3.26 Testbench (Verilog)

A ***testbench*** provides a sequence of input values to test a module. A testbench is itself a module with no inputs or outputs,
two main elements:

1. A module instantiation that creates an instance of the module being tested.
2. A procedure for generating the sequence of input values to test the module.

The testbench creates an instance of the module being tested. A ***module instantiation*** creates an instance of a module, giv
a name, and specifies how the module's ports are connected. Variables connect to the module's inputs, while wires connec
output. A ***wire*** is a named connection within a module.

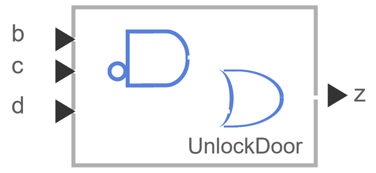
A ***port connection*** connects variables and wires to the inputs and outputs of the module instance. An ***ordered port connect***
connections following the order of the module definition's port list.

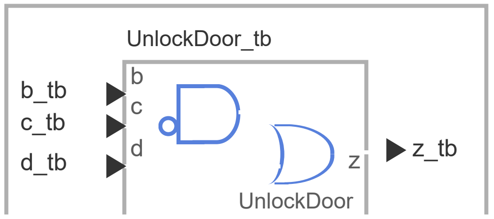
This material appends _tb to the names for module instances, variables, and wires within the testbench.

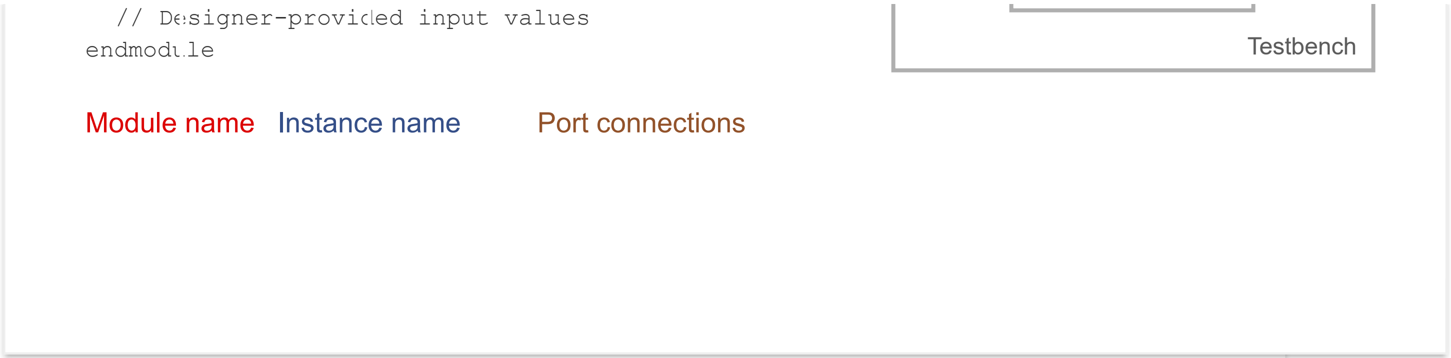
PARTICIPATION ACTIVITY 3.26.1: A testbench instantiates the module to test, and prepares to set the module's input values and check output values.

Start 2x speed

```
module UnlockDoor(b, c, d, z);
  input b, c, d;
  output reg z;

  // Module description goes here

endmodule


module Testbench();
  reg b_tb, c_tb, d_tb;
  wire z_tb;

  UnlockDoor UnlockDoor_tb(b_tb, c_tb, d_tb, z_tb);
```







```
  // Designer-provided input values
endmodule
```

PARTICIPATION ACTIVITY 3.26.2: Module instantiation.

Complete the testbench for each module. Use the naming convention of appending _tb for all variable, wire, and module instance names.

