Week 3

Ex1:

Initial and always block

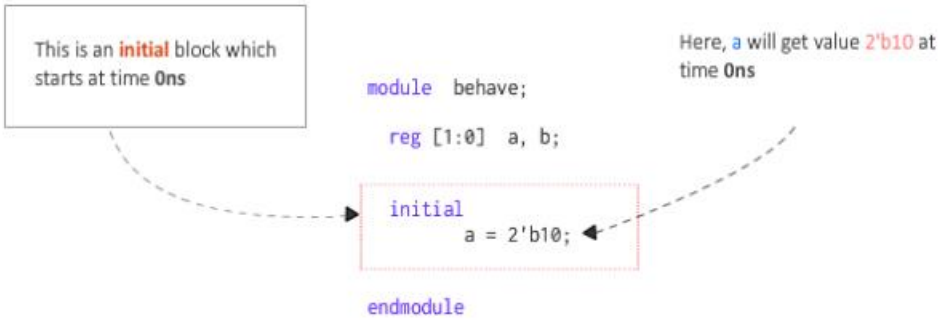
Those are procedural block, all the statements inside initial anad always blocks are executed sequentially.

Those can have more than 1 initial or always block in our design. Those should be used when more than one variable are assigned in initial or always blocks.

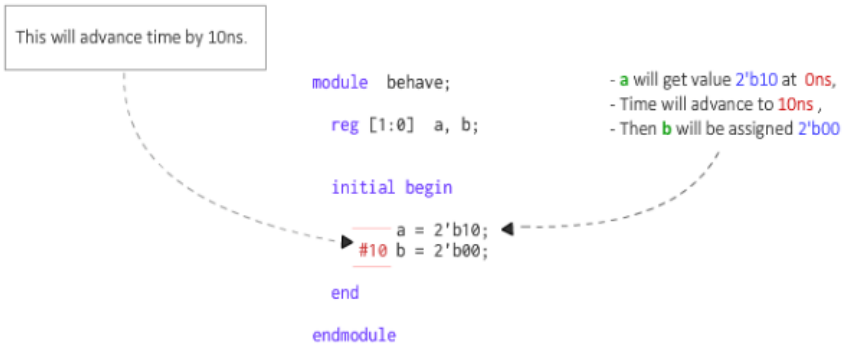
Initial block

An initial block is not synthesizable and hence cannot be converted into a hardware schematic with digital elements. Hence initial blocks do not serve much purpose than being used in simulations as these are primarily used to initialize variables and drive design ports with specific values.

An initial block started at the beginning of a simulation at time 0 unit. This block will be executed only once during the entire simulation. Execution of an initial block finishes once all the statements within the block are executed.

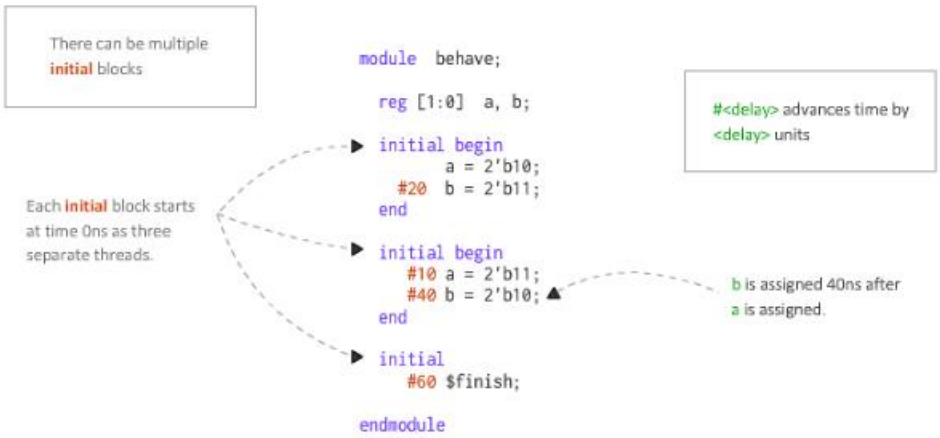


Supposing that we add a delay to each gate:



This means that after 10 time units from execution from prev statement, a is assigned 1st with the given value and then after 10 time units, b is assigned to 0.

There are no limits to the number of initial blocks that can be defined in the module. In this example we have 3 initial blocks at the same time.



Initial is not synthesizable for ASIC(synthesizable in some FPGA)

Always blocks

Always block is one of the procedural blocks in Verilog. Statements inside an always block are executed sequentially. The always block is executed at some particular event as this event is defined by a sensitivity list

Sensitivity list is the expression that defines when the always block should be executed and is specified after the @ operator within the parentheses(). This list may contain either 1 or a group of signals whose value change will execute the always block. Particularly, all statements inside the always block get executed whenever the value of signals a or b change.

This block can be used to realize combinational or sequential elements. A sequential element like flip flop becomes active when it’s provided with a clock and reset. Similarly, a combinational block becomes active when one of its input values change. These hardware blocks are all working concurrently independent of each other. The connection between each is what determines the flow of data. To model this behavior, an always block is made as a continuous process that gets triggered and performs some action when a signal within the sensitivity list becomes active.

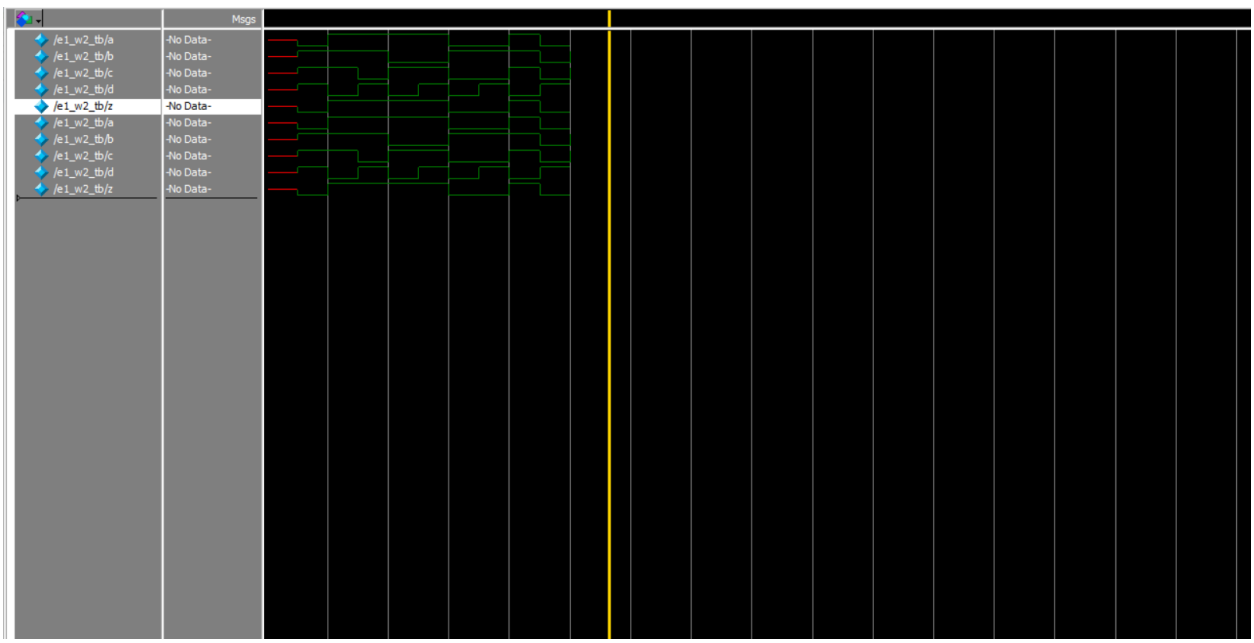
Ex2

K-map ex

Testbench

```
module e1_w2_tb();
reg a,b,c,d;
wire z;
e1_w2 dut(
 .a(a),
 .b(b),
 .c(c),
 .d(d),
 .z(z)
);
initial begin
repeat(20) begin
#10;
a = $random();
b = $random();
c = $random();
d =  $random();
end
$stop;
end
endmodule:e1_w2_tb
```

