

Here, we will design some directed testbench for simple modules.

Mux Module:

```
module mux21 (input a,b,s,  
    assign y=~s & a | s & b;  
endmodule
```

Flat Testbench with MUX

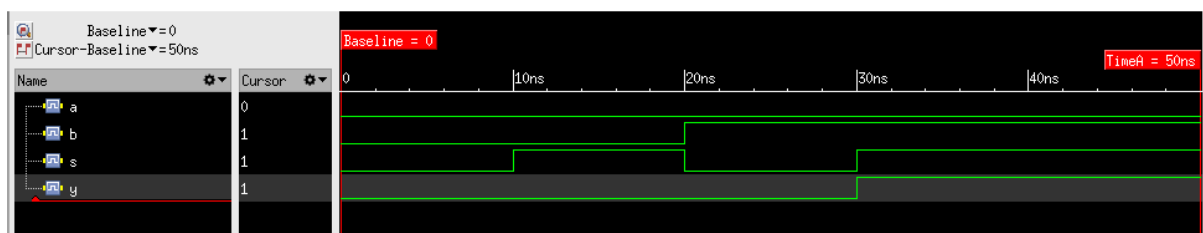
```
module stimulus( );  
    reg a,b,s;  
    wire y;  
  
    mux21 dut (  
        .a (a),  
        .b (b),  
        .s (s),  
        .y (y));  
  
    initial begin  
        a = 0; b = 0; s = 0; #10;  
        s = 1; #10;  
        b = 1; s = 0; #10;  
        s = 1; #10;  
    end  
  
    initial begin  
        $display("At Time: %d    input a=%d", $time, a);  
        $display("At Time: %d    input b=%d", $time, b);  
        $display("At Time: %d    input s=%d", $time, s);  
        $display("At Time: %d    output y=%d", $time, y);  
        #12;  
        $display("At Time: %d    input a=%d", $time, a);  
        $display("At Time: %d    input b=%d", $time, b);  
        $display("At Time: %d    input s=%d", $time, s);  
        $display("At Time: %d    output y=%d", $time, y);  
        #12;  
        $display("At Time: %d    input a=%d", $time, a);  
        $display("At Time: %d    input b=%d", $time, b);  
        $display("At Time: %d    input s=%d", $time, s);  
        $display("At Time: %d    output y=%d", $time, y);  
        #12;  
        $display("At Time: %d    input a=%d", $time, a);  
        $display("At Time: %d    input b=%d", $time, b);  
        $display("At Time: %d    input s=%d", $time, s);  
        $display("At Time: %d    output y=%d", $time, y);  
    end  
  
    initial begin  
        $dumpfile("dump.vcd");  
        $dumpvars;  
    end  
endmodule
```

Output:

```
[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim stimulus
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
ncsim: *W,DVEXACC2: some objects excluded from $dumpvars due to -access -R.
      File: ./mux2l_tb.v, line = 49, pos = 12
      Scope: stimulus
      Time: 0 FS + 0

At Time:          2 input a=0
At Time:          2 input b=0
At Time:          2 input s=0
At Time:          2 output y=0
At Time:         12 input a=0
At Time:         12 input b=0
At Time:         12 input s=1
At Time:         12 output y=0
At Time:         22 input a=0
At Time:         22 input b=1
At Time:         22 input s=0
At Time:         22 output y=0
At Time:         32 input a=0
At Time:         32 input b=1
At Time:         32 input s=1
At Time:         32 output y=1
Simulation complete via $finish(1) at time 50 NS + 0
./mux2l_tb.v:51      $finish;
ncsim> exit
```

Waveform



Flat Testbench with Self Checking MUX

```
module stimulus2( );
    reg a,b,s;
    wire y;

    mux21 dut (
        .a (a),
        .b (b),
        .s (s),
        .y (y));

    initial begin

        a = 0; b = 0; s = 0; #10;

        if (y !== 0) $display("000 failed.");
        else
            $display("000 Correct.");

        s = 1; #10;
        if (y !== 0) $display("001 failed.");
        else
            $display("001 Correct.");

        a = 1; s = 0; #10;
        if (y !== 1) $display("100 failed.");
    else
        $display("001 Correct.");
        a = 1; s = 0; #10;
        if (y !== 1) $display("100 failed.");
    else
        $display("100 Correct.");

        a = 1; s = 1; #10;
        if (y !== 0) $display("101 failed.");
        else
            $display("101 Correct.");
    end
endmodule
```

