Placement preparation_VERILOG Assignment (7th AUGUST)

SYSTEM-TASKS & COMPILER DIRECTIVES Assignment

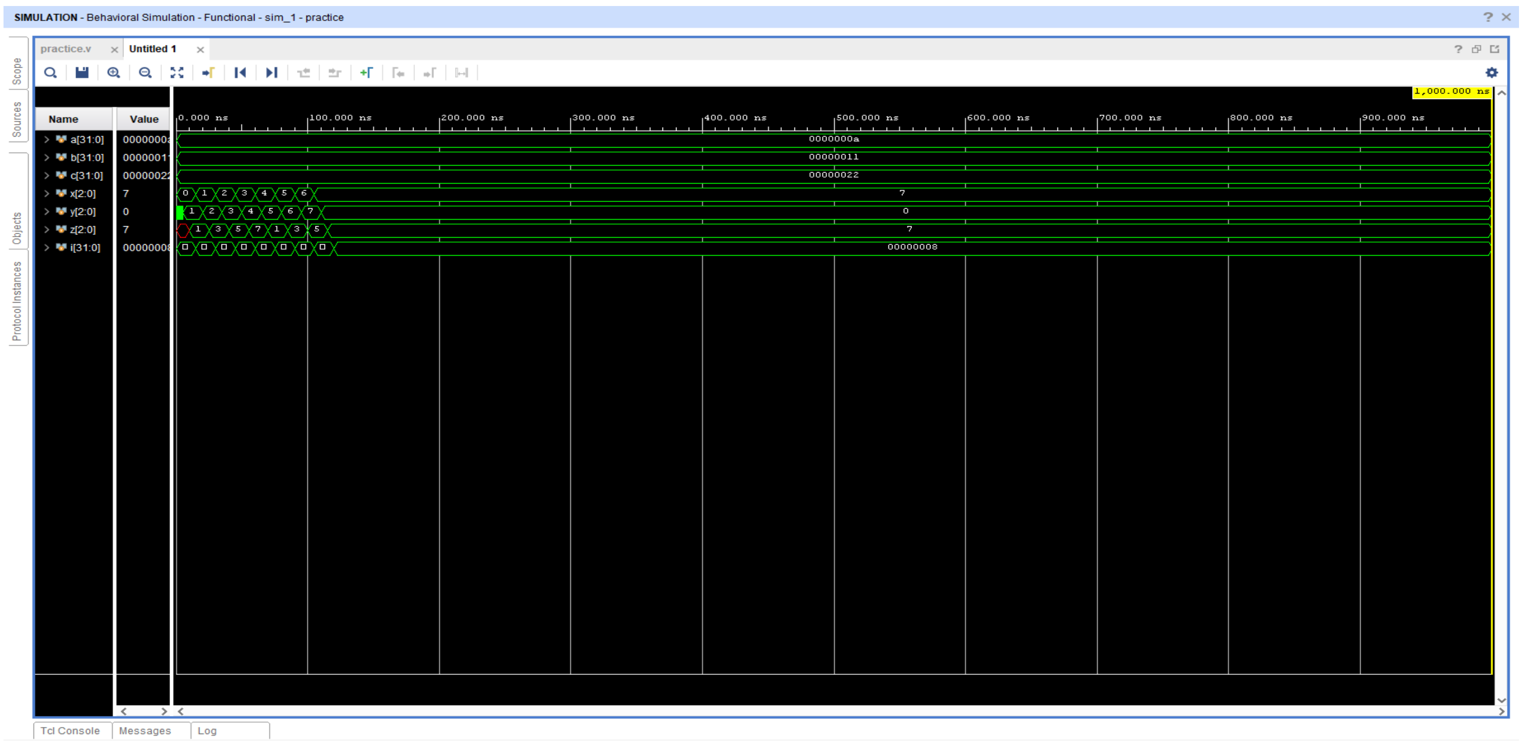
Snippet #1: -

```
module practice;

integer a,b,c;
reg [2:0] x,y,z;
integer i;

initial begin
a = 10;
b =17;
c = 34;
for(i=0;i<8;i=i+1) begin
x <= i;
#5;
y <= x+1'b1;
#5;
z <= x+y;
#5;
end
$strobe("$STROBE OUTPUT__Value of x = %0d, y=%0d, z=%0d",x,y,z);
$display("$DISPLAY OUTPUT__Value of x = %0d, y=%0d, z=%0d",x,y,z);
end
initial
$monitor("$MONITOR OUTPUT__Value of x = %0d, y=%0d, z=%0d",x,y,z);
endmodule
```

Output Snippet #1:-

```
Time resolution is 1 ps
$MONITOR OUTPUT__Value of x = 0, y=x, z=x
$MONITOR OUTPUT__Value of x = 0, y=1, z=x
```

```
$MONITOR OUTPUT__Value of x = 0, y=1, z=1

$MONITOR OUTPUT__Value of x = 1, y=1, z=1

$MONITOR OUTPUT__Value of x = 1, y=2, z=1

$MONITOR OUTPUT__Value of x = 1, y=2, z=3

$MONITOR OUTPUT__Value of x = 2, y=2, z=3

$MONITOR OUTPUT__Value of x = 2, y=3, z=3

$MONITOR OUTPUT__Value of x = 2, y=3, z=5

$MONITOR OUTPUT__Value of x = 3, y=3, z=5

$MONITOR OUTPUT__Value of x = 3, y=4, z=5

$MONITOR OUTPUT__Value of x = 3, y=4, z=7

$MONITOR OUTPUT__Value of x = 4, y=4, z=7

$MONITOR OUTPUT__Value of x = 4, y=5, z=7

$MONITOR OUTPUT__Value of x = 4, y=5, z=1

$MONITOR OUTPUT__Value of x = 5, y=5, z=1

$MONITOR OUTPUT__Value of x = 5, y=6, z=1

$MONITOR OUTPUT__Value of x = 5, y=6, z=3

$MONITOR OUTPUT__Value of x = 6, y=6, z=3

$MONITOR OUTPUT__Value of x = 6, y=7, z=3

$MONITOR OUTPUT__Value of x = 6, y=7, z=5

$MONITOR OUTPUT__Value of x = 7, y=7, z=5

$MONITOR OUTPUT__Value of x = 7, y=0, z=5

$MONITOR OUTPUT__Value of x = 7, y=0, z=7

$DISPLAY OUTPUT__Value of x = 7, y=0, z=7

$STROBE OUTPUT__Value of x = 7, y=0, z=7

relaunch_sim: Time (s): cpu = 00:00:02 ; elapsed = 00:00:07 . Memory (MB): peak = 875.629 ; gain = 0.
```



SIMULATION - Behavioral Simulation - Functional - sim_1 - practice

practice.v | Untitled 1

```
Name        Value
a[31:0]     0000000a    0000000a
b[31:0]     00000011    00000011
c[31:0]     00000022    00000022
x[2:0]      7           0 1 2 3 4 5 6 7
y[2:0]      0           1 2 3 4 5 6 7 0
z[2:0]      7           1 3 5 7 1 3 5 7
i[31:0]     00000008    ... 00000008
```

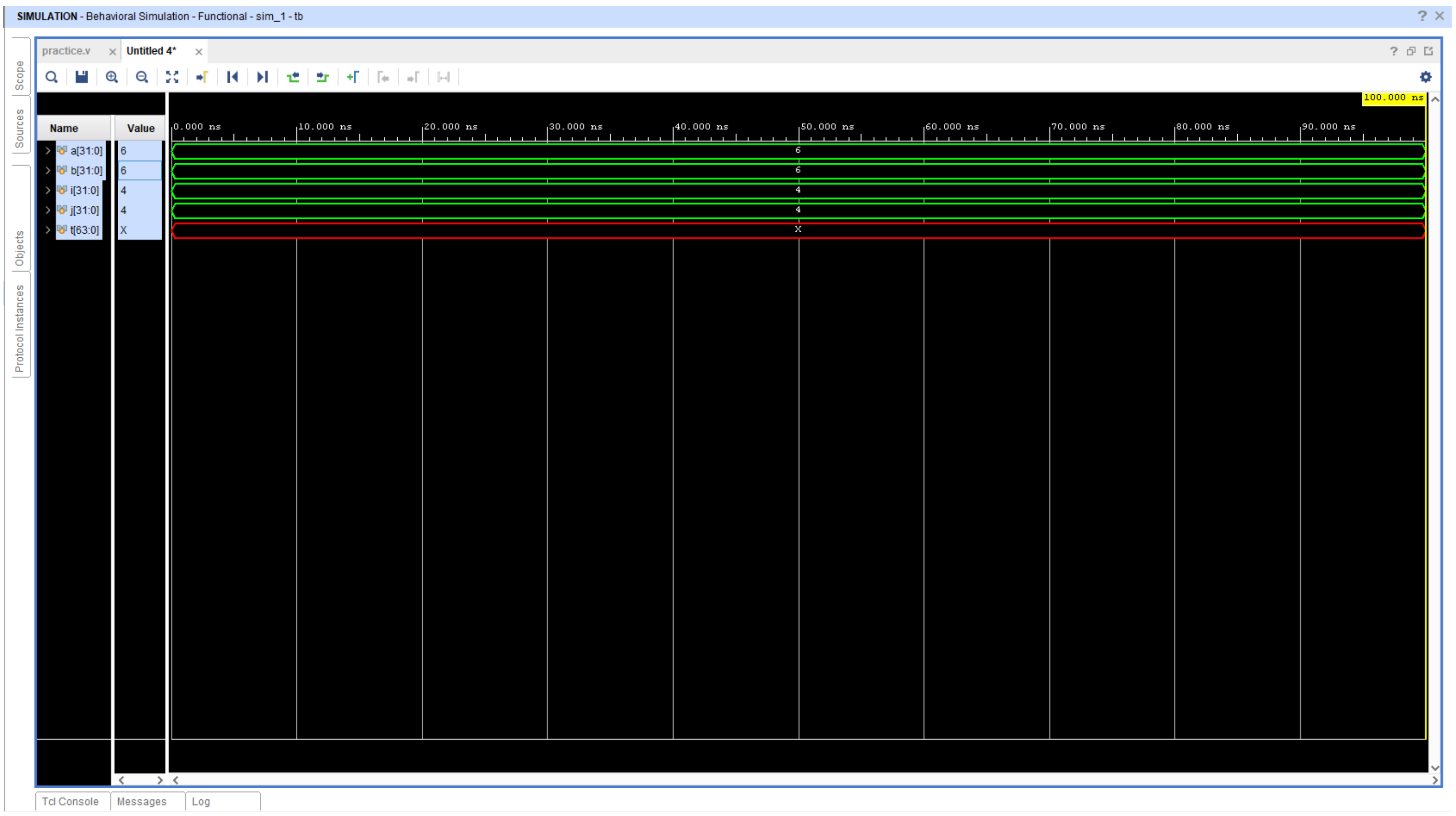
----------##----------

Snippet #2: -

```
module tb;


 integer a=0;

 integer b = 0;

 integer i,j;

 time t;

 initial begin


  for(i=0;i<4;i=i+1) begin

   a = a+i;

   #0;

 $strobe("Value of a is: %0d",a,$time);

   end


 for(j=0;j<4;j=j+1) begin

 b = b+j;

   #0;

 $display("Value of b is: %0d",b,$time);

   end
   #100$finish;
 end
endmodule
```

