, <n> can be an integer ranging from <-15 to 0>.
- ❑ Use -prec <n> to set precision field of \$timeformat.
- ❑ Use -suffix "<suffix_string>" to set suffix_string field of \$timeformat.
- ❑ Use -mwidth <n> to set the minimum_field_width field of \$timeformat.
- ❑ Use -default to reset arguments of \$timeformat to the default values, overriding any other settings.
- ❑ Use -show option to list the current argument values of \$timeformat.

For example, suppose the default settings for gfifoDispTimeformat are set as follows:

- unit=-12
- prec=0
- suffix=
- mwidth=20

Use the xc gfifoDispTimeformat command to set the following values for the arguments of gfifoFispTimeformat:

```
ncsim> xc gfifoDispTimeformat -unit -6 -prec 2 -suffix us -mwidth 10
```

Display the current setting for gfifoDispTimeformat as follows:

UXE Command Reference Manual

SA Run-time and Debug Commands

```
ncsim> xc gfifoDispTimeformat -show
```

Following is the output:

```
gfifoDispTimeformat: unit=-6 , prec=2 , suffix= us, mwidth=10
```

Use the `xc gfifoDispTimeformat` command to reset the arguments to default values, and show the values for the arguments of `gfifoFispTimeformat`:

```
ncsim> xc gfifoDispTimeformat -default -show
```

Following is the output:

```
gfifoDispTimeformat: unit=-12 , prec=0 , suffix=, mwidth=20
```

xc help

Lists and briefly describes the `xc` command set.

The syntax for this command is:

```
xc help
```

xc ixe_sync

Reduces the performance overhead in xeDebug when running the designs in SA mode. In SA mode, xeDebug needs to be notified when a simulation run stops to start the interactive mode. The notifications are required for `save` or `restart` and `suspend` or `resume` operations and accessing the objects through dbEngine.

Using a command loop in a simulation run poses significant performance overhead. For example,

```
for {set i 0} { $i < 100} {incr i} {  
run 1000  
}
```

In such cases, issue the `ixe_sync` command if you do not need to access the design objects. This command disables the notifications for xeDebug and improves the performance.

For example,

The calls to `sa_start_run` and `sa_end_run` functions from xeDebug might need around 90ms per invocation of the `run` command. This operation involves multiple passes through xeDebug and IES. The actual net time to execute the run in ncsim is around 100ms and almost 90ms is spent in the control transfer from ncsim to xeDebug and vice-versa. Here, you can use the `ixe_sync` commands to reduce the overhead.

The syntax of this command is:

```
ixe_sync [-on | -off]
```

ixe_sync [-on | -off]

Turns on or off the run stop notification to xeDebug when entering into the interactive mode. The `-on` option enables the run stop notification to xeDebug. The `-off` option disables the run stop notifications.

xc matchmemon

Enables the memory comparison capability of the match. The simulation values of memories mapped to the primitives are compared in software versus hardware.

The syntax for this command is:

```
xc matchmemon
```

xc matchmemoff

Disables the memory comparison capability of the Match.

The syntax for this command is:

```
xc matchmemoff
```

xc matchregon

Enables the register comparison capability of the Match. The simulation values of registers in the DUV are compared in software versus hardware.

The syntax for this command is:

```
xc matchregon
```

xc matchregoff

Disables the register comparison capability of the Match.

The syntax for this command is:

```
xc matchregoff
```

xc memory

Loads a DUV memory from a file or dumps a DUV memory to a file. This command can be used in both software and hardware modes.

Note: This command works for Verilog and VHDL memory, but not for axis_smem.



The xc memory command is available only in the irun tool. In the xeDebug tool, to load and dump memories, use the memory command instead of the xc memory command. Executing the xc memory command in the xeDebug tool results in a syntax error.

The syntax for this command is:

```
xc memory -dump %<file_format> <memory_name> -file <file_name> [-start <n>] [-end <n>] [-z]  
xc memory -load [%<file_format>] <memory_name> -file <file_name> [-start <n>] [-end <n>] [-z]  
xc memory -list  
xc memory -set [<memory> | -all]  
xc memory -reset [<memory> | -all]  
xc memory -setvalue [<memory> | -all] [-start n] [-end n] -value [<value> | all0 | all1]
```

xc memory -dump %<file_format> <memory_name> -file <file_name> [-start <n>] [-end <n>] [-z]

Dumps the contents of the specified memory to the specified file.

Argument	Type	Description
<file_format>	Required	Specifies the format of the output file in which the data will be dumped. The supported file formats are: <ul style="list-style-type: none">■ %h: IES format, radix hex■ %b: IES format, radix bin■ %o: IES format, radix oct■ %readmemb: readmemb format■ %readmemh: readmemh format■ %pd_b: Palladium ASCII format, radix bin■ %pd_o: Palladium ASCII format, radix oct■ %pd_d: Palladium ASCII format, radix dec■ %pd_h: Palladium ASCII format, radix hex■ %pd_memtran: memtran format■ %pd_raw: raw format■ %pd_raw2: raw2 format
<memory_name>	Required	Reads from the memory instance specified in <memory_name>.
-file <file_name>	Required	Specifies the name of the file where the data needs to be dumped. If the file you specify contains some data, it will be overwritten.

UXE Command Reference Manual

SA Run-time and Debug Commands

Argument	Type	Description
<code>[-start <n>] [-end <n>]</code>	Optional	<p>The <code>-start <start_number></code> option specifies the starting memory address. The <code>-end <end_number></code> option specifies an ending memory address. If you do not specify a starting address, the read operation starts from the first address in memory. If you do not specify an ending address, the read operation ends at the last address in memory.</p> <p>The <code><start_number></code> and <code><end_number></code> arguments must fall within the address range of the memory instance specified. The radix for <code><start_number></code> and <code><end_number></code> follows the C/C++ convention. For example, 0x12 is a hex number, 12 is a decimal number, and 012 is an octal number.</p>
<code>-z</code>	Optional	<p>Saves the memory data file in a compressed format, which can be uncompressed using the UNIX <code>uncompress</code> or <code>zcat</code> commands.</p> <p>Note: This option can be used only for <code>raw</code> and <code>raw2</code> file formats.</p>

xc memory -load [%<file_format>] <memory_name> -file <file_name> [-start <n>] [-end <n>] [-z]

Loads data from the specified file into the specified memory.

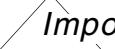
Argument	Type	Description
<code><memory_name></code>	Required	Loads data from a file into memory specified in <code><memory_name></code> argument.
<code>-file <file_name></code>	Required	Specifies the name of the file from where the data needs to be loaded into the memory.

UXE Command Reference Manual

SA Run-time and Debug Commands

Argument	Type	Description
<code>[-start <n>] [-end <n>]</code>	Optional	<p>The <code>-start <start_number></code> option specifies the starting memory address. The <code>-end <end_number></code> option specifies an ending memory address. If you do not specify a starting address, the load operation starts from the first address in memory. If you do not specify an ending address, the load operation ends at the last address in memory.</p> <p>The <code><start_number></code> and <code><end_number></code> arguments must fall within the address range of the memory instance specified. The radix for <code><start_number></code> and <code><end_number></code> follows the C/C++ convention. For example, 0x12 is a hex number, 12 is a decimal number, and 012 is an octal number.</p>

Argument	Type	Description
<i><file_format></i>	Optional	<p>Specifies the format in which the data needs to be loaded in the memory. The format needs to be specified only when the file is in %pd_raw2, %readmemb, or %readmemh format.</p> <p>Note: The <code>memory -dump</code> command must specify the <i><file_format></i>. It is optional for the <code>memory -load</code> command if the file format can be identified from the memory content. However, if the file format cannot be derived from the file content, for example %readmemb, %readmemh, and %pd_raw2 format, it is mandatory to specify <i><file_format></i> with the <code>memory -load</code> command. If the <i><file_format></i> is specified, ensure that it is the first option specified with the <code>memory -load</code> command and is followed by the <i><memory_name></i> argument.</p>

 *Important*

If you are using this option, ensure that this option is the first option specified with the `memory -load` command and is followed by the *<memory_name>* option.

Other formats can identify themselves by the file content. However, you can still specify format, but that should be consistent with the file.

xc memory -list

Lists all the DUV memories.

xc memory -set [<memory> | -all]

Sets the content of all memory instances, or the specified memory instance to one.

xc memory -reset [<memory> | -all]

Sets the content of all memory instances, or the specified memory instance to zero.

xc memory -setvalue [<memory> | -all] [-start n] [-end n] -value [<value> | all0 | all1]

Initializes the specified memory instance, or all the memory instances in DUV, to a given value.

The `-start` and `-end` options specify the range of memory addresses for which this value is set. This range of memory addresses applies to all the specified memory instances, regardless of the size of these memory instances. If the `-start` and `-end` options are not specified with the command, the default start and end are taken from the beginning and end of each memory instance.

The `-value` option can be set to `all0`, `all1`, or any other `<value>`. To set a value other than 0 and 1, ensure that the value is specified in a format that follows C++ (preceding `0x` for hex, 0 for octal) convention.

xc off

First swaps the design at the end of the current time step to software simulation mode from hardware, then continues running in software simulation mode.

The syntax for this command is:

```
xc off
```

xc on

Swaps the design into hardware at the end of the current timestep.

The syntax for this command is:

```
xc on [-autorun | -bmatch | -initok | -match | -nbrun <delay> | -piok | -run |  
-tbrun | -wait_init | -wait_init_stop | -xt0 | -xt1 | -zt0 | -zt1]
```

-autorun

Runs a DUV in targetless emulation or In-Circuit Emulation (ICE) and enables internal clock generation.

Autorun reports an error if the design has any input ports to the DUV. In such cases, use the `xc on -piok` command before the `xc on -autorun` command to ignore the input port error.

-bmatch

Runs the match in batch mode, working the same as the `xc on -match` command, except simulation is not interrupted when mismatches are found. The `xeDebug.log` file contains the results.

-initok

Swaps to co-emulation modes even if events are pending from `initial` blocks. The events continue to be processed by the software simulator, but are ignored by the emulator.

If the design contains `initial` blocks in DUV modules, you should run in software simulation until all events generated by those blocks are simulated. If you attempt to swap to the hardware while events are still pending and the `xc on -initok` command has not been used, an error is generated by default.

-match

Runs the emulator match mode on your design. By default, memories, registers, and Primary Outputs of the DUV are compared in software versus hardware. Each of these comparisons can be individually disabled. Simulation is interrupted when a mismatch is encountered.

To go back to the simulator mode, run the `xc stop` command.

-nbrun <delay>

Runs the design in NBrun mode, which is a hardware run mode where the hardware runs unblocked from the software run. The *<delay>* can be specified as a number with or without time units, such as 100ns or just 100.

-piok

Ignores all PI events during autorun.

-run

Swaps the design into hardware at the end of the current timestep.

The device under verification (DUV) runs in the emulator hardware, and the remaining parts of the design run in software simulation.

-tbrun

Runs a DUV in targetless emulation and enables behavioral delay control.

Example

For the following code:

```
initial
    do some PI activities with short delay;
    #1000000;
    do some other PI activities with short delay;
```

If you use `xc` on `-tbrun` when there are PI activities, the emulator will run like `xc` on `-run` mode. During the long delay period of `#1000000`, the emulator will run like `xc` on `-autorun` mode.

-wait_init

This command option applies to both `xeDebug` and *irun* modes.

To use this option, at compile time place the clocks in the DUV and use either `ixcom +iscDelay` or `$ixc_ctrl("map_delays")` in the `clock` module.

Use of the `xc on -wait_init` command at run time enables you to continue the simulation in software mode and check the active DUV initial blocks at every clock-edge until none are active, and then swap-in.

Note: Execute the `xc verbose` command to view all active initial blocks at every clock-edge.

-wait_init_stop

Stops the simulation after all initial blocks finish execution. This option stops the simulation after the design download and before swap-in.

-xt0

Sets the emulator hardware to translate the signal values to 0 if signals have simulation values of X, W, -, or U (VHDL STD_LOGIC or STD_ULOGIC) or X (Verilog nets or registers) in either of the following situations:

- Anytime during emulation mode, if an input to the DUV goes to any of these values, or if an attempt is made to write these values to a DUV signal.

OR

- At the time the design is swapped from software to hardware, if a VHDL signal or Verilog register signal happens to be at one of these values.

DUV inputs and signals being written to with the above-specified simulation values will be assigned 0 during simulation modes. However, VHDL signals or Verilog registers / signals with any of these values will be assigned 0 when swapped to simulation modes.

-xt1

Sets the emulator hardware to translate the signal values to 1 if signals have simulation values of X, W, -, or U (VHDL STD_LOGIC or STD_ULOGIC) or X (Verilog nets or registers) in either of the situations defined in the `-xt0` command option.

DUV inputs and signals being written to with the above-specified simulation values will be assigned 1 during simulation modes. However, VHDL signals or Verilog registers / signals with any of these values will be assigned 1 when swapped to simulation modes.

-zt0

Sets the emulator hardware to translate the signal values to 0 if signals have simulation values of z in either of the situations defined in the [-xt0](#) command option.

DUV inputs and signals being written to with simulation value of z will be assigned 0 during simulation modes. However, VHDL signals or Verilog registers / signals with simulation value of z will be assigned 0 when swapped to simulation modes.

-zt1

Sets the emulator hardware to translate the signal values to 1 if signals have simulation values of z in either of the situations defined in the [-xt0](#) command option.

DUV inputs and signals being written to with simulation value of z will be assigned 1 during simulation modes. However, VHDL signals or Verilog registers / signals with simulation value of z will be assigned 1 when swapped to simulation modes.

xc pitraceoff

Turns off tracing of the primary inputs to a DUV module.

The syntax for this command is:

```
xc pitraceoff
```

xc pitraceon

Turns on tracing of the primary inputs to a DUV module.

The syntax for this command is:

`xc pitraceon [level]`

level

Specifies the level for tracing the primary inputs. The accepted levels are:

-
- 1 Indicates that the trace data should report only the evaluation.
 - 2 Indicates that the trace data should report the primary inputs, force/release, deposit, and memory write. This is the default level.
 - 3 Indicates that like level 2, the trace data should report the primary inputs, force/release, and deposit. However, memory write should be reported in detail.
-

xc poclk

Displays the current set of DUV or hardware model outputs that are marked as Primary Output Clock (POCLK) signals. Such signals are updated before the other outputs driven by the emulator, enabling you to effectively *delay* the other outputs.

The syntax for this command is:

```
xc poclk
```

Use the `xc poclk add` ([xc poclk add](#)) command to add an output signal as a POCLK and the `xc poclk rm` ([xc poclk rm](#)) command to remove a signal from the POCLK list.

xc poclk add

Treats the specified *hierarchical_signal_name* as a POCLK signal when it is driven by the emulator to the testbench.

The syntax for this command is:

```
xc poclk add <hierarchical_signal_name>
```

hierarchical_signal_name

The hierarchical name of an output or inout port (Primary Output) on the DUV module.

The specified output signal will be updated by the emulator hardware earlier than the other outputs, that is, the other outputs are *delayed* relative to the POCLK output signal. This feature is especially helpful when you are using a functional verification device like the emulator hardware, and you need a *delayed* relationship between different output signals driven by the emulator hardware.

In some cases, the DUV port that you want to add as a POCLK might be named a little differently from that specified in the source code, or you might need to add POCLK to a signal connected to the port rather than the port itself. If you get an error when using the previous command, try using the `xc showcpo` command to get a listing of all Primary Outputs. You can add a POCLK on any of the listed signals.

Example

In the following example, the DUV drives a gated-clock output (`mclk`) signal and a 4-bit output data bus (`dout`). These signals are connected as inputs to a memory instantiated in the testbench. The 4-bit data coming from the DUV is written on the positive-edge of the clock. To make this memory work, the DUV data signals must be *delayed* relative to the DUV clock signal, that is, if both clock and data signals change simultaneously, the memory must write the *old* values that occurred just before the clock-edge. To do this, the `mclk` output signal must be added as a POCLK with the following CLI command:

```
xc poclk add top.mclk
```

Verilog Code

```
module top;
reg clk, ctrl;
reg[3:0] din;
wire[3:0] memout, addr, dutout;
wire mclk;
dut dut (clk, din, ctrl, mclk, addr, dutout);
```

UXE Command Reference Manual

SA Run-time and Debug Commands

```
mem mem (mclk, dutout, memout, addr);
initial begin
  clk=0;
  forever #5 clk = ~clk;
end
initial begin
  #0  ctrl <= 1;
  #25 ctrl <= 0;
  #5   din<=1;
  #10  din<=2;
  #10  din<=3;
  #50 $finish;
end
endmodule

module dut (clk, din, ctrl, mclk, addr, dout);
  input clk, ctrl;
  input [3:0] din;
  output mclk;
  output[3:0] addr, dout;
  reg gclk, mclk;
  reg[3:0] addr, dout;
  always @(clk or ctrl) begin
    gclk = ~(clk | ctrl);
    mclk = ~(clk | ctrl);
  end
  always @ (posedge gclk) begin
    dout <= din;
    addr <= din;
  end
endmodule

module mem (clk, din, dout, adr);
  input clk;
  input [3:0] din, adr;
  output [3:0] dout;
  reg [3:0] dout;
  reg [3:0] mem [0:7];
  wire we = 1;
  always @ (posedge clk) begin
    if (we) begin
      mem[adr] <= din;
      dout <= din;
    end
    else dout <= mem[adr];
  end
endmodule
```

xc poclk rm

Removes a signal with the specified *POCLK_ID* from the list of POCLKs.

