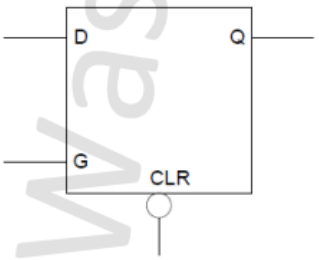
1) Implement Data Latch with active low Clear. Implement a randomized testbench and check the output correctness from the waveform. Do not treat G input as a clock so add a delay (#1) after randomizing the inputs.



Input	Output
CLR, D, G	Q

Truth Table

CLR	G	D	Q
0	X	X	0
1	0	X	Q
1	1	D	D

assignment3.v

```
1   module d_latch ( output reg q,
2                    input d,clr,g);
3       always @(*) begin
4           if (!clr)
5               q <= 0;
6           else if (g)
7               q <= d;
8       end
9
10  endmodule
11
12  module d_latch_ts ();
13      wire q;
14      reg d,clr,g;
15
16      d_latch w (q,d,clr,g);
17
18      initial begin
19          clr = 1; g = 0; d = 0; #1;
20          clr = 0; g = 1; d = 1; #1;
21          clr = 1; g = 1; d = 1; #1;
22          clr = 1; g = 0; d = 0; #1;
23          clr = 1; g = 1; d = 0; #1;
24          clr = 0; g = 1; d = 1; #1;
25          clr = 1; g = 1; d = 1; #1;
26          clr = 1; g = 1; d = 0; #1;
27          repeat (10) begin
28              d = $random;
29              g = $random;
30              #1;
31              end
32
33      end
34
35      initial begin
36          $monitor("q = %b, d = %b, clr = %b g = %b", q,d,clr,g);
37      end
38
39  endmodule
```

```
VSIM 23> run -all
# q = x, d = 0, clr = 1 g = 0
# q = 0, d = 1, clr = 0 g = 1
# q = 1, d = 1, clr = 1 g = 1
# q = 1, d = 0, clr = 1 g = 0
# q = 0, d = 0, clr = 1 g = 1
# q = 0, d = 1, clr = 0 g = 1
# q = 1, d = 1, clr = 1 g = 1
# q = 0, d = 0, clr = 1 g = 1
# q = 1, d = 1, clr = 1 g = 1
# q = 1, d = 1, clr = 1 g = 0
# q = 1, d = 1, clr = 1 g = 1
# q = 0, d = 0, clr = 1 g = 1
# q = 0, d = 1, clr = 1 g = 0
# q = 1, d = 1, clr = 1 g = 1
```

A. Implement T-type (toggle) Flipflop with active low asynchronous reset (forces q to 0 and forces qbar to 1). T-Flipflop has input t, when t input is high the outputs toggle else the output values do not change.
   - Inputs: t, rstn, clk
   - Outputs: q, qbar

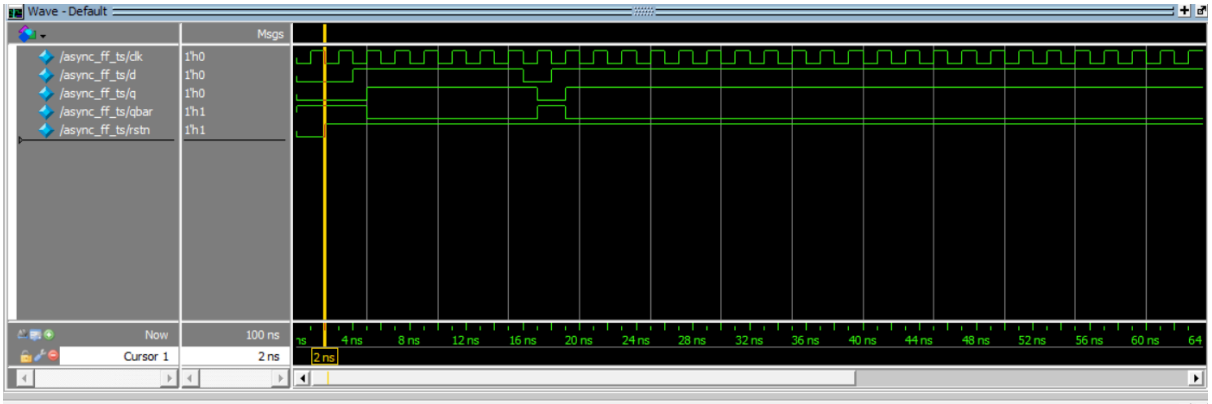
B. Implement Asynchronous D Flip-Flop with Active low reset (forces q to 0 and forces qbar to 1).
   - Inputs: d, rstn, clk
   - Outputs: q, qbar