1)
```
module CheckSensor(a, b, m);
   input a, b;
   output reg m;

   // Module description
endmodule
```

```
module Testbench();
   [                    ]
   wire m_tb;

   CheckSensor
CheckSensor_tb(a_tb, b_tb,
m_tb);

   // Designer-provided input
values
endmodule
```

Check Show answer

2)

```
module ControlValve(a, p, r);
   input a;
   output reg p, r;

   // Module description
endmodule
```

```
module Testbench();
   reg a_tb;
   

   ControlValve
ControlValve_tb(a_tb, p_tb,
r_tb);
   // Designer-provided input
values
endmodule
```

Check Show answer

3)
```
module SafetyCheck(a, x, y, z);
   input a;
   output reg x, y, z;

   // Module description
endmodule
```

```
module Testbench();
   reg a_tb;
   wire x_tb, y_tb, z_tb;

   
(a_tb, x_tb, y_tb, z_tb);
   // Designer-provided input
values
endmodule
```

Check Show answer

4)
```
module OpenVault(f, s, t, u);
   input f;
   output reg s, t, u;

   // Module description
endmodule

module Testbench();
   reg f_tb;
   wire s_tb, t_tb, u_tb;

   OpenVault OpenVault_b(
                          );
   // Designer-provided input
values
endmodule
```

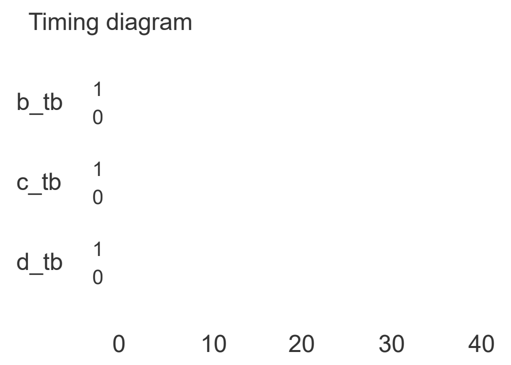
Check Show answer

The sequence of input values for testing a module appears within an initial procedure. An ***initial procedure*** defines a staten that executes once at the start of simulation. Each unique sequence of values used to test a module is known as a ***test vec***

PARTICIPATION ACTIVITY 3.26.3: A testbench generates input values using an initial procedure.

Start 2x speed

```
`timescale 1 ns / 1 ns

module UnlockDoor(b, c, d, z);
  input b, c, d;
  output reg z;

  // Module description goes here

endmodule

module Testbench();
  reg b_tb, c_tb, d_tb;
```



```
  wire z_tb;
  UnlockDoor UnlockDoor_tb(b_tb, c_tb, d_tb, z_tb);

  initial begin
    b_tb = 0;
    c_tb = 0;
    d_tb = 0;
    #10 b_tb = 1;
    #10 c_tb = 1;
    #10 c_tb = 0;
    d_tb = 1;
    #10 b_tb = 0;
  end
endmodule
```

Time (ns)

A ***delay control*** delays simulation of a procedure for a specified time, starting with the hash character (#) followed by the n units to delay simulation. Ex: `#10` delays simulation for 10 time units. Delay controls can be prepended to statements like `#` or appear as separate statements like `#10;`.

A ***timescale directive*** defines the length of each time unit. The timescale directive starts with `` `timescale `` followed by the specification, a slash character (/), and the time precision specification. Note the timescale directive starts with an accent (`) not an apostrophe ('). The time precision defines how the simulator should internally keep track of time. Ex: `` `timescale `` specifies each time unit is 1 ns (or nanosecond).

The drawn value over time for an input or output is called a ***waveform***, as in the above animation's timing diagram.

Table 3.26.1: Time unit specifications.

String	Time units
s	seconds
ms	milliseconds

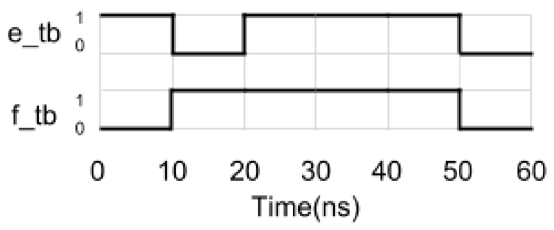
us	microseconds
ns	nanoseconds
ps	picoseconds
fs	femtoseconds

