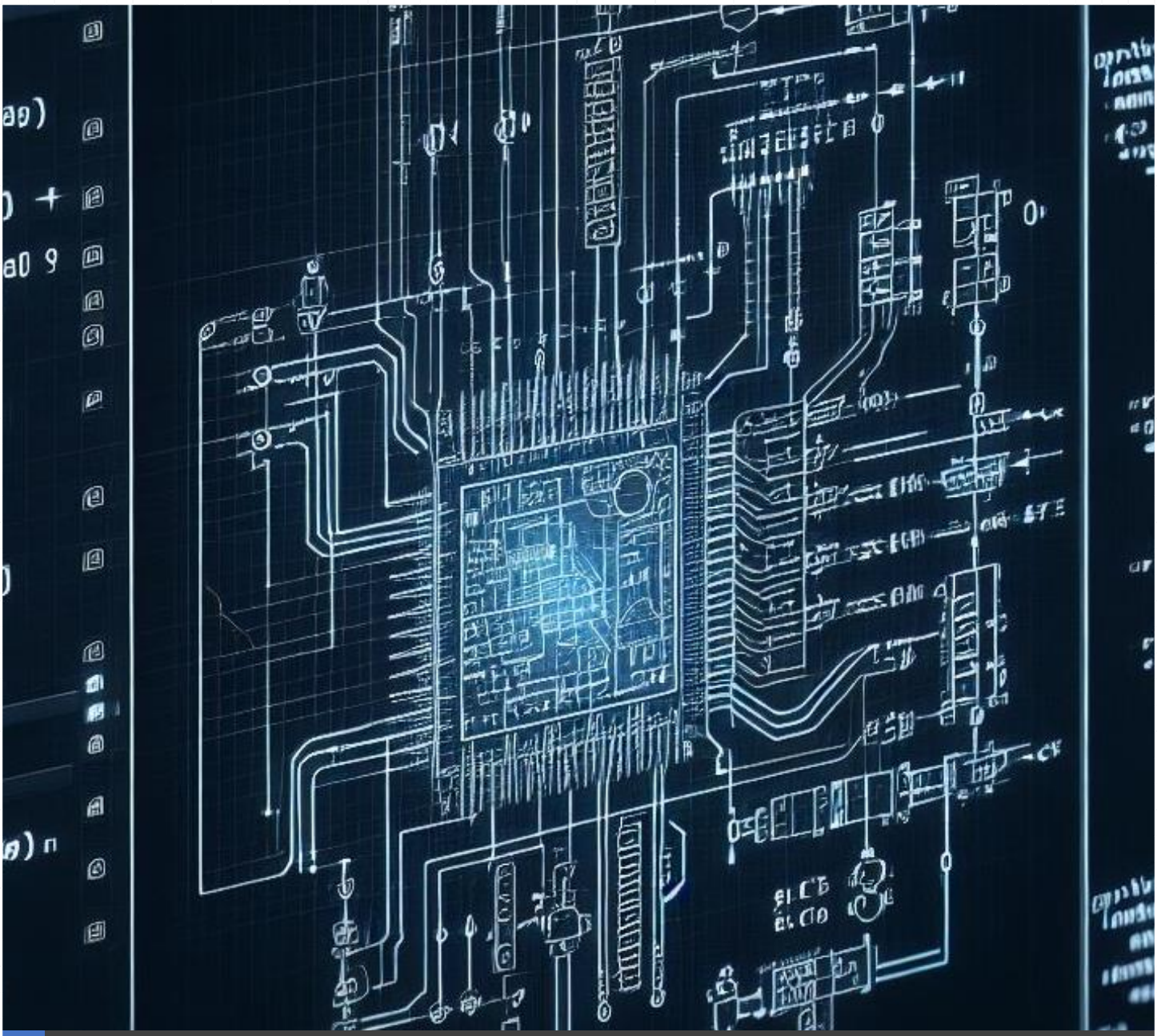


Verilog Tutorial

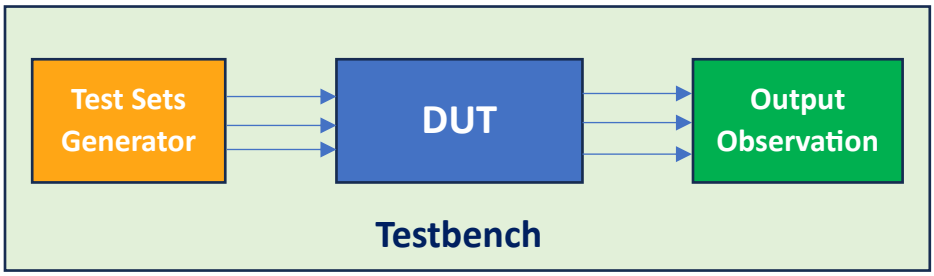
Part 2

Sameh Mohamed
GitHub: SamehM20
LinkedIn: Sameh Elbatsh

Contents

1. Testbench:

A testbench is an environment for simulating a functional interaction with the design to test its functionality against multiple test sets. It is a way to catch bugs and check the correctness of the design early in the development to save cost and time.



