Verilog testbenches

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

October 22, 2024

Introduction

Objectives:

- Build testbench units for verification of Verilog modules

Reading:

- Lattice Semiconductor: "A Verilog HDL Test Bench Primer", Application note, [Latt99]

The Verilog module to be exercised by the testbench is referred as Circuit Under Test (CUT).

The testbench:

- Generates input vectors for the CUT
- Analyzes the CUT's outputs
- Provides textual information on the passed/failed tests

Testbench approach

Testbench construction **method**:

- for each CUT input, provide in the testbench a *reg* signal with the same name and the same width
- for each CUT output, provide in the testbench a *wire* signal with the same name and the same width
- instantiate the CUT module, connecting each of its ports to the corresponding signals defined above
- generate the CUT inputs

For generating CUT inputs, use any of the patterns described next.

Generating CUT's clock input

Generation of a 50% duty cycle clock, with given period:

```
localparam CLK_PERIOD = 100;
reg clk;
initial begin
  clk = 1'd0;
  forever #(CLK_PERIOD/2) clk = ~clk;
end
```

Important: The *clk* signal constructed above is running indefinitely, making the simulation to never stop!

Generating CUT's clock input (contd.)

Generation of a 50% duty cycle clock, with given period, running for a specified number of cycles:

```
localparam CLK_PERIOD = 100;
localparam RUNNING_CYCLES = 50;
reg clk;
initial begin
  clk = 1'd0;
  repeat (2*RUNNING_CYCLES) #(CLK_PERIOD/2) clk = ~clk;
end
```

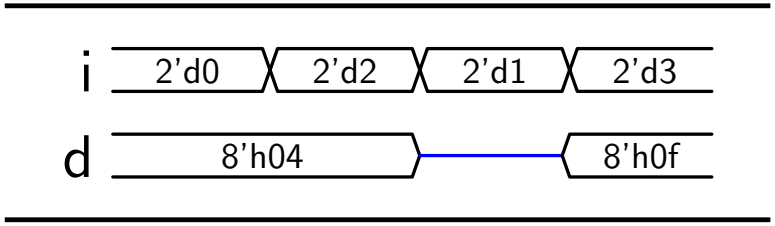
Generating CUT's reset input

Generation of an active low, reset signal, asserted from the initial moment for a given duration:

```
localparam RST_DURATION = 2;
initial begin
  rst_b = 1'd0;
  #RST_DURATION rst_b = 1'd1;
end
```

Generating CUT's inputs with custom waveform

Consider a CUT with 2 inputs, *i*, on 2 bits and *d*, on 8 bits and consider the test process to generate the inputs as in the timing diagram bellow



Since no time unit is given, for brevity, a duration of 10 time units is consider for each configuration on input *i*. Because input *d* modifies synchronously with *i*, it will change at moments multiple of 10 time units.

Each of the two signals in the diagram above will be generated in its own `initial` block.

Note: The blue, mid-height line in *d*'s diagram means the signal is not driven by any source: it is in high impedance.

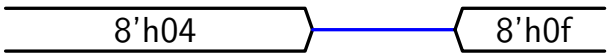
Generating CUT's inputs with custom waveform (contd.)

The code bellow delivers stimuli on input *i*, 10 time units apart:


```
initial begin
  i = 2'd0;      //value of i at moment 0
  #10 i = 2'd2; //value of i at moment 10
  #10 i = 2'd1; //value of i at moment 20
  #10 i = 2'd3; //value of i at moment 30
end
```

Generating CUT's inputs with custom waveform (contd.)

The code bellow delivers stimuli on input *d*:




```
initial begin
  d = 8'h04;      //value of d at moment 0
  #20 d = 8'dz;   //value of d at moment 20
  #10 d = 8'd0f; //value of d at moment 30

end
```

Generating CUT's inputs exhaustively

Consider a CUT with 3 inputs: a 2-bit input x, a 4-bit input d and a single-bit input en.

The fragment bellow generates all 128 possible input configurations (2^7), each one being stable for 20 time units:

```
integer i;
initial begin
  {x, d, en} = 0;
  for (i = 1; i < 128; i = i + 1)
    #20 {x, d, en} = i;
  #20;
end
```

Case study

Exercise: Construct a testbench for exhaustive verification of a 2-to-4 decoder with enable signal and active low outputs, whose implementation is available ▸ here (slide 12).

Solution:

```
1  module dec_2x4_tb;
2    reg [1:0] s;
3    reg e;
4    wire [3:0] y;
6    dec_2x4 cut (
7      .s(s),
8      .e(e),
9      .y(y)
10   );
14   integer i;
15   initial begin
16     {s, e} = 0;
17     for (i=1; i<8; i=i+1)
18       #20 s = i;
19     #20;
20   end
21 endmodule
```

Simulating the testbench in Modelsim

Download the customizable *run.txt* script from here and prepare the script for your project:

- add all Verilog source files, separated by space, to the *sourcefiles* list of line 5
- change the name of the top module for the *topmodule* variable in line 10; typically, this is the name of the testbench module (**not** the name of a Verilog source file)
- run the script with `do run.txt`
- use any of the specific Modelsim commands for simulation

Modelsim commands for simulation

`add wave *`

add the top module's signals to the wave window for visual inspection of signals

`run -all`

runs the simulation forever, or until no signal change value

`run 600`

runs the simulation for 600 time units

`restart`

restarts the simulation from moment 0

`quit -sim`

unload the simulated module without exiting the Modelsim environment

`do run.txt`

recompile and restarts simulation

References

[Latt99] L. Semiconductor. A verilog hdl test bench primer. [Online]. Available: https://people.ece.cornell.edu/land/courses/ece5760/Verilog/LatticeTestbenchPrimer.pdf (Last accessed 17/04/2016).