VHDL Week 7

Ex1

1 Encoder 8x3

Verilog

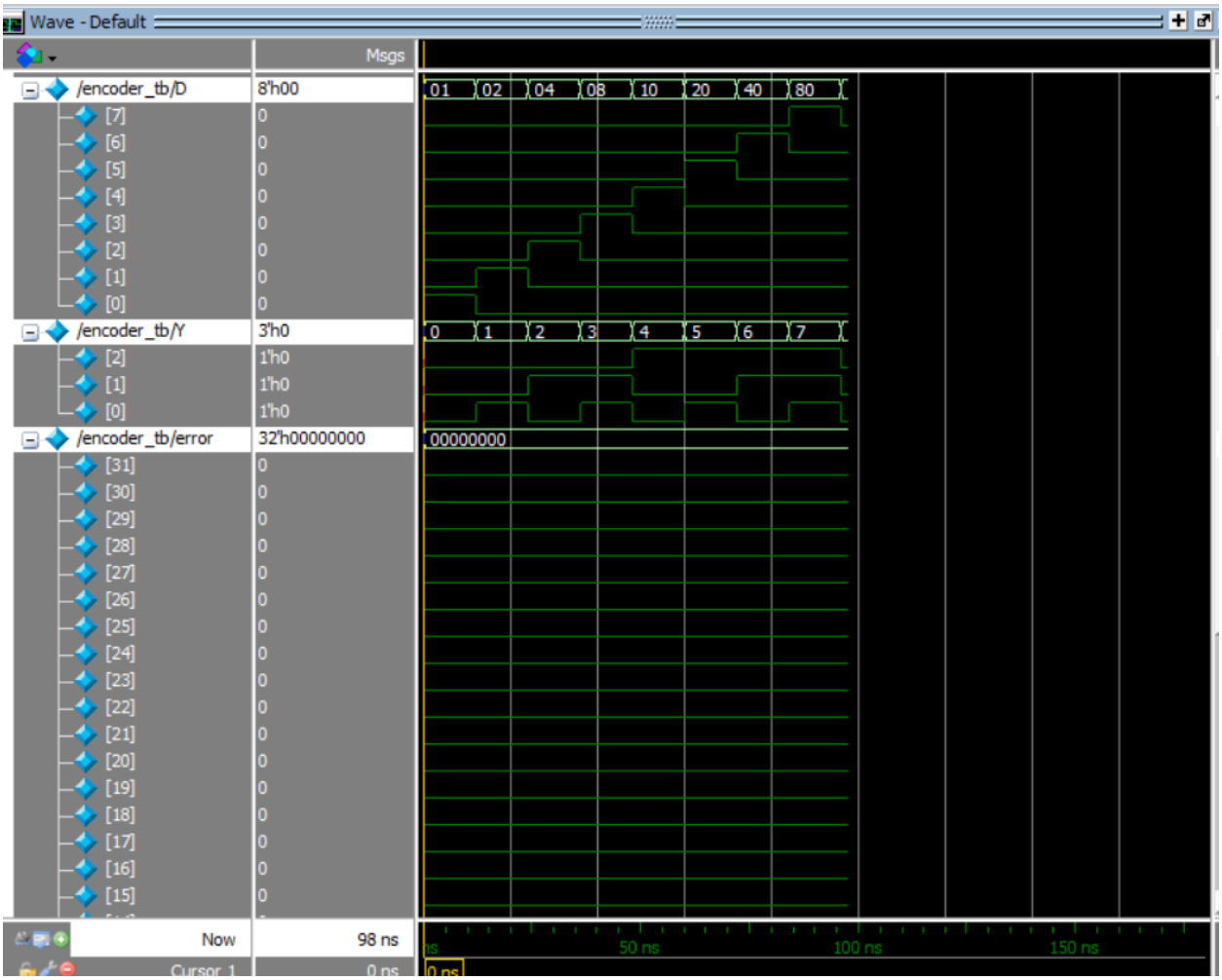
```
module encoder8_3 (
  input      [7:0] D,
  output reg [2:0] Y
);

  integer i;
  always @(D) begin
    Y = 0;
    for (i = 0; i < 8; i = i + 1) begin
      if (D[i] == 1)
        Y = i;
    end
  end

endmodule : encoder8_3
```

Testbench

F:/VHDL/ET5080E - 20221/HW7/BCD/encoder_tb.v (/encoder_tb) - Default

```
module encoder_tb;

  reg  [7:0] D;
  wire [2:0] Y;
  integer   error ;

  encoder8_3 encoder (
    .D(D),
    .Y(Y)
  );

  initial begin
    error = 0;
    D = 4'b0001;
    #2;
      if (Y != $clog2(D)) begin
        error = error + 1;
      end
      $display($stime, ":\t\tD = %b", D, "\t\tY = %b", Y, "\t\terror = %d", error);
    repeat(8) begin
      #10;
      D = D << 1;
      #2;
      if (Y != $clog2(D)) begin
        error = error + 1;
      end
      $display($stime, ":\t\tD = %b", D, "\t\tY = %b", Y, "\t\terror = %d", error);
    end
    $stop;
  end

endmodule : encoder_tb
```

Wave



Script

```
add wave -position insertpoint sim:/encoder_tb/*
VSIM 8> run -all
#          2:   D = 00000001    Y = 000         error =          0
#         14:   D = 00000010    Y = 001         error =          0
#         26:   D = 00000100    Y = 010         error =          0
#         38:   D = 00001000    Y = 011         error =          0
#         50:   D = 00010000    Y = 100         error =          0
#         62:   D = 00100000    Y = 101         error =          0
#         74:   D = 01000000    Y = 110         error =          0
#         86:   D = 10000000    Y = 111         error =          0
#         98:   D = 00000000    Y = 000         error =          0
# ** Note: $stop    : F:/VHDL/ET5080E - 20221/HW7/BCD/encoder_tb.v(30)
#    Time: 98 ns  Iteration: 0  Instance: /encoder_tb
# Break in Module encoder_tb at F:/VHDL/ET5080E - 20221/HW7/BCD/encoder
_tb.v line 30

VSIM 9>
```

Report

Testplan | Design | DesUnits

encoder_tb

ModelSim Coverage Report

Number of tests run:	1
Passed:	1
Warning:	0
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

List of Design Units included in report...

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
encoder_tb	31.31%	53.16%
encoder	30.85%	81.10%

Coverage Summary by Type:

Total Coverage:					31.31%	53.16%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	17	15	2	1	88.23%	88.23%
Branches	6	4	2	1	66.66%	66.66%
FEC Conditions	3	1	2	1	33.33%	33.33%
Toggles	172	42	130	1	24.41%	24.41%

Report generated by ModelSim (ver. 10.7) on Tuesday 29 November 2022 22:51:37 with command line:
vcover report -html encoder_cov.ucdb