The syntax for this command is:

```
xc poclk rm, POCLK_ID
```

The signal will no longer be updated earlier than other outputs driven by the emulator hardware.

Note:

- Use the `xc poclk` command to view the current POCLK signals and to obtain their *POCLK_ID*.
- Use the `xc poclk add` command to add a DUV or hardware model output or inout port to the list of POCLK signals.

xc potraceoff

Turns off tracing of the primary outputs of a DUV module.

The syntax for this command is:

```
xc potraceoff
```

xc potraceon

Turns on tracing of the primary outputs of a DUV module.

The syntax for this command is:

`xc potraceon [level]`

level

Specifies the level for tracing the primary outputs. The accepted levels are:

-
- 1 Indicates that the trace data should report only the primary output events.
This is the default level.
 - 2 Indicates that the trace data should also report the DPI argument updates.
-

xc proff

Stops and reports the `xc` run-time profiling during the hardware run. If this command is not specified, the `xc` profiler will automatically finish when the run-time process finishes.

When the `xc` profile finishes, it will display a report as illustrated below:

```
--- xc Profile: (%)  
    97.29  emu  
    1.96  IUS  
    0.58  decode  
    0.16  xc_runtime
```

Here,

- The `emu` time is percentage of time spent in hardware evaluation. This includes sending `dma` input to the hardware, initiate hardware evaluation, polling the hardware for stop conditions, and sending `dma` output back.
- The `IUS` time is percentage of time spent in IES space during the profile time. This includes the simulation events in testbench modules, the `tbcall` tasks triggered by hardware `tbcalls`, and any overhead imposed by VPI/VHPI/DPI calls to the IES space.
- The `decode` time is the percentage of time spent in decoding the hardware output changes. If there are many DUV outputs that change very frequently, this time might be significant.
- The `xc_runtime` is the percentage of time spent in the rest of the `xc` run-time software.

The syntax for this command is:

```
xc proff
```

xc profon

Starts the `xc` run-time profiling during the hardware run. This command can be specified anytime before executing the `xc on -run` or `xc on -tbrun` command.

The `xc` profiler will start only after the design states have swapped into the hardware.

The syntax for this command is:

```
xc profon
```

xc showcpi

Displays all primary input signals in the DUV.

The syntax for this command is:

```
xc showcpi
```

xc showcpo

Displays all primary output signals in the DUV.

The syntax for this command is:

```
xc showcpo
```

xc showfrc

Displays all DUV signals that are precompiled for force or release.

The syntax for this command is:

```
xc showfrc
```

xc signal

Enables force, release, set, and get operations on the DUV signals, without needing to precompile the DUV signals for force or release when using *irun* run-time tool.



Caution

The xc signal command is available only in the irun tool. In the xeDebug tool, use the force, release, deposit, and value commands instead of the xc signal command. Executing the xc signal command in the xeDebug tool results in a syntax error.

The syntax for this command is:

```
xc signal -force <signal_name> <value> |  
        -release <signal_name> |  
        -set <signal_name> <value> |  
        -get <signal_name>
```

-force <signal_name> <value>

Forces the *<value>* on the specified signal until explicitly released by a *xc signal -release <signal>* command.

-release <signal_name>

Releases a previous force on the specified signal. The value of the specified signal is recalculated at the next cycle.

-set <signal_name> <value>

An equivalent to *xc signal -force* followed by *xc signal -release*.

Note: When accessing DUV signals with memories, the signal -set command works only on a whole memory word.

-get <signal_name>

Gets the current value of the specified signal.

xc sigTrace

Performs the following signal trace operations in the hardware mode when *irun* run-time tool is used:

- Adds or removes signals to trace
- Uploads trace results to a trace file
- Clears the contents of the trace file



The `xc sigTrace` command is available only in the `irun` tool. In the `xeDebug` tool, use the database and probe commands instead of the `xc sigTrace` command. Executing the `xc sigTrace` command in the `xeDebug` tool results in a syntax error.

The syntax for this command is:

```
xc sigTrace [-add <signal>... |  
                -addinst [-depth <n>] <instance> ... |  
                -addinstpin <instance> ... |  
                -addcone [-depth <n>] <signal> ... |  
                -addpath [-depth <n>] <net1> <net2> |  
                -rm <signal>... | * |  
                -import <filename> | -save [-qel] <filename> |  
                -upload | -clear  
                -regexp <pattern> | -glob <pattern> |  
                -outfile <file_format> <basename> |  
                -assertion -add | -rm <assertion_name> |  
                -assertion -addfanin | -rmfanin <assertion_name> |  
                -assertion -addInst | -rmInst <inst_name> |  
                -assertion -addinstfanin | -rminstfanin <inst_name> |  
                -assertion -addall | -assertion -rm * |  
                -prepareoffline]
```

-add <signal>...

Adds one or more signals to the trace signal list.

-addinst [-depth <n>] <instance> ...

Adds all the nets in the database hierarchy of the given instances to the trace signal list.

The `-depth` option is used to specify a maximum depth to restrict the search in the database, where `<n>` specifies the maximum levels of hierarchy. By default one hierarchy level is searched. A depth of zero means the whole hierarchy of an instance is searched.

-addinstpin <instance> ...

Adds all input/outputs of the specified `<instance>` and all instances within it.

-addcone [-depth <n>] <signal> ...

Adds all signals in the given nets input cone of logic to the trace signal list.

-addpath [-depth <n>] <net1> <net2>

Traces back from `<net1>` to `<net2>` and adds all nets on the path. The `-depth <n>` parameter indicates the number of levels that are traced back. The default value is 10 levels.

-rm <signal>... | *

Removes the specified signal(s) from the trace signal list. The `-rm * command removes all signals from the trace signal list.`

-import <filename>

Brings in signals from the specified file. This file can be:

- `probe.upld`, previously saved using the `sigTrace -save` command or the Save button in the GUI.
- `probes.rc`, saved from the waveform when the signals were added.

-save [-qel] <filename>

Saves signals to a specified file as a flat list.

If the `-qel` option is included in the command, the probe list is saved in the form of a QEL script.

-upload

Uploads trace results from the emulator into a trace file. Also calculates virtual probes and writes them to the trace result file `trace.out` or `<vector>.out` file. In offline debug mode, only virtual probes are calculated and written to the resultant files, `trace.out` or `<vector>.out`.

-clear

Removes the `trace.out` file in STB mode, so that the next time, a new file is created.

The `sigTrace -clear` command clears only trace files, but not the emulation cycle count.

-regexp <pattern> | -glob <pattern>

Specifies the search pattern for the hierarchical path component.

Using the `-regexp <pattern>` option displays a list of all nets in the trace upload list, where the signal names match the specified pattern using regular expression matching rules.

Using the `-glob <pattern>` option displays a list of all nets in the trace upload list with names that match the specified pattern using the *glob* expression matching rules.

-outfile <file_format> <basename>

Sets the name and format of the output to be used. The output will be saved in one or more files or a directory with format and name as determined by `<file_format>`. The value of `<file_format>` can be one of the following:

- `-fsdb`: Generates a file with the name `<basename>.fsdb` in FSDB format.
- `-sst2`: Generates a directory with the name `<basename>` containing an SST2 (or SHM) database.

-assertion -add | -rm <assertion_name>

Specifies the assertion to be added or removed from the tracelist. The *<assertion_name>* specifies the full hierarchical name of the assertion.

-assertion -addfanin | -rmfanin <assertion_name>

Specifies that the fanin signals for the assertion should be added or removed from the tracelist. The *<assertion_name>* specifies the full hierarchical name of the assertion.

-assertion -addInst | -rmInst <inst_name>

Specifies that all the assertions in the specified instance /scope should be added or removed from the tracelist. This does not affect the fanin signals.

-assertion -addinstfanin | -rminstfanin <inst_name>

Specifies that the fanin signals for the assertions in the specified instance /scope should be added or removed from the tracelist.

-assertion -addall

Specifies that all the assertions should be added to the tracelist.

-assertion -rm *

Specifies that all the assertions should be removed from the tracelist.

-prepareoffline

Uploads the trace results from the emulator into a *.phy file, which can be used later in offline mode, but cannot be read directly by the waveform browsers. Since this command does not process the data into SHM or FSDB format, it is faster than the -upload option. This option is supported in the FullVision mode, but not in DYNP (Dynamic Probes) mode. Use this option in online mode.

xc status

Checks the status of the emulator hardware, such as current usage and the current debug mode - emulation or SA.

The syntax for this command is:

```
xc status
```

xc stop

Swaps the design into software simulation mode and stops the running of the design at the end of the current timestep. Displays the CLI prompt after next `run` command is executed.

The syntax for this command is:

```
xc stop
```

xc swapInMem

Specifies whether the contents of the memories in DUT are to be swapped into the hardware at the next swap-in time.

The syntax for this command is:

```
xc swapInMem [0|1]
```

0

Specifies that the DUT memories are not to be swapped into the hardware.

1

Specifies that the DUT memories are to be swapped into the hardware. This is set by default.

xc verbose

Turns on the mechanism to display detailed messages in the `xc` run time.

The syntax for this command is:

```
xc verbose [<level>]
```

level

Specifies that the level up to which the messages should be detailed. The following levels are accepted:

- 0: Indicates the messages should not be displayed.
- 1: Indicates the basic level of messages should be displayed. This is the default level.
- 2: Indicates that detailed messages should be displayed.

xc wait

Tells the simulator to wait for the emulator hardware to be available.

The syntax for this command is:

```
xc wait [<tries>] [status]
```

tries

Specifies a positive integer indicating the number of tries for which the simulator should wait for the emulator; otherwise, the simulator will wait forever (unless you enter **Ctrl-C** to cancel the wait).

status

Specifies a Tcl error status to be displayed when the emulator is not still not available after the specified number of tries.

Example

To wait for the emulator until it's available, execute the following command:

```
xc wait
```

To wait for the emulator and try 3600 times before giving up and want Tcl error code 99 to be displayed after giving up, execute the following command:

```
xc wait 3600, 99
```

xc xreg

Randomizes all registers or only the registers that had X or Z values before swap-in.

The syntax for this command is:

```
xc xreg -save |  
  -scope <file_scope> |  
  -rand <seed> |  
  [-all0 | -all1] |  
  -verbose
```

-save

Saves the list of registers with X or Z values.

Note:

- ❑ The `xc xreg -save` command can be used only in the software mode.
- ❑ Execute the `xc xreg verbose` command before `xc xreg -save` to view the registers that are saved by the `-save` option.

-scope <file_scope>

Specifies the module instance scope of registers to be set. You can also specify multiple instances in a file and provide the file to the `-scope` option.

For example, `-scope top` specifies that registers under the `top` hierarchy should be randomized. You can also specify `-scope scopeFile` where `scopeFile` is a text file containing lines in the following format:

```
include top.dut  
exclude top.dut.clock_gen  
exclude top.dut.reset_module
```

With the above file, all the registers in the `top.dut` hierarchy except the ones under `top.dut.clock_gen` and `top.dut.reset_module` are randomized.

-rand <seed>

You must also specify `-scope <file_scope>` when you do `xc -rand <seed>`. The random `<seed>` value can be any integer number. For registers in the scopes specified by

UXE Command Reference Manual

SA Run-time and Debug Commands

the `-scope` option, the `-rand` option randomizes only the registers that were saved by a prior `xc -save` command in the same simulation session.

The `xc xreg -rand` command does not require a prior save. All registers are randomized if the `xc xreg -save` command was not used before swap-in. However, randomizing all registers might increase the probability of changing a critical part of the design, such as reset circuitry getting changed. So, you must exclude the critical parts of the design in the file specified with the `-scope` option.

Note: The `xc xreg -rand <seed>` command can be used only in the hardware mode.

-all0 | -all1

Sets all registers in `<file_scope>` to 0 or 1. You must also specify `-scope <file_scope>`. When used with `-scope`, only the registers that were saved by `xc -save` are changed.

-verbose

Prints out registers being changed by `-rand <seed>`, `-all0`, or `-all1`.

xrun

Advances the hardware time when using the *irun* run-time tool in autorun mode.



The `xrun` command is available only in the `irun` tool. In the `xeDebug` tool, to advance time in autorun mode, use the `run` command instead of the `xrun` command. Executing the `xrun` command in the `xeDebug` tool results in a syntax error.

Specifying the `xrun` command without a time argument runs the hardware indefinitely. For example, executing the `xrun 100ns` command advances the hardware time by `100ns` and then stops.

The syntax for this command is:

```
xrun [<time>]
```

ixcc

The `ixcc` command generates a text-based cumulative coverage counter report file, and/or a binary cumulative coverage counter file according to the specified coverage counter file(s).

The syntax for this command is:

```
ixcc [{-inputDir <dir_name>}]
[{-input <test_name>} | {-inputAll}]
[{-outputDir <dir_name>}]
[{-output <test_name>}]
[{-format <format_name>}]
[-summary]
[-listInst]
[-matchInst]
[-help]
```

Following are the options of the `ixcc` command:

-inputDir

Specifies the directory to look for test files. This option can be specified multiple times if multiple source directories exist. When this option is not specified, the current directory, or “.” is used by default.

The value passed as an argument with the `-inputDir` option must match the `+ixccWorkDir` specified at run time.

-input

Specifies the name of the test. This name must match the test name specified when the coverage test was created or an output test name generated by the `ixcc` utility. If multiple input files are specified, the output is merged into a single file. When multiple input directories are specified, the directories are searched in the specified order, and the first one that matches the test name is used.

The value passed as an argument with the `-input` option must match the `+ixccTest` option specified at run time. The model file is automatically loaded on the basis of the information stored with the test file.

-inputAll

Specifies that all test name files under the specified input directory should be regarded as inputs. When the `inputAll` option is used, the `-input` option is not considered.

-outputDir

Used as an optional switch to specify a directory for output files. When not specified, the first specified `<outputDir>` is used. Only one `<outputDir>` is allowed. The *ixcc* utility creates the `outputDir`, if needed.

The `-outputDir` option only applies when a file-based output format is used.

-output

Specifies the name of the output test when file-based format is used. The output test file must not exist, or an error message is issued.

-format <format_name>

Specifies the output format, `<format_name>`, as one of the following:

- `cdf` - Creates a file-based output in the CDF format that can be read by the *IXCC* utility. This option is used for merging data sets.
- `stdout` - Send report to standard out (default).

-summary

Sends a summary coverage table to `stdout`.

The summary table includes a one-line report for each hierarchical instance that contains code-coverage data. Instances without coverage items are printed only if at least one of their sub-instances contains coverage items.

-listInst

Creates a text file that contains the full path to each instance that has coverage data. No coverage counts are generated.

-matchInst

Limits the coverage reporting to the instances that match the full paths that are listed in the file. Default behavior is to match all instances in the design. This only applies for the stdout format.

Note: Even if the `-matchInst` option is included, the summary report always contains the block and toggle information for the entire design.

-help

Prints the usage of the `ixcc` command.

Emulating with XEL

This chapter describes the usage of Tcl scripts for emulating designs. We will use here the term Tcl script to refer to a script that contains a mixture of XEL commands and regular Tcl commands. Tcl scripts provide a way to automate and combine several emulation steps into one command, which would otherwise be run interactively using various options in the Palladium XP user menus. The chapter also illustrates the use of Tcl scripts by giving sample scripts for each function that is performed in the design verification process. This identifies the verification process stages in which each group of commands is used.

The topics discussed in this chapter are as follows.

- “[Using the Scripts](#)” on page 1160
 - “[Running a Script in Batch Mode \(non-interactive\)](#)” on page 1160
 - “[Sourcing a Script File](#)” on page 1160
 - “[Compiler and Debugger Initialization Scripts](#)” on page 1160
- “[Creating Tcl Scripts](#)” on page 1161
- “[Tcl Script Examples](#)” on page 1164
 - “[Compile Scripts](#)” on page 1164
 - “[Importing a Design](#)” on page 1166
 - “[Synthesis For Emulation](#)” on page 1167
 - “[Emulating in Vector Debug Mode](#)” on page 1167
 - “[Emulating in Logic Analyzer Mode](#)” on page 1168
 - “[Emulating in STB Mode](#)” on page 1168
 - “[Assigning IP Instances](#)” on page 1169

Using the Scripts

Scripts are written into files which are later read by xeDebug or xeCompile. You can create script files with a text editor, or generate them by the software. For example, you might request during a debug session to save the run environment as a script file. Scripts can be read and executed by the compiler or debugger in one of several ways as described below:

Running a Script in Batch Mode (non-interactive)

You can request either xeCompile or xeDebug to execute a script file and exit when finished, by specifying the file name as a command line argument to the O/S shell as in the following example:

```
xeDebug myscript.tcl
```

Sourcing a Script File

During execution of xeCompile or xeDebug, in either GUI or command line mode, you can request to execute a script file using the `source` command, which is a generic Tcl command, as in the following example of running under the debugger:

```
XE> source myscript.tcl
```

When running the compiler, the prompt is `XEC>` instead of `XE>`

Compiler and Debugger Initialization Scripts

Tcl scripts can be called automatically during the executable's initialization phase, whether they are run in GUI mode, batch mode, or interactive command line mode.

If a file named `.xerc` exists in either the home directory or in the directory from which xeDebug was invoked, this file will be executed as the first step by xeDebug. If both directories include the `.xerc` file, both files will be executed, starting from the home directory.