Output Snippet #2:-

```
# KERNEL: Value of b is: 0            0
# KERNEL: Value of b is: 1            0
# KERNEL: Value of b is: 3            0
# KERNEL: Value of b is: 6            0
# KERNEL: Value of a is: 6            0
# RUNTIME: Info: RUNTIME_0068 design.sv (48): $finish called.
```



```
Name      Value
a[31:0]   6
b[31:0]   6
i[31:0]   4
j[31:0]   4
t[63:0]   X
```

----------##----------

Snippet #3: -

```
module prac;
  integer i, j;
   initial
  begin
   for(i = 0 ; i < 8; i = i+1)
     begin
       $display("[$display],Value of i = %d",i);
     end
```

```
 end


 initial
  begin
   for(i = 0 ; i < 8; i = i+1)
     begin
       $monitor("[$monitor],Value of i = %d",i);
     end
 end
endmodule
```

Output Snippet #3:-

```
# KERNEL: [$display],Value of i =          0
# KERNEL: [$display],Value of i =          1
# KERNEL: [$display],Value of i =          2
# KERNEL: [$display],Value of i =          3
# KERNEL: [$display],Value of i =          4
# KERNEL: [$display],Value of i =          5
# KERNEL: [$display],Value of i =          6
# KERNEL: [$display],Value of i =          7
# KERNEL: [$strobe],Value of i =          8
# KERNEL: Simulation has finished.
```

Snippet #4: - (DUAL PORT RAM(`define compiler directive))

RTL: -

```
`define width 8
`define depth 16
module dual_ram(we_clk,re_clk,rst,din,we,re,we_addr,re_addr,dout);


input rst,we,re,we_clk,re_clk,we,re;
input [(`width-1):0] din;
input [3:0]we_addr,re_addr;
output reg[3:0]dout;


integer i,j;
```

```
reg[(`width-1):0]mem[(`depth-1):0];


always@(posedge we_clk)

begin


if(rst)

begin

for(i=0;i<`depth;i=i+1)

begin

mem[i]=0;

dout=0;

end

end

else

begin

mem[we_addr]=din;

end


end


always@(posedge re_clk)

begin


if(rst)

begin

for(j=0;j<`depth;j=j+1)

begin

mem[j]=0;

dout=0;

end

end

else

begin

dout=mem[re_addr];

end
```

```
end

endmodule
```

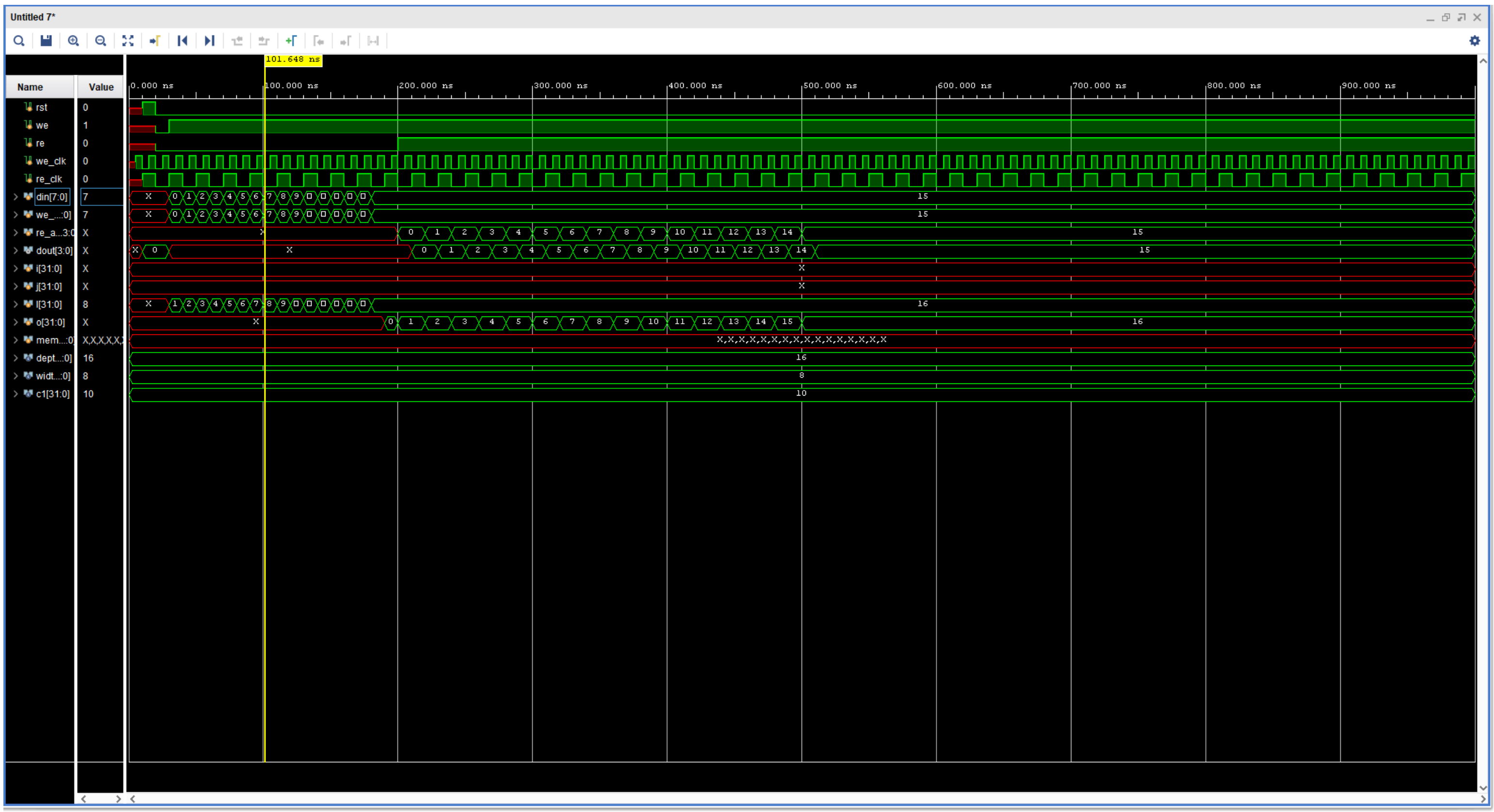
TESTBENCH: -

```
module dual_ram_tb();

reg rst,we,re,we_clk,re_clk;
reg [7:0] din;
reg [3:0]we_addr,re_addr;
wire [3:0]dout;

integer i,j,l,o;
parameter depth=16;
parameter width=8;
parameter c1=10;
reg[(width-1):0]mem[(depth-1):0];

dual_ram DUT(we_clk,re_clk,rst,din,we,re,we_addr,re_addr,dout);

always
begin
#(c1/2);
we_clk=1'd1;
#(c1/2);
we_clk=1'd0;
end


always
begin
#(c1);
re_clk=1'd1;
#(c1);
re_clk=1'd0;
```

```
end


task initialise;
begin
we=0;re=0;
end
endtask


task write(input [3:0]i,input[7:0]j);
begin
@(negedge we_clk)
we=1'b1;
we_addr=i;
din=j;
end
endtask


task read(input [3:0]y);
begin
@(negedge re_clk)
re=1'b1;
re_addr=y;
end
endtask


task reset;
begin
#c1
rst=1'd1;
#c1
rst=1'd0;
end
endtask
```

```
task delay;

begin

#10;

end

endtask


initial

begin

reset();

initialise();

delay();

for(l=0;l<16;l=l+1)

write(l,l);

delay();

for(o=0;o<16;o=o+1)

read(o);

delay();

end


initial

begin

$dumpfile("file.vcd");

$dumpvars();

#2000 $finish;

end


endmodule
```

<u>Output Snippet #4:-</u>



----------###----------

Snippet #5: - (JK FLIP FLOP(`include compiler directive))

RTL: -

```
#1

`include "include.v"

module jk_flip_flop(clk,reset, j,k,q);


input clk,reset,j,k;

output reg q;


always@(posedge clk) begin

if(reset)

q<=0;


else

begin

case({j,k})

`HOLD: q<=q;

`RESET: q<=0;

`SET: q<=1;

`TOGGLE: q<=~q;
```

```
endcase


end

end

endmodule
```

RTL: -

```
#2

`define HOLD 0

`define RESET 1

`define SET 2

`define TOGGLE 3
```