Wave

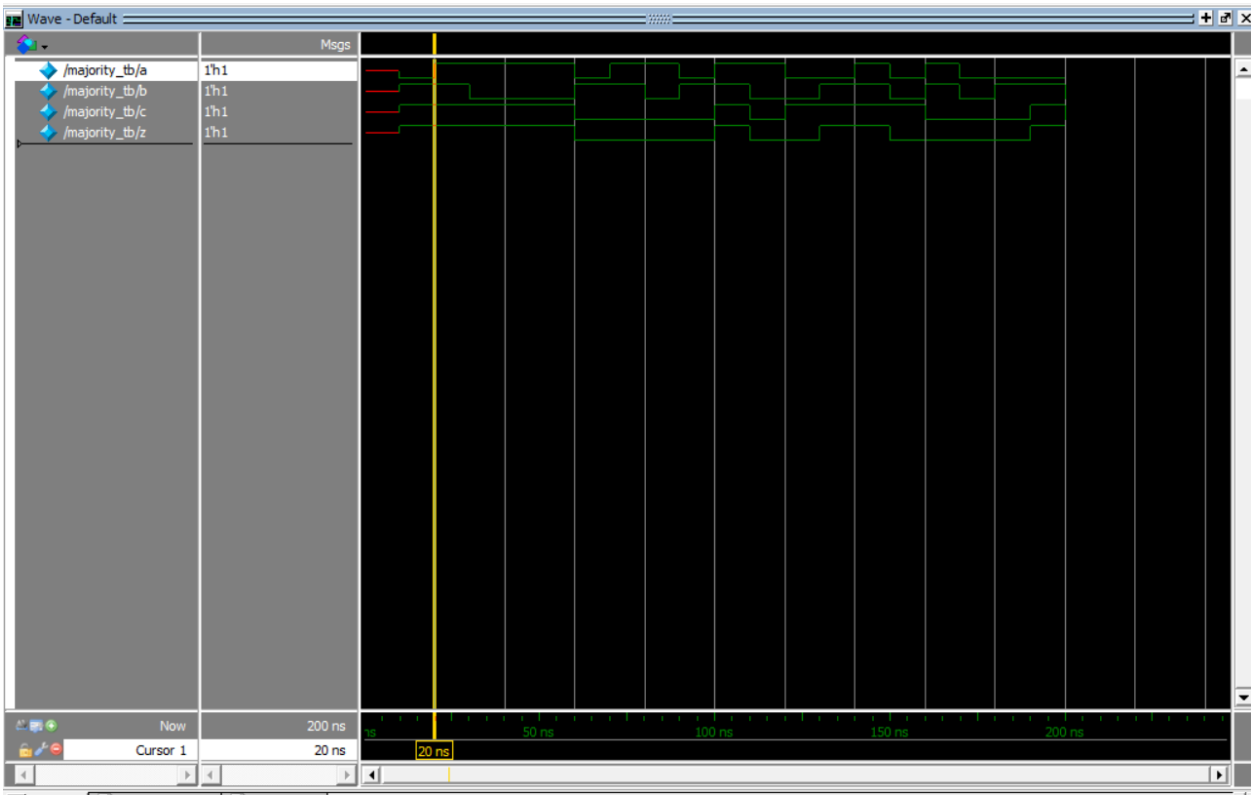


Majority

Testbench

```
module majority_tb();
reg a,b,c;
wire z;

majority dut(
.a(a),
.b(b),
.c(c),
.z(z)
);

initial begin
repeat(20) begin
#10;
a = $random();
b = $random();
c = $random();
end
$stop;
end
endmodule : majority_tb
```

Wave



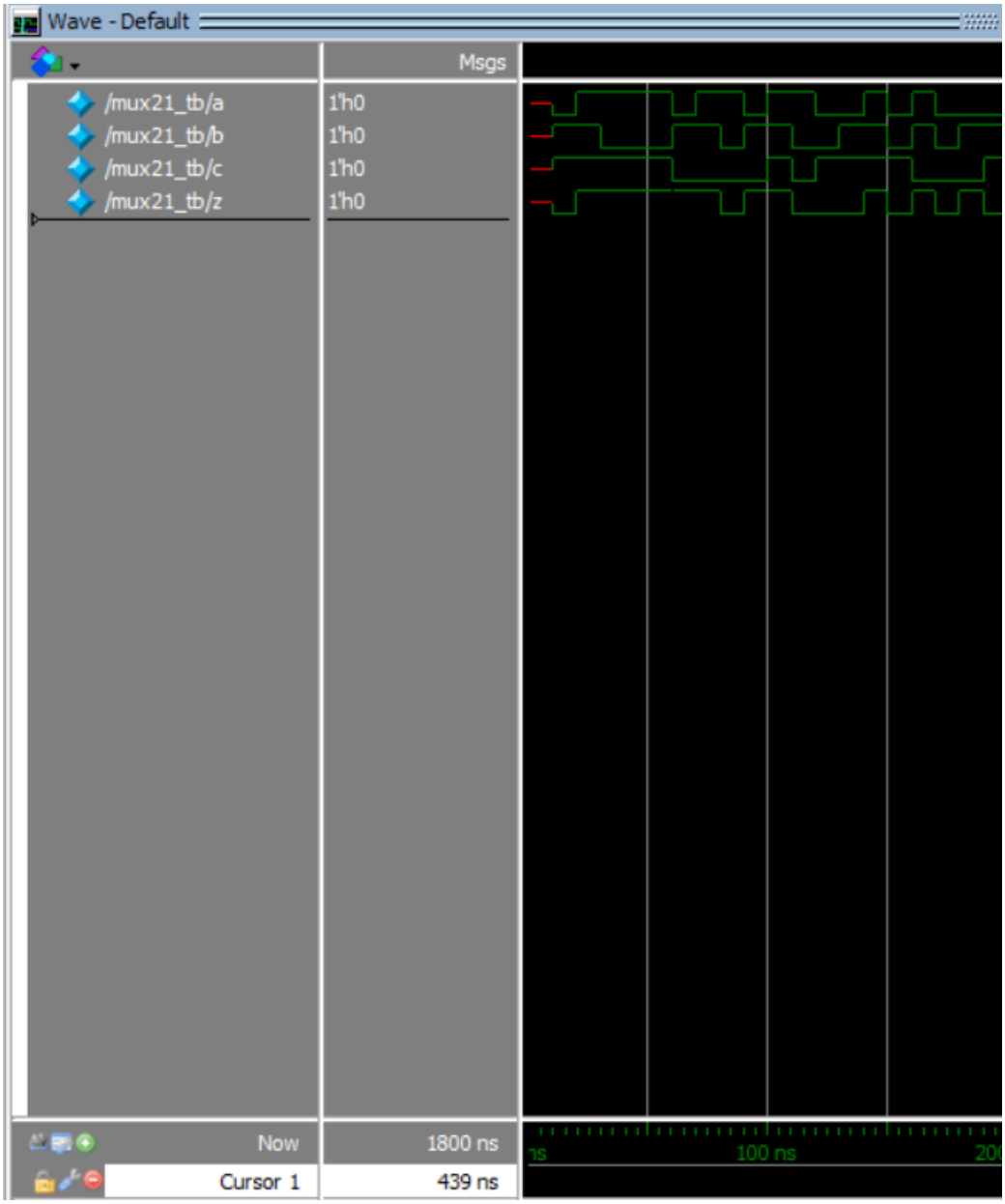
MUX

Testbench

C:/modeltech64_10.7/examples/mux21_tb.v (/mux21_tb) - Default

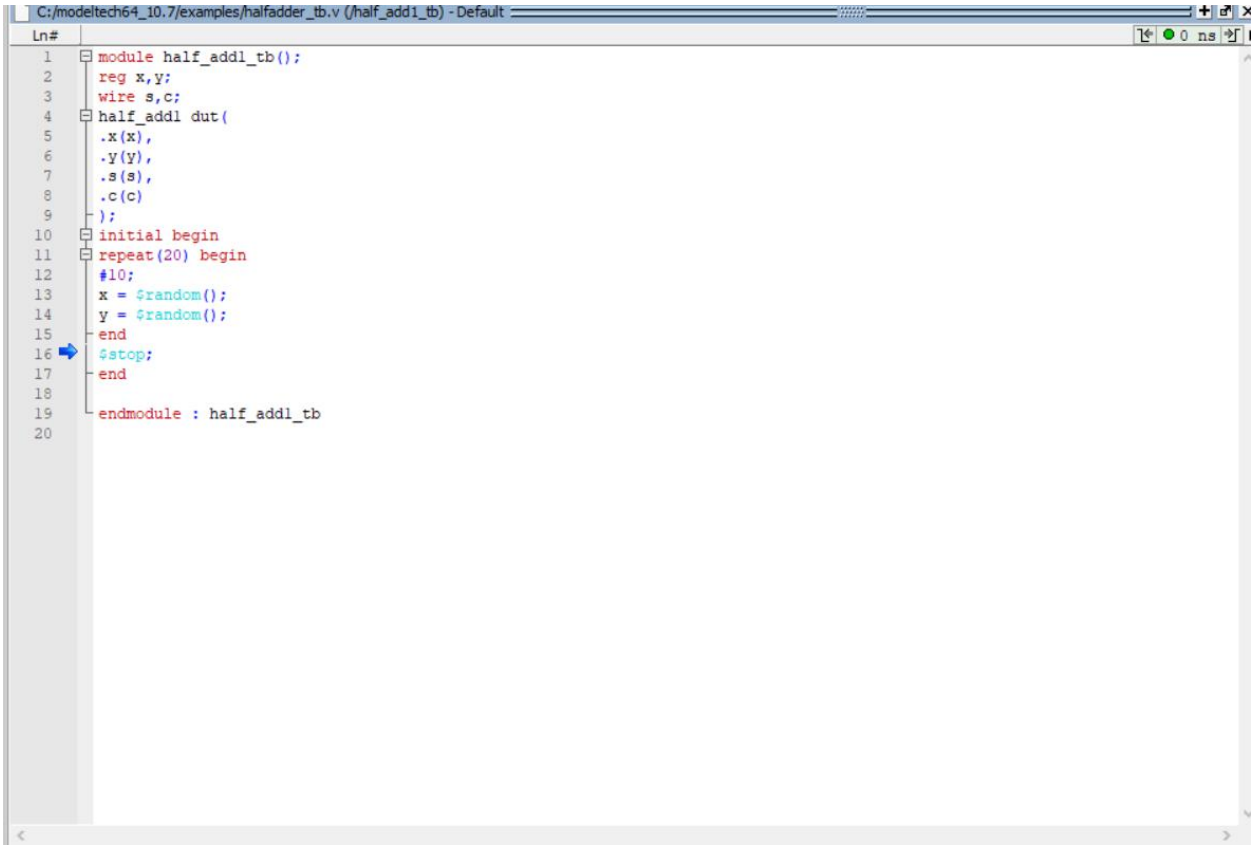
```
module mux21_tb();
reg a,b,c;
wire z;

mux21 dut(
.a(a),
.b(b),
.c(c),
.z(z)
);
initial begin
repeat(20) begin
#10;
a = $random();
b = $random();
c = $random();
end
$stop;
end

endmodule : mux21_tb
```