Output:

```
[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim stimulus2
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
000 Correct.
001 Correct.
001 Correct.
100 Correct.
101 Correct.
Simulation complete via $finish(1) at time 50 NS + 0
./mux21_tb2.v:35      $finish;
ncsim> exit
```

ALU module:

```
module alu (out,a,b,s,clk);
    input clk;
    input [7:0]a,b;
    input [3:0]s;
    output [15:0] out;
    reg [15:0] out;

    always@(posedge clk) begin
        case(s)
            4'b0000: out=a+b; //8-bit addition
            4'b0001: out=a-b; //8-bit subtraction
            4'b0010: out=a*b; //8-bit multiplication
            4'b0011: out=a/b; //8-bit division
            4'b0100: out=a%b; //8-bit modulo division
            4'b0101: out=a&b; //8-bit logical and
            4'b0110: out=a|b; //8-bit logical or
            4'b0111: out=!a; //8-bit logical negation
            4'b1000: out=~a; //8-bit bitwise negation
            4'b1001: out=a&b; //8-bit bitwise and
            4'b1010: out=a|b; //8-bit bitwise or
            4'b1011: out=a^b; //8-bit bitwise xor
            4'b1100: out=a<<1; //left shift
            4'b1101: out=a>>1; //right shift
            4'b1110: out=a+1; //increment
            4'b1111: out=a-1; //decrement
        endcase
    end
endmodule
```

Self Checking ALU in a flat testbench

```
module alustimulus;
    reg clk;
    reg [7:0] a,b;
    reg [3:0] s;
    wire [15:0] out;
    // Clock generation
    initial
        forever #5 clk = ~clk;
    // Module instantiation
    alu DUT( .clk (clk),
        .a (a),
        .b (b),
        .s (s),
        .out (out)
    );
    initial begin
        clk = 1'b0;
        a = $random;
        b = $random;
        s = 4'bx;
        #30
        s = 0;
        #10
        result_checker(s,a,b,out);
        #10
        s = 14;
        #10
        result_checker(s,a,b,out);
        #10
        s = $random;
        #10
        result_checker(s,a,b,out);
        $finish;
    end
end
```

```

task result_checker(input [3:0] s,input [7:0] a,b,input [15:0]resulted_out);
reg [15:0] expected_out;
begin
if(s==0) expected_out=a + b;
else if(s==1) expected_out=a - b;
else if(s==2) expected_out=a * b;
else if(s==3) expected_out=a / b;
else if(s==4) expected_out=a % b;
else if(s==5) expected_out=a && b;
else if(s==6) expected_out=a || b;
else if(s==7) expected_out=!a;
else if(s==8) expected_out=~a;
else if(s==9) expected_out=a & b;
else if(s==10) expected_out=a | b;
else if(s==11) expected_out=a ^ b;
else if(s==12) expected_out=a << 1;
else if(s==13) expected_out=a >> 1;
else if(s==14) expected_out=a + 1;
else if(s==15) expected_out=a - 1;

    if(resulted_out == expected_out)
        $display("Passed : a=%d, b=%d, s=%d, resulted_out=%d, expected_out=%d",
a,b,s,resulted_out,expected_out);
    else
        $display("Failed : a=%d, b=%d, s=%d, resulted_out=%d, expected_out=%d",
a,b,s,resulted_out,expected_out);
    end
endtask

// Waveform dumping
initial begin
    $dumpfile("aludump.vcd");
    $dumpvars;
end
endmodule

```

Output:

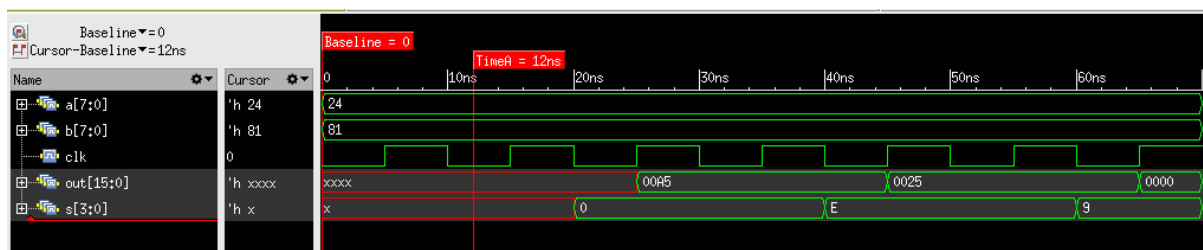
```

[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim alustimulus
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
ncsim: *W,DVEXACC2: some objects excluded from $dumpvars due to -access -R.
      File: ./alu_tb.v, line = 76, pos = 12
      Scope: alustimulus
      Time: 0 FS + 0

Passed : a= 36, b=129, s= 0, resulted_out= 165, expected_out= 165
Passed : a= 36, b=129, s=14, resulted_out=  37, expected_out=  37
Passed : a= 36, b=129, s= 9, resulted_out=   0, expected_out=   0
Simulation complete via $finish(1) at time 70 NS + 0
./alu_tb.v:36      $finish;
ncsim> exit

```

Waveform



Flat testbench with testvector in a separate testvector file for MUX

```
module testbench3( );
reg clk, reset;
reg a, b, s, yexpected;

wire y;
reg [31:0] vectornum, errors; //bookkeeping variables
reg [3:0] testvectors[10000:0];

mux21 dut (
.a (a),
.b (b),
.s (s),
.y (y));

always begin
clk = 1; #5; clk = 0; #5; //10ns period
end

initial begin
$readmemb("testvectormux.tv", testvectors); //read vec
vectornum = 0; errors = 0;
reset =1; #27; reset = 0; //apply reset wait
end

always@(posedge clk)
begin
#1; {a, b, s, yexpected} = testvectors[vectornum];
end

always@(negedge clk)
if (~reset) begin
if (y != yexpected) begin
$display("Error: inputs = %b", {a,b,s});
$display("Outputs = %b (%b exp)", y, yexpected);
errors = errors + 1;
end
vectornum = vectornum + 1;

if (testvectors[vectornum] === 4'bx) begin
$display("%d tests completed with %d errors", vectornum, errors);
$finish; end

end
endmodule
```

Output:

```
[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim stimulus3
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
          5 tests completed with          0 errors
Simulation complete via $finish(1) at time 75 NS + 0
./mux21_tb3.v:56          $finish;
ncsim> exit
```

Report Task 1

Write the verilog code of a 4/1 mux. Test your code with a testbench in both directed testing and self checking testing. In your report show the code of the circuit, its testbench and the output obtained from the simulator.

4to1 MUX module code

```
mux41.v X
1 module mux41(input a,b,c,d,
2             input [0:1]s,
3             output y);
4 assign y = ((~s[0] & ~s[1] & a)|(s[0] & ~s[1] & b)|(~s[0] & s[1] & c)|(s[0] & s[1] & d));
5 endmodule
```