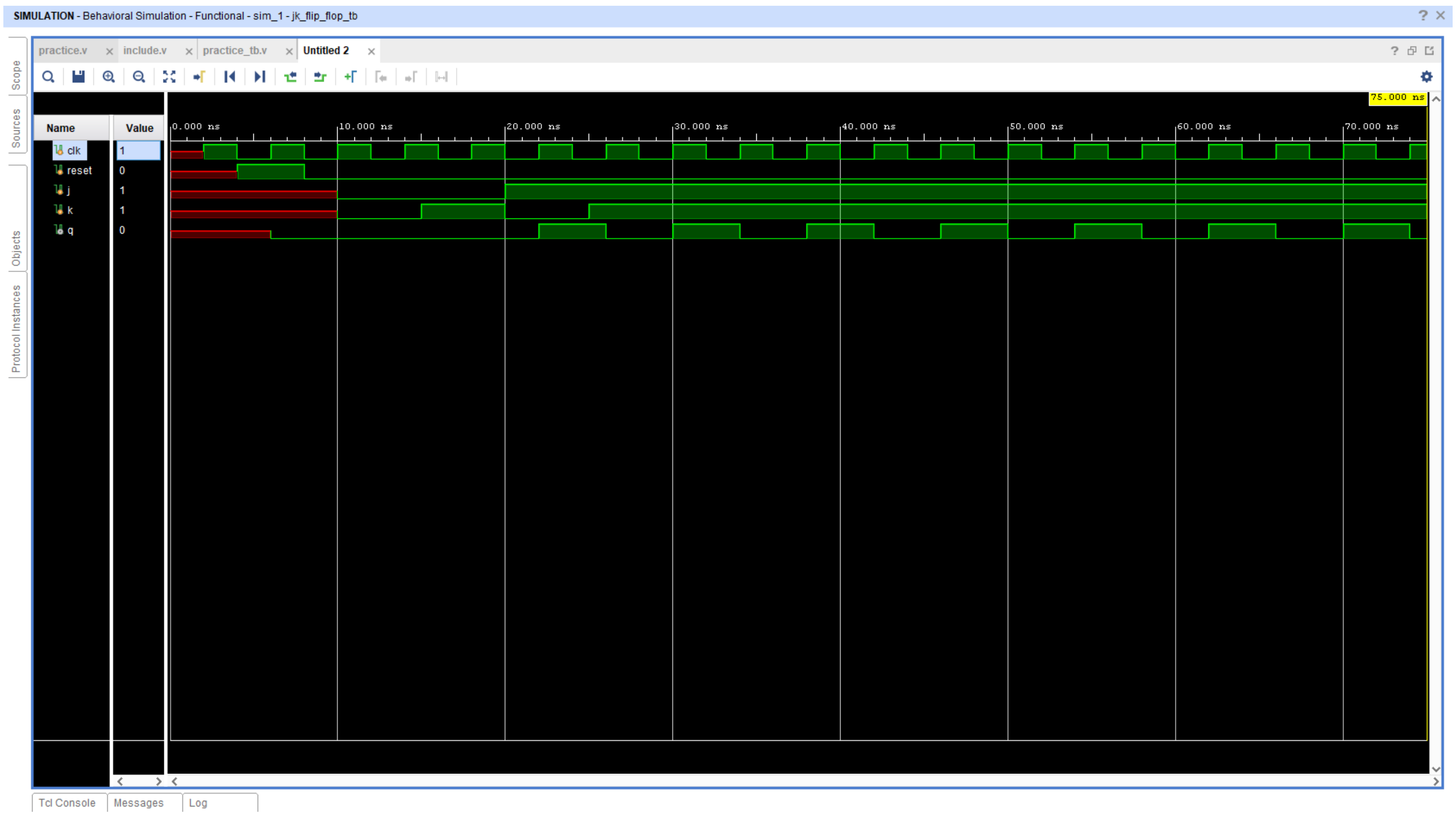
TESTBENCH: -

```
module jk_flip_flop_tb;

reg clk,reset,j,k;

wire q;

jk_flip_flop DUT(clk,reset,j,k,q);


always begin

#2 clk = 1'b1;

#2 clk = ~clk;

end

initial begin

@(negedge clk)

reset = 1'b1;

@(negedge clk)

reset = 1'b0;

#2;

{j,k} = 2'b00;

#5;

{j,k} = 2'b01;

#5;

{j,k} = 2'b10;
```

```
#5;
{j,k} = 2'b11;


#50 $finish;
end
initial
$monitor("FOR j:%b and k:%b, the output is q:%b", j,k,q);
endmodule
```

Output Snippet #5:-

```
Time resolution is 1 ps
FOR j:x and k:x, the output is q:x
FOR j:x and k:x, the output is q:0
FOR j:0 and k:0, the output is q:0
FOR j:0 and k:1, the output is q:0
FOR j:1 and k:0, the output is q:0
FOR j:1 and k:0, the output is q:1
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
FOR j:1 and k:1, the output is q:1
FOR j:1 and k:1, the output is q:0
$finish called at time : 75 ns
```



----------##----------

Snippet #6: - (Dual-Ram RTL (File Operations))

RTL: -

```
module dual_ram(clk,rst,din,we,re,we_addr,re_addr,dout);

  input clk,we,re,rst;
  input  [3:0]we_addr,re_addr;
  input  [7:0]din;
  output reg [7:0]dout;
  integer i;
  parameter dp=16,wd=8,adrs=4;
  reg[(wd-1):0] mem[(dp-1):0];

  always@(clk,din,we,re,we_addr,re_addr)
  begin
    if(rst)
    begin
    for(i=0;i<16;i=i+1)
    mem[i]<=0;
    dout<=0;
    end

    else
    begin
    if(we)
    begin
    mem[we_addr]<=din;
    end

    if(re)
    begin
    dout<=mem[re_addr];
    end
    end
  end
  endmodule
```

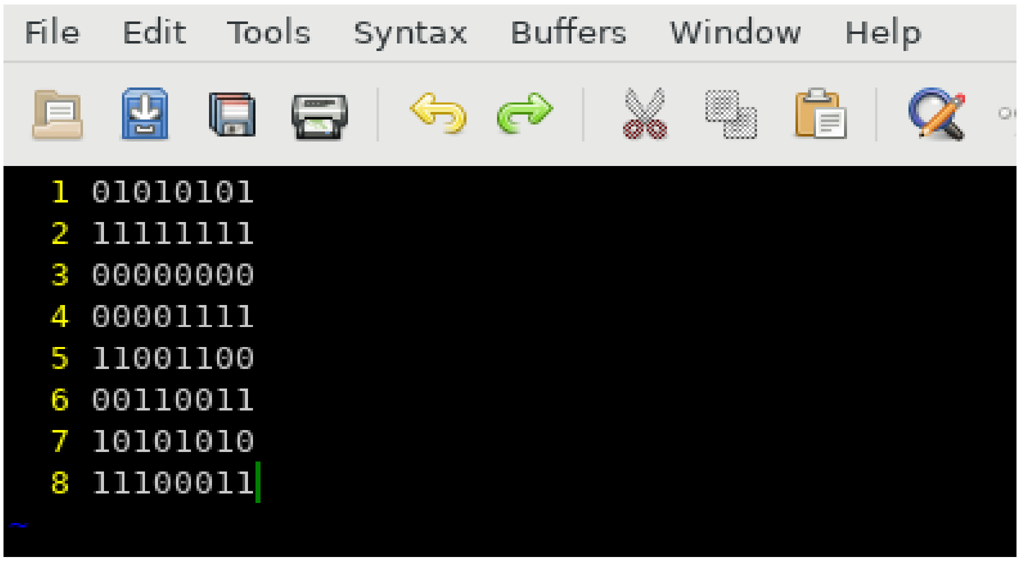
TESTBENCH: -

```
module File_Operation;

    reg clk,we,re,rst;
    reg [3:0]we_addr,re_addr;
    reg [7:0]din;
    wire [7:0]dout;
    integer l,chanel_1;
    parameter cycle=10;

dual_ram DUT(clk,rst,din,we,re,we_addr,re_addr,dout);

  always
    begin
    #(cycle/2);
    clk=1'b1;
    #(cycle/2)
    clk=1'b0;
    end

  task reset();
  begin
  @(negedge clk)
  rst=1'b1;
  @(negedge clk)
  rst=1'b0;
  end
  endtask

  initial
  begin
       reset();

              begin
          we=1; re=0;
                @(negedge clk)
                begin
          $readmemb("text.txt",DUT.mem);
          chanel_1=$fopen("optext.out");
          for(l=0;l<8;l=l+1)
          $fdisplay(chanel_1,"memory[%d]=%b",l,DUT.mem[l]);
          $fclose(chanel_1);
          end
              end
  end

initial
begin
$dumpfile("file.vcd");
```

```
$dumpvars();
#2000 $finish;
end

endmodule
```

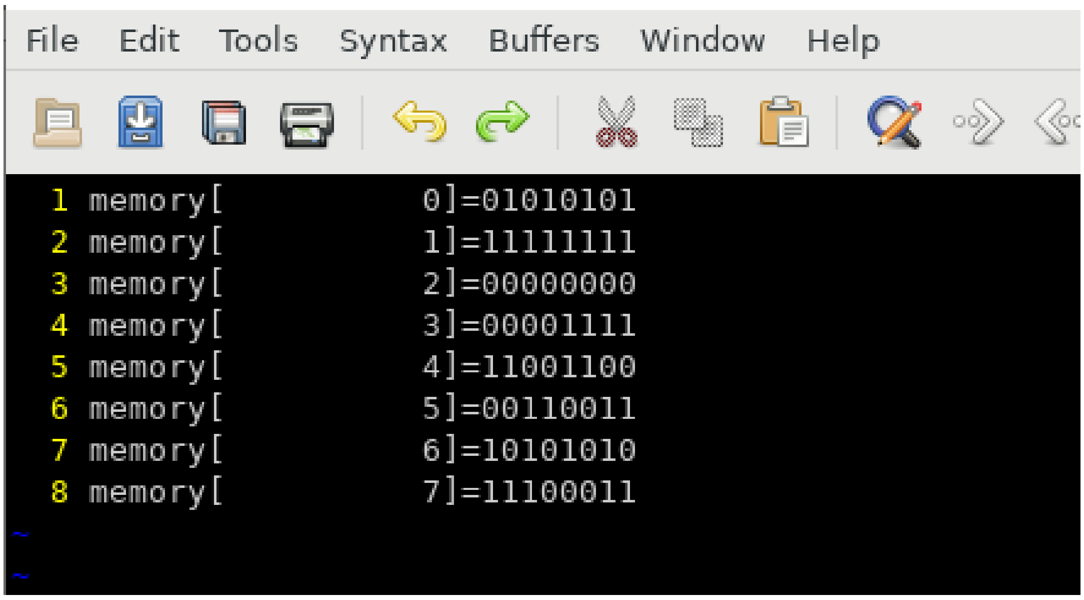
<u>Text file in SIM folder "text.txt"</u>

```
01010101
11111111
00000000
00001111
11001100
00110011
10101010
11100011
```



```
01010101
11111111
00000000
00001111
11001100
00110011
10101010
11100011
~
```

<u>Output file in SIM folder "optext.out"</u>



```
memory[          0]=01010101
memory[          1]=11111111
memory[          2]=00000000
memory[          3]=00001111
memory[          4]=11001100
memory[          5]=00110011
memory[          6]=10101010
memory[          7]=11100011
~
~
```

----------##----------

----------##----------

SYNTHESIS CODING STYLES.

