

## List of Code Samples

Sample 1-1	Driving the APB pins	16
Sample 1-2	A task to drive the APB pins	17
Sample 1-3	Low-level Verilog test	17
Sample 1-4	Basic transactor code	21
Sample 2-1	Using the logic type	26
Sample 2-2	Signed data types	27
Sample 2-3	Checking for 4-state values	27
Sample 2-4	Declaring fixed-size arrays	28
Sample 2-5	Declaring and using multidimensional arrays	28
Sample 2-6	Unpacked array declarations	29
Sample 2-7	Initializing an array	29
Sample 2-8	Using arrays with for- and foreach-loops	30
Sample 2-9	Initialize and step through a multidimensional array	30
Sample 2-10	Output from printing multidimensional array values	30
Sample 2-11	Printing a multidimensional array	31
Sample 2-12	Output from printing multidimensional array values	31
Sample 2-13	Array copy and compare operations	32
Sample 2-14	Using word and bit subscripts together	32
Sample 2-15	Packed array declaration and usage	33
Sample 2-16	Declaration for a mixed packed/unpacked array	33
Sample 2-17	Using dynamic arrays	35
Sample 2-18	Using a dynamic array for an uncounted list	35
Sample 2-19	Queue operations	37
Sample 2-20	Queue operations	37
Sample 2-21	Declaring, initializing, and using associative arrays	39
Sample 2-22	Using an associative array with a string index	40
Sample 2-23	Creating the sum of an array	41
Sample 2-24	Picking a random element from an associative array	42
Sample 2-25	Array locator methods: min, max, unique	43
Sample 2-26	Array locator methods: find	43
Sample 2-27	Declaring the iterator argument	43

Sample 2-28	Array locator methods	44
Sample 2-29	Sorting an array	44
Sample 2-30	Sorting an array of structures	45
Sample 2-31	A scoreboard with array methods	45
Sample 2-32	User-defined type-macro in Verilog	49
Sample 2-33	User-defined type in SystemVerilog	49
Sample 2-34	Definition of uint	49
Sample 2-35	User-defined array type	50
Sample 2-36	Creating a single pixel type	50
Sample 2-37	The pixel struct	50
Sample 2-38	Initializing a struct	51
Sample 2-39	Using typedef to create a union	51
Sample 2-40	Packed structure	52
Sample 2-41	Converting between int and real with static cast	53
Sample 2-42	Basic streaming operator	53
Sample 2-43	Converting between queues with streaming operator	54
Sample 2-44	Converting between a structure and array with streaming operators	55
Sample 2-45	A simple enumerated type	55
Sample 2-46	Enumerated types	56
Sample 2-47	Specifying enumerated values	56
Sample 2-48	Incorrectly specifying enumerated values	57
Sample 2-49	Correctly specifying enumerated values	57
Sample 2-50	Stepping through all enumerated members	58
Sample 2-51	Assignments between integers and enumerated types	58
Sample 2-52	Declaring a const variable	59
Sample 2-53	String methods	60
Sample 2-54	Expression width depends on context	61
Sample 3-1	New procedural statements and operators	64
Sample 3-2	Using break and continue while reading a file	64
Sample 3-3	Void function for debug	65
Sample 3-4	Ignoring a function's return value	65
Sample 3-5	Simple task without begin...end	66
Sample 3-6	Verilog-1995 routine arguments	66
Sample 3-7	C-style routine arguments	66
Sample 3-8	Verbose Verilog-style routine arguments	67
Sample 3-9	Routine arguments with sticky types	67
Sample 3-10	Passing arrays using ref and const	68
Sample 3-11	Using ref across threads	69
Sample 3-12	Function with default argument values	70
Sample 3-13	Using default argument values	70
Sample 3-14	Binding arguments by name	71
Sample 3-15	Original task header	71
Sample 3-16	Task header with additional array argument	71
Sample 3-17	Task header with additional array argument	71
Sample 3-18	Return in a task	72

