Laboratory Supplement

Writing testbenches in Verilog language

1. Objectives

O1. Construction of testbench units for checking Verilog modules

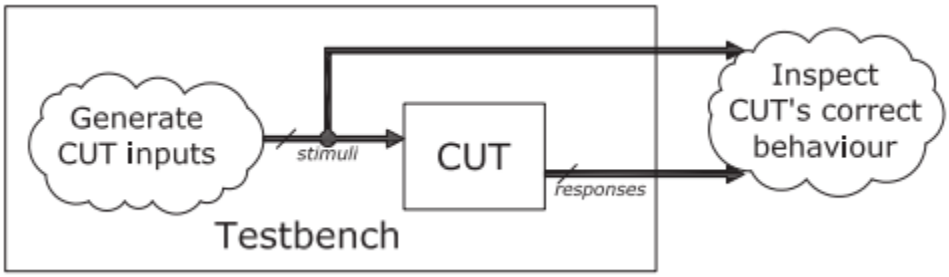
Testbench Approach

The implemented Verilog module, whose correctness must be verified, is also known as **Circuit Under Test (CUT)**, or **Device Under Test (DUT)**.

The **testbench** file:

- evaluates the correctness of the CUT in order to simulate it
- generates input vectors for the CUT
- analyzes the CUT outputs (autonomous or manual)
- provides textual information on successful / failed tests (optional)

The diagram below shows the input-output waveforms corresponding to how to inspect the correctness of a CUT.



The testbench instantiates the CUT, generates its inputs, and routes all CUT ports to the outside for detailed inspection.

Note: Each stimulus will be generated in a dedicated **`initial block.`** The signals that carry stimuli to the CUT inputs are named the same as the inputs to which they connect. These

signals are declared as ***output register ports***. The signals that carry the outputs of the CUTs to the outside are declared as regular ***output ports***.

Generating the clock input for a CUT

The following code generates a clock signal, with a period of 100ns and a filling factor of 50%, starting from logic level 0:

```
reg clk;
initial begin
  clk = 1'd0;
  forever #50 clk = ~clk;
end
```

The ***clk*** signal is switched every half cycle, using the ***# 50*** delay specifier.

To limit a clock running indefinitely, the ***$*** completion symbol can be used to force the simulation to end. The following code fragment finishes the testbench after ***2000ns***:

```
initial begin
#2000 $finish;
end
```

The code below generates ***100 clock cycles*** starting from logic level 1, each cycle having a period of ***150 ns*** and a filling factor of 50%:

```
initial begin
clk = 1'd1;
repeat (200) #75 clk = ~clk;
end
```

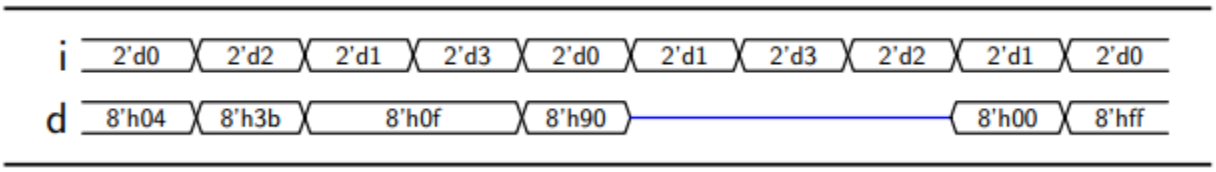
The ***clk*** signal is switched ***200 times*** to generate ***100 cycles*** using the ***repeat (200)*** construct.

The following fragment generates a ***reset*** signal, active on 0 (***LOW***), initially stated for ***2ns***:

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

The reset signal ***rst_b*** is set to logic level 0 and, after 2ns, using the delay specifier ***# 2***, the signal is deactivated by setting it to logic level 1.

Consider a CUT with 2 inputs, ***i***, on 2 bits and ***d***, on 8 bits, and the testing process to generate the inputs as in the time range in the diagram below:



Since no time unit is given, for short circuit (high impedance), a duration of 10ns is considered for each configuration on input ***i***. Since input ***d*** changes synchronously with ***i***, it will change at multiple times of ***10 ns***.

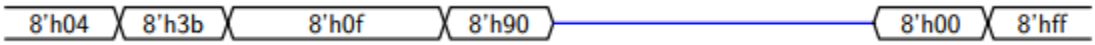
Each of the two signals in the diagram above will be generated in its initial block.

Note: The **d input** is in high impedance from 50ns to 80ns, marked in the timing diagram with a medium height line.

The code below provides stimuli at input i, every 10ns (we will consider the supporting diagram):

```
initial begin
i = 2’d0; // value of i at the moment 0 ns
#10 i = 2’d2; // value of i at the moment 10 ns
#10 i = 2’d1; // value of i at the moment 20 ns
#10 i = 2’d3; // value of i at the moment 30 ns
#10 i = 2’d0; // value of i at the moment 40 ns
#10 i = 2’d1; // value of i at the moment 50 ns
#10 i = 2’d3; // value of i at the moment 60 ns
#10 i = 2’d2; // value of i at the moment 70 ns
#10 i = 2’d1; // value of i at the moment 80 ns
#10 i = 2’d0; // value of i at the moment 90 ns
end
```

Thus, the only aspect that remained is to generate stimuli on the input line ***d***.



The code below will perform this task:

```
initial begin
d = 8’h04; // value of i at the moment 0 ns
#10 d = 8’d3b; // value of i at the moment 10 ns
#10 d = 8’d0f; // value of i at the moment 20 ns
#20 d = 8’d90; // value of i at the moment 30 ns
#10 d = 8’dz; // high impedance at the moment 50 ns
#30 d = 8’d00; // value of d at the moment 80 ns
#10 d = 8’dff; // value of d at the moment 90 ns
end
```

Consider a CUT, with a single input ***x*** on 5 bits. The code fragment below generates all possible input configurations, each input vector being stable for 20 ns:

```
integer i;
initial begin
x = 5'd0;
for (i = 0; i < 32; i = i + 1)
#20 x = i[4:0];
end
```

The signal ***x*** is assigned the value of the integer containing the least significant ***5 bits***, after a corresponding delay. This type of testbench is also called ***exhaustive verification***.

2. References:

[Latt99] L. Semiconductor. A Verilog HDL Test Bench Primer. [Online]. Disponibil: Lattice - A Verilog HDL Test Bench Primer