[1]. Combinational Circuit Modelling using always

RTL: -

```
module decoder(in,out);
  input [2:0] in;
output reg [7:0] out;

always @ (in)
begin
 out = 0;  //###___if we don't initialize the out value to logic '0', we might  get a latch.___###
 case (in)
    3'b001 : out = 8'b0000_0001;
    3'b010 : out = 8'b0000_0010;
    3'b011 : out = 8'b0000_0100;
    3'b100 : out = 8'b0000_1000;
    3'b101 : out = 8'b0001_0000;
    3'b110 : out = 8'b0100_0000;
              3'b111 : out = 8'b1000_0000;
 endcase
 end

endmodule
```

TESTBENCH: -

```
module decoder_tb;
reg [2:0] in;
wire [7:0] out;
integer i;
```

```
  decoder DUT(in,out);


  initial begin


    for(i=0;i<8;i=i+1) begin

      in = i;

      #5;

    end


  end

initial

  $monitor("For input:%b, Output is:%b",in,out);


endmodule
```

OUTPUT: -

```
# KERNEL: For input:000, Output is:00000000
```

[-if'out' is not initialized→# KERNEL: For input:000, Output is:xxxxxxxx]

```
# KERNEL: For input:001, Output is:00000001

# KERNEL: For input:010, Output is:00000010

# KERNEL: For input:011, Output is:00000100

# KERNEL: For input:100, Output is:00001000

# KERNEL: For input:101, Output is:00010000

# KERNEL: For input:110, Output is:01000000

# KERNEL: For input:111, Output is:10000000

# KERNEL: Simulation has finished. There are no more test vectors to simulate.
```

[2]. Combinational Circuit Modelling taking care of sensitivity list and missing case values.

RTL: -

```
module mux_41 (a, b, c, d, sel, out);

output reg out;

input a, b, c, d;
input [1:0] sel;


always @(sel or a or b or c or d)
begin
 case (sel)
 2'd0: out = a;
 2'd1: out = b;
 2'd3: out = d;
 default: out = 0;


 endcase
end
endmodule
```

TESTBENCH: -

```
module mux_41_tb;
reg a, b, c, d;
  reg[1:0] sel;
wire out;
integer i;


  mux_41 DUT(a, b, c, d, sel, out);


  initial begin
    sel = 0;
    a = 1;
    b = 0;
    c = 1;
    d = 1;
    #5;
    for(i=0;i<4;i=i+1) begin
      sel = i;
      #5;
```

```
   end


  end
initial
  $monitor("For Select input %b, Output is:%b",sel,out);


endmodule
```

OUTPUT: -

[-Although we didn’t consider ‘sel’ value of ‘2’, it is giving logic ‘0’ at the output for sel=2. This is because we are mentioning the default assignment value in the case.

- Also, we have included all the signals in the sensitivity list of the always block to avoid missing any transition for a particular signal.]

```
# KERNEL: For Select input 00, Output is:1
# KERNEL: For Select input 01, Output is:0
# KERNEL: For Select input 10, Output is:0
# KERNEL: For Select input 11, Output is:1
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
```

[3]. Priority logic Vs. Parallel logic.

RTL#1: -

```
module encoder(x,y);
input[3:0] x;
output reg[1:0]y;

always @(x)
begin : encode
if (x == 4'b0001) y = 2'b00;
else if (x == 4'b0010) y = 2'b01;
else if (x == 4'b0100) y = 2'b10;
else if (x == 4'b1000) y = 2'b11;

else y = 2'bxx;

end


endmodule
```

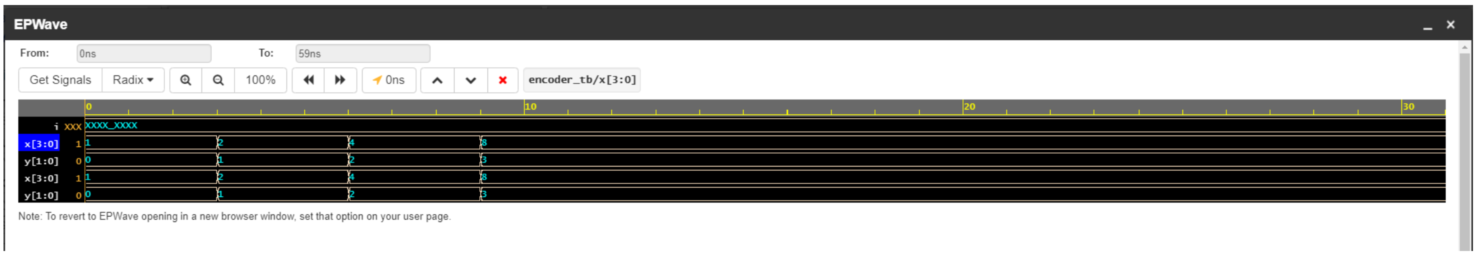
TESTBENCH#1: -

```
module encoder_tb;

reg [3:0] x;

wire [1:0]y;

encoder DUV(x,y);

initial begin

x = 1;

#3;

x = 2;

#3;

x = 4;

#3;

x = 8;

#50 $finish;

end

initial

$monitor("for input x:%b, output is:%b",x,y);

initial

begin

$dumpfile("encoder.vcd");

$dumpvars();

end

endmodule
```

OUTPUT#1: -

[This style of cascaded logic may adversely affect the performance of the circuit.]

```
# KERNEL: for input x:0001, output is:00

# KERNEL: for input x:0010, output is:01

# KERNEL: for input x:0100, output is:10

# KERNEL: for input x:1000, output is:11
# RUNTIME: Info: RUNTIME_0068 testbench.sv (18): $finish called.

# KERNEL: Time: 59 ns
```



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

RTL#2: -

```
module encoder(x,y);

input[3:0] x;

output reg[1:0]y;


  always @(*)begin

  y = 0;

case(x)


4'b0001: y = 2'b00;

4'b0010: y = 2'b01;

4'b0100: y = 2'b10;

4'b1000: y = 2'b11;


default: y = 2'bxx;


endcase

end


endmodule
```

TESTBENCH#2: -

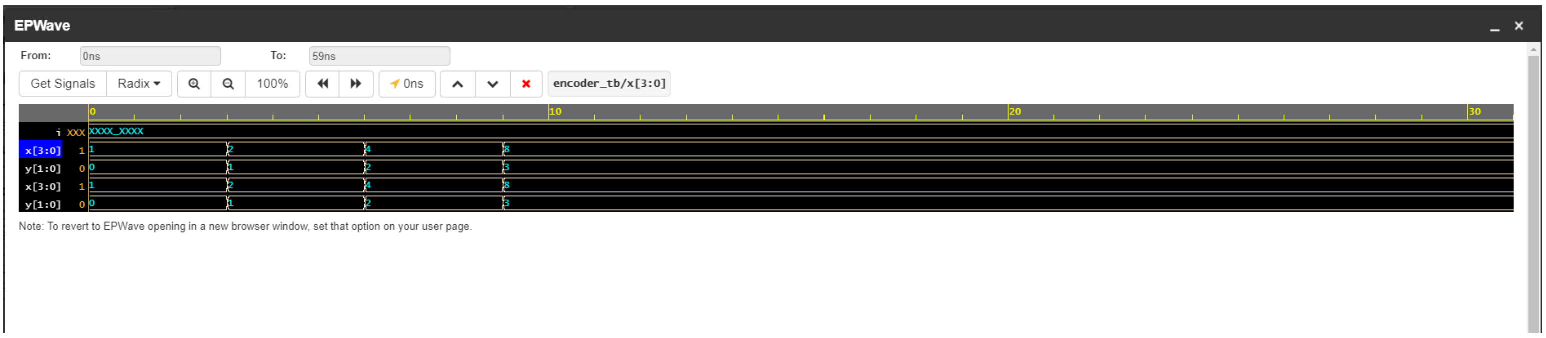
```
module encoder_tb;


reg [3:0] x;

wire [1:0]y;

encoder DUV(x,y);


initial begin
```

```
x = 1;

#3;

x = 2;

#3;

x = 4;

#3;

x = 8;


#50 $finish;


end

initial

$monitor("for input x:%b, output is:%b",x,y);


initial

begin

$dumpfile("encoder.vcd");

$dumpvars();

end


endmodule
```

<u>OUTPUT#2: -</u>

[To avoid “priority logic” use the case construct]

```
# KERNEL: for input x:0001, output is:00

# KERNEL: for input x:0010, output is:01

# KERNEL: for input x:0100, output is:10

# KERNEL: for input x:1000, output is:11

# RUNTIME: Info: RUNTIME_0068 testbench.sv (18): $finish called.
```



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

[4]. Avoiding Latch formation.