Wave

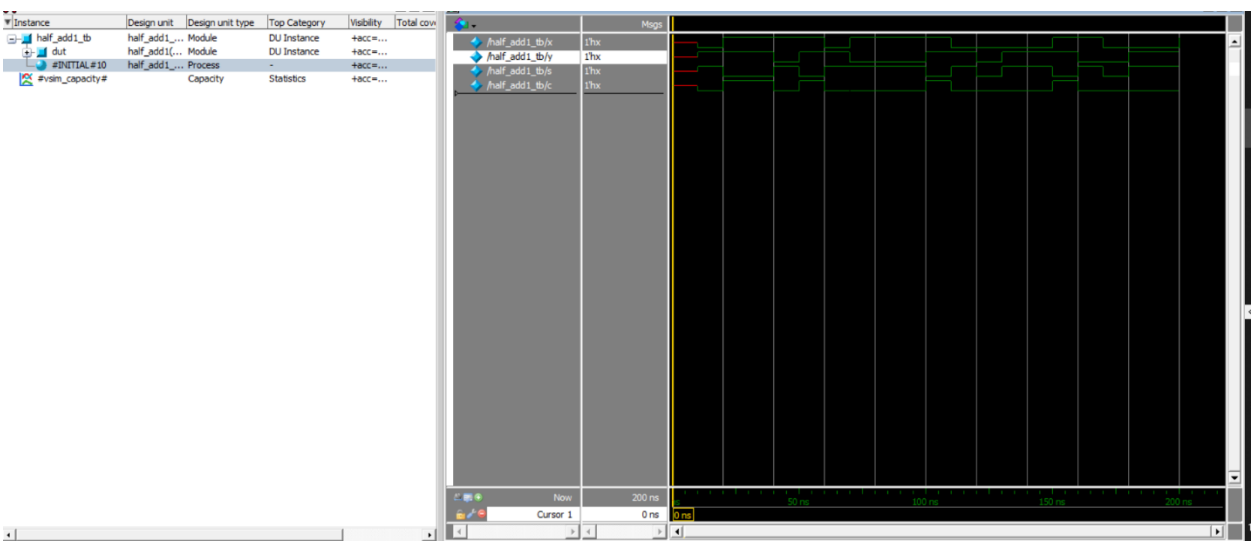


Half Adder

Testbench



Wave



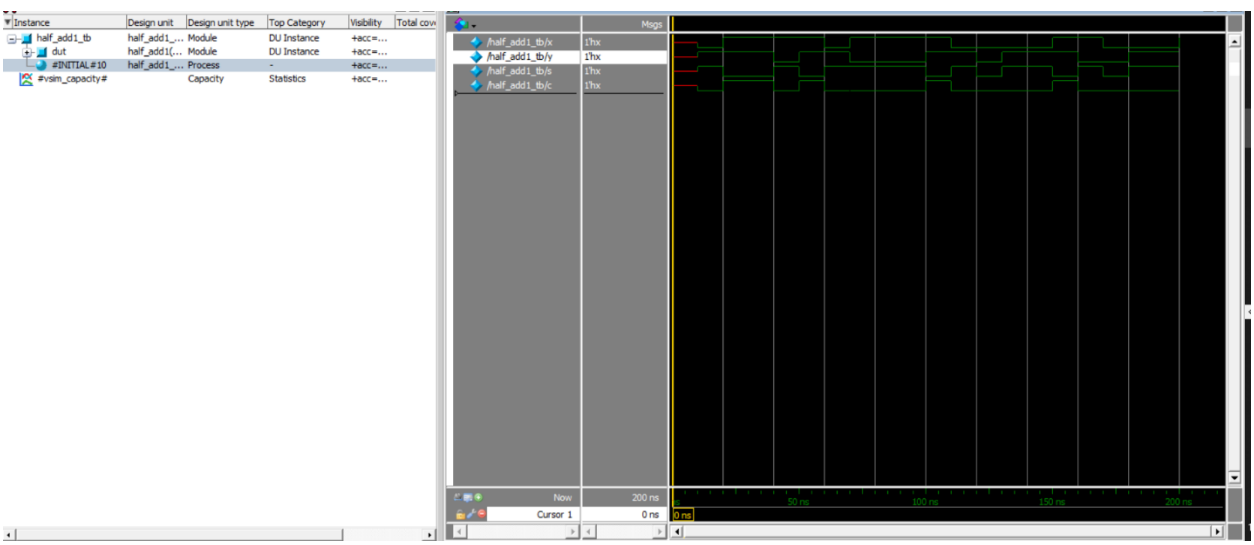
Full Adder

Testbench

C:/modeltech64_10.7/examples/fa1_tb.v (/fa1_tb) - Default

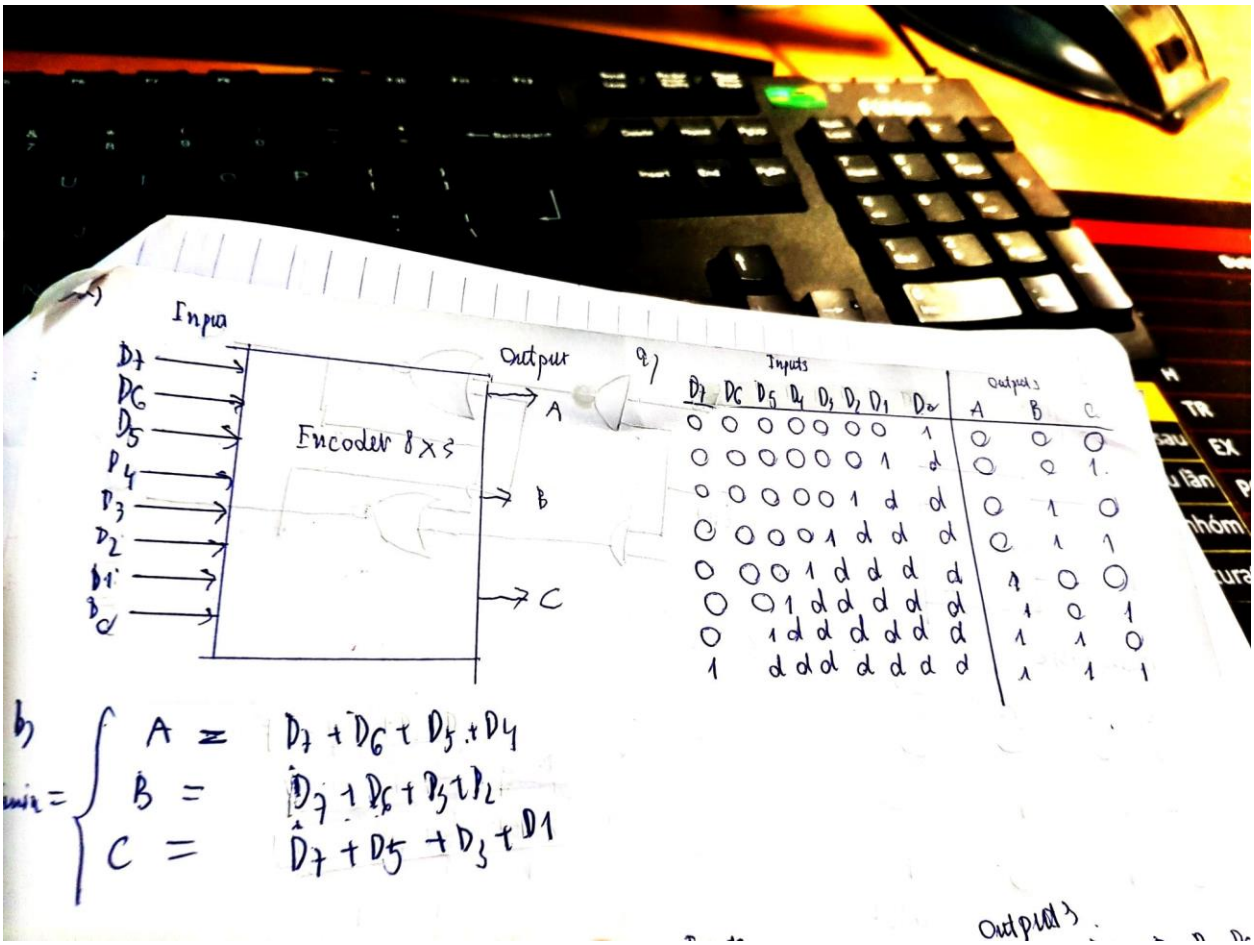
```
module fa1_tb();
reg a,b,cin;
wire s,c;

fa1 dut(
.a(a),
.b(b),
.cin(cin),
.s(s),
.c(c)
);

initial begin
cin =1'b0;
repeat(20) begin
#10;
a = $random();
b = $random();
end
$stop;
end

endmodule : fa1_tb
```