2 Majority 4 bit

Verilog

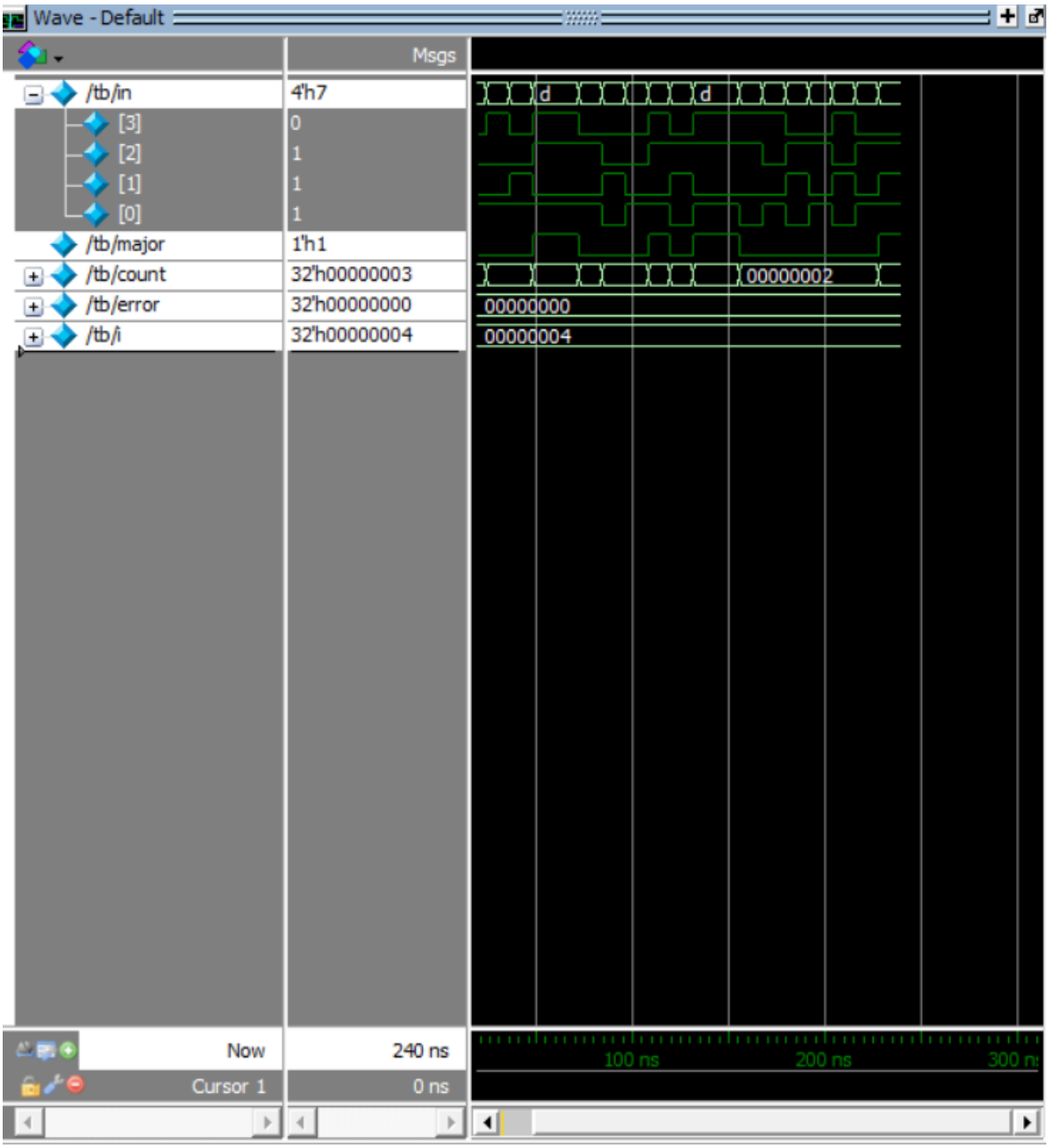
F:/VHDL/ET5080E - 20221/HW 7/Majority/majority.v (/tb/dut) - Default

```
module majority_4bit (
  input [3:0] in   ,
  output reg  major
);

  reg pre_major;
  always @(in) begin
    pre_major = (in[0] & in[1]) |
                (in[1] & in[2]) |
                (in[0] & in[2]);
  end

  always @(in, pre_major) begin
    major = (in[2] & in[1] & in[0] == 1) ? 1 : (pre_major & in[3]);
  end

endmodule : majority_4bit
```

Testbench

```
module tb;

  reg  [3:0] in   ;
  wire       major;
  integer    count;
  integer    error;
  integer    i    ;

  majority_4bit dut(
    .in   (in   ),
    .major(major)
  );

  initial begin
    error = 0;
    repeat(20) begin
      in = $random();
      count = 0;
      for (i = 0; i < 4; i = i + 1)
        count = count + in[i];

      #2;
      if (((count > 2) && (major == 0)) || ((count < 3) && (major == 1))) begin
        error = error + 1;
      end
      $display($stime, ":\t\tin = %b", in, "\t\tmajor = %b", major, "\t\terror = %d", error);

      #10;
    end
    $stop;
  end

endmodule : tb
```

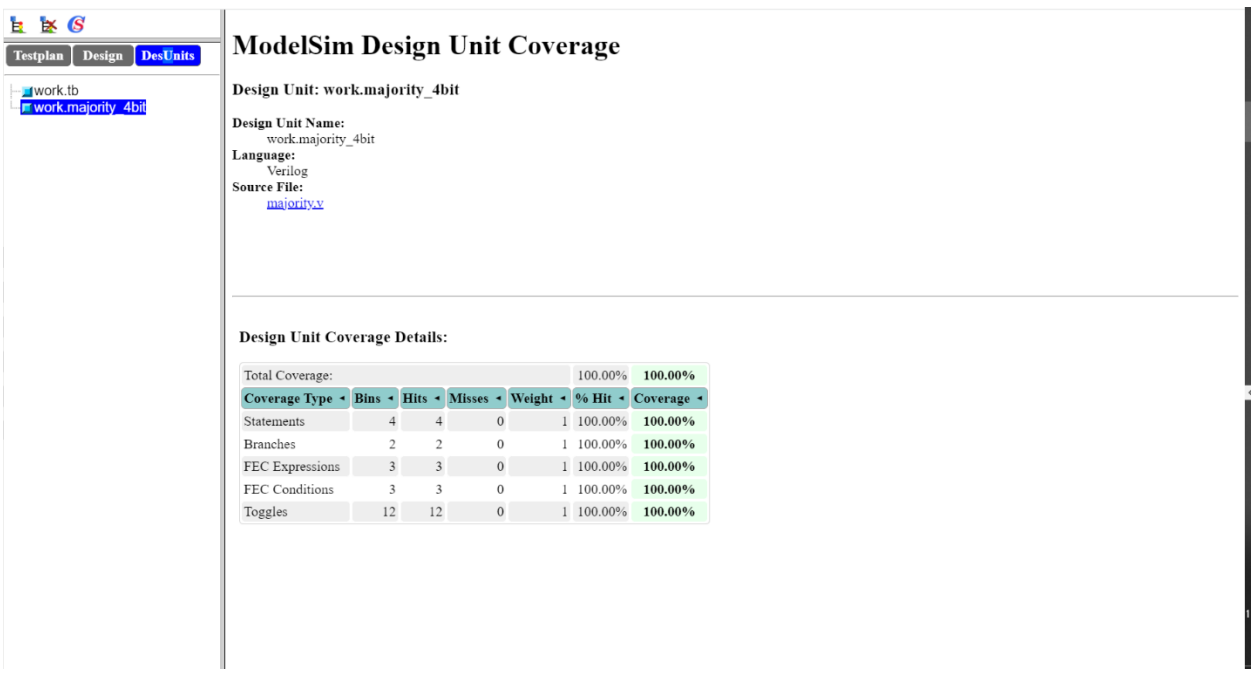
Wave


Wave - Default
Msgs
/tb/in
4'h7
[3]
0
[2]
1
[1]
1
[0]
1
/tb/major
1'h1
/tb/count
32'h00000003
/tb/error
32'h00000000
/tb/i
32'h00000004
d
d
00000002
00000000
00000004
Now
240 ns
Cursor 1
0 ns
100 ns
200 ns

Script

```
VSIM 10> run -all
#          2:   in = 0100       major = 0       error =          0
#         14:   in = 0001       major = 0       error =          0
#         26:   in = 1001       major = 0       error =          0
#         38:   in = 0011       major = 0       error =          0
#         50:   in = 1101       major = 1       error =          0
#         62:   in = 1101       major = 1       error =          0
#         74:   in = 0101       major = 0       error =          0
#         86:   in = 0010       major = 0       error =          0
#         98:   in = 0001       major = 0       error =          0
#        110:   in = 1101       major = 1       error =          0
#        122:   in = 0110       major = 0       error =          0
#        134:   in = 1101       major = 1       error =          0
#        146:   in = 1101       major = 1       error =          0
#        158:   in = 1100       major = 0       error =          0
#        170:   in = 1001       major = 0       error =          0
#        182:   in = 0110       major = 0       error =          0
#        194:   in = 0101       major = 0       error =          0
#        206:   in = 1010       major = 0       error =          0
#        218:   in = 0101       major = 0       error =          0
#        230:   in = 0111       major = 1       error =          0
```

Report



Total Coverage:					100.00%	100.00%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	4	4	0	1	100.00%	100.00%
Branches	2	2	0	1	100.00%	100.00%
FEC Expressions	3	3	0	1	100.00%	100.00%
FEC Conditions	3	3	0	1	100.00%	100.00%
Toggles	12	12	0	1	100.00%	100.00%

3 BCD 7-segment

Verilog

```
module encoder_7segment (
  input      [3:0] in ,
  output reg [6:0] out
);


  always @(in) begin
```