RTL: -

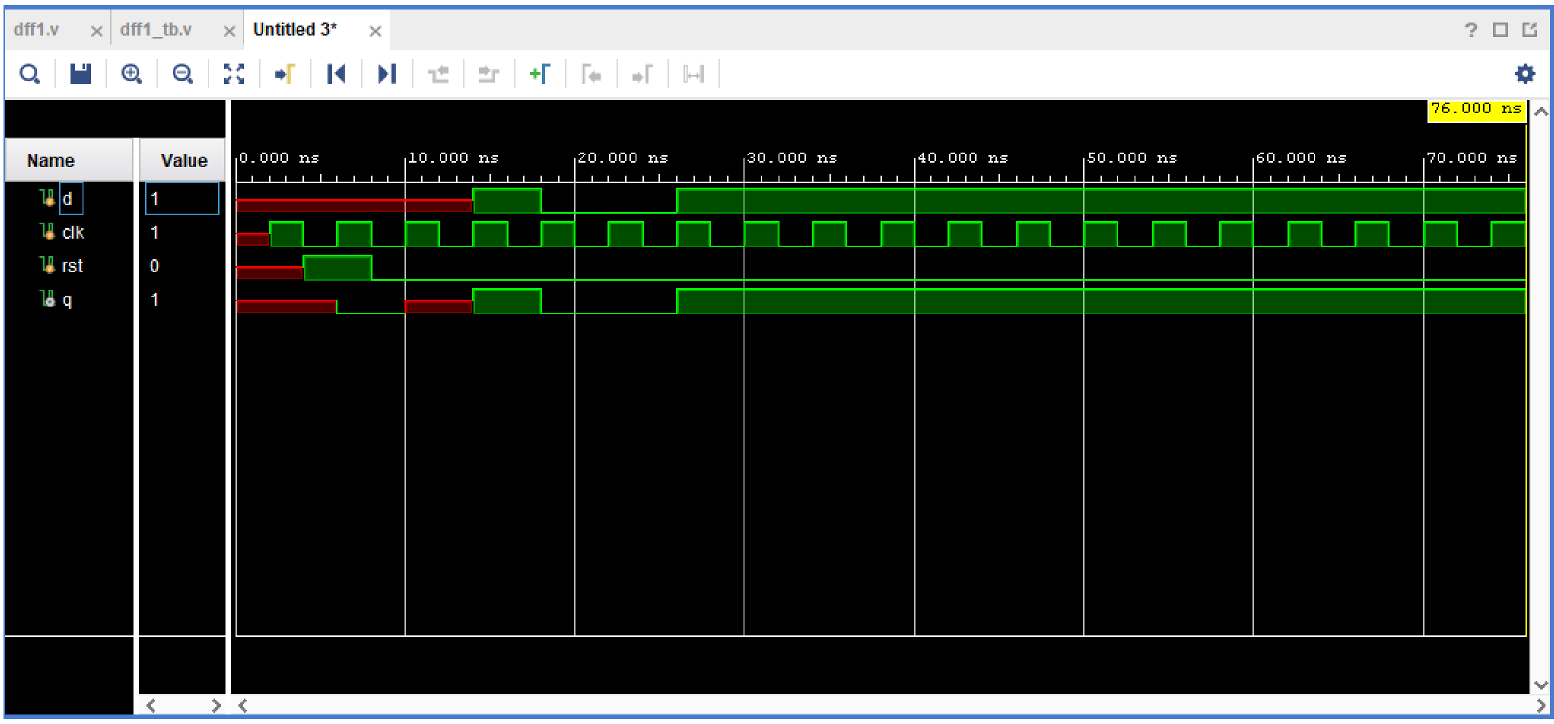
```
module dff(q, d, clk, set, rst);


input d, clk, set, rst;

output reg q;

always @(posedge clk)


if (rst)

q <= 0;

else if (set)

q <= 1;

else

q <= d;  //IF WE DONT MENTION THIS ELSE CONDITION< WE WILL ENCOUNTER A LATCH


endmodule
```

TESTBENCH: -

```
module dff_tb;

reg d, clk, set, rst;

wire q;

dff DU(q, d, clk, set, rst);


always begin

  #2clk=1'b1;
  #2clk=1'b0;
end

initial begin
  @(negedge clk)
  rst = 1;
  @(negedge clk)
  rst = 0;

  #5;
  @(posedge clk)d = 1;
  @(posedge clk)d = 0;
  @(posedge clk)d = 0;
  @(posedge clk)d = 1;
#50 $finish;
end

initial
  $monitor("for input d:%b, output is:%b",d,q);
initial
begin
  $dumpfile("dff.vcd");
$dumpvars();
end
endmodule
```

<u>OUTPUT: -</u>

```
Time resolution is 1 ps
for input d:x, output is:x
for input d:1, output is:1
for input d:0, output is:0
for input d:1, output is:1
$finish called at time : 76 ns
```



[5]. Race Condition.

#1_READ-WRITE RACE.

```
module test;

reg a,b,clk;


  always begin

    #3clk = 1'b1;

    #3clk = ~clk;

  end

  always @(posedge clk) begin

            a <= 1;

  end

  always @(posedge clk) begin

            b <= a;

    #50 $finish;

  end

endmodule
```

[If we didn't opt for non-blocking assignments for 'a' and 'b', one always block value is assigned to 'a' while simultaneously its value is assigned to 'b', which means 'a' is writing and read parallel. This type of race condition is solved by using nonblocking assignment for both 'a' and 'b'.]

#2_WRITE-WRITE RACE.

```
module test;
reg a,b,clk;
  always begin
    #3clk = 1'b1;
    #3clk = ~clk;
  end
always @(posedge clk)
a = 1;
always @(posedge clk)
a = 5;
    #50 $finish;
  end
endmodule
```

[In this case, we can see that one block is updating value of ‘a’ while another one is doing the same thing with a different value. So there is an ambiguity as to which ‘always’ block should execute first. This is nondeterministic and it completely depends on the simulator algorithm.]

----------##----------

----------##----------

<u>FSM Assignment</u>

<u>Snippet #1: - (SEQUENCE DETECTOR(1011) FSM CODING STYLE #1)</u>

RTL: -

```
module sequence(clk, reset, din, dout);

input clk, reset, din;
output dout;

parameter IDLE = 5'b00001,
        S1 = 5'b00010,
        S2 = 5'b00100,
        S3 = 5'b01000,
        S4 = 5'b10000;

reg [4:0]next_state, state;
```

```
//PRESENT STATE LOGIC (SEQUENTIAL)
always@(posedge clk) begin

if(reset)
state<= IDLE;
else
state<= next_state;

end

always@(*) begin
case(state)

IDLE: begin
    if(din==1)
    next_state = S1;
    else
    next_state = IDLE;
    end

S1:   begin
    if(din==0)
    next_state = S2;
    else
    next_state = S1;
    end

S2:   begin
    if(din==1)
    next_state = S3;
    else
    next_state = IDLE;
    end
```

```
S3:   begin
      if(din==1)
      next_state = S4;
      else
      next_state = S2;
      end

S4:   begin
      if(din==1)
      next_state = S1;
      else
      next_state = S2;
      end

endcase
end

assign dout = (state==S4) ? 1:0;

endmodule
```

TESTBENCH: -

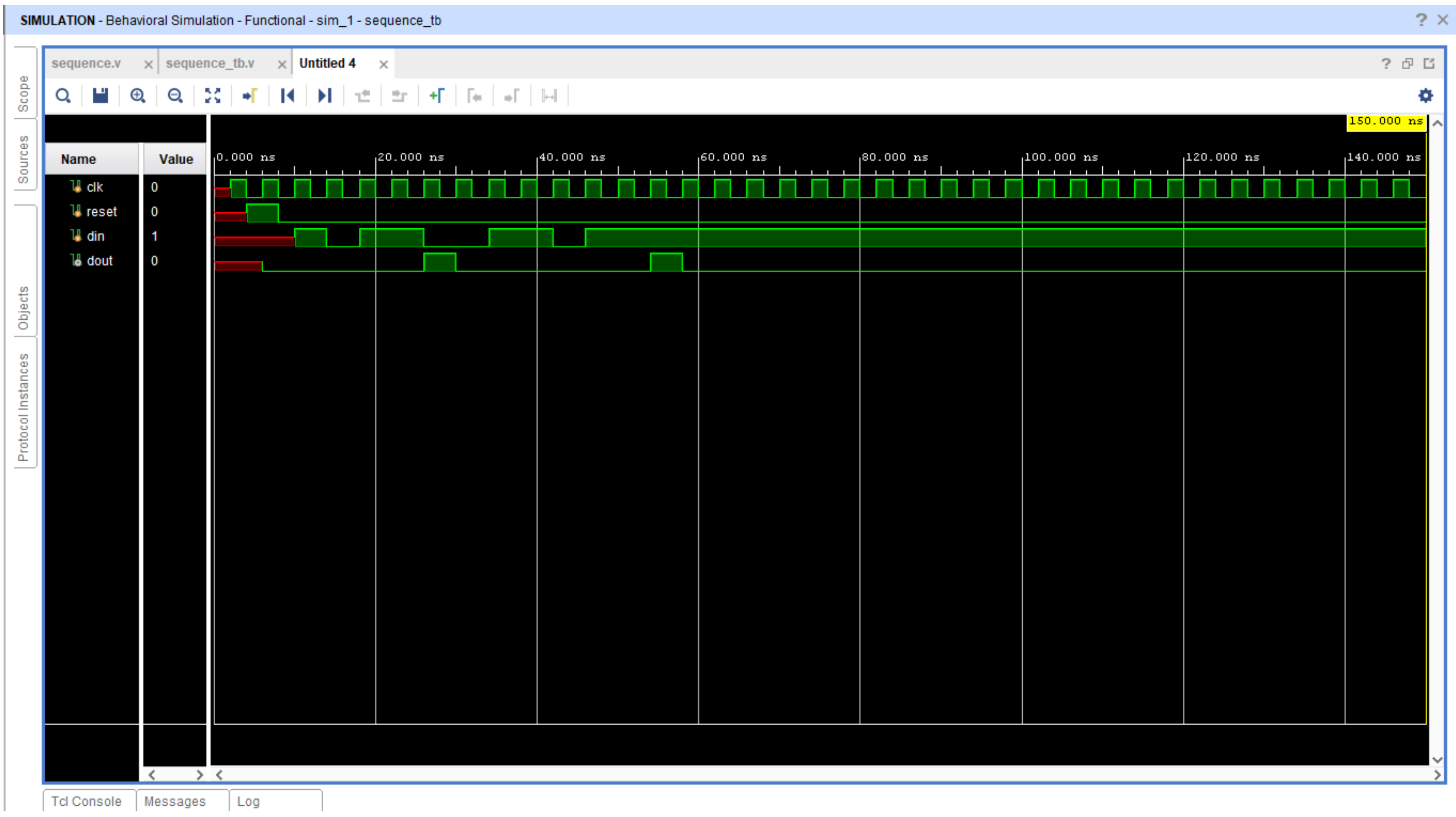
```
module sequence_tb;

reg clk, reset, din;
wire dout;

sequence DUT(clk, reset, din, dout);


always begin
#2clk = 1'b1;
#2 clk = ~clk;
end

initial begin
```

```
@(negedge clk)
reset = 1'b1;
@(negedge clk)
reset = 1'b0;

   @(posedge clk) din <= 1;
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 1;                              // Sequence is detected.
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 1;

#100 $finish;
end
initial
$monitor($time,"For input:%0b, Output is %0b",din, dout);

endmodule
```

