DISCLAIMER:

This presentation is strictly an overview — it is *NOT* the full IEEE standard, and does *NOT* reflect the full details of the enhancements to the Verilog standard!

The IEEE Verilog 1364-~~2000~~ 2001 Standard

What's New, and Why You Need It

by Stuart Sutherland

Sutherland HDL, Inc.

Verilog Training and Consulting Experts

Presented at the HDLCON-2000 Conference
March 10, 2000 San Jose, California

This presentation was updated August, 2001
to clarify some points and make minor corrections in some examples

(my thanks to Cliff Cummings of Sunburst Design for suggesting the changes)

Verilog-~~2000~~ 2001 Status

- The specification of the Verilog-~~2000~~ 2001 standard is complete
  - Voting draft completed March 1st, 2000
  - The final IEEE balloting process has started
  - Expect Verilog-2000 to be ratified in Q3-2000
  - The official standard will be IEEE Std. 1364-~~2000~~ 2001

UPDATE

The IEEE officially ratified the proposed standard in March, 2001.
The official name for the new Verilog standard is IEEE Std. 1364-2001.
The common nickname is "Verilog-2001".

The original version of this presentation was prepared in 2000.
At that time, it was anticipated that the IEEE would ratify the standard that year.
Therefore, the original version of this presentation used the nickname "Verilog-2000".

Why a New Standard?

- Add enhancements to Verilog
  - Design methodologies are evolving
    - System level design, intellectual property models, design re-use, very deep submicron, etc.
  - Cliff Cumming's "Top Five Enhancement Requests" from HDLCON-1996
- Clarify ambiguities in Verilog 1364-1995
  - The 1364-1995 reference manual came the Gateway Design Automation Verilog-XL User's Manual
  - Verilog-2001 more clearly defines Verilog syntax and semantics

Goals for Verilog-2001

- Enhance Verilog for
  - Higher level, abstract system level modeling
  - Intellectual Property (IP) modeling
  - Greater timing accuracy for very deep submicron
- Make Verilog even easier to use
- Correct errata and ambiguities
- Maintain backward compatibility — existing models will work with the new standard
- Ensure that EDA vendors will implement all enhancements!

original: 10 Mar 2000
updated: 18 Oct 2001

The IEEE 1364 Verilog Standards Committee

SUTHERLAND HDL

- A main working group
  - Final approval of all changes to 1364-1995
  - About 20 active participants
- Three task forces
  - Behavioral Task Force (Cliff Cummings, chair)
    - RTL and behavioral modeling enhancements
  - ASIC Task Force (Steve Wadsworth, chair)
    - ASIC and FPGA library modeling enhancements
  - PLI Task Force (Drew Lynch, Stu Sutherland, co-chairs)
    - PLI enhancements

Overview of HDL Enhancements

- 33 major enhancements were added to the Verilog HDL
  - Brief description and examples
  - New reserved words
- Errata and clarifications
  - Dozens of corrections were made to 1364-1995
  - Do not affect Verilog users
  - Very important to Verilog tool implementors
  - Not listed in this paper — refer to the 1364-2001 Verilog Language Reference Manual (LRM)

1: Verilog Configurations

- Verilog-1995 leaves design management up to the software tools
  - Every tool has different ways to manage large designs
- Verilog-2001 adds configuration blocks
  - All software tools will have a consistent method
  - The version for each module instance can be specified
    - Virtual libraries specified within Verilog source code
    - Physical file locations specified in a "map" file
  - New reserved words added: ***config***, ***endconfig***, ***design***, ***instance***, ***cell***, ***use***, ***liblist***

Verilog Configuration Notes

- Verilog design hierarchy is modeled the same as always
- Configurations can be used to specify which module source code should be used for each instance of a module.
  - With Verilog-1995, it is up to the simulator on how to specify which model version should be used for each instance (if the simulator can do it at all)
- The configuration block is specified outside of all modules
  - Can be in the same file as the Verilog source code
  - Can be in a separate file
  - Verilog model source code does not need to be modified in order to change the design configuration!
- A separate file maps logical library names to physical file locations
  - Verilog source code does not need to be modified when a design is moved to a different physical source location!