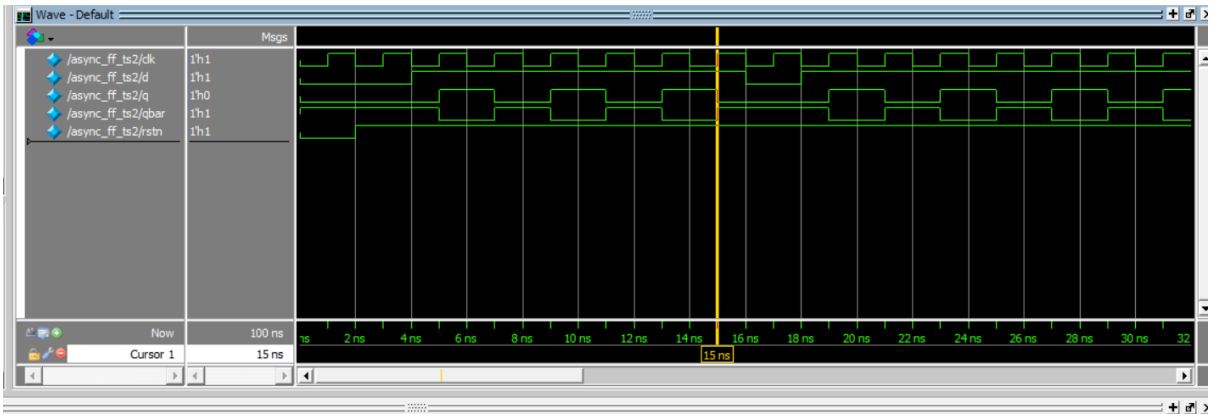
C. Implement a parameterized asynchronous FlipFlop with Active low reset with the following specifications.
   - Inputs: d, rstn , clk
   - Outputs: q, qbar
   - Parameter: FF_TYPE that can take two valid values, DFF or TFF. Default value = “DFF”. Design should act as DFF if FF_TYPE = “DFF” and act as TFF if FF_TYPE = “TFF”. When FF_TYPE equals “DFF”, d input acts as the data input “d”, and when FF_TYPE equals “TFF”, d input acts the toggle input “t”.

D. Test the above parameterized Design using 2 testbenches, testbench 1 that overrides the design with FF_TYPE = “DFF” and the testbench 2 overrides parameter with FF_TYPE = “TFF”
   - Testbench 1 should instantiate the design of part B. as a golden model to check for the output of the parameterized design with FF_TYPE = “DFF”
     - Use a do file to run the simulation
   - Testbench 2 should instantiate the design of part A. as a golden model to check for the output of the parameterized design with FF_TYPE = “TFF”
     - Use a do file to run the simulation