<u>Output Snippet #1:-</u>

```
Time resolution is 1 ps
                 0For input:x, Output is x
                 6For input:x, Output is 0
                10For input:1, Output is 0
                14For input:0, Output is 0
                18For input:1, Output is 0
                26For input:0, Output is 1
                30For input:0, Output is 0
                34For input:1, Output is 0
                42For input:0, Output is 0
```

```
46For input:1, Output is 0

54For input:1, Output is 1

58For input:1, Output is 0

$finish called at time : 150 ns
```



----------##----------

<u>Snippet #2: - (SEQUENCE DETECTOR(0101/1011) FSM CODING STYLE #2)</u>

RTL: -

```
module sequence(clk , reset, din, dout);

input clk, reset, din;

output  reg dout;

parameter IDLE = 4'b0000,

        S0 = 4'b0001,

        S1 = 4'b0010,

        S2 = 4'b0011,

        S3 = 4'b0100,

        S4 = 4'b0101,

        S5 = 4'b0110,

        S6 = 4'b0111,

        S7 = 4'b1000;
```

```
reg [3:0]next_state, state;

always@(posedge clk) begin

if(reset)
state<= IDLE;
else
state<= next_state;
end

always@(*) begin
case(state)

IDLE: begin
    dout = 0;
    if(din==0)
    next_state = S0;
    else
    next_state = S4;
    end

S0:   begin
    dout = 0;
    if(din==1)
    next_state = S1;
    else
    next_state = S0;
    end

S1:   begin
    if(din==0)
    next_state = S2;
    else
    next_state = S4;
    end
```

```
S2:   begin
      dout = 0;
      if(din==1)
      next_state = S3;
      else
      next_state = S0;
      end

S3:   begin
      dout = 1;
      if(din==1)
      next_state = S7;
      else
      next_state = S5;
      end

S4:   begin
      dout = 0;
      if(din==0)
      next_state = S5;
      else
      next_state = S4;
      end

S5:   begin
      dout = 0;
      if(din==1)
      next_state = S6;
      else
      next_state = S0;
      end

S6:   begin
      dout = 0;
```

```
    if(din==1)

    next_state = S7;

    else

    next_state = S2;

    end


S7:   begin

    dout = 1;

    if(din==1)

    next_state = S4;

    else

    next_state = S5;

    end


   endcase

  end


  endmodule
```

TESTBENCH: -

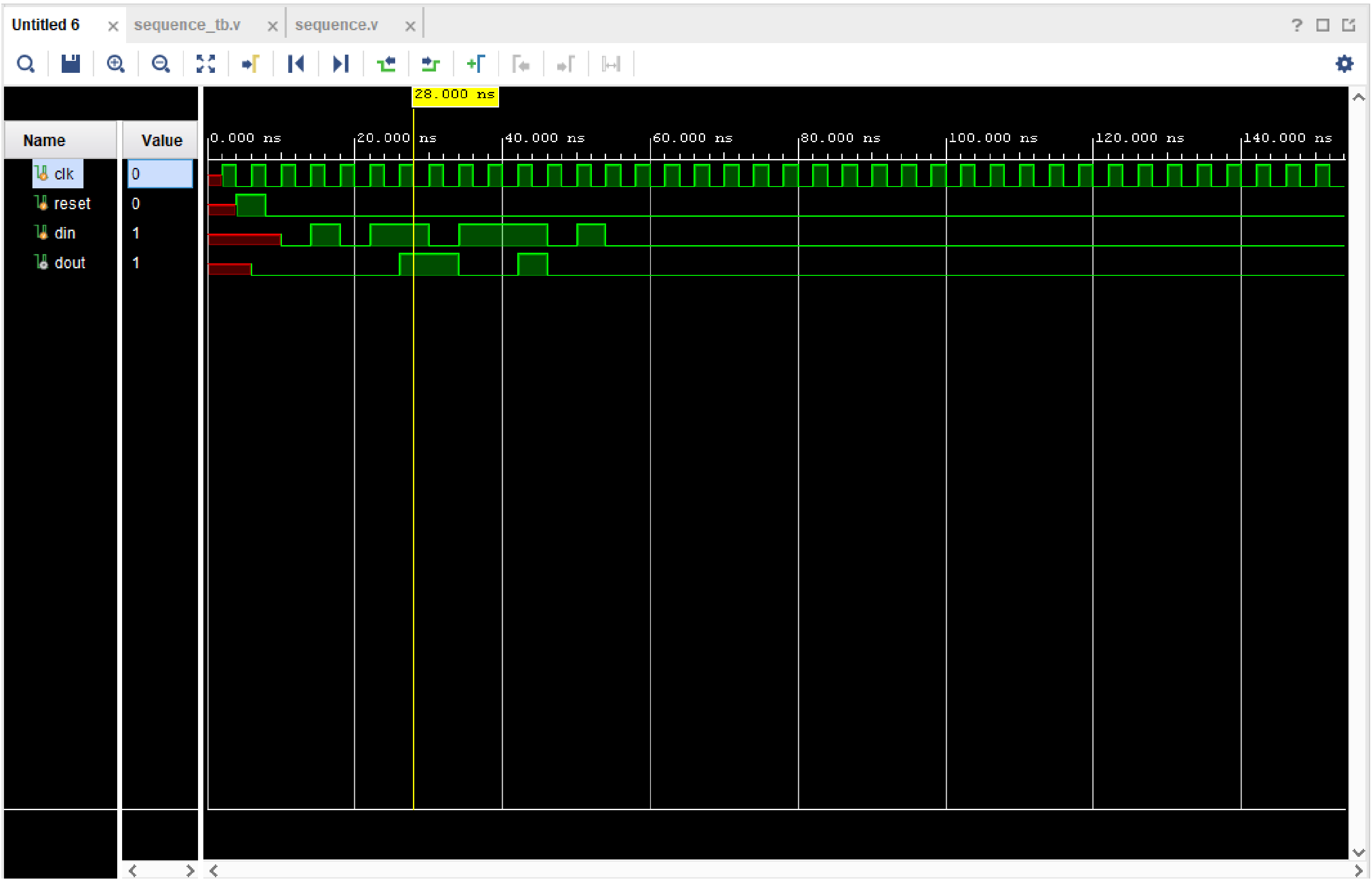
```
module sequence_tb;

reg clk, reset, din;

wire dout;

sequence DUT(clk, reset, din, dout);

always begin

#2clk = 1'b1;

#2 clk = ~clk;

end


initial begin

@(negedge clk)

reset = 1'b1;

@(negedge clk)
```

```
reset = 1'b0;


   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;                                  // Sequence is detected.
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 0;
   @(posedge clk) din <= 1;
   @(posedge clk) din <= 0;


#100 $finish;
end
initial
$monitor($time,"For input:%0b, Output is %0b",din, dout);


endmodule
```

<u>Output Snippet #2:-</u>

```
             Time resolution is 1 ps
             0For input:x, Output is x
             6For input:x, Output is 0
            10For input:0, Output is 0
            14For input:1, Output is 0
            18For input:0, Output is 0
            22For input:1, Output is 0
            26For input:1, Output is 1
            30For input:0, Output is 1
            34For input:1, Output is 0
            42For input:1, Output is 1
```

```
46For input:0, Output is 0

50For input:1, Output is 0

54For input:0, Output is 0

$finish called at time : 154 ns
```

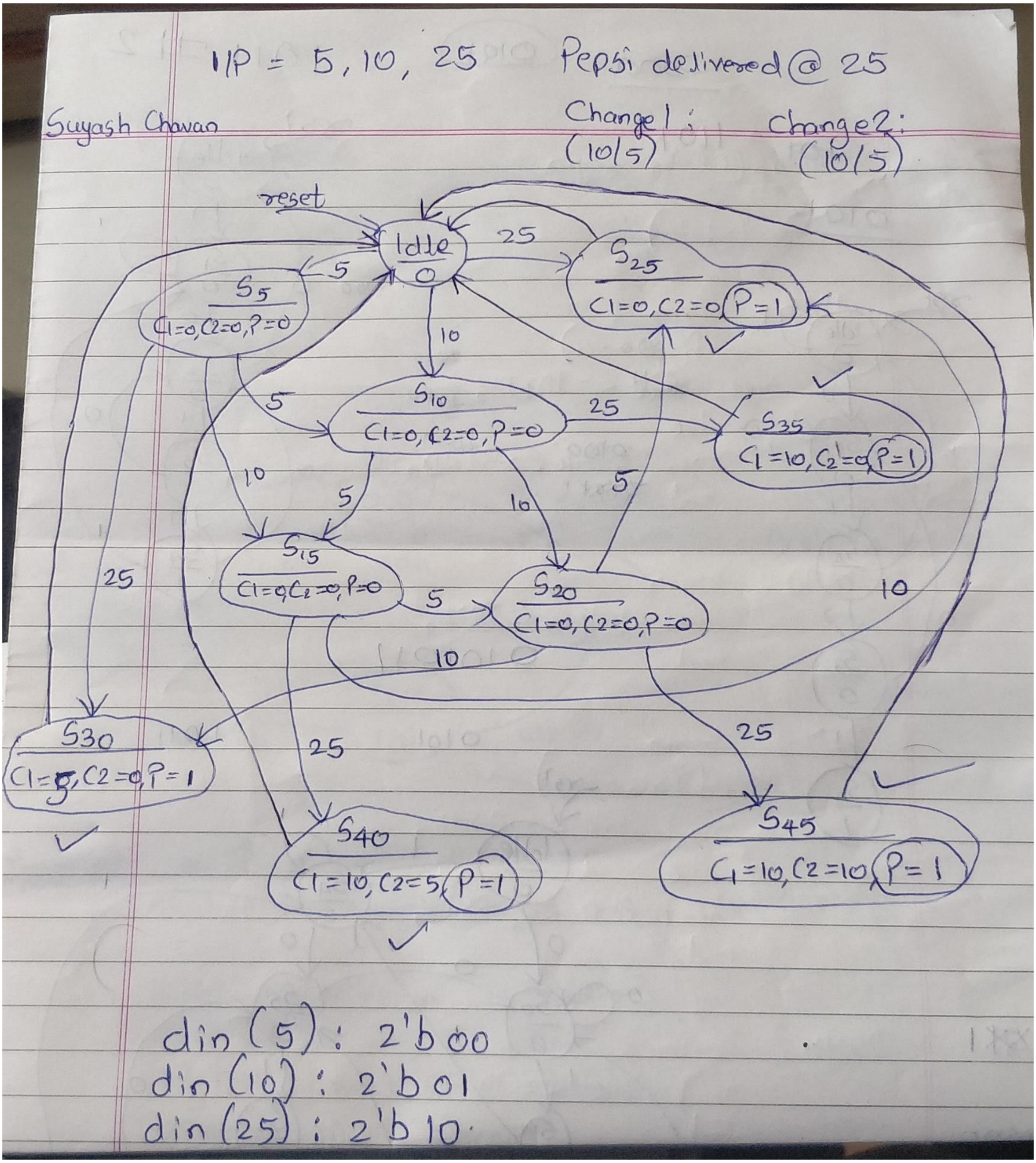


----------##----------

You are hired by Pepsi to design their next generation Pepsi (Generation Next) machine, the reason being that Pepsi is going to drop the price of their product to a quarter(1/4th of a dollar = 25 cents). The new machine will accept the industry standard nickel=5 cents, dime=10 cents and quarter=25 cents . Design a totally synchronous FSM and write the RTL code for the same. Assume the only inputs to the system are the type of change put in (nickel, dime, or quarter). Your outputs will be two change amount(Change 1 & Change 2), and a Pepsi release signal that goes high when enough change has been put in.

