

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	28
Sample 2-2	Signed data types	29
Sample 2-3	Checking for 4-state values	29
Sample 2-4	Declaring fixed-size arrays	30
Sample 2-5	Calculating the address width for a memory	30
Sample 2-6	Declaring and using multi-dimensional arrays	30
Sample 2-7	Unpacked array declarations	31
Sample 2-8	Initializing an array	31
Sample 2-9	Printing with %p print specifier	32
Sample 2-10	Using arrays with for- and foreach loops	32
Sample 2-11	Initialize and step through a multi-dimensional array	32
Sample 2-12	Output from printing multi-dimensional array values	33
Sample 2-13	Printing a multi-dimensional array	33
Sample 2-14	Output from printing multi-dimensional array values	33
Sample 2-15	Array copy and compare operations	34
Sample 2-16	Using word and bit subscripts together	35
Sample 2-17	Packed array declaration and usage	35
Sample 2-18	Declaration for a mixed packed/unpacked array	36
Sample 2-19	Using dynamic arrays	37
Sample 2-20	Using a dynamic array for an uncounted list	38
Sample 2-21	Multi-dimensional dynamic array	38
Sample 2-22	Queue methods	39
Sample 2-23	Queue operations	40
Sample 2-24	Declaring, initializing, and using associative arrays	41
Sample 2-25	Using an associative array with a string index	42
Sample 2-26	Initializing and printing associative arrays	43
Sample 2-27	Array reduction operations	43

Sample 2-28	Picking a random element from an associative array	44
Sample 2-29	Array locator methods: min, max, unique	44
Sample 2-30	Array locator methods: find	45
Sample 2-31	Declaring the iterator argument	45
Sample 2-32	Array locator methods	45
Sample 2-33	Creating the sum of an array of single bits	46
Sample 2-34	Sorting an array	46
Sample 2-35	Sorting an array of structures	47
Sample 2-36	A scoreboard with array methods	47
Sample 2-37	User-defined type-macro in Verilog	51
Sample 2-38	User-defined type in SystemVerilog	51
Sample 2-39	Definition of uint	51
Sample 2-40	User-defined array type	52
Sample 2-41	User-defined associative array index	52
Sample 2-42	Creating a single pixel type	52
Sample 2-43	The pixel struct	53
Sample 2-44	Initializing a struct	53
Sample 2-45	Using typedef to create a union	54
Sample 2-46	Packed structure	54
Sample 2-47	Package for ABC bus	55
Sample 2-48	Importing packages	55
Sample 2-49	Importing selected symbols from a package	56
Sample 2-50	Converting between int and real with static cast	57
Sample 2-51	Basic streaming operator	57
Sample 2-52	Converting between queues with streaming operator	58
Sample 2-53	Converting between a structure and an array with streaming operators	59
Sample 2-54	A simple enumerated type, not recommended	59
Sample 2-55	Enumerated types, recommended style	60
Sample 2-56	Specifying enumerated values	60
Sample 2-57	Incorrectly specifying enumerated values	61
Sample 2-58	Correctly specifying enumerated values	61
Sample 2-59	Stepping through all enumerated members	62
Sample 2-60	Assignments between integers and enumerated types	62
Sample 2-61	Declaring a const variable	63
Sample 2-62	String methods	64
Sample 2-63	Expression width depends on context	65
Sample 3-1	New procedural statements and operators	72
Sample 3-2	Using break and continue while reading a file	72
Sample 3-3	Case-inside statement with ranges	73
Sample 3-4	Void function for debug	73
Sample 3-5	Ignoring a function's return value	73
Sample 3-6	Simple task without begin...end	74
Sample 3-7	Verilog-1995 routine arguments	74
Sample 3-8	C-style routine arguments	75
Sample 3-9	Verbose Verilog-style routine arguments	75