Directed testing module code

```
*mux41_tb2.v X
1 module stimulus5( );
2 reg a,b,c,d;
3 reg [0:1]s; // inst. In dut
4 wire y;
5
6 mux41 dut ( .a (a), .b (b), .c (c), .d (d), .s (s), .y (y));
7 initial begin
8
9     a = 0; b = 0; c = 0; d = 0; s = 2'b00;
10    a = 1; #10;
11    s = 2'b01; #10;
12    b = 1; c = 1; #10;
13    a = 0; c = 0; s = 2'b11; #10;
14    d = 1; #10;
15
16    end
17
18 initial begin
19
20    #3;
21    $display("At Time: %d input a=%d", $time, a);
22    $display("At Time: %d input b=%d", $time, b);
23    $display("At Time: %d input c=%d", $time, c);
24    $display("At Time: %d input d=%d", $time, d);
25    $display("At Time: %d input s=%d", $time, s);
26    $display("At Time: %d output y=%d", $time, y);
27    #10;
28    $display("At Time: %d input a=%d", $time, a);
29    $display("At Time: %d input b=%d", $time, b);
30    $display("At Time: %d input c=%d", $time, c);
31    $display("At Time: %d input d=%d", $time, d);
32    $display("At Time: %d input s=%d", $time, s);
33    $display("At Time: %d output y=%d", $time, y);
34    #10;
35    $display("At Time: %d input a=%d", $time, a);
36    $display("At Time: %d input b=%d", $time, b);
37    $display("At Time: %d input c=%d", $time, c);
38    $display("At Time: %d input d=%d", $time, d);
39    $display("At Time: %d input s=%d", $time, s);
40    $display("At Time: %d output y=%d", $time, y);
41    #10;
```

```

42 $display("At Time: %d input a=%d", $time, a);
43 $display("At Time: %d input b=%d", $time, b);
44 $display("At Time: %d input c=%d", $time, c);
45 $display("At Time: %d input d=%d", $time, d);
46 $display("At Time: %d input s=%d", $time, s);
47 $display("At Time: %d output y=%d", $time, y);
48 #10;
49 $display("At Time: %d input a=%d", $time, a);
50 $display("At Time: %d input b=%d", $time, b);
51 $display("At Time: %d input c=%d", $time, c);
52 $display("At Time: %d input d=%d", $time, d);
53 $display("At Time: %d input s=%d", $time, s);
54 $display("At Time: %d output y=%d", $time, y);
55 end
56
57 // Waveform dumping
58 initial begin
59     $dumpfile("dump.vcd");
60     $dumpvars;
61     #60;
62     $finish;
63 end
64
65 endmodule

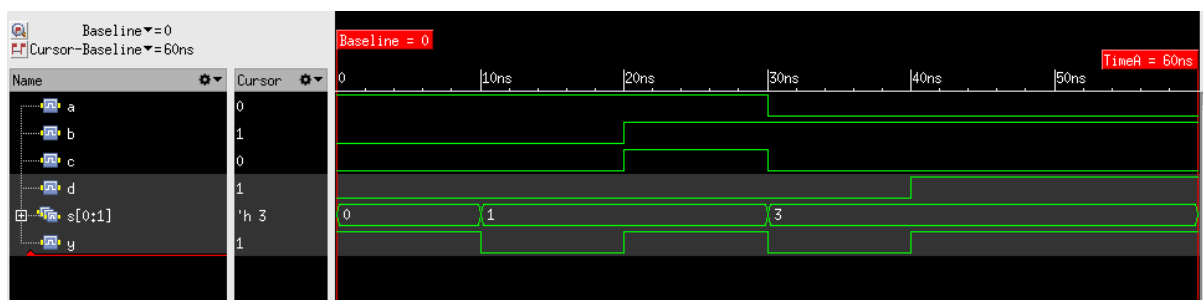
```


Directed testing module output

```
[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim stimulus5
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
ncsim: *W,DVEXACC2: some objects excluded from $dumpvars due to -access -R.
      File: ./mux4l_tb2.v, line = 61, pos = 12
      Scope: stimulus5
      Time: 0 FS + 0

At Time:          3 input a=1
At Time:          3 input b=0
At Time:          3 input c=0
At Time:          3 input d=0
At Time:          3 input s=0
At Time:          3 output y=1
At Time:         13 input a=1
At Time:         13 input b=0
At Time:         13 input c=0
At Time:         13 input d=0
At Time:         13 input s=1
At Time:         13 output y=0
At Time:         23 input a=1
At Time:         23 input b=1
At Time:         23 input c=1
At Time:         23 input d=0
At Time:         23 input s=1
At Time:         23 output y=1
At Time:         33 input a=0
At Time:         33 input b=1
At Time:         33 input c=0
At Time:         33 input d=0
At Time:         33 input s=3
At Time:         33 output y=0
At Time:         43 input a=0
At Time:         43 input b=1
At Time:         43 input c=0
At Time:         43 input d=1
At Time:         43 input s=3
At Time:         43 output y=1
Simulation complete via $finish(1) at time 60 NS + 0
./mux4l_tb2.v:63      $finish;
```