Sample 3-19	Return in a function	72
Sample 3-20	Returning an array from a function with a typedef	73
Sample 3-21	Passing an array to a function as a ref argument	73
Sample 3-22	Specifying automatic storage in program blocks	74
Sample 3-23	Static initialization bug	75
Sample 3-24	Static initialization fix: use automatic	75
Sample 3-25	Static initialization fix: break apart declaration and initialization	75
Sample 3-26	Time literals and \$timeformat	76
Sample 3-27	Time variables and rounding	77
Sample 4-1	Arbiter model using ports	81
Sample 4-2	Testbench using ports	82
Sample 4-3	Top-level netlist without an interface	82
Sample 4-4	Simple interface for arbiter	83
Sample 4-5	Arbiter using a simple interface	83
Sample 4-6	Testbench using a simple arbiter interface	84
Sample 4-7	Top module using a simple arbiter interface	84
Sample 4-8	Bad test module includes interface	85
Sample 4-9	Connecting an interface to a module that uses ports	85
Sample 4-10	Interface with modports	86
Sample 4-11	Arbiter model with interface using modports	86
Sample 4-12	Testbench with interface using modports	86
Sample 4-13	Arbiter model with interface using modports	87
Sample 4-14	Interface with a clocking block	90
Sample 4-15	Interface with a clocking block	91
Sample 4-16	Race condition between testbench and design	93
Sample 4-17	Testbench using interface with clocking block	95
Sample 4-18	Signal synchronization	97
Sample 4-19	Synchronous interface sample and drive from module	97
Sample 4-20	Testbench using interface with clocking block	98
Sample 4-21	Interface signal drive	99
Sample 4-22	Driving a synchronous interface	99
Sample 4-23	Interface signal drive	100
Sample 4-24	Bidirectional signals in a program and interface	101
Sample 4-25	Bad clock generator in program block	102
Sample 4-26	Good clock generator in module	103
Sample 4-27	Top module using a simple arbiter interface	103
Sample 4-28	Module with just port connections	103
Sample 4-29	Module with an interface	104
Sample 4-30	Top module connecting DUT and interface	104
Sample 4-31	Top-level scope for arbiter design	105
Sample 4-32	Cross-module references with \$root	106
Sample 4-33	Checking a signal with an if-statement	107
Sample 4-34	Simple immediate assertion	107
Sample 4-35	Error from failed immediate assertion	107
Sample 4-36	Creating a custom error message in an immediate assertion	108

Sample 4-37	Error from failed immediate assertion	108
Sample 4-38	Creating a custom error message	108
Sample 4-39	Concurrent assertion to check for X/Z	109
Sample 4-40	ATM router model header without an interface	111
Sample 4-41	Top-level netlist without an interface	112
Sample 4-42	Verilog-1995 testbench using ports	113
Sample 4-43	Rx interface	115
Sample 4-44	Tx interface	115
Sample 4-45	ATM router model with interface using modports	116
Sample 4-46	Top-level netlist with interface	116
Sample 4-47	Testbench using an interface with a clocking block	117
Sample 4-48	A <b>final</b> block	118
Sample 4-49	Fetch block Verilog code	120
Sample 4-50	Fetch block interface	121
Sample 4-51	Fetch block directed test	122
Sample 4-52	Top level block for fetch testbench	124
Sample 5-1	Simple transaction class	127
Sample 5-2	Declaring and using a handle	129
Sample 5-3	Simple user-defined new() function	130
Sample 5-4	A new() function with arguments	130
Sample 5-5	Calling the right new() function	131
Sample 5-6	Allocating multiple objects	132
Sample 5-7	Creating multiple objects	133
Sample 5-8	Using variables and routines in an object	134
Sample 5-9	Class with a static variable	135
Sample 5-10	The class scope resolution operator	136
Sample 5-11	Static storage for a handle	137
Sample 5-12	Static method displays static variable	138
Sample 5-13	Routines in the class	139
Sample 5-14	Out-of-block method declarations	140
Sample 5-15	Out-of-body task missing class name	141
Sample 5-16	Name scope	142
Sample 5-17	Class uses wrong variable	143
Sample 5-18	Move class into package to find bug	143
Sample 5-19	Using this to refer to class variable	144
Sample 5-20	Statistics class declaration	145
Sample 5-21	Encapsulating the Statistics class	145
Sample 5-22	Using a typedef class statement	146
Sample 5-23	Passing objects	148
Sample 5-24	Bad transaction creator task, missing ref on handle	149
Sample 5-25	Good transaction creator task with ref on handle	149
Sample 5-26	Bad generator creates only one object	149
Sample 5-27	Good generator creates many objects	150
Sample 5-28	Using an array of handles	150
Sample 5-29	Copying a simple class with new	151