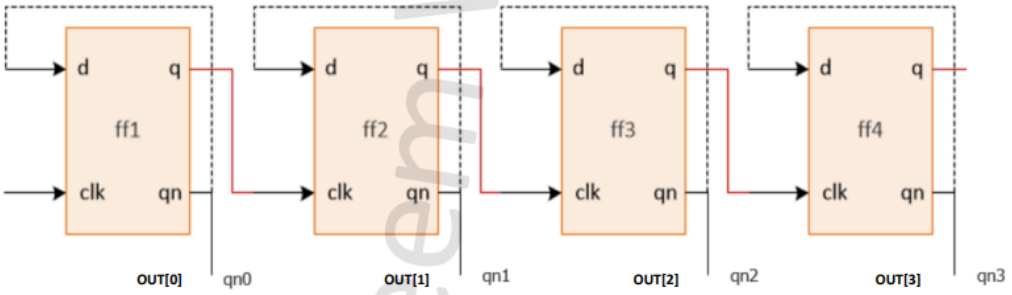
```
module t_flipflop ( output reg q,
                    output qbar,
                    inout t,clk,rstn);
        assign qbar = ~q;
        always @(posedge clk,negedge rstn) begin
            if (!rstn)
                q <= 0;
            else if (t)
                q <= ~q;
        end

endmodule

module d_flipflop ( output reg q,
                    output qbar,
                    input d,clk,rstn);
       assign qbar = ~q;
       always @(posedge clk,negedge rstn) begin
            if (!rstn)
                q <= 0;
            else
                q <= d;
       end
endmodule

module async_ff #(parameter [5*8-1:0] ff_type = "dff")
                 (output reg q,
                  output qbar,
                  input d,clk,rstn);
         assign qbar = ~q;
         always @(posedge clk,negedge rstn) begin
              if (!rstn)
                 q <= 0;
              else begin
                 if (ff_type == "dff")
                     q <= d;
                 else if (ff_type == "tff")
                     if (d)
                     q <= ~q;
              end

         end
endmodule
```

```
module async_ff_ts ();
   wire q,qbar;
   reg d,clk,rstn;
   async_ff #(.ff_type("dff")) tr (q,qbar,d,clk,rstn);
   initial begin
      clk = 0;
      forever #1 clk = ~clk;
   end
   initial begin
      d = 0;
      rstn = 0;
      @(negedge clk);
      rstn = 1;
      repeat (10) begin
          d = $random;
          @(negedge clk);
      end

   end
   initial begin
    $monitor ("clk = %b d = %b rstn = %b q = %b qbar = %b", clk, d, rstn, q, qbar);
   end

endmodule

module async_ff_ts2 ();
   wire q,qbar;
   reg d,clk,rstn;
   async_ff #(.ff_type("tff")) tr (q,qbar,d,clk,rstn);
   initial begin
      clk = 0;
      forever #1 clk = ~clk;
   end
   initial begin
      d = 0;
      rstn = 0;
      @(negedge clk);
      rstn = 1;
      repeat (10) begin
          d = $random;
          @(negedge clk);
      end

   end
   initial begin
    $monitor ("clk = %b d = %b rstn = %b q = %b qbar = %b", clk, d, rstn, q, qbar);
   end

endmodule
```

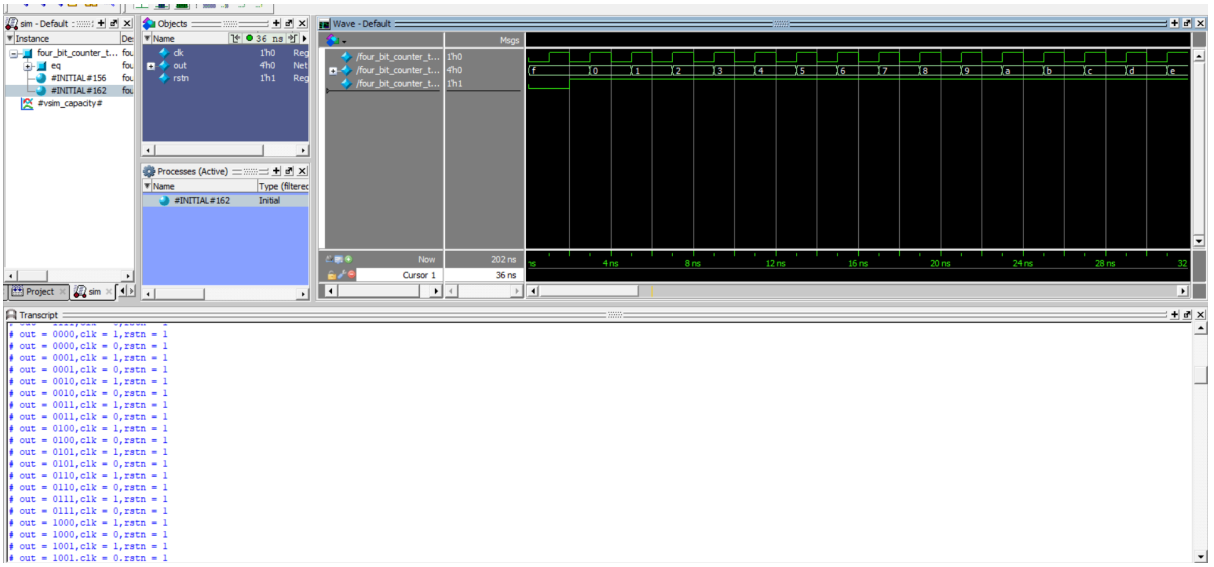




3) Implement the 4-bit Ripple counter shown below using structural modelling (Instantiate the Dff from question 2 part B where the output is taken from the qn as shown below). Implement a randomized testbench and check the output correctness from the waveform.
   - Inputs: clk, rstn;
   - Outputs: [3:0] out;



