

INTERNATIONAL STANDARD

IEC 61691-4

First edition
2004-10

IEEE 1364™

Behavioural languages –

Part 4: Verilog® hardware description language

Copyright © IEEE 2004 — All rights reserved

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Inc.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

The Institute of Electrical and Electronics Engineers, Inc, 3 Park Avenue, New York, NY 10016-5997, USA
Telephone: +1 732 562 3800 Telefax: +1 732 562 1571 E-mail: stds-info@ieee.org Web: www.standards.ieee.org



Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия



CONTENTS

FOREWORD	19
IEEE Introduction.....	23
1. Overview.....	25
1.1 Objectives of this standard.....	25
1.2 Conventions used in this standard.....	25
1.3 Syntactic description.....	26
1.4 Contents of this standard.....	26
1.5 Header file listings	28
1.6 Examples.....	29
1.7 Prerequisites.....	29
2. Lexical conventions	30
2.1 Lexical tokens	30
2.2 White space.....	30
2.3 Comments	30
2.4 Operators.....	30
2.5 Numbers.....	30
2.5.1 Integer constants	31
2.5.2 Real constants	34
2.5.3 Conversion	34
2.6 Strings.....	34
2.6.1 String variable declaration.....	35
2.6.2 String manipulation.....	35
2.6.3 Special characters in strings.....	35
2.7 Identifiers, keywords, and system names	36
2.7.1 Escaped identifiers	36
2.7.2 Generated identifiers.....	37
2.7.3 Keywords	37
2.7.4 System tasks and functions.....	37
2.7.5 Compiler directives.....	38
2.8 Attributes.....	38
2.8.1 Examples.....	39
2.8.2 Syntax	40
3. Data types.....	44
3.1 Value sets	44
3.2 Nets and variables.....	44
3.2.1 Net declarations	44
3.2.2 Variable declarations	46
3.3 Vectors.....	47
3.3.1 Specifying vectors.....	47
3.3.2 Vector net accessibility	48
3.4 Strengths	48
3.4.1 Charge strength.....	48
3.4.2 Drive strength.....	48
3.5 Implicit declarations.....	49
3.6 Net initialization.....	49
3.7 Net types	49
3.7.1 Wire and tri nets.....	49
3.7.2 Wired nets	50

3.7.3	Trireg net.....	50
3.7.4	Tri0 and tri1 nets.....	54
3.7.5	Supply nets.....	55
3.8	regs.....	55
3.9	Integers, reals, times, and realsimes.....	55
3.9.1	Operators and real numbers.....	56
3.9.2	Conversion.....	56
3.10	Arrays.....	57
3.10.1	Net arrays.....	57
3.10.2	reg and variable arrays.....	57
3.10.3	Memories.....	57
3.11	Parameters.....	59
3.11.1	Module parameters.....	59
3.11.2	Local parameters—localparam.....	60
3.11.3	Specify parameters.....	61
3.12	Name spaces.....	62
4.	Expressions.....	64
4.1	Operators.....	64
4.1.1	Operators with real operands.....	65
4.1.2	Binary operator precedence.....	66
4.1.3	Using integer numbers in expressions.....	67
4.1.4	Expression evaluation order.....	67
4.1.5	Arithmetic operators.....	68
4.1.6	Arithmetic expressions with regs and integers.....	69
4.1.7	Relational operators.....	70
4.1.8	Equality operators.....	70
4.1.9	Logical operators.....	71
4.1.10	Bit-wise operators.....	71
4.1.11	Reduction operators.....	72
4.1.12	Shift operators.....	73
4.1.13	Conditional operator.....	74
4.1.14	Concatenations.....	75
4.1.15	Event or.....	76
4.2	Operands.....	76
4.2.1	Vector bit select and part-select addressing.....	76
4.2.2	Array and memory addressing.....	78
4.2.3	Strings.....	79
4.3	Minimum, typical, and maximum delay expressions.....	81
4.4	Expression bit lengths.....	83
4.4.1	Rules for expression bit lengths.....	83
4.4.2	An example of an expression bit-length problem.....	84
4.4.3	Example of self-determined expressions.....	85
4.5	Signed expressions.....	86
4.5.1	Rules for expression types.....	86
4.5.2	Steps for evaluating an expression.....	86
4.5.3	Steps for evaluating an assignment.....	87
4.5.4	Handling X and Z in signed expressions.....	87
5.	Scheduling semantics.....	88
5.1	Execution of a model.....	88
5.2	Event simulation.....	88

5.3	The stratified event queue	88
5.4	The Verilog simulation reference model	89
5.4.1	Determinism.....	90
5.4.2	Nondeterminism.....	90
5.5	Race conditions.....	90
5.6	Scheduling implication of assignments	90
5.6.1	Continuous assignment.....	91
5.6.2	Procedural continuous assignment.....	91
5.6.3	Blocking assignment.....	91
5.6.4	Nonblocking assignment.....	91
5.6.5	Switch (transistor) processing.....	91
5.6.6	Port connections.....	92
5.6.7	Functions and tasks.....	92
6.	Assignments.....	93
6.1	Continuous assignments.....	93
6.1.1	The net declaration assignment.....	94
6.1.2	The continuous assignment statement	94
6.1.3	Delays	96
6.1.4	Strength.....	96
6.2	Procedural assignments.....	97
6.2.1	Variable declaration assignment.....	97
6.2.2	Variable declaration syntax.....	98
7.	Gate and switch level modeling.....	99
7.1	Gate and switch declaration syntax.....	99
7.1.1	The gate type specification	101
7.1.2	The drive strength specification.....	101
7.1.3	The delay specification	102
7.1.4	The primitive instance identifier.....	102
7.1.5	The range specification	102
7.1.6	Primitive instance connection list	103
7.2	and, nand, nor, or, xor and xnor gates.....	105
7.3	buf and not gates	106
7.4	bufif1, bufif0, norif1, and notif0 gates.....	107
7.5	MOS switches.....	109
7.6	Bidirectional pass switches.....	110
7.7	CMOS switches	110
7.8	pullup and pulldown sources	111
7.9	Logic strength modeling	112
7.10	Strengths and values of combined signals	113
7.10.1	Combined signals of unambiguous strength.....	113
7.10.2	Ambiguous strengths: sources and combinations	114
7.10.3	Ambiguous strength signals and unambiguous signals	119
7.10.4	Wired logic net types	123
7.11	Strength reduction by nonresistive devices.....	126
7.12	Strength reduction by resistive devices.....	126
7.13	Strengths of net types.....	126
7.13.1	tri0 and tri1 net strengths	126
7.13.2	triereg strength.....	126
7.13.3	supply0 and supply1 net strengths	126