Waveform



Self-checking module code

```
mux41_tb.v X
1 module stimulus4( );
2 reg a,b,c,d;
3 reg [0:1]s; // inst. In dut
4 wire y;
5
6 mux41 dut ( .a (a), .b (b), .c (c), .d (d), .s (s), .y (y));
7
8 initial begin
9
10     a = 0; b = 0; c = 0; d = 0; s = 2'b00; #10;
11     if (y !== 0) $display("00000 failed.");
12     else $display("00000 Correct.");
13
14     a = 1; #10;
15     if (y !== 1) $display("10001 failed.");
16     else $display("10001 Correct.");
17
18     s = 2'b01; #10;
19     if (y !== 0) $display("10000 failed.");
20     else $display("10000 Correct.");
21
22     b = 1; c = 1; #10;
23     if (y !== 1) $display("11101 failed.");
24     else $display("11101 Correct.");
25
26     a = 0; s = 2'b10; #10;
27     if (y !== 1) $display("01101 failed.");
28     else $display("01101 Correct.");
29
30     c = 0; s = 2'b11; #10;
31     if (y !== 0) $display("01000 failed.");
32     else $display("01000 Correct.");
33
34     d = 1; #10;
35     if (y !== 1) $display("01011 failed.");
36     else $display("01011 Correct.");
37     $finish;
38 end
39
40 endmodule
```

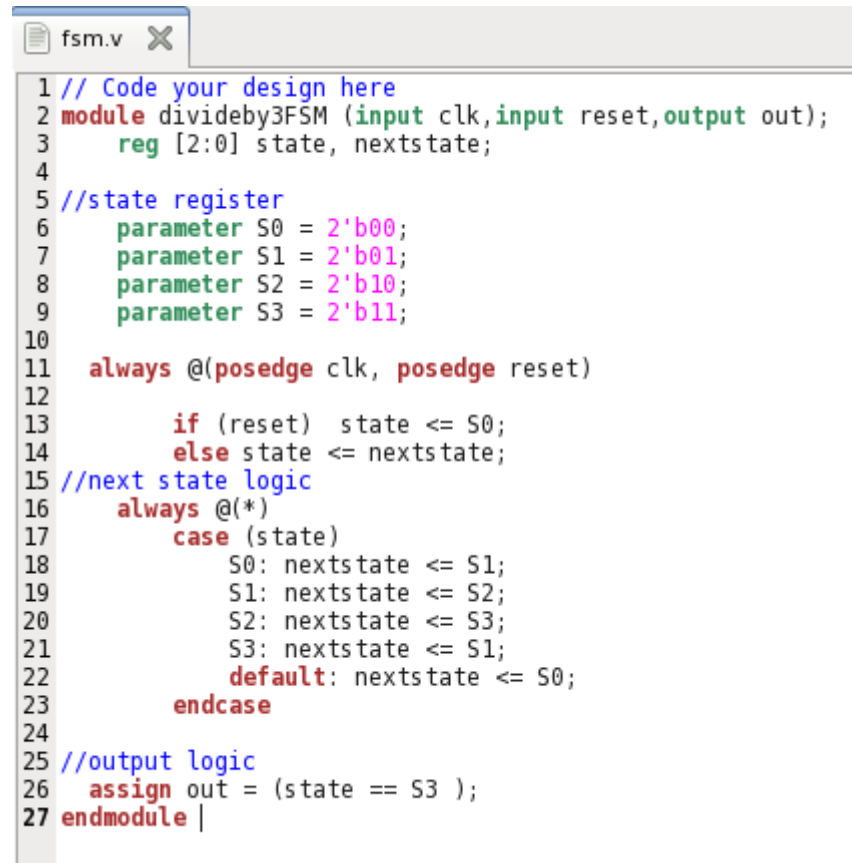
Self-checking module output

```
[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim stimulus4
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
00000 Correct.
10001 Correct.
10000 Correct.
11101 Correct.
01101 Correct.
01000 Correct.
01011 Correct.
Simulation complete via $finish(1) at time 70 NS + 0
./mux41_tb.v:45      $finish;
ncsim> exit
```

Report task 2

Write a test bench for the FSM of divider by 3 counter. Simulate the test bench and verify the functionality of your FSM in directed and self checking verification. In your report show the code of the testbench and the output obtained from the simulator.