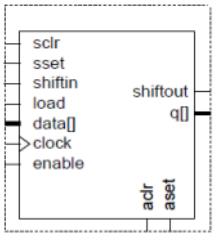
```
module four_bit_counter ( output [3:0] out,
                          input clk,rstn);
        wire [3:0] q;
        d_flipflop q1 (q[0],out[0],out[0],clk,rstn);
        d_flipflop q2 (q[1],out[1],out[1],q[0],rstn);
        d_flipflop q3 (q[2],out[2],out[2],q[1],rstn);
        d_flipflop q4 (q[3],out[3],out[3],q[2],rstn);

endmodule

module four_bit_counter_ts ();
   wire [3:0] out;
   reg clk,rstn;

four_bit_counter eq (out,clk,rstn);
initial begin
   clk = 0;
   forever begin
      #1 clk = ~clk;
   end
end
initial begin
   rstn = 0;
   @(negedge clk);
   rstn = 1;
   repeat(100) @(negedge clk);
   $stop;
end
initial begin
   $monitor ("out = %b,clk = %b,rstn = %b",out,clk,rstn);
end

endmodule
```



4) Implement the following Parameterized Shift register

1. Parameters

Name	Value	Description
LOAD_AVALUE	Integer > 0	Value loaded with aset is high
SHIFT_DIRECTION	"LEFT" or "RIGHT"	Direction of the shift register. Default = "LEFT"
LOAD_SVALUE	Integer > 0	Value loaded with sset is high with the rising clock edge
SHIFT_WIDTH	Integer > 0	Width of data[] and q[] ports



Default value for LOAD_AVALUE and LOAD_SVALUE is 1. SHIFT_WIDTH default value is 8.

2. Ports

Name	Type	Description
sclr	Input	Synchronous clear input. If both sclr and sset are asserted, sclr is dominant.
sset		Synchronous set input that sets q[] output with the value specified by LOAD_SVALUE. If both sclr and sset are asserted, sclr is dominant.
shiftin		Serial shift data input
load		Synchronous parallel load. High: Load operation with data[], Low: Shift operation
data[]		Data input to the shift register. This port is SHIFT_WIDTH wide
clock		Clock Input
enable		Clock enable input
aclr		Asynchronous clear input. If both aclr and aset are asserted, aclr is dominant.
aset		Asynchronous set input that sets q[] output with the value specified by LOAD_AVALUE. If both aclr and aset are asserted, aclr is dominant.
shiftout	Output	Serial Shift data output
q[]		Data output from the shift register. This port is SHIFT_WIDTH wide

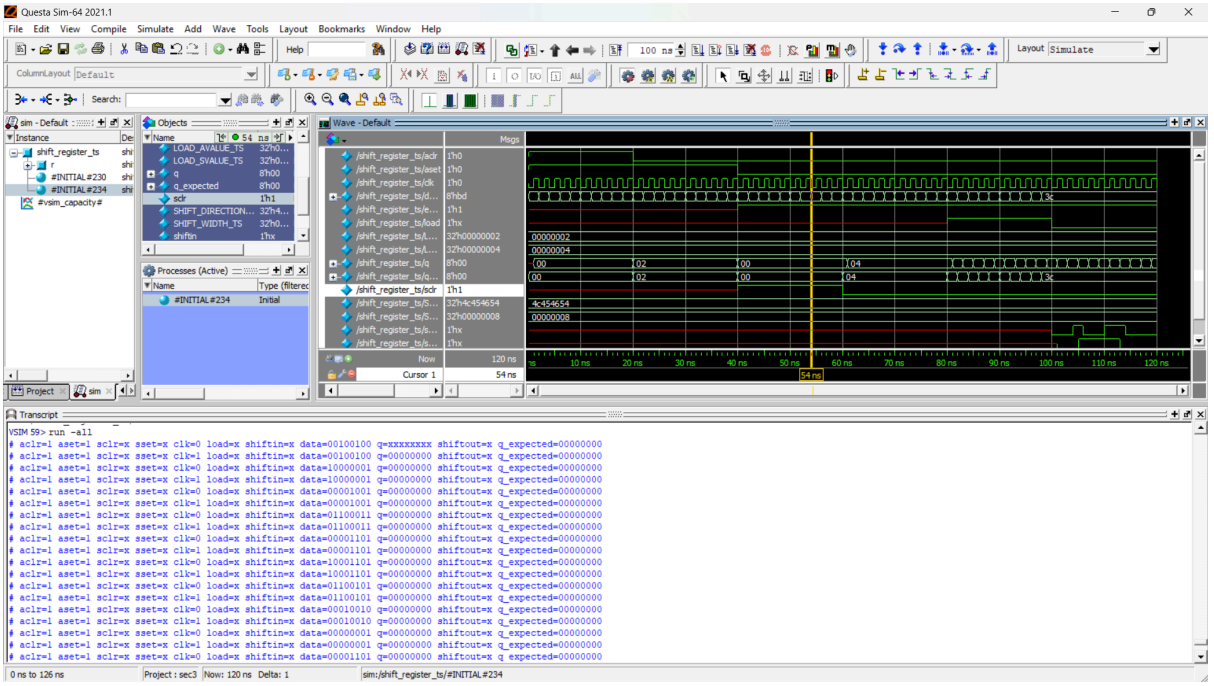
Notes:

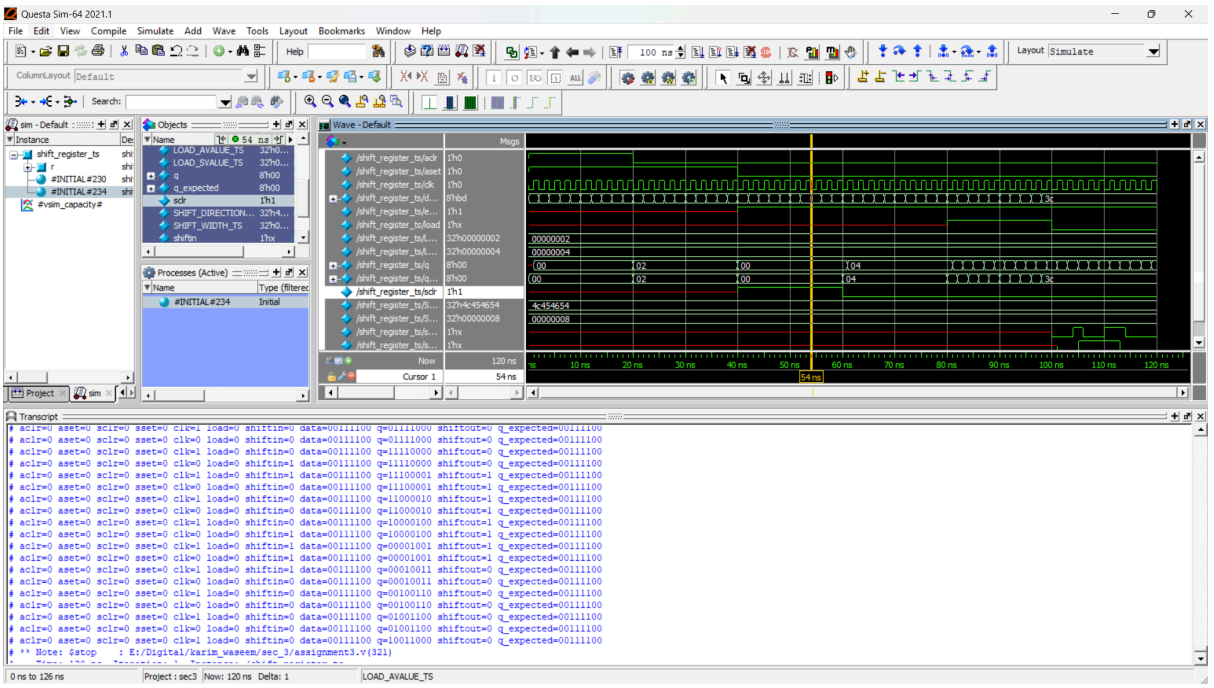
```
module shift_register #( parameter LOAD_AVALUE =1,
                         parameter LOAD_SVALUE =1,
                         parameter SHIFT_WIDTH =8,
                         parameter SHIFT_DIRECTION = "LEFT")
                       ( output reg [SHIFT_WIDTH-1:0] q,
                         output reg shiftout,
                         input [SHIFT_WIDTH-1:0] data,
                         input sclr,sset,shiftin,load,clk,enable,aclr,aset);
    always @(posedge clk , negedge aclr , negedge aset) begin
          if (aclr)
             q <= 0;
          else if (aset && !aclr)
             q <= LOAD_AVALUE;
          else if (!aclr && !aset) begin
           if (enable) begin
                  if (sclr)
                     q <= 0;
                  else  if (sset && !sclr)
                     q <= LOAD_SVALUE;

                  else begin
                   if (load)
                     q <= data;
                  else begin
                    if (SHIFT_DIRECTION == "LEFT") begin
                       shiftout <= q [SHIFT_WIDTH-1];
                       q <= {q[SHIFT_WIDTH-2:0],shiftin};
                    end
                    else if (SHIFT_DIRECTION == "RIGHT") begin
                       shiftout <= q [0];
                       q <= {shiftin,q[SHIFT_WIDTH-1:1]};
                    end

                  end
          end
           end
          end
    end

endmodule
```

```
module shift_register_ts ();
        parameter LOAD_AVALUE_TS =2;
        parameter LOAD_SVALUE_TS =4;
        parameter SHIFT_WIDTH_TS =8;
        parameter SHIFT_DIRECTION_TS = "LEFT";
        wire [SHIFT_WIDTH_TS-1:0] q;
        reg [SHIFT_WIDTH_TS-1:0] q_expected;
        wire shiftout;
        reg [SHIFT_WIDTH_TS-1:0] data;
        reg sclr,sset,shiftin,load,clk,enable,aclr,aset;
        shift_register # (.LOAD_AVALUE(LOAD_AVALUE_TS),.LOAD_SVALUE(LOAD_SVALUE_TS),.SHIFT_WIDTH(SHIFT_WIDTH_TS),.SHIFT_DIRECTION(SHIFT_DIRECTION_TS))
                        r (q,shiftout,data,sclr,sset,shiftin,load,clk,enable,aclr,aset);
        initial begin
          clk = 0;
          forever #1 clk = ~clk;
        end
        initial begin
          aclr = 1;
          aset = 1;
          repeat (10) begin
             data = $random;
             if (aclr)
                q_expected = 0;
             else begin
                if (aset)
                   q_expected = LOAD_AVALUE_TS;
             end
          @(negedge clk);
          end



          aclr = 0;
          aset = 1;
          repeat (10) begin
             data = $random;
             if (aclr)
                q_expected = 0;
             else begin
                if (aset)
                   q_expected = LOAD_AVALUE_TS;
             end
          @(negedge clk);
          end
```