7.14	Gate and net delays	127
7.14.1	min:typ:max delays	128
7.14.2	trireg net charge decay	129
8.	User-defined primitives (UDPs)	131
8.1	UDP definition	131
8.1.1	UDP header	133
8.1.2	UDP port declarations	133
8.1.3	Sequential UDP initial statement	133
8.1.4	UDP state table	133
8.1.5	Z values in UDP	134
8.1.6	Summary of symbols	134
8.2	Combinational UDPs	135
8.3	Level-sensitive sequential UDPs	136
8.4	Edge-sensitive sequential UDPs	136
8.5	Sequential UDP initialization	137
8.6	UDP instances	139
8.7	Mixing level-sensitive and edge-sensitive descriptions	140
8.8	Level-sensitive dominance	141
9.	Behavioral modeling	142
9.1	Behavioral model overview	142
9.2	Procedural assignments	143
9.2.1	Blocking procedural assignments	143
9.2.2	The nonblocking procedural assignment	145
9.3	Procedural continuous assignments	148
9.3.1	The assign and deassign procedural statements	149
9.3.2	The force and release procedural statements	150
9.4	Conditional statement	151
9.4.1	If-else-if construct	152
9.5	Case statement	154
9.5.1	Case statement with don't-cares	157
9.5.2	Constant expression in case statement	157
9.6	Looping statements	158
9.7	Procedural timing controls	160
9.7.1	Delay control	161
9.7.2	Event control	162
9.7.3	Named events	162
9.7.4	Event or operator	163
9.7.5	Implicit event_expression list	164
9.7.6	Level-sensitive event control	165
9.7.7	Intra-assignment timing controls	166
9.8	Block statements	170
9.8.1	Sequential blocks	170
9.8.2	Parallel blocks	171
9.8.3	Block names	172
9.8.4	Start and finish times	172
9.9	Structured procedures	173
9.9.1	Initial construct	174
9.9.2	Always construct	174

10.	Tasks and functions.....	176
10.1	Distinctions between tasks and functions	176
10.2	Tasks and task enabling	176
10.2.1	Task declarations	177
10.2.2	Task enabling and argument passing	178
10.2.3	Task memory usage and concurrent activation.....	180
10.3	Functions and function calling.....	181
10.3.1	Function declarations	182
10.3.2	Returning a value from a function	183
10.3.3	Calling a function.....	184
10.3.4	Function rules.....	185
10.3.5	Use of constant functions.....	185
11.	Disabling of named blocks and tasks.....	187
12.	Hierarchical structures	190
12.1	Modules.....	190
12.1.1	Top-level modules	192
12.1.2	Module instantiation	192
12.1.3	Generated instantiation	194
12.2	Overriding module parameter values.....	204
12.2.1	defparam statement	206
12.2.2	Module instance parameter value assignment	207
12.2.3	Parameter dependence	209
12.3	Ports	209
12.3.1	Port definition	209
12.3.2	List of ports.....	209
12.3.3	Port declarations.....	210
12.3.4	List of ports declarations.....	212
12.3.5	Connecting module instance ports by ordered list.....	212
12.3.6	Connecting module instance ports by name	213
12.3.7	Real numbers in port connections.....	214
12.3.8	Connecting dissimilar ports	215
12.3.9	Port connection names.....	215
12.3.10	Net type resulting from dissimilar port connections	216
12.3.11	Connecting signed values via ports.....	217
12.4	Hierarchical names	217
12.5	Upward name referencing	220
12.6	Scope rules	222
13.	Configuring the contents of a design	224
13.1	Introduction.....	224
13.1.1	Library notation	224
13.1.2	Basic configuration elements.....	225
13.2	Libraries	225
13.2.1	Specifying libraries - the library map file.....	225
13.2.2	Using multiple library mapping files	227
13.2.3	Mapping source files to libraries.....	227
13.3	Configurations.....	227
13.3.1	Basic configuration syntax.....	227
13.3.2	Hierarchical configurations.....	230

13.4	Using libraries and configs	231
13.4.1	Precompiling in a single-pass use-model.....	231
13.4.2	Elaboration-time compiling in a single-pass use-model.....	231
13.4.3	Precompiling using a separate compilation tool	231
13.4.4	Command line considerations.....	231
13.5	Configuration examples	232
13.5.1	Default configuration from library map file	232
13.5.2	Using the default clause.....	232
13.5.3	Using the cell clause	233
13.5.4	Using the instance clause.....	233
13.5.5	Using a hierarchical config.....	233
13.6	Displaying library binding information	233
13.7	Library mapping examples	234
13.7.1	Using the command line to control library searching.....	234
13.7.2	File path specification examples.....	234
13.7.3	Resolving multiple path specifications	235
14.	Specify blocks.....	236
14.1	Specify block declaration.....	236
14.2	Module path declarations.....	237
14.2.1	Module path restrictions	238
14.2.2	Simple module paths.....	238
14.2.3	Edge-sensitive paths.....	239
14.2.4	State-dependent paths	240
14.2.5	Full connection and parallel connection paths.....	244
14.2.6	Declaring multiple module paths in a single statement	245
14.2.7	Module path polarity.....	246
14.3	Assigning delays to module paths.....	247
14.3.1	Specifying transition delays on module paths	248
14.3.2	Specifying x transition delays.....	249
14.3.3	Delay selection.....	250
14.4	Mixing module path delays and distributed delays.....	251
14.5	Driving wired logic.....	252
14.6	Detailed control of pulse filtering behavior.....	253
14.6.1	Specify block control of pulse limit values.....	254
14.6.2	Global control of pulse limit values.....	255
14.6.3	SDF annotation of pulse limit values.....	255
14.6.4	Detailed pulse control capabilities.....	256
15.	Timing checks.....	262
15.1	Overview.....	262
15.2	Timing checks using a stability window.....	265
15.2.1	\$setup.....	266
15.2.2	\$hold	266
15.2.3	\$setuphold.....	267
15.2.4	\$removal	269
15.2.5	\$recovery.....	270
15.2.6	\$crem	271
15.3	Timing checks for clock and control signals	273
15.3.1	\$skew	274
15.3.2	\$timeskew	275
15.3.3	\$fullskew.....	277