Verilog Configuration Example

Verilog Design

```
module test;
  ...
  myChip dut (...);
  ...
endmodule
```

```
module myChip(...);
  ...
  adder a1 (...);
  adder a2 (...);
  ...
endmodule
```

Configuration Block (part of Verilog source code)

```
/* define a name for this configuration */
config cfg4

  /* specify where to find top level modules */
  design rtlLib.test

  /* set the default search order for finding
     instantiated modules */
  default liblist rtlLib gateLib;

  /* explicitly specify which library to use
     for the following module instance */
  instance test.dut.a2 liblist gateLib;
endconfig
```

Library Map File

```
/* location of RTL models (current directory) */
library rtlLib ./*.v;

/* Location of synthesized models */
library gateLib ./synth_out/*.v;
```

2: Verilog Generate

- Verilog-2001 adds true generate capability
  - Use ***for*** loops to generate any number of instances of:
    - Modules, primitives, procedures, continuous assignments, tasks, functions, variables, nets
  - Use ***if–else*** and ***case*** decisions to control what instances are generated
    - Provides greater control than the VHDL generate
  - New reserved words added:
    - ***generate***, ***endgenerate***, ***genvar***, ***localparam***

Verilog Generate Example

```
module multiplier (a, b, product);
  parameter a_width = 8, b_width = 8;
  localparam product_width = a_width + b_width;
  input    [a_width-1:0]       a;
  input    [b_width-1:0]       b;
  output   [product_width-1:0] product;

  generate
    if ((a_width < 8) || (b_width < 8))
      CLA_multiplier #(a_width, b_width) u1 (a, b, product);
    else
      WALLACE_multiplier #(a_width, b_width) u1 (a, b, product);
  endgenerate

endmodule
```

localparams are constants, which cannot be redefined

- **If the input bus widths are 8-bits or less, generate and instance of a carry-look-ahead multiplier**
- **If the input bus widths are greater than 8-bits, generate an instance of a wallace-tree multiplier**

3: Constant Functions

- Verilog-2001 adds constant functions
  - Same syntax as standard Verilog functions
  - Limited to statements that can be evaluated at compile time
  - Can be called anywhere a constant expression is required
    - Vector width declarations
    - Array declarations
    - Replicate operations
- Provides for more scalable, re-usable models

Constant Functions Example

```
module ram (...);
  parameter RAM_SIZE = 1024;
  parameter ADDRESS = 12;
  input [ADDRESS-1:0] address_bus;
```

Verilog 1995:

Vector widths can be calculated using simple constant expressions

```
module ram (...);
  parameter RAM_SIZE = 1024;
  input [clogb2(RAM_SIZE)-1:0] address_bus;
  ...
  function integer clogb2;
    input [31:0] depth;
    begin
      for(clogb2=0; depth>0; clogb2=clogb2+1)
        depth = depth >> 1;
    end
  endfunction
  ...
```

Verilog 2001:

Vector widths can be calculated using complex constant functions

4: Indexed Vector Part Selects

- Verilog-2001 adds the capability to use variables to select a group of bits from a vector
  - The starting point of the part-select can vary
  - The width of the part-select remains constant

```
reg [63:0] word;
reg  [3:0] byte_num; //a value from 0 to 7
wire [7:0] byteN = word[byte_num*8 +: 8];
```

The starting point of the part-select is variable

The width of the part-select is constant

+: indicates the part-select increases from the starting point
-: indicates the part-select decreases from the starting point

5: Multi-dimensional Arrays

- Verilog-1995 allows 1-dimensional arrays of reg, integer and time variables
  - Typically used to model RAM and ROM memories
- Verilog-2001 adds:
  - Multidimensional arrays of any variable data type
  - Multidimensional arrays of any net data type

```
//declare a 3-dimensional array of 8-bit wire nets
wire [7:0] array3 [0:255][0:255][0:15];

//select one word out of a 3-dimensional array
wire [7:0] out3 = array3[addr1][addr2][addr3];
```

6: Array Bit and Part Selects

- Verilog-2001 adds:
  - Bit-selects out of an array
  - Part-selects out of an array

```
//select the high-order byte of one word in a
//2-dimensional array of 32-bit reg variables
reg [31:0] array2 [0:255][0:15];
wire [7:0] out2 = array2[100][7][31:24];
```

7: Signed Arithmetic Extensions

- Verilog-2001 adds:
  - reg and net data types can be declared as signed

```
reg  signed [63:0] data;
wire signed [11:0] address;
```