Similarly, if a file named `.xecrc` exists in either the home directory or in the directory from which xeCompile was invoked, this file will be executed as the first step by xeCompile. If both directories include the `.xecrc` file, both files will be executed, starting from the home directory.

The `.xerc` file cannot request to execute any XEL command whose execution depends on availability of resources, such as an allocated emulator or an opened design database. This practically includes most of the XEL commands that are used by the debugger. The reason

is that such resources are granted to xeDebug only after successful execution of the `configPM` command. On the other hand, the `.xerc` file might contain definitions of Tcl procedures containing XEL commands which you can call later. For automatic execution of XEL commands after such resources become available at run time, you can use a `.rtxerc` file.

If a file with the name `.rtxerc` exists in either the home directory or in the directory from which `xeDebug` was invoked, this file will be executed during run time after successful configuration of the emulator (on a successful conclusion of the `configPM` command). If both directories include a `.rtxerc` file, both files will be executed, starting from the home directory.

The `.xerc` file is therefore useful only for generic setup or definition of custom commands (through the Tcl `proc` command). The `.rtxerc` file is useful for emulation-specific initialization and setup during run time that can only be executed after the `configPM` command.

At the time of invoking either `xeCompile` or `xeDebug`, the `-init` option can be used to call additional initialization files. The syntax for using these additional files is:

```
xeCompile [-init <file1>] [<batchmode_script>]
```

Creating Tcl Scripts

To create a Tcl script, simply type the commands for each function in a file and save the file with a suitable name. XEL does not impose any restrictions for naming the Tcl script files. However, it is a good idea to specify the name of the file as the function to which the commands are related, followed by a meaningful extension such as `.tcl`. The compiler and debugger make use of other extensions for Tcl script files depending on their purpose. For example, they generate each session a script file with an extension of `.key` that can be used to reproduce the run. This becomes useful if you are not familiar with the exact syntax of the XEL commands. Run a compile or debug session from the `xeCompile` or `xeDebug` GUI tool. All the commands executed during the session are stored in a file called `xeCompile.key` or `xeDebug.key` in the emulation design directory (the directory from where the session has been started).

Note: To stop execution of a Tcl script that is running, use the standard UNIX `CTRL-C` keystroke combination. If it does not work, use the `CTRL-\` (backslash) keystroke combination.

Debugging Tcl Scripts

xeDebug or xeCompile scripts might contain TCL programming constructs such as loops, if statements, procedure calls, and more. Complete description of the TCL language is available on the Internet or printed literature, and is beyond the scope of this manual.

When things go wrong due to programming errors in your script, you can either insert puts statements at strategic points, or use a GUI driven TCL debugger such as the one provided with the *Tcl Dev Kit (TDK)* from *ActiveState*. This debugger provides a software debugging environment in which you can insert breakpoints, single step the script, inspect TCL variables, procedure calling stack, and more.

Below is an example how to install and use the TDK debugger with xeDebug. The procedures outlined below were tested on TclDevKit version 5.3. Other releases might use slightly different steps.

Keep in mind that Cadence does not maintain TDK and cannot guarantee its operation or its compatibility.

Installation of the Tcl Dev Kit (TDK):

Download and install the Tcl Dev Kit from the ActiveState Web site at:

www.activestate.com

Before installing TDK you need to have a valid installation of *ActiveTcl* (which you can also download from the ActiveState web site). Make sure that when you install TDK, your path includes also the path to the executable binaries of ActiveTcl. For example, if running with C shell, execute the following before starting installation of TDK.

```
setenv PATH <ActiveTcl-installation-path>/bin:$PATH
```

Attaching the TDK debugger to an xeDebug process - basic procedure

This basic procedure is useful if you can source the TCL script that you want to debug from the xeDebug command line prompt.

Bring up the TDK debugger by executing the following line from the Linux/Unix shell:

```
<TDK-INSTALL-PATH>/bin/tcldebugger
```

<TDK-INSTALL-PATH> is the directory path where the TDK package was installed.

In the TDK debugger GUI select *New Project* under the *File* pull down menu.

Under *Application -> Debugging* Type select *Remote Debugging*.

Click *OK*.

Make sure that the port number is unique and not being used by a different program running on the same workstation. If not sure, fill in a different unique integer number.

Bring up `xeDebug`.

Attach the TDK debugger to the `xeDebug` process by typing the following commands at the `xeDebug` command prompt:

```
source <TDK-INSTALL-PATH>/lib/tcldebugger_attach/attach.tcl  
debugger_init <DEBUG-HOST> <PORT-NUMBER>
```

`<DEBUG-HOST>` is the host name (or its IP address) on which the TDK debugger is running. If `xeDebug` and the TDK debugger are running on the same host, then `<DEBUG-HOST>` can be the address `127.0.0.1`.

`<PORT-NUMBER>` is the same integer value that you specified as port number in the TDK debugger's GUI (for Remote Debugging under *File -> Application -> Debugging*).

Assuming no failure condition, `xeDebug` is now under the control of the TDK debugger.

Now you can source the TCL script that you want to debug from the `xeDebug` command line prompt.

Automatic attachment of TDK debugger at `xeDebug` initialization

You can attach the TDK debugger to `xeDebug` automatically when `xeDebug` starts execution by putting the statements to source the `attach.tcl` script and call `debugger_init` directly inside the `.xerc` initialization file (in your home directory). Here is an example of TCL code snippet which you can insert inside your `.xerc` file:

```
set debug_host <DEBUG-HOST>  
set debug_port <PORT-NUMBER>  
set debug_init_script <TDK-INSTALL-PATH>/lib/tcldebugger_attach/attach.tcl  
if [file exists $debug_init_script] {  
    source $debug_init_script  
    if [debugger_init $debug_host $debug_port] {  
        puts "**** Connected successfully to TDK Debugger ****"  
    }  
}
```

This code can be left in your `.xerc` file whether you are using the TDK debugger or not because it will have no effect unless the TDK debugger is already running and waiting for connections. With this code inside the `.xerc` file, all you have to do in order to debug `xeDebug` with the TDK debugger is to bring up the TDK debugger and put it in remote

debugging mode before you bring up xeDebug. This is especially useful if xeDebug is by itself called from Unix shell scripts which you do not want to modify.

If the TDK debugger is attached successfully to xeDebug, then the above code will print out the message: **** Connected successfully to TDK Debugger ****

A few hints:

- You can make the TDK debugger always start up with your specified port number in remote debugging mode by setting it as default through the "Default Project Settings" under the File Menu (Close current project if open, before setting the default).
- The arguments to `tcldebugger` can be omitted if you are using the default setting of the TDK debugger (host=127.0.0.1, port=2576).

Tcl Script Examples

Compile Scripts

The scripts defined in this section compile a design for In-Circuit Emulation and Vector Debug modes.

Sample scripts for compiling designs are available in the `<install_dir>/share/uxe/gift` directory.

Generic Compile (VD Mode)

The following script is a sample of a generic compile for Vector Debug mode.

```
hdlLang verilog          // set language to Verilog
hdlInputFile this_design.v //define this_design as a Verilog file
hdlInputFile dram.v       //define Verilog memory model
hdlImport             //read in files for synthesis
hdlSynthesize -memory my_design //synthesize the design using memory synthesis
design TOP mydesign      //open the design named my_design with top
                           module as TOP
userData -load userdata   //load in the userdata information
emulatorConfiguration -add {host host_name} {boards 0.4+0.5}
                           // use the configuration file from the emulator named
                           host_name and only use domains 4 and 5 from board 0
compilerOption -add {mode vd}    //set up for vector debug mode compile
precompile           //check to make sure all compile steps are completed
compile            //compile the design
```

UXE Command Reference Manual

Emulating with XEL

```
quit          //end script
```

Compile Script for In-Circuit Emulation (ICE)

```
refLib qtref
importOption {mode incremental}
importOption {library LIB}
netlistFile {verilog counter_ice.v}
designImport
# Command below is REQUIRED
design LIB COUNTER
# Command below is REQUIRED
emulatorConfiguration -add { host lifeboat }
# Command below is REQUIRED
compilerOption -add { mode ice }
# Specify information for automatic clock generation
# Commands below are OPTIONAL
clockSource -add CK
clockOption -add {technology CAKE 2}
# Specify probes (if using non-full-vision mode)
# Commands below are OPTIONAL
compilerOption -add {visionMode DYNP}
probe -add SYNC5__NXOR3
# Specify keepNets as needed
# Command below is OPTIONAL
keepNet -add SYNC5__INP3
# Command below is REQUIRED
precompile
# Commands below are REQUIRED for connection to a target
cableConnection -add {my_tib_1_1.MTIB-88 CB1 J1}
terminalAssign -add \
{my_tib_1_1 MTIB-88 P2 1 TERM CKOUT} \
{my_tib_1_1 MTIB-88 P2 3 TERM RESET} \
{my_tib_1_1 MTIB-88 P2 5 TERM AQN} \
{my_tib_1_1 MTIB-88 P2 7 TERM BQN} \
{my_tib_1_1 MTIB-88 P2 9 TERM CQN} \
{my_tib_1_1 MTIB-88 P2 11 TERM DQN} \
{my_tib_1_1 MTIB-88 P2 13 TERM EQN}
# The delayBox and terminalTiming commands below are OPTIONAL.
# However, if clock and non-clock outputs to the target can
# change in the same FCLK cycle, these commands
```

```
# might be required to get correct function.  
# The following command creates a phantom signal named  
# after_clock that will be scheduled at least 10 steps  
# after CKOUT  
delayBox -add {after_clock 10 CKOUT}  
# outputs are scheduled at or after the step where after_clock  
# is scheduled. In general for a unidirectional non-clock output you  
# should specify  
#   terminalTiming -add {<signal_name> after_clock}  
# and for a bidirectional non-clock signal you should specify  
#   terminalTiming -add {<signal_name> after_clock * after_clock}  
# to make sure that non-clock outputs change later than clock outputs  
# to the target.  
terminalTiming -add \  
  {AQN after_clock} \  
  {BQN after_clock} \  
  {CQN after_clock} \  
  {DQN after_clock} \  
  {EQN after_clock}  
# Command below is REQUIRED  
icePrepare  
# Command below is REQUIRED  
compile
```

Importing a Design

The following Tcl script imports a design that includes memory instances, and configures memory before importing the netlists. Typically, Palladium configures memories after importing.

```
# Import design into 'counter' library  
importOption {library counter}  
# Set reference library search order for cell definitions  
reflib lsi_lca100k qtref generic  
# Select netlist for import  
netlistFile {netlists countermem.verilog}  
# Save preceding import configuration in 'myconfig' file  
impDumpConfig myconfig  
# Start importing the design  
designImport
```

The following Tcl script sets import options, saves the option settings for future design import sessions, and imports a design using the new option settings:

```
# Import design into 'counter' library
importOption {library counter}

# Set reference library search order for cell definitions
reflib lsi_lca100k qtref generic

# Select netlist file for import
netlistFile {netlists}

# Save preceding import configuration in 'myconfig' file
impDumpConfig myconfig

# Start importing the design
designImport
```

Synthesis For Emulation

The following Tcl script imports a VHDL design, and synthesizes a gate-level netlist from the **RTL description**:

```
# Specify HDL design language.
hdlLang vhdl

# Make mylib the current working library.
hdlWorkPath mylib

# Select VHDL files, and import them.
hdlInputFile top.vhd f1.vhd f2.vhd
hdlInputFile f3.vhd
hdlImport

# Create 'netlist1' VHDL output file for other tools.
hdlOutputFile -add -f vhdl netlist1.vhdl

# Synthesize 'top' module and all hierarchy below it.
hdlSynthesize top
```

Emulating in Vector Debug Mode

The following Tcl script runs the design named COUNTER in Vector Debug mode, using the machine named egbert as the emulation host:

```
# Set directory of design database and start Debugger
debug ~/counterdir
# Select emulation host and download design
host egbert
download
# Configure for (select emulation mode)
configPM -vd
# Compile the stimulus file second.cbc (given in CBC format)
vector second.cbc
# run all the vectors that were provided in file second.cbc
run
# Restart from first vector
reset
# define trigger conditions using SDL
sdl -enable
sdl -setFile trigcond.tdf
```

UXE Command Reference Manual

Emulating with XEL

```
# run first 1000 vectors or until SDL detects a trigger condition
run 1000
quit
```

Emulating in Logic Analyzer Mode

The following Tcl script emulates the design in Logic Analyzer mode. It downloads a design, defines a trigger file, and begins execution.

```
# set path to design database
debug /path/to/design/database

# set name of emulation host
host emulation_hostname

# download the design into the emulator
download

# configure instrumentation to work in LA mode
configPM -la

# define trigger conditions through the use of SDL
sdl -setfile triggerFile.tdf

# enable the use of SDL for triggering
sdl -enable

# set trigger position at 80% of the trace memory size
xeset triggerPos 80

# define trace memory size as 10000 FCLK cycles
xeset traceMemSize 10000

# start emulator clocks and LA instrumentation (probing and triggering)
run

# wait until trigger, or until a timeout of 60 seconds
waitWhileBusy 60

# stop probe data capture if run does not trigger after waiting for 60 seconds
stop

# upload trace results from the emulator's trace memory to the disk.
database -upload
```

Emulating in STB Mode

The following shows a printout from an interactive script based session in STB mode. In STB mode the trigger causes the emulation clocks to stop (break).

Break on Trigger Using SDL File To Define Trigger Condition

```
# set path to design database
XE> debug ~/mySTBtest
# set name of emulation host
XE> host emulation_hostname
# download the design into the emulator
XE> download
# configure instrumentation to work in STB mode
XE> configPM -stb
# set up trigger condition using a SDL file
XE> sdl -enable
XE> sdl -setFile trigger.tdf
XE> run 1000000
# trigger condition at cycle 50000. Emulator stopped with a 2 cycle delay
XE> getCycleNum
50002
# set a simple trigger condition without SDL file
XE> sdl -disable
XE> sdl -expression {cntout[15:0] == 765}
# set up emulator to stop without delay (emulator runs slower!)
XE> stopDelay 0
XE> run
# trigger happened after 28 cycles
XE> getCycleNum
50030
# report equivalent simulation time in nanoseconds
XE> getTime ns
400240 ns
XE> database -upload
XE> probe -waveform
```

Assigning IP Instances

The following Tcl script compiles the design using caCell for IP assignment.

```
importOption {mode full}
importOption {library test1}
refLib qtref generic
netlistFile {verilog test1.v}
designImport
design test1 top
emulatorConfiguration -add {host myEmulator}
compilerOption -add {mode ice}
caCell -add {test1 C1 123}
precompile -removeSourceless
icePrepare
compile
```

UXE Command Reference Manual
Emulating with XEL

Deprecated Commands and Variables

This chapter describes the following obsoleted XEL commands and variables that appear here only for backward compatibility:

Obsolete	Type	Replaced with
<u>attr</u>	Database Command	none
<u>cell</u>	Database Command	none
<u>closelib</u>	Database Command	none
<u>compileFindAll</u>	Compile-time Command in ICE Mode	<u>compileFind -all</u>
<u>compileFindAllMemUti</u>	Compile-time Command in ICE Mode	<u>compileFind -mem</u>
<u>compileFindBest</u>	Compile-time Command in ICE Mode	<u>compileFind -best</u>
<u>compileFindBestOptimization</u>	Compile-time Command in ICE Mode	<u>compileFind -opt</u>
<u>compileFindFirst</u>	Compile-time Command in ICE Mode	<u>compileFind -first</u>
<u>cosimAcs</u>	User Data Command	none
<u>createUserSession</u>	Run-time Command	<u>debug <dbPath> -session <session_name></u>
<u>criticalAccess</u>	User Data Command	none
<u>dontAcqOnEvent</u>	Run-time Command	global no_acquire defined inside the SDL program
<u>eventSignal</u>	User Data Command	none
<u>group</u>	Run-time Command	<u>symbol</u> , <u>deposit</u> , <u>force</u> , <u>probe</u> , and <u>value</u>

UXE Command Reference Manual

Deprecated Commands and Variables

Obsoleted	Type	Replaced with
<u>inst</u>	Database Command	none
<u>lib</u>	Database Command	none
<u>net</u>	Database Command	none
<u>parallelCompileFind</u>	Compile-time	<u>compileFind -lsf</u>
<u>Best</u>	Command in ICE Mode	
<u>parallelCompileFind</u>	Compile-time	<u>compileFind -opt -lsf</u>
<u>BestOptimization</u>	Command in ICE Mode	
<u>primaryClockSource</u>	User Data Command	none
<u>qtCounter1</u>	Tcl Variable	Values of Tcl variables are directly accessible inside the SDL program.
<u>qtCounter2</u>	Tcl Variable	Values of Tcl variables are directly accessible inside the SDL program.
<u>pdFvHost</u>	Tcl Variable	Run-time Parameter: <u>mtHost</u>
<u>qtDatabase</u>	Tcl Variable	Run-time Parameter: <u>database</u>
<u>qtDBPath</u>	Tcl Variable	Run-time Parameter: <u>dbPath</u>
<u>qtDesign</u>	Tcl Variable	Run-time Parameter: <u>design</u>
<u>qtEndOfVector</u>	Tcl Variable	Run-time Parameter: <u>endOfVector</u>
<u>qtHost</u>	Tcl Variable	Run-time Parameter: <u>host</u>
<u>qtMaxMismatches</u>	Tcl Variable	Run-time Parameter: <u>maxMismatches</u>
<u>qtPostTriggerSamples</u>	Tcl Variable	Run-time Parameter: <u>postTriggerSamples</u>
<u>qtSignalRadix</u>	Tcl Variable	Run-time Parameter: <u>signalRadix</u>
<u>qtSuspendEnable</u>	Tcl Variable	Run-time Parameter: <u>suspendEnable</u>
<u>qtTimeUnit</u>	Tcl Variable	Run-time Parameter: <u>timeUnit</u>
<u>qtTraceMemMax</u>	Tcl Variable	Run-time Parameter: <u>traceMemMax</u>
<u>qtTraceMemSize</u>	Tcl Variable	Run-time Parameter: <u>traceMemSize</u>
<u>qtTriggerPos</u>	Tcl Variable	Run-time Parameter: <u>triggerPos</u>
<u>qtVector</u>	Tcl Variable	Run-time Parameter: <u>vector</u>
<u>qtVersion</u>	Tcl Variable	Run-time Parameter: <u>version</u>