15.3.4	\$width	279
15.3.5	\$period	280
15.3.6	\$nochange	281
15.4	Edge-control specifiers	283
15.5	Notifiers: user-defined responses to timing violations	284
15.5.1	Requirements for accurate simulation	286
15.5.2	Conditions in negative timing checks	288
15.5.3	Notifiers in negative timing checks	290
15.5.4	Option behavior	290
15.6	Enabling timing checks with conditioned events.....	290
15.7	Vector signals in timing checks	291
15.8	Negative timing checks.....	292
16.	Backannotation using the Standard Delay Format (SDF).....	294
16.1	The SDF annotator	294
16.2	Mapping of SDF constructs to Verilog.....	294
16.2.1	Mapping of SDF delay constructs to Verilog declarations.....	294
16.2.2	Mapping of SDF timing check constructs to Verilog	296
16.2.3	SDF annotation of speparams	297
16.2.4	SDF annotation of interconnect delays	298
16.3	Multiple annotations	299
16.4	Multiple SDF files.....	300
16.5	Pulse limit annotation	300
16.6	SDF to Verilog delay value mapping.....	301
17.	System tasks and functions	302
17.1	Display system tasks	302
17.1.1	The display and write tasks.....	303
17.1.2	Strobed monitoring	310
17.1.3	Continuous monitoring	311
17.2	File input-output system tasks and functions.....	311
17.2.1	Opening and closing files	311
17.2.2	File output system tasks	313
17.2.3	Formatting data to a string	314
17.2.4	Reading data from a file.....	315
17.2.5	File positioning	319
17.2.6	Flushing output	319
17.2.7	I/O error status	319
17.2.8	Loading memory data from a file	320
17.2.9	Loading timing data from an SDF file.....	321
17.3	Timescale system tasks	322
17.3.1	\$prnttimescale.....	322
17.3.2	\$timeformat.....	323
17.4	Simulation control system tasks.....	326
17.4.1	\$finish	326
17.4.2	\$stop.....	326
17.5	PLA modeling system tasks.....	327
17.5.1	Array types.....	327
17.5.2	Array logic types.....	328
17.5.3	Logic array personality declaration and loading.....	328
17.5.4	Logic array personality formats	328
17.6	Stochastic analysis tasks	331

17.6.1	\$q_initialize.....	331
17.6.2	\$q_add.....	332
17.6.3	\$q_remove.....	332
17.6.4	\$q_full.....	332
17.6.5	\$q_exam.....	332
17.6.6	Status codes.....	333
17.7	Simulation time system functions.....	333
17.7.1	\$time.....	333
17.7.2	\$stime.....	334
17.7.3	\$realtime.....	334
17.8	Conversion functions.....	335
17.9	Probabilistic distribution functions.....	336
17.9.1	\$random function.....	336
17.9.2	\$dist_ functions.....	337
17.9.3	Algorithm for probabilistic distribution functions.....	338
17.10	Command line input.....	345
17.10.1	\$test\$plusargs (string).....	346
17.10.2	\$value\$plusargs (user_string, variable).....	346
18.	Value change dump (VCD) files.....	349
18.1	Creating the four state value change dump file.....	349
18.1.1	Specifying the name of the dump file (\$dumpfile).....	349
18.1.2	Specifying the variables to be dumped (\$dumpvars).....	350
18.1.3	Stopping and resuming the dump (\$dumpoff/\$dumpson).....	351
18.1.4	Generating a checkpoint (\$dumpall).....	352
18.1.5	Limiting the size of the dump file (\$dumplimit).....	352
18.1.6	Reading the dump file during simulation (\$dumpflush).....	353
18.2	Format of the four state VCD file.....	354
18.2.1	Syntax of the four state VCD file.....	354
18.2.2	Formats of variable values.....	356
18.2.3	Description of keyword commands.....	357
18.2.4	Four state VCD file format example.....	363
18.3	Creating the extended value change dump file.....	364
18.3.1	Specifying the dump file name and the ports to be dumped (\$dumpports).....	364
18.3.2	Stopping and resuming the dump (\$dumpportsoff/\$dumpportson).....	365
18.3.3	Generating a checkpoint (\$dumpportsall).....	366
18.3.4	Limiting the size of the dump file (\$dumpportslimit).....	366
18.3.5	Reading the dump file during simulation (\$dumpportsflush).....	367
18.3.6	Description of keyword commands.....	367
18.3.7	General rules for extended VCD system tasks.....	368
18.4	Format of the extended VCD file.....	368
18.4.1	Syntax of the extended VCD file.....	368
18.4.2	Extended VCD node information.....	370
18.4.3	Value changes.....	372
18.4.4	Extended VCD file format example.....	373
19.	Compiler directives.....	375
19.1	`celldefine and `endcelldefine.....	375
19.2	`default_nettype.....	375
19.3	`define and `undef.....	376
19.3.1	`define.....	376
19.3.2	`undef.....	378

19.4	`ifdef, `else, `elsif, `endif, `ifndef	378
19.5	`include	382
19.6	`resetall.....	382
19.7	`line	383
19.8	`timescale	383
19.9	`unconnected_drive and `nounconnected_drive	385
20.	PLI overview.....	386
20.1	PLI purpose and history (informative).....	386
20.2	User-defined system task or function names	386
20.3	User-defined system task or function types	387
20.4	Overriding built-in system task and function names	387
20.5	User-supplied PLI applications.....	387
20.6	PLI interface mechanism	387
20.7	User-defined system task and function arguments	388
20.8	PLI include files.....	388
20.9	PLI Memory Restrictions.....	388
21.	PLI TF and ACC interface mechanism.....	389
21.1	User-supplied PLI applications.....	389
21.1.1	The sizetf class of PLI applications	389
21.1.2	The checktf class of PLI applications	389
21.1.3	The calltf class of PLI applications.....	390
21.1.4	The mistf class of PLI applications.....	390
21.1.5	The consumer class of PLI applications.....	390
21.2	Associating PLI applications to a class and system task/function name	390
21.3	PLI application arguments	391
21.3.1	The data C argument.....	391
21.3.2	The reason C argument.....	391
21.3.3	The paramvc C argument.....	392
22.	Using ACC routines.....	393
22.1	ACC routine definition	393
22.2	The handle data type	393
22.3	Using ACC routines	394
22.3.1	Header files	394
22.3.2	Initializing ACC routines	394
22.3.3	Extending ACC routines.....	394
22.4	List of ACC routines by major category	394
22.4.1	Fetch routines.....	395
22.4.2	Handle routines	396
22.4.3	Next routines.....	397
22.4.4	Modify routines.....	399
22.4.5	Miscellaneous routines.....	399
22.4.6	VCL routines.....	400
22.5	Accessible objects	400
22.5.1	ACC routines that operate on module instances	402
22.5.2	ACC routines that operate on module ports	402
22.5.3	ACC routines that operate on bits of a port	403
22.5.4	ACC routines that operate on module paths or data paths	403
22.5.5	ACC routines that operate on intermodule paths	404

