

Contents

- 1 Introduction.** 1
 - 1.1 Functional Design Verification: Current State of Affair 2
 - 1.2 Where Are the Bugs? 3
- 2 Functional Verification: Challenges and Solutions** 5
 - 2.1 Verification Challenges and Solutions 5
 - 2.1.1 Reduce Time to Develop 6
 - 2.1.2 Reduce Time to Simulate. 7
 - 2.1.3 Reduce Time to Debug. 8
 - 2.1.4 Reduce Time to Cover: Check How Good
Is Your Testbench. 8
 - 2.2 A Comprehensive Verification Plan. 9
- 3 SystemVerilog Paradigm** 13
 - 3.1 SystemVerilog Language Umbrella. 13
 - 3.2 SystemVerilog Language Evolution. 15
- 4 UVM (Universal Verification Methodology).** 17
 - 4.1 What Is UVM? 17
 - 4.2 Polymorphism 19
 - 4.3 UVM Hierarchy 20
 - 4.3.1 UVM Testbench. 22
 - 4.3.2 UVM Test. 23
 - 4.3.3 UVM Environment 23
 - 4.3.4 UVM Agent 24
 - 4.3.5 UVM Sequence Item 25
 - 4.3.6 UVM Sequence 26
 - 4.3.7 UVM Sequencer. 27
 - 4.3.8 UVM Driver. 27
 - 4.3.9 UVM Monitor 28
 - 4.3.10 UVM Scoreboard. 29

4.4	UVM Class Library	29
4.5	UVM Transaction-Level Communication Protocol: Basics	30
4.5.1	Basic Transaction-Level Communication	30
4.5.2	Hierarchical Connections.	33
4.5.3	Analysis Ports and Exports	34
4.6	UVM Phases	35
4.6.1	Build Phases.	35
4.6.2	Run-Time Phases	37
4.6.3	Cleanup Phases	39
4.7	UVM Example: One	41
4.7.1	Modeling a Sequence Item.	41
4.7.2	Building UVM Driver	43
4.7.3	Basic Sequencer and Driver Interaction.	44
4.7.4	Building UVM Sequencer	46
4.7.5	Building UVM Monitor.	46
4.7.6	UVM Agent: Connecting Driver, Sequencer, and Monitor	47
4.7.7	Building the Environment	48
4.7.8	UVM Top-Level Module (Testbench) Example	49
4.8	UVM Example: Two	51
4.8.1	DUT: lpi.sv.	51
4.8.2	lpi_if.sv	52
4.8.3	lpi_seq_item.sv	53
4.8.4	lpi_sequencer.sv	53
4.8.5	lpi_driver.sv	53
4.8.6	lpi_monitor.sv	54
4.8.7	lpi_agent.sv	55
4.8.8	lpi_basic_sequence.sv	57
4.8.9	lpi_basic_test.sv	58
4.8.10	lpi_env.sv	59
4.8.11	lpi_top_v_sequencer.sv	60
4.8.12	lpi top environment.sv	60
4.8.13	lpi_testbench.sv	62
4.9	UVM Is Reusable.	63
5	Constrained Random Verification (CRV)	65
5.1	Productivity Gain with CRV	65
5.2	CRV Methodology.	66
5.3	Basics of CRV	67
5.3.1	Random Variables: Basics	70
5.3.2	Random Number System Functions and Methods.	71
5.3.3	Random Weighted Case: <i>Randcase</i>	73
6	SystemVerilog Assertions (SVA)	75
6.1	Evolution of SystemVerilog Assertions	75
6.2	SystemVerilog Assertion Advantages	76