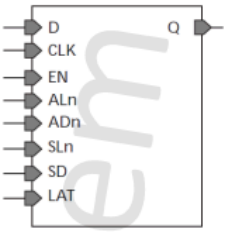
```
          enable = 1;
          aset = 0;
          aclr = 0;
          sclr = 1;
          sset = 1;
          repeat (10) begin
             data = $random;
             if (!aset && !aclr)
                 if (sclr)
                    q_expected = 0;
                 else begin
                 if (sset)
                    q_expected = LOAD_SVALUE_TS;
                 end
             @(negedge clk);
          end



          aset = 0;
          aclr = 0;
          sclr = 0;
          sset = 1;
          repeat (10) begin
             data = $random;
             if (!aset && !aclr)
                 if (sclr)
                    q_expected = 0;
                 else begin
                 if (sset)
                    q_expected = LOAD_SVALUE_TS;
                 end
            @(negedge clk);
          end
```

```
    aclr = 0;
    aset = 0;
    sclr = 0;
    sset = 0;
    load = 1;
    repeat (10) begin
      data = $random;
      if (load) begin
        q_expected = data;
        @(negedge clk);
      end
    end


    aclr = 0;
    aset = 0;
    sclr = 0;
    sset = 0;
    load = 0;
    repeat (10) begin
    shiftin = $random;
    @(negedge clk);
    end
    $stop;
    end
    initial begin
      $monitor("aclr=%b aset=%b sclr=%b sset=%b clk=%b load=%b shiftin=%b data=%b q=%b shiftout=%b q_expected=%b",aclr,aset,sclr,sset,clk,load,shiftin,data,q,shiftout,q_expected);
    end

module
```


