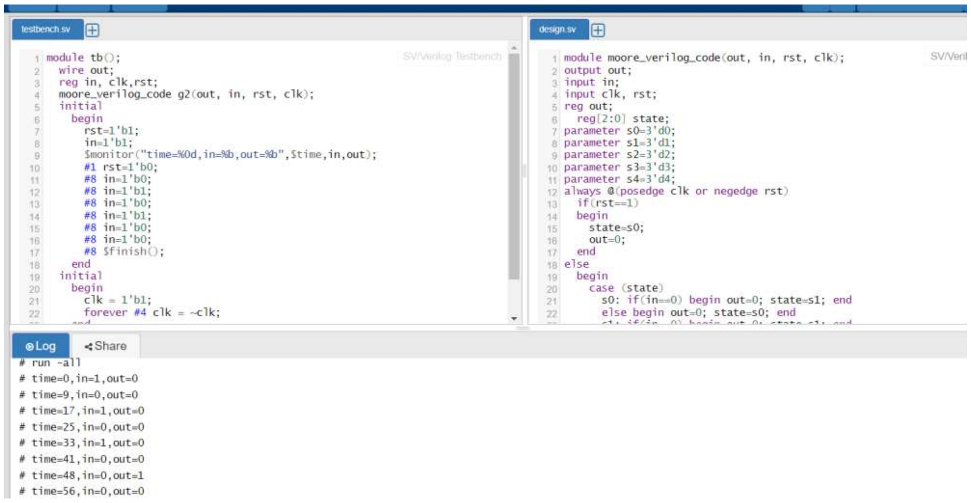
1.)verilog code for 4-bit fsm moore and mealy machine



testbench.sv

```
module tb();
  wire out;
  reg in, clk,rst;
  moore_verilog_code g2(out, in, rst, clk);
  initial
    begin
      rst=1'b1;
      in=1'b1;
      $monitor("time=%0d,in=%b,out=%b",$time,in,out);
      #1 rst=1'b0;
      #8 in=1'b0;
      #8 in=1'b1;
      #8 in=1'b0;
      #8 in=1'b1;
      #8 in=1'b0;
      #8 in=1'b0;
      #8 $finish();
    end
  initial
    begin
      clk = 1'b1;
      forever #4 clk = ~clk;
```

design.sv

```
module moore_verilog_code(out, in, rst, clk);
output out;
input in;
input clk, rst;
reg out;
  reg[2:0] state;
parameter s0=3'd0;
parameter s1=3'd1;
parameter s2=3'd2;
parameter s3=3'd3;
parameter s4=3'd4;
always @(posedge clk or negedge rst)
  if(rst==1)
  begin
    state=s0;
    out=0;
  end
else
  begin
    case (state)
      s0: if(in==0) begin out=0; state=s1; end
      else begin out=0; state=s0; end
```

Log