UXE Command Reference Manual

Deprecated Commands and Variables

Obsoleted	Type	Replaced with
<u>qtTriggerByState</u>	Tcl Variable	<u>sdl -enable [-now]</u> and <u>sdl -disable [-now]</u>
<u>qtTriggerFile</u>	Tcl Variable	<u>sdl -setFile [<sdl_filename>]</u>
<u>-force</u> option of the <code>readCPFFile</code> command.	Compile-time Command in ICE Mode	none
<u>resetCycleNum</u>	Run-time Command	<u>resetTime</u>
<u>restoreState</u>	Run-time Command	<u>restart</u>
<u>runtimeCLOCKconf</u>	Run-time Command	<u>clockConfig</u>
<u>savelib</u>	Database Command	none
<u>saveState</u>	Run-time Command	<u>save</u>
<u>signal</u>	Run-time Command	<u>deposit</u> (replaces the <code>signal -set</code> command) <u>force</u> (replaces the <code>signal -force</code> command) <u>release</u> (replaces the <code>signal -release</code> command) <u>value</u> (replaces the <code>signal -get</code> command)
<u>sigTrace</u>	Run-time Command	<u>probe</u> and <u>dpa</u>
<u>stimulusWaveform</u>	Run-time Command	none
<u>stopDelay</u>	Run-time Command	Run-time Parameter: <u>stopDelay</u>
<u>term</u>	Database Command	none
<u>termInst</u>	Database Command	none
<u>traceWaveform</u>	Run-time Command	none
<u>trEvent</u>	Run-time Command	Values of Tcl variables are directly accessible inside the SDL program using the syntax <code>\$variable</code>
<u>verifyTrigFile</u>	Run-time Command	<u>sdl -verify [<filename>]</u>

UXE Command Reference Manual

Deprecated Commands and Variables

Obsoleted	Type	Replaced with
<u>xc uaPollLimit</u>	SA Run-time Command	<p>Use the <code>uaPollAbort</code> file. Execute the following command from another xterm to abort the emulation hang:</p> <pre>Linux> touch <run_dir>/tmp/uaPollAbort</pre> <p>For information on how to control the emulation hang between two <code>tbsync</code> events, refer to the <i>Controlling Emulation Hang in SA Mode</i> section in the UXE User Guide.</p>

attr

Returns the current value of a selected attribute for a specified database object.

Before using this command, execute `design` to specify the design library and cell name. This command returns the current value of the specified attribute for the selected database object.

The syntax for this command is:

```
attr <dbObj> -all | -get [<attrType>]
```

<dbObj>

Specifies the database object identifier. To access an object attribute and its value, you need to use a database object identifier. The object identifier might be created with one of following XEL commands:

- `lib`
This is a library object identifier that is used to find the library attributes.
- `cell`
This is a cell object identifier that is used to find (and add) the cell attributes.
- `inst`
This is an instance object identifier that is used to find the instance attributes.
- `net`
This is a net object identifier that is used to find the net or signal attributes.
- `term`
This is a terminal object identifier that is used to find the terminal attributes.
- `termInst`
This is a terminal instance object (defined in a `termInst` command) is used to find the instance's terminal attributes.

-all

List the current value of all attributes for the given database object identifier, `<dbObj>`.

-get

Lists all attributes/values of the specified *<attrType>* for the current database object, without listing the attribute type. For example:

```
attr libobj -get cells  
CELL0 CELL1 CELL2
```

Without the **-get** parameter, this command returns the attribute type as well as the list of attributes. This list provides more detail about the attributes you are viewing, but it is harder to parse during hierarchical searches. For example:

```
attr libobj cells  
{cells {CELL0 CELL1 CELL2}}
```

<attrType>

Type of attribute you want to view the current value on a given database object. The type of attributes you can view values depends on the type of database object used in the *dbObj* parameter.

You can use the command *attr <dbObj>* to get all the attribute types. For example:

```
attr libobj name objtype viewType \  
openMode hierarchyNumber numberOfRowsInSection cells
```

Attribute Types

Library Objects

For database objects defined using the *lib* command, *<attrType>* can be any of the following:

- name (displays the name of the current library).
- objtype (displays library as the object type).
- viewType (displays the view type of the current library). For an explanation of the different view types, see the *lib* command or type *help lib*.
- openMode (displays r for read mode or w for write mode).
- hierarchyNumber (reports the level of hierarchy where the library is used).
- numberOfRowsInSection (reports the number of cells in the library).

- cells (lists all cells in the library).

Cell Objects

For database objects defined using `cell`, *<attrType>* can be any of the following:

- name (displays the name of the current cell).
- objtype (displays cell as the object type).
- cellType (displays the cell type of the current cell, such as fblock for a functional block).
- libName (displays the name of the library containing the current cell).
- viewType (displays the view type of the library containing the current cell). For an explanation of the different view types, see the `lib` command or type `help lib`.
- signatures (displays the signature string of the object).
- numberOfGate (reports the number of gates in the cell).
- numberOfReg (reports the number of registers in a cell).
- numberOfTbuf (reports the number of tristate buffers in the cell).
- numberOfInst (reports the number of primitive and hierarchical instances in the cell hierarchy).
- numberOfNet (reports the number of nets in the cell).
- nets (lists all nets in the cell).
- numberOfTerm (reports the number of terminals in the cell).
- terms (lists all terminals in the cell).

Instance Objects

For database objects defined using the `inst` command, *<attrType>* can be any of the following:

- name (displays the name of the current instance).
- objtype (displays inst as the object type).
- libName (displays the name of the library containing the cell that defines the current instance).

- viewType (displays the view type of the library containing the cell that defines the current instance). For an explanation of the different view types, see the lib command or type help lib.
- defcellName (displays the name of the cell that defines the current instance).
- cellName (displays the name of the higher-level cell in your design hierarchy that contains the current instance).
- numberOfTermInst (reports the number of terminal instances in the hierarchy of the current instance).
- termInsts (lists all terminal instances in the hierarchy of the current instance).

Net Objects

For database objects defined using the net command, <attrType> can be any of the following:

- name (displays the name of the current net).
- objtype (displays net as the object type).
- type (displays the type of the current net, such as regular, tristate, power, or ground).
- cellName (displays the name of the design cell that contains the net).
- connections (lists all pairs of cells that the current net connects together).

Terminal Objects

For database objects defined using the term command, <attrType> can be any of the following:

- name (displays the name of the current terminal).
- objtype (displays term as the object type).
- type (displays the terminal type of the current terminal, such as data).
- dir (displays the direction of the current terminal, such as input, output, or bidir).
- class (displays signal, probe, component adapter interface, or interface).
- attribute (displays sync or async).
- cellName (displays the name of the cell that contains the current terminal).

- connections (lists all nets to which the current terminal connects).

Terminal Instance Objects

For database objects defined using the `termInst` command, `<attrType>` can be any of the following:

- name (displays the name of the current terminal instance).
- objtype (displays `termInst` as the object type).
- type (displays the type of the current terminal instance, such as data).
- dir (displays the direction of the current terminal instance, such as input, output, or bidir).
- instName (displays the name of the instance that contains the current terminal).
- connections (lists all nets to which the current terminal instance connects).

Examples

```
attr libObj -all
```

Lists the current value of all attributes for the `libObj` library object, as specified in a `lib` command.

```
attr cellObj -get name  
attr cellObj -get libName
```

Displays the cell name of the `cellObj` database object, and the name of the library that contains the cell, as specified in a `cell` command.

```
attr dbObj
```

Returns a list of attribute types for the given `dbObj` identifier.

```
attr dbObj -all
```

Returns all the attribute types and their associated attributes/values for the given `dbObj` identifier.

```
attr dbObj [-get attrType]
```

Returns the attribute/value of the `attrType` for the given `dbObj` identifier.

cell

Creates a database object for a cell in your design.

Before using this command, execute `design` to specify the design library and cell names for your design. Also use `lib` to define a database object for the library containing the cell.

Use this command before executing `inst`, `net`, `term`, and `termInst` commands.

The syntax for this command is:

```
cell <cellobj> <cellName> <libobj>
```

<cellobj>

Database object identifier to assign to the cell.

<cellName>

Name of the cell to define as a database object.

<libobj>

Name of the library database object containing the cell, as defined in the `lib` command.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical r
cell cellobj CLOCK_LAB libobj
```

Creates a database object named `libobj` for the `DESIGNS` library, and then a database object named `cellobj` for the `CLOCK_LAB` cell in that library.

closelib

Closes the current database object for an ASIC or reference library specified in a previous `lib` command. Other XEL commands no longer recognize the database object name, or other database objects (such as `cellobj`) that refer to the closed library.

Before using this command, execute `design` to specify the design library and cell names for your design. Also execute `lib` to define the library object.

The syntax for this command is:

```
closelib <libobj>
```

<libobj>

Name of the library object to close. The `lib` command defines the library object.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical r
...
closelib libobj
```

Closes the `libobj` library database object for the `DESIGNS` library. Other XEL commands no longer recognize `libobj` as a database object name.

compileFindAll

Runs a compile several times as a trial, records the results of each run to a log file, and reports the success rate and best result. For the best result, the first priority is fastest emulation speed, and the second priority is least number of dropped probes.

Note: This command does not create a downloadable database like the [compileFindBest](#) and the [compileFindFirst](#) commands.

This command can identify the optimal hardware configuration to be recommended for initial purchase or an upgrade.

For example, if you have a 12M ASIC gate design that is initially configured for a PD-3, run 100 trials with starting seed 1 (that is, seeds 1...100) and record the results. You can then increase the hardware configuration (using the Virtual Hardware Configuration feature) to PD-4, and repeat the evaluation. The results of such trials can be plotted into a table such as the following:

Table 16-1 Identifying a Suitable Hardware Configuration

Hardware Configuration	Cost	Success Rate	Emulation Speed			
			Best	Median	Average	Worst

This information lets you choose the hardware configuration with the optimum price/performance point to order.

Before using this command, execute [design](#) to specify the design library and cell name. Also, [precompile](#) must complete successfully on the design database.

The syntax for this command is:

```
compileFindAll [-max_trials <max_trials>]
compileFindAll [-seed <start_seed>]
compileFindAll [-max_steps <max_steps>]
compileFindAll [-min_steps <min_steps>]
compileFindAll [-crit_paths <crit_paths>]
compileFindAll [-etOnly]
```

```
compileFindAll [-2pass]  
compileFindAll [-logfile <logfile>]
```

-max_trials <max_trials>

Specifies the number of trial compiles to run. This number must be a positive integer. The default value is 30.

-seed <start_seed>

Specifies the seed number for the first trial that is incremented by one for every trial. This number must be a positive integer. The default is 0, which instructs the Compiler to select a random seed number for each trial.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* variable must be a multiple of 2. The range for -max_steps is 1-4096.

-min_steps <min_steps>

The *<min_steps>* parameter is usually a complement to *<max_steps>*. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both *<min_steps>* and *<max_steps>* are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in *<min_steps>* should not be greater than the number specified in *<max_steps>*.

The *number* specified in *<min_steps>* must be a multiple of 2. The range for *<min_steps>* is 1-4096.

-crit_paths <crit_paths>

Specifies the number of critical paths to be reported for each successful trial compile. By default, the compiler reports one critical path. The critical path information for the *<i>*th successful trial is stored in file *./tmp/et3pass2.<i>.msg*, where *<i>* is the number of successful trial compiles.

For example, by default, the `compileFindAll` command runs 30 compile iterations. In this case, for the first compile result, the critical path information will be stored in a file called `./tmp/et3pass2.1.msg`. Similarly, for the 29th compile result, the critical path information will be stored in the file `./tmp/et3pass2.29.msg` and for the 30th compile result, the information will be stored in the `./tmp/et3pass2.30.msg` file.

This switch does not affect design compilation in any other way.

-etOnly

Disables flow control checking. By default, the `compileFindAll` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-2pass

Makes the compile process run faster. However, if `compile` is run again using the same options and seed, the compile speed might be different.

-logfile <logfile>

Specifies the filename of the log file. The default is `./tmp/xeCompile.log`.

Examples

`compileFindAll`

Runs 30 (default) compiles with sequential seeds starting with a random seed and logs results to file `./tmp/xeCompile.log`.

`compileFindAll -max_trials 50 -seed 759`

Runs 50 compiles with sequential seeds starting with 759 and logs results to file `./tmp/xeCompile.log`.

`compileFindAll -max_trials 100 -logfile ./tmp/run_100.log`

Runs 100 compiles and logs results to file `./tmp/run_100.log`.

compileFindAllMemUtil

Runs a compile with different values of memory utilization, several times with each value as a trial, starting from a maximum memory utilization value, with given decrements, down to a minimum value. The number of compile runs can be defined either explicitly or implicitly using the switches provided with this command.

For each trial run, a record will be generated in the log file. Each record will contain the memory utilization value, the number of the trial, the seed value, the compilation result (pass/fail), and the step count. In addition, the success rate, the average step count, and the gate count for each memory utilization value will also be recorded.

There will also be a summary report similar to that for the [compileFindAll](#) command.

For each trial compile run <N> (where N=1,2,3...) and each memory utilization value <U>, there will be an output file tmp/et3pass2.util<U>.<N>.msg.

For example, for third trial compile run for memory utilization value 80, the file generated will be tmp/et3pass2.util80.3.msg.

This command enables you to explore the effects of changing the memory utilization on the compile success rate, capacity, and step count.

The `compileFindAllMemUtil` command can run after the [precompile](#) command.

To run the `compileFindAllMemUtil` command with some `compilerOptions` related to memory compaction, you must previously run [compile](#) (or [compileFindBest](#), or [compileFindAll](#), or [compileFindFirst](#)) command with the required `compilerOptions`.

Note: The `compileFindAllMemUtil` command can be very time consuming.

The syntax for this command is:

```
compileFindAllMemUtil [-max_trials <max_trials>]
compileFindAllMemUtil [-max_util <max_util>]
compileFindAllMemUtil [-min_util <min_util>]
compileFindAllMemUtil [-dec_util <dec_util>]
compileFindAllMemUtil [-trials_per_util <trials_per_util>]
compileFindAllMemUtil [-seed <start_seed>]
compileFindAllMemUtil [-max_steps <max_steps>]
compileFindAllMemUtil [-min_steps <min_steps>]
compileFindAllMemUtil [-crit_paths <crit_paths>]
```

```
compileFindAllMemUtil [-eOnly]  
compileFindAllMemUtil [-2pass]  
compileFindAllMemUtil [-logfile <logfile>]  
compileFindAllMemUtil [-minMem]
```

-max_trials <max_trials>

Specifies the number of trial compiles to run. This number must be a positive integer. The default value is 30.

-max_util <max_util>

Specifies the maximum value of memory utilization. This number must be a positive integer. The default value is 100.

-min_util <min_util>

Specifies the minimum value of memory utilization. This number must be a non-negative integer. The default value is 50.



The value of `min_util` **must not** exceed the value of `max_util`.

-dec_util <dec_util>

Specifies the number by which decrements should be made in the memory utilization value during the trial compile runs. This number must be a positive integer. The default value is 10.

-trials_per_util <trials_per_util>

Specifies the number of trial compiles to be run for each memory utilization value. If this option is not provided, the value of `max_trials` defines the total number of trials for all memory utilization values.

If the value of `max_trials` is not a multiple of the value of memory utilization, the number of trials might differ by 1 for different memory utilization values. Refer to the second example in the Examples section below.

-seed <start_seed>

Specifies the seed number for the first trial that is incremented by one for every trial. This number must be a positive integer. The default is 0, which instructs the Compiler to select a random seed number for each trial.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* variable must be a multiple of 2. The range for -max_steps is 1-4096.

-min_steps <min_steps>

The *<min_steps>* parameter is usually a complement to *<max_steps>*. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both *<min_steps>* and *<max_steps>* are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in *<min_steps>* should not be greater than the number specified in *<max_steps>*.

The *number* specified in *<min_steps>* must be a multiple of 2. The range for *<min_steps>* is 1-4096.

-crit_paths <crit_paths>

Specifies the number of critical paths to be reported for each successful trial compile. By default, the compiler reports one critical path. The critical path information for the *<i>*th successful trial is stored in file `./tmp/et3pass2.<i>.msg`, where *<i>* is the number of successful trial compiles.