Sample 3-10	Routine arguments with sticky types	75
Sample 3-11	Passing arrays using ref and const	76
Sample 3-12	Using ref across threads	77
Sample 3-13	Function with default argument values	78
Sample 3-14	Using default argument values	78
Sample 3-15	Binding arguments by name	79
Sample 3-16	Original task header	79
Sample 3-17	Task header with additional array argument	79
Sample 3-18	Task header with additional array argument	79
Sample 3-19	Return in a task	80
Sample 3-20	Return in a function	80
Sample 3-21	Returning an array from a function with a typedef	81
Sample 3-22	Passing an array to a function as a ref argument	81
Sample 3-23	Specifying automatic storage in program blocks	82
Sample 3-24	Static initialization bug	83
Sample 3-25	Static initialization fix: use automatic	83
Sample 3-26	Static initialization fix: break apart declaration and initialization	83
Sample 3-27	Time literals and \$timeformat	84
Sample 3-28	Time variables and rounding	85
Sample 4-1	Arbiter model using ports	91
Sample 4-2	Testbench module using ports	92
Sample 4-3	Top-level module with ports	92
Sample 4-4	Simple interface for arbiter	93
Sample 4-5	Arbiter using a simple interface	94
Sample 4-6	Testbench using a simple arbiter interface	94
Sample 4-7	Top module with a simple arbiter interface	94
Sample 4-8	Bad test module includes interface	95
Sample 4-9	Connecting an interface to a module that uses ports	95
Sample 4-10	Interface with modports	96
Sample 4-11	Arbiter model with interface using modports	96
Sample 4-12	Testbench with interface using modports	96
Sample 4-13	Top level module with modports	97
Sample 4-14	Arbiter monitor with interface using modports	98
Sample 4-15	Driving logic and wires in an interface	99
Sample 4-16	Interface with a clocking block	101
Sample 4-17	Race condition between testbench and design	103
Sample 4-18	Testbench using interface with clocking block	105
Sample 4-19	Signal synchronization	107
Sample 4-20	Synchronous interface sample and drive from module	107
Sample 4-21	Testbench using interface with clocking block	108
Sample 4-22	Interface signal drive	109
Sample 4-23	Driving a synchronous interface	109
Sample 4-24	Interface signal drive	110
Sample 4-25	Bidirectional signals in a program and interface	111
Sample 4-26	Clocking block with default statement	111

Sample 4-27	Clocking block with delays on individual signals	112
Sample 4-28	A final block	112
Sample 4-29	Bad clock generator in program block	113
Sample 4-30	Good clock generator in module	114
Sample 4-31	Top module with implicit port connections	114
Sample 4-32	Module with just port connections	115
Sample 4-33	Module with an interface	115
Sample 4-34	Top module connecting DUT and interface	115
Sample 4-35	Top-level scope for arbiter design	116
Sample 4-36	Cross-module references with \$root	117
Sample 4-37	Checking a signal with an if-statement	118
Sample 4-38	Simple immediate assertion	118
Sample 4-39	Error from failed immediate assertion	118
Sample 4-40	Creating a custom error message in an immediate assertion	119
Sample 4-41	Error from failed immediate assertion	119
Sample 4-42	Creating a custom error message	119
Sample 4-43	Concurrent assertion to check for X/Z	120
Sample 4-44	ATM router model header with ports	122
Sample 4-45	Top-level module without an interface	123
Sample 4-46	Verilog-1995 testbench using ports	124
Sample 4-47	Rx interface with modports and clocking block	125
Sample 4-48	Tx interface with modports and clocking block	126
Sample 4-49	ATM router model with interface using modports	126
Sample 4-50	Top-level module with interface	127
Sample 4-51	Testbench using an interface with a clocking block	128
Sample 4-52	Ref ports	129
Sample 5-1	Simple transaction class	135
Sample 5-2	Class in a package	136
Sample 5-3	Importing a package in a program	136
Sample 5-4	Declaring and using a handle	138
Sample 5-5	Simple user-defined new() function	138
Sample 5-6	A new() function with arguments	139
Sample 5-7	Calling the right new() function	139
Sample 5-8	Allocating multiple objects	140
Sample 5-9	Creating multiple objects	141
Sample 5-10	Using variables and routines in an object	142
Sample 5-11	Routines in the class	143
Sample 5-12	Out-of-block method declarations	144
Sample 5-13	Out-of-body method missing class name	145
Sample 5-14	Class with a static variable	146
Sample 5-15	The class scope resolution operator	147
Sample 5-16	Static storage for a handle	147
Sample 5-17	Static method displays static variable	148
Sample 5-18	Name scope	149
Sample 5-19	Class uses wrong variable	150