Sample 5-30	Copying a complex class with new operator	152
Sample 5-31	Simple class with copy function	153
Sample 5-32	Using a copy function	153
Sample 5-33	Complex class with deep copy function	154
Sample 5-34	Statistics class declaration	154
Sample 5-35	Copying a complex class with new operator	155
Sample 5-36	Transaction class with pack and unpack functions	156
Sample 5-37	Using the pack and unpack functions	156
Sample 5-38	Basic Transactor	159
Sample 6-1	Simple random class	165
Sample 6-2	Constraint without random variables	167
Sample 6-3	Constrained-random class	168
Sample 6-4	Bad ordering constraint	168
Sample 6-5	Result from incorrect ordering constraint	169
Sample 6-6	Constrain variables to be in a fixed order	169
Sample 6-7	Weighted random distribution with dist	170
Sample 6-8	Dynamically changing distribution weights	170
Sample 6-9	Random sets of values	171
Sample 6-10	Specifying minimum and maximum range with \$	171
Sample 6-11	Inverted random set constraint	171
Sample 6-12	Random set constraint for an array	172
Sample 6-13	Equivalent set of constraints	172
Sample 6-14	Repeated values in inside constraint	173
Sample 6-15	Output from inside constraint operator and weighted array	173
Sample 6-16	Class to choose from an array of possible values	174
Sample 6-17	Choosing from an array of values	174
Sample 6-18	Using randc to choose array values in random order	175
Sample 6-19	Constraint block with implication operator	176
Sample 6-20	Constraint block with if-else operator	176
Sample 6-21	Bidirectional constraint	176
Sample 6-22	Expensive constraint with mod and unsized variable	177
Sample 6-23	Efficient constraint with bit extract	178
Sample 6-24	Class Unconstrained	178
Sample 6-25	Class with implication	179
Sample 6-26	Class with implication and constraint	180
Sample 6-27	Class with implication and solve...before	181
Sample 6-28	Using constraint_mode	183
Sample 6-29	Checking write length with a valid constraint	183
Sample 6-30	The randomize() with statement	184
Sample 6-31	Building a bathtub distribution	186
Sample 6-32	\$random_range usage	187
Sample 6-33	Constraint with a variable bound	188
Sample 6-34	dist constraint with variable weights	188
Sample 6-35	rand_mode disables randomization of variables	189
Sample 6-36	Randomizing a subset of variables in a class	190

Sample 6-37	Using the implication constraint as a case statement	191
Sample 6-38	Turning constraints on and off with <code>constraint_mode</code>	191
Sample 6-39	Class with an external constraint	192
Sample 6-40	Program defining an external constraint	193
Sample 6-41	Signed variables cause randomization problems	194
Sample 6-42	Randomizing unsigned 32-bit variables	194
Sample 6-43	Randomizing unsigned 8-bit variables	194
Sample 6-44	Constraining dynamic array size	195
Sample 6-45	Random strobe pattern class	197
Sample 6-46	First attempt at sum constraint: <code>bad_sum1</code>	198
Sample 6-47	Program to try constraint with array sum	198
Sample 6-48	Output from <code>bad_sum1</code>	198
Sample 6-49	Second attempt at sum constraint: <code>bad_sum2</code>	198
Sample 6-50	Output from <code>bad_sum2</code>	199
Sample 6-51	Third attempt at sum constraint: <code>bad_sum3</code>	199
Sample 6-52	Output from <code>bad_sum3</code>	199
Sample 6-53	Fourth attempt at sum constraint: <code>bad_sum4</code>	199
Sample 6-54	Output from <code>bad_sum4</code>	200
Sample 6-55	Simple foreach constraint: <code>good_sum5</code>	200
Sample 6-56	Output from <code>good_sum5</code>	200
Sample 6-57	Creating ascending array values with foreach	201
Sample 6-58	Creating unique array values with foreach	201
Sample 6-59	Creating unique array values with a <code>randc</code> helper class	202
Sample 6-60	Unique value generator	202
Sample 6-61	Class to generate a random array of unique values	203
Sample 6-62	Using the <code>UniqueArray</code> class	203
Sample 6-63	Constructing elements in a random array	204
Sample 6-64	Command generator using <code>randsequence</code>	205
Sample 6-65	Random control with <code>randcase</code> and <code>\$random_range</code>	207
Sample 6-66	Equivalent constrained class	208
Sample 6-67	Creating a decision tree with <code>randcase</code>	209
Sample 6-68	Simple pseudorandom number generator	210
Sample 6-69	Test code before modification	212
Sample 6-70	Test code after modification	212
Sample 6-71	Ethernet switch configuration class	213
Sample 6-72	Building environment with random configuration	214
Sample 6-73	Simple test using random configuration	215
Sample 6-74	Simple test that overrides random configuration	215
Sample 7-1	Interaction of <code>begin...end</code> and <code>fork...join</code>	219
Sample 7-2	Output from <code>begin...end</code> and <code>fork...join</code>	220
Sample 7-3	<code>Fork...join_none</code> code	221
Sample 7-4	<code>Fork...join_none</code> output	221
Sample 7-5	<code>Fork...join_any</code> code	222
Sample 7-6	Output from <code>fork...join_any</code>	222
Sample 7-7	Generator / Driver class with a run task	223