- **Test Sets Generator**: generating inputs values to test the design. It can be directive generation where you put specific values, random generation where random numbers generation functions are used to determine input values, or covering all the values automatically using loops.
- **DUT**: Design Under Test. It's the design being tested by the testbench. It is instantiated inside the testbench.
- **Output Observation**: The output can be observed to check its correctness by multiple ways: outputting every output to log window, saving the outputs to a log file, checking every output against its expected value, or checking every output generated and only outputting the wrong set.

2. Testbench Syntax:

The syntax is like a normal module except that there are no inputs nor outputs expected, so the port list is discarded and the inner signals are used instead. Then, the inputs to the DUT are declared as **reg** since they are holding values and the outputs of the DUT are declared as **wire** since they are only for observation. The simple syntax is as the following:

```
`timescale [time_unit]/ [time_precision]
module [testbench_name];

[list_of_i/o_for_the_DUT]
[inner_signals&variables_declarations]

[instantiation_of_the_DUT]

[initial_block]
[...any_other_block_if_needed]
endmodule
```

We will illustrate every part soon.

3. Delay and Timescale:

To simulate the functionality, delays are required between every consecutive test sets. this can be simply done using '#' followed by the delay value. Ex:

10ns → waits 10 nano seconds before executing the next statement.

`timescale: It defines the unit of time used in the testbench (for delay) and the precision time which is the resolution for the value of the time (how time values are rounded). Its syntax is:

`timescale [time_unit]/[time_precision]

For example: **`timescale** 2ns/1ps

That means if we used # 10, the value of that delay will be 10*2ns = 20ns

`timescale 1ns/1ns

That means if we used # 10, the value of that delay will be 10*1ns = 10ns and if we used # 10.7, the value of that delay should have been 10.7*1 = 10.7ns, but since the precision is 1ns, the time steps are in orders of 1ns, so the final delay values will be 11ns.

4. Initial Block:

The **initial** block follows the rules of **always** block, but it differs in a very important feature. The **initial** block is executed only one time, but the **always** block executes forever or every time a member of the **sensitivity list** triggers it. The syntax is as the following:

- For single statement: **initial statement;**
- For multiple statements:

```
initial begin
statement1;
statement2;
……
end
```

5. Another Variable Types:

There are some other types which can be helpful in the testbench. Some of them are as the following:

integer: can hold 32-bits integer values, **integer** x = 4165;

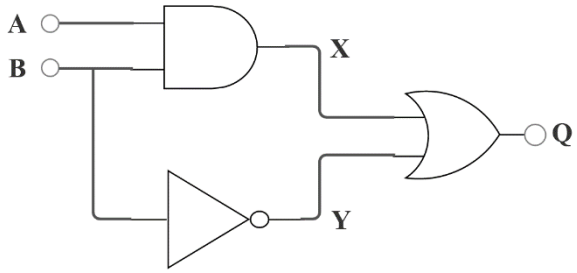
time: can hold 64-bits of time values, **time** x = 23ps;

real: can hold floating point values, **real** x = 854.265;

realtime: can hold floating point time values, **realtime** x = 562.32ps;

6. Example:

Consider the circuit used in part 1.



A	B	X	Y	Q
0	0	0	1	1
0	1	0	0	0
1	0	0	1	1
1	1	1	0	1

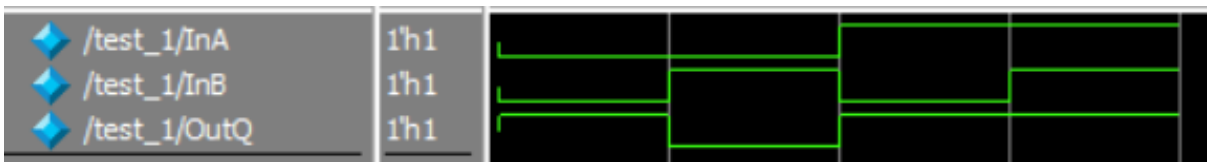
It has two inputs (A, B), one output (Q). The module used was named "**circuit_1**". We can write a simple testbench to simulate the 4 sets of inputs in a directed testing style as the following:

```
`timescale 1ns/1ps
// Choosing the testbench name to be test_1
module test_1;
 reg InA, InB; // Declaring the test inputs.
 wire OutQ;    // Declaring the observed output.

// Instantiating the design.
circuit_1 DUT (.A(InA), .B(InB), .Q(OutQ));

initial begin
// Test Sets.
  InA = 0;
  InB = 0;
  #10   // Waiting for 10*1ns.
  InA = 0;
  InB = 1;
  #10
  InA = 1;
  InB = 0;
  #10
  InA = 1;
  InB = 1;
end
endmodule
```

Running the testbench and observing the waveform, we find the result matched the truth table.