Sample 5-20	Move class into package to find bug	150
Sample 5-21	Using this to refer to class variable	151
Sample 5-22	Statistics class declaration	152
Sample 5-23	Encapsulating the Statistics class	152
Sample 5-24	Using a typedef class statement	154
Sample 5-25	Passing objects	155
Sample 5-26	Bad transaction creator task, missing ref on handle	156
Sample 5-27	Good transaction creator task with ref on handle	156
Sample 5-28	Bad generator creates only one object	157
Sample 5-29	Good generator creates many objects	157
Sample 5-30	Using an array of handles	158
Sample 5-31	Copying a simple class with new	158
Sample 5-32	Copying a complex class with new operator	159
Sample 5-33	Simple class with copy function	160
Sample 5-34	Using a copy function	160
Sample 5-35	Complex class with deep copy function	161
Sample 5-36	Statistics class declaration	161
Sample 5-37	Copying a complex class with new operator	162
Sample 5-38	Transaction class with pack and unpack functions	163
Sample 5-39	Using the pack and unpack functions	163
Sample 5-40	Basic Transactor	166
Sample 6-1	Simple random class	175
Sample 6-2	Randomization check macro and example	176
Sample 6-3	Constraint without random variables	177
Sample 6-4	Constrained-random class	178
Sample 6-5	Bad ordering constraint	179
Sample 6-6	Result from incorrect ordering constraint	179
Sample 6-7	Constrain variables to be in a fixed order	179
Sample 6-8	Weighted random distribution with dist	180
Sample 6-9	Dynamically changing distribution weights	181
Sample 6-10	Random sets of values	181
Sample 6-11	Inverted random set constraint	182
Sample 6-12	Random set constraint for an array	182
Sample 6-13	Equivalent set of constraints	182
Sample 6-14	Choose any value except those in an array	182
Sample 6-15	Printing a histogram	183
Sample 6-16	Histogram for inside constraint	183
Sample 6-17	Class to choose from an array of possible values	183
Sample 6-18	Choosing from an array of values	184
Sample 6-19	Using randc to choose array values in random order	184
Sample 6-20	Testbench for randc choosing array values in random order	185
Sample 6-21	Bidirectional constraints	185
Sample 6-22	Constraint block with implication operator	186
Sample 6-23	Implication operator	187
Sample 6-24	Constraint block with if implication operator	187

Sample 6-25	Constraint block with if-else operator	187
Sample 6-26	Constraint block with multiple if-else operator	188
Sample 6-27	Equivalence constraint	188
Sample 6-28	Class Unconstrained	189
Sample 6-29	Class with implication constraint	190
Sample 6-30	Class with implication constraint and additional constraint	191
Sample 6-31	Class with implication and solve...before	191
Sample 6-32	Using constraint_mode	193
Sample 6-33	Checking write length with a valid constraint	194
Sample 6-34	The randomize() with statement	195
Sample 6-35	Building a bathtub distribution	196
Sample 6-36	\$random_range usage	197
Sample 6-37	Constraint with a variable bound	198
Sample 6-38	dist constraint with variable weights	198
Sample 6-39	rand_mode disables randomization of variables	199
Sample 6-40	Randomizing a subset of variables in a class	200
Sample 6-41	Class with an external constraint	201
Sample 6-42	Program defining an external constraint	202
Sample 6-43	Signed variables cause randomization problems	203
Sample 6-44	Randomizing unsigned 32-bit variables	203
Sample 6-45	Randomizing unsigned 8-bit variables	203
Sample 6-46	Expensive constraint with mod and unsized variable	204
Sample 6-47	Efficient constraint with bit extract	204
Sample 6-48	Constraining dynamic array size	205
Sample 6-49	Random strobe pattern class	206
Sample 6-50	First attempt at sum constraint: bad_sum1	207
Sample 6-51	Program to try constraint with array sum	207
Sample 6-52	Output from bad_sum1	207
Sample 6-53	Second attempt at sum constraint: bad_sum2	208
Sample 6-54	Output from bad_sum2	208
Sample 6-55	Third attempt at sum constraint: bad_sum3	208
Sample 6-56	Output from bad_sum3	208
Sample 6-57	Fourth attempt at sum_constraint: bad_sum4	209
Sample 6-58	Output from bad_sum4	209
Sample 6-59	Simple foreach constraint: good_sum5	209
Sample 6-60	Output from good_sum5	209
Sample 6-61	Creating ascending array values with foreach	210
Sample 6-62	Creating unique array values with foreach	210
Sample 6-63	Creating unique array values with a randc helper class	211
Sample 6-64	Unique value generator	211
Sample 6-65	Class to generate a random array of unique values	212
Sample 6-66	Using the UniqueArray class	212
Sample 6-67	Constructing elements in a random array class	213
Sample 6-68	Simple random sequence with ascending values	215
Sample 6-69	Command generator using randsequence	216