  - Function returns can be declared as signed

```
function signed [128:0] alu;
```

  - Literal integer numbers can be declared as signed

```
16'shC501  //a signed 16-bit hex value
```

  - New arithmetic shift operators, **<<<** and **>>>**, maintain the sign of a value
  - New **$signed()** and **$unsigned()** system functions can "cast" a value to signed or unsigned

8: Power Operator

- Verilog-2001 add an exponential power operator
  - Represented by the ** token
  - Similar to the C pow() function
  - If either operand is real, a real value is returned
  - If both operands are integers, an integer value is returned

```
module ram (...);
  parameter WORD_SIZE = 64;
  parameter ADDR_SIZE = 24;

  reg [WORD_SIZE-1:0] core [0:(2**ADDR_SIZE)-1];
  ...
```

9: Re-entrant Tasks and Recursive Functions

SUTHERLAND HDL

- Verilog-2001 adds automatic tasks and functions
  - Each call to the task/function allocates unique storage
    - In Verilog-1995, tasks and functions are static; each call shares the same storage space
  - Concurrent task calls will not interfere with each other
  - Recursive calls to a function are stacked
  - New reserved word added: ***automatic***

```
function automatic [63:0] factorial;
  input [31:0] n;
  if (n == 1)
    factorial = 1;
  else
    factorial = n * factorial(n-1);
endfunction
```

Recursive function call

10: Comma-separated Sensitivity List

- Verilog-2001 adds a second syntax style for listing signals in a sensitivity list
  - Signals in the list can be separated with a comma
    - The old "or" separated list will still work

Verilog-1995

```
always @(sel or a or b or c or d)
  case (sel)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
    2'b11: y = d;
  endcase
```

Verilog-2001

```
always @(sel, a, b, c, d)
  case (sel)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
    2'b11: y = d;
  endcase
```

11: Combinational Logic Sensitivity

- Verilog-2001 adds a "wildcard" token to indicate a combinational logic sensitivity list
  - The @* token indicates automatic sensitivity to any change on any signal that is read by the following statement or statement group

Verilog-1995

```
always @(sel or a or b or c or d)
  case (sel)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
    2'b11: y = d;
  endcase
```

Verilog-2001

```
always @*
  case (sel)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
    2'b11: y = d;
  endcase
```

12: Enhanced File I/O

- Verilog-1995 has limited built-in file I/O tasks
  - Up to 31 files can be opened for writing
    - Only ASCII characters can be written to files
  - More complex file I/O is done using the Verilog Programming Language Interface (PLI)
- Verilog-2001 adds:
  - The ability to open up to 2^{30} files
  - New file I/O tasks: **$ferror**, **$fgetc**, **$fgets**, **$fflush**, **$fread**, **$fscanf**, **$fseek**, **$ftel**, **$rewind**, **$ungetc**
  - New string tasks: **$sformat**, **$swrite**, **$swriteb**, **$swriteh**, **$swriteo**

13: Automatic Width Extension Past 32 bits

- In Verilog-1995:
  - Verilog assignments zero fill when the left-hand side is wider than the right-hand side
  - Unsized integers default to 32-bits wide; therefore, the widths of integers must be hard-coded

Verilog-1995

```
parameter WIDTH = 64;
reg [WIDTH-1:0] data;
data = 'bz;    //fills with 'h00000000zzzzzzzz
data = 64'bz;  //fills with 'hzzzzzzzzzzzzzzzz
```

- Verilog-2001 will automatically extend a logic Z or X to the full width of the left-hand side

Verilog-2001

```
parameter WIDTH = 64;
reg [WIDTH-1:0] data;
data = 'bz;    //fills with 'hzzzzzzzzzzzzzzzz
```

14: Default Nets with Continuous Assigns

- Verilog-2001 will default to a net data type on the left-hand side of any continuous assignment
  - The net will be scalar (1-bit), if not connected to a port of the module
  - In Verilog-1995, the left-hand side must be explicitly declared, if not connected to a port of the module

Verilog-1995

```
module mult32 (y, a, b);
  output [63:0] y;
  input  [31:0] a, b;
  assign y = a * b;      //defaults to wire, width of port y
  assign eq = (a == b); //ERROR: 'eq' not declared
endmodule
```