6.2.1	Assertions Shorten Time to Develop	76
6.2.2	Assertions Improve Observability	77
6.2.3	Assertions Shorten Time to Cover	77
6.2.4	One-Time Effort: Many Benefits	80
6.3	Creating an Assertion Test Plan: PCI Read Example	81
6.3.1	PCI: Read Protocol Assertion Test Plan (Verification Team)	82
6.3.2	PCI: Read Protocol Assertions Test Plan (Design Team)	83
6.4	SVA Assertion Methodology Components	83
6.4.1	What Type of Assertions Should I Add?	83
6.4.2	Protocol for Adding Assertions	85
6.4.3	How Do I know I Have Enough Assertions?	85
6.4.4	Use Assertions for Specification and Review	86
6.5	Immediate Assertions	87
6.6	Concurrent Assertions	89
6.6.1	Overlapping and Nonoverlapping Operators	90
6.7	Clocking Basics	90
6.7.1	Sampling Edge (Clock Edge)	92
6.8	Concurrent Assertions: Binding Properties	93
6.8.1	Binding Properties (Scope Visibility)	95
6.9	Operators	95
6.9.1	##m: Clock Delay	95
6.9.2	##[m:n]: Clock Delay Range	98
6.9.3	[*m]: Consecutive Repetition Operator	99
6.9.4	[*m:n]: Consecutive Repetition Range	100
6.9.5	[=m]: Repetition Non-consecutive	102
6.9.6	[=m:n]: Repetition Non-consecutive Range	103
6.9.7	[->m] Non-consecutive GoTo Repetition Operator	105
6.9.8	sig1 <i>throughout</i> seq1	106
6.9.9	seq1 <i>within</i> seq2	108
6.9.10	seq1 <i>and</i> seq2	108
6.9.11	seq1 <i>or</i> seq2	110
6.9.12	seq1 <i>intersect</i> seq2	110
6.10	Local Variables	111
6.11	SystemVerilog Assertions: Applications	114
6.11.1	SVA Application: Infinite Delay Range Operator	114
6.11.2	SVA Application: Consecutive Delay Range Operator	115
6.11.3	SVA Application: Consecutive Delay Range Operator	115
6.11.4	SVA Application: Antecedent as Property Check. Consequent as Hard Failure	116
6.11.5	SVA Application: State Transition Check of a State Machine	117

6.11.6	SVA Application: Multi-threaded Operation	120
6.11.7	SVA Application: A Request $\Leftrightarrow$ Grant Bus Protocol	126
6.11.8	SVA Application: Machine Check Exception	126
6.11.9	SVA Application: “req” followed by “ack”	128
7	SystemVerilog Functional Coverage (SFC)	129
7.1	Difference Between Code Coverage and Functional Coverage	129
7.2	SystemVerilog Components for Complete Coverage	130
7.3	Assertion (ABV) and Functional Coverage (SFC)-Based Methodology	131
7.3.1	Follow the Bugs!	134
7.4	SystemVerilog “Covergroup” Basics	135
7.5	SystemVerilog “Coverpoint” Basics	136
7.6	SystemVerilog “Bins”: Basics	136
7.7	“Covergroup” in a “Class”	138
7.8	“Cross” Coverage	139
7.9	“Bins” for Transition Coverage	141
7.10	Performance Implications of Coverage Methodology	142
7.10.1	Know <i>What</i> You Should Cover	143
7.10.2	Know <i>When</i> You Should Cover	144
7.11	When to “Cover” (Performance Implication)	144
7.12	SystemVerilog Functional Coverage: Applications	145
7.12.1	PCI Cycles	145
7.12.2	Frame Length Coverage	147
8	Clock Domain Crossing (CDC) Verification	149
8.1	Design Complexity and CDC	149
8.2	Metastability	150
8.3	Synchronizer	151
8.3.1	Two-Flop Synchronizer (Identical Transmit and Receive Clock Frequencies)	151
8.3.2	Three-Flop Synchronizer (High-Speed Designs)	152
8.3.3	Synchronizing Fast-Clock (Transmit) into Slow-Clock (Receive) Domains	153
8.3.4	Multi-bit Synchronization	155
8.3.5	Design of an Asynchronous FIFO Using Gray Code Counters	156
8.4	CDC Checks Using SystemVerilog Assertions	159
8.5	CDC Verification Methodology	161
8.5.1	Automated CDC Verification	163
8.5.2	Step 1: Structural Verification	163
8.5.3	Step 2: Protocol Verification	164
8.5.4	Step 3: Debug	164
8.6	CDC Verification at Gate Level	164
8.7	EDA Vendors and CDC Tools Support	165
8.7.1	Mentor	166

