

Comprehensive Test Suite for System Verilog Compliance

Key Advantages

- Backed by Interra's field-proven expertise in developing HDL-based test suites for VHDL and Verilog
- Developed in partnership with significant EDA majors
- Conforming to accepted definition and interpretation of the language
- Providing an unbiased quality analysis of EDA tools
- Complete coverage of constructs and styles
- Comprehensive validation of syntax, synthesizability, and simulation semantics

Highlights

- Over 4,000 test cases along with test benches
 - Design Constructs: 2,669 test cases
 - Test bench features: 867 test cases
 - Assertions: 814 test cases
 - DPI: 127 test cases
- Golden output for comparison
- Detailed test plans with cross-reference to test cases
- Well organized test cases highlighting testing objectives
- Both positive and negative test cases

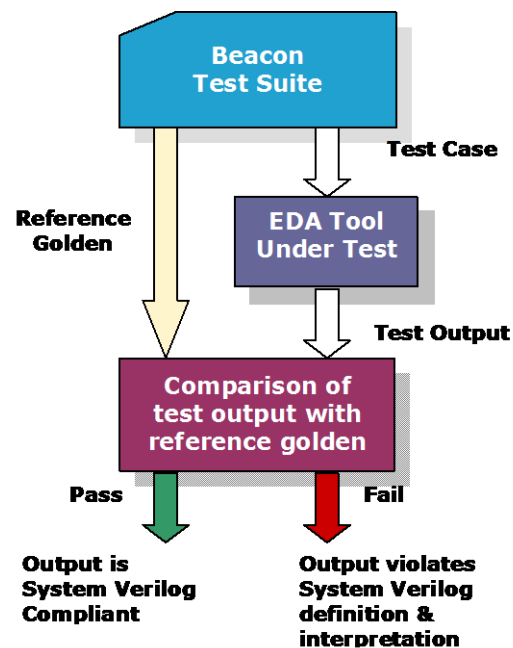
Addressing the needs of EDA tool developers to quickly evaluate the quality of their products, Interra's Beacon-SV delivers a comprehensive test suite based on System Verilog IEEE 1800-2005 standards.

You can use Beacon-SV to characterize EDA tools for coverage and quality across various language constructs and styles.

Enabling you to discover language and RTL non-compliance early in the product development and testing life cycle, Beacon-SV offers:

- Reduced development costs and time-to-market EDA products
- Development of standard-based products
- Precise evaluation of bugs and errors in the product
- Measure of product quality
- Unbiased feedback on product quality
- Regression tests for quality assurance

The test cases and test benches can be applied to the EDA tool under evaluation and results can be compared with golden reference that is provided with Beacon-SV.



The Beacon-SV Features

Comprehensive Test Suite

Well organized test cases that enable you to evaluate syntax, semantics, synthesizability, and simulation aspects of a System Verilog based tool.

Syntax and Semantics Cover

Includes over 4,000 test cases that cover constructs and styles.

Language Construct	Number of Test Cases
Design Constructs	
Literal Values	84
Data Types	216
Arrays	328
Data Declarations	169
Attributes	40
Operators and Expressions	146
Procedural Statements and Control Flow	167
Processes	97
Hierarchy	173
Interfaces	248
Parameters	78
Configurations	53
System Tasks and Functions	61
VCD Data	11
Compiler Directives	49
Tasks and Functions	121
Combined Constructs	628
Test Bench Features	
Classes	213
Random Constraints	298
Inter-Process Communication	96
Cover Group	47
Clocking Domains	107
Program Block	107
Assertions	
Sequence (CUnit, Clocking Block)	115
Properties (CUnit, Clocking Block)	64
System Functions	74
Concurrent Assertion	402
Binding	38
Assertion Action Block	10
Expect	12
Program Assertion Block	43
Immediate Assertion	56
DPI	127
Total	4778

Synthesis Subset Cover

Over 650 test cases cover supported and unsupported styles and constructs for synthesis under a combination of constructs category. More than 250 cases contain validated golden netlists.

Simulation Cover

Positive test cases, with test benches and expected output, validate simulator or equivalent tools.

Well Documented Test Plans

The test plans describe all test objectives and are categorized by sections. Each test case has a reference to the section number of the test plan.

Test Benches and Reference Golden

Provides test benches to instantiate test cases and apply vectors on inputs. Outputs are captured after an appropriate interval and written on to a file. You can easily apply the test bench to the test tool and compare the outputs.

Sample Test Case

```
--** Purpose:      Data declaration of constant type
                   on basic types: param initialization
--** LRM :        Sections 6.3.1
--** TestPlan:    Sections 4.2.7.1
--** Kind :       Simulation
--** Status:      SIMULATION_SHOULD_PASS

*****
`timescale 1ns/1ns

module const_data1(in,out1,out2,out3,out4);
parameter integer signed ig1 = -3;
input [ig1:0] in;
output reg out1,out2,out3,out4;
parameter shortint signed si1 = ig1;
parameter int signed i1 = ig1;
parameter longint signed li1 = ig1;
parameter byte b1 = "HELLO";
parameter time t1 = 5;
parameter bit bt1 = 0;
parameter [ig1:0] p = b1; // default logic
always @ (in)
begin
    out1 = (p[0] == bt1) ? 1'b0 : 1'b1;
    out2 = (p[li1] == in[si1]) ? 1 : (in[si1] <<
p[-1:0]) ? 1'b1 : in[li1];
    out3 = (t1 == 3'b101) ? b1[0] : b1[1];
    out4 = (1'b1) ? 1 : (out3 == 1) ? 1 : 0;
end
endmodule
```