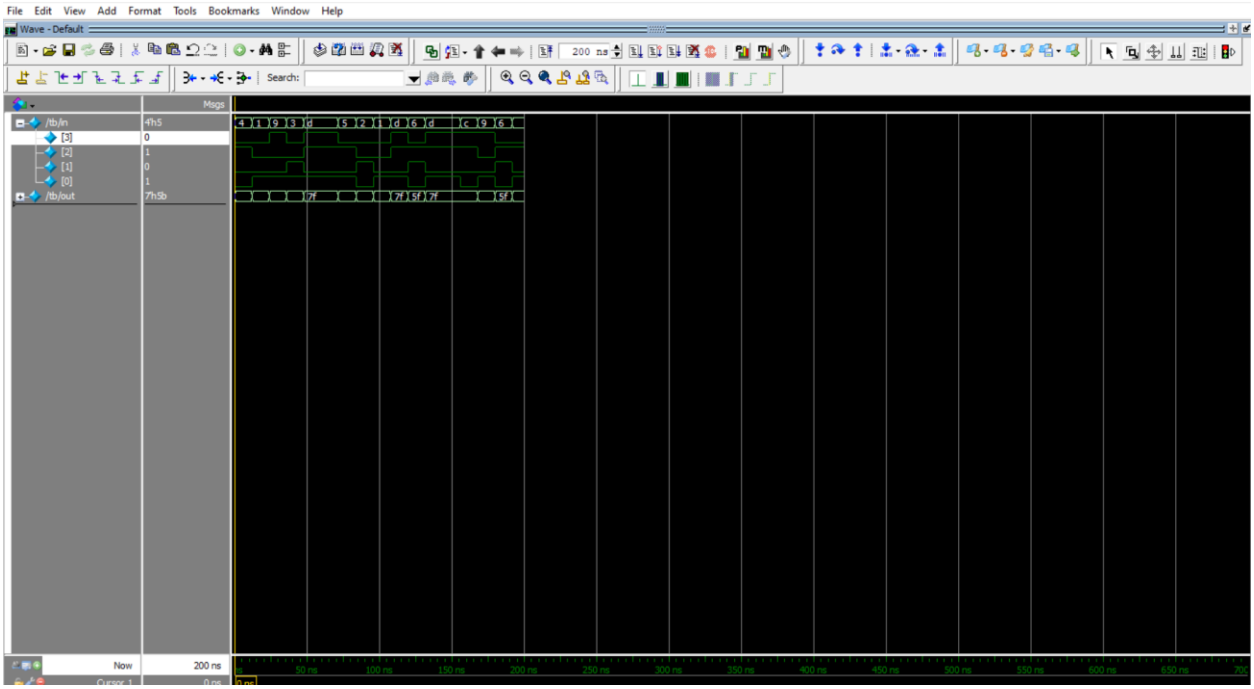
```
case (in)
  4'b0000: begin
    out = 7'b1111110;
  end

  4'b0001: begin
    out = 7'b0110000;
  end

  4'b0010: begin
    out = 7'b1101101;
  end

  4'b0011: begin
    out = 7'b1111001;
  end

  4'b0100: begin
    out = 7'b0110011;
  end

  4'b0101: begin
    out = 7'b1011011;
  end

  4'b0110: begin
    out = 7'b1011111;
  end

  4'b0111: begin
```

```
      out = 7'b1110000;
    end

    4'b1000: begin
      out = 7'b1111111;
    end

    4'b1001: begin
      out = 7'b1111011;
    end

    default : begin
      out = 7'b1111111;
    end
  endcase
end

endmodule : encoder_7segment
```

Testbench

```
module tb;

  reg  [3:0] in ;
  wire [6:0] out;

  encoder_7segment encoder (
    .in (in ),
    .out(out)
  );

  initial begin
    repeat(20) begin
      in = $random();

      #2;
      $display($stime, ":\t\tin = %b", in, "\t\tout = %b", out);

      #10;
    end
    $stop;
  end

endmodule : tb
```

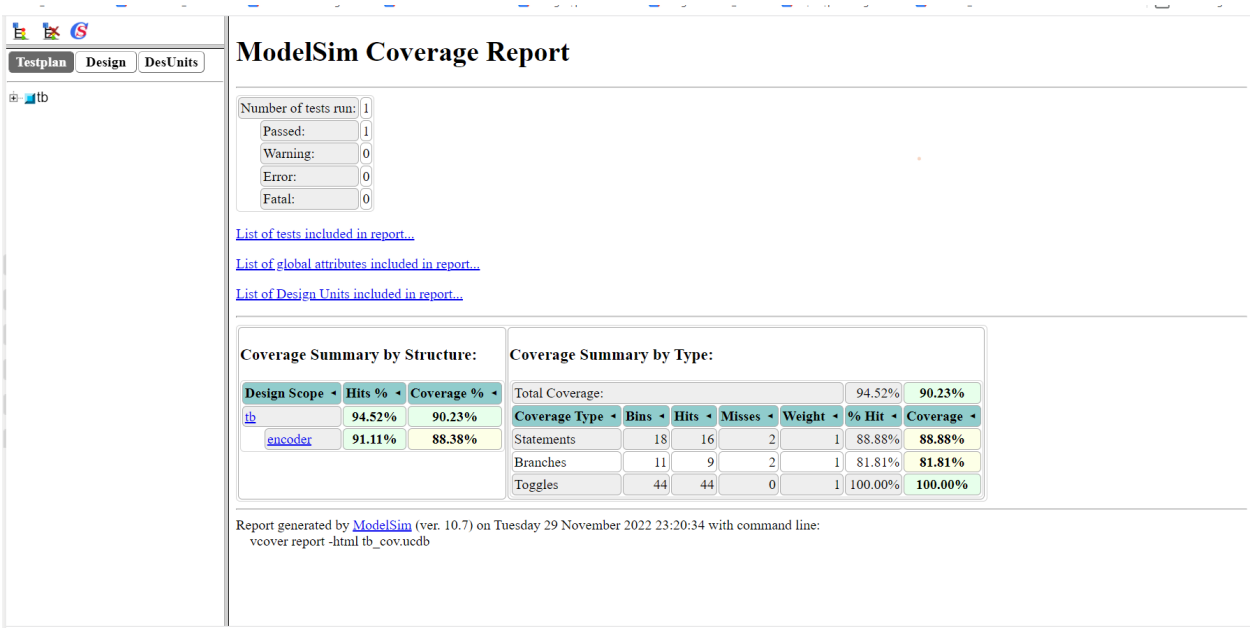
Wave



Script

```
VSIM 8> run
#          2:   in = 0100       out = 0110011
#         14:   in = 0001       out = 0110000
#         26:   in = 1001       out = 1111011
#         38:   in = 0011       out = 1111001
#         50:   in = 1101       out = 1111111
#         62:   in = 1101       out = 1111111
#         74:   in = 0101       out = 1011011
#         86:   in = 0010       out = 1101101
#         98:   in = 0001       out = 0110000
#        110:   in = 1101       out = 1111111
#        122:   in = 0110       out = 1011111
#        134:   in = 1101       out = 1111111
#        146:   in = 1101       out = 1111111
#        158:   in = 1100       out = 1111111
#        170:   in = 1001       out = 1111011
#        182:   in = 0110       out = 1011111
#        194:   in = 0101       out = 1011011
```

Report



Testplan Design DesUnits

tb

ModelSim Coverage Report

Number of tests run:	1
Passed:	1
Warning:	0
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

List of Design Units included in report...

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
tb	94.52%	90.23%
encoder	91.11%	88.38%

Coverage Summary by Type:

Total Coverage:					94.52%	90.23%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	18	16	2	1	88.88%	88.88%
Branches	11	9	2	1	81.81%	81.81%
Toggles	44	44	0	1	100.00%	100.00%

Report generated by ModelSim (ver. 10.7) on Tuesday 29 November 2022 23:20:34 with command line:
vcover report -html tb_cov.ucdb