Verilog-2001

```
  assign eq = (a == b); //defaults to 1-bit wire
```

15: Disable Default Net Declarations

- In Verilog-1995, undeclared signals can default to a wire data type
  - The default data type can be changed to another net data type using `default_nettype <data_type>
- Verilog-2001 provides a means to disable default net declarations
  - `default_nettype **none**
  - Any undeclared signals will be a syntax error
    - Prevents hard-to-debug wiring errors due to a mistyped name
  - none is not a new reserved word

16: Explicit In-line Parameter Passing

- Verilog-2001 adds the ability to explicitly name parameters when passing parameter values
  - Provides better self-documenting code
  - Parameter values can be passed in any order

```
module ram (...);
  parameter WIDTH = 8;
  parameter SIZE = 256;
  ...
endmodule
```

Verilog-1995

```
module my_chip (...);
  ...
  RAM #(8,1023) ram2 (...);
endmodule
```

Verilog-2001

```
module my_chip (...);
  ...
  RAM #(.SIZE(1023)) ram2 (...);
endmodule
```

17: Combined Port/Data Type Declarations

- Verilog-2001 permits combining port declarations and data type declarations into one statement

Verilog-1995

```
module mux8 (y, a, b, en);
  output [7:0] y;
  input  [7:0] a, b;
  input        en;

  reg  [7:0] y;
  wire [7:0] a, b;
  wire       en;
  ...
```

Verilog-2001

```
module mux8 (y, a, b, en);
  output reg  [7:0] y;
  input  wire [7:0] a, b;
  input  wire       en;
  ...
```

18: ANSI-style Port Lists

- Verilog-2001 adds ANSI C style input and output declarations
  - For modules, tasks and functions

Verilog-1995

```
module mux8 (y, a, b, en);
  output [7:0] y;
  input  [7:0] a, b;
  input        en;

  reg  [7:0] y;
  wire [7:0] a, b;
  wire       en;
  ...
```

Verilog-2001

```
module mux8 (output reg  [7:0] y,
             input  wire [7:0] a,
             input  wire [7:0] b,
             input  wire       en);
  ...
```

19: Reg Declaration With Initialization

- Verilog-2001 permits initializing variables at the time they are declared
  - The initialization is executed in time-step zero, just like initial procedures

Verilog-1995

```
reg clock;

initial
  clk = 0;
```

Verilog-2001

```
reg clock = 0;
```

20: "Register" Changed To "Variable"

- The Verilog-2001 standard changes the term "register" to "variable"
  - "register" is not a reserved word; it is just a term
  - Since its inception in 1984, Verilog manuals have used the term "register" to describe a class of data types
    - reg (unsigned variable), integer (signed variable), real (double precision variable), etc.
  - The term "register" often confuses new Verilog users
    - register is a hardware term for storage elements
    - Verilog registers do not imply a hardware register

21: Enhanced Conditional Compilation

- Verilog-1995 supports limited conditional compilation
  - `ifdef, `else, `endif and `undef compiler directives
- Verilog-2001 adds more extensive conditional compilation control
  - New directives: **`ifndef** and **`elsif**

22: File and Line Compiler Directive

- Verilog-2001 adds a **`line** file and line compiler directive
  - Documents the original location of Verilog source code
    - Verilog tools often include file name and line number information in error and warning messages
    - If a pre-process utility program modifies the Verilog source code, the original file and line information could be lost
    - The preprocessor can add a `line directive to the modified code to preserve the original source file location

23: Attributes

- Verilog-2001 adds "attribute" properties
  - A standard means to specify non-Verilog tool specific information to Verilog models
  - Adds new tokens **(*** and ***)**
  - Eliminates need to hide commands in comments
  - The standard does not define any specific attributes
    - Software vendors can define proprietary attributes
    - Other standards might define standard attributes

Verilog-1995

```
case (1'b1) /* synopsys parallel_case */ //1-hot FSM
```

Verilog-2001

```
(* rtl_synthesis, parallel_case *) case (1'b1) //1-hot FSM
```

24: Standard Random Number Generator

- Verilog-2001 defines the C source code for the generator used by $random
  - All simulators can generate the same random number sequence when given the same seed value
    - Simulation results from different simulators can be compared
    - New products do not need to re-invent the wheel
  - Uses the random number generator from Verilog-XL