22.5.6	ACC routines that operate on top-level modules	404
22.5.7	ACC routines that operate on primitive instances	404
22.5.8	ACC routines that operate on primitive terminals	405
22.5.9	ACC routines that operate on nets	405
22.5.10	ACC routines that operate on reg types	406
22.5.11	ACC routines that operate on integer, real, and time variables	406
22.5.12	ACC routines that operate on named events	406
22.5.13	ACC routines that operate on parameters and specparams	407
22.5.14	ACC routines that operate on timing checks	407
22.5.15	ACC routines that operate on timing check terminals	407
22.5.16	ACC routines that operate on user-defined system task/function arguments	408
22.6	ACC routine types and fulltypes	408
22.7	Error handling	411
22.7.1	Suppressing error messages	412
22.7.2	Enabling warnings	412
22.7.3	Testing for errors	412
22.7.4	Example	412
22.7.5	Exception values	413
22.8	Reading and writing delay values	413
22.8.1	Number of delays for Verilog HDL objects	414
22.8.2	ACC routine configuration	414
22.8.3	Determining the number of arguments for ACC delay routines	415
22.9	String handling	419
22.9.1	ACC routines share an internal string buffer	419
22.9.2	String buffer reset	420
22.9.3	Preserving string values	421
22.9.4	Example of preserving string values	421
22.10	Using VCL ACC routines	421
22.10.1	VCL objects	422
22.10.2	The VCL record definition	422
22.10.3	Effects of acc_initialize() and acc_close() on VCL consumer routines	425
22.10.4	An example of using VCL ACC routines	425
23.	ACC routine definitions	428
23.1	acc_append_delays()	429
23.2	acc_append_pulse()	433
23.3	acc_close()	435
23.4	acc_collect()	436
23.5	acc_compare_handles()	438
23.6	acc_configure()	439
23.7	acc_count()	448
23.8	acc_fetch_argc()	449
23.9	acc_fetch_argv()	450
23.10	acc_fetch_attribute()	452
23.11	acc_fetch_attribute_int()	456
23.12	acc_fetch_attribute_str()	457
23.13	acc_fetch_defname()	458
23.14	acc_fetch_delay_mode()	459
23.15	acc_fetch_delays()	461
23.16	acc_fetch_direction()	465
23.17	acc_fetch_edge()	466
23.18	acc_fetch_fullname()	468
23.19	acc_fetch_fulltype()	470

23.20	acc_fetch_index()	473
23.21	acc_fetch_location()	475
23.22	acc_fetch_name()	477
23.23	acc_fetch_paramtype()	479
23.24	acc_fetch_paramval()	480
23.25	acc_fetch_polarity()	482
23.26	acc_fetch_precision()	483
23.27	acc_fetch_pulsere()	484
23.28	acc_fetch_range()	487
23.29	acc_fetch_size()	488
23.30	acc_fetch_tfarg(), acc_fetch_itfarg()	489
23.31	acc_fetch_tfarg_int(), acc_fetch_itfarg_int()	491
23.32	acc_fetch_tfarg_str(), acc_fetch_itfarg_str()	492
23.33	acc_fetch_timescale_info()	493
23.34	acc_fetch_type()	495
23.35	acc_fetch_type_str()	497
23.36	acc_fetch_value()	498
23.37	acc_free()	503
23.38	acc_handle_by_name()	504
23.39	acc_handle_calling_mode_m()	506
23.40	acc_handle_condition()	507
23.41	acc_handle_conn()	508
23.42	acc_handle_datapath()	509
23.43	acc_handle_hiconn()	510
23.44	acc_handle_interactive_scope()	512
23.45	acc_handle_loconn()	513
23.46	acc_handle_modpath()	514
23.47	acc_handle_noti.er()	516
23.48	acc_handle_object()	517
23.49	acc_handle_parent()	519
23.50	acc_handle_path()	520
23.51	acc_handle_pathin()	521
23.52	acc_handle_pathout()	522
23.53	acc_handle_port()	523
23.54	acc_handle_scope()	525
23.55	acc_handle_simulated_net()	526
23.56	acc_handle_tchk()	528
23.57	acc_handle_tchkarg1()	532
23.58	acc_handle_tchkarg2()	534
23.59	acc_handle_terminal()	535
23.60	acc_handle_tfarg(), acc_handle_itfarg()	536
23.61	acc_handle_t.nst()	538
23.62	acc_initialize()	539
23.63	acc_next()	540
23.64	acc_next_bit()	544
23.65	acc_next_cell()	546
23.66	acc_next_cell_load()	547
23.67	acc_next_child()	549
23.68	acc_next_driver()	550
23.69	acc_next_hiconn()	551
23.70	acc_next_input()	553
23.71	acc_next_load()	555
23.72	acc_next_loconn()	557
23.73	acc_next_modpath()	558

23.74	acc_next_net()	559
23.75	acc_next_output()	560
23.76	acc_next_parameter()	562
23.77	acc_next_port()	563
23.78	acc_next_portout()	565
23.79	acc_next_primitive()	566
23.80	acc_next_scope()	567
23.81	acc_next_specparam()	568
23.82	acc_next_tchk()	569
23.83	acc_next_terminal()	571
23.84	acc_next_topmod()	572
23.85	acc_object_in_typelist()	573
23.86	acc_object_of_type()	575
23.87	acc_product_type()	577
23.88	acc_product_version()	579
23.89	acc_release_object()	580
23.90	acc_replace_delays()	581
23.91	acc_replace_pulsere()	585
23.92	acc_reset_buffer()	588
23.93	acc_set_interactive_scope()	589
23.94	acc_set_pulsere()	590
23.95	acc_set_scope()	592
23.96	acc_set_value()	594
23.97	acc_vcl_add()	599
23.98	acc_vcl_delete()	601
23.99	acc_version()	602
24.	Using TF routines	603
24.1	TF routine definition	603
24.2	TF routine system task/function arguments	603
24.3	Reading and writing system task/function argument values	603
24.3.1	Reading and writing 2-state parameter argument values	603
24.3.2	Reading and writing 4-state values	603
24.3.3	Reading and writing strength values	604
24.3.4	Reading and writing to memories	604
24.3.5	Reading and writing string values	604
24.3.6	Writing return values of user-defined functions	604
24.3.7	Writing the correct C data types	604
24.4	Value change detection	605
24.5	Simulation time	605
24.6	Simulation synchronization	605
24.7	Instances of user-defined tasks or functions	606
24.8	Module and scope instance names	606
24.9	Saving information from one system TF call to the next	606
24.10	Displaying output messages	606
24.11	Stopping and finishing	606
25.	TF routine definitions	607
25.1	io_mcdprintf()	608
25.2	io_printf()	609
25.3	mc_scan_plusargs()	610
25.4	tf_add_long()	611