Sample 7-8	Dynamic thread creation	224
Sample 7-9	Bad fork...join_none inside a loop	225
Sample 7-10	Execution of bad fork...join_none inside a loop	225
Sample 7-11	Automatic variables in a fork...join_none	226
Sample 7-12	Steps in executing automatic variable code	226
Sample 7-13	Automatic variables in a fork...join_none	227
Sample 7-14	Using wait fork to wait for child threads	227
Sample 7-15	Bug using shared program variable	228
Sample 7-16	Disabling a thread	229
Sample 7-17	Limiting the scope of a disable fork	230
Sample 7-18	Using disable label to stop threads	231
Sample 7-19	Using disable label to stop a task	232
Sample 7-20	Blocking on an event in Verilog	233
Sample 7-21	Output from blocking on an event	234
Sample 7-22	Waiting for an event	234
Sample 7-23	Output from waiting for an event	234
Sample 7-24	Waiting on event causes a zero delay loop	235
Sample 7-25	Waiting for an edge on an event	235
Sample 7-26	Passing an event into a constructor	236
Sample 7-27	Waiting for multiple threads with wait fork	237
Sample 7-28	Waiting for multiple threads by counting triggers	237
Sample 7-29	Waiting for multiple threads using a thread count	238
Sample 7-30	Semaphores controlling access to hardware resource	239
Sample 7-31	Bad generator creates only one object	241
Sample 7-32	Good generator creates many objects	242
Sample 7-33	Good driver receives transactions from mailbox	243
Sample 7-34	Exchanging objects using a mailbox: the Generator class	243
Sample 7-35	Exchanging objects using a mailbox: the Driver class	244
Sample 7-36	Exchanging objects using a mailbox: the program block	244
Sample 7-37	Bounded mailbox	245
Sample 7-38	Output from bounded mailbox	246
Sample 7-39	Producer–consumer without synchronization	247
Sample 7-40	Producer–consumer without synchronization output	248
Sample 7-41	Producer–consumer synchronized with bounded mailbox	249
Sample 7-42	Output from producer–consumer with bounded mailbox	249
Sample 7-43	Producer–consumer synchronized with an event	250
Sample 7-44	Producer–consumer synchronized with an event, continued	251
Sample 7-45	Output from producer–consumer with event	251
Sample 7-46	Producer–consumer synchronized with a mailbox	252
Sample 7-47	Output from producer–consumer with mailbox	253
Sample 7-48	Basic Transactor	254
Sample 7-49	Configuration class	255
Sample 7-50	Environment class	255
Sample 7-51	Basic test program	257
Sample 8-1	Base Transaction class	261

Sample 8-2	Extended Transaction class	262
Sample 8-3	Constructor with argument in an extended class	263
Sample 8-4	Driver class	264
Sample 8-5	Generator class	265
Sample 8-6	Generator class using blueprint pattern	267
Sample 8-7	Environment class	268
Sample 8-8	Simple test program using environment defaults	268
Sample 8-9	Injecting an extended transaction into testbench	269
Sample 8-10	Using inheritance to add a constraint	270
Sample 8-11	Base and extended class	271
Sample 8-12	Copying extended handle to base handle	271
Sample 8-13	Copying a base handle to an extended handle	272
Sample 8-14	Using \$cast to copy handles	272
Sample 8-15	Transaction and BadTr classes	273
Sample 8-16	Calling class methods	273
Sample 8-17	Building an Ethernet frame with composition	276
Sample 8-18	Building an Ethernet frame with inheritance	277
Sample 8-19	Building a flat Ethernet frame	278
Sample 8-20	Base transaction class with a virtual copy function	279
Sample 8-21	Extended transaction class with virtual copy method	279
Sample 8-22	Base transaction class with copy_data function	280
Sample 8-23	Extended transaction class with copy_data function	281
Sample 8-24	Base transaction class with copy function	281
Sample 8-25	Extended transaction class with new copy function	282
Sample 8-26	Abstract class with pure virtual methods	283
Sample 8-27	Transaction class extends abstract class	283
Sample 8-28	Bodies for Transaction methods	284
Sample 8-29	Base callback class	286
Sample 8-30	Driver class with callbacks	286
Sample 8-31	Test using a callback for error injection	287
Sample 8-32	Simple scoreboard for atomic transactions	288
Sample 8-33	Test using callback for scoreboard	289
Sample 8-34	Stack using the int type	290
Sample 8-35	Parameterized class for a stack	291
Sample 8-36	Using the parameterized stack class	291
Sample 8-37	Parameterized generator class using blueprint pattern	292
Sample 8-38	Simple testbench using parameterized generator class	292
Sample 9-1	Incomplete D-flip flop model missing a path	299
Sample 9-2	Functional coverage of a simple object	303
Sample 9-3	Coverage report for a simple object	304
Sample 9-4	Coverage report for a simple object, 100% coverage	305
Sample 9-5	Functional coverage inside a class	307
Sample 9-6	Test using functional coverage callback	308
Sample 9-7	Callback for functional coverage	309
Sample 9-8	Cover group with a trigger	309