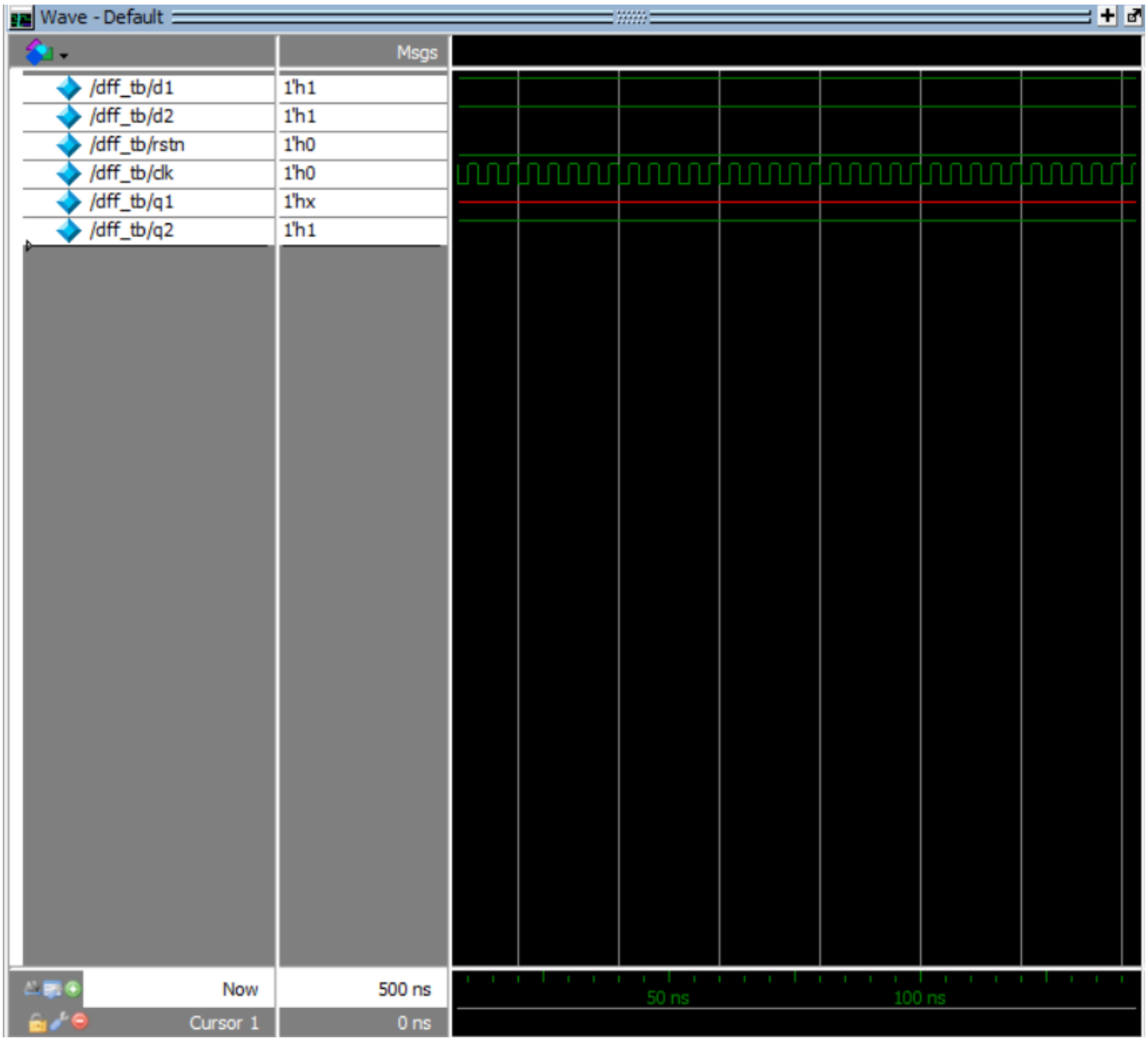
4

DFF

Asynchronous

```
module dff(
input d1,d2,
input rstn,
input clk,
output reg q1,q2);
always @(posedge clk or negedge rstn)
if(!rstn)
q2=d2;
else
q2=0;
endmodule
```

```
module dff_tb();
reg d1,d2,rstn,clk;
wire q1,q2;
dff a1(d1,d2,rstn,clk,q1,q2);
initial begin
$monitor("Value of d1 = %b,d2 = %b, rstn=%b,clk = %b,q1=5b,q2=%b",d1,d2,rstn,clk,q1,q2);
clk = 0;
d1=0;d2=0;rstn=0;
#2 d1=1; d2=1; rstn=0;
#2 d1=1; d2=1; rstn=0;
#2 d1=1; d2=1; rstn=0;
end
always #2 clk=~clk;
endmodule
```

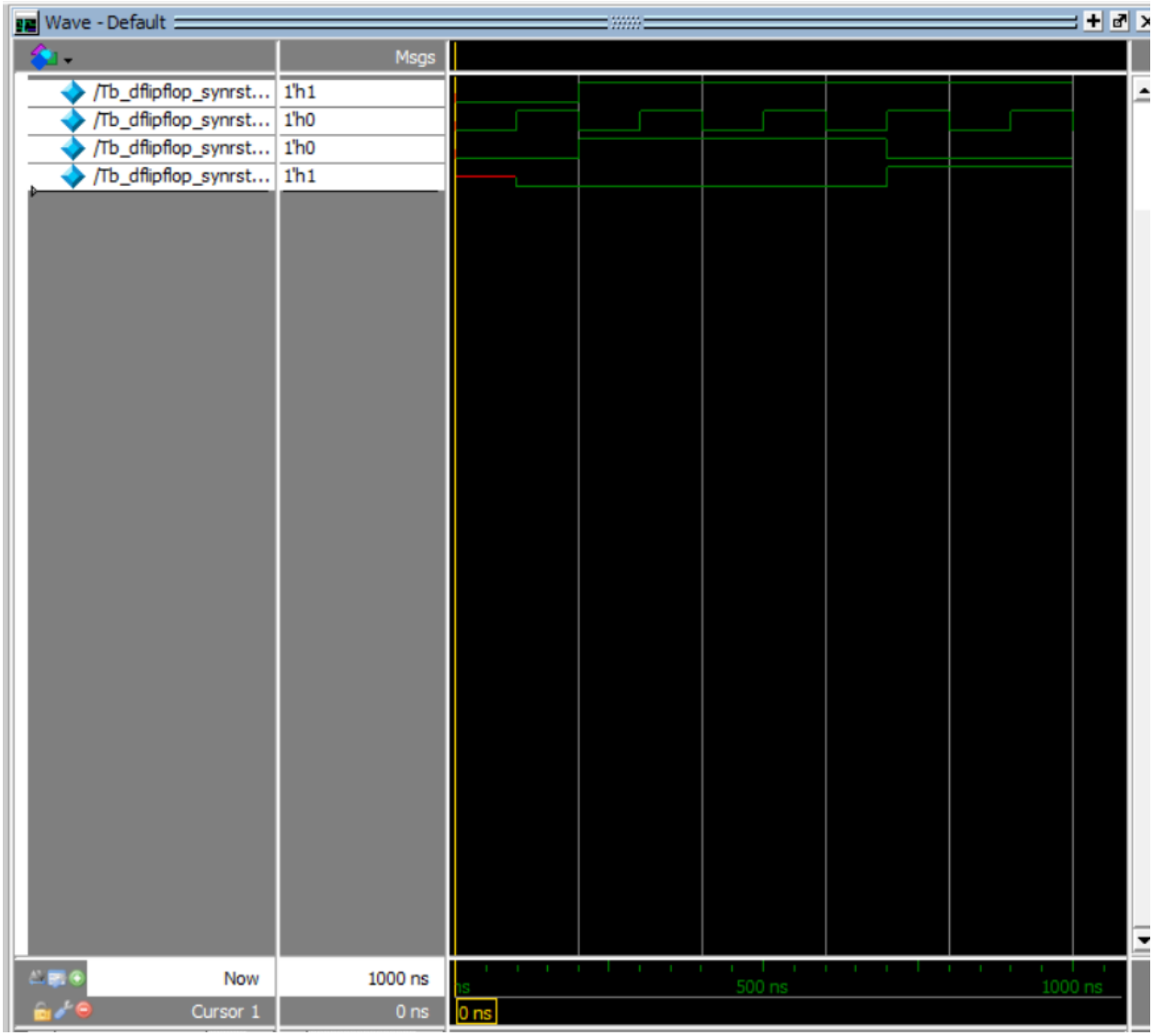


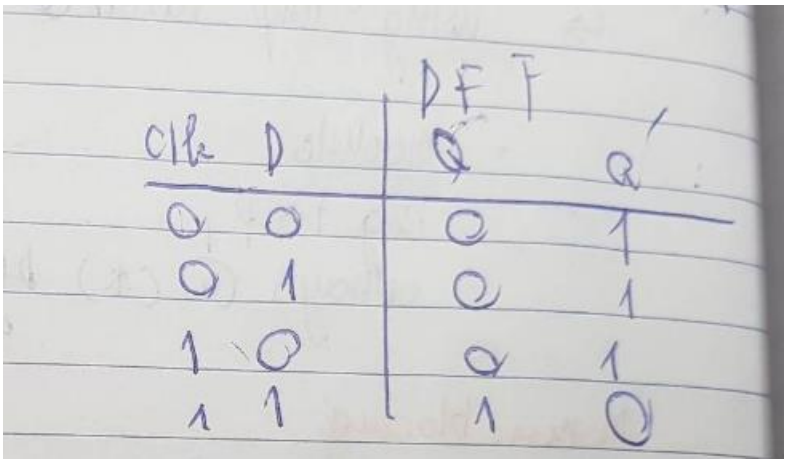
Synchronous

```
module d_flipflop_synrst(data_in,data_out,clock,reset);
input data_in;
input clock,reset;

output reg data_out;

always@(posedge clock)
begin
if(reset)
data_out<=1'd0;
else
data_out<=data_in;
end
endmodule
```

F:/VHDL/ET5080E - 20221/HW7/Flip Flop/DFF/Synchronous/dff_tb.v (/Tb_dflipflop_synrst) - Default

```
module Tb_dflipflop_synrst();
reg data_in;
reg clock,reset;
wire data_out;

d_flipflop_synrst UUT(.data_in(data_in),
.data_out(data_out),
.clock(clock),
.reset(reset));
initial begin
// Initiliase Input Stimulus
data_in = 0;
clock = 0;
reset=0;
end

always #100 clock=~clock;

//Stimulus
initial
begin
#200 data_in = 1'b1;
reset = 1'b1;
#200 data_in = 1'b1;
reset = 1'b1;

#300 data_in = 1'b1;
reset=1'b0;
#600 data_in = 1'b0;
#500 data_in = 1'b1;
#200 data_in = 1'b0;
#400 $stop;
end

endmodule
```