25.5	tf_asynchoff(), tf_iasynchoff()	612
25.6	tf_asynchon(), tf_iasynchon()	613
25.7	tf_clearalldelays(), tf_icallearalldelays()	614
25.8	tf_compare_long()	615
25.9	tf_copypvc_ag(), tf_ico pypvc_ag()	616
25.10	tf_divide_long()	617
25.11	tf_do.nish()	618
25.12	tf_dostop()	619
25.13	tf_error()	620
25.14	tf_evaluatep(), tf_ievaluatep()	621
25.15	tf_exprinfo(), tf_iexprinfo()	622
25.16	tf_getcstringp(), tf_igetcstringp()	625
25.17	tf_getinstance()	626
25.18	tf_getlongp(), tf_igetlongp()	627
25.19	tf_getlongtime(), tf_igetlongtime()	628
25.20	tf_getnextlongtime()	629
25.21	tf_getp(), tf_igetp()	630
25.22	tf_getpchange(), tf_igetpchange()	631
25.23	tf_getrealp(), tf_igetrealp()	632
25.24	tf_getrealtime(), tf_igetrealtime()	633
25.25	tf_gettime(), tf_igettime()	634
25.26	tf_gettimeprecision(), tf_igettimeprecision()	635
25.27	tf_gettimeunit(), tf_igettimeunit()	636
25.28	tf_getworkarea(), tf_igetworkarea()	637
25.29	tf_long_to_real()	638
25.30	tf_longtime_tostr()	639
25.31	tf_message()	640
25.32	tf_mipname(), tf_imipname()	642
25.33	tf_movepvc_ag(), tf_im ovepvc_ag()	643
25.34	tf_multiply_long()	644
25.35	tf_nodeinfo(), tf_inodeinfo()	645
25.36	tf_nump(), tf_inump()	649
25.37	tf_propagatep(), tf_ipropagatep()	650
25.38	tf_putlongp(), tf_iputlongp()	651
25.39	tf_putp(), tf_iputp()	652
25.40	tf_putrealp(), tf_iputrealp()	653
25.41	tf_read_restart()	654
25.42	tf_real_to_long()	655
25.43	tf_rosynchronize(), tf_irosynchronize()	656
25.44	tf_scale_longdelay()	657
25.45	tf_scale_realdelay()	658
25.46	tf_setdelay(), tf_isetdelay()	659
25.47	tf_setlongdelay(), tf_isetlongdelay()	660
25.48	tf_setrealdelay(), tf_isetrealdelay()	661
25.49	tf_setworkarea(), tf_isetworkarea()	662
25.50	tf_sizep(), tf_ysizep()	663
25.51	tf_spname(), tf_ispname()	664
25.52	tf_strdelputp(), tf_istrdelputp()	665
25.53	tf_strgetp(), tf_istrgetp()	667
25.54	tf_strgettime()	668
25.55	tf_strlongdelputp(), tf_istrlongdelputp()	669
25.56	tf_strrealdelputp(), tf_istrrealdelputp()	671
25.57	tf_subtract_long()	673
25.58	tf_synchronize(), tf_isynchronize()	675

25.59	tf_testpvc_.ag(), tf_itestpvc_.ag().....	676
25.60	tf_text()	677
25.61	tf_typep(), tf_itypep()	678
25.62	tf_unscale_longdelay().....	679
25.63	tf_unscale_realdelay().....	680
25.64	tf_warning()	681
25.65	tf_write_save()	682
26.	Using VPI routines.....	683
26.1	VPI system tasks and functions	683
26.2	The VPI interface	683
26.2.1	VPI callbacks	683
26.2.2	VPI access to Verilog HDL objects and simulation objects	684
26.2.3	Error handling	684
26.2.4	Function availability	684
26.2.5	Traversing expressions.....	684
26.3	VPI object classifications.....	685
26.3.1	Accessing object relationships and properties	686
26.3.2	Object type properties	687
26.3.3	Object file and line properties.....	687
26.3.4	Delays and values	688
26.4	List of VPI routines by functional category.....	688
26.5	Key to data model diagrams	690
26.5.1	Diagram key for objects and classes	691
26.5.2	Diagram key for accessing properties.....	691
26.5.3	Diagram key for traversing relationships	692
26.6	Object data model diagrams.....	693
26.6.1	Module	694
26.6.2	Instance arrays	695
26.6.3	Scope	696
26.6.4	IO declaration	696
26.6.5	Ports	697
26.6.6	Nets and net arrays.....	698
26.6.7	Regs and reg arrays.....	700
26.6.8	IO declaration	702
26.6.9	Memory	703
26.6.10	IO declaration	704
26.6.11	Named event	704
26.6.12	Parameter, specparam	705
26.6.13	Primitive, prim term.....	706
26.6.14	UDP.....	707
26.6.15	Model path, path term.....	708
26.6.16	Intermodule path	708
26.6.17	Timing check	709
26.6.18	Task, function declaration	709
26.6.19	Task and function call.....	710
26.6.20	Frames.....	711
26.6.21	Delay terminals	712
26.6.22	Net drivers and loads	712
26.6.23	Reg drivers and loads	712
26.6.24	Continuous assignment	713
26.6.25	Simple expressions.....	714
26.6.26	Expressions	715

26.6.27	Process, block, statement, event statement	716
26.6.28	Assignment	717
26.6.29	Delay control	717
26.6.30	Event control	717
26.6.31	Repeat control	717
26.6.32	While, repeat, wait	718
26.6.33	For	718
26.6.34	Forever	718
26.6.35	If, if-else	719
26.6.36	Case	719
26.6.37	Assign statement, deassign, force, release	720
26.6.38	Disable	720
26.6.39	Callback	721
26.6.40	Time queue	721
26.6.41	Active time format	721
26.6.42	Attributes.....	722
26.6.43	Iterator.....	723
27.	VPI routine definitions.....	724
27.1	vpi_chk_error()	725
27.2	vpi_compare_objects().....	726
27.3	vpi_control()	727
27.4	vpi_flush().....	728
27.5	vpi_free_object().....	729
27.6	vpi_get().....	730
27.7	vpi_get_cb_info().....	731
27.8	vpi_get_data()	732
27.9	vpi_get_delays().....	734
27.10	vpi_get_str().....	737
27.11	vpi_get_systf_info().....	738
27.12	vpi_get_time().....	739
27.13	vpi_get_userdata().....	740
27.14	vpi_get_value()	741
27.15	vpi_get_vlog_info()	747
27.16	vpi_handle()	748
27.17	vpi_handle_by_index()	749
27.18	vpi_handle_by_multi_index().....	750
27.19	vpi_handle_by_name()	751
27.20	vpi_handle_multi().....	752
27.21	vpi_iterate().....	753
27.22	vpi_mcd_close().....	754
27.23	vpi_mcd_flush()	755
27.24	vpi_mcd_name()	756
27.25	vpi_mcd_open()	757
27.26	vpi_mcd_printf()	758
27.27	vpi_mcd_vprintf().....	759
27.28	vpi_printf().....	760
27.29	vpi_put_data().....	761
27.30	vpi_put_delays()	763
27.31	vpi_put_userdata()	766
27.32	vpi_put_value().....	767
27.33	vpi_register_cb().....	770
27.33.1	Simulation-event-related callbacks	771