Wave



Ex3

Encoder 8x3


Inpu
D7
D6
D5
D4
D3
D2
D1
D0
Encoder 8x3
Output
A
B
C
Inputs
Outputs
b)
A = D7 + D6 + D5 + D4
B = D7 + D6 + D3 + D2
C = D7 + D5 + D3 + D1
Outputs

Verilog

```
module encoder(
input [7:0] x,
output a,b,c
);

//output a with x7,x6,x5,x4
or w1(a,x[7],x[6],x[5],x[4]);
//output b with x7,x6,x3 and x2
or w2(b,x[7],x[6],x[3],x[2]);
//output c with x7,x5,x3 and x1
or w3(c,x[7],x[5],x[3],x[1]);
endmodule : encoder
```

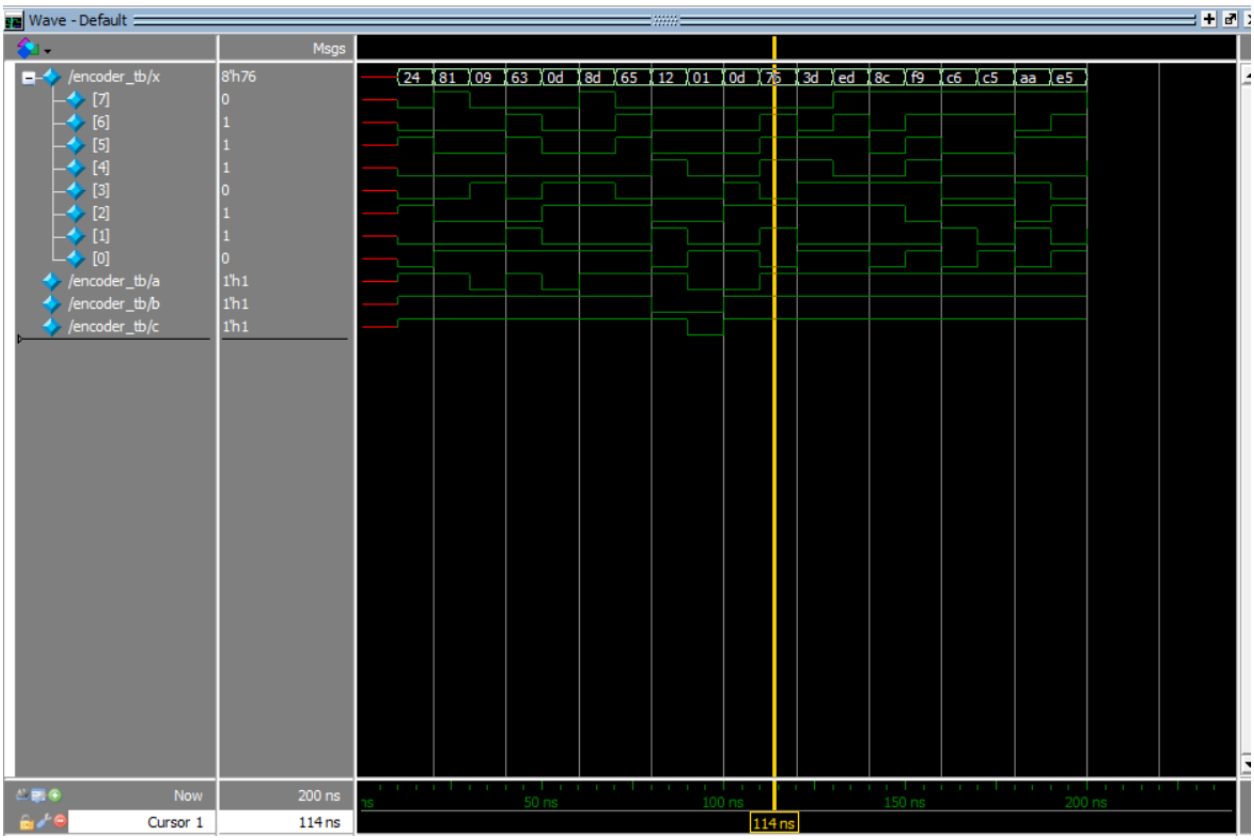
Testbench

C:/modeltech64_10.7/examples/encoder_tb.v (/encoder_tb) - Default

114 ns

```
module encoder_tb();
reg [7:0] x;
wire a,b,c;
encoder dut(
.x(x),
.a(a),
.b(b),
.c(c)
);

initial begin
repeat(20) begin
#10;
x = $random;
end
$stop;
end

endmodule : encoder_tb
```

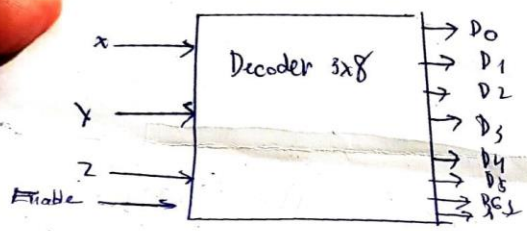
Wave


Wave - Default
Msgs
/encoder_tb/x
8'h76
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/encoder_tb/a
1'h1
/encoder_tb/b
1'h1
/encoder_tb/c
1'h1
24 81 09 63 0d 8d 65 12 01 0d 76 3d ed 8c f9 c6 c5 aa e5
Now
200 ns
Cursor 1
114 ns
50 ns
100 ns
150 ns
200 ns
114 ns

Decoder 3x8

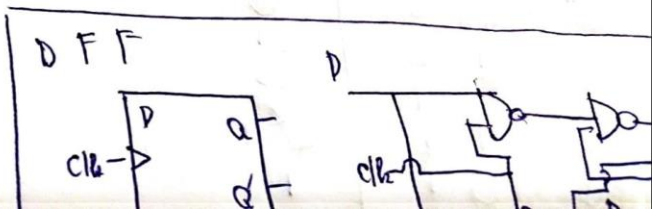
b)

$$f_{min} = \begin{cases} A = D_7 + D_6 + D_5 + D_4 \\ B = D_7 + D_6 + D_3 + D_2 \\ C = D_7 + D_5 + D_3 + D_1 \end{cases}$$



Inputs X	Y	Z	Outputs D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$$f_{min} = \begin{cases} D_0 = x'y'z' \\ D_1 = x'y'z \\ D_2 = \cancel{x}\, xy'z' \\ D_3 = x'y'z \\ D_4 = xy'z \\ D_5 = xy'z \\ D_6 = xyz' \\ D_7 = xyz \end{cases}$$