Sample 6-70	Random control with randcase and \$urandom_range	217
Sample 6-71	Equivalent constrained class	217
Sample 6-72	Creating a decision tree with randcase	218
Sample 6-73	Simple pseudorandom number generator	219
Sample 6-74	Test code before modification	221
Sample 6-75	Test code after modification	222
Sample 6-76	Ethernet switch configuration class	223
Sample 6-77	Building environment with random configuration	223
Sample 6-78	Simple test using random configuration	224
Sample 6-79	Simple test that overrides random configuration	225
Sample 7-1	Interaction of begin...end and fork...join	233
Sample 7-2	Output from begin...end and fork...join	234
Sample 7-3	Fork...join_none code	234
Sample 7-4	Fork...join_none output	235
Sample 7-5	Fork...join_any code	235
Sample 7-6	Output from fork...join_any	236
Sample 7-7	Generator / Driver class with a run task	236
Sample 7-8	Dynamic thread creation	237
Sample 7-9	Bad fork...join_none inside a loop	238
Sample 7-10	Execution of bad fork...join_none inside a loop	239
Sample 7-11	Automatic variables in a fork...join_none	239
Sample 7-12	Steps in executing automatic variable code	240
Sample 7-13	Automatic variables in a fork...join_none	240
Sample 7-14	Using wait fork to wait for child threads	241
Sample 7-15	Bug using shared program variable	242
Sample 7-16	Disabling a thread	243
Sample 7-17	Limiting the scope of a disable fork	244
Sample 7-18	Using disable label to stop threads	245
Sample 7-19	Using disable label to stop a task	245
Sample 7-20	Blocking on an event in Verilog	247
Sample 7-21	Output from blocking on an event	247
Sample 7-22	Waiting for an event	248
Sample 7-23	Output from waiting for an event	248
Sample 7-24	Waiting on event causes a zero delay loop	249
Sample 7-25	Waiting for an edge on an event	249
Sample 7-26	Passing an event into a constructor	250
Sample 7-27	Waiting for multiple threads with wait fork	251
Sample 7-28	Waiting for multiple threads by counting triggers	251
Sample 7-29	Waiting for multiple threads using a thread count	252
Sample 7-30	Semaphores controlling access to hardware resource	253
Sample 7-31	Mailbox declarations	255
Sample 7-32	Bad generator creates only one object	255
Sample 7-33	Good generator creates many objects	256
Sample 7-34	Good driver receives transactions from mailbox	256
Sample 7-35	Exchanging objects using a mailbox: the Generator class	257