27.33.2 Simulation-time-related callbacks	774
27.33.3 Simulator action and feature related callbacks	775
27.34 vpi_register_systf()	778
27.34.1 System task and function callbacks	779
27.34.2 Initializing VPI system task/function callbacks.....	780
27.34.3 Registering multiple system tasks and functions	781
27.35 vpi_remove_cb()	782
27.36 vpi_scan().....	783
27.37 vpi_vprintf().....	784
Annex A (normative) Formal syntax definition.....	785
A.1 Source text	785
A.1.1 Library source text	785
A.1.2 Configuration source text.....	785
A.1.3 Module and primitive source text	786
A.1.4 Module parameters and ports	786
A.1.5 Module items.....	786
A.2 Declarations	787
A.2.1 Declaration types.....	787
A.2.2 Declaration data types.....	789
A.2.3 Declaration lists.....	789
A.2.4 Declaration assignments	790
A.2.5 Declaration ranges.....	790
A.2.6 Function declarations	790
A.2.7 Task declarations.....	790
A.2.8 Block item declarations.....	791
A.3 Primitive instances	792
A.3.1 Primitive instantiation and instances.....	792
A.3.2 Primitive strengths	792
A.3.3 Primitive terminals.....	793
A.3.4 Primitive gate and switch types	793
A.4 Module and generated instantiation	793
A.4.1 Module instantiation	793
A.4.2 Generated instantiation	793
A.5 UDP declaration and instantiation	794
A.5.1 UDP declaration.....	794
A.5.2 UDP ports.....	794
A.5.3 UDP body.....	795
A.5.4 UDP instantiation.....	795
A.6 Behavioral statements	795
A.6.1 Continuous assignment statements	795
A.6.2 Procedural blocks and assignments.....	795
A.6.3 Parallel and sequential blocks	796
A.6.4 Statements	796
A.6.5 Timing control statements.....	796
A.6.6 Conditional statements.....	797
A.6.7 Case statements	798
A.6.8 Looping statements	798
A.6.9 Task enable statements.....	798
A.7 Specify section	798
A.7.1 Specify block declaration.....	798
A.7.2 Specify path declarations	799
A.7.3 Specify block terminals.....	799

A.7.4 Specify path delays	795
A.7.5 System timing checks.....	801
A.8 Expressions	803
A.8.1 Concatenations	803
A.8.2 Function calls	803
A.8.3 Expressions	804
A.8.4 Primaries	805
A.8.5 Expression left-side values.....	805
A.8.6 Operators.....	806
A.8.7 Numbers	806
A.8.8 Strings	807
A.9 General.....	807
A.9.1 Attributes.....	807
A.9.2 Comments	807
A.9.3 Identifiers	807
A.9.4 Identifier branches.....	808
A.9.5 White space	809
 Annex B (normative) List of keywords	 810
 Annex C (informative) System tasks and functions	 812
C.1 \$countdrivers.....	812
C.2 \$getpattern.....	813
C.3 \$input	814
C.4 \$key and \$nokey	814
C.5 \$list.....	815
C.6 \$log and \$nolog.....	815
C.7 \$reset, \$reset_count, and \$reset_value.....	815
C.8 \$save, \$restart, and \$incsave.....	816
C.9 \$scale.....	817
C.10 \$scope	817
C.11 \$showscopes	817
C.12 \$showvars	818
C.13 \$readmemb and \$readmemh.....	818
 Annex D (informative) Compiler directives	 819
D.1 `default_decay_time.....	819
D.2 `default_triireg_strength.....	819
D.3 `delay_mode_distributed.....	820
D.4 `delay_mode_path.....	820
D.5 `delay_mode_unit	820
D.6 `delay_mode_zero.....	820
 Annex E (normative) acc_user.h.....	 821
 Annex F (normative) veriusers.h.....	 830
 Annex G (normative) vpi_user.h.....	 838
 Annex H (informative) Bibliography.....	 852
 Annex I (informative) List of Participants.....	 853

INTERNATIONAL ELECTROTECHNICAL COMMISSION

BEHAVIOURAL LANGUAGES –

Part 4: Verilog® hardware description language

FOREWORD

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
- 6) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC/IEEE 61691-4 has been processed through IEC technical committee 93: Design automation.

The text of this standard is based on the following documents:

IEEE Std	FDIS	Report on voting
1364 (2001)	93/192/FDIS	93/197/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives.

The committee has decided that the contents of this publication will remain unchanged until 2006.

IEC 61691 consists of the following parts, under the general title *Behavioural languages*:

IEC/IEEE 61691-1-1, Part 1: *VHDL language reference manual*

IEC 61691-2, Part 2: *VHDL multilogic system for model interoperability*

IEC 61691-3-1, Part 3-1: *Analog description in VHDL* (under consideration)

IEC 61691-3-2, Part 3-2: *Mathematical operation in VHDL*

IEC 61691-3-3, Part 3-3: *Synthesis in VHDL*

IEC 61691-3-4, Part 3-4: *Timing expressions in VHDL* (under consideration)

IEC 61691-3-5, Part 3-5: *Library utilities in VHDL* (under consideration)

IEC/IEEE 61691-4, Part 4: *Verilog® hardware description language*

IEC/IEEE 61691-5, Part 5: *VITAL ASIC (application specific integrated circuit) modeling specification*

IEC/IEEE Dual Logo International Standards

This Dual Logo International Standard is the result of an agreement between the IEC and the Institute of Electrical and Electronics Engineers, Inc. (IEEE). The original IEEE Standard was submitted to the IEC for consideration under the agreement, and the resulting IEC/IEEE Dual Logo International Standard has been published in accordance with the ISO/IEC Directives.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEC/IEEE Dual Logo International Standard is wholly voluntary. The IEC and IEEE disclaim liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEC or IEEE Standard document.

The IEC and IEEE do not warrant or represent the accuracy or content of the material contained herein, and expressly disclaim any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEC/IEEE Dual Logo International Standards documents are supplied "AS IS".

The existence of an IEC/IEEE Dual Logo International Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEC/IEEE Dual Logo International Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEC and IEEE are not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Neither the IEC nor IEEE is undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEC/IEEE Dual Logo International Standards or IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations – Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEC/IEEE Dual Logo International Standards are welcome from any interested party, regardless of membership affiliation with the IEC or IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA and/or General Secretary, IEC, 3, rue de Varembe, PO Box 131, 1211 Geneva 20, Switzerland.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

NOTE – Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE Standard Verilog[®] Hardware Description Language

Sponsor

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 17 March 2001

IEEE-SA Standards Board

Abstract: The Verilog[®] Hardware Description Language (HDL) is defined in this standard. Verilog HDL is a formal notation intended for use in all phases of the creation of electronic systems. Because it is both machine readable and human readable, it supports the development, verification, synthesis, and testing of hardware designs; the communication of hardware design data; and the maintenance, modification, and procurement of hardware. The primary audiences for this standard are the implementors of tools supporting the language and advanced users of the language.