Verilog

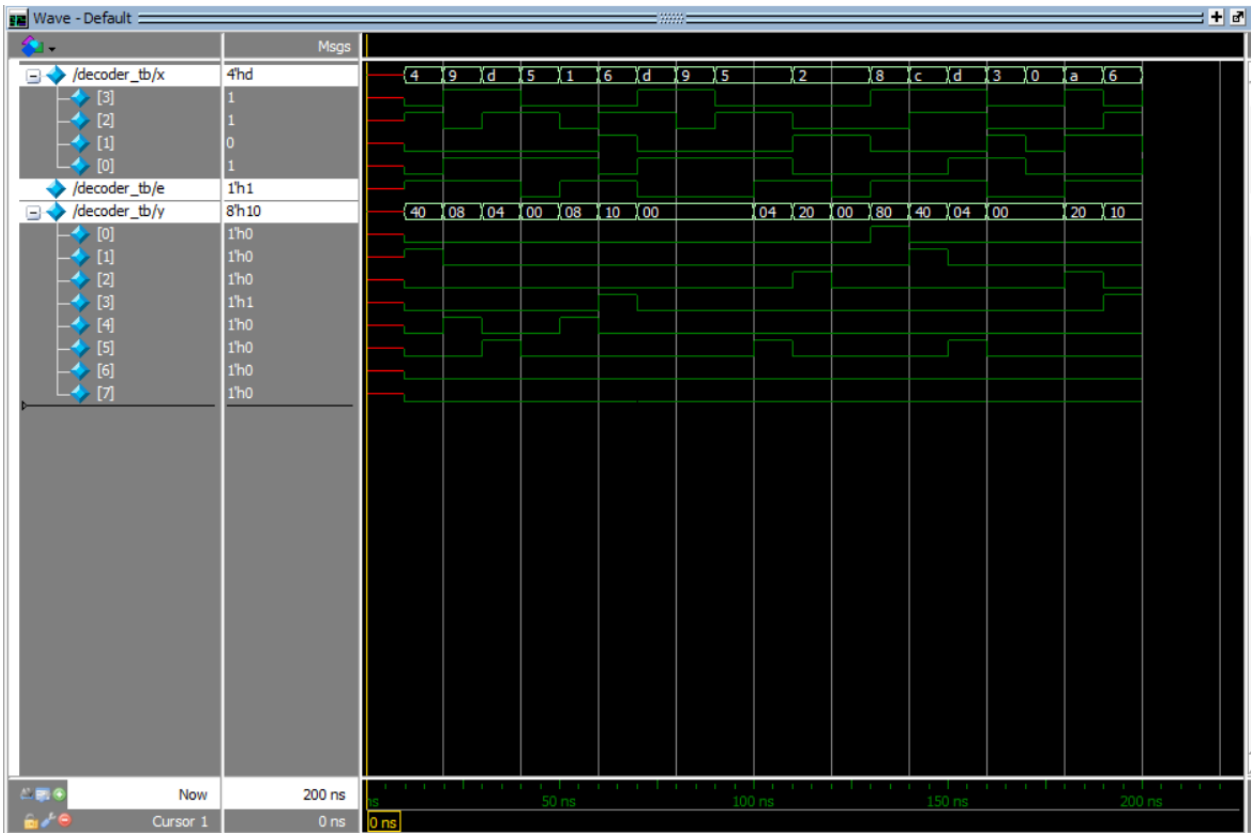
C:/modeltech64_10.7/examples/decoder.v (/decoder_tb/dut) - Default

```
module decoder(x,e,y);
input [2:0] x;
input e;
output [0:7] y;

// output y0 = xo' ^ x1' ^ x2'
and w1(y[0],e,~x[0],~x[1],~x[2]);
//output y1 = x0' ^ x1' ^ x2
and w2(y[1],e,~x[0],~x[1],x[2]);
//output y2 = xo' ^ x1 ^ x2'
and w3(y[2],e,~x[0],x[1],~x[2]);
//output y3 = x0' ^ x1 ^ x2
and w4(y[3],e,~x[0],x[1],x[2]);
//output y4 = x0 ^ x1' ^ x2'
and w5(y[4],e,x[0],~x[1],~x[2]);
//output y5 = x0 ^x1' ^ x2
and w6(y[5],e,x[0],~x[1],x[2]);
//output y6 = x0 ^ x1 ^ x0'
and w7(y[6],e,x[0],x[1],~x[2]);
//output y7 = x0 ^ x1 ^ x2
and w8(y[7],e,x[0],x[1],x[2]);

endmodule : decoder
```

Testbench

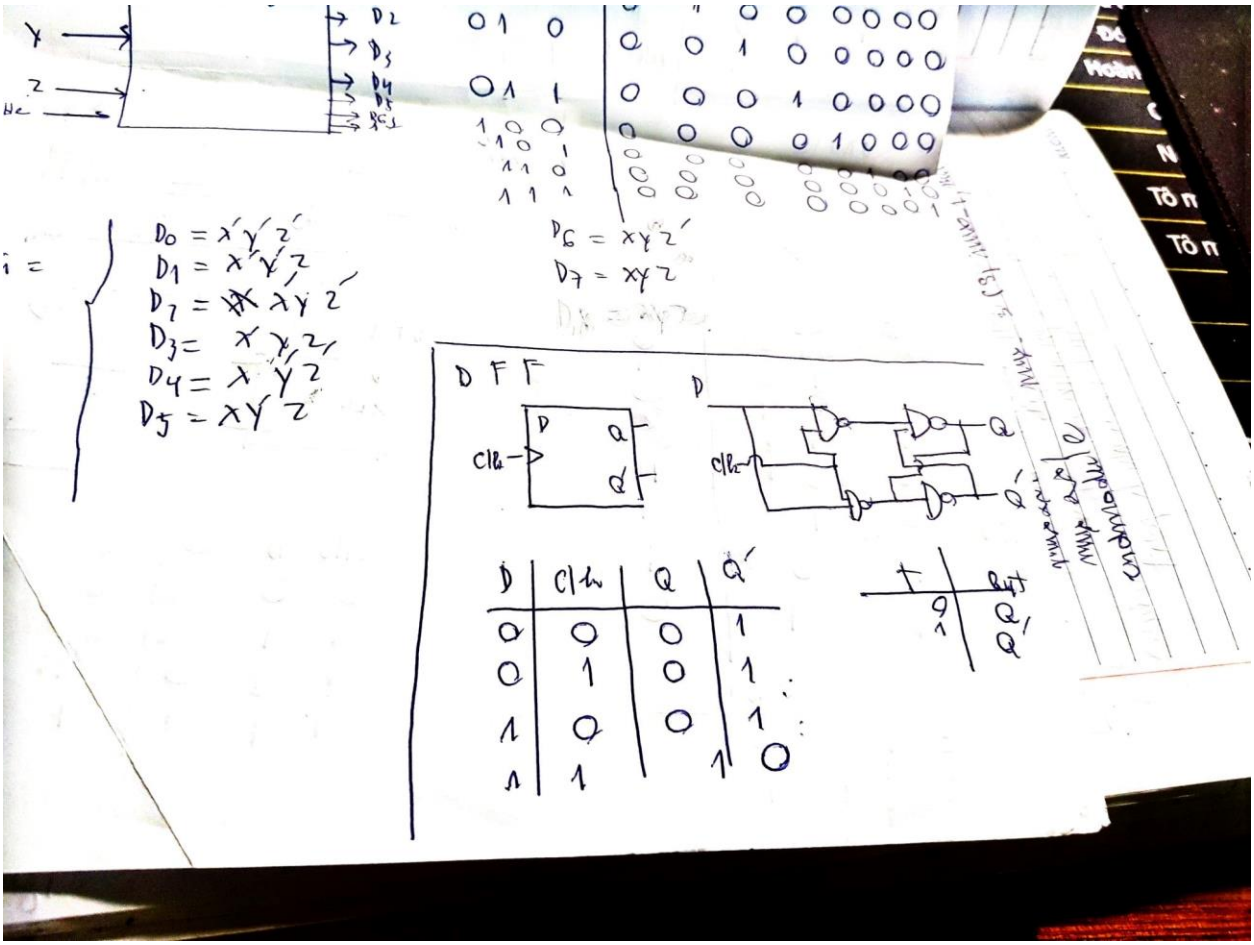
```
module decoder_tb();
reg [3:0] x;
reg e;
wire [0:7] y;

decoder dut(
.x(x),
.e(e),
.y(y)
);
initial begin
repeat(20) begin
#10;
x = $random();
e = $random();
end
$stop;
end

endmodule : decoder_tb
```

Wave


Wave - Default
Msgs
/decoder_tb/x
4'hd
[3]
[2]
[1]
[0]
/decoder_tb/e
1'h1
/decoder_tb/y
8'h10
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
Now
200 ns
Cursor 1
0 ns
50 ns
100 ns
150 ns
200 ns

Ex4

DFF


D0 = x'y'z'
D1 = x'y'z
D7 = xyz'
D3 = x'yz
D4 = xy'z
D5 = xy'z
D6 = xyz'
D7 = xyz
D F F
D
Clk
Q
Q'
D | Clk | Q | Q'
0 | 0 | 0 | 1
0 | 1 | 0 | 1
1 | 0 | 0 | 1
1 | 1 | 1 | 0

Verilog

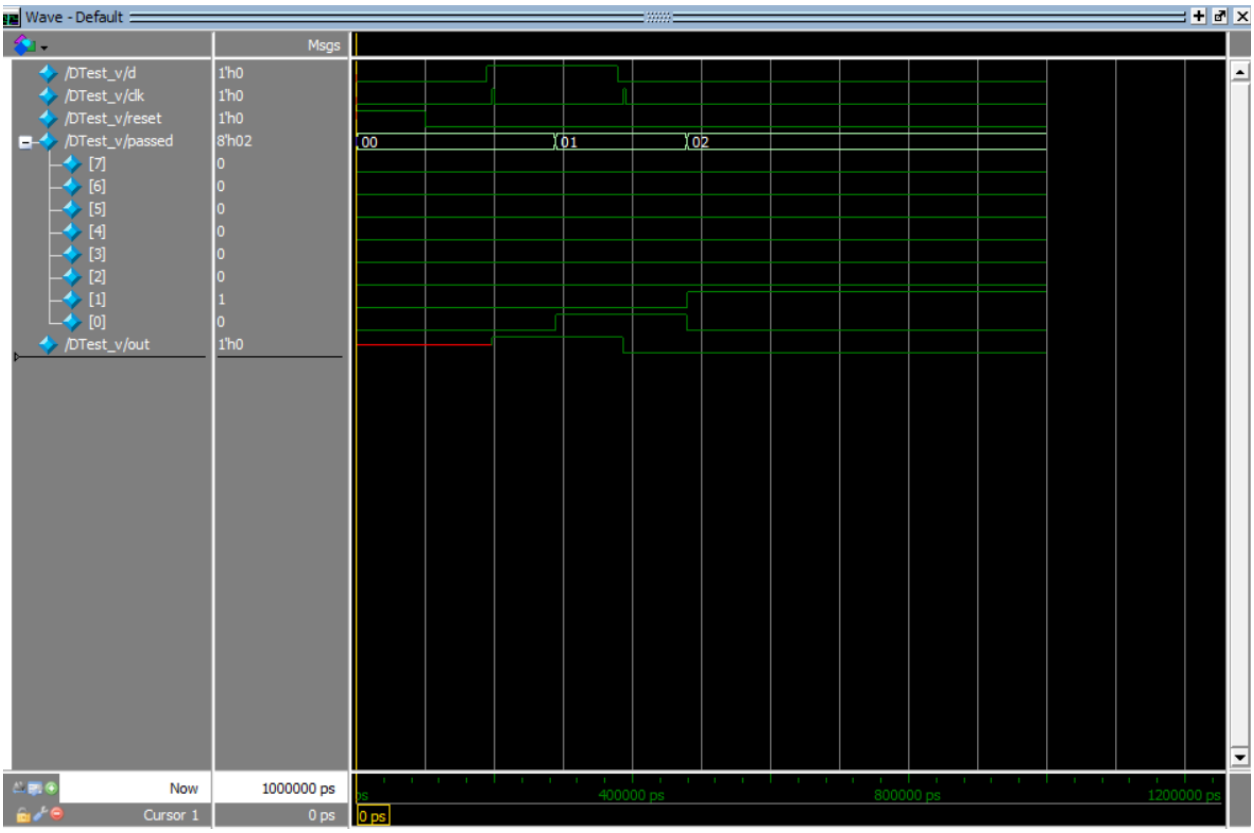
```
`timescale 1ns / 1ps
module d_flip_flop ( out, din ,clk ,reset  );

output reg out ; //needs to be an reg because it is used to hold values even after the clock pulse.
input wire din, clk,reset ;//can be wires as it need not hold a value.

always @ (posedge (clk)) begin
 if (reset)
  out <= 0;
 else
  out <= din ;
end

endmodule
```