For example, by default, the `compileFindAllMemUtil` command runs 30 compile iterations. In this case, for the first compile result, the critical path information will be stored in a file called `./tmp/et3pass2.1.msg`. Similarly, for the 29th compile result, the critical path information will be stored in the file `./tmp/et3pass2.29.msg` and for the 30th compile result, the information will be stored in the `./tmp/et3pass2.30.msg` file.

This switch does not affect design compilation in any other way.

-etOnly

Disables flow control checking. By default, the `compileFindAllMemUtil` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-2pass

Makes the compile process run faster. However, if `compile` is run again using the same options and seed, the compile speed might be different.

-logfile <logfile>

Specifies the filename of the log file. The default is `./tmp/xeCompile.log`.

-minMem

Reduces memory required for compilation by running sub-processes sequentially instead of concurrently. If memory usage is very large, performance is negatively impacted due to memory swapping to disk. In such a case, using the `-minMem` option reduces the time required for compilation.

Examples

```
compileFindAllMemUtil -min_util 75 -dec_util 4 -trials_per_util 5
```

Runs 5 compiles for each memory utilization value (100,96,92,88,84,80,76). 35 compiles in total.

```
compileFindAllMemUtil -min_util 75 -dec_util 4
```

Runs 30 compiles in total. 5 for each memory utilization value (100,96) and 4 for each memory utilization value (92,88,84,80,76).

compileFindBest

Runs a compile several times as a trial so that it finds the compile with the best emulation speed (lowest number of steps).

Before using this command, execute the [design](#) command to specify the design library and cell name. Also, [precompile](#) must complete successfully on the design database.

The syntax for this command is:

```
compileFindBest [-max_trials <max_trials>]
compileFindBest [-max_steps <max_steps>]
compileFindBest [-min_steps <min_steps>]
compileFindBest [-crit_paths <crit_paths>]
compileFindBest [-minMem]
compileFindBest [-etOnly]
compileFindBest [-2pass]
```

-max_trials <max_trials>

Specifies the number of trial compiles to run in order to find the first successful compile. This must be a positive integer. The default value is 30.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* variable must be a multiple of 2. The range for **-max_steps** is 1-4096.

-min_steps <min_steps>

The *<min_steps>* parameter is usually a complement to *<max_steps>*. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both *<min_steps>* and *<max_steps>* are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in *<min_steps>* should not be greater than the number specified in *<max_steps>*.

The *<number>* specified in *<min_steps>* must be a multiple of 2. The range for *<min_steps>* is 1-4096.

-crit_paths <crit_paths>

Specifies number of critical paths to be reported in the `tmp/et3compile.msg` file. By default, the compiler reports one critical path.

This switch does not affect design compilation in any other way.

-minMem

Reduces memory required for compilation by running sub-processes sequentially instead of concurrently. If memory usage is very large, performance is negatively impacted due to memory swapping to disk. In such a case, using the `-minMem` option reduces the time required for compilation.

-etOnly

Disables flow control checking. By default, the `compileFindBest` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-2pass

Makes the compile process run faster. However, if `compile` is run again using the same options and seed, the compile speed might be different.

Note: Once a `compile` passes with a certain number of steps, the *<max_steps>* parameter is automatically set to the next lower valid value for succeeding iterations. As a result, compiles that are not strictly better are terminated early to reduce compilation run time.

Examples

`compileFindBest`

Tries up to 30 times (default) to find the best compile.

`compileFindBest -max_trials 100 -max_steps 300`

UXE Command Reference Manual

Deprecated Commands and Variables

Tries up to 100 times to find the best compile that takes no more than 300 steps.

```
compileFindBest -max_trials 40 -max_steps 300 -min_steps 280
```

Tries up to 40 times to find the best compile that takes no more than 300 steps, and no less than 280 steps.

```
compileFindBest -max_steps 280 -min_steps 280
```

Tries up to 30 times to find the best compile that takes exactly 280 steps.

compileFindBestOptimization

Runs a compile several times as a trial for each pair of values given for optimizationEffort and placementEffort. This command enables you to explore the effects of changing the optimization parameters on the compile success rate, capacity, and step count.

For each trial run, a record will be generated in a log file containing the following information:

- Date and time
- optimizationEffort value
- placementEffort value
- Number of the trial
- Seed value
- Compilation result (that is, pass or fail)
- Step count

The log file will also include the following additional information for each pair of optimizationEffort and placementEffort values:

- Success rate
- Average step count
- Gate count

There will also be a summary report similar to that generated for the compileFindAll command.

For each trial compile run <N> (where N=1, 2, 3...) for each pair of optimizationEffort and placementEffort values, <OE> and <PE>, there will be an output file named the following:

tmp/et3pass2.oe<OE>.pe<PE>.<N>.msg

For example, for third trial compile run for optimizationEffort value ULTRA and placementEffort value LOW, the file generated will be named
tmp/et3pass2.oeULTRA.peLOW.3.msg.

Before using this command, execute the design command to specify the design library and cell name. Also, precompile must complete successfully on the design database.



The `compileFindBestOptimization` command can be very time consuming.

The syntax for this command is:

```
compileFindBestOptimization [-max_trials_each <max_trials_each>]
compileFindBestOptimization [-oE {<values>}]
compileFindBestOptimization [-pE {<values>}]
compileFindBestOptimization [-seed <start_seed>]
compileFindBestOptimization [-max_steps <max_steps>]
compileFindBestOptimization [-min_steps <min_steps>]
compileFindBestOptimization [-crit_paths <crit_paths>]
compileFindBestOptimization [-etOnly]
compileFindBestOptimization [-2pass]
compileFindBestOptimization [-logfile <logfile>]
compileFindBestOptimization [-minMem]
```

-max_trials_each <max_trials_each>

Specifies the number of trial compiles to run for each pair of optimizationEffort and placementEffort values. This number must be a positive integer. The default value is 5.

-oE {<values>}

Specifies the list of values for optimizationEffort. Possible values are any subset of {LOW MEDIUM HIGH ULTRA}. The default list is {LOW HIGH}. Any shortcut is acceptable and characters are case-insensitive. For example, H HI HIGH h hig are all equivalent.

-pE {<values>}

Specifies the list of values for placementEffort. Possible values are any subset of {LOW HIGH}. The default list is {LOW HIGH}. Any shortcut is acceptable and characters are case-insensitive.

-seed <start_seed>

Specifies the seed number for the first trial that is incremented by one for every trial. This number must be a positive integer. The default is 0, which instructs the compiler to select a random seed number for each trial.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* variable must be a multiple of 2. The range for `-max_steps` is 1-4096.

-min_steps <min_steps>

The *<min_steps>* parameter is usually a complement to *<max_steps>*. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both *<min_steps>* and *<max_steps>* are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in *<min_steps>* should not be greater than the number specified in *<max_steps>*.

The *number* specified in *<min_steps>* must be a multiple of 2. The range for *<min_steps>* is 1-4096.

-crit_paths <crit_paths>

Specifies the number of critical paths to be reported for each successful trial compile. By default, the compiler reports one critical path. The critical path information for the *<i>*th successful trial is stored in file `./tmp/et3pass2.<i>.msg`, where *<i>* is the number of successful trial compiles.

For example, by default, the `compileFindAllMemUtil` command runs 30 compile iterations. In this case, for the first compile result, the critical path information will be stored in a file called `./tmp/et3pass2.1.msg`. Similarly, for the 29th compile result, the critical path information will be stored in the file `./tmp/et3pass2.29.msg` and for the 30th compile result, the information will be stored in the `./tmp/et3pass2.30.msg` file.

This switch does not affect design compilation in any other way.

-etOnly

Disables flow control checking. By default, the `compileFindBestOptimization` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-2pass

Makes the compile process run faster. However, if `compile` is run again using the same options and seed, the compile speed might be different.

-logfile <logfile>

Specifies the filename of the log file. The default is `./tmp/xeCompile.log`.

-minMem

Reduces memory required for compilation by running sub-processes sequentially instead of concurrently. If memory usage is very large, performance is negatively impacted due to memory swapping to disk. In such a case, using the `-minMem` option reduces the time required for compilation.

Examples

```
compileFindBestOptimization
```

Runs 5 trials for each of 4 pairs of `optimizationEffort` values {LOW HIGH} and `placementEffort` values {LOW HIGH}, in total 20 trials.

```
compileFindBestOptimization -oE {l m u} -max_trials_each 3
```

Runs 3 trials for each of 6 pairs of `optimizationEffort` values {LOW MED ULTRA} and `placementEffort` values {LOW HIGH}, in total 18 trials.

compileFindFirst

Runs a compile several times as a trial and stops after the first successful compile.

Before using this command, execute the [design](#) command to specify the design library and cell name. Also, [precompile](#) must complete successfully on the design database.

The syntax for this command is:

```
compileFindFirst [-max_trials <max_trials>]  
compileFindFirst [-max_steps <max_steps>]  
compileFindFirst [-min_steps <min_steps>]  
compileFindFirst [-crit_paths <crit_paths>]  
compileFindFirst [-minMem]  
compileFindFirst [-etOnly]  
compileFindFirst [-2pass]
```

-max_trials <max_trials>

Specifies the number of trial compiles to run in order to find the first successful compile. This must be a positive integer. The default value is 5.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The *<number>* variable must be a multiple of 2. The range for -max_steps is 1-4096.

-min_steps <min_steps>

The *<min_steps>* parameter is usually a complement to *<max_steps>*. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both *<min_steps>* and *<max_steps>* are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in *<min_steps>* should not be greater than the number specified in *<max_steps>*.

The *<number>* specified in *<min_steps>* must be a multiple of 2. The range for *<min_steps>* is 1-4096.

-crit_paths <crit_paths>

Specifies number of critical paths to be reported in the `tmp/et3compile.msg` file. By default, the compiler reports one critical path.

This switch does not affect design compilation in any other way.

-minMem

Reduces memory required for compilation by running sub-processes sequentially instead of concurrently. If memory usage is very large, performance is negatively impacted due to memory swapping to disk. In such a case, using the `-minMem` option reduces the time required for compilation.

-etOnly

Disables flow control checking. By default, the `compileFindFirst` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-2pass

Makes the compile process run faster. However, if `compile` is run again using the same options and seed, the compile speed might be different.

Examples

`compileFindFirst`

Tries up to 5 times (default) to find the first compile that passes.

`compileFindFirst -max_trials 100 -max_steps 300`

Tries up to 100 times to find the first compile that passes and takes no more than 300 steps.

`compileFindFirst -min_steps 280`

Tries up to 5 times to find the first compile that passes and takes no less than 280 steps.

cosimAcs

Specifies the active clock synchronization information. Using this command, you can name the signals that will cause synchronization to occur on their rising edge, falling edge, or both edges.

Using the `cosimAcs` command without any options lists the nets on which synchronization will take place.

The syntax for this command is:

```
cosimAcs -add {<net> <edge>}  
-rm {<net>}  
-clear
```

-add {*<net>* *<edge>*}

Adds the net to the list of the nets on which the synchronization will take place. Here, *<edge>* can be positive, negative, or both.

-rm {*<net>*}

Removes the net from the list of nets on which synchronization will take place.

-clear

Clears the entire list of nets on which synchronization would have taken place.

createUserSession

Creates a user directory with the name as `<dir_name>.session` and makes it the current directory so that all the subsequent logs, for example, `xe.msg` and waveforms are stored in the new directory.

The syntax for this command is:

```
createUserSession <dir_name>
```

<dir_name>

Specifies the name with which a new directory will be created.

criticalAccess

Forces the User Interface operations of examine, deposit, or force signals to be implemented through the DAS access (use of which has been deprecated in UXE) for high performance.

The syntax for this command is:

```
criticalAccess -add {<signal_name> [R | RW | W]} |  
-rm <list of signals>
```

-add {<signal_name> [R | RW | W]}

Adds the specified signal with the specified access permissions.

The R option enables the specified signal to be examined.

The RW option enables the specified signal to be examined and a value to be deposited (short-lived forcing).

The W option enables UI values to be forced on the specified signal.

-rm <list_of_signals>

Removes the specified signals from the list of signals for UI critical access.

dontAcqOnEvent

Specifies a condition for not capturing probed data.

This command is disabled when SDL is not active.

The syntax for this command is:

`dontAcqOnEvent [<expression>]`

<expression>

Specifies that probed data should not be captured when the specified expression is true. However, data is captured if the expression is false and the SDL program does not disable the capture.

The expression syntax is the same as the one accepted by the SDL program inside an IF statement, and might contain any design signal names, as well as previous definitions from trEvent and symbol commands.

eventSignal

Adds or deletes event signal nets from the event signal list. Legacy SDL (not supported any more) required that all signals used in the SDL program must be defined during compilation as event signals. This command is now completely ignored and can be safely removed without any side effects.

The syntax for this command is:

```
eventSignal -add <net>... |  
    -addNreport <net_name>  
    -rm <net>...
```

group

Creates or deletes a group associated with one or more signals. This command also gets, sets, forces, or releases the current state of the signals included in the group. In addition, this command lists or displays the groups. Without parameters, it lists the names of all the currently defined groups.

The syntax for this command is:

```
group [-add <group> <sig>... | -rm <group>  
-list <group>  
-get <group>  
-set <group> <value>  
-force <group> <value>  
-release <group>
```

-add <group> <sig>...

Creates the specified group, which includes one or more signals as specified in the *<sig>...* signal list. The first signal name in the list represents the most significant bit (MSB).

-rm <group>

Deletes the specified group(s).

-list <group>

Lists or displays all signals associated with the specified group.

-get <group>

Gets or displays the current state of the specified group. The returned value is a binary string containing zeroes and ones. The string is as long as the total width of the group.

-set <group> <value>

Sets the states of the signals in the specified group to the specified *<value>*. The syntax of *<value>* is a binary string. The string must be of the same length as the total width of the signals in the specified group. The states are only set for a single cycle. This option is equivalent to setting each signal of the group using the `signal -set` command.

UXE Command Reference Manual

Deprecated Commands and Variables

-force <group> <value>

Sets the states of the signals in the specified group to the given *<value>*. The syntax of *<value>* is a binary string. The string must be of the same length as the total width of the signals in the specified group. The state remains set until explicitly released by a *-release <group>* command. This option is equivalent to setting each signal of a group using the signal *-force* command.

-release <group>

Releases a previous force on the specified signal. The signals are recalculated at the beginning of the next cycle, or by the next *group -get* or *signal -get* command, whichever is executed first.

inst

Creates a database object for an instance in a cell of your design.

Before using this command, execute `design` to specify the design library and cell names.

Also use `lib` to define a database object for the library containing the cell, and `cell` to define a database object for the cell containing the instance.

Use this command before using `termInst`.

The syntax for this command is:

```
inst <instobj> <inst> <celfobj>
```

<instobj>

Database object identifier to assign to the instance.

<inst>

Name of the instance to define as a database object.

<celfobj>

Name of the cell database object containing the instance, as defined in `cell`.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical r
cell celfobj CLOCK_LAB libobj
inst instobj FF0 celfobj
```

Creates a database object named `libobj` for the `DESIGNS` library, a database object named `celfobj` for the `CLOCK_LAB` cell in that library, and finally a database object named `instobj` for the `FF0` instance within that cell.

lib

Creates a database object for a library in your design.

Before using this command, execute `design` to specify the design library and cell names for your design.

Use this command before using `cell`, `inst`, `net`, `term`, and `termInst`.

The syntax for this command is:

```
lib <libobj> <lib> <viewtype>
```

<libobj>

Database object identifier to assign to the library.

<lib>

Name of the library to define as a database object.

<viewtype>

<viewtype> must be the following keyword (verbatim):

logical

This is maintained for backward compatibility with previous software versions.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical
```

Creates a database object named `libobj` that represents the `DESIGNS` library.

net

Creates a database object for a net in a design cell.

Before using this command, execute `design` to specify the design library and cell names for your design.

Also use `lib` to define a database object for the library containing the cell, and the `cell` to define a database object for the cell containing the net.

The syntax for this command is:

```
net <netobj> <net> <celfobj>
```

<netobj>

Database object identifier to assign to the net.

<net>

Name of the net to define as a database object.