Questa Sim-64 2021.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript
VSIM 59> run -all
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=00100100 q=xxxxxxxx shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=00100100 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=10000001 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=10000001 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=00001001 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=00001001 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=01100011 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=01100011 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=00001101 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=00001101 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=10001101 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=10001101 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=01100101 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=01100101 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=00010010 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=00010010 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=00000001 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=1 load=x shiftin=x data=00000001 q=00000000 shiftout=x q_expected=00000000
aclr=1 aset=1 sclr=x sset=x clk=0 load=x shiftin=x data=00001101 q=00000000 shiftout=x q_expected=00000000
0 ns to 126 ns
Project : sec3
Now: 120 ns
Delta: 1
sim:/shift_register_ts/#INITIAL#234


Questa Sim-64 2021.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript
** Note: $stop : E:/Digital/karim_waseem/sec_3/assignment3.v(321)
0 ns to 126 ns
Project : sec3
Now: 120 ns
Delta: 1
LOAD_AVALUE_TS

5) Implement the following SLE (sequential logic element). This design will act as flipflop or latch based on the LAT signal as demonstrated in the truth table.



Input		Output
Name	Function	
D	Data	
CLK	Clock	
EN	Enable	
ALn	Asynchronous Load (Active Low)	Q
ADn*	Asynchronous Data (Active Low)	
SLn	Synchronous Load (Active Low)	
SD*	Synchronous Data	
LAT*	Latch Enable	

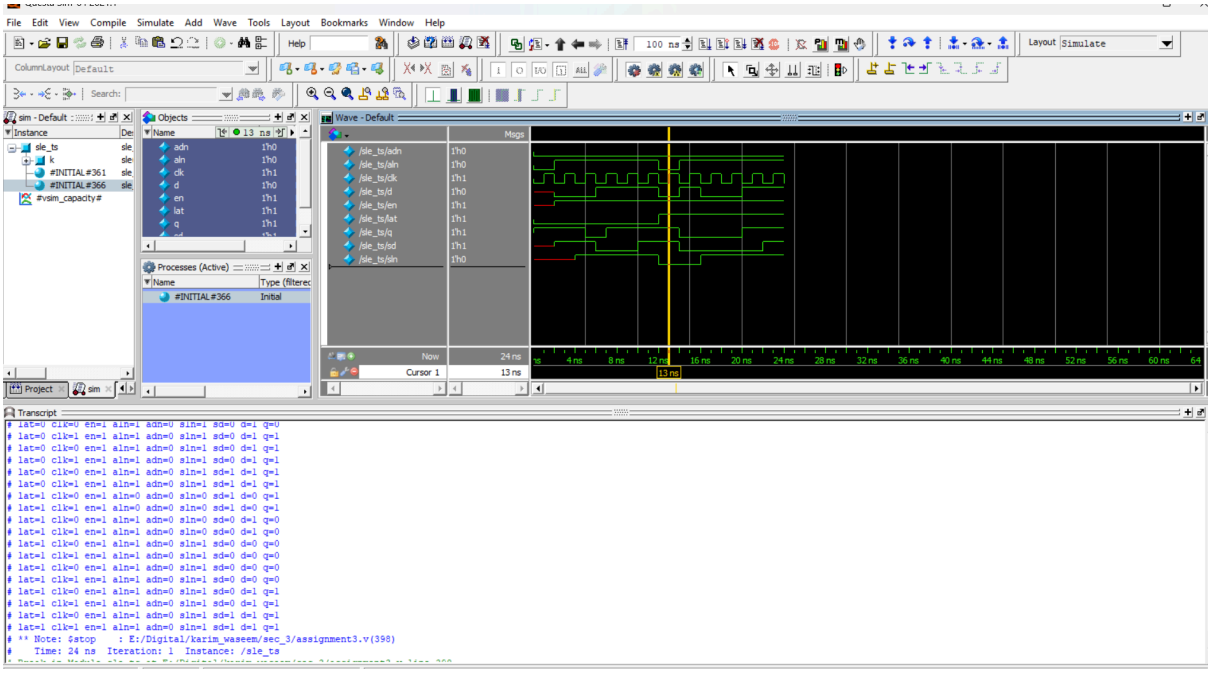
Truth Table

ALn	ADn	LAT	CLK	EN	SLn	SD	D	Q_{n+1}
0	ADn	X	X	X	X	X	X	!ADn
1	X	0	Not rising	X	X	X	X	Qn
1	X	0	↑	0	X	X	X	Qn
1	X	0	↑	1	0	SD	X	SD
1	X	0	↑	1	1	X	D	D
1	X	1	0	X	X	X	X	Qn
1	X	1	1	0	X	X	X	Qn
1	X	1	1	1	0	SD	X	SD
1	X	1	1	1	1	X	D	D

