Lab #4
Using Verilog for Testbenches

Objective

Introduce test benches in Verilog. The trainee will be able to specify a sequence of inputs to be applied by the simulator to an HDL model to test the output.

Discussion

In the previous tutorials, we saw how to perform simulations of our verilog models using Quartus Prime and viewing waveforms with QuestaSim. This is a very useful approach to testing digital models, but can become very cumbersome if the amount of signals that you are looking at is more than a dozen, or you are running very long simulations.

Instead of relying solely on visual inspection of waveforms, your Verilog testbench can actually do an inspection for you - this is called a testbench. In order to build a test bench, you need to know what goes into a good test bench.

- **The Basic Testbench**

The most basic test bench is comprised of the following set of items:

1. A device under test, called a DUT. This is what your testbench is testing.

2. A set of stimulus for your DUT. This can be simple, or complex.

3. A monitor, which captures or analyzes the output of your DUT.

4. You need to connect the inputs of the DUT to the testbench.

5. You need to connect the outputs of the DUT to the testbench.

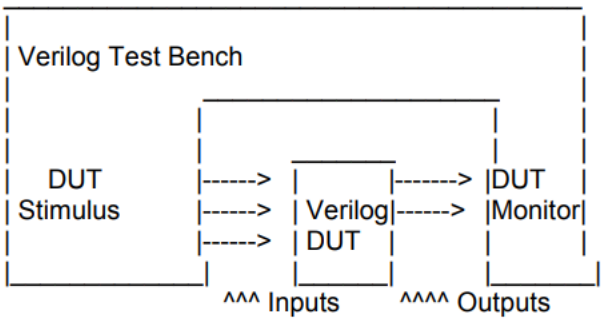
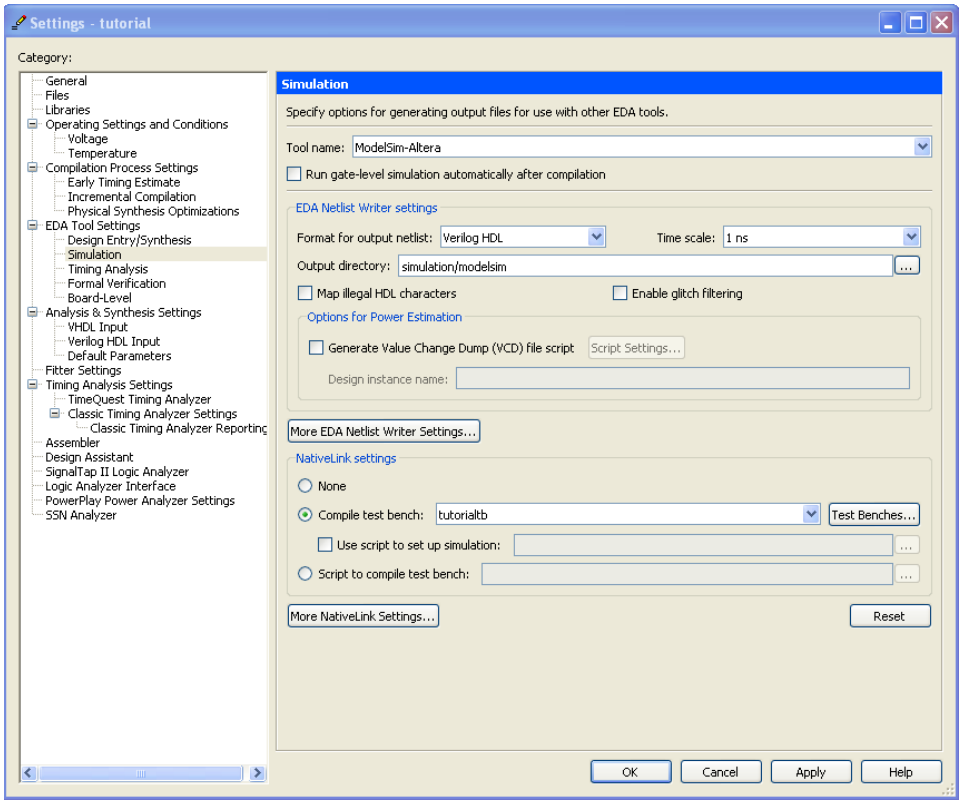


Figure 1. the basic Testbench

How to Create Test Bench File

1. In the Quartus project manager select File-->new-->Design Files-->Verilog HDL File to create a new Verilog file.
2. Write test bench. Save the file as tutorialtb.v –> Compile and verify it is error free.
3. Remove file from project. N.B. It is will still exist in the directory
4. Next, you must provide Quartus with your test bench module name (tutorialtb) and the location and name of the file. To do this, select the "Compile test bench:" settings (under NativeLink settings) and click on the "Test Benches..." button. A window will open, click New. In the "Test bench name" field, enter " tutorialtb ". The "Top level module in test bench" field will automatically fill. In the "test bench files" field, browse for your file tutorialtb.v and add it to the list. Keep all the other settings as they are. Click OK, saving the windows. Your settings window should now look like this



You can see in the below example, from lab #2, Half_Adder.v, the basic requirements for a testbench have been satisfied.