PARTICIPATION ACTIVITY

3.26.4: Testbench: Generating waveforms.

```
`timescale 1 ns/ 1 ns

module SetTimer(e, f, m);
   input e, f;
   output reg m;

   always @(e, f) begin
      m = e | f;
   end
endmodule

module Testbench();
   reg e_tb, f_tb;
   wire m_tb;

   SetTimer SetTimer_tb(e_tb, f_tb, m_tb);

   initial begin
      (a)
      f_tb = 0;
      #10 (b)
      f_tb = 1;
      (c) e_tb = 1;
      (d) e_tb = 0;
      f_tb = 0;
   end
endmodule
```

Timing diagram



1) (a)

- ○ e_tb = 0;
- ○ e_tb = 1;

2) (b)

- ○ e_tb = 0;
- ○ e_tb = 1;
  f_tb = 1;
- ○ f_tb = 0;

3) (c)

- ○ #10
- ○ 10
- ○ #1

4) (d)

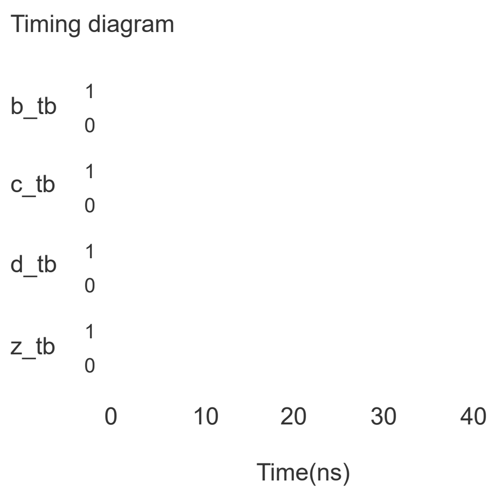
- ○ #10
- ○ #20
- ○ #30

A simulator will simulate both the testbench and module instance simultaneously, generating waveforms showing the inpu values. As the testbench assigns input values for the module being tested, those changes will cause the module's always p execute, which will then update the module's output values.

PARTICIPATION ACTIVITY 3.26.5: Testbench simulation process.

Start ☐ 2x speed

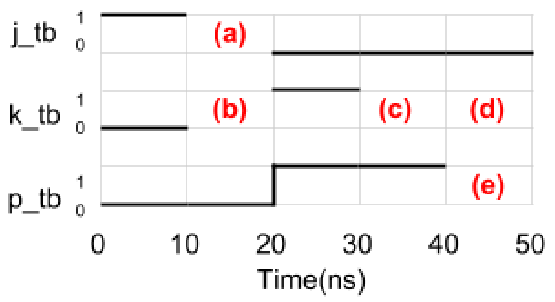
```
`timescale 1 ns / 1 ns

module UnlockDoor(b, c, d, z);
  input b, c, d;
  output reg z;

  always @(b, c, d) begin
    z = ~b + (c & ~d);
  end
endmodule


module Testbench();
  reg b_tb, c_tb, d_tb;
  wire z_tb;

  UnlockDoor UnlockDoor_tb(b_tb, c_tb, d_tb, z_tb);

  initial begin
    b_tb = 0;
    c_tb = 0;
    d_tb = 0;
    #10 b_tb = 1;
    #10 c_tb = 1;
    #10 c_tb = 0;
    d_tb = 1;
    #10 b_tb = 0;
  end
endmodule
```



PARTICIPATION ACTIVITY 3.26.6: Testbench simulation.

```
`timescale 1 ns/ 1 ns

module AlarmOff(j, k, p);
   input j, k;
   output reg p;

   always @(j, k) begin
      p = ~j & k;
   end
endmodule

module Testbench();
   reg j_tb, k_tb;
   wire p_tb;

   AlarmOff AlarmOff_tb(j_tb, k_tb, p_tb);

   initial begin
      j_tb = 1;
      k_tb = 0;
      #10 k_tb = 1;
      #10 j_tb = 0;
      #20 k_tb = 0;
   end
endmodule
```

Timing diagram



1) (a)

- 0
- 1

2) (b)

- 0
- 1

3) (c)

○ 0

○ 1

4) (d)

○ 0

○ 1

5) (e)

○ 0

○ 1

Provide feedback on this section