25: Enhanced Invocation Option Tests

- Verilog-1995 contains a true/false test to see if simulation was invoked with a specific option
  - $test$plusargs
- Verilog-2001 adds the ability to read arguments of invocation options
  - New system function: **$value$plusargs**

26: Enhanced PLA Modeling

- Verilog-2001 extends the capability of the PLA system tasks ($async$or$array, $async$and$array, etc.)
  - In Verilog-1995, arguments had to be scalar
  - In Verilog-2001, arguments can be vectors

27: Accurate BNF, with Subsections

- The Verilog-1995 BNF (Backus-Naur Form) had errors and inconsistencies
- Verilog-2001 contains a much stronger BNF definition of the Verilog language
  - Consistent terminology
  - More definitive terms
  - Divided into sub-sections to make it easier to find specific definitions
  - Checked for accuracy

28: On-detect Pulse Error Propagation

- Verilog-1995 has on-event pulse error propagation
  - A pulse is a glitch on the inputs of a module path that is less than the delay of the path
  - An input pulse propagates to a path output as an X, with the same delay as if a valid input change had propagated to the output
- Verilog-2001 adds on-detect pulse error propagation
  - As soon as an input pulse is detected, a logic X is propagated to a path output, without the path delay
  - New reserved words added:
    - **pulsestyle_onevent**, **pulsestyle_ondetect**

29: Negative Pulse Detection

- Verilog-2001 adds negative pulse detections
  - Due to different rising-transition and falling-transition delays, it is possible for the trailing edge of a glitch to propagate before the leading edge has propagated
    - In Verilog-1995, a negative pulse is cancelled
  - Negative pulse detection will propagate a logic X for the duration of the negative pulse
  - New reserved words added:
    - **showcancelled**, **noshowcancelled**

30: New Timing Constraint Checks

- Verilog-2001 adds new timing constraint checks
  - More accurately model very deep submicron input constraints:
  - New timing constraint tasks added:
    - **$removal**
    - **$recrem**
    - **$timeskew**
    - **$fullskew**
  - Refer to the proposed IEEE 1364-2001 Verilog standard for details on these tasks

31: Negative Timing Constraints

- Verilog-2001 adds the ability to specify negative values for:
  - $setuphold setup and hold times
    - Adds new, optional arguments to the Verilog-1995 $setuphold task
  - $recrem recovery and removal times
    - A new timing check task in Verilog-2001

32: Enhanced SDF support

SUTHERLAND HDL

- The Verilog-2001 standard defines:
  - How timing objects in SDF map to objects in Verilog
  - Based on the latest SDF standard, IEEE 1497-1999
- Verilog-2001 changes the syntax of the **specparam** constant
  - Can now be declared at the module level as well as within a specify block (to support SDF labels)
- Verilog-2001 adds a standard **$sdf_annotate** system task
  - Already a de-facto standard in all simulators

33: Extended VCD Files

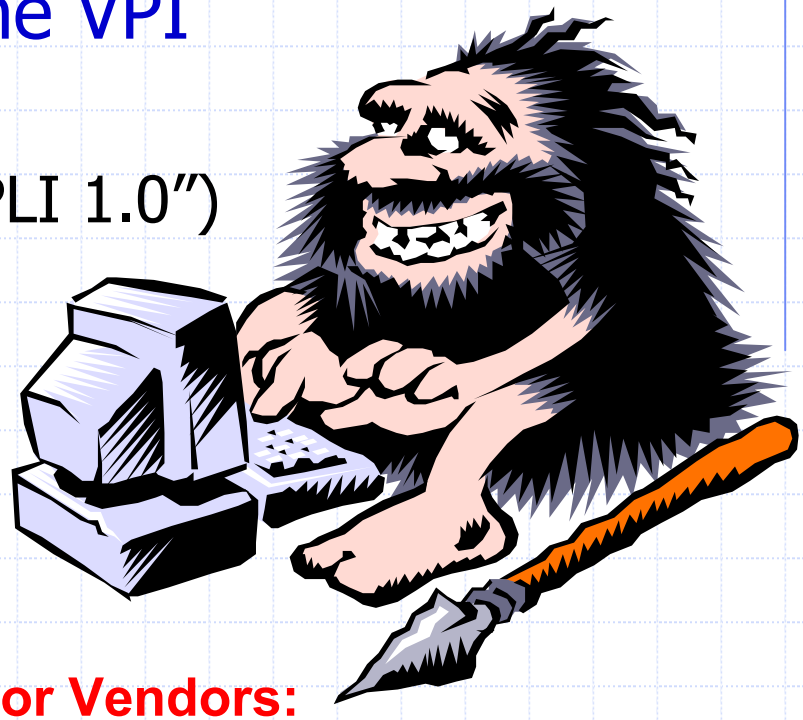
- Verilog-2001 adds new Value Change Dump (VCD) capabilities
  - Dump port change values
  - Dump strength level changes
  - Dump the time at which simulation finishes
  - New system tasks added:
    - **$dumpports**, **$dumpportsall**, **$dumpportsoff**, **$dumpportson**, **$dumpportslimit** and **$dumpportsflush**