Example: If the total = 40 cents then Output : Pepsi = 1, Change 1 = 10 & Change 2 = 5 (Order of change can be interchangeable)

FSM for the vending machine:-



din (5) : 2'b 00
din (10) : 2'b 01
din (25) : 2'b 10.

RTL: -

```
module vending_fsm(din,clock,reset,p,c1,c2);
  input [1:0]din;
  input clock,reset;
  output reg p;
  output reg[3:0]c1,c2;

   parameter                                              IDLE=4'b0000,
                                                                  S5=4'b0001,
                                                    S10=4'b0010,
                                                                  S15=4'b0011,
                                                                  S20=4'b0100,
                                                                  S25=4'b0101,
                                                           S30=4'b0110,
                                                                  S35=4'b0111,
                                                        S40=4'b1000,
                                                                  S45=4'b1001;

 reg [3:0] next_state, state;

always@(posedge clock)begin
                                                  if(reset)
                                                  state<=IDLE;
                                                 else
                                                 state<=next_state;
                                                end

  always@(*)begin

    case (state)
  IDLE:
                                                           if(din==2'b00)
                                                           next_state=S5;
                                                else if(din==2'b01)
      next_state=S10;
```

```
                                                                       else if(din==2'b10)

                                                               next_state=S25;

                                                                       else

                                                               next_state=IDLE;


 S5:

     if (din==2'b00)

                                                                        next_state=S10;

                                                                       else if(din==2'b01)

     next_state=S15;

                                                                       else if(din==2'b10)

                                                                       next_state=S30;


                                                            else

                                                            next_state=IDLE;


S10:

     if (din==2'b00)

     next_state=S15;

                                                                       else if(din==2'b01)

     next_state=S20;

                                                                       else if(din==2'b10)

                                                                       next_state=S35;


     else

     next_state=IDLE;


S15:

     if (din==2'b00)

     next_state=S20;

                                                                       else if(din==2'b01)

     next_state=S25;

                                                                       else if(din==2'b10)

                                                              next_state=S40;
```

```
       else
      next_state=IDLE;

S20:
       if (din==2'b00)
       next_state=S25;
                                                                          else if(din==2'b01)
       next_state=S30;
                                                                          else if(din==2'b10)
                                                                          next_state=S45;

       else
                                                              next_state=IDLE;


S25     : next_state=IDLE;
S30     : next_state=IDLE;
S35     : next_state=IDLE;
S40     : next_state=IDLE;
S45     : next_state=IDLE;

default : next_state=IDLE;
    endcase
  end



always@(*)
  begin
                                                              p=1'b0; c1=4'd0; c2=4'd0;

    case (state)

       IDLE :begin
                                                                                      p=1'b0; c1=4'd0; c2=4'd0;
                                                                                      end
```

```
                                        S5 : begin
                                                p=1'b0; c1=4'd0; c2=4'd0;
                                                end
                                        S10 : begin
                                                p=1'b0; c1=4'd0; c2=4'd0;
                                                end
                                        S15 : begin
                                                p=1'b0; c1=4'd0; c2=4'd0;
                                                end
                                        S20 : begin
                                                p=2'b0; c1=4'd0; c2=4'd0;
                                                end
                                        S25 : begin
                                                p=1'b1; c1=4'd0; c2=4'd0;
                                                end
                                        S30 : begin
                                                p=1'b1; c1=4'd5; c2=4'd0;
                                                end
                                        S35 : begin
                                                p=1'b1; c1=4'd10; c2=4'd0;
                                                end
                                        S40 : begin
                                                p=1'b1; c1=4'd10; c2=4'd5;
                                                end
                                        S45 : begin
                                                p=1'b1; c1=4'd10; c2=4'd10;
                                                end
                                        default : begin
                                                p=1'b0; c1=4'd0; c2=4'd0;
                                                end
                                        endcase
                                end
endmodule
```

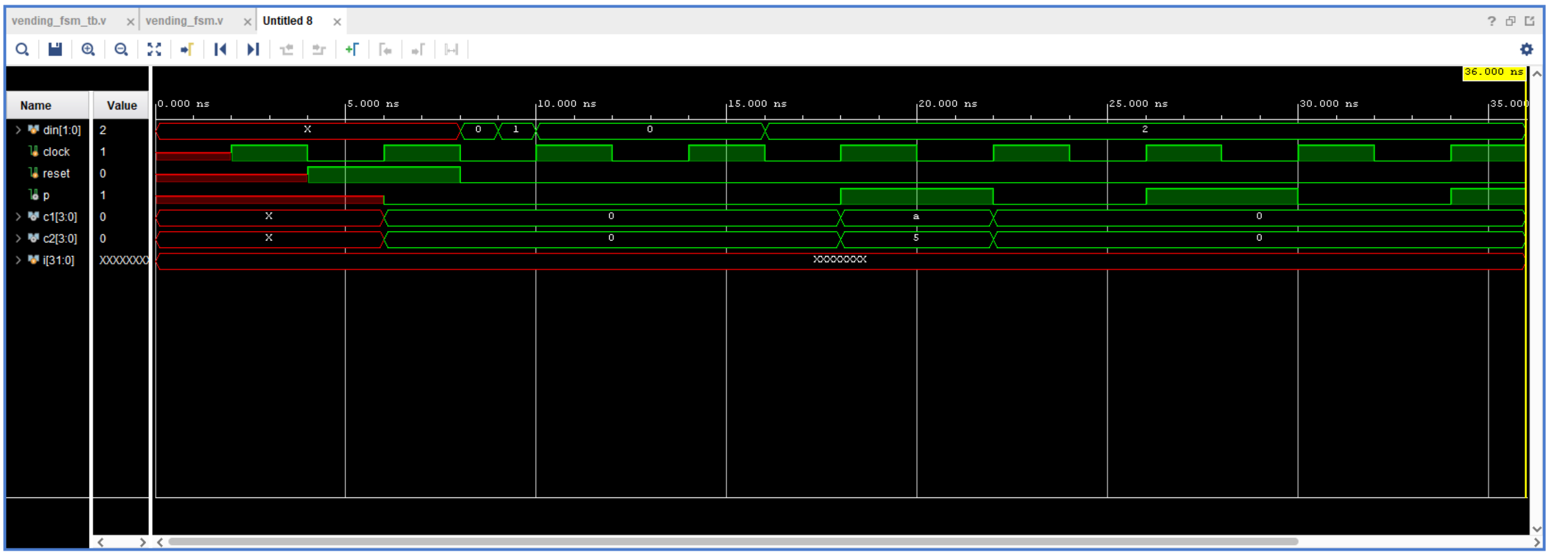
TESTBENCH: -

```
module vending_fsm_tb;

reg [1:0]din;
reg clock,reset;
wire p;
wire[3:0]c1,c2;

integer i;

vending_fsm DUT(din,clock,reset,p,c1,c2);

always begin
#2clock = 1'b1;
#2 clock = ~clock;
end

initial begin

@(negedge clock)
reset = 1'b1;
@(negedge clock)
reset = 1'b0;

din = 2'b00;
#1;
din = 2'b01;
#1;
din = 2'b00;
#1;
din = 2'b00;
#5;
din = 2'b10;

#20 $finish;
end
```

```
initial

$monitor($time,"For input coins:%b, Pepsi delivered:%b",din, p);


endmodule
```

Output: -

```
Time resolution is 1 ps

                 0For input coins:xx, Pepsi delivered:x

                 6For input coins:xx, Pepsi delivered:0

                 8For input coins:00, Pepsi delivered:0

                 9For input coins:01, Pepsi delivered:0

                10For input coins:00, Pepsi delivered:0

                16For input coins:10, Pepsi delivered:0

                18For input coins:10, Pepsi delivered:1

                22For input coins:10, Pepsi delivered:0

                26For input coins:10, Pepsi delivered:1

                30For input coins:10, Pepsi delivered:0

                34For input coins:10, Pepsi delivered:1

$finish called at time : 36 ns
```



----------###----------

Snippet #4: - (SEQUENCE DETECTOR (0110) FSM CODING STYLE #3)

RTL: -

```
module mealy_fsm(clk,reset,din,dout);

input clk,reset,din;
output reg dout;

parameter IDLE=3'b001,
        S1=3'b010,
        S2=3'b011,
        S3=3'b100;

reg[2:0]next_state, state;

always@(posedge clk) begin
if(reset)
state<=IDLE;
else
state<=next_state;
end



always@(*) begin
   next_state=IDLE;

   case(state)

   IDLE: begin
        if(din==0)
        next_state=S1;
        else
        next_state=IDLE;
        end

    S1:  begin
        if(din==1)
        next_state=S2;
        else
        next_state=S1;
        end

    S2:  begin
        if(din==1)
        next_state=S3;
        else
        next_state=S1;
        end

    S3:  begin
        if(din==0)
        next_state=S1;
        else
        next_state=IDLE;
        end

   endcase

  end

always@(posedge clk) begin

   if(reset)
     dout<=0;

   else
     begin
       dout<=0;
```

```
          case(state)


             IDLE: dout<=0;

             S1:   dout<=0;

             S2:   dout<=0;

             S3:   dout<=1;


           endcase

        end

   end


 endmodule
```