Testbench

C:/modeltech64_10.7/examples/d_tb.v (/DTest_v) - Default

```
`timescale 1ns / 1ps

`define STRLEN 15
module DTest_v;
task passTest;
input actualOut, expectedOut;
input [`STRLEN*8:0] testType;
inout [7:0] passed;

if (actualOut==expectedOut)
begin
        $display ("%s passed", testType);
        passed  = passed + 1;
end

else
        $display ("%s failed : %d should be %d",testType,actualOut,expectedOut );

endtask

task allPassed;

input   [7:0] passed;
input   [7:0] numTests;

if (passed==numTests)
        $display ("All tests passed");
else
        $display("Some tests failed");

        endtask

//inputs
reg d;
reg clk;
reg reset;

reg[7:0] passed;
```

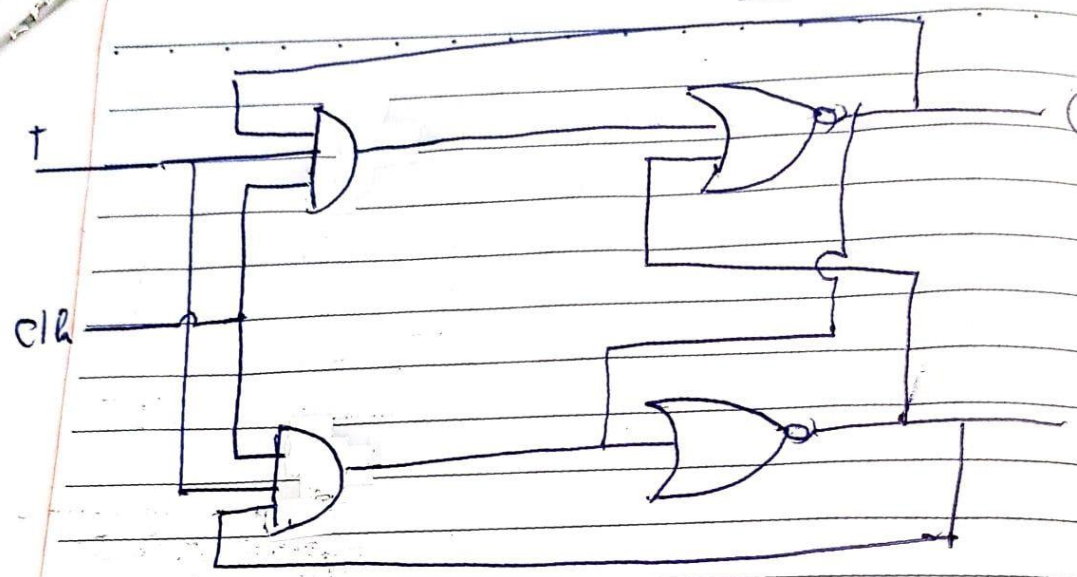
Wave


Wave - Default
Msgs
/DTest_v/d
/DTest_v/clk
/DTest_v/reset
/DTest_v/passed
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/DTest_v/out
1'h0
1'h0
1'h0
8'h02
0
0
0
0
0
0
1
0
1'h0
00
01
02
Now
1000000 ps
Cursor 1
0 ps
400000 ps
800000 ps
1200000 ps
0 ps

TFF

TFF

Date . . No.....



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

Verilog

C:/modeltech64_10.7/examples/t.v (/TFlipFlopTB/uut) - Default

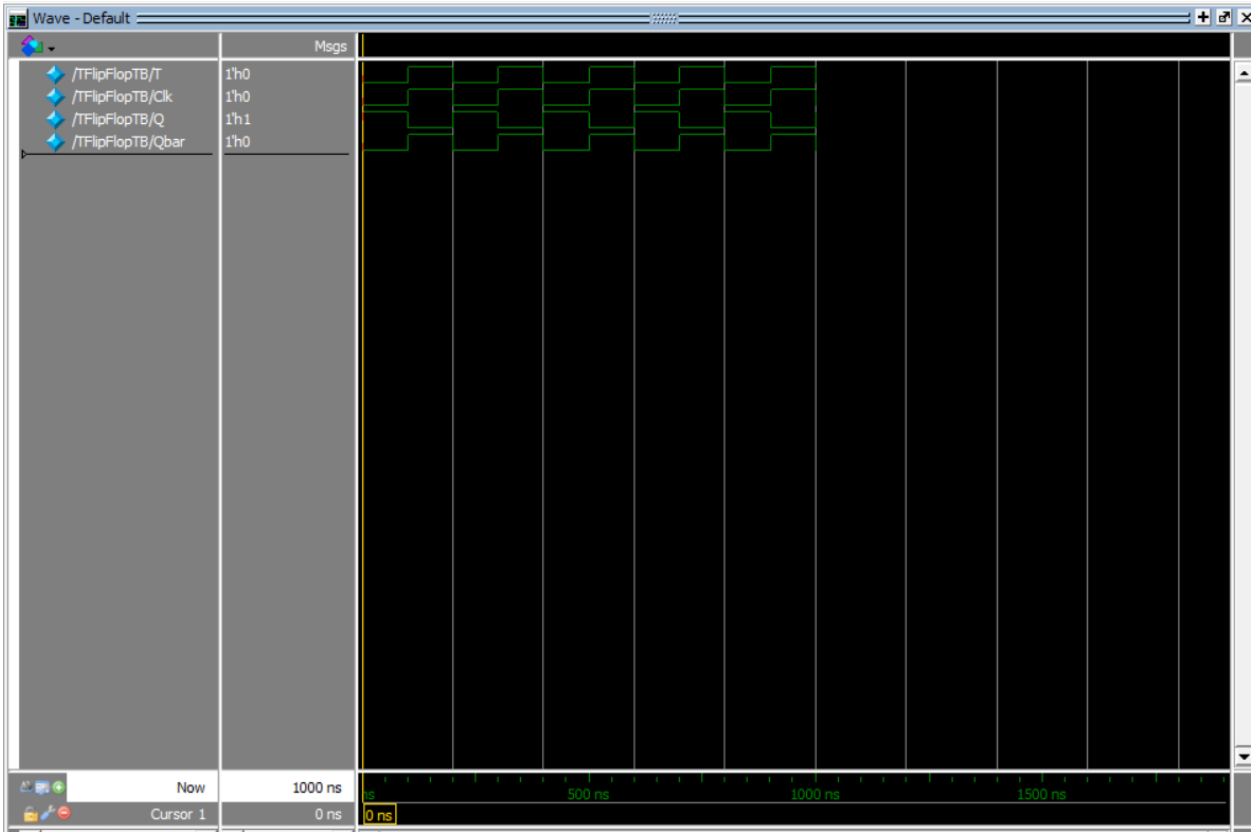
```
module TFlipFlop(T,Clk,Q,Qbar );
input T,Clk;
output Q,Qbar;
reg Q,Qbar;
always@(T,posedge(Clk))
begin
if(T==0)
begin
 Q =1'b1;
 Qbar=1'b0;
end
else
begin
 Q =1'b0;
 Qbar=1'b1;
 end
end
endmodule
```

