How To Write a Test Bench Verilog File To Verify Your Design

A test bench file is a Verilog file but with some special features. It is used to send inputs to your designs and displays the output values from your design (*Simulation*). If the output values match the expected values, then your designs are correct. If not, you have to rewrite your code in Verilog again to better describe the function (behavior) of your designs.

Remember always that the input values you apply in this test bench file have to cover most (if not all) the combinations of the inputs (they are called *test vectors*). For each of these combinations, there exists a unique value of each output. This unique value is the one you expect the simulator to display for you for proper operation of the design.

To define the test vectors is very difficult task specially for big or complicated designs. In our *Advanced Logic Design* course, we use simple designs and hence, the test vectors are few lines and in most cases, they will be given to you.

The golden rules that you have to follow when you write the test bench Verilog file are:

1- The test bench file has neither inputs nor outputs.
Example: *module test ; //* test *is the test bench module name.*

2- All the inputs in your design (the original File_Name or module you are going to test) have to be declared as registers (*reg*) in the test bench file.

3- If any of the outputs in your design is more than one bit, they have to be declares as *wire*.

4- Each time you *instantiate* your design in your test bench, you have to assign it a different local name.

Example: If your design file looks like

```
module  prob_3 (a,b,c);
 input          a,b;    // one bit inputs.
output [2:0] c   ;   // 3-bit output.
-                      // here you define the bulk of your code to describe the
-                      // function (behavior) of your design.
endmodule
```

Then your test bench file will look like;

```
module  test;              // test bench file has no I/O
  reg         a,b;         // inputs in the design file are defined as reg in test bench.
 wire   [2:0] c   ;        // outputs that are more than one bit in the design file are
                           //  declared as wire in the test bench.
 prob_3     p1(a,b,c);  // the design file is instantiated with local name p1.
     -                    // here you write your test vectors and $monitor function.
 endmodule
```

5- Your test bench will normally, have two ***initial*** procedures. The first one has the test vectors and must be ended with ***$finish*** task to end the simulation process. The

second one has the ***$monitor*** task to display the input/output values during simulation.

Example: If your design file looks like:

```
module  compare(a,b,c);  // this module compare the values of two inputs for equality.
  input a,b;
 output  c;
 reg c;

always @(a or b)
  begin
    if (a = = b)
      c=1;
    else
      c=0;
  end
endmodule
```

The test bench will look like:

```
module  test;  // this module name is test.
   reg a,b;       // Inputs in  the design file are declared as reg type.
                  //  one-bit Outputs in the design file need not to be declared.

compare  c1(a,b,c);   // the design file is instantiated with local name c1.

initial                 // first initial statement is  to define test vectores.
  begin
     a=0;               // first input combination at time t=0.
     b=1;

 #5 a=0;
     b=0;               // second input combination at time t=5.

 #5 a=1;
     b=0;               // third input combination at time t=10 (the time is accumulated. So, 0+5+5=10).

 #5 a=1;                //  the last combination at time t=15.
    b=1;

#5 $finish;          // Remember, this line has to be added at the end of the test vectors to stop the simulation.
end


initial                // second initial statement is to display I/O.
   begin
    $monitor( $time, “   a=%b       b=%b        c=%b”, a , b , c); // %b defines the binary type. You can use %d for decimal.
  end
endmodule
```

when you run this files using a Veilog simulator (for example, VeriWell), the output will look like:

```
                0    a=0   b=1    c=0
                5    a=0   b=0    c=1
                10   a=1   b=0    c=0
                15   a=1   b=1    c=1
VeriWell finish simulation at  20.
```

How To use The *VeriWell* Simulator

1- To start writing a Verilog code (file), from the *File* menu, select *New*. (If you already wrote the Verilog file, select *Open* instead and locate the file from your hard drive or from drive *a:* by the same way you open any file in *Word*, for example).

2- Note that while editing, the Verilog keywords are in blue colors, your words are in black, and the simulator commands *($monitor, $finish, $time*) are in red.