Code 1: Behavioral Verilog model for Half Adder

```
module Half_Adder
(
input wire A,
input wire B,
output reg sum,
output reg carry_out
);

always@ (*)
begin
    sum = A ^ B;
    carry_out = A & B;
end

endmodule
```

Code 2: Testbench model for Half Adder

```
module Half_Adder_TB ();


reg A_TB;
reg B_TB;
wire sum_TB;
wire carry_out_TB;

Half_Adder DUT(
A_TB,
B_TB,
sum_TB,
carry_out_TB
);

initial
begin
    A_TB = 1'b0; B_TB = 1'b0; #10;
    A_TB = 1'b1; B_TB = 1'b0; #10;
    A_TB = 1'b0; B_TB = 1'b1; #10;
    A_TB = 1'b1; B_TB = 1'b1; #10;
end

endmodule
```

Code 3: Behavioral Verilog model for 4 to 1 MUX

```
module mux (
    output reg  [3:0]  o_out,
    input  wire [3:0]  i_in0,
    input  wire [3:0]  i_in1,
    input  wire [3:0]  i_in2,
    input  wire [3:0]  i_in3,
    input  wire [1:0]  i_sel
);

always @ (*)
    begin
        case (i_sel)
        2'b00:   o_out = i_in0;
        2'b01:   o_out = i_in1;
        2'b10:   o_out = i_in2;
        2'b11:   o_out = i_in3;
        default: o_out = 'b0;
        endcase
    end

endmodule
```

Code 4: Testbench model for 4 to 1 MUX

```
`timescale 1ns/1ps

module mux_tb ();

wire [3:0] o_out_tb;
reg  [3:0] i_in0_tb;
reg  [3:0] i_in1_tb;
reg  [3:0] i_in2_tb;
reg  [3:0] i_in3_tb;
reg  [1:0] i_sel_tb;

// Module Instantiation
mux dut (
    .o_out (o_out_tb),
    .i_in0 (i_in0_tb),
    .i_in1 (i_in1_tb),
    .i_in2 (i_in2_tb),
    .i_in3 (i_in3_tb),
    .i_sel (i_sel_tb)
);

initial
    begin
        // Initialization
        $monitor ("[%0t] i_sel_tb = 0x%0h i_in0_tb = 0x%0h i_in1_tb =
0x%0h i_in2_tb = 0x%0h i_in3_tb = 0x%0h o_out_tb = 0x%0h", $time,
i_sel_tb, i_in0_tb, i_in1_tb, i_in2_tb, i_in3_tb, o_out_tb);

        i_sel_tb = 0;
        i_in0_tb = $random;
        i_in1_tb = $random;
        i_in2_tb = $random;
        i_in3_tb = $random;

        for (int i = 0; i < 4; i++)
            begin
                #5
                i_sel_tb <= i;
            end

        #5 $finish;
        end
endmodule
```

Code 5: Behavioral model for decoder

```
module decoder(in,out);
input [1:0] in;
output reg [3:0] out;
always@(*)
begin
    case(in)
    0:out=4'b1000;
    1:out=4'b0100;
    2:out=4'b0010;
    3:out=4'b0001;
    default:out=4'b0000;
    endcase
end
endmodule
```

Code 6: Testbench model for decoder

```
module decoder_tb();
//Signal declaration
reg[1:0] in_tb;
wire [3:0] out_tb;
//DUT instantiation
decoder dut(in_tb,out_tb);
//Test stimulus generator
integer i;
initial
begin
    for(i=0;i<4;i=i+1)
    begin
        in_tb=i;
        #100;
    end
    $stop;
end
//Test monitor to report the result
initial begin
    $monitor("in = %d , out =
%b,",in_tb,out_tb);
end

endmodule
```

Code 7: Behavioral model for comparator

```
module comparator(A,C,out);

input [3:0]A;
input [3:0]C;
output reg[1:0]out;

always@(*)
begin

if(A>C)
out=2'b00;
else if(A<C)
out=2'b01;
else
out=2'b10;

end
endmodule
```