Sample 9-9	Module with SystemVerilog Assertion	309
Sample 9-10	Triggering a cover group with an SVA	310
Sample 9-11	Using auto_bin_max set to 2	311
Sample 9-12	Report with auto_bin_max set to 2	311
Sample 9-13	Using auto_bin_max for all cover points	312
Sample 9-14	Using an expression in a cover point	312
Sample 9-15	Defining bins for transaction length	313
Sample 9-16	Coverage report for transaction length	313
Sample 9-17	Specifying bin names	314
Sample 9-18	Report showing bin names	314
Sample 9-19	Specifying ranges with \$	315
Sample 9-20	Conditional coverage — disable during reset	315
Sample 9-21	Using stop and start functions	316
Sample 9-22	Functional coverage for an enumerated type	316
Sample 9-23	Coverage report with enumerated types	316
Sample 9-24	Specifying transitions for a cover point	317
Sample 9-25	Wildcard bins for a cover point	317
Sample 9-26	Cover point with ignore_bins	318
Sample 9-27	Cover point with auto_bin_max and ignore_bins	318
Sample 9-28	Cover point with illegal_bins	318
Sample 9-29	Basic cross coverage	320
Sample 9-30	Coverage summary report for basic cross coverage	320
Sample 9-31	Specifying cross coverage bin names	321
Sample 9-32	Cross coverage report with labeled bins	322
Sample 9-33	Excluding bins from cross coverage	322
Sample 9-34	Specifying cross coverage weight	323
Sample 9-35	Cross coverage with bin names	324
Sample 9-36	Cross coverage with binsof	325
Sample 9-37	Mimicking cross coverage with concatenation	325
Sample 9-38	Simple argument	326
Sample 9-39	Pass-by-reference	326
Sample 9-40	Specifying per-instance coverage	327
Sample 9-41	Specifying comments for a cover group	328
Sample 9-42	Specifying comments for a cover group instance	328
Sample 9-43	Report all bins including empty ones	329
Sample 9-44	Specifying the coverage goal	329
Sample 9-45	Original class for transaction length	330
Sample 9-46	solve...before constraint for transaction length	330
Sample 10-1	Rx interface with clocking block	334
Sample 10-2	Tx interface with clocking block	334
Sample 10-3	Testbench using physical interfaces	335
Sample 10-4	Top level module with array of interfaces	336
Sample 10-5	Testbench using virtual interfaces	337
Sample 10-6	Testbench using virtual interfaces	337
Sample 10-7	Driver class using virtual interfaces	338