DFF

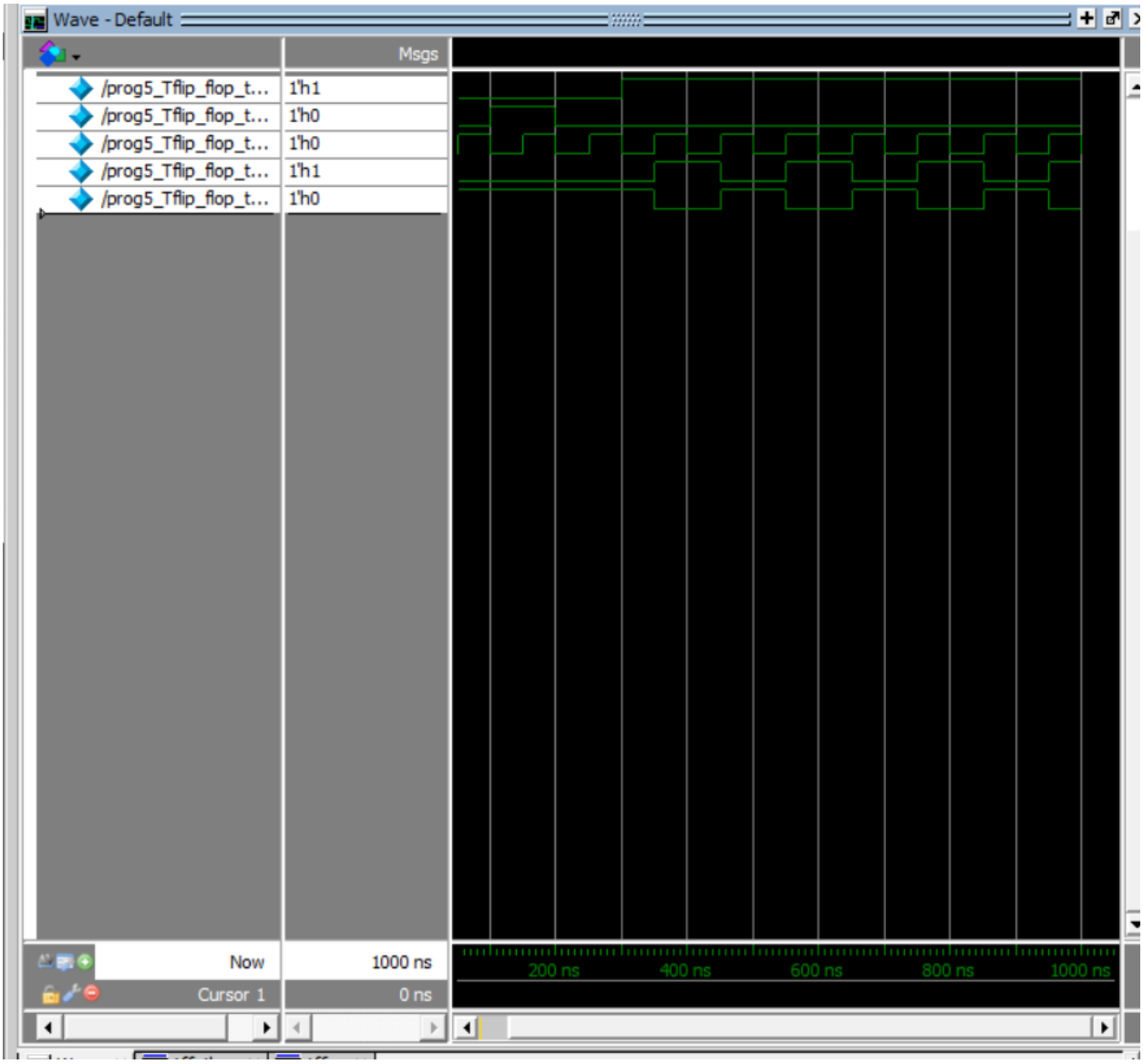
Clk	D	Q	Q'
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

TFF

Asynchronous

```
module prog5_Tflip_flop(q, qb, t, rst, clk);
input t, rst, clk;
output q, qb;
reg q, qb;
initial
q = 0;
always @(posedge clk or negedge rst)
begin
        case({rst, t})
        2'b10 : q = 0;
        2'b00 : q = q;
        2'b01 : q = ~q;
        endcase
qb = ~q;
end
endmodule
```

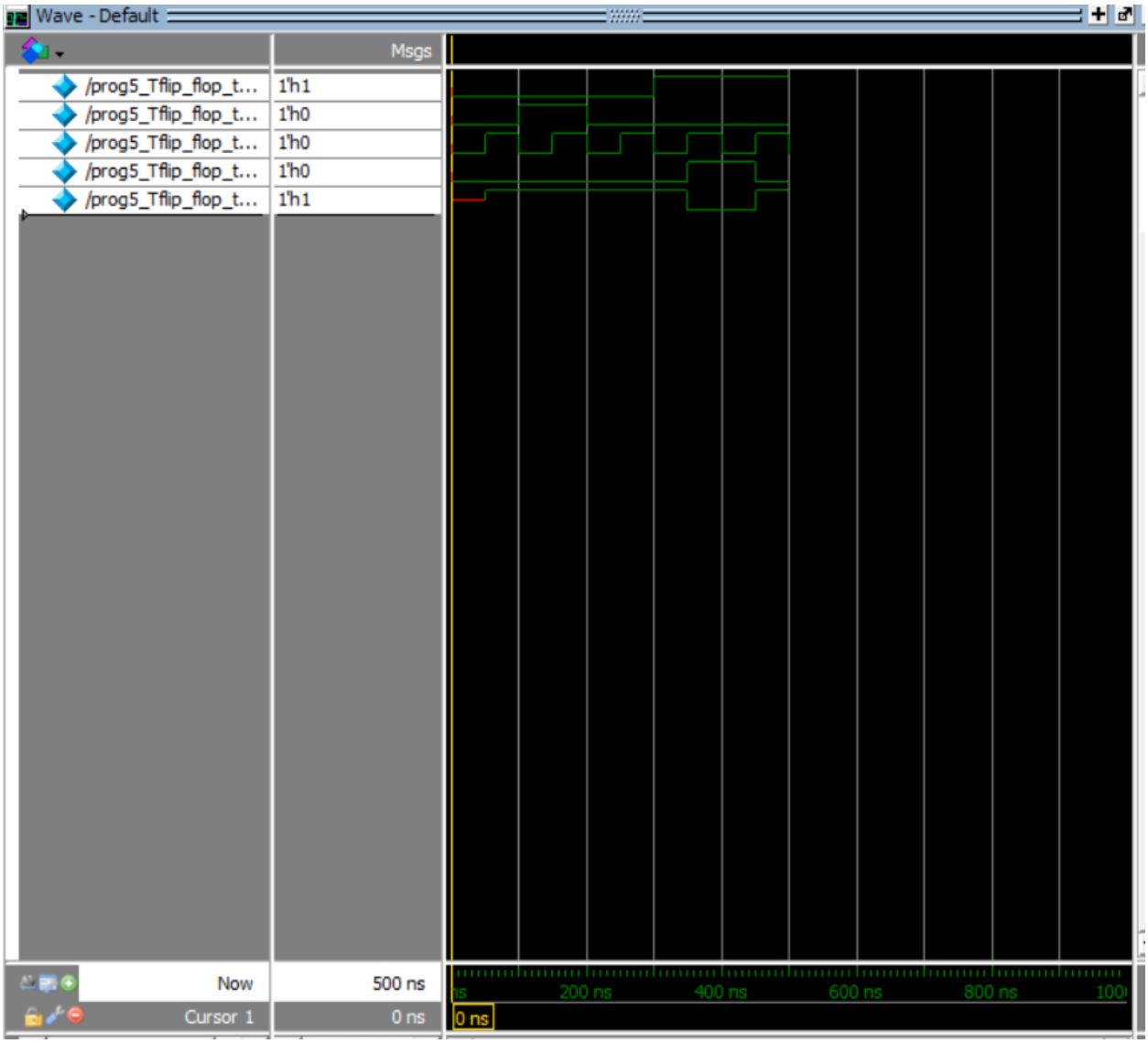
```
Ln#
1
2   module prog5_Tflip_flop_tb_v;
3
4           // Inputs
5           reg t;
6           reg rst;
7           reg clk;
8
9           // Outputs
10          wire q;
11          wire qb;
12
13          // Instantiate the Unit Under Test (UUT)
14          prog5_Tflip_flop uut (
15                  .q(q),
16                  .qb(qb),
17                  .t(t),
18                  .rst(rst),
19                  .clk(clk)
20          );
21
22          initial begin
23                  // Initialize Inputs
24                  t = 0;
25                  rst = 0;
26                  clk = 0;
27
28                  // Wait 100 ns for global reset to finish
29                  #100;
30                  t = 0;
31                  rst = 1;
32
33                  #100;
34                  t = 0;
35                  rst = 0;
36
37                  #100;
38                  t = 1;
39                  rst = 0;
```