Code 7: Testbench model for comparator

```
module comparator_tb();

reg [3:0]A_t,C_t;
wire [1:0] out_t;

//wire [1:0]out_exp;
comparator DUT(.A(A_t),.C(C_t),.out(out_t));

initial
begin

A_t=4'b0000;
C_t=4'b0000;
check(2'b00);

#10
A_t=4'b0100;
C_t=4'b0010;
check(2'b00);

#10
A_t=4'b0000;
C_t=4'b1000;
check(2'b10);

end

task check (input [1:0] out_exp);
begin
#1
if(out_exp==out_t)
$display("test_case_is_successfull_out_is  %b and out_exp is
%b",out_t,out_exp);

else
begin
$display("test_case_is_failed_out_is %b and out_exp is %b",out_t,out_exp);
end
#10
$display("comparison completed at time = %0t", $time);
end

endtask

endmodule
```

Code 8: Behavioral model for 7 segment decoder

```
module Seven_Segment_Decoder
(
input wire [1:0] Binary_Num,
output reg [6:0] Decimal_Num
);

localparam NUM_0 = 7'b0111111,
              NUM_1 = 7'b0000110,
              NUM_2 = 7'b01011011,
              Default = 7'b0000000;

always@ (*)
begin
    case(Binary_Num)
    2'b00: Decimal_Num = NUM_0;
    2'b01: Decimal_Num = NUM_1;
    2'b10: Decimal_Num = NUM_2;
    default: Decimal_Num = Default;
    endcase
end

endmodule
```

Code 8: Testbench model for 7segment decoder

```
module Seven_Segment_Decoder_TB ();

reg [1:0] Binary_Num_TB;
wire [6:0] Decimal_Num_TB;

Seven_Segment_Decoder DUT(
Binary_Num_TB,
Decimal_Num_TB
);

localparam NUM_0 = 7'b0111111,
              NUM_1 = 7'b0000110,
              NUM_2 = 7'b01011011,
              Default = 7'b0000000;
initial
begin
    // Case 0
    Binary_Num_TB = 2'b00; #10;
    if(Decimal_Num_TB == NUM_0)
        $display("case 0 passed");
    else
        $display("case 0 failed");
    // Case 1
    Binary_Num_TB = 2'b01; #10;
    if(Decimal_Num_TB == NUM_1)
        $display("case 1 passed");
    else
        $display("case 1 failed");
    // Case 2
    Binary_Num_TB = 2'b10; #10;
    if(Decimal_Num_TB == NUM_2)
        $display("case 2 passed");
    else
        $display("case 2 failed");
    // Case 3
    Binary_Num_TB = 2'b11; #10;
    if(Decimal_Num_TB == Default)
        $display("case 3 passed");
    else
        $display("case 3 failed");
end
endmodule
```

Code 9: Behavioral model for TOP (half adder + 7 segment decoder)

```
module TOP
(
input wire operand_A,
input wire operand_B,
output reg [6:0] Decimal_Num
);

wire sum_signal, carry_out_signal;
wire [6:0] Decimal_Num_signal;

Half_Adder U0 (
operand_A,
operand_B,
sum_signal,
carry_out_signal
);

Seven_Segment_Decoder U1 (
{carry_out_signal,sum_signal},
Decimal_Num_signal
);

always@ (*)
begin
    Decimal_Num = Decimal_Num_signal;
end

endmodule
```

Test vector	Golden reference
00	0111111
01	0000110
10	0000110
11	1011011

Code 9: Testbench model for TOP (half adder + 7 segment decoder)

```
module TOP_TB ();

reg operand_A_TB;
reg operand_B_TB;
wire [6:0] Decimal_Num_TB;

TOP DUT (
    operand_A_TB,
    operand_B_TB,
    Decimal_Num_TB
);

reg [1:0] mem_in [3:0];
reg [6:0] mem_out [3:0];

task initialize;
begin
    $readmemb("C:/Users/sondo/Desktop/New folder (2)/INPUT.txt",mem_in);
    $readmemb("C:/Users/sondo/Desktop/New folder (2)/OUTPUT.txt",mem_out);
end
endtask

integer i;

initial
begin
    initialize;
    for (i = 0; i < 4 ; i = i + 1)
    begin
        {operand_A_TB,operand_B_TB} = mem_in [i]; #10;
        if (Decimal_Num_TB == mem_out [i]) $display("Case %d Passed",i+1);
        else $display("Case %d Failed",i+1);
    end
    $stop;
end


endmodule
```

Note: put the whole path of the testvector and golden reference files in the readmem in the testbench.

Assignment:

Write testbench for the following design:

4-bit Comparator using subtractor.

Has three outputs denote the following:

Z = 1 if the result is 0. (zero flag)

N = 1 if the result is negative. (negative flag)

V = 1 if the result overflows. (overflow flag)

Bonus :

Write testbench model for the following:

BCD Adder.

Deliverables

Verilog codes for all testbench with screenshots for simulated waveform including all test cases.