Keywords: computer, computer languages, digital systems, electronic systems, hardware, hardware description languages, hardware design, HDL, PLI, programming language interface, Verilog HDL, Verilog PLI, Verilog[®]

IEEE Introduction

The Verilog®* Hardware Description Language (Verilog HDL) became an IEEE standard in 1995 as IEEE Std 1364-1995. It was designed to be simple, intuitive, and effective at multiple levels of abstraction in a standard textual format for a variety of design tools, including verification simulation, timing analysis, test analysis, and synthesis. It is because of these rich features that Verilog has been accepted to be the language of choice by an overwhelming number of IC designers.

Verilog contains a rich set of built-in primitives, including logic gates, user-definable primitives, switches, and wired logic. It also has device pin-to-pin delays and timing checks. The mixing of abstract levels is essentially provided by the semantics of two data types: nets and variables. Continuous assignments, in which expressions of both variables and nets can continuously drive values onto nets, provide the basic structural construct. Procedural assignments, in which the results of calculations involving variable and net values can be stored into variables, provide the basic behavioral construct. A design consists of a set of modules, each of which has an I/O interface, and a description of its function, which can be structural, behavioral, or a mix. These modules are formed into a hierarchy and are interconnected with nets.

The Verilog language is extensible via the Programming Language Interface (PLI) and the Verilog Procedural Interface (VPI) routines. The PLI/VPI is a collection of routines that allows foreign functions to access information contained in a Verilog HDL description of the design and facilitates dynamic interaction with simulation. Applications of PLI/VPI include connecting to a Verilog HDL simulator with other simulation and CAD systems, customized debugging tasks, delay calculators, and annotators.

The language that influenced Verilog HDL the most was HILO-2, which was developed at Brunel University in England under a contract to produce a test generation system for the British Ministry of Defense. HILO-2 successfully combined the gate and register transfer levels of abstraction and supported verification simulation, timing analysis, fault simulation, and test generation.

In 1990, Cadence Design Systems placed the Verilog HDL into the public domain and the independent Open Verilog International (OVI) was formed to manage and promote Verilog HDL. In 1992, the Board of Directors of OVI began an effort to establish Verilog HDL as an IEEE standard. In 1993, the first IEEE Working Group was formed and after 18 months of focused efforts Verilog became an IEEE standard as IEEE Std 1364-1995.

After the standardization process was complete the 1364 Working Group started looking for feedback from 1364 users worldwide so the standard could be enhanced and modified accordingly. This led to a five year effort to get a much better Verilog standard in IEEE Std 1364-2001.

Objective of the IEEE Std 1364-2001 effort

The starting point for the IEEE 1364 Working Group for this standard was the feedback received from the IEEE Std 1364-1995 users worldwide. It was clear from the feedback that users wanted improvements in all aspects of the language. Users at the higher levels wanted to expand and improve the language at the RTL and behavioral levels, while users at the lower levels wanted improved capability for ASIC designs and signoff. It was for this reason that the 1364 Working Group was organized into three task forces: Behavioral, ASIC, and PLI.

* Verilog® is a registered trademark of Cadence Design Systems, Inc.

The clear directive from the users for these three task forces was to start by solving some of the following problems:

Consolidate existing IEEE Std 1364-1995

Verilog Generate statement

Multi-dimensional arrays

Enhanced Verilog file I/O

Re-entrant tasks

Standardize Verilog configurations

Enhance timing representation

Enhance the VPI routines

Achievements

Over a period of four years the 1364 Verilog Standards Group (VSG) has produced five drafts of the LRM. The three task forces went through the IEEE Std 1364-1995 LRM very thoroughly and in the process of consolidating the existing LRM have been able to provide nearly three hundred clarifications and errata for the Behavioral, ASIC, and PLI sections. In addition, the VSG has also been able to agree on all the enhancements that were requested (including the ones stated above).

Three new sections have been added. Clause 13, Configuring the contents of a design, deals with configuration management and has been added to facilitate both the sharing of Verilog designs between designers and/or design groups and the repeatability of the exact contents of a given simulation session. Clause 15, Timing checks, has been broken out of Clause 17, System tasks and functions, and details more fully how timing checks are used in specify blocks. Clause 16, Backannotation using the Standard Delay Format (SDF), addresses using back annotation (IEEE Std 1497-1999) within IEEE Std 1364-2001.

Extreme care has been taken to enhance the VPI routines to handle all the enhancements in the Behavioral and other areas of the LRM. Minimum work has been done on the PLI routines and most of the work has been concentrated on the VPI routines. Some of the enhancements in the VPI are the save and restart, simulation control, work area access, error handling, assign/deassign and support for array of instances, generate, and file I/O.

Work on this standard would not have been possible without funding from the CAS society of the IEEE and Open Verilog International.

The IEEE Std 1364-2001 Verilog Standards Group organization

Many individuals from many different organizations participated directly or indirectly in the standardization process. The main body of the IEEE Std 1364-2001 working group is located in the United States, with a subgroup in Japan (EIAJ/1364HDL).

The members of the IEEE Std 1364-2001 working group had voting privileges and all motions had to be approved by this group to be implemented. The three task forces focused on their specific areas and their recommendations were eventually voted on by the IEEE Std 1364-2001 working group.

BEHAVIOURAL LANGUAGES –

Part 4: Verilog® hardware description language

1. Overview

1.1 Objectives of this standard

The intent of this standard is to serve as a complete specification of the Verilog® Hardware Description Language (HDL). This document contains

- The formal syntax and semantics of all Verilog HDL constructs
- The formal syntax and semantics of Standard Delay Format (SDF) constructs
- Simulation system tasks and functions, such as text output display commands
- Compiler directives, such as text substitution macros and simulation time scaling
- The Programming Language Interface (PLI) binding mechanism
- The formal syntax and semantics of access routines, task/function routines, and Verilog procedural interface routines
- Informative usage examples
- Informative delay model for SDF
- Listings of header files for PLI

1.2 Conventions used in this standard

This standard is organized into clauses, each of which focuses on a specific area of the language. There are subclauses within each clause to discuss individual constructs and concepts. The discussion begins with an introduction and an optional rationale for the construct or the concept, followed by syntax and semantic descriptions, followed by some examples and notes.

The term *shall* is used through out this standard to indicate mandatory requirements, whereas the term *can* is used to indicate optional features. These terms denote different meanings to different readers of this standard:

- a) To the developers of tools that process the Verilog HDL, the term *shall* denotes a requirement that the standard imposes. The resulting implementation is required to enforce the requirements and to issue an error if the requirement is not met by the input.
- b) To the Verilog HDL model developer, the term *shall* denotes that the characteristics of the Verilog HDL are natural consequences of the language definition. The model developer is required to adhere to the constraint implied by the characteristic. The term *can* denotes optional features that the model

developer can exercise at discretion. If used, however, the model developer is required to follow the requirements set forth by the language definition.