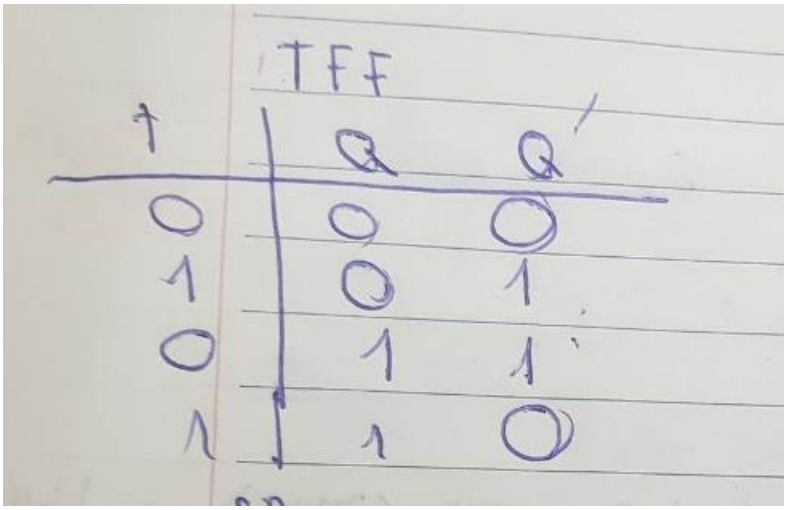


Synchronous

```
module prog5_Tflip_flop(q, qb, t, rst, clk);
input t, rst, clk;
output q, qb;
reg q, qb;
initial
q = 0;
always @(posedge clk)
begin
        case({rst, t})
        2'b10 : q = 0;
        2'b00 : q = q;
        2'b01 : q = ~q;
        endcase
qb = ~q;
end
endmodule
```

```
12
13          // Instantiate the Unit Under Test (UUT)
14          prog5_Tflip_flop uut (
15                  .q(q),
16                  .qb(qb),
17                  .t(t),
18                  .rst(rst),
19                  .clk(clk)
20          );
21
22          initial begin
23                  // Initialize Inputs
24                  t = 0;
25                  rst = 0;
26                  clk = 0;
27
28                  // Wait 100 ns for global reset to finish
29                  #100;
30                  t = 0;
31                  rst = 1;
32
33                  #100;
34                  t = 0;
35                  rst = 0;
36
37                  #100;
38                  t = 1;
39                  rst = 0;
40
41                  #100;
42                  t = 1;
43                  rst = 0;
44
45                  // Add stimulus here
46
47          end
48                  always #50 clk = ~clk;
49
50  endmodule
```





TFF

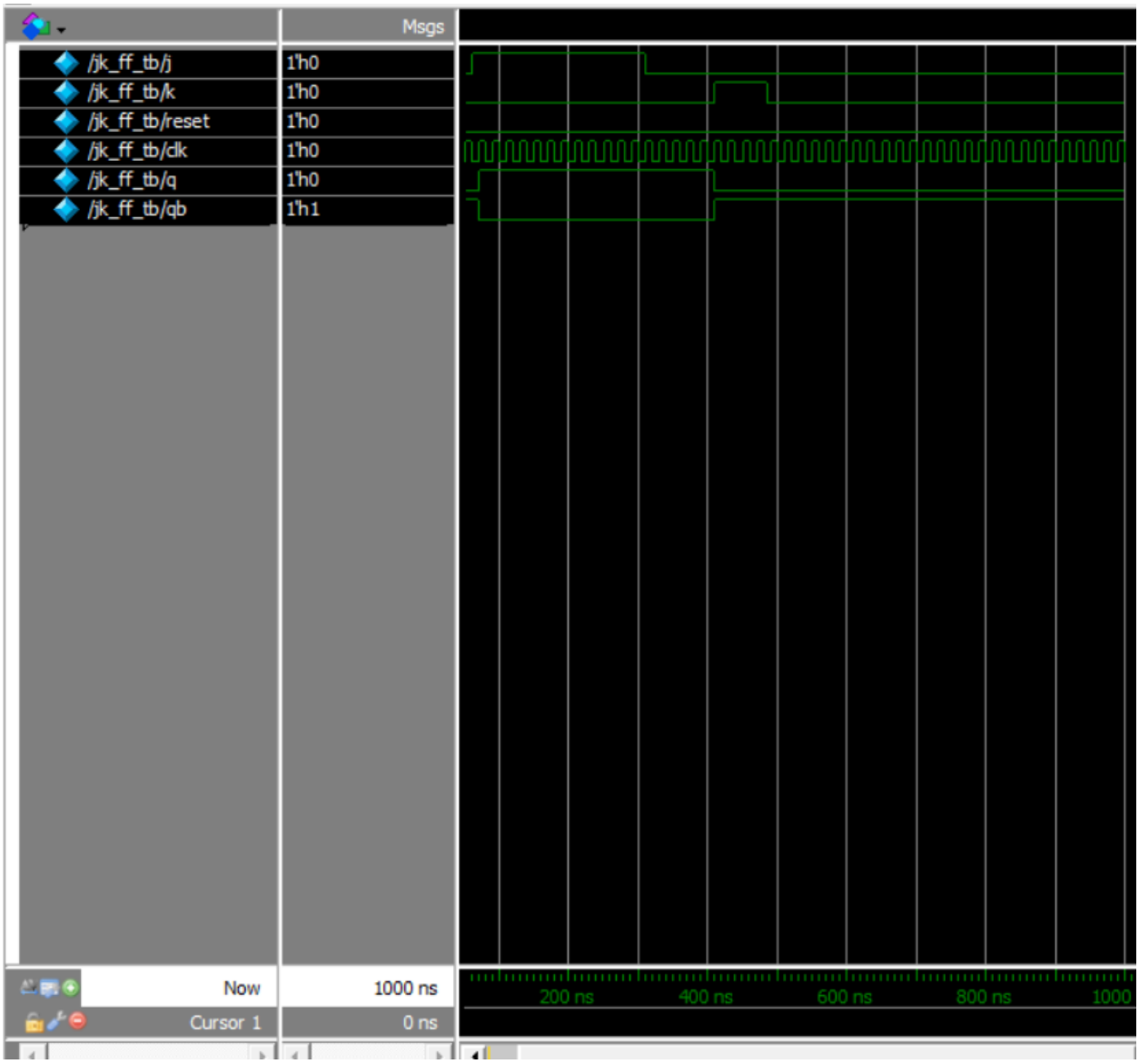
T	Q	Q'
0	0	0
1	0	1
0	1	1
1	1	0

JKFF

Asynchronous

```
module jk_ff( j, k  , clk , reset, q ,qb );
input j,k;
input clk, reset ;
output q,qb;
reg q,qb;
reg [1:0]jk;

always @ ( posedge clk or posedge reset)
begin
jk={j,k};
if (reset)
begin
q = 1'b0;
qb = ~q;
end
else
begin
case (jk)
2'd0 : q = q;
2'd1 : q = 1'b0;
2'd2 : q = 1'b1;
2'd3 : q = ~q;
endcase
qb = ~q;
end
end
endmodule
```

```
module jk_ff_tb;

    // Inputs
    reg j;
    reg k;
    reg reset;
    reg clk;

    // Outputs
    wire q;
    wire qb;

    // Instantiate the Unit Under Test (UUT)
    jk_ff uut (
        .j(j),
        .k(k),
        .reset(reset),
        .clk(clk),
        .q(q),
        .qb(qb)
    );

    initial begin
        // Initialize Inputs
        j = 1'b0;
        k = 1'b0;
        reset = 1'b1;
        clk = 1'b0;

        // Wait 100 ns for global reset to finish
        #10 reset = 1'b0;

        // Add stimulus here

    end
    always #10 clk = ~clk;
        initial begin
        #60 j = 1'b1;
        #50 k = 1'b0;
        #200 j = 1'b0;
        #100 k = 1'b1;
        #30 j = 1'b0;
        #45 k = 1'b0;
        end

endmodule
```