Sample 7-36	Bounded mailbox	258
Sample 7-37	Output from bounded mailbox	259
Sample 7-38	Producer–consumer without synchronization	260
Sample 7-39	Producer–consumer without synchronization output	261
Sample 7-40	Producer–consumer synchronized with bounded mailbox	262
Sample 7-41	Output from producer–consumer with bounded mailbox	262
Sample 7-42	Producer–consumer synchronized with an event	263
Sample 7-43	Output from producer–consumer with event	264
Sample 7-44	Producer–consumer synchronized with a mailbox	265
Sample 7-45	Output from producer–consumer with mailbox	266
Sample 7-46	Basic Transactor	267
Sample 7-47	Configuration class	268
Sample 7-48	Environment class	268
Sample 7-49	Basic test program	270
Sample 8-1	Base Transaction class	277
Sample 8-2	Extended Transaction class	278
Sample 8-3	Constructor with arguments in an extended class	279
Sample 8-4	Driver class	280
Sample 8-5	Bad generator class	281
Sample 8-6	Generator class using blueprint pattern	283
Sample 8-7	Environment class	284
Sample 8-8	Simple test program using environment defaults	284
Sample 8-9	Injecting an extended transaction into testbench	285
Sample 8-10	Adding a constraint with inheritance	286
Sample 8-11	Base and extended class	287
Sample 8-12	Copying extended handle to base handle	287
Sample 8-13	Copying a base handle to an extended handle	288
Sample 8-14	Using \$cast to copy handles	288
Sample 8-15	Transaction and BadTr classes	289
Sample 8-16	Calling class methods	289
Sample 8-17	Building an Ethernet frame with composition	292
Sample 8-18	Building an Ethernet frame with inheritance	293
Sample 8-19	Building a flat Ethernet frame	294
Sample 8-20	Base transaction class with a virtual copy function	295
Sample 8-21	Extended transaction class with virtual copy method	295
Sample 8-22	Base transaction class with copy function	296
Sample 8-23	Extended transaction class with new copy function	296
Sample 8-24	Abstract class with pure virtual methods	297
Sample 8-25	Transaction class extends abstract class	298
Sample 8-26	Base callback class	300
Sample 8-27	Driver class with callbacks	300
Sample 8-28	Test using a callback for error injection	301
Sample 8-29	Simple scoreboard for atomic transactions	302
Sample 8-30	Test using callback for scoreboard	303
Sample 8-31	Stack using the int type	304

Sample 8-32	Parameterized class for a stack	305
Sample 8-33	Creating the parameterized stack class	305
Sample 8-34	Parameterized generator class using blueprint pattern	306
Sample 8-35	Simple testbench using parameterized generator class	306
Sample 8-36	Common base class for parameterized generator class	307
Sample 8-37	Dynamic print class with static variables	308
Sample 8-38	Transactor class with dynamic print object	309
Sample 8-39	Static print class	310
Sample 8-40	Transactor class with static print class	310
Sample 8-41	Configuration database with global methods	311
Sample 8-42	Configuration database with parameterized class	312
Sample 8-43	Configuration database with static parameterized class	312
Sample 8-44	Testbench for configuration database	312
Sample 8-45	Base test class	314
Sample 8-46	Test registry class	314
Sample 8-47	Simple test in a class	315
Sample 8-48	Program block for test classes	315
Sample 8-49	Test class that puts a bad transaction in the generator	316
Sample 8-50	Common SVM base class	317
Sample 8-51	Component class	317
Sample 8-52	Common base class for proxy class	318
Sample 8-53	Parameterized proxy class	319
Sample 8-54	Factory class	320
Sample 8-55	Base test class and registration macro	321
Sample 8-56	Test program	321
Sample 8-57	UVM factory build example	322
Sample 9-1	Incomplete D-flip flop model missing a path	329
Sample 9-2	Functional coverage of a simple object	333
Sample 9-3	Coverage report for a simple object	334
Sample 9-4	Coverage report for a simple object, 100% coverage	335
Sample 9-5	Functional coverage inside a class	337
Sample 9-6	Test using functional coverage callback	338
Sample 9-7	Callback for functional coverage	339
Sample 9-8	Defining an argument list to the sample method	339
Sample 9-9	Cover group with a trigger	340
Sample 9-10	Module with SystemVerilog Assertion	340
Sample 9-11	Triggering a cover group with an SVA	340
Sample 9-12	Using auto_bin_max set to 2	342
Sample 9-13	Report with auto_bin_max set to 2	342
Sample 9-14	Using auto_bin_max for all cover points	342
Sample 9-15	Using an expression in a cover point	343
Sample 9-16	Defining bins for transaction length	343
Sample 9-17	Coverage report for transaction length	344
Sample 9-18	Specifying bin names	345
Sample 9-19	Report showing bin names	345