Testbench should start with activating ALn then deactivating it. Then for simplicity we keep the LAT to 0 and in a repeat block randomize all inputs. Then drive LAT to 1 and in a repeat block randomize all inputs. Check the functionality correctness of each input from the waveform.

```
module sle ( output q,
             input d,clk,en,aln,adn,sln,sd,lat);
reg q_flipflop,q_latch;
assign q = (lat) ? q_latch : q_flipflop;

always @(posedge clk , negedge aln , negedge adn) begin
    if (!aln)
        q_flipflop <= ~adn;
    else begin
      if (sln)
        q_flipflop <= d;
      else if (en)
        q_flipflop <= sd;
    end
end
always @(*) begin
    if (!aln)
        q_latch <= ~adn;
    else begin
      if (sln)
        q_latch <= d;
      else if (en)
        q_latch <= sd;
    end
end
endmodule
```

```
module sle_ts ();
    wire q;
    reg d,clk,en,aln,adn,sln,sd,lat;
    sle k (q,d,clk,en,aln,adn,sln,sd,lat);

    initial begin
      clk = 0;
      forever #1 clk = ~clk;
    end

    initial begin
      lat = 0;
      aln = 0;
      adn = 0;
      @(negedge clk);
      aln = 1;
      en = 1;
      repeat (5) begin
         d = $random;
         sd = $random;
         @(negedge clk);
         sln = 1;
         repeat (5) begin
         d = $random;
         end
      end
      lat = 1;
      aln = 0;
      adn = 0;
      sln = 0;
      @(negedge clk);
      aln = 1;
      en = 1;
      repeat (5) begin
         d = $random;
         sd = $random;
         @(negedge clk);
         sln = 1;
         repeat (5) begin
         d = $random;
         end
      end
      $stop;


    end
    initial begin
      $monitor ("lat=%b clk=%b en=%b aln=%b adn=%b sln=%b sd=%b d=%b q=%b",lat,clk,en,aln,adn,sln,sd,d,q);
    end

endmodule
```


File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript
lat=0 clk=0 en=1 aln=1 adn=0 sln=1 sd=0 d=1 q=0
lat=0 clk=1 en=1 aln=1 adn=0 sln=1 sd=0 d=1 q=1
lat=0 clk=0 en=1 aln=1 adn=0 sln=1 sd=0 d=1 q=1
lat=0 clk=1 en=1 aln=1 adn=0 sln=1 sd=0 d=1 q=1
lat=0 clk=0 en=1 aln=1 adn=0 sln=1 sd=1 d=1 q=1
lat=0 clk=1 en=1 aln=1 adn=0 sln=1 sd=1 d=1 q=1
lat=1 clk=0 en=1 aln=0 adn=0 sln=0 sd=1 d=0 q=1
lat=1 clk=1 en=1 aln=0 adn=0 sln=0 sd=1 d=0 q=1
lat=1 clk=0 en=1 aln=1 adn=0 sln=0 sd=0 d=1 q=0
lat=1 clk=1 en=1 aln=1 adn=0 sln=0 sd=0 d=1 q=0
lat=1 clk=0 en=1 aln=1 adn=0 sln=1 sd=0 d=0 q=0
lat=1 clk=1 en=1 aln=1 adn=0 sln=1 sd=0 d=0 q=0
lat=1 clk=0 en=1 aln=1 adn=0 sln=1 sd=0 d=0 q=0
lat=1 clk=1 en=1 aln=1 adn=0 sln=1 sd=0 d=0 q=0
lat=1 clk=0 en=1 aln=1 adn=0 sln=0 sd=0 d=1 q=1
lat=1 clk=1 en=1 aln=1 adn=0 sln=0 sd=0 d=1 q=1
lat=1 clk=0 en=1 aln=1 adn=0 sln=1 sd=1 d=1 q=1
lat=1 clk=1 en=1 aln=1 adn=0 sln=1 sd=1 d=1 q=1
** Note: $stop : E:/Digital/karim_waseem/sec_3/assignment3.v(398)
Time: 24 ns Iteration: 1 Instance: /sle_ts