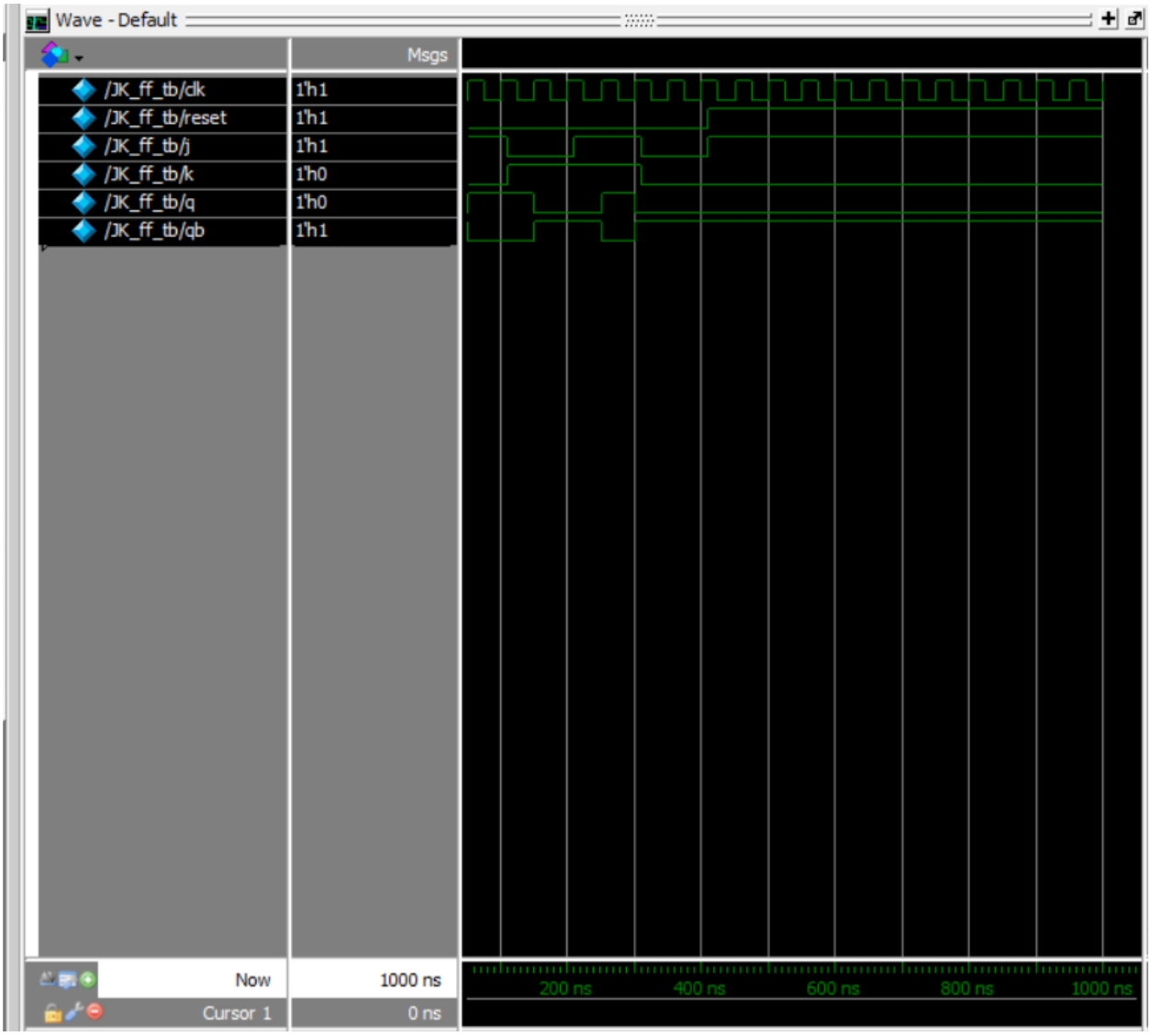


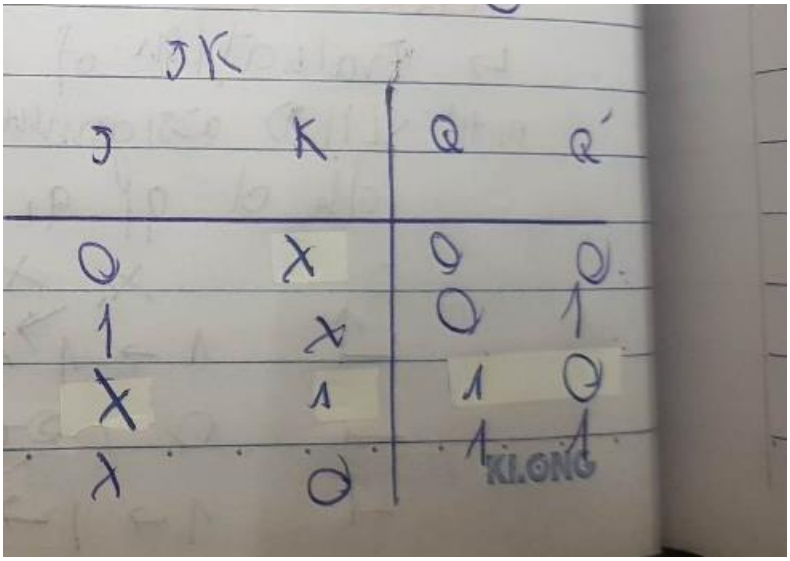
Synchronous

F:/VHDL/ET5080E - 20221/HW7/Flip Flop/JKFF/syn/jk.v (/JK_ff_tb/jkflipflop) - Default

```
module JK_ff(j,k,clk,reset,q,q_bar);

input j,k,clk,reset;
output q,q_bar;

wire j,k,clk,reset;
reg q,q_bar;

always @(posedge clk) begin

if(reset) begin
q=1'b0;
q_bar=1'b1;

end else begin

case({j,k})
{1'b0,1'b0}: begin q=q;q_bar=q_bar; end
{1'b0,1'b1}: begin q=1'b0;q_bar=1'b1; end
{1'b1,1'b0}: begin q=1'b1;q_bar=1'b0; end
{1'b1,1'b1}: begin q=~q; q_bar=~q_bar; end
endcase

end

end

endmodule
```

```
module JK_ff_tb;

reg clk;
reg reset;
reg j,k;

wire q;
wire qb;

JK_ff jkflipflop( .clk(clk), .reset(reset), .j(j), .k(k), .q(q), .q_bar(qb) );

initial begin
$monitor(clk,j,k,q,qb,reset);

j = 1'b0;
k = 1'b0;
reset = 1;
clk=1;

#10
reset=0;
j=1'b1;
k=1'b0;

#100
reset=0;
j=1'b0;
k=1'b1;

#100
reset=0;
j=1'b1;
k=1'b1;

#100
reset=0;
j=1'b0;
k=1'b0;
```





