Verilog Test Suites

Highlights

- The test suite covers SystemVerilog 3.1 (approved by Accellera, 2003), Verilog-AMS 2.0 (approved by Open Verilog International, 2000), IEEE-1364 and Verilog-2001.
- All test cases are arranged according to the section number as provided in the LRM. Also each test case clearly states the feature that is being tested.
- All related test cases are grouped together under the same category.
- Inside a group every test case is stored under a separate directory.
- Each synthesizable test case has a bench file. Designs which are not synthesizable are self testing.
- A large set of miscellaneous designs combines different SystemVerilog features and also features of Verilog-95/Verilog-2001.
- Includes a rich set of negative test cases. All negative test cases have a 'neg_' prefix.

Verific Design Automation's Verilog Test Suites cover syntax and semantics of Verilog 2001, Verilog-AMS, and SystemVerilog. Other than conventional LRM tests, Verific's tests concentrate on the synthesizable subset of Verilog, thus providing superior coverage for EDA products.

Our test suites are industry proven and in use with several leading EDA companies. We currently have the following suites available:

	# of tests	# synthesizable
VRLG 2001	1000	940
VRLG AMS	130	na
System VRLG	2000	1200

As the leading provider of formal verification solutions, we maintain extensive Verilog test suites to ensure our software quality. We were impressed with the breadth of Verific's Verilog 2001 test suites and made them an integral part of our regression tests.

-- Andy Lin, VP of Engineering, Verplex Systems, Inc.

SYSTEM VERILOG

Verific's SystemVerilog 3.1 test suite comprises 2000 different tests (1200 synthesizable) that cover the following features:

- Unsized literal using apostrophe (').
- Specifying time units like ps, ns etc to time values.
- Special string characters like \v (vertical tab), \f (form feed), \a (bell) etc.
- Array initialization using concat or multiconcat.
- Structure and union initialization using concat and multiconcat.
- New data types viz. char, int, shortint, longint, byte, bit, logic which is integer type and shortreal which is equivalent to C float.
- User defined data type declaration using typedef.
- New constructs like structure, union and enum.
- Complex data type declaration using structure, union and enum.
- Packed structure and unions.
- Packed and unpacked arrays and their use.
- Array querying system functions like $left, $right etc.
- Casting using the cast operator (`).
- Type conversion using PLI functions like $itor, $rtoi, $bitstoreal, $realtobits etc.
- Constant declaration using the keyword const.
- Static and automatic variable declaration.
- Assignment operators and special bitwise assignment operators, such as +=, -+, *=, /=, %=, &=, |=, ^=, <<=, >>=, <<<=, >>>=, and incrementor decrementor operators ++ and --.
- Selection statements using keywords priority and unique.
- New loop statement using the do-while construct.
- New jump statements like break, continue or return.
- Named block with matching name at the end of the block.
- Declaration in unnamed block.
- New event control constructs using iff and new keyword like changed.
- New process constructs using always_comb, always_latch, always_ff. Process keyword is used for dynamic processes.
- Task and function with input, output and inout ports.
- Function and task returning control using the keyword return.
- Automatic task or functions with static variables.
- Procedural assertion.
- System function like $asserton, $assertoff etc with assertion.
- Use of constructs like declaration, instantiation and procedural statements in the $root scope.
- Nested module declaration.
- Module instantiation using (.name) connection, using (.*) connection.
- Validation of connection in terms of size and data type.
- Interfaces and modport definition.
- Interface instantiation, generic interface and import export of subroutines.
- Parameter declaration using the keyword type and instantiating modules with type parameters.

VERILOG-AMS

Verific's Verilog-AMS test suite comprises 130 different tests that cover the following features:

- Analog block
- Analog sequential block
- Analog branch contribution
- Analog indirect branch assignment
- Analog conditional statement
- Analog for statement
- Analog case statement
- While, repeat and for statement must not include analog operators.
- Analog event controlled statement
- Initial step event - Occurs only within analog block
- Final step event - Occurs only within analog block
- Cross event
- Above event
- Timer event
- Procedural assignment in analog block
- Generate statement
- Operators in analog block - The operands of the operators can be integer or real.
- Standard mathematical functions, trigonometric and hyperbolic functions.
- Time derivative function ddt and time integral functions idt and idtmod
- Branch declaration
- Analog function declaration
- Real number - Unit like 'T', 'G' etc. can be used in representing real constant.
- Optional range specification to designate permissible values of a parameter in parameter declaration.
- Dynamic parameter declaration
- Nature declaration
- Discipline declaration
- Net-discipline declaration
- Use ground declaration to specify an already declared net of continuous discipline.
- wreal net declaration
- Simulator functions like discontinuity, bound_step, last_crossing, abstime, realtime, temperature, vt.
- Access functions - The access function names are defined by the access attributes specified for the discipline's natures.
- Port Access operator (<>)
- Hierarchical referencing operator can be used to access the attributes for a node or branch
- Functions to examine driver properties like driver_count, driver_state, driver_strength, driver_update
- Analysis dependent functions like analysis, ac_stim, white_noise, flicker_noise, noise_table
- Analog operators like idt, ddt etc.
- Connect module
- Connect specification - direction overridden and discipline overridden
- Compiler directive like `default_transition, `default_discipline etc.

Verific®
Design Automation

VERILOG-2001

Verific's Verilog-2001 test suite comprises 1000 different tests (940 synthesizable) that cover the following features:

- Configuration and liblist.
- Generate (if, for, case).
- Use of genvar. Positive as well as negative examples.
- localparam declaration and examples to check that they cannot be overridden.
- Use of new compiler directives `ifndef, `elsif, `undef, `line.
- Examples on attributes. These attributes are just to test whether the parser allows them.
- New operators >>>, <<<. **.
- Constant functions and expressions.
- Multidimensional arrays of reg, integer, wire etc.
- signed object declaration and use of signed constants, objects.
- Indexed vector part select both in LHS and RHS.
- Examples to check whether sign bit is extended when the value is 'x' or 'z'.
- Use of $signed and $unsigned system function calls.
- Automatic function and task calls.
- Different kinds of sensitivity lists (comma separated, wildcard).
- Examples to check default vector net declaration and disable default net declaration (`default_nettype none).
- Explicit parameter overriding (named association).
- Combined port data type declaration.
- reg initialization with declaration.
- ANSI style module parameter and port declaration list.
- Function and task declaration in ANSI style.
- Enhanced file I/O, including build in file I/O task like $fgetc, $fscanf.
- Distribution PLI task functions (dist_normal, dist_poisson etc).
- Enhanced invocation option tests ($value$plusargs).

About Verific Design Automation

Verific Design Automation, with offices in Calcutta, India, and Alameda, CA, was founded in 1998 by EDA industry veteran Rob Dekker. Prior to founding Verific, Dekker was a software developer, manager, and director at Exemplar Logic. A leading provider of VRLG and VHDL front-ends, Verific's software is used worldwide in synthesis, formal verification, emulation, debugging, virtual prototyping, and design-for-test applications, which combined have shipped over 20,000 copies.

Verific®
Design Automation

Jan 2004