

Figure 1—Simulation values of a trireg and its driver	27
Figure 2—Simulation results of a capacitive network	28
Figure 3—Simulation results of charge sharing.....	29
Figure 4—Schematic diagram of interconnections in array of instances.....	81
Figure 5—Scale of strengths	89
Figure 6—Combining unequal strengths	90
Figure 7—Combination of signals of equal strength and opposite values.....	90
Figure 8—Weak x signal strength.....	91
Figure 9—Bufifs with control inputs of x	91
Figure 10—Strong H range of values	91
Figure 11—Strong L range of values.....	92
Figure 12—Combined signals of ambiguous strength.....	92
Figure 13—Range of strengths for an unknown signal	92
Figure 14—Ambiguous strengths from switch networks	93
Figure 15—Range of two strengths of a defined value	93
Figure 16—Range of three strengths of a defined value	93
Figure 17—Unknown value with a range of strengths	94
Figure 18—Strong X range.....	94
Figure 19—Ambiguous strength from gates.....	94
Figure 20—Ambiguous strength signal from a gate.....	95
Figure 21—Weak 0.....	95
Figure 22—Ambiguous strength in combined gate signals.....	95
Figure 23—Elimination of strength levels.....	96
Figure 24—Result demonstrating a range and the elimination of strength levels of two values97	
Figure 25—Result demonstrating a range and the elimination of strength levels of one value98	
Figure 26—A range of both values.....	99

Figure 27—Wired logic with unambiguous strength signals	100
Figure 28—Wired logic and ambiguous strengths	101
Figure 29—Trireg net with capacitance	106
Figure 30—Module schematic and simulation times of initial value propagation.....	115
Figure 31—Repeat event control utilizing a clock edge.....	145
Figure 32—Hierarchy in a model	195
Figure 33—Hierarchical path names in a model	196
Figure 34—Scopes available to upward name referencing	199
Figure 35—Module path delays.....	213
Figure 36—The difference between parallel and full connection paths	221
Figure 37—Module path delays longer than distributed delays	227
Figure 38—Module path delays shorter than distributed delays	228
Figure 39—Legal and illegal module paths.....	228
Figure 40—Illegal module paths	229
Figure 41—Legal module paths.....	229
Figure 42—On-detect -vs.- on-event	232
Figure 43—Current event cancellation problem and correction.....	234
Figure 44—NAND gate with nearly simultaneous input switching where one event is scheduled prior to another that has not matured	235
Figure 45—Input NAND gate with nearly simultaneous input switching with output event scheduled at same time.	236
Figure 46—Sample \$timeskew	252
Figure 47—Sample \$fullskew	254
Figure 48—Timing check violation windows	265
Figure 49—Data constraint interval, positive setup/hold	268
Figure 50—Data constraint interval, negative setup/hold	269
Figure 51—Creating the four state VCD file.....	325
Figure 52—Creating the extended VCD file	340

Figure 53—Using <code>acc_error_flag</code> to detect errors	388
Figure 54—How ACC routines store strings in the internal buffer.....	395
Figure 55—Buffer reset causes data in the string buffer to be overwritten.....	396
Figure 56—The VCL <code>s_vc_record</code> structure	399
Figure 57—The VCL <code>s_strengths</code> structure.....	400
Figure 58—Using <code>acc_append_delays()</code> in single delay value mode.....	407
Figure 59—Using <code>acc_append_delays()</code> in min:typ:max mode	408
Figure 60—Using <code>acc_close()</code>	411
Figure 61—Using <code>acc_collect()</code>	413
Figure 62—Using <code>acc_compare_handles()</code>	414
Figure 63—Using <code>acc_configure()</code> to set <code>accDefaultAttr0</code>	420
Figure 64—Using <code>acc_configure()</code> to set <code>accEnableArgs</code>	421
Figure 65—Using <code>acc_configure()</code> to set <code>accPathDelayCount</code>	422
Figure 66—Using <code>acc_configure()</code> to set <code>accToHiZDelay</code>	423
Figure 67—Using <code>acc_count()</code>	424
Figure 68—Using <code>acc_fetch_argc()</code>	425
Figure 69—Using <code>acc_fetch_argv()</code>	427
Figure 70—Using <code>acc_fetch_attribute()</code>	431
Figure 71—Using <code>acc_fetch_defname()</code>	434
Figure 72—Using <code>acc_fetch_delay_mode()</code>	436
Figure 73—Using <code>acc_fetch_delays()</code> in single delay mode	439
Figure 74—Using <code>acc_fetch_delays()</code> in min:typ:max delay mode	440
Figure 75—Using <code>acc_fetch_direction()</code>	441
Figure 76—Using <code>acc_fetch_edge()</code>	443
Figure 77—A design hierarchy; the fullname of net w4 is “top1.mod3.w4”	444
Figure 78—Using <code>acc_fetch_fullname()</code>	445
Figure 79—Using <code>acc_fetch_fulltype()</code> to display the fulltypes of timing checks.....	447