Testbench

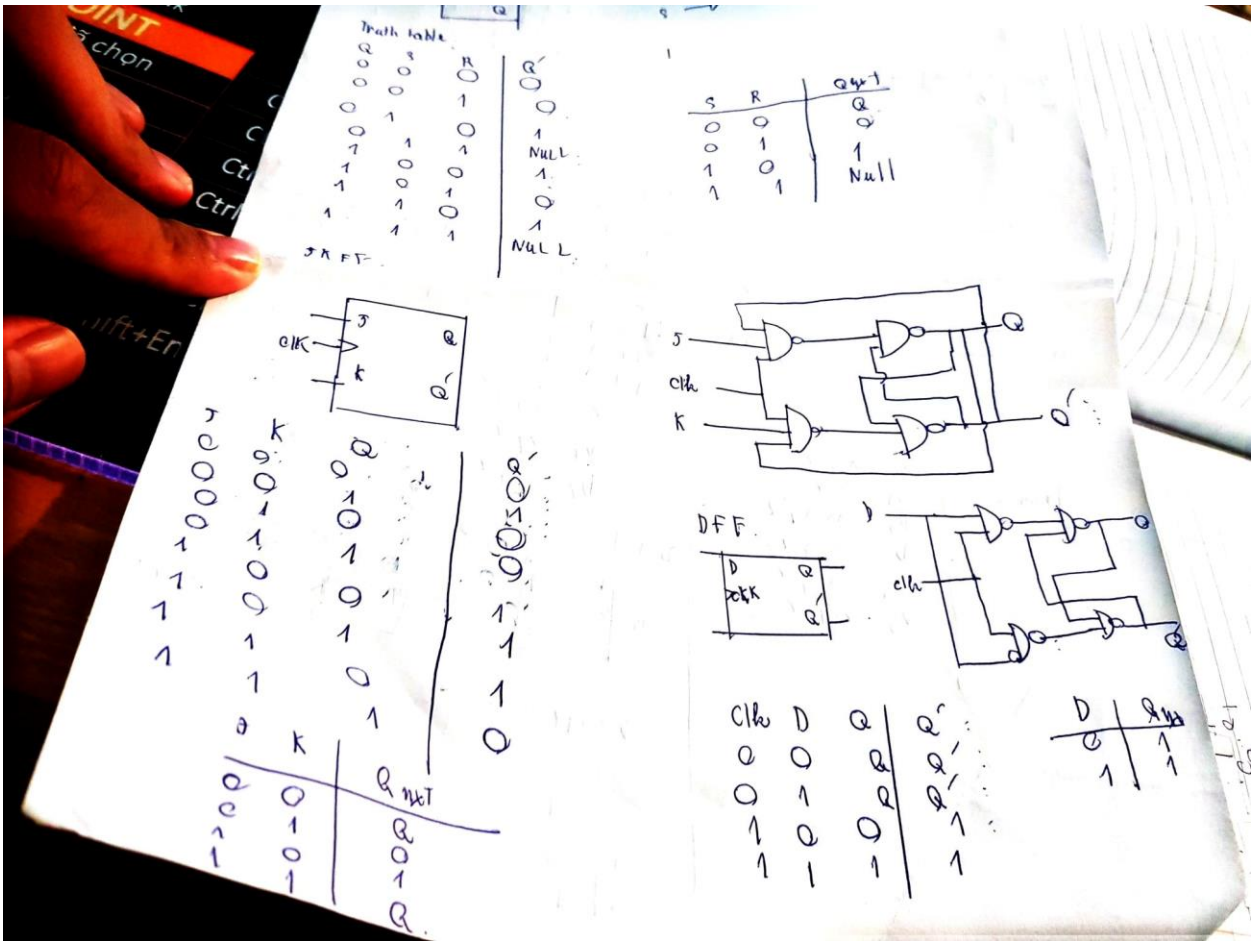
C:/modeltech64_10.7/examples/tff_tb.v (/TFlipFlopTB) - Default

```
module TFlipFlopTB;

// Inputs
reg T;
reg Clk;

// Outputs
wire Q;
wire Qbar;

// Instantiate the Unit Under Test (UUT)
TFlipFlop uut (
.T(T),
.Clk(Clk),
.Q(Q),
.Qbar(Qbar)
);

 initial Clk = 0;
    always #100 Clk = ~Clk;
 initial T=0;
   always #100 T=~T;
initial begin
// Initialize Inputs
T = 0;
Clk = 0;
// Wait 100 ns for global reset to finish
 // Add stimulus here
end
endmodule
```

Wave


Wave - Default
Msgs
/TFlipFlopTB/T
/TFlipFlopTB/Clk
/TFlipFlopTB/Q
/TFlipFlopTB/Qbar
1'h0
1'h0
1'h1
1'h0
Now
1000 ns
Cursor 1
0 ns
500 ns
1000 ns
1500 ns
0 ns

JKFF



Verilog

C:/modeltech64_10.7/examples/jkff.v (/jk_tb/dut) - Default

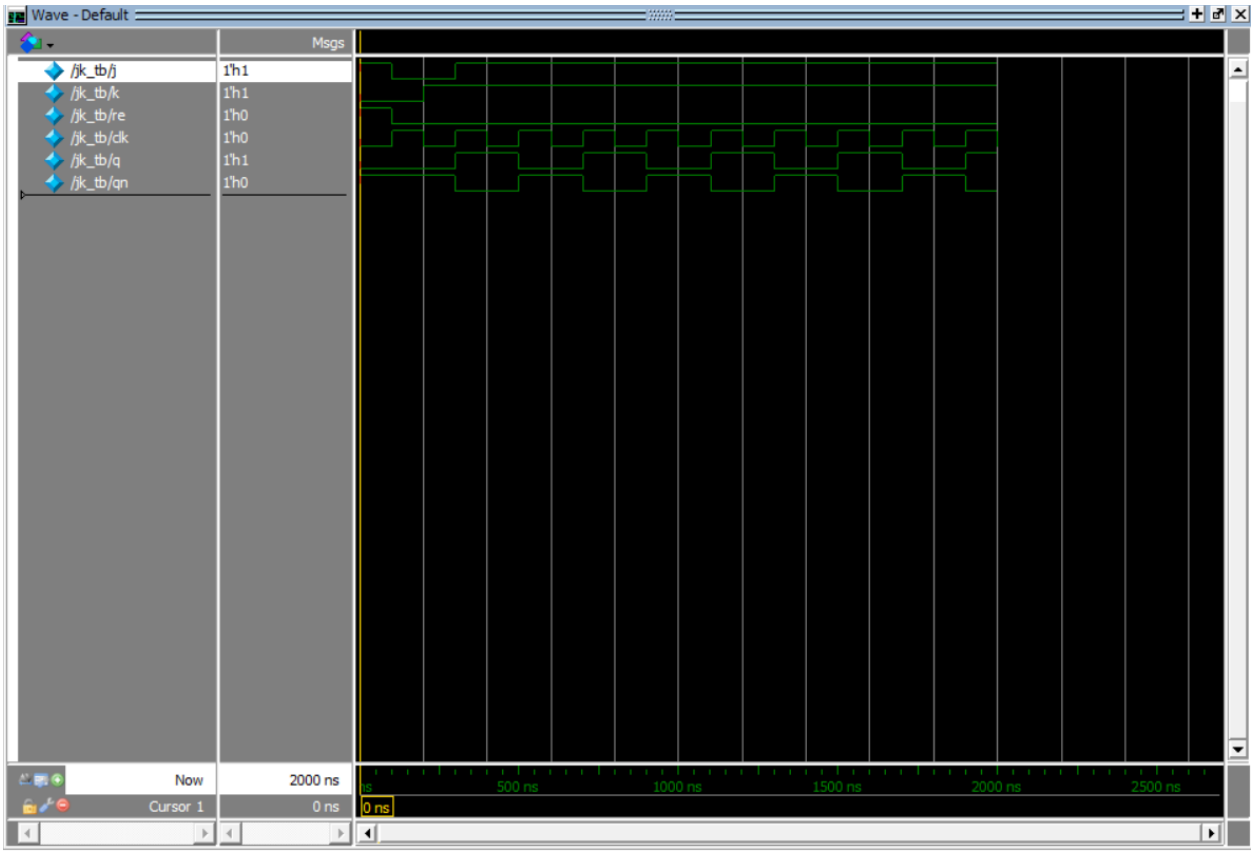
```
module jk(j,k,re,clk,q,qn);
input j,k,re,clk;
output q,qn;
reg q,qn;
initial begin
q = 1'b0;
qn = ~q;
end

always @(posedge clk)
begin
        case({j,k})
        2'b01:q=0;
        2'b10:q=1;
        2'b11:q=~q;
        endcase
        qn=~q;
end

endmodule :jk
```