<celfobj>

Name of the cell database object containing the net, as defined in `cell`.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical r
cell celfobj CLOCK_LAB libobj
net netobj D_IN celfobj
```

Creates a database object named `libobj` for the `DESIGNS` library, a database object named `celfobj` for the `CLOCK_LAB` cell in that library, and finally a database object named `netobj` for the `D_IN` net in that cell.

parallelCompileFindBest

Performs the required number of back-end compiles (trials) with different seeds on different machines in parallel using LSF and chooses the best trial (the one with the minimum step count).

This command is a fast parallel alternative to the [compileFindBest](#), [compileFindFirst](#) and [compileFindAll](#) commands performing trials sequentially.

Before using this command, execute the [design](#) command to specify the design library and cell name. Also, [precompile](#) must complete successfully on the design database.

The syntax for this command is:

```
parallelCompileFindBest [-max_trials <max_trials>]  
parallelCompileFindBest [-seed <start_seed>]  
parallelCompileFindBest [-max_steps <max_steps>]  
parallelCompileFindBest [-min_steps <min_steps>]  
parallelCompileFindBest [-crit_paths <crit_paths>]  
parallelCompileFindBest [-etOnly]  
parallelCompileFindBest [-first]  
parallelCompileFindBest [-1pass]  
parallelCompileFindBest [-keep_all]  
parallelCompileFindBest [-rerun_best]  
parallelCompileFindBest [-no_export]
```

-max_trials <max_trials>

Specifies the number of trial compiles to run in order to find the best successful compile. This must be a positive integer. The default value is 30.

-seed <start_seed>

Specifies the seed number for the first trial that is incremented by one for every trial. This number must be a positive integer. The default is 0, which instructs the compiler to select a random seed number for each trial.

By default, each trial gets a random seed. If *<start_seed>* is defined, trials 1, 2, 3,... receive seeds *<start_seed>*, *<start_seed>+1*, *<start_seed>+2*,..., respectively.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The *<max_steps>* variable must be a positive number in multiple of 2.

Note: The range for *<max_steps>* is 1-4096.

If this option is present, trials with step counts exceeding the specified *<max_steps>* are considered as failed.

-min_steps <min_steps>

The *<min_steps>* parameter is usually a complement to *<max_steps>*. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both *<min_steps>* and *<max_steps>* are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in *<min_steps>* should not be greater than the number specified in *<max_steps>*.

If this option is present and step count for a trial is less than *<min_steps>*, the natural step count will be increased to *<min_steps>*.

The *<min_steps>* must be a positive number in multiple of 2. The range for *<min_steps>* is 1-4096.

-crit_paths <crit_paths>

Specifies number of critical paths to be reported in the `tmp/et3compile.msg` file. By default, the compiler reports one critical path.

This switch does not affect design compilation in any other way.

-etOnly

Disables flow control checking. By default, the `compileFindBest` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-first

Terminates the remaining trials as soon as one of the trials succeeds. This option makes the `parallelCompileFindBest` command similar to `compileFindFirst` rather than `compileFindBest`.

-1pass

Performs each trial in full with LSF.

By default, back-end compile is split into two parts. The first part (from the beginning to `mpart`) is *seed-independent* and executed only once, in the design directory, without LSF. At the end the intermediate proto file is saved. Then the second, seed-dependent part (from `upart` to the end) is executed with LSF.

-keep_all

Blocks the removal of the temporary subdirectories and files.

-rerun_best

Restricts saving of the final proto files at the end of trials. Instead, the best trial is reran in the design directory.

This option slows down the compile but might be useful if the disc space is limited because final proto files need not be saved.

-no_export

Restricts saving of the final proto files at the end of trials. This option might be useful for experiments, such as collecting compile-related statistics.

parallelCompileFindBestOptimization

Performs the required number of back-end compiles (trials) with different seeds and different combinations of optimization effort and placement effort on different machines in parallel using LSF and chooses the best trial (the one with the minimum step count).

The goal of this command is to speed-up the iterative compiles because being executed sequentially they would take time for big designs. This command is a parallel alternative to the [compileFindBestOptimization](#) command. This command is similar to [parallelCompileFindBest](#) command.

Before using this command, execute the [design](#) command to specify the design library and cell name. Also, [precompile](#) must complete successfully on the design database.

The syntax for this command is:

```
parallelCompileFindBestOptimization [-max_trials_each <max_trials_each>]
parallelCompileFindBestOptimization [-oE {<values>}]
parallelCompileFindBestOptimization [-pE {<values>}]
parallelCompileFindBestOptimization [-seed <start_seed>]
parallelCompileFindBestOptimization [-max_steps <max_steps>]
parallelCompileFindBestOptimization [-min_steps <min_steps>]
parallelCompileFindBestOptimization [-crit_paths <crit_paths>]
parallelCompileFindBestOptimization [-etOnly]
parallelCompileFindBestOptimization [-first]
parallelCompileFindBestOptimization [-1pass]
parallelCompileFindBestOptimization [-keep_all]
parallelCompileFindBestOptimization [-rerun_best]
parallelCompileFindBestOptimization [-no_export]
parallelCompileFindBestOptimization [-logfile <logfile>]
```

-max_trials_each <max_trials_each>

Specifies the number of trial compiles to run for each pair of optimizationEffort and placementEffort values. This number must be a positive integer. The default value is 5.

-oE {<values>}

Specifies the list of values for optimizationEffort. Possible values are any subset of {LOW MEDIUM HIGH ULTRA}. The default list is {LOW HIGH}. Any shortcut is acceptable and characters are case-insensitive. For example, H HI HIGH h hiG are all equivalent.

-pE {<values>}

Specifies the list of values for placementEffort. Possible values are any subset of {LOW HIGH}. The default list is {LOW HIGH}. Any shortcut is acceptable and characters are case-insensitive.

-seed <start_seed>

Specifies the seed number for the first trial that is incremented by one for every trial. This number must be a positive integer. The default is 0, which instructs the compiler to select a random seed number for each trial.

By default, each trial gets a random seed. If <start_seed> is defined, trials 1, 2, 3,... receive seeds <start_seed>, <start_seed>+1, <start_seed>+2,..., respectively.

-max_steps <max_steps>

Specifies the maximum number of steps that can be accepted by the compiler. The <max_steps> variable must be a positive number in multiple of 2.

Note: The range for <max_steps> is 1-4096.

If this option is present, trials with step counts exceeding the specified <max_steps> are considered as failed.

-min_steps <min_steps>

The <min_steps> parameter is usually a complement to <max_steps>. It sets the minimum number of steps for the compiler scheduler. In certain cases, this is crucial for in-circuit emulation, for example, when the design generates a clock for the target that must run at a specific frequency.

If both <min_steps> and <max_steps> are set to the same number, the design will run at this exact step count (if it can be scheduled at this step count). The number specified in <min_steps> should not be greater than the number specified in <max_steps>.

If this option is present and step count for a trial is less than `<min_steps>`, the natural step count will be increased to `<min_steps>`.

The `<min_steps>` must be a positive number in multiple of 2. The range for `<min_steps>` is 1-4096.

-crit_paths <crit_paths>

Specifies number of critical paths to be reported in the `tmp/et3compile.msg` file. By default, the compiler reports one critical path.

This switch does not affect design compilation in any other way.

-etOnly

Disables flow control checking. By default, the `compileFindBest` command performs flow control checking. It determines whether the `precompile` and `icePrepare` commands need to run, and runs them if necessary before running the actual compilation step. With the `-etOnly` option, flow control checking is disabled so that only the actual compilation step is run.

-first

Terminates the remaining trials as soon as one of the trials succeeds. This option makes the `parallelCompileFindBest` command similar to `compileFindFirst` rather than `compileFindBest`.

-1pass

Performs each trial in full with LSF.

By default, back-end compile is split into two parts. The first part (from the beginning to `mpart`) is *seed-independent* and executed only once, in the design directory, without LSF. At the end the intermediate proto file is saved. Then the second, seed-dependent part (from `upart` to the end) is executed with LSF.

-keep_all

Blocks the removal of the temporary subdirectories and files.

-rerun_best

Restricts saving of the final proto files at the end of trials. Instead, the best trial is reran in the design directory.

This option slows down the compile but might be useful if the disc space is limited because final proto files need not be saved.

-no_export

Restricts saving of the final proto files at the end of trials. This option might be useful for experiments, such as collecting compile-related statistics.

-logfile <logfile>

Specifies the filename of the log file. The default is ./tmp/xeCompile.log.

primaryClockSource

Specifies the name of the fastest clock. Clock sources are specified by net name. Execute this command before running precompile.

This command returns the existing primary clock. Nothing is returned if there is no existing primary clock source.

Note: The `primaryClockSource` command enables you to specify the fastest CAKE clock without needing to check its frequency. If the frequency of the specified clock source is not the highest, the precompiler automatically raises the frequency as required.

The syntax for this command is:

```
primaryClockSource -add <primary_clock_source_name>  
primaryClockSource -rm <primary_clock_source_name>
```

-add <primary_clock_source_name>

Specifies fast clock.

-rm <primary_clock_source_name>

Removes clock as a primary clock, but retains the clock source.

qtCounter1

Specifies initial load value for SDL's `counter1`. If the initial load value of `counter1` needs to be controlled from the command line, it now can be done by accessing the values of regular Tcl variables directly from within the SDL program. For details, refer to the *Controlling the Run with SDL* chapter of the *UXE User Guide*.

qtCounter2

Specifies initial load value for SDL's `counter2`. If the initial load value of `counter2` needs to be controlled from the command line, it now can be done by accessing the values of regular Tcl variables directly from within the SDL program. For details, refer to the *Controlling the Run with SDL* chapter of the *UXE User Guide*.

qtTriggerByState

Specifies whether triggering by SDL (State Definition Language) is enabled or disabled. A value of 1 means SDL is enabled. A value of 0 means SDL is disabled.

qtTriggerFile

Specifies the name of the trigger definition file, which contains the SDL program.

-force

The behavior of the **-force** option of the `readCPFfile <cpfFileName>` command is now the default behavior on executing the `readCPFfile` command. Therefore, the **-force** option is now deprecated.

The compiler ignores the names of the instances, pins, or nets that are given in the specified CPF file, but not found in the Palladium database. For example, you specify the following command in the CPF file giving the instance names as `path_to_inst1`, `path_to_inst2`, and `path_to_inst3`.

```
create_power_domain -name pd1 \
    -instances {path_to_inst1 path_to_inst2 path_to_inst3}
```

However, the instance `path_to_inst2` does not exist in the Palladium database. In this case, the `readCPFfile -force <cpfFileName>` command or `readCPFfile <cpfFileName>` command will run the above command as follows and eventually print a warning about `path_to_inst2`.

```
create_power_domain -name pd1 -instances {path_to_inst1 path_to_inst3}
```

For more information, refer to the description of the [readCPFfile](#) command.

resetCycleNum

Resets the cycle number (or simulated time) to 0. You can execute this command only after downloading the design.

This command only affects the value returned by the [getCycleNum](#) and [getTime](#) commands, but does not affect other data (such as time/cycles reported on waveform or from SDL).

Note: This command does not reset the emulation state or restart emulation from the beginning. To reset/restart emulation from beginning, you must save the initial state after download and before executing the [run](#) command, using [save](#) and then restore that state after a run using [restart](#). This command is available in the STB mode. This command is not supported with vvmDebug

The syntax for this command is:

```
resetCycleNum
```

restoreState

Restores a previously saved design state.



Important
The saved data must be from the same compiled design and at the same software release level. The restored state overwrites any memory commands.

Run this command after downloading a design to a Palladium XP verification system. This command can be used to restore a design state that was saved either in this session or in a previous session.

The syntax for this command is:

```
restoreState <save_name>
```

<save_name>

Specifies the basename of the files in which the states are saved. The basename is associated with .hdr or .dmp extensions.

Note: Text format is not supported with the restoreState command.

runtimeCLOCKconf

All the `runtimeCLOCKconf` options are supported by the [clockConfig](#) command.

savelib

Saves a library database object that you created using `lib`.

Before using this command, execute `design` to specify the design library and cell names for your design. Also, use `lib` to define a database object for the library.

The syntax for this command is:

`savelib <libobj>`

<libobj>

Name of the library database object to save, as defined in `lib`.

saveState

Saves the entire state of the current emulated design in files with the specified name and extensions, such as .hdr and .dmp. This command can also be used to save the run-time environment as a Tcl script.

Before executing the `saveState` command, the emulation clocks must be stopped using the `stop` command. In LA mode, use the `stop -emulation` command to stop the emulation clocks.

The syntax for saving the design state is:

```
saveState <save_name> [bin | text]
```

The syntax for saving the run-time environment is:

```
saveState -environment <save_name>
```

<save_name>

Specifies the basename of the files where the information will be saved.

When saving the design state, the state information is saved into files under the `<save_name>` directory. The `<save_name>` parameter can also include the path information. For example, `mysave/ss1`, where the state will be saved under the `mysave/ss1` directory.

bin

Saves the design state in binary format. This is the default mode if no option is specified.

text

Saves the state in text format.



Only the design states saved in binary format can be restored. The only purpose of saving a design state in the text mode is for viewing it in the text editor.

-environment <save_name>

Saves the run-time environment into a Tcl script file with the specified *<save_name>*.

When the saved Tcl script is sourced using the `source <save_name>` command, the saved run-time environment will be restored, which includes the following:

- Predefined values of the read-write run-time parameters
- Stop delays
- Probes
- Symbols
- SDL settings
- SDL events
- Clock settings (Note that these setting are not saved in the VD mode.)

signal

Gets, sets, forces, or releases the current state of a single signal, a symbol, or a bus. A signal can be set, forced, or released if it is specified as a keepNet during compile. You can get the value of any signal in the design.

The syntax for this command is:

```
signal -get <signal>
      -get <symbol>
      -get <bus>[<x>:<y>]
      -alias <signal>
      -set <signal> 0 | 1 | -set <symbol> <value>
      -force <signal> 0 | 1 | -force <symbol> <value>
      -allff { -set | -force } { 0 | 1 | random [-seed <number>] } [-instance
<instance_name>]
      -allff -release [-instance <instance_name>]
      -isForced <signal>
      -listForce
      -release <signal> ... | -release <symbol> ...
```

-get <signal>

Gets the current state of the specified signal. It also gets value for the specified memory words.

-get <symbol>

Gets the current state of the specified symbol.

-get <bus>[<x>:<y>]

Gets the current state of the signal specified using the `<bus> [<x> : <y>]` option. For the `<bus> [<x> : <y>]` signals, the hierarchy delimiter for the internal signals can be given using a double underscore “__” or a period “.”, depending upon the definition set during compile time.

Note: When using signal `-get` to get the value of a signal, bus, or symbol using a radix other than binary, specify the radix using the `signalRadix` parameter.

-alias <signal>

Lists all signal names (at different levels of hierarchy, or connected by feedthroughs or Q_ASSIGNs) that are equivalent to the specified signal name. Knowing all the equivalent signal names can help debug the cases where a signal is declared as power or ground at an unknown place in the hierarchy.

-set <signal> 0 | 1 | -set <symbol> <value>

In STB or VD mode, it holds the specified state for the specified signal for only one clock cycle. This value is overwritten during the emulation run. Gets the value for the specified memory words.

In LA mode, signal -set is equivalent to signal -force followed by signal -release. The time interval between the -force and -release is usually many thousands of clock cycles.

A symbol can be set with a value of radix like hex, binary, decimal, or octal, prefaced by the appropriate letter. For example, `b011, `hffcd, `01237, `d089.

-force <signal> 0 | 1 | -force <symbol> <value>

Forces the state of the specified signal to the given binary value until explicitly released by a signal -release <signal> command. In the LA mode, to force the value of a <bus>[<x>:<y>] or <symbol>, the FCLK needs to be stopped if all the signals need to be changed at the same time.

A bus or symbol can be forced to a value of radix like hex, binary, decimal, or octal.

-allff {-set | -force} {0 | 1 | random [-seed <number>]} [-instance <instance_name>]

With the -set option, this command is equivalent to the following deposit command:

```
XEL> deposit -allff (0 | 1 | random [-seed <number>]) [-instance <instance_name>]
```

With the -force option, this command is equivalent to the following force command:

```
XEL> force -allff (0 | 1 | random [-seed <number>]) [-instance <instance_name>]
```

-allff -release [-instance <instance_name>]

This command is equivalent to the following `release` command:

```
XEL> release -allff [-instance <instance_name>]
```

-isForced <signal>

Returns 1 if the specified signal is being forced. Otherwise, it returns 0. The value 1 is returned even if a different (but equivalent) signal name was used in the `signal -force` command. Different signal names are regarded as equivalent if these names refer to the same signal. When a signal is unexpectedly flat high or flat low, use this command to check whether the signal is being forced using a different (but equivalent) name.

-listForce

Lists the forced signals, symbols, and buses.

-release <signal> ... | -release <symbol> ...

Releases a previous force on the specified signal. The value of the specified signal is recalculated at the next cycle. The sourceless nets that were declared as `keepNets` will retain their values (state).

sigTrace

Performs the following signal trace operations:

- Adds or removes signals to trace
- Uploads trace results to a trace file in the STB, LA, or VD modes
- Clears the contents of the `trace.out` file
- Lists all the signals that are currently selected for tracing
- Adds power domain probe-related functionality
- Collects toggle count data during run time for dynamic power analysis

The syntax for this command is:

```
sigTrace -add <signal>... [-must] <signal> ...
sigTrace -add [-must] <signal> ...
sigTrace -add *
sigTrace -addinst [-allnets] -depth <n> <instance> ... [-must]
        [-iregexp | -iglob | -ipattern <inst_pattern>]
        [-nregexp | -nglob | -npattern <net_pattern>] <top_inst_name>
sigTrace -addio <instance> ... [-must] <inst_name> ...
sigTrace -addcone -depth <n> <signal> ...
sigTrace -addpath -depth <n> <net1> <net2>
sigTrace -rm <signal>...
sigTrace -rm [-regexp | -glob | -pattern <pattern>]*
sigTrace -list [-regexp | -glob | -pattern <pattern>]
sigTrace -save [-gell <filename>]
sigTrace -import <filename> [-must] <filename>
sigTrace -clear | -upload | -prepareoffline
sigTrace -outfile <file format> <basename>
sigTrace -assertion -add | -rm <assertion name>
        -assertion -addfanin | -rmfanin <assertion name>
        -assertion -addAll | -rmAll
        -assertion -addInst | -rmInst <inst name>
sigTrace -captureMode [controlled | uncontrolled]
sigTrace -addinstpin <instance>
sigTrace -addPowerDomain <powerDomainName>
```

UXE Command Reference Manual

Deprecated Commands and Variables