9	Low-Power Verification	167
9.1	Power Requirements: Current Industry Trend	167
9.2	Dynamic Low-Power Verification Challenges	169
9.3	UPF (Unified Power Format)	170
9.3.1	UPF Evolution	171
9.4	UPF Methodology	172
9.4.1	Low-Power Design Terminology/Definitions	174
9.5	UPF: Detailed SoC Example	175
9.5.1	Design/Logic Hierarchy Navigation	175
9.5.2	Power Domain Creation	176
9.5.3	Supply Power to the Power Domains: Supply Network	178
9.5.4	Power Switch Creation	181
9.5.5	Supply Port States	183
9.5.6	Power State Table	183
9.5.7	State Retention Strategies	184
9.5.8	Isolation Strategies	186
9.5.9	Level Shifting Strategies	188
9.6	Power Estimation at Architecture Level	189
9.7	UPF Features Subset (IEEE 1801–2009)	191
10	Static Verification (Formal-Based Technologies)	193
10.1	What Is Static Verification?	193
10.2	Static Verification Umbrella	195
10.3	Static Formal Verification (Aka Model Checking Aka Static Functional Verification)	196
10.3.1	Critical Logic Blocks for Static Formal	196
10.3.2	SystemVerilog Assertions and Assumptions for Static Formal and Simulation	199
10.3.3	SystemVerilog “Assume” and Static Formal Verification	200
10.3.4	Static Formal vs. Simulation	201
10.4	Static Formal + Simulation Hybrid Verification Methodology	202
10.5	Logic Equivalence Check (LEC)	203
10.5.1	LEC Technology	204
10.5.2	RTL to RTL Verification	206
10.5.3	RTL to Gate Verification	207
10.5.4	Gate to Gate Verification	207
10.5.5	ESL (C/ C++/ SystemC model) to RTL (Sequential Equivalence Checking—SEC)	207
10.5.6	Layout vs. Schematic (LVS) Physical Verification	212
10.5.7	RTL Lint	215
10.6	Structural Checks	216
10.7	Low Power Structural Checks	217
10.8	X-State Verification	219
10.9	Connectivity Verification	220

11	ESL (Electronic System Level) Verification Methodology	221
11.1	ESL (Electronic System Level)	221
11.1.1	How Does ESL Help with Verification?	222
11.1.2	ESL Virtual Platform Use Cases	223
11.2	OSCI TLM 2.0 Standard for ESL	224
11.2.1	Loosely Timed (LT) TLM 2.0 Transaction-Level Modeling	226
11.2.2	Approximately Timed (AT) TLM 2.0 Transaction-Level Modeling	227
11.3	Virtual Platform Example	229
11.3.1	Advantages of a Virtual Platform	230
11.3.2	Open Virtual Platform (OVP) Initiative	230
11.3.3	Rationale for Software Virtual Platforms (OVP <i>n.d.</i>)	231
11.4	ESL/Virtual Platform for Design Verification	232
11.4.1	Overview	232
11.4.2	Virtual Platform and RTL Co-simulation and Verification	233
11.4.3	Virtual Platform as a Reference Model in UVM Scoreboard	234
11.4.4	ESL to RTL Reuse Methodology	235
11.4.5	Design and Verification Reuse: Algorithm $\Leftrightarrow$ ESL: TLM 2.0	237
11.4.6	Design and Verification Reuse: ESL/TLM 2.0 $\Leftrightarrow$ RTL	238
11.4.7	Design and Verification Reuse: Algorithm $\Leftrightarrow$ ESL-TLM 2.0 $\Leftrightarrow$ RTL	239
12	Hardware/Software Co-verification	243
12.1	Overview	243
12.2	Hardware/Software Co-verification Using Virtual Platform with Hardware Emulation	244
12.2.1	Hardware Emulation and Prototyping	244
12.2.2	Emulation System Compile Time	245
12.2.3	Difference Between Emulator and FPGA-Based Prototype	246
12.2.4	Myths About Emulation-Based Acceleration (Rizzati)	247
12.3	Speed Bridge	248
12.4	Virtual Platform $\Leftrightarrow$ Hardware Emulation Interface and Methodology	249
12.4.1	Different Types of Hardware/Software Co-verification Configurations	250

