

## Comprehensive Test Suite for Verilog-RTL Compliance

### Key Advantages

- Field-proven test suite that has been used to mature several industry-standard EDA tools
- Developed in partnership with significant EDA majors
- Conforming to accepted definition and interpretation of RTL semantics
- Providing an unbiased quality analysis of EDA tools
- Complete coverage of RTL subsets and styles
- Comprehensive validation of a synthesis-based test tool

### Highlights

- Over 1,500 test cases along with test benches
- Simulation and netlist golden for comparison
- Detailed test plans with cross-reference to test cases
- Well organized test cases highlighting testing objectives
- Both positive and negative test cases

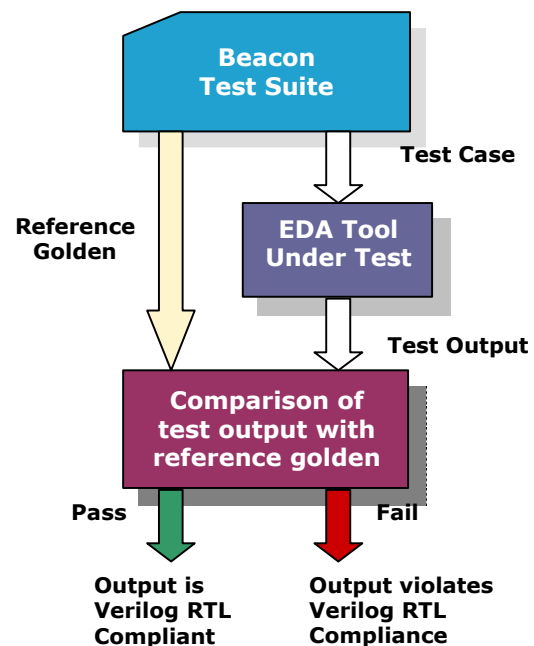
Addressing the needs of EDA tool developers to quickly evaluate the quality of their products, Interra's Beacon-Verilog-RTL delivers a comprehensive test suite to check RTL compliance in a Verilog-based EDA product.

Based on Verilog (IEEE-1364-1995, 2001) standards, the test suite lets you characterize EDA tools for compliance and quality across various synthesis styles, RTL semantics, and supported/unsupported RTL subsets.

Enabling you to discover RTL non-compliance early in the product development and testing life cycle, Beacon-RTL-Verilog 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 reference golden.



# The Beacon-RTL-Verilog

## Features

### Comprehensive Test Suite

Beacon-RTL-Verilog is a comprehensive test suite covering syntax and semantics of Verilog-based RTL synthesis. The test cases include several styles including complicated test cases that combine different synthesizable constructs with post-synthesis aspects, such as parallel output instances.

### Well-Categorized Test Suite

The test cases are well organized based on synthesizable constructs and styles.

#### Synthesizable Construct-based Test Cases

Construct-based test cases target all synthesizable Verilog constructs of varying usage. These test cases test the language coverage of an EDA tool.

#### Style-based Test Cases

Style-based test cases verify if an EDA tool is compliant with the various RTL modeling styles. These test cases include standard styles of RTL modeling, such as implicit style state machine, explicit style state machine, synchronous and asynchronous reset modeling.

#### Negative and Ignored Test Cases

Negative test cases include styles that do not comply with the RTL subset. Ideally, the test tool should detect such styles as errors.

Ignored test cases include styles, which can be ignored by the tool, but which should generate an appropriate warning.

### Test Benches for all Tests

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. You can also apply the test bench to reference golden tool and compare the outputs.

### 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.

RTL subset/Style	Number of Test Cases
Data type	26
Expression	278
Assignments	80
Statements	251
Tri State	28
Module	61
Gates	139
Tasks and Functions	53
Compiler Directives	5
Ignorable Construct	12
Not Supported Constructs	49
Not RTL	61
Data Storage Element	53
Combinational Machine	26
Sequential Machine	9
State Machine	30
Miscellaneous	244
Synthesis Directives	19
Verilog-2001	131
<b>Total</b>	<b>1,555</b>

### Sample Test Case

```
// Module having gate instance &  
// "AND" gate having bit select  
// Section 7.3.1 and Section 4.1.7 of test plan
```

```
module GATE1(in1,in2,out1);  
  input [2:0] in1,in2;  
  output [2:0] out1;  
    and and1(out1[0],in1[0],in2[0]);  
    and and2(out1[1],in1[1],in2[1]);  
    and and3(out1[2],in1[2],in2[2]);  
endmodule
```