Figure 80—Using <code>acc_fetch_fulltype()</code> to display the fulltypes of primitives	448
Figure 81—Using <code>acc_fetch_index()</code>	450
Figure 82— <code>s_location</code> data structure	451
Figure 83—Using <code>acc_fetch_location()</code>	452
Figure 84—A design hierarchy; the name of net w4 is “w4”	453
Figure 85—Using <code>acc_fetch_name()</code>	454
Figure 86—Using <code>acc_fetch_paramtype()</code>	455
Figure 87—Using <code>acc_fetch_paramval()</code>	457
Figure 88—Using <code>acc_fetch_polarity()</code>	458
Figure 89—Using <code>acc_fetch_pulse()</code>	462
Figure 90—Using <code>acc_fetch_range()</code>	463
Figure 91—Using <code>acc_fetch_size()</code>	464
Figure 92—Using <code>acc_fetch_tfarg()</code> , <code>acc_fetch_tfarg_int()</code> , and <code>acc_fetch_tfarg_str()</code>	466
Figure 93— <code>s_timescale_info</code> data structure	469
Figure 94—Using <code>acc_fetch_type()</code>	472
Figure 95—Using <code>acc_fetch_type_str()</code>	473
Figure 96— <code>s_acc_value</code> structure.....	475
Figure 97— <code>s_acc_vecval</code> structure.....	476
Figure 98—Using <code>acc_fetch_value()</code> to retrieve the logic values as strings	477
Figure 99—Using <code>acc_fetch_value()</code> to retrieve values into a data structure.....	478
Figure 100—Using <code>acc_free()</code>	479
Figure 101—Using <code>acc_handle_by_name()</code>	481
Figure 102—Using <code>acc_handle_condition()</code>	483
Figure 103—Using <code>acc_handle_conn()</code>	484
Figure 104—Using <code>acc_handle_datapath()</code>	485
Figure 105—Using <code>acc_handle_hiconn()</code> and <code>acc_handle_loconn()</code>	487
Figure 106—Using <code>acc_handle_modpath()</code>	491

Figure 107—Using <code>acc_handle_object()</code>	494
Figure 108—Using <code>acc_handle_parent()</code>	495
Figure 109—Using <code>acc_handle_path()</code>	496
Figure 110—Using <code>acc_handle_pathin()</code>	497
Figure 111—Using <code>acc_handle_pathout()</code>	498
Figure 112—Using <code>acc_handle_port()</code>	500
Figure 113—Using <code>acc_handle_scope()</code>	501
Figure 114—Using <code>acc_handle_simulated_net()</code>	503
Figure 115—Edge sums model edge-control specifiers	506
Figure 116—Using <code>acc_handle_tchk()</code>	507
Figure 117—Using <code>acc_handle_tchkarg1()</code> , <code>acc_handle_tchkarg2()</code> and <code>acc_handle_notifier()</code>	509
Figure 118—Using <code>acc_handle_terminal()</code>	511
Figure 119—Using <code>acc_handle_tfarg()</code>	513
Figure 120—Using <code>acc_initialize()</code>	515
Figure 121—Using <code>acc_next()</code>	519
Figure 122—Using <code>acc_next_bit()</code> with module ports	521
Figure 123—Using <code>acc_next_bit()</code> with a vector net	521
Figure 124—Using <code>acc_next_cell()</code>	522
Figure 125—The difference between <code>acc_next_load()</code> and <code>acc_next_cell_load()</code>	523
Figure 126—Using <code>acc_next_cell_load()</code>	524
Figure 127—Using <code>acc_next_child()</code>	525
Figure 128—Using <code>acc_next_driver()</code>	526
Figure 129—Using <code>acc_next_hiconn()</code> and <code>acc_next_loconn()</code>	528
Figure 130—Using <code>acc_next_input()</code>	530
Figure 131—The difference between <code>acc_next_load()</code> and <code>acc_next_cell_load()</code>	531
Figure 132—Using <code>acc_next_load()</code>	532
Figure 133—Using <code>acc_next_modpath()</code>	534

