verification

using systemverilog

Oct 22 2025

Pratyay Gopal

1 About Verification

General

systemverilog has some features geared toward verification.

Features:

- Constrained-random stimulus generation
- Functional coverage
- Higher-level structures, especially Object Oriented Programming
- Multi-threading and interprocess communication
- Support for HDL types such as Verilog's 4-state values
- Tight integration with event-simulator for control of the design

Goals

What is the goal of verification?

Goals (ii)

If you answered, “Finding bugs,” you are only partly correct. Your purpose as a verification engineer is to make sure the device can accomplish that task successfully that is, the design is an accurate representation of the specification. Bugs are what you get when there is a discrepancy. The behavior of the device when used outside of its original purpose is not your responsibility although you want to know where those boundaries lie.

Basic Functionality

- Generate stimulus
- Apply stimulus to the DUT
- Capture the response
- Check for correctness
- Measure progress against the overall verification goals

Some steps are accomplished automatically by the testbench, while others are manually determined by you. The methodology you choose determines how the above steps are carried out

Verif Flow

- Understand specification and features.
- Develop testbench (SystemVerilog/UVM).
- Write directed and random test cases.
- Add scoreboards, monitors, checkers.
- Collect coverage (code + functional).
- Debug and iterate.

Non-Synthesizeable syntax

Timing Controls

```
#10, #(delay) //explicit time delays
@(posedge clk), @(signal) //outside synthesizable always blocks
wait(condition) // waits for an event or signal level (simulation only)
```

Initial and Final Blocks

```
initial begin ... end //runs once at simulation start
final begin ... end //runs at simulation end
// Used for testbench setup, not for hardware logic
```

Non-Synthesizeable syntax (ii)

Simulation System Tasks

```
$display, $monitor, $strobe //print outputs during simulation
$dumpfile, $dumpvars //generate waveform files
$finish, $stop //end or pause simulation
```

Force and Release

```
force signal = value; // override DUT signal in simulation
release signal; // return control to DUT logic
Useful for debugging, not synthesizable
```

Non-Synthesizeable syntax (iii)

System Tasks for File I/O

```
$readmemh, $readmemb // load files into memory
$fopen, $fdisplay, $fwrite // read/write text files
// Not supported in hardware synthesis
```

Randomization and Simulation Time

```
$random, $urandom, $urandom_range // random stimulus
$time, $realtime // simulation time values
// Only valid during simulation
```

Non-Synthesizeable syntax (iv)

Fork-Join Constructs

```
fork ... join, join_any, join_none // parallel execution threads
// Allowed in simulation to mimic concurrency
// Not synthesizable into flip-flops or gates
```

Real and Time Types

```
real, shortreal, time, realtime
// Floating-point values for calculations and simulation logs
// Hardware requires fixed-point or integer math
```

Challenges

- Increasing design complexity (billions of transistors).
- Coverage closure takes time.
- Debugging random test failures.
- Balancing simulation speed vs accuracy.