Testbench

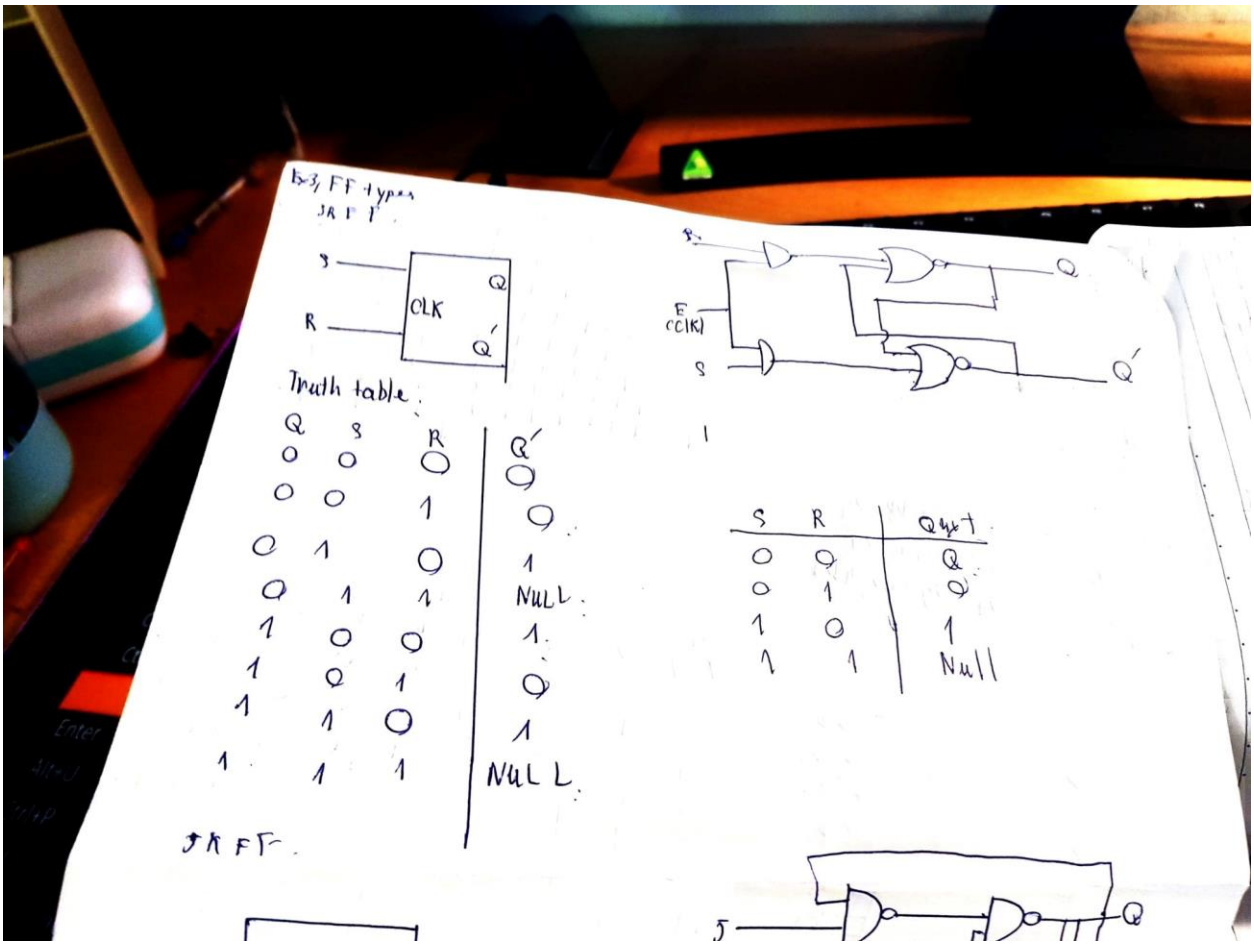
C:/modeltech64_10.7/examples/jk_tb.v (/jk_tb) - Default

```
module jk_tb();
reg j,k,re,clk;
wire q,qn;

jk dut(
.j(j),
.k(k),
.clk(clk),
.re(re),
.q(q),
.qn(qn)
);

initial begin
  forever begin
      clk<=0;
      #100;
      clk<=1;
      #100;
  end
end

initial begin
re=1;j=1;k=0;
#100;
re=0;j=0;k=0;
#100;
re=0;j=0;k=1;
#100;
re=0;j=1;k=1;
#100;
end

endmodule : jk_tb
```

Wave


Wave - Default
Msgs
/jk_tb/j
/jk_tb/k
/jk_tb/re
/jk_tb/clk
/jk_tb/q
/jk_tb/qn
1'h1
1'h1
1'h0
1'h0
1'h1
1'h0
Now
2000 ns
Cursor 1
0 ns
500 ns
1000 ns
1500 ns
2000 ns
2500 ns
0 ns

SRFF


S
R
CLK
Q
Q'
Truth table
Q S R Q'
0 0 0 0
0 0 1 0
0 1 0 1
0 1 1 NULL
1 0 0 1
1 0 1 0
1 1 0 1
1 1 1 NULL
S R Qnext
0 0 Q
0 1 Q'
1 0 1
1 1 Null
E (CLK)
S
Q
Q'

Verilog

C:/modeltech64_10.7/examples/sr.v (/srff_tb/dut) - Default

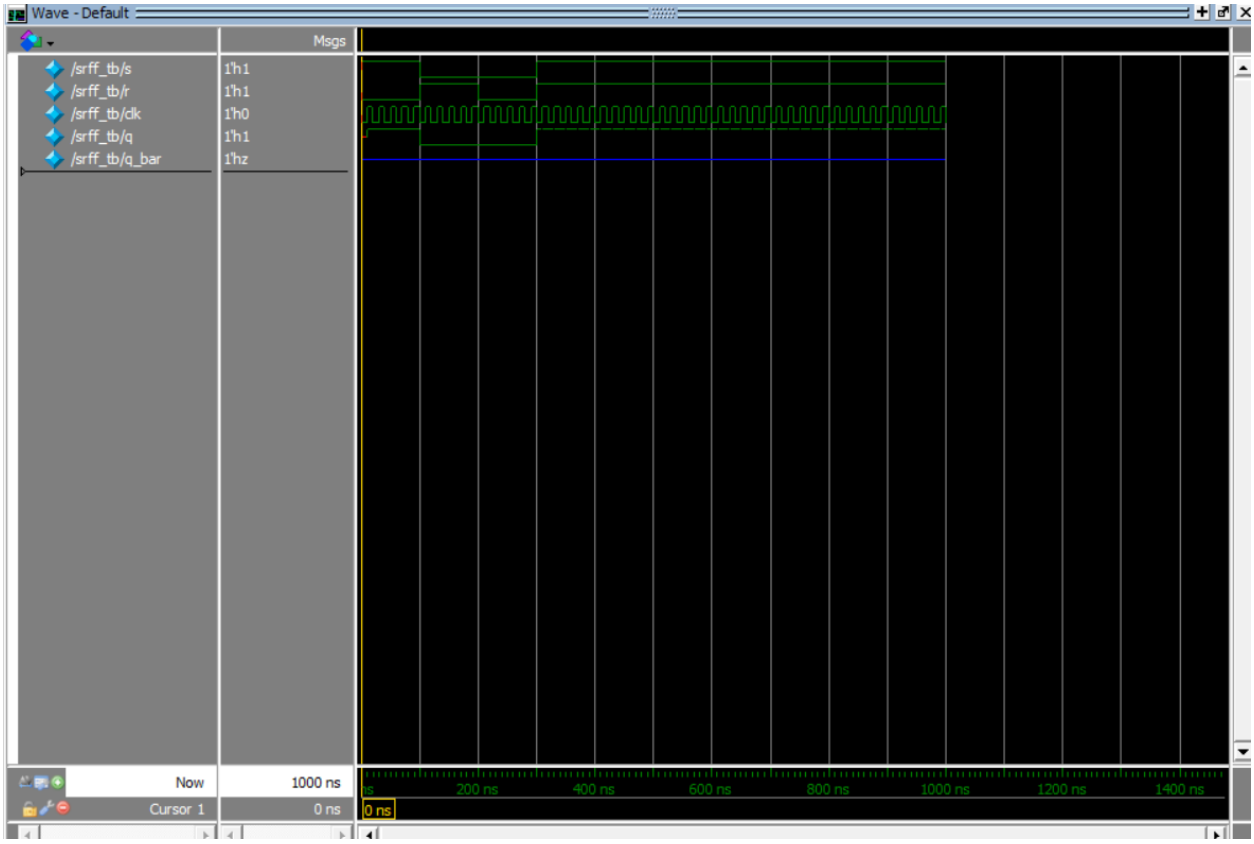
```
module srff(s,r,clk,q,q_bar);
input s,r,clk;
output q,q_bar;
wire nand1_out,nand2_out;

nand(nand1_out,clk,s);
nand(nand2_out,clk,r);
nand(q,nand1_out,qbar);
nand(qbar,nand2_out,q);
endmodule :srff
```

Testbench

C:/modeltech64_10.7/examples/sr_tb.v (/srff_tb) - Default

```
module srff_tb();
reg s,r,clk;
wire q,q_bar;

srff dut(
.s(s),
.r(r),
.clk(clk),
.q(q),
.q_bar(q_bar)
);

initial begin
clk =0;
forever #10 clk = ~clk;
end
initial begin
s=1;r=0;
#100; s=0;r=1;
#100; s=0;r=0;
#100; s=1;r=1;
end
endmodule :srff_tb
```

Wave


Wave - Default
Msgs
/srff_tb/s
/srff_tb/r
/srff_tb/clk
/srff_tb/q
/srff_tb/q_bar
1'h1
1'h1
1'h0
1'h1
1'hz
Now
1000 ns
Cursor 1
0 ns
200 ns
400 ns
600 ns
800 ns
1000 ns
1200 ns
1400 ns
0 ns