JK

J	K	Q	Q'
0	X	0	0
1	X	0	1
X	1	1	0
X	0	1	1

SRFF

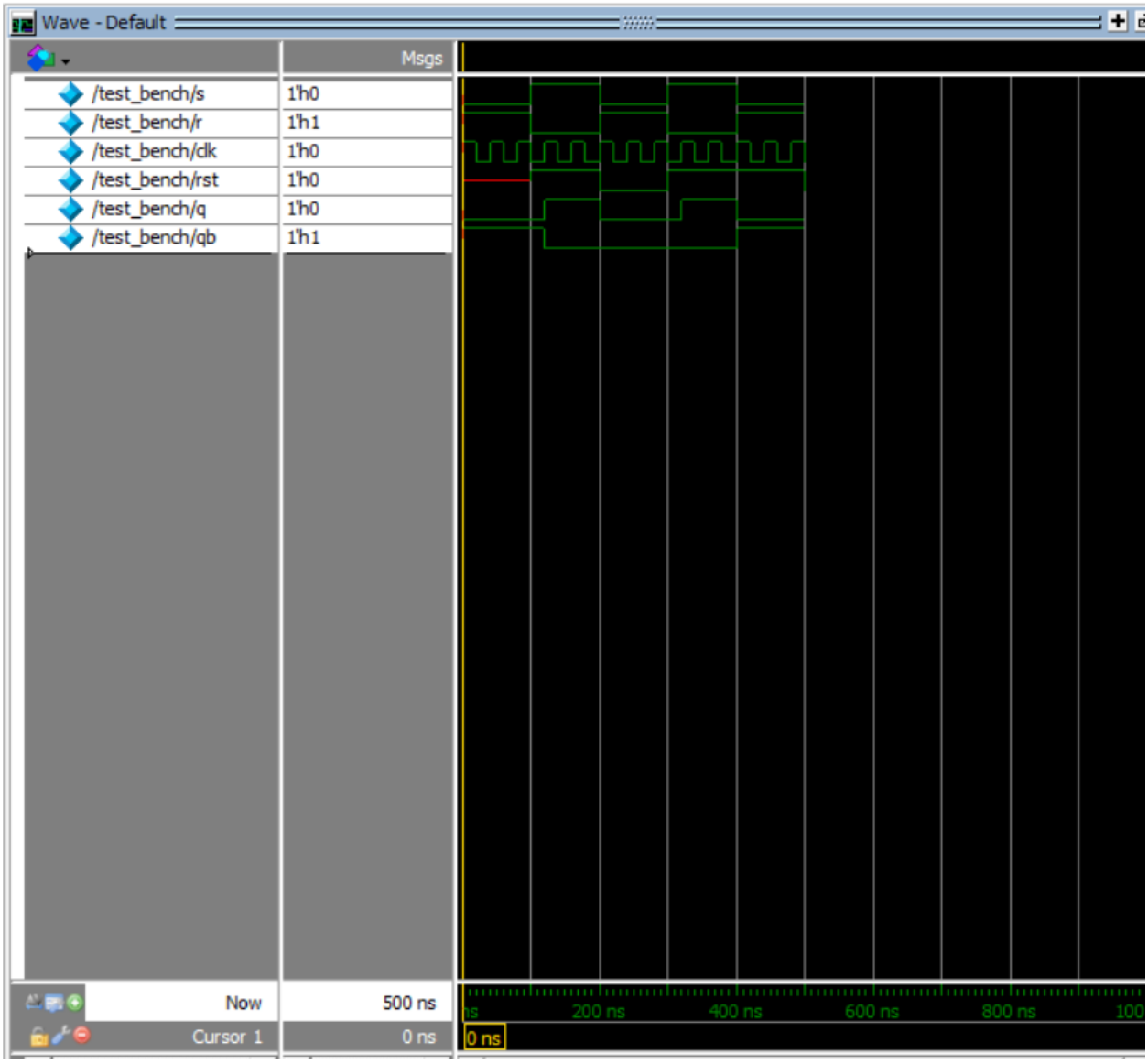
Asynchronous

F:/VHDL/ET5080E - 20221/HW7/Flip Flop/SRFF/Syn/sr.v (/test_bench/GG1) - Default

```
module sr_activelow_(q,qb,s,r,clk,rst);
input s,r,clk,rst;
output reg q,qb;
always@(posedge clk or negedge rst)
begin
if (!rst)
q <=0;
else
if (s==1 & r==0) begin
q <= 1;
qb <=0;
end
else
if (s==0 && r==1) begin
q <=0;
qb <=1;
end
end
endmodule
```

F:/VHDL/ET5080E - 20221/HW7/Flip Flop/SRFF/Syn/sr_tb.v (/test_bench) - Default

```
module test_bench(

    );
    reg s,r,clk,rst;
    wire q,qb;
    sr_activelow_ GG1(q,qb,s,r,clk,rst);
    initial
    begin
        s=0; r=1; clk=1;
   #100 rst=1; s=1; r=0;
   #100 rst=0; s=0; r=1;
   #100 rst=1; s=1; r=0;
   #100 rst=1; s=0; r=1;
   #100 rst=0; s=0; r=1;
   end
    always #20 clk <= ~clk;
endmodule
```

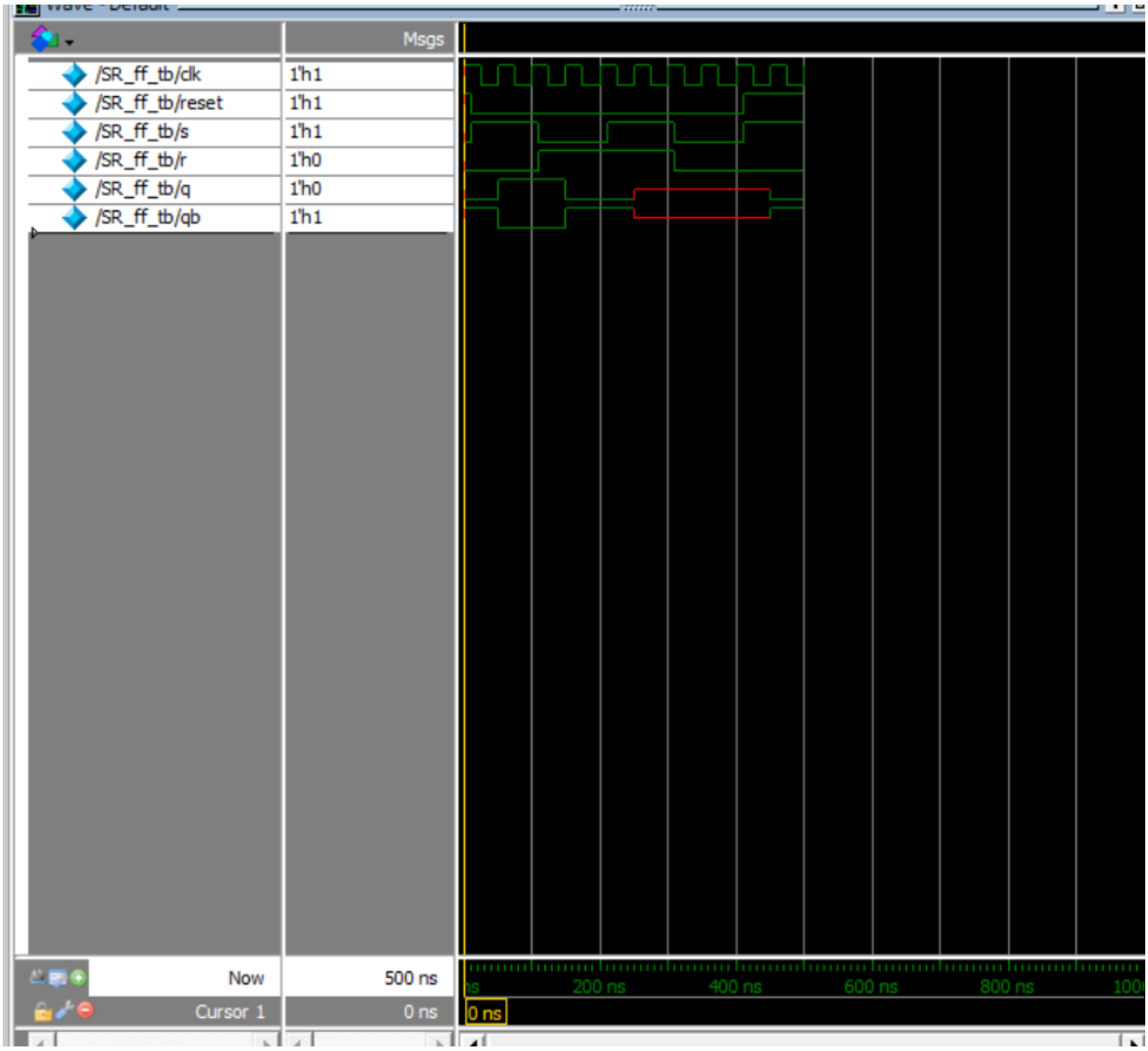


Synchronous

F:/VHDL/ET5080E - 20221/HW7/Flip Flop/SRFF/Syn/sr.v (/SR_ff_tb/srflipflop) - Default

```
module SR_ff(s,r,clk,reset,q,q_bar);

input s,r,clk,reset;

output q,q_bar;

wire s,r,clk;
reg q,q_bar;

always @(posedge clk) begin

if (reset) begin
q=1'b0;
q_bar=1'b1;

end else begin

case({s,r})
{1'b0,1'b0}: begin q=q;q_bar=q_bar; end
{1'b0,1'b1}: begin q=1'b0;q_bar=1'b1; end
{1'b1,1'b0}: begin q=1'b1;q_bar=1'b0; end
{1'b1,1'b1}: begin q=1'bx; q_bar=1'bx; end
endcase

end

end
endmodule
```

```
reg clk;
reg reset;
reg s,r;

wire q;
wire qb;

SR_ff srflipflop( .clk(clk), .reset(reset), .s(s), .r(r), .q(q), .q_bar(qb) );

initial begin
$monitor(clk,s,r,q,qb,reset);

s = 1'b0;
r = 1'b0;
reset = 1;
clk=1;

#10
reset=0;
s=1'b1;
r=1'b0;

#100
reset=0;
s=1'b0;
r=1'b1;

#100
reset=0;
s=1'b1;
r=1'b1;

#100
reset=0;
s=1'b0;
r=1'b0;

#100
reset=1;
s=1'b1;
r=1'b0;

end
always #25 clk <= ~clk;
```


Msgs
/SR_ff_tb/clk
/SR_ff_tb/reset
/SR_ff_tb/s
/SR_ff_tb/r
/SR_ff_tb/q
/SR_ff_tb/qb
1'h1
1'h1
1'h1
1'h0
1'h0
1'h1
Now
500 ns
Cursor 1
0 ns
200 ns
400 ns
600 ns
800 ns
0 ns

SR

S	R	Q	Q'
0	0	0	1
0	1	0	1
1	0	1	0
1	1	∞	∞

Ex2

Always and initial blocks

Similarity: All starts at t =0;

Differences:

Initial

Syntax for initial statement cam be indicated as below

```
<initial_statement>

::= initial <statement>
```

The instruction executes only once in the whole process. It begins its execution at the start of the simulation at the time t =0. If there exists more than 1 initial block, then all the initial blocks are executed concurrently

Always

Always statement executes repeatedly, although the execution starts at time t = 0 and keep on executing all the simulation time. It works like an infinite loop. It is generally used to model a functionality that ‘s continuously repeated

Syntax:

```
always [timing_control] procedural_statement
```

To control the always statement we can use the trigger depending on what you are choosing to control