```
sigTrace -addRetention <stateRetentionRule>
sigTrace -addIsolation <isolationRuleName>
sigTrace -addPowerModeTransition
sigTrace -setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>]
<instance_names>
sigTrace -setupToggleCount -rminst <instance_names>
sigTrace -setupToggleCount -rminst -all
sigTrace -setupToggleCount -listinst [-verbose]
sigTrace -getToggleCount [-append] [-weighted] [-cycle <n>] [-file <name>]
sigTrace -addinstff <instanceName> [-asicLibrary <.llib name> ... <.llib name>]
[-hardmacroLibrary <.llib name> .... <.llib name>]
sigTrace -addinstReg <instanceName> [-asicLibs <.libName> ....<.libName> ] [-
hardmacroLibs <.libName> ....<.libName>]
sigTrace -outfile -tcf <filename> [-instance <instanceName>] [-masterTCF] [-
slaveTCF] [-noregularTCF] [-segment <n>% | <cycleNumber>]
sigTrace -genPowerProfile -segmentLogFile <segmentLogFile> -
reportFileNamePattern <FileNamePattern>
```

-add <signal>...

Adds one or more signals to the trace signal list.

-addinst [-allnets] -depth <n> <instance> ...

Adds all the nets in the database hierarchy of the given instances to the trace signal list.

The **-allnets** option is used to add nets starting with **n#**, which are by default filtered and not added in the database hierarchy.

The **-depth** option is used to specify a maximum depth to restrict the search in the database, where **<n>** specifies the maximum levels of hierarchy. By default one hierarchy level is searched. A depth of zero means the whole hierarchy of an instance is searched.

-addinst [-iregexp | -iglob | -ipattern <inst_pattern>] | -addinst [-nregexp |

-nglob | -npattern <net_pattern>] <instance>|[-nregexp | -nglob | -npattern

<net_pattern>] <top_inst_name>

Used to specify the patterns to filter out nets by their names.

UXE Command Reference Manual

Deprecated Commands and Variables

- The `-iregexp`, `-iglob`, and `-ipattern` options are used to specify the search patterns for the hierarchical path component, excluding the net name. The instance name specified on the command line is also excluded.
- The `-nregexp`, `-nglob`, `-npattern` options specify the search patterns for the net name component, without the hierarchical path.
- The `-iregexp` and `-nregexp` options support regular expressions, while the `-iglob` and `-nglob` options support UNIX-style global pattern matching, which match range specifications. The `-ipattern` and `-npattern` options are similar to `-iglob` and `-nglob`, but more restrictive. They do not support range specification.

-addio <instance> ...

Adds all I/O nets for the given instances to the trace signal list.

-addcone -depth <n> <signal> ...

Adds all signals in the given nets input cone of logic to the trace signal list.

-addpath -depth <n> <net1> <net2>

Traces back from `<net1>` to `<net2>` and adds all nets on the path. The `-depth <n>` parameter indicates the number of levels that are traced back. The default value is 10 levels.

-rm <signal>...

Removes the specified signal(s) from the trace signal list. The `-rm *` command removes all signals from the trace signal list.

-rm [-regexp | -glob | -pattern <pattern>]*

Removes signals with names that match the specified pattern.

-list [-regexp | -glob | -pattern <pattern>]

Lists the signals in the trace signal list.

This command can also list signals with names that match the specified pattern.

Using the `-glob <pattern>` option returns a list of all nets in the trace upload list with names that match the specified pattern using the *glob* expression matching rules.

Using the `-regexp <pattern>` option returns a list of all nets in the trace upload list, where the signal names match the specified pattern using regular expression matching rules.

-save [-qel] <filename>

Saves signals to a specified file as a flat list.

If the `-qel` option is included in the command, the probe list is saved in the form of a QEL script.

-import <filename>

Brings in signals from the specified file. This file can be:

- `probe.upld`, previously saved using the `sigTrace -save` command or the Save button in the GUI.
- `probes.rc`, saved from the waveform when the signals were added.

-clear

Removes the `trace.out` file in STB mode, so that the next time, a new file is created.

The `sigTrace -clear` command clears only trace files, but not the emulation cycle count.

-upload

Uploads trace results from the emulator into a trace file. Also calculates virtual probes and writes them to the trace result file `trace.out` or `<vector>.out` file. In offline debug mode, only virtual probes are calculated and written to the resultant files, `trace.out` or `<vector>.out`.

-prepareoffline

Used in online mode to save data for offline mode. The data will be saved in a `*.phy` file.

-outfile <file_format> <basename>

Sets the name and format of the output to be used. The output will be saved in one or more files or a directory with format and name as determined by *<file_format>*. The value of *<file_format>* can be one of the following:

- *-vbf*: Generates a file with the name *<basename>.out* in VBF format.
- *-fsdb*: Generates a file with the name *<basename>.fsdb* in FSDB format.
- *-sst2*: Generates a directory with the name *<basename>* containing an SST2 (or SHM) database.
- *-vcd*: Generates a file with the name *<basename>.vcd* in VCD format.

All the modes will also generate *<basename>.phy* file for debugging in the offline mode.

-must

Attempts to force probe configuration.

-assertion -add | -rm <assertion_name>

Specifies the assertion to be added or removed from the tracelist. The *<assertion_name>* specifies the full hierarchical name of the assertion.

-assertion -addfanin | -rmfanin <assertion_name>

Specifies that the fanin signals for the assertion should be added or removed from the tracelist. The *<assertion_name>* specifies the full hierarchical name of the assertion.

-assertion -addAll | -rmAll

Specifies that all the assertions should be added or removed from the tracelist.

-assertion -addInst | -rmInst <inst_name>

Specifies that all the assertions in the specified instance /scope should be added or removed from the tracelist. This does not affect the fanin signals.

-captureMode [controlled | uncontrolled]

Sets the capture mode for tracing signals.

If this option is set to controlled, any probes selected on signals that change state in the independent domain suffer from collapsed values due to under sampling.

If this option is set to uncontrolled, any probes selected on signals that change state in the software-controlled domain suffer from long gaps of no changed due to excessive oversampling.

-addinstpin <instance>

Adds all input/outputs of the specified <instance> and all instances within it.

-addPowerDomain <powerDomainName>

Displays PSO status for <powerDomainName>. This command should be executed before running the design or issuing sigTrace -upload.

-addRetention <stateRetentionRule>

Displays the save/restore and on/off status of <stateRetentionRule>. This command should be executed before running the design or issuing sigTrace -upload.

-addIsolation <isolationRuleName>

Displays the isolation status (on or off) of <isolationRuleName>. This command should be executed before running the design or issuing sigTrace -upload.

-addPowerModeTransition

Displays the power modes defined using the create_power_mode and create_nominal_condition commands at the compile time.

**-setupToggleCount -addinst [-top <n>] [-depth <n>] [-min <n>]
<instance_names>**

Specifies the list of module instance names for which toggle count data is to be collected. Following options are used to filter module selection:

- **-top <n>**: Adds only the top n largest instances based on the number of nets. The default setting is top 1000.
- **-depth <n>**: Adds only those child instances that are within n levels below the root instance. The default depth setting is 0, which means that all children at all levels are added.
- **-min <n>**: Adds only those child instances whose size (number of nets contained) is larger than n percent of the specified instance. The default setting is 1 percent.

-setupToggleCount -rminst <instance_names>

Removes the specified module instances and their children from the list of modules set for toggle count data collection.

-setupToggleCount -rminst -all

Removes all the module instances and their children from the list of modules set for toggle count data collection.

-setupToggleCount -listinst [-verbose]

Lists all the module instances that are currently set for toggle count data collection. If the **-verbose** option is specified, the size of each instance is also listed.

-getToggleCount [-append] [-weighted] [-cycle <n>] [-file <name>]

Uploads the toggle count data from the emulator and saves it in the specified data file. The following options are used to control the toggle count data capture:

- **-append**: Appends the new data to the existing data file. By default, the new data overwrites the existing data in the data file.
- **-weighted**: Captures weighted toggle count data instead of normal toggle count data. By default, normal toggle count data is captured.

- **-cycle <n>**: Defines the number of cycles in the upload window, ending at the last cycle of emulation. If this option is not specified, the upload window size is determined by the QEL parameter `qtTraceMemSize`. The maximum value for the upload window size is limited by the maximum trace depth.
- **-file <name>**: Specifies the base name of the data file in which the captured toggle count data is written. The data file is created with the `.ppfdata` extension. If no file name is specified, the current waveform name, which is set by the `sigTrace -outfile` command, is used as the base name.

**-addinstff <instanceName> [-asicLibrary <.llib name> ... <.llib name>]
[-hardmacroLibrary <.llib name> <.llib name>]**

Adds all the pins of the specified module instance to the probe set for generating TCF files that are used by RC to perform power computation for gate-level netlist.

The `-hardmacroLibs` option specifies the liberty libraries (.lib) for hard macro cells.

**-addinstReg <instanceName> [-asicLibs <.libName><.libName>]
[-hardmacroLibs <.libName><.libName>]**

Adds all the pins and registers of the specified module instance to the probe set for generating TCF files that are used by RC to perform power computation for RTL-level netlist.

The `-asicLibs` and `-hardmacroLibs` options specify the liberty libraries (.lib) for the ASIC cells and hard macro cells, respectively.

**-outfile -tcf <filename> [-instance <instanceName>] [-masterTCF]
[-slaveTCF][-noregularTCF] [-segment <n>% | <cycleNumber>]**

Specifies the format of the TCF file, which is used by RC to calculate the power consumption. The `-tcf <filename>` option specifies the name of the TCF file.

To generate a module-based TCF file, use the `-instance <instanceName>` option.

The `-masterTCF` and `-slaveTCF` options are used to generate different types of TCF files.

The `-segment` option specifies the number of segment windows for which toggle counts are calculated.

UXE Command Reference Manual

Deprecated Commands and Variables

-genPowerProfile -segmentLogFile <segmentLogFileName> - reportFileNamePattern <FileNamePattern>

Generates power consumption profile for the design based on the power report files generated by RC. The `-reportFileNamePattern` option specifies the directory where RC-generated power reports are located. The `-segmentLogFile` option specifies the name of the TCF file generation log file that was created by the `sigTrace -outfile -tcf <tcfFileName> -segment <n>%` command.

stimulusWaveform

Opens the Waveform Viewer and displays the contents of the stimulus file.

The syntax for this command is:

```
stimulusWaveform -display  
-quit  
-save <file> |  
-help
```

-display

Loads or reloads vectors and displays the waveform according to the saved display environment.

-quit

Closes the current waveform window.

-save <file>

Saves the waveform display environment to the specified file name. If you do not specify a filename, this command uses the default filename as *<vector>.out.wave*.

-help

Displays help related to the usage of the waveform viewer.

stopDelay

Specifies the delay between the SDL trigger point and the actual breakpoint.

The syntax for this command is:

```
stopDelay [0|1]
```

0

With an argument 0, the emulator stops immediately in the same design cycle as the trigger point. However, clock speed in such case is one-third of the full speed.

1

With an argument 1, the emulator stops with a delay of two design cycles after the trigger point. The default argument is 1.

term

Creates a database object for a terminal in a cell of your design.

Before using this command, execute `design` to specify the design library and cell names for your design.

Also, use `lib` to define a database object for the library containing the cell, and `cell` to define a database object for the cell containing the terminal.

The syntax for this command is:

```
term <termobj> <term> <celfobj>
```

<termobj>

Database object identifier to assign to the terminal.

<term>

Name of the terminal to define as a database object.

<celfobj>

Name of the cell database object containing the terminal, as defined in `cell`.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical r
cell celfobj CLOCK_LAB libobj
term termobj EN celfobj
```

Creates a database object named `libobj` for the `DESIGNS` library, a database object named `celfobj` for the `CLOCK_LAB` cell in that library, and finally a database object named `termobj` for the `EN` terminal in that cell.

termInst

Creates a database object for a terminal instance in an instance within a cell of your design.

This command creates an object in memory that represents an object in the database. Creating an object in memory does not modify the database.

Before using this command, execute `design` to specify the design library and cell names for your design. Also execute `lib` to define a database object for the library containing the cell, `cell` to define a database object for the cell containing the instance, and `inst` to define a database object for the instance containing the terminal instance.

The syntax for this command is:

```
termInst <terminstobj> <terminst> <instobj>
```

<terminstobj>

Database object identifier to assign to the terminal instance.

<terminst>

Name of the terminal instance to define as a database object.

<instobj>

Name of the instance database object containing the terminal instance, as defined in `inst`.

Examples

```
design DESIGNS CLOCK_LAB
lib libobj DESIGNS logical r
cell cellobj CLOCK_LAB libobj
inst instobj FF0 cellobj
termInst terminstobj CD instobj
```

Creates four database objects:

- `libobj` for the `DESIGNS` library.
- `cellobj` for the `CLOCK_LAB` cell in that library.
- `instobj` for the `FF0` instance in that cell.

UXE Command Reference Manual

Deprecated Commands and Variables

- `terminstobj` for the `CD` terminal instance in the `FF0` instance.

traceWaveform

Opens the Waveform Viewer and displays all the probed data that was captured into the trace file `trace.out` (VBF mode) or `trace.fsdb` (FSDB mode).

The syntax for this command is:

```
traceWaveform -display <readback_file> | 0  
             -restore <rc_file> |  
             -update |  
             -quit |  
             -parseSymbols |  
             -save <file>
```

-display <readback_file> | 0

Loads or reloads vectors and displays the waveform according to the saved display environment. The `<readback_file>` specifies the name of the VBF or FSDB trace file. If the argument is specified as 0, this means that no `readback_file` is included.

If the output mode and output file name are not specified, the default mode is taken as VBF mode and filename is assumed to be `trace.out`.

-restore <rc_file>

Restores the TurboDebug waveform signals from the specified `<rc_file>`.

If the output mode and output file name are not specified, the default mode is taken as VBF mode and filename is assumed to be `trace.out`.

-update

Updates the currently open waveform. Typically, this option is used after using the `sigTrace -upload` command.

-quit

Closes the Waveform Viewer.

-parseSymbols

Adds newly defined symbols to the waveform.

-save <file>

Saves the waveform display environment to the specified file.

If you do not specify a file name, this command uses the default file name, `trace.out.wave`.

trEvent

Defines a new Event, modifies or removes an existing Event definition, or returns information about a specific Event for use in SDL programs.

Note: Names of Events, user-defined symbols, and design nets are mutually exclusive.

The syntax for this command is:

```
trEvent [-add <event> = <expression>]  
[-rm <event> ...]  
[-get <event> <symbol>]  
[-save <filename>]
```

-add <event> = <expression>

Defines an event using an expression.

The text following the `-add` keyword has the same syntax as that of the expression alias declaration inside SDL programs.

The `<expression>` part in the Event definition might include user-defined symbols and names of other Events. To avoid conflict between special characters inside the expression and Tcl syntax you might need to escape the expression inside curly braces.

-rm <event> ...

Removes the specified event definition(s).

-get <event> <symbol>

Without the optional `<symbol>` parameter, it returns the value and symbol list for the trace event in the format:

```
<value> <symbol> ...
```

With the `<symbol>` parameter, it returns the value that is associated only with that symbol in the trace event.

This will work only with Events that were defined with the first form of the `-add` option.

-save <filename>

The entire set of events and expression aliases defined through XEL can be saved in the form of XEL commands in the specified file.

To import the definitions of events and expression aliases from the file, source the file using the `source <filename>` command.

The `trEvent` command also supports a few syntax constructs, which are currently deprecated:

```
trEvent [-add <event> <value> <symbol> ...]  
        [-edit <event> <symbol> <value>]
```

-add <event> <value> <symbol> ...

This syntax is still supported for backward compatibility purposes, but its use is deprecated.

Adds events using the specified name and value of the event, and the specified list of symbols.

-edit <event> <symbol> <value>

Selectively changes a portion of the event value. Assigns a new value to the bits that correspond to the `<symbol>` parameter. Other bits in the event value, which correspond to other symbols, remain unchanged.

This will work only with events that were defined with the first form of the `-add` option.

verifyTrigFile

This command verifies the SDL program for syntax and logic errors.

xc uaPollLimit

Helps debug the hang behavior of a behavioral cycle. If a behavioral cycle has entered an infinite loop and, the hardware run will hang with no time advance. To debug this hang behavior, use the following command:

```
xc uaPollLimit <seconds>
```

The *<seconds>* is the number of real-time seconds that the run-time scheduler will wait for the behavior loop. By default, it waits forever. The minimum value for this is 10.