Figure 134—Using <code>acc_next_net()</code>	535
Figure 135—Using <code>acc_next_output()</code>	537
Figure 136—Using <code>acc_next_parameter()</code>	538
Figure 137—Using <code>acc_next_port()</code> with a module handle.....	540
Figure 138—Using <code>acc_next_port()</code> with a net handle.....	540
Figure 139—Using <code>acc_next_portout()</code>	541
Figure 140—Using <code>acc_next_primitive()</code>	542
Figure 141—Using <code>acc_next_specparam()</code>	544
Figure 142—Using <code>acc_next_tchk()</code>	546
Figure 143—Using <code>acc_next_terminal()</code>	547
Figure 144—Using <code>acc_next_topmod()</code>	548
Figure 145—Using <code>acc_object_in_typelist()</code>	550
Figure 146—Using <code>acc_object_of_type()</code>	552
Figure 147—Using <code>acc_product_type()</code>	554
Figure 148—Using <code>acc_product_version()</code>	555
Figure 149—Using <code>acc_release_object()</code>	556
Figure 150—Using <code>acc_replace_delays()</code> in single delay mode.....	559
Figure 151—Using <code>acc_replace_delays()</code> in min:typ:max delays mode	560
Figure 152—Using <code>acc_replace_pulsere()</code>	563
Figure 153—Using <code>acc_set_pulsere()</code>	567
Figure 154—Using <code>acc_set_scope()</code>	569
Figure 155—The <code>s_setval_value</code> structure used by <code>acc_set_value()</code>	571
Figure 156— <code>s_acc_vecval</code> structure.....	571
Figure 157—The <code>s_setval_delay</code> structure for <code>acc_set_value()</code>	572
Figure 158—The <code>s_acc_time</code> structure for <code>acc_set_value()</code>	573
Figure 159—Using <code>acc_set_value()</code>	574
Figure 160—Using <code>acc_version()</code>	578

Figure 161—Adding with <code>tf_add_long()</code>	587
Figure 162—Dividing with <code>tf_divide_long()</code>	593
Figure 163—The <code>s_tfexprinfo</code> structure definition.....	599
Figure 164—The <code>s_vecval</code> structure definition	600
Figure 165—Multiplying with <code>tf_multiply_long()</code>	620
Figure 166—The <code>s_tfnodeinfo</code> structure definition	622
Figure 167—The <code>s_vecval</code> structure definition	623
Figure 168—The <code>s_strengthval</code> structure definition	623
Figure 169—The <code>memval</code> structure definition	624
Figure 170—Subtracting with <code>tf_subtract_long()</code>	649
Figure 171—Using <code>tf_subtract_long()</code>	650
Figure 172—The <code>s_vpi_error_info</code> structure definition	701
Figure 173—The <code>s_cb_data</code> structure definition	707
Figure 174—The <code>s_vpi_delay</code> structure definition	710
Figure 175—The <code>s_vpi_time</code> structure definition.....	710
Figure 176—The <code>s_vpi_systf_data</code> structure definition	714
Figure 177—The <code>s_vpi_time</code> structure definition.....	715
Figure 178—The <code>s_vpi_value</code> structure definition	718
Figure 179—The <code>s_vpi_vecval</code> structure definition	718
Figure 180—The <code>s_vpi_strengthval</code> structure definition.....	718
Figure 181—The <code>s_vpi_vlog_info</code> structure definition	723
Figure 182—The <code>s_vpi_delay</code> structure definition	739
Figure 183—The <code>s_vpi_time</code> structure definition.....	739
Figure 184—The <code>s_vpi_value</code> structure definition	745
Figure 185—The <code>s_vpi_time</code> structure definition.....	745
Figure 186—The <code>s_vpi_vecval</code> structure definition	745
Figure 187—The <code>s_vpi_strengthval</code> structure definition.....	745

Figure 188—The s_cb_data structure definition 746

Figure 189—The s_vpi_systf_data structure definition 755