7. $display & $monitor:

Verilog has some built-in functions. For outputting the values on log window, we can use **$display** or **$monitor**. Their syntax is the same:

$display("String", data1, data2, …)

$monitor("String", data1, data2, …)

"String": The string contains characters and format specifier. For example:

"The output will be **%d**" → **%d** saves location for a decimal value.

"Output is **%d**, input is **%h**" → **%d** decimal, **%h** hexadecimal.

data1, data2, …: They are the values at the locations of the specifiers in the **String.** The order of these data must be in order of specified in the string.

The difference between **$display** and **$monitor** is that **$display** only executes once, but **$monitor** executes every time one of the data specified changes.

We can write the previous testbench with the monitoring function to be able to observe the output outside the waveform.

```
`timescale 1ns/1ps
// Choosing the testbench name to be test_1
module test_1;
 reg InA, InB; // Declaring the test inputs.
 wire OutQ;    // Declaring the observed output.

// Instantiating the design.
circuit_1 DUT (.A(InA), .B(InB), .Q(OutQ));

initial begin
  $monitor("Input A= %d, B = %d → Output Q = %d", InA, InB, OutQ);
// Test Sets.
  InA = 0;
  InB = 0;
  #10
  InA = 0;
  InB = 1;
  #10
  InA = 1;
  InB = 0;
  #10
  InA = 1;
  InB = 1;
end
endmodule
```

Running the simulation, the log window will have the following lines:

```
VSIM 10> run
# Input A = 0, B = 0 ? Output Q = 1
# Input A = 0, B = 1 ? Output Q = 0
# Input A = 1, B = 0 ? Output Q = 1
# Input A = 1, B = 1 ? Output Q = 1
```

8. For Loop:

The **for** loop iterates over the included statement/s until the iteration index condition is violated. It can be used inside a block only. The syntax is as the following:

for([initial_condtion]**;** [iteration_condition]**;** [step_defining]**) statement;**

Or

for([initial_condtion]**;** [iteration_condition]**;** [step_defining]**) begin**

list_of_statements;

end

For example:

```
integer i; // Declaring the iteration index.

// Any block like always.
initial begin
  for(i = 0; i < 3; i = i + 1) begin
        $display("Current Index = %d", i);
  end
end
```

The output will be:

Current Index = 0

Current Index = 1

Current Index = 2

Using loops, we can cover all possible combination of inputs with ease and a small amount of line. So, we can rewrite the last testbench as:

```
`timescale 1ns/1ps
// Choosing the testbench name to be test_1
module test_1;
 reg InA, InB; // Declaring the test inputs.
 wire OutQ;    // Declaring the observed output.

 integer i, j; // Declaring the iteration indices.

 circuit_1 DUT (.A(InA), .B(InB), .Q(OutQ));

initial begin
  $monitor("Input A= %d, B = %d → Output Q = %d", InA, InB, OutQ);
  // Test Sets.
  for(i = 0; i < 2; i = i + 1) begin
     InA = i;
    for(j = 0; j < 2; j = j + 1) begin
        InB = j;
        #10;
    end
  end
end
endmodule
```

The output will be the same as the last testbench.

```
VSIM 10> run
# Input A = 0, B = 0 ? Output Q = 1
# Input A = 0, B = 1 ? Output Q = 0
# Input A = 1, B = 0 ? Output Q = 1
# Input A = 1, B = 1 ? Output Q = 1
```

9. While Loop:

The **while** loop is much like the **for** loop, but more general. It iterates over the included statement/s until the condition is violated. It can be used inside a block only. The syntax is as the following:

while([iteration_condition]**) statement;**

Or

while([iteration_condition]**) begin**

list_of_statements;

end

For example: To make the one done with for loop.

```
integer i = 0; // Declaring the iteration index with initial value.

// Any block like always.
initial begin
  while(i < 3) begin
        $display("Current Index = %d", i);
        i = i + 1;
  end
end
```

The output will be:

Current Index = 0

Current Index = 1

Current Index = 2

Using **while** loop to make the testbench as in the **for** loop, we can rewrite the testbench as:

```
`timescale 1ns/1ps
// Choosing the testbench name to be test_1
module test_1;
 reg InA, InB; // Declaring the test inputs.
 wire OutQ;    // Declaring the observed output.

 // Declaring the iteration indices with initial values.
 integer i = 0;
 integer j = 0;

 circuit_1 DUT (.A(InA), .B(InB), .Q(OutQ));

initial begin
  $monitor("Input A= %d, B = %d → Output Q = %d", InA, InB, OutQ);
  // Test Sets.
  while(i < 2) begin
     InA = i;
    j = 0; // Don't forget to initialize the index for every full loop.
    while(j < 2) begin
        InB = j;
        #10
        j = j + 1;
    end
    i = i + 1;
  end
end
endmodule
```

The output will be the same as the last testbench.

```
VSIM 10> run
# Input A = 0, B = 0 ? Output Q = 1
# Input A = 0, B = 1 ? Output Q = 0
# Input A = 1, B = 0 ? Output Q = 1
# Input A = 1, B = 1 ? Output Q = 1
```