Sample 9-20	Specifying ranges with \$	346
Sample 9-21	Conditional coverage — disable during reset	346
Sample 9-22	Using stop and start functions	346
Sample 9-23	Functional coverage for an enumerated type	347
Sample 9-24	Coverage report with enumerated types	347
Sample 9-25	Specifying transitions for a cover point	347
Sample 9-26	Wildcard bins for a cover point	348
Sample 9-27	Cover point with ignore_bins	348
Sample 9-28	Cover point with auto_bin_max and ignore_bins	349
Sample 9-29	Cover point with illegal_bins	349
Sample 9-30	Basic cross coverage	350
Sample 9-31	Coverage summary report for basic cross coverage	351
Sample 9-32	Specifying cross coverage bin names	352
Sample 9-33	Cross coverage report with labeled bins	352
Sample 9-34	Excluding bins from cross coverage	353
Sample 9-35	Specifying cross coverage weight	354
Sample 9-36	Cross coverage with bin names	355
Sample 9-37	Cross coverage with binsof	355
Sample 9-38	Mimicking cross coverage with concatenation	356
Sample 9-39	Covergroup with simple argument	356
Sample 9-40	Pass-by-reference	357
Sample 9-41	Specifying per-instance coverage	358
Sample 9-42	Specifying comments for a cover group	358
Sample 9-43	Specifying comments for a cover group instance	358
Sample 9-44	Report all bins including empty ones	359
Sample 9-45	Specifying the coverage goal	360
Sample 9-46	Original class for packet length	360
Sample 9-47	solve...before constraint for packet length	361
Sample 10-1	Rx interface with clocking block	366
Sample 10-2	Tx interface with clocking block	366
Sample 10-3	Testbench using physical interfaces	367
Sample 10-4	Top level module with array of interfaces	368
Sample 10-5	Testbench using virtual interfaces	369
Sample 10-6	Testbench using virtual interfaces	369
Sample 10-7	Monitor class using virtual interfaces	370
Sample 10-8	Test harness using an interface in the port list	371
Sample 10-9	Test with an interface in the port list	372
Sample 10-10	Top module with a second interface in the test's port list	372
Sample 10-11	Test with two interfaces in the port list	372
Sample 10-12	Test with virtual interface and XMR	372
Sample 10-13	Test harness without interfaces in the port list	373
Sample 10-14	Test harness with a second interface	373
Sample 10-15	Test with two virtual interfaces and XMRs	373
Sample 10-16	Interface for 8-bit counter	374
Sample 10-17	Counter model using X_if interface	375

Sample 10-18	Top-level module with an array of virtual interfaces	375
Sample 10-19	Counter testbench using virtual interfaces	376
Sample 10-20	Driver class using virtual interfaces	377
Sample 10-21	Interface with a typedef	377
Sample 10-22	Testbench using a typedef for virtual interfaces	378
Sample 10-23	Driver using a typedef for virtual interfaces	378
Sample 10-24	Testbench using an array of virtual interfaces	378
Sample 10-25	Testbench passing virtual interfaces with a port	379
Sample 10-26	Parameterized counter model using X_if interface	380
Sample 10-27	Parameterized interface for 8-bit counter	380
Sample 10-28	Parameterized top-level module with an array of virtual interfaces	380
Sample 10-29	Parameterized counter testbench using virtual interfaces	381
Sample 10-30	Driver class using virtual interfaces	381
Sample 10-31	Interface with tasks for parallel protocol	382
Sample 10-32	Interface with tasks for serial protocol	383
Sample 11-1	Top level module	389
Sample 11-2	Testbench program	390
Sample 11-3	CPU Management Interface	390
Sample 11-4	Utopia interface	391
Sample 11-5	Environment class header	392
Sample 11-6	Environment class methods	393
Sample 11-7	Callback class connects driver and scoreboard	396
Sample 11-8	Callback class connects monitor and scoreboard	396
Sample 11-9	Callback class connects the monitor and coverage	397
Sample 11-10	Environment configuration class	398
Sample 11-11	Cell configuration type	398
Sample 11-12	Configuration class methods	399
Sample 11-13	UNI cell format	399
Sample 11-14	NNI cell format	399
Sample 11-15	ATMCellType	400
Sample 11-16	UNI_cell definition	400
Sample 11-17	UNI_cell methods	401
Sample 11-18	UNI_generator class	404
Sample 11-19	driver class	404
Sample 11-20	Driver callback class	407
Sample 11-21	Monitor callback class	407
Sample 11-22	The Monitor class	407
Sample 11-23	The Scoreboard class	409
Sample 11-24	Functional coverage class	411
Sample 11-25	The CPU_driver class	412
Sample 11-26	Test with one cell	414
Sample 11-27	Test that drops cells using driver callback	415
Sample 12-1	SystemVerilog code calling C factorial routine	420
Sample 12-2	C factorial function	420
Sample 12-3	Changing the name of an imported function	421