PLI Enhancements

- Several enhancements added to the VPI library
  - Simulation control
    - Stop, finish, save, restart, etc.
  - Support for new Verilog-2001 HDL constructs
    - Array of instances, attributes, signed arithmetic, recursive functions, enhanced file I/O, etc.
- Maintenance updates to TF and ACC libraries
  - Corrected errata
  - Clarified ambiguities

The VPI Library Is The Future!

- All enhancements to the Verilog language will only be supported in the VPI library of the PLI
  - The TF and ACC libraries ("PLI 1.0") are only being maintained

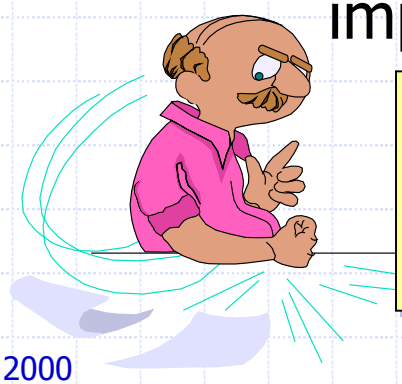


Warning To Simulator Vendors:
"PLI 1.0" is OVI's 1990 Verilog PLI standard; It isn't 1990 anymore!
Your customers do not want prehistoric simulators!

When Will These Enhancements Be Available?

SUTHERLAND HDL

- The "official" word from several EDA vendors is:
  - They will not comment on future product plans
  - They will not begin to implement Verilog-2001 until it is ratified
- The "unofficial" word from EDA vendors is:
  - Some have already started implementing Verilog-2001
  - One essentially says they do not see any need to implement the new features in Verilog-2001



UPDATE

Many of the Verilog-2001 features are now in shipping products, and *every* vendor has plans to support Verilog-2001

Summary

- Verilog-2001 is complete
  - The proposed IEEE 1364-2000 Verilog standard is now in the final balloting phase
- Verilog-2001 contains
  - Over 30 major enhancements
  - Many clarifications and errata corrections
- Verilog-2001 adds powerful capabilities
  - Greater deep submicron accuracy
  - More abstract system level modeling
  - Scalable, re-usable modeling
- Final approval is expected in late 2000

UPDATE: The definition of the new Verilog standard was completed in 2000, but the IEEE did not finishing ratifying the standard until March, 2001.

For More Information

- The book "Verilog-2001: A Guide to the New Features of the Verilog HDL" covers the enhancements in Verilog-2001 in more detail
  - Author: Stuart Sutherland
  - Publisher: Kluwer Academic Publishers, *www.wkap.com*
  - ISBN: 07923-7568-8
  - Price: $72.00 US (suggested retail price)
  - 152 pages

Special Offer: Sutherland HDL, Inc. is selling this book for $60.00 US (plus S&H). call 1+503-692-0898 to order

About The Author

- Stuart Sutherland
  - President of Sutherland HDL, Inc., Portland, Oregon
  - Provides expert Verilog design consulting and training
  - More than 15 years design experience, and over 12 years working with Verilog
  - Author of "Verilog-2001: A Guide to the New Features of the Verilog HDL", "The Verilog HDL Quick Reference Guide", "The Verilog PLI Quick Reference Guide" and "The Verilog PLI Handbook"
  - Member of the IEEE 1364 Verilog standards committee since 1993; co-chair of the PLI task force

UPDATE: Verilog-2001 versions of the Verilog HDL and PLI Quick Reference Guides are now available at www.sutherland-hdl.com.