- c) To the Verilog HDL model user, the term *shall* denotes that the characteristics of the models are natural consequences of the language definition. The model user can depend on the characteristics of the model implied by its Verilog HDL source text.

1.3 Syntactic description

The formal syntax of the Verilog HDL is described using Backus-Naur Form (BNF). The following conventions are used:

- a) Lowercase words, some containing embedded underscores, are used to denote syntactic categories. For example:

```
module_declaration
```

- b) Boldface words are used to denote reserved keywords, operators, and punctuation marks as a required part of the syntax. These words appear in a larger font for distinction. For example:

```
module => ;
```

- c) A vertical bar separates alternative items unless it appears in boldface, in which case it stands for itself. For example:

```
unary_operator ::=
    + | - | ! | ~ | & | ~& | | | ~ | ^ | ~^ | ^~
```

- d) Square brackets enclose optional items. For example:

```
input_declaration ::= input [range] list_of_variables ;
```

- e) Braces enclose a repeated item unless it appears in boldface, in which case it stands for itself. The item may appear zero or more times; the repetitions occur from left to right as with an equivalent left-recursive rule. Thus, the following two rules are equivalent:

```
list_of_param_assignments ::= param_assignment { , param_assignment }
list_of_param_assignments ::=
    param_assignment
    | list_of_param_assignment , param_assignment
```

- f) If the name of any category starts with an italicized part, it is equivalent to the category name without the italicized part. The italicized part is intended to convey some semantic information. For example, *msb_constant_expression* and *lsb_constant_expression* are equivalent to *constant_expression*.

The main text uses *italicized* font when a term is being defined, and `constant-width` font for examples, file names, and while referring to constants, especially 0, 1, x, and z values.

1.4 Contents of this standard

A synopsis of the clauses and annexes is presented as a quick reference. There are 27 clauses and 8 annexes. All clauses, as well as Annex A, Annex B, Annex E, Annex F, and Annex G, are normative parts of this standard. Annex C, Annex D, and Annex H are included for informative purposes only.

Clause 1—Overview: This clause discusses the conventions used in this standard and its contents.

Clause 2—This clause describes the lexical tokens used in Verilog HDL source text and their conventions.: This clause describes how to specify and interpret the lexical tokens.

Clause 3—Data types: This clause describes net and variable data types. This clause also discusses the parameter data type for constant values and describes drive and charge strength of the values on nets.

Clause 4—Expressions: This clause describes the operators and operands that can be used in expressions.

Clause 5—Scheduling semantics: This clause describes the scheduling semantics of the Verilog HDL.

Clause 6—Assignments: This clause compares the two main types of assignment statements in the Verilog HDL—continuous assignments and procedural assignments. It describes the continuous assignment statement that drives values onto nets.

Clause 7—Gate and switch level modeling: This clause describes the gate and switch level primitives and logic strength modeling.

Clause 8—User-defined primitives (UDPs): This clause describes how a primitive can be defined in the Verilog HDL and how these primitives are included in Verilog HDL models.

Clause 9—Behavioral modeling: This clause describes procedural assignments, procedural continuous assignments, and behavioral language statements.

Clause 10—Tasks and functions: This clause describes tasks and functions—procedures that can be called from more than one place in a behavioral model. It describes how tasks can be used like subroutines and how functions can be used to define new operators.

Clause 11—Disabling of named blocks and tasks: This clause describes how to disable the execution of a task and a block of statements that has a specified name.

Clause 12—Hierarchical structures: This clause describes how hierarchies are created in the Verilog HDL and how parameter values declared in a module can be overridden. It describes how generated instantiations can be used to do conditional or multiple instantiations in a design.

Clause 13—Configuring the contents of a design: This clause describes how to configure the contents of a design.

Clause 14—Specify blocks: This clause describes how to specify timing relationships between input and output ports of a module.

Clause 15—Timing checks: This clause describes how timing checks are used in specify blocks to determine if signals obey the timing constraints.

Clause 16—Backannotation using the Standard Delay Format (SDF): This clause describes syntax and semantics of Standard Delay Format (SDF) constructs.

Clause 17—System tasks and functions: This clause describes the system tasks and functions.

Clause 18—Value change dump (VCD) files: This clause describes the system tasks associated with Value Change Dump (VCD) file, and the format of the file.

Clause 19—Compiler directives: This clause describes the compiler directives.

Clause 20—PLI overview: This clause previews the C language procedural interface standard (Programming Language Interface or PLI) and interface mechanisms that are part of the Verilog HDL.

Clause 21—PLI TF and ACC interface mechanism

This clause describes the interface mechanism that provides a means for users to link PLI task/function (TF) routine and access (ACC) routine applications to Verilog software tools.

Clause 22—Using ACC routines: This clause describes the ACC routines in general, including how and why to use them.

Clause 23—ACC routine definitions: This clause describes the specific ACC routines, explaining their function, syntax, and usage.

Clause 24—Using TF routines: This clause provides an overview of the types of operations that are done with the TF routines.

Clause 25—TF routine definitions: This clause describes the specific TF routines, explaining their function, syntax, and usage.

Clause 26—Using VPI routines: This clause provides an overview of the types of operations that are done with the Verilog Programming Interface (VPI) routines.

Clause 27—VPI routine definitions: This clause describes the VPI routines.

Annex A—Formal syntax definition: This normative annex describes, using BNF, the syntax of the Verilog HDL.

Annex B—List of keywords: This normative annex lists the Verilog HDL keywords.

Annex C—System tasks and functions: This informative annex describes system tasks and functions that are frequently used, but that are not part of the standard.

Annex D—Compiler directives: This informative annex describes compiler directives that are frequently used, but that are not part of the standard.

Annex E—`acc_user.h`: This normative annex provides a listing of the contents of the `acc_user.h` file.

Annex F—`veriusers.h`: This normative annex provides a listing of the contents of the `veriusers.h` file.

Annex G—`vpi_user.h`: This normative annex provides a listing of the contents of the `vpi_user.h` file.

Annex H—Bibliography: This informative annex contains bibliographic entries pertaining to this standard.

1.5 Header file listings

The header file listings included in the annexes E, F, and G for `acc_user.h`, `veriusers.h`, and `vpi_user.h` are a normative part of this standard. All compliant software tools should use the same function declarations, constant definitions, and structure definitions contained in these header file listings.

1.6 Examples

Several small examples in the Verilog HDL and the C programming language are shown throughout this standard. These examples are *informative*—they are intended to illustrate the usage of Verilog HDL constructs and PLI functions in a simple context and do not define the full syntax.

1.7 Prerequisites

Clauses 20 through 27 and Annexes E through G presuppose a working knowledge of the C programming language.