Sample 12-4	Argument directions	422
Sample 12-5	C factorial routine with const argument	422
Sample 12-6	Importing a C math function	423
Sample 12-7	Counter routine using a static variable	424
Sample 12-8	Testbench for an 7-bit counter with static storage	425
Sample 12-9	Counter routine using instance storage	426
Sample 12-10	Testbench for an 7-bit counter with per-instance storage	427
Sample 12-11	Testbench for counter that checks for Z or X values	429
Sample 12-12	Counter routine that checks for Z and X values	430
Sample 12-13	Counter class	431
Sample 12-14	Static methods and linkage	432
Sample 12-15	C++ counter communicating with methods	433
Sample 12-16	Static wrapper for C++ transaction level counter	434
Sample 12-17	Testbench for C++ model using methods	435
Sample 12-18	C routine to compute Fibonacci series	436
Sample 12-19	Testbench for Fibonacci routine	437
Sample 12-20	C routine to compute Fibonacci series with 4-state array	437
Sample 12-21	Testbench for Fibonacci routine with 4-state array	437
Sample 12-22	Testbench code calling a C routine with an open array	438
Sample 12-23	C code using a basic open array	438
Sample 12-24	Testbench calling C code with multi-dimensional open array	440
Sample 12-25	C code with multi-dimensional open array	440
Sample 12-26	Testbench for packed open arrays	441
Sample 12-27	C code using packed open arrays	441
Sample 12-28	C code to share a structure	442
Sample 12-29	Testbench for sharing structure	443
Sample 12-30	Returning a string from C	444
Sample 12-31	Returning a string from a heap in C	444
Sample 12-32	Importing a pure function	445
Sample 12-33	Imported context tasks	445
Sample 12-34	Exporting a SystemVerilog function	446
Sample 12-35	Calling an exported SystemVerilog function from C	446
Sample 12-36	Output from simple export	446
Sample 12-37	C code to read simple command file and call exported function	447
Sample 12-38	SystemVerilog module for simple memory model	447
Sample 12-39	Command file for simple memory model	448
Sample 12-40	SystemVerilog module for memory model with exported tasks	448
Sample 12-41	C code to read command file and call exported function	449
Sample 12-42	Command file for simple memory model	450
Sample 12-43	Command file for exported methods with OOP memories	450
Sample 12-44	SystemVerilog module with memory model class	450
Sample 12-45	C code to call exported tasks with OOP memory	451
Sample 12-46	Second module for simple export example	453
Sample 12-47	Output from simple example with two modules	453
Sample 12-48	C code getting and setting context	454

Sample 12-49	Modules calling methods that get and set context	455
Sample 12-50	Output from svSetScope code	455
Sample 12-51	SystemVerilog code calling C wrapper for Perl	456
Sample 12-52	C wrapper for Perl script	456
Sample 12-53	Perl script called from C and SystemVerilog	456
Sample 12-54	VCS command line to run Perl script	457