Sample 10-8	Test harness using an interface in the port list	340
Sample 10-9	Test with an interface in the port list	340
Sample 10-10	Top module with a second interface in the test's port list	340
Sample 10-11	Test with two interfaces in the port list	340
Sample 10-12	Test with virtual interface and XMR	341
Sample 10-13	Test harness without interfaces in the port list	341
Sample 10-14	Test harness with a second interface	341
Sample 10-15	Test with two virtual interfaces and XMRs	341
Sample 10-16	Interface for 8-bit counter	342
Sample 10-17	Counter model using X_if interface	343
Sample 10-18	Testbench using an array of virtual interfaces	343
Sample 10-19	Counter testbench using virtual interfaces	344
Sample 10-20	Driver class using virtual interfaces	345
Sample 10-21	Testbench using a typedef for virtual interfaces	346
Sample 10-22	Driver using a typedef for virtual interfaces	346
Sample 10-23	Testbench using an array of virtual interfaces	346
Sample 10-24	Testbench passing virtual interfaces with a port	347
Sample 10-25	Interface with tasks for parallel protocol	348
Sample 10-26	Interface with tasks for serial protocol	349
Sample 11-1	Top level module	353
Sample 11-2	Testbench program	354
Sample 11-3	CPU Management Interface	354
Sample 11-4	Utopia interface	355
Sample 11-5	Environment class header	356
Sample 11-6	Environment class methods	357
Sample 11-7	Callback class connects driver and scoreboard	360
Sample 11-8	Callback class connects monitor and scoreboard	360
Sample 11-9	Callback class connects the monitor and coverage	361
Sample 11-10	Environment configuration class	362
Sample 11-11	Cell configuration type	362
Sample 11-12	Configuration class methods	363
Sample 11-13	UNI cell format	363
Sample 11-14	NNI cell format	363
Sample 11-15	ATMCellType	364
Sample 11-16	UNI_cell definition	364
Sample 11-17	UNI_cell methods	365
Sample 11-18	UNI_generator class	368
Sample 11-19	driver class	368
Sample 11-20	Driver callback class	371
Sample 11-21	Monitor callback class	371
Sample 11-22	The Monitor class	371
Sample 11-23	The Scoreboard class	373
Sample 11-24	Functional coverage class	375
Sample 11-25	The CPU_driver class	376
Sample 11-26	Test with one cell	378

Sample 11-27	Test that drops cells using driver callback	379
Sample 12-1	SystemVerilog code calling C factorial routine	382
Sample 12-2	C factorial function	382
Sample 12-3	Changing the name of an imported function	383
Sample 12-4	Argument directions	383
Sample 12-5	C factorial routine with const argument	384
Sample 12-6	Importing a C math function	385
Sample 12-7	Counter method using a static variable	386
Sample 12-8	Testbench for an 7-bit counter with static storage	387
Sample 12-9	Counter method using instance storage	388
Sample 12-10	Testbench for an 7-bit counter with per-instance storage	389
Sample 12-11	Testbench for counter that checks for Z or X values	391
Sample 12-12	Counter method that checks for Z and X values	392
Sample 12-13	Counter class	393
Sample 12-14	Static methods and linkage	394
Sample 12-15	C++ counter communicating with methods	395
Sample 12-16	Static wrapper for C++ transaction level counter	396
Sample 12-17	Testbench for C++ model using methods	397
Sample 12-18	Testbench for C++ model using methods	398
Sample 12-19	C routine to compute Fibonacci series	398
Sample 12-20	Testbench for Fibonacci routine	399
Sample 12-21	C routine to compute Fibonacci series with 4-state array	399
Sample 12-22	Testbench for Fibonacci routine with 4-state array	399
Sample 12-23	Testbench code calling a C routine with an open array	400
Sample 12-24	C code using a basic open array	401
Sample 12-25	Testbench calling C code with multidimensional open array	402
Sample 12-26	C code with multidimensional open array	403
Sample 12-27	Testbench for packed open arrays	403
Sample 12-28	C code using packed open arrays	404
Sample 12-29	C code to share a structure	404
Sample 12-30	Testbench for sharing structure	405
Sample 12-31	Returning a string from C	406
Sample 12-32	Returning a string from a heap in C	406
Sample 12-33	Importing a pure function	407
Sample 12-34	Imported context tasks	407
Sample 12-35	Exporting a SystemVerilog function	408
Sample 12-36	Calling an exported SystemVerilog function from C	408
Sample 12-37	Output from simple export	408
Sample 12-38	SystemVerilog module for simple memory model	409
Sample 12-39	C code to read simple command file and call exported function	410
Sample 12-40	Command file for simple memory model	410
Sample 12-41	SystemVerilog module for memory model with exported tasks	411
Sample 12-42	C code to read command file and call exported function	411
Sample 12-43	Command file for simple memory model	412
Sample 12-44	Command file for exported methods with OOP memories	413

Sample 12-45	SystemVerilog module with memory model class	413
Sample 12-46	C code to call exported tasks with OOP memory	414
Sample 12-47	Second module for simple export example	415
Sample 12-48	Output from simple example with two modules	416
Sample 12-49	C code getting and setting context	416
Sample 12-50	Modules calling methods that get and set context	417
Sample 12-51	Output from svSetScope code	418
Sample 12-52	SystemVerilog code calling C wrapper for Perl	418
Sample 12-53	C wrapper for Perl script	419
Sample 12-54	Perl script called from C and SystemVerilog	419