3- After finishing writing the code, use the *Save As* option from the *File* menu to save the file (the default is with extension **.v**, if not, make sure the file has an extension **.v**). **You have to name the file with the same *module name*.**

example:-

module ***prob_2****(a,b,c);*

-

-

-

endmodule

Then, you have to save the file as **prob_2.v**

4- Select *New Project* from the *Project* menu. You name the project with appropriate name. (again, if you already used the project before, you use **Open Project** instead).

5- Use the *Add File* option from the *Project* menu to add the file you saved in step 3. **(Make sure that you add the *test* file first, then *your design code*)**

6- Use the *Run* option from the *Project* menu to compile the file. The Veriwell simulator will open a new window (*Console*) that gives you the simulation time, the Verilog file being simulated, and displays the errors, if any, each on a separate line starting with the line number. For example, (*L7: missing "*) means that Veriwell simulator is expecting " in line # 7 on your Verilog code.

7- If you have any errors, repeat steps 1-6 again but you use *Open* instead of *New* in step 1 and *Save* instead of *Save As* in step 3.

8- When you code is error free, the simulator will display for you:
No errors in compilation
Top-level modules:
File_Name

Where, File_Name is your module name (the Verilog file you saved and being simulated).

Your code is syntactically correct. now, you have to write another code (test bench) to check that the behavior (function) if your design is correct.

9- Close your Project using the *CloseProject* option from the *Project* menu.

10- Write a new test_bench Veilog file (see attachments to see how you write a test_bench Verilog file). Follow the same steps from 1-9 above.

11- Now you are ready to test the function of your design. First open and new project. Then add the test_bench file first. Add the File_Name.v file you tested in steps 1-9. You will see the order of the files being added on the console window looks like:

C:\Program Files\VeriWell\exe\test_bench.v
C:\Program Files\VeriWell\exe\file_Name.v

Be sure to **add the test_bench.v file to your project first**.

12- Run the new project. The first round will check the Verilog syntax of your modules. If your codes are error free, click on the green right arrow on the menu bar on the top of your console window. It is going to send the inputs in your test_bench to your File_Name and return the expected output values for you, this is what is called simulation.

Example: *D* type Flip Flop (copy and paste in your VeriWell simulator)

```
module d_ff(reset,clk,d,q);

input d,clk,reset;
output q;
reg q;

always @ (posedge clk)  // D FF with synchronous reset.
   begin
         if(reset = =1)
           q=0;
         else
           q=d;
   end
endmodule
```

The test code is:

```
module test_dff;

reg d,clk,reset;

d_ff  d1(reset,clk,d,q);

initial   // simulate the behavior of the clock with 20 time unit period
         begin
                  clk=1'b0;
                  forever #10 clk=~clk;
         end
```

```
initial
        begin
                reset=1; d=1;   // first test bench

                #15 reset=0; d=1; // make sure you passed the clk edge (after 10 time units)

                #20 d=0;            // another test bench

                #20 $finish;       // stop the simulator
        end

initial
        begin    // monitor the values of the I/O during simulation
          $monitor ($time, " clk=%b      reset=%b     d=%b       q=%b", clk,reset,d,q);
        end

endmodule
```

Here are the results after running the VerWell simulator:

```
Memory Available: 0
Entering Phase I...
Compiling source file : test_dff.V
Compiling source file : d_ff.V
The size of this model is [2%, 2%] of the capacity of the free version

Entering Phase II...
Entering Phase III...
No errors in compilation       // Your code is Verilog-HDL syntactically correct
Top-level modules:
  test_dff                     // Click on the green arrow (third form right) to display the results

C1> .
                0 clk=0     reset=1    d=1      q=x
               10 clk=1     reset=1    d=1      q=0
               15 clk=1     reset=0    d=1      q=0
               20 clk=0     reset=0    d=1      q=0
               30 clk=1     reset=0    d=1      q=1
               35 clk=1     reset=0    d=0      q=1
               40 clk=0     reset=0    d=0      q=1
               50 clk=1     reset=0    d=0      q=0
Exiting VeriWell for Win32 at time 55
0 Errors, 0 Warnings, Memory Used: 39675
Compile time = 0.0, Load time = 0.0, Simulation time = 0.0
```