TESTBENCH: -

```
module mealy_fsm_tb();

      reg  din,

          clk,

          reset;


      wire dout;


parameter CYCLE=10;


  mealy_fsm DUT(.din(din),

            .clk(clk),

              .reset(reset),

            .dout(dout));




always

begin

#(CYCLE/2);

clk=1'b0;

#(CYCLE/2);

clk=(~clk);
```

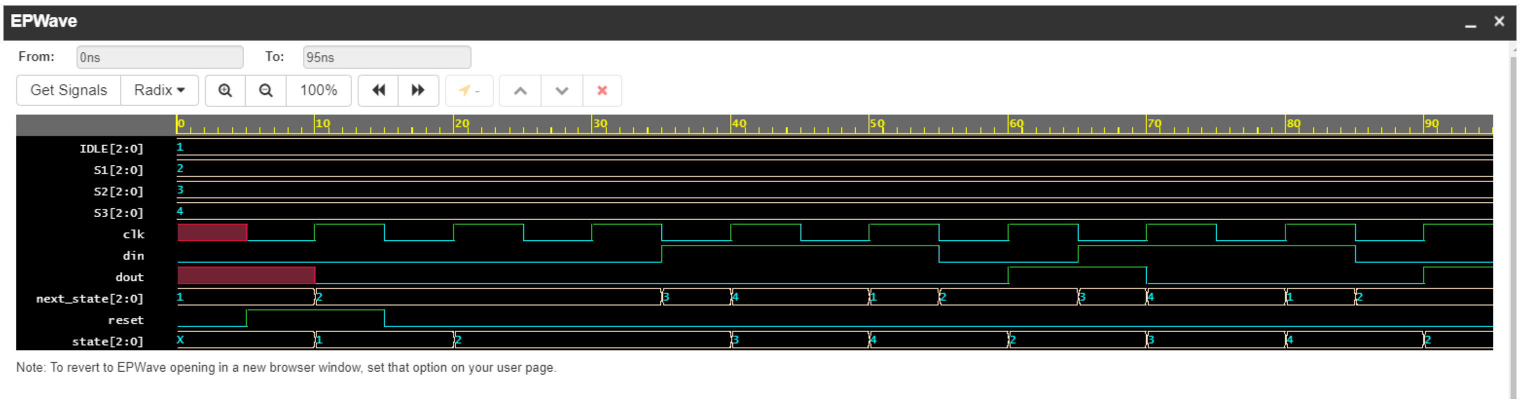
```
end



task initialize;
begin
din=1'b0;
reset=1'b0;

end
endtask

  task delay(input integer i);
    begin
      #i;
    end
  endtask

task RESET;
begin
@(negedge clk)
reset= 1'b1;
@(negedge clk)
reset=1'b0;
end
endtask




task stimulus(input i);
begin
@(negedge clk)
din=i;
end
endtask
```

```
  initial $monitor(" Reset=%b, state=%d, Din=%b, Output Dout=%b",
                 reset,DUT.state,din,dout);

  initial
    begin
      initialize;
      RESET;
      stimulus(0);
      delay(10);

      stimulus(1);
      delay(10);

      stimulus(1);
      delay(10);

      stimulus(0);
      delay(10);

      stimulus(1);
      delay(10);

      stimulus(1);
      delay(10);

      stimulus(0);
delay(10);

      $finish;
    end

initial
begin
```

```
$dumpfile("seq.vcd");

$dumpvars();

end

endmodule
```

Output: -

```
# KERNEL:  Reset=0, state=x, Din=0, Output Dout=x
# KERNEL:  Reset=1, state=x, Din=0, Output Dout=x
# KERNEL:  Reset=1, state=1, Din=0, Output Dout=0
# KERNEL:  Reset=0, state=1, Din=0, Output Dout=0
# KERNEL:  Reset=0, state=2, Din=0, Output Dout=0
# KERNEL:  Reset=0, state=2, Din=1, Output Dout=0
# KERNEL:  Reset=0, state=3, Din=1, Output Dout=0
# KERNEL:  Reset=0, state=4, Din=1, Output Dout=0
# KERNEL:  Reset=0, state=4, Din=0, Output Dout=0
# KERNEL:  Reset=0, state=2, Din=0, Output Dout=1
# KERNEL:  Reset=0, state=2, Din=1, Output Dout=1
# KERNEL:  Reset=0, state=3, Din=1, Output Dout=0
# KERNEL:  Reset=0, state=4, Din=1, Output Dout=0
# KERNEL:  Reset=0, state=4, Din=0, Output Dout=0
# KERNEL:  Reset=0, state=2, Din=0, Output Dout=1
# RUNTIME: Info: RUNTIME_0068 testbench.sv (144): $finish called.
# KERNEL: Time: 95 ns
```



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Snippet #5: - (SEQUENCE DETECTOR (1011) FSM CODING STYLE #2)

RTL: -

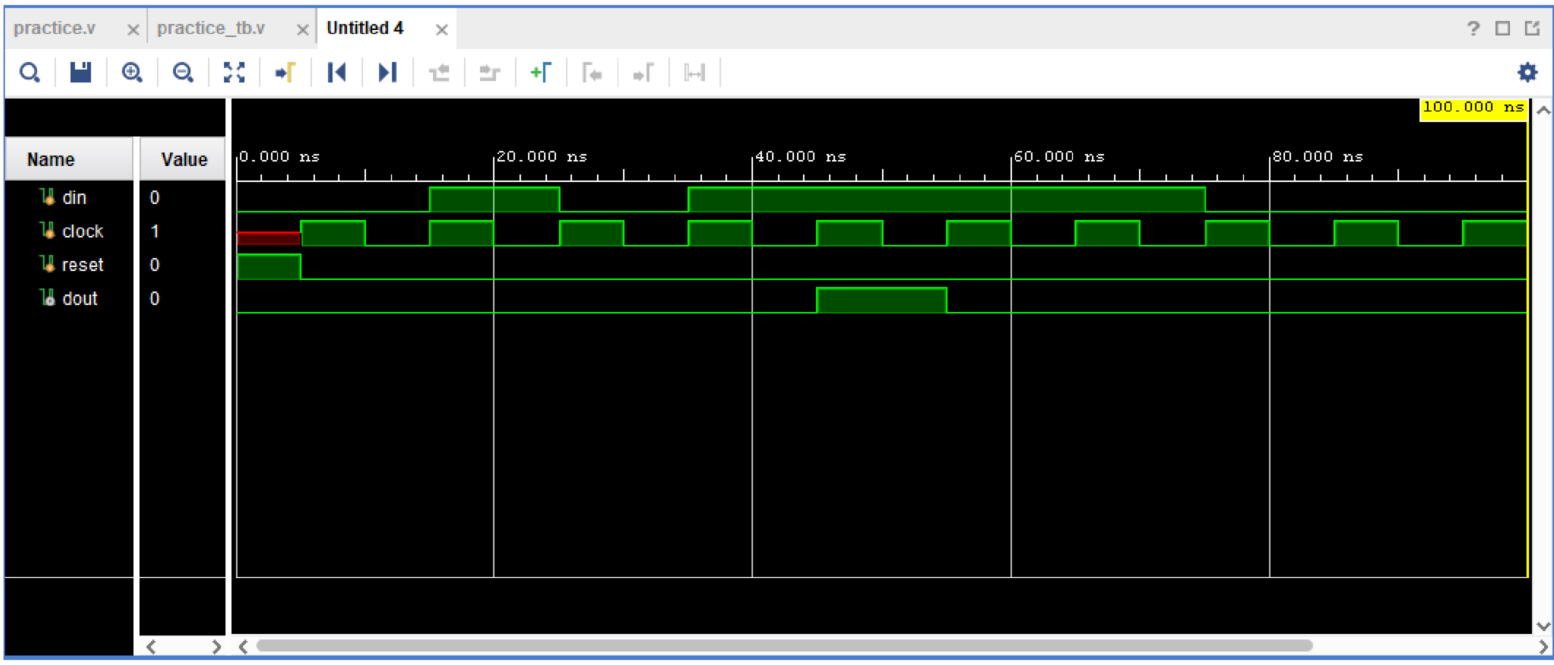
```
module sequence_detector(clock,reset,din,dout);

input clock;

input reset;

input din;

output reg dout;

parameter  S0=3'b000,
  S1=3'b001,
  S2=3'b011,
  S3=3'b010,
  S4=3'b110;
reg [2:0] current_state, next_state;



always @(posedge clock, posedge reset)
begin
 if(reset==1)
 current_state <= S0;
 else
 current_state <= next_state;
end

always @(current_state,din)
begin
 case(current_state)
 S0:begin
  if(din==1)
   next_state = S1;
  else
   next_state = S0;
 end
 S1:begin
  if(din==0)
   next_state = S2;
  else
   next_state = S1;
 end
 S2:begin
  if(din==0)
   next_state = S0;
  else
   next_state = S3;
 end
 S3:begin
  if(din==0)
   next_state = S2;
  else
   next_state = S4;
 end
 S4:begin
  if(din==0)
   next_state = S2;
  else
   next_state = S1;
 end
 default:next_state = S0;
 endcase
end

always @(current_state)
begin
 case(current_state)
 S0:   dout = 0;
 S1:   dout = 0;
 S2:  dout = 0;
 S3:  dout = 0;
 S4:  dout = 1;
 default:  dout = 0;
 endcase
end
endmodule
```

TESTBENCH: -

```
module sequence_detector_tb;

 reg din;
 reg clock;
 reg reset;

 wire dout;

sequence_detector DUT(clock,reset,din,dout);

 always begin
 #5clock = 1;
 #5clock = ~clock;
 end

 initial begin
  din = 0;
  reset = 1;
  #5;
      reset = 0;
  #10;
  din = 1;
  #10;
  din = 0;
  #10;
  din = 1;
  #20;
  din = 1;
  #20;
  din = 0;
  #20;
  din = 0;
  #5 $finish;
 end

  initial
    $monitor($time,"For the input:%b, detector output is %b", din, dout);

endmodule
```

Output: -

```
 0For the input:0, detector output is 0
15For the input:1, detector output is 0
25For the input:0, detector output is 0
35For the input:1, detector output is 0
45For the input:1, detector output is 1
55For the input:1, detector output is 0
75For the input:0, detector output is 0
$finish called at time : 100 ns
```


practice.v
practice_tb.v
Untitled 4
Name
Value
din 0
clock 1
reset 0
dout 0
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns