

Contents

1	Introduction	1
1.1	How Will This Book Help You?	5
1.2	SystemVerilog Assertions and Functional Coverage Under IEEE 1800 SystemVerilog Umbrella	6
1.3	SystemVerilog Assertions Evolution	7
 Part I System Verilog Assertions (SVA)		
2	System Verilog Assertions	11
2.1	What Is an Assertion?	12
2.2	Why Assertions? What Are the Advantages?	12
2.3	Assertions Shorten Time to Develop	13
2.4	Assertions Improve Observability	13
2.5	Assertions in Static Formal	14
2.6	Assertion Synthesis	16
2.7	One-Time Effort, Many Benefits	19
2.8	Assertions Whining	20
2.9	Who Will Add Assertions? War Within!	22
2.10	A Simple PCI Read Example—Creating an Assertions Test Plan	22
2.11	What Type of Assertions Should I Add?	24
2.12	Protocol for Adding Assertions	25
2.13	How Do I Know I Have Enough Assertions?	26
2.14	Assertion-Based Methodology Allows for Full Random Verification	26
2.15	Assertions Help Detect Bugs Not Easily Observed at Primary Outputs	27
2.16	Other Major Benefits	28
2.17	Use Assertions for Specification and Review	29
2.18	Assertion Types	30

3	Sequential Domain Coverage (“Cover” Property and “Cover” Sequence)	33
3.1	“Cover” Property	35
3.2	“Cover” Sequence	36
4	Conventions Used in the Book	37
5	Immediate Assertions	39
5.1	Deferred Immediate Assertions	42
5.2	Disabling a Deferred Assertion	45
5.3	Deferred Assertion in a <i>Function</i>	46
6	Concurrent Assertions: Basics	49
6.1	Implication Operator, Antecedent and Consequent	54
6.2	Clocking Basics	57
6.3	Sampling Edge (Clock Edge) Value: How Are Assertions Evaluated in a Simulation Time Tick?	59
6.3.1	Active Region	59
6.3.2	Observed Region	59
6.3.3	Reactive Region	59
6.3.4	Preponed Region	60
6.4	Default Clocking Block	64
6.5	Gated Clk	69
6.6	Concurrent Assertions Are Multi-Threaded	69
6.7	Formal Arguments	71
6.8	Disable (Property) Operator—“disable iff”	74
6.9	Default disable iff	76
6.10	Nested <i>disable iff</i> —ILLEGAL	77
6.11	Severity Levels (for Both Concurrent and Immediate Assertions)	78
6.12	Binding Properties	79
6.13	Binding Properties (Scope Visibility)	80
6.14	VHDL DUT Binding with SystemVerilog Assertions	81
6.15	Assertion Adoption in Existing Design	82
6.16	Difference Between “sequence” and “property”	83
7	Sampled Value Functions	85
7.1	\$rose—Edge Detection in Property/Sequence	87
7.2	Edge Detection is Useful Because	87
7.3	\$fell—Edge Detection in Property/Sequence	89
7.4	\$rose, \$fell—in Procedural	89
7.5	\$stable	91
7.6	\$stable in Procedural Block	91
7.7	\$past	92
7.8	Application: \$past ()	96
7.8.1	Specification	97
7.8.2	Solution	97
7.9	\$past Rescues \$fell!	97
7.10	\$past() in Procedural Block	98

8	Operators	99
8.1	##m: Clock Delay	100
8.2	Clock Delay operator: ##m where $m = 0$	101
8.3	Application: Clock Delay Operator:: ##m ($m = 0$)	102
8.4	##[m:n]—Clock Delay Range	102
8.5	Clock Delay Range Operator: ##[m:n]: Multiple Threads	104
8.6	Clock Delay Range Operator: ##[m:n] ($m = 0$; $n = \$$)	110
8.7	[*m]—Consecutive Repetition Operator	112
8.8	[*m:n]—Consecutive Repetition Range Operator	114
8.9	Application: Consecutive Repetition Range Operator	118
8.10	[=m]: Non-consecutive Repetition	125
8.11	[=m:n]: Non-consecutive Repetition Range	127
8.12	Application: Non-consecutive Repetition Operator	128
8.13	[− > m] Non-consecutive GoTo Repetition Operator	130
8.14	Difference Between [=m:n] and [− > m:n]	131
8.15	Application: GoTo repetition: Non-consecutive Operator	132
8.16	sig1 Throughout seq1	133
8.17	Application: sig1 Throughout seq1	133
8.18	seq1 Within seq2	136
8.19	Application: seq1 <i>within</i> seq2	137
8.19.1	“within” operator PASS Cases	138
8.19.2	“within” operator: FAIL Cases	139
8.20	seq1 <i>and</i> seq2	141
8.21	Application: “and” Operator	142
8.22	seq1 “or” seq2	142
8.23	Application: or Operator	144
8.24	seq1 “intersect” seq2	144
8.25	Application: “intersect” Operator	145
8.26	Application: <i>intersect</i> Operator (<i>Interesting Application</i>)	149
8.27	“intersect” and “and”:: What’s the Difference?	152
8.28	first_match	153
8.29	Application: first_match	153
8.30	not < property expr>	156
8.31	Application: not Operator	156
8.32	if (expression) property_expr1 <i>else</i> property_expr2	159
8.33	Application: if .. else	160
8.34	“iff” and “implies”	160
9	System Functions and Tasks	163
9.1	\$onehot, \$onehot0	164
9.2	\$isunknown	165
9.2.1	Application \$isunknown	165
9.3	\$countones	166
9.4	\$countones (as Boolean)	168
9.5	\$countbits	168
9.6	\$assertoff, \$asserton, \$assertkill	169

10 Multiple Clocks	171
10.1 Multiply Clocked Sequences and Properties	172
10.2 Multiply Clocked Sequences	173
10.3 Multiply Clocked Sequences—Legal and Illegal Sequences	174
10.4 Multiply Clocked Properties—“and” Operator	174
10.5 Multiply Clocked Properties—“or” Operator	176
10.6 Multiply Clocked Properties—“not”-Operator	176
10.7 Multiply Clocked Properties—Clock Resolution	178
10.8 Multiply Clocked Properties—Legal and Illegal Conditions	180
11 Local Variables	183
11.1 Application: Local Variables	195
12 Recursive Property	197
12.1 Application: Recursive Property	199
12.2 Application: Recursive Property	200
13 Endpoint of a Sequence (.triggered and .matched)	203
13.1 .triggered (Replaces .ended)	204
13.2 .matched	212
13.3 Application: .matched	214
14 “expect”	217
15 “assume” and “restrict” for Simulation and Formal (Static Functional) Verification	221
15.1 “assume” Statement	222
15.2 “restrict” Statement	223
15.3 “dist” (Distribution Operator) and “inside” Operator	224
16 Clock Domain Crossing (CDC) Verification Using Assertions	227
16.1 Automated CDC Verification	231
16.1.1 Step 1: Structural Verification	232
16.1.2 Step 2: Protocol Verification	232
16.1.3 Step 3: Debug	233
17 Important Topics	235
17.1 Test the Test-Bench	236
17.2 Embedding Concurrent Assertions in Procedural Block	237
17.3 Calling Subroutines on the Match of a Sequence	243
17.4 Sequence as a Formal Argument	246
17.5 Sequence as an Antecedent	247
17.6 Sequence in Sensitivity List	248
17.7 Building a Counter	250
17.8 Clock Delay: What If You Want <i>Variable</i> Clock Delay?	251
17.9 What If the “Action Block” Is Blocking?	253
17.10 Nested Implications in a Property. Be Careful	255
17.11 Subsequence in a Sequence	257
17.12 Cyclic Dependency—Mutually Recursive Property	258

17.13 Refinement on a Theme	259
17.14 Simulation Performance Efficiency	259
17.15 It’s a Vacuous World! Huh?	260
17.16 Concurrent Assertion—Without—an Implication	260
17.17 Concurrent Assertion—with—an Implication	261
17.18 Vacuous Pass	262
17.19 Concurrent Assertion—With “cover”	263
17.20 Empty Sequence	264
18 Asynchronous FIFO Assertions	269
18.1 Asynchronous FIFO Design	270
18.2 Asynchronous FIFO Test-Bench and Assertions	273
19 Asynchronous Assertions	279
19.1 Glitch Detection	283
20 IEEE-1800-2009/2012 Features	285
20.1 Strong and Weak Sequences	286
20.2 \$changed	287
20.3 \$sampled	288
20.4 Global Clocking past and future Sampled Value Functions	290
20.5 future Global Clocking Sampled Value Functions	292
20.6 past Global Clocking Sampled Value Functions	293
20.7 Illegal Use of Global Clocking Future Sampled Value Functions	293
20.8 “followed by” Properties #-# and #=#	295
20.9 “always” and “s_always” Property	297
20.10 “eventually,” “s_eventually”	299
20.11 “until,” “s_until,” “until_with,” and “s_until_with”	302
20.12 “nexttime” and “s_nexttime”	306
20.13 “case” Statement	310
20.14 \$inferred_clock and \$inferred_disable	311
20.15 “restrict” for Formal Verification	313
20.16 Abort Properties: reject_on, accept_on, sync_reject_on, sync_accept_on	314
20.17 \$assertpassoff, \$assertpasson, \$assertfailoff, \$assertfailon, \$assertnonvacuouson, \$assertvacuousoff	318
20.18 \$assertcontrol	318
21 “let” Declarations	325
21.1 let: Local Scope	326
21.2 “let”: With Parameters	328
21.3 “let”: In Immediate and Concurrent Assertions	329
22 Checkers	335
22.1 Nested Checkers	340
22.2 Checker: Legal Conditions	341

22.3	Checker: Illegal Conditions	342
22.4	Checker: Important Points	345
22.5	Checker: Instantiation Rules	348
22.6	Checker: "Formal" and "Actual" Rules	350
22.7	Checker: In a Package	351
23	SystemVerilog Assertions LABs.	353
23.1	LAB1: Assertions With/Without Implication and "bind"	354
23.2	LAB1: "bind" DUT Model and Test-Bench	354
23.3	LAB1: Questions	357
23.4	LAB2: Overlap and Nonoverlap Operators	360
23.5	LAB2 DUT Model and Test-Bench	360
23.6	LAB2: Questions	362
23.7	LAB3: Synchronous FIFO Assertions	364
23.8	LAB3: DUT Model and Test-Bench	364
23.9	LAB3: Questions	368
23.10	LAB4: Counter	374
23.11	LAB4: Questions	376
23.12	LAB5: Data Transfer Protocol	380
23.13	LAB5: Questions	385
23.14	LAB6: PCI Read Protocol	387
23.15	LAB6: Questions	389
24	System Verilog Assertions: LAB Answers.	397
24.1	LAB1: Answers: "bind" and Implication Operators	398
24.2	LAB2: Answers: Overlap and Nonoverlap Operators	403
24.3	LAB3: Answers: Synchronous FIFO	407
24.4	LAB4: Answers: Counter	410
24.5	LAB5: Answers: Data Transfer Protocol	412
24.6	LAB6: Answers: PCI Read Protocol	414
24.7	Further PCI Protocol Assertion Examples	416

Part II System Verilog Functional Coverage (FC)

25	Functional Coverage	421
25.1	Difference Between Code Coverage and Functional Coverage	422
25.1.1	Code Coverage	422
25.1.2	Functional Coverage	424
25.2	Assertion Based Verification (ABV) and Functional Coverage (FC) Based Methodology	425
25.3	"cover" Property, "cover" Sequence	427
25.4	"covergroup" (With Its "coverpoints," "bins," etc.)	427
25.5	Functional Coverage Methodology	427
25.6	Follow the Bugs!!	430

26	Functional Coverage: Language Features	431
26.1	Covergroup/Coverpoint	432
26.1.1	What Is a Covergroup?	432
26.1.2	What Is a Coverpoint?	432
26.2	System Verilog "covergroup": Basics... ..	432
26.3	System Verilog Coverpoint Basics... ..	433
26.4	Coverpoint Using a Function or an Expression	435
26.5	Coverpoint: Other Nuances	436
26.6	Covergroup/Coverpoint Example... ..	436
26.7	System Verilog "bins": Basics... ..	439
26.8	"bins" with Expressions	441
26.9	Bin Filtering Using the "with" Clause	442
26.10	Covergroup/Coverpoint with bins: Example... ..	445
26.11	"covergroup": Formal and Actual Arguments	446
26.12	Coverpoint: Hierarchical References	447
26.13	Class: Embedded "covergroup" in a "class"	448
26.14	Class: Embedded covergroup: Hierarchical Accessibility	449
26.15	Class: Multiple Covergroups in a Class	450
26.16	Class: Overriding Covergroups in a Class	451
26.17	Class: Parameterizing Coverpoints	453
26.18	Class: Creating Array of Instances of a "covergroup"	454
26.19	Further Methodology Guidelines	456
26.20	"cross" Coverage	459
26.21	"bins" for Transition Coverage	468
26.22	"wildcard bins"	472
26.23	"ignore_bins"	473
26.24	"illegal_bins"	478
26.25	"binsof" and "intersect"	479
27	Performance Implications of Coverage Methodology	483
27.1	Know <i>What</i> You Should Cover	484
27.2	Know <i>When</i> You Should Cover for Better Performance	484
27.3	sample() Method	484
27.4	User Defined sample() Method	485
27.5	Querying for Coverage	489
27.6	strobe() Method	490
27.7	Application: Have You Transmitted All Different Lengths of a Frame?	491
28	Coverage Options	493
28.1	Coverage Options: Instance Specific: Example	495
28.2	Coverage Options: Instance Specific Per-syntactic Level	496
28.3	Coverage Options for "covergroup" Type: Example	499

28.4	Coverage System Tasks, Functions, and Methods	500
28.4.1	sample () Method	500
28.4.2	real get_coverage (ref int, ref int)	501
28.4.3	real get_inst_coverage (ref int, ref int)	501
28.4.4	void set_inst_name (string)	501
	Index	503