Divider by 3 FSM code

A screenshot of a Verilog code editor window titled 'fsm.v'. The code is a Verilog module named 'divideby3FSM' with inputs 'clk' and 'reset', and output 'out'. It defines a 3-bit state register 'state' and a 'nextstate' variable. The code includes a state register block that resets to state S0 on a reset signal or updates to 'nextstate' on a clock edge. A next state logic block uses a case statement to determine the next state based on the current state: S0 goes to S1, S1 to S2, S2 to S3, and S3 back to S1. The output 'out' is assigned the value 1 when the state is S3. The code is as follows:

```
1 // Code your design here
2 module divideby3FSM (input clk,input reset,output out);
3     reg [2:0] state, nextstate;
4
5 //state register
6     parameter S0 = 2'b00;
7     parameter S1 = 2'b01;
8     parameter S2 = 2'b10;
9     parameter S3 = 2'b11;
10
11     always @(posedge clk, posedge reset)
12
13         if (reset) state <= S0;
14         else state <= nextstate;
15 //next state logic
16     always @(*)
17         case (state)
18             S0: nextstate <= S1;
19             S1: nextstate <= S2;
20             S2: nextstate <= S3;
21             S3: nextstate <= S1;
22             default: nextstate <= S0;
23         endcase
24
25 //output logic
26     assign out = (state == S3 );
27 endmodule |
```

```

fsm_tb.v X
1 // Code your testbench here
2 // or browse Examples
3 module FSMstimulus;
4     reg clk;
5     reg reset;
6     reg[0:1] s;
7     wire out;
8     reg expected_out;
9
10 // Clock generation
11     initial forever #5 clk = ~(clk);
12
13 // Module instantiation
14     divideby3FSM dut (
15         .clk (clk),
16         .reset (reset),
17         .out (out));
18
19     initial begin
20         clk = 1'b0;
21         reset= 1'b1;
22         #2;
23         reset= 1'b0;
24         s = 0;
25         expected_out = 0;
26
27     end
28
29
30     always @(posedge clk, reset)
31     begin
32         #1;
33         if(reset)
34             begin
35                 s=0;
36                 expected_out = 0;
37             end
38         else if (clk)
39             begin
40                 s=s+1;
41                 $display ("s=%d",s);
42                 if(s==3)
43                     begin
44                         expected_out = 1;
45                         s = 0;
46                     end
47                 else
48                     expected_out = 0;
49             end
50         end
51         if(out == expected_out)
52             $display("Passed : resulted_out=%d, expected_out=%d ",out,expected_out);
53         else
54             $display("Failed : resulted_out=%d, expected_out=%d ",out,expected_out);
55     end
56
57 // Waveform dumping
58     initial begin
59         $dumpfile("FSM.vcd");
60         $dumpvars;
61         #100;
62         $finish;
63     end
64 endmodule

```

Divider by 3 FSM output

```
[vlsi05@CadenceServer3 ~/cds_digital_THA]$ ncsim FSMstimulus
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
ncsim: *W,DVEXACC2: some objects excluded from $dumpvars due to -access -R.
      File: ./fsm_tb.v, line = 63, pos = 10
      Scope: FSMstimulus
      Time: 0 FS + 0

Passed : resulted_out=0, expected_out=0
s=1
Passed : resulted_out=0, expected_out=0
s=2
Passed : resulted_out=0, expected_out=0
s=3
Passed : resulted_out=1, expected_out=1
s=1
Passed : resulted_out=0, expected_out=0
s=2
Passed : resulted_out=0, expected_out=0
s=3
Passed : resulted_out=1, expected_out=1
s=1
Passed : resulted_out=0, expected_out=0
s=2
Passed : resulted_out=0, expected_out=0
s=3
Passed : resulted_out=1, expected_out=1
s=1
Passed : resulted_out=0, expected_out=0
Simulation complete via $finish(1) at time 100 NS + 0
./fsm_tb.v:65          $finish;
ncsim> exit
```

Divider by 3 FSM wave