12.5	Hardware/Software Co-verification Using Virtual Platform with Hardware Accelerator	251
12.5.1	Cadence Palladium	252
12.5.2	Mentor Veloce	252
12.5.3	Synopsys Zebu.	253
13	Analog/Mixed Signal (AMS) Verification	255
13.1	Overview	255
13.2	Major AMS Verification Challenges and Solutions	256
13.2.1	Disparate Methodologies	256
13.2.2	Analog Model Abstractions and Simulation Performance	257
13.2.3	Low-Power Management	261
13.3	Real Number Modeling (RNM) of Analog Blocks	264
13.3.1	“wreal”	265
13.3.2	“nettype”	267
13.4	AMS Assertion (SVA)-Based Methodology	269
13.5	AMS Simulator: What Features Should It Support?	270
13.5.1	Integrated Simulation Solution for Fastest Simulation Throughput	270
13.5.2	Support for Wide Spectrum of Design Languages	270
13.5.3	Support for Different Levels of Model Abstraction.	271
13.5.4	AMS Low-Power Verification Support	271
13.5.5	Support for SystemVerilog-Based UVM Methodology Including Coverage-Driven and Assertion-Based Methodologies	271
14	SoC Interconnect Verification	273
14.1	Overview	273
14.2	SoC Interconnect Verification: Challenges and Solutions	274
14.2.1	Performance Analysis	276
14.3	Interconnect Functional Correctness and Verification Completeness	277
14.3.1	SoC Interconnect Stimulus Generation	277
14.4	Stress Verification: Random Concurrent Tests	278
14.5	SoC Interconnect Response Checker	278
14.6	SoC Interconnect Coverage Measurement	279
14.7	Cadence® Interconnect Solution (Cadence-VIP <i>n.d.</i>)	280
14.7.1	Cadence ® Interconnect Validator (Basic)	280
14.7.2	Cadence ® Interconnect Validator (Cache Coherent)	282
14.7.3	Cadence ® Interconnect Workbench	282
14.8	Synopsys Cache Coherent Subsystem Verification Solution for Arteris Ncore Interconnect (NoC)	282

15	The Complete Product Design Life Cycle	285
15.1	Overview	285
15.2	Product Design and Development Flow	286
15.2.1	Design Specification	286
15.3	PCB (Printed Circuit Board) Design	288
15.3.1	Schematic Design	288
15.3.2	Pre-layout Signal Integrity (SI), Power Integrity (PI), and Thermal Integrity (TI) Simulation	288
15.3.3	Layout, Fabrication, and Assembly	291
15.3.4	Post-layout Signal Integrity (SI), Power Integrity (PI), and Thermal Integrity (TI) Simulation	292
15.3.5	Hardware Bring-Up and Debug	292
15.4	ASIC/FPGA Design	294
15.4.1	HDL Design	294
15.4.2	Pre-synthesis Simulation	295
15.4.3	Post-synthesis/Place and Route Simulation, Timing Closure	295
15.4.4	Integration	296
15.5	Verification	296
15.5.1	Test Plan Specification	296
15.5.2	Testbench and Test Program Development	297
15.5.3	Functional Test	298
15.5.4	Gate-Level Verification	298
15.5.5	Functional/Gate Regression	299
15.6	Emulation	300
16	Voice Over IP (VoIP) Network SoC Verification	301
16.1	Voice Over IP (VoIP) Network SoC	301
16.2	VoIP Network SoC Verification Plan	301
16.3	Identify Subsystems Within VoIP Network SoC	302
16.4	Determine Subsystem Stimulus and Response Methodology	303
16.5	SoC Interconnect Verification	305
16.6	Low-Power Verification	306
16.7	Static Formal or Static + Simulation Hybrid Methodology	306
16.8	Assertion Methodology	307
16.9	Functional Coverage	307
16.10	Software/Hardware Co-verification	308
16.11	Simulation Regressions: Hardware Acceleration or Emulation or FPGA Prototyping	308
16.12	Virtual Platform	309
17	Cache Memory Subsystem Verification: UVM Agent Based	311
17.1	Cache Subsystem	311
17.2	Identify Subsystems Within the Cache Subsystem	312
17.3	Determine Subsystem Stimulus and Response Methodology	313
17.4	Cache Subsystem Interconnect Verification	315

17.5 Low-Power Verification.	315
17.6 Static Formal or Static + Simulation Hybrid.	316
17.7 Assertions Methodology (SVA)	316
17.8 Coverage Methodology (Functional: SFC, Code, and SVA “cover”)	317
17.9 Software/Hardware Co-verification.	317
17.10 Simulation Regressions: Hardware Acceleration or Emulation or FPGA Prototyping.	318
17.11 ESL: Virtual Platform for Software and Test Development.	318
18 Cache Memory Subsystem Verification: ISS Based.	319
Bibliography	323
Index.	325

<http://www.springer.com/978-3-319-59417-0>

ASIC/SoC Functional Design Verification
A Comprehensive Guide to Technologies and
Methodologies

Mehta, A.B.

2018, XXXI, 328 p. 175 illus., 160 illus. in color.,

Hardcover

ISBN: 978-3-319-59417-0