UXE Command Reference Manual

Deprecated Commands and Variables

Index

Symbols

'celldefine [661](#)
'default_nettpe [661](#)
'define [661](#)
'else [661](#)
'endcelldefine [661](#)
'endif [661](#)
'endprotect [442, 598, 661](#)
'ifdef [661](#)
'include [661](#)
'nounconnected_drive [661](#)
'protect [442, 598, 661](#)
'resetall [661](#)
'timescale [661](#)
'unconnected_drive [661](#)
'undef [661](#)
'uselib [661](#)
+assertCover [670](#)
+assertCoverCnt [671](#)
+assertStatus [675](#)
+bc [478](#)
+bufdisp [630](#)
+convertUdp [433](#)
+define [427, 586](#)
+delay [541](#)
+dut [554](#)
+dutignore [555](#)
+hwfrc [633](#)
+inmdir [428, 587](#)
+iscDelay [631](#)
+libext [426, 429, 585, 589](#)
+liborder [430, 590](#)
+librescan [431, 591](#)
+libverbose [432, 592](#)
+loadpli1 [448, 600](#)
+loadvpi [449, 601](#)
+localDiskComp [629](#)
+max_error_count [436, 544](#)
+maxdelays [437, 594](#)
+mindelays [438, 595](#)
+no_psl [681](#)
+no_rtlxmr
 synthesis policy checker [649](#)
+no_sva [669](#)
+nowarn [439, 545](#)

+primchange [447](#)
+protect [442, 598](#)
+rtlCommentPragma [410, 638](#)
+rtlHonorCasePragma [596](#)
+rtlIgnBlkDelay [542](#)
+rtlIgnPragma [557](#)
+rtlIgnTask [445, 599](#)
+rtlKeepDelays [560](#)
 synthesis policy checker [646](#)
+rtlMemsize [561](#)
 synthesis policy checker [647](#)
+rtlVarwidth [563](#)
 synthesis policy checker [648](#)
+salgnDanglingClock [543](#)
+salgnXmrWrite [556](#)
+specblk [450](#)
+sv [634](#)
+sva+ [668](#)
+tfReadOnly [446, 551](#)
+timescale [453](#)
+v1995 [635](#)
+v2000 [418, 636, 643, 644](#)
+vhdlmcx [602](#)
+xsimprotect [598](#)
\$bitstoreal [660](#)
\$dist_uniform [660](#)
\$export_deposit() [653](#)
\$export_frcrel() [656](#)
\$export_read() [658](#)
\$getpattern [660](#)
\$initmem [655](#)
\$itor [660](#)
\$realtime [660](#)
\$realtobits [660](#)
\$recrrem [660](#)
\$rtoi [660](#)
\$stime [660](#)
\$time [660](#)
\$timeformat [660](#)
\$xc_cmd() [652](#)

Numerics

-200x [391](#)
-32 [380](#)

-32 | -64 [380](#)
-64 [380](#), [482](#)
-87 [390](#)
-93 [390](#)

A

adding
 events [1246](#), [1247](#)
 symbols to waveforms [1245](#)
alphabetical command list [75](#)
 common XEL commands [92](#)
 xeDebug commands [89](#)
-append_log [408](#), [516](#)
ASIC libraries [162](#)
assert
 illegal domain configurations [721](#)
assertions
 states [782](#), [783](#)
 summary [786](#)
assigning
 terminals to connectors [345](#)
attribute files [167](#)
attributes
 cells [167](#)
 database objects [1175](#)
axis_tbcall [664](#)

B

beginSyntax [141](#)
binary [804](#)
boards
 emulatorConfiguration option [265](#)
breakNet [210](#)

C

-cIC [381](#)
cableConnection [217](#)
capturing data [1201](#)
capturing samples
 trace memory [1074](#)
case sensitivity
 import [161](#)
cells
 attributes [167](#)
 database objects [1177](#)

ignoring definitions [161](#)
names [705](#)
primitive [162](#)
properties [167](#)
top [705](#)
cells, mapping to IP [372](#)
-clean [468](#)
clock delay
 specifying with XEL [794](#)
clock frequency
 specifying with XEL [225](#)
clock options
 changing [794](#)
closing
 library objects [1181](#)
command list [75](#), [89](#), [92](#)
command scripts
 XEL [1161](#)
commands
 event signals [1202](#)
 exiting XEL [148](#), [151](#)
 logic analyzer [1202](#)
 quitting XEL [148](#), [151](#)
 trigger signals [1202](#)
 XEL [75](#)
common XEL commands [92](#)
compiler commands [683](#)
compiling designs [686](#)
connectors
 assigning to terminals [345](#)
converting radix [804](#)
-coverage functional [677](#)
CPF version [752](#)
create
 isolation rule [722](#)
 mode [726](#)
 mode transition [727](#)
 nominal operating condition [728](#)
 power domains [730](#)
 power mode [737](#)
 state retention rule [740](#)
creating XEL scripts [1161](#)
cross-module reference (XMR) [649](#)
current library
 ignoring [161](#)
current value [782](#)
cycle number
 design verification [854](#)

D

data
 capturing [1201](#)
data types
 user data [368](#)
database
 closing library objects [1181](#)
 library objects [1176](#)
 listing attributes
 attributes
 listing [1176](#)
 saving libraries [1224](#)
database commands [197](#)
database name [1041](#)
database objects
 attributes [1175](#)
 cells [1177](#)
 instances [1177, 1205](#)
 libraries [1206](#)
 nets [1178, 1207](#)
 terminal instances [1179, 1242](#)
 terminals [1178, 1241](#)
database path [1041](#)
debugger
 capturing data [1201](#)
 logic analyzer mode [802](#)
 vector debug mode [803](#)
debugger commands [763](#)
decimal [804](#)
-default_ext [454](#)
define
 design scope [745](#)
 macro model [746, 760](#)
define design scope [753](#)
delayAsynClockInputs [256](#)
delayBox [258](#)
delayBoxInput [261](#)
deleting signal symbols [1014](#)
design database
 name [1041](#)
 path [1041](#)
design library [162](#)
 import [162](#)
design name [1043](#)
design verification
 cycle number [854](#)
designs
 compiling [686](#)
 downloading [819](#)

importing HDL Compiler [196](#)
specifying names [705](#)
Verilog [171](#)
VHDL [171](#)
directories
 /QTDB/ [165](#)
 <install_dir>/qtlb [165](#)
design database [857 to 929, 966, 1015, 1029 to 1033](#)
emulation design directory [705 to 706](#)
searching Verilog files [162](#)
Verilog [162](#)
directory path
 database [1041](#)
disabled [782, 783](#)
downloading to emulator [819](#)
dumping (see saving)

E

emulation
 compiling [686](#)
 logic analyzer mode [802](#)
 restarting [956](#)
 synthesis [1167](#)
 trigger by state [1216, 1217, 1218](#)
 trigger position [1075](#)
 vector debug mode [803](#)
emulation design database
 name [1041](#)
 path [1041](#)
emulation design name [1043](#)
emulation state
 saving [967, 1225](#)
emulator
 downloading designs [819](#)
 host name [1054](#)
 selecting [866](#)
 status [855](#)
emulatorConfiguration [264](#)
enabling assertions [784, 786](#)
endSyntax [141](#)
escape character
 Tcl [72](#)
event
 define an [1246](#)
events
 adding [1246, 1247](#)
 current values [1246](#)
 setting values [1247](#)

examples

XEL scripts [1164](#)
exiting XEL [148, 151](#)
export_frcrel() [656](#)
export_read() [658](#)

F

-F [532](#)
-f [532](#)
 vhan [389](#)
-flF
 vlan [416](#)
failed [782](#)
files
 .llib [165](#)
 .xerc [70](#)
 <cell>.et3config [264](#)
 <vector>.out.wave [1239](#)
 attribute [167](#)
 countermem.ptn [1166](#)
 foo.et3config [267](#)
 importing netlists [167](#)
 jpeg.test [1041](#)
 myvector.cbc [956, 966](#)
 qtref.llib [166](#)
 trigger state [1219](#)
 triggerFile.tdf [1168](#)
 user data [366](#)
 vectors [1079](#)
 Verilog [162](#)
 xe.msg [71](#)
 xeCompile.log [688](#)
 xelrc [70](#)
find
 design objects [747](#)
finished [782](#)
force
 Verilog [649](#)
full import [160](#)
full_case [596, 597](#)

G

general-purpose commands [147](#)
generic library [162](#)
get
 parameter [750](#)
ground

nets [301, 364](#)
groups
 signals [1203](#)

H

-h
 vlan [414](#)
-hIH
 vhan [388](#)
HDL
 output files [184](#)
 synthesis [189](#)
HDL Compilation commands [171](#)
-help [517](#)
help command [149](#)
hexadecimal [804](#)
host emulator name [866](#)
host name
 emulator [1054](#)
host status
 emulator [855](#)

I

IEEE 1364-1995/2001 (for Verilog
 HDL) [660, 661](#)
IEEE 1364-2001 (for Verilog HDL) [636](#)
IEEE Std 1076-1987 for VHDL [390](#)
IEEE Std 1076-1993 for VHDL [390, 391](#)
IEEE Std 1076-2000 for VHDL [391](#)
IEEE Std 1076-2008 for VHDL [390](#)
IEEE Std 1364-2001 for Verilog HDL [418](#)
-ignore_pragma [409](#)
ignoring
 cell definitions [161](#)
import
 case sensitivity [161](#)
 design library [162](#)
 full [160](#)
 incremental [160](#)
 library type [162](#)
 no search [161](#)
import commands [155](#)
import options
 saving [158](#)
importing designs
 working library [196](#)
Incisive Comprehensive Coverage [677](#)

UXE Command Reference Manual

include [751](#)
include files
 Verilog [162](#)
incremental import [160](#)
initial block [1126](#)
initializing
 XEL commands [70](#)
input stimulus [1079](#)
 waveforms [1239](#)
instances
 database objects [1177, 1205](#)
ixc_pulse [663](#)
IXCOM [464](#)
IXCOM_UXE
 compiler directive [650](#)

J

-just [381](#)

K

keywords
 user data [368](#)

L

-l
 vhan [398](#)
 vlan [420, 540](#)
libraries
 names [705](#)
-libmaprc [395, 419](#)
libraries
 ASIC [162](#)
 database objects [1206](#)
 design [162](#)
 generic [162](#)
 import [162](#)
 reference [162](#)
 saving in database [1224](#)
library objects
 closing [1181](#)
 database [1176](#)
library type
 import [162](#)
list of assertions [787](#)
listing

signals in symbols [1014](#)
loading
 user data [366](#)
logic analyzer
 trigger position [1075](#)
logic analyzer emulation
 XEL script example [1168](#)
logic analyzer mode
 emulation [802](#)

M

memory
 trace [1074](#)
 writing [902, 911](#)
-mti
 vhan [399](#)

N

name
 database [1041](#)
names
 library [705](#)
 top cell [705](#)
-nccompile [523](#)
NCSim [400, 522](#)
-ncsim [522](#)
-ncsim (vhan) [400](#)
netlists
 case sensitivity [161](#)
 importing [155, 158, 160, 167](#)
 selecting for import [167](#)
 synthesizing [189](#)
nets
 assigning probes to [336](#)
 database objects [1178, 1207](#)
 ground [301, 364](#)
 power [301, 364](#)
no search
 import [161](#)
-nocasestaticerror [401](#)

O

objects
 cells [1177](#)
 database [1175](#)

instances [1177, 1205](#)
libraries [1176, 1181, 1206](#)
listing attributes [1176](#)
nets [1178, 1207](#)
terminal instances [1179, 1242](#)
terminals [1178, 1241](#)
octal [804](#)
online help [149](#)
options
 import [158](#)
output
 user data [366](#)
output files
 HDL Compilation [184](#)
overview
 XEL [69](#)

P

-P [443, 549](#)
-p [444, 550](#)
parallel_case [596, 597](#)
path
 database [1041](#)
pins
 assigning [345](#)
 assigning interface [345](#)
POCLK [1135](#)
power
 nets [301, 364](#)
power consumption profile [846](#)
pragma
 case statement [596](#)
Primary Output Clock [1135](#)
primitive cells [162](#)
programs
 SDL [1219](#)
 utilities [101](#)
programs and utilities [76](#)
 compiler programs and utilities [76](#)
 run-time programs and utilities [77](#)
properties
 cells [167](#)
-profile [407](#)
-psl [402, 403, 404, 405, 406](#)

Q

-q [421, 564](#)

vlan [421](#)
-qlQ
 vhan [385](#)
quitting XEL [148, 151](#)

R

radix
 converting [804](#)
reference library [162](#)
release
 Verilog [649](#)
reporting assertions [783, 786](#)
reserved characters
 Tcl [72](#)
restarting
 emulation [956](#)
result waveforms [961](#)
RTL designs
 Verilog [171](#)
 VHDL [171](#)
run-time parameters
 XEL [91](#)

S

-sIS
 vhan [394](#)
samples
 number to trace [1074](#)
saving
 import options [158](#)
 user data [366](#)
 vector files [1079](#)
 waveforms [1239, 1245](#)
saving libraries [1224](#)
scripts
 XEL [1161](#)
SDL programs [1219](#)
search directory
 Verilog files [162](#)
selecting
 importing netlists [167](#)
selecting emulator [866](#)
sensitivity
 case [161](#)
set
 hierarchy separator [756](#)
 instance [757](#)

UXE Command Reference Manual

setting
 events values [1247](#)
Signal Analysis (SA) [446](#), [551](#)
signal symbols
 deleting [1014](#)
signals
 groups [1203](#)
 listing in symbols [1014](#)
-skip [383](#)
software version [1080](#)
standard output
 user data [366](#)
state
 trigger file [1219](#)
 triggering [1216](#), [1217](#), [1218](#)
states [782](#)
status
 emulator [855](#)
stimulus
 input [1079](#)
stimulus waveforms [1239](#)
-strictLRM [637](#)
summary report [787](#), [788](#)
symbols
 adding to waveforms [1245](#)
 deleting [1014](#)
 listing signals [1014](#)
synthesis
 HDL Compilation [189](#)
 XEL script example [1167](#)
synthesis_off [557](#)
System Tasks and Procedures [650](#)
-sysv_ext [456](#)

T

target connection
 specifying [345](#)
target system
 ICE prepare [709](#)
Tcl [70](#)
 escape character [72](#)
 references, books [73](#)
 references, online [73](#)
 reserved characters [72](#)
terminal instances
 database objects [1179](#), [1242](#)
terminals
 assigning to connectors [345](#)
 database objects [1178](#), [1241](#)

tie nets [301](#), [364](#)
-timescale [632](#)
Tool Command Language (Tcl) [70](#)
-top [475](#), [518](#)
top cell name [705](#)
trace memory
 capturing samples [1074](#)
trace waveforms [1244](#)
traced signals
 uploading [1144](#), [1230](#)
tracing
 number of samples [1074](#)
tran [447](#)
translate_off [410](#), [557](#), [638](#)
translate_on [410](#), [557](#), [638](#)
trigger position
 logic analyzer [1075](#)
trigger state file [1219](#)
triggering by state [1216](#), [1217](#), [1218](#)

U

-u [422](#), [565](#), [566](#)
-ua [552](#), [553](#)
uploading
 traced signals [1144](#), [1230](#)
user data
 files [366](#)
 keywords [368](#)
 loading [366](#)
 saving [366](#)
 standard output [366](#)
user data commands [205](#)
-uservars [969](#)

V

-v [423](#), [583](#)
-vIV
 vhan [393](#)
values
 events [1246](#), [1247](#)
vector debug
 XEL script example [1167](#)
vector debug mode
 emulation [803](#)
vector files
 saving [1079](#)
Verilog

include files [162](#)
RTL designs [171](#)
Verilog files
 search directory [162](#)
-version
 vhan [397](#)
vhan.log [398](#)
VHDL
 RTL designs [171](#)
 working library [196](#)
vlan.log [420, 540](#)
-vlog_ext [458](#)
-vlog95_ext [457](#)
-vtimescale [440](#)

W

-W [424](#)
-w [425](#)
-w|W
 vhan [386](#)
waveform viewer [961, 1239, 1244](#)
waveforms
 adding symbols [1245](#)
 saving [1239, 1245](#)
-work
 vlan [415](#)
-work (library) [518](#)
working library
 importing designs [196](#)
 VHDL [196](#)
writing memory [902, 911](#)

X

xc checkxoff [1089](#)
xc checkxon [1090](#)
xc free [1110](#)
xc help [1113](#)
xc off [1125](#)
xc on -autorun [1090](#)
xc on -match [1090](#)
xc on -run [1090](#)
xc on -tbrun [1090](#)
xc on -xt0 [1090](#)
xc on -xt1 [1090](#)
xc poclk [1132](#)
xc poclk add [1133](#)
xc poclk rm [1135](#)

xc showcpo [1141](#)
xc showfrc [1142](#)
xc signal [1143](#)
xc status [1148](#)
xc stop [1149](#)
xc verbose [1151](#)
xc wait [1152](#)
xc xrun [1155](#)
xeDebug commands
 alphabetical list [89](#)
XEL
 .xerc file [70](#)
 accessing the design database [68](#)
 accessing user data [68](#)
 commands [75](#)
 compiling designs [68](#)
 exiting [148, 151](#)
 help [149](#)
 initializing XEL [70](#)
 online help [149](#)
 overview [69](#)
 quitting [148, 151](#)
 RTL mapping [68](#)
 run-time parameters [91](#)
 script, examples [1164](#) to ??
 Tcl functionality [70](#)
 xeDebug
 Simulation Acceleration-specific run-time command list [79](#)

XEL commands
 alphabetical list [75](#)
 changing data [71](#)
 changing user data [71](#)
 compiler [683](#)
 database [197](#)
 debugger [763](#)
 event signals [1202](#)
 general-purpose [147](#)
 HDL Compilation [171](#)
 import [155](#)
 logic analyzer [1202](#)
 online help files [70](#)
 script files [72](#)
 shell window [70](#)
 trigger signals [1202](#)
 user data [205](#)
 XEL log file [70](#)
XEL scripts
 creating [1161](#)
 examples [1164](#)
 logic analyzer [1168](#)

UXE Command Reference Manual

xerc file 70

Y

-y 426, 585

Z

-Z 435, 593
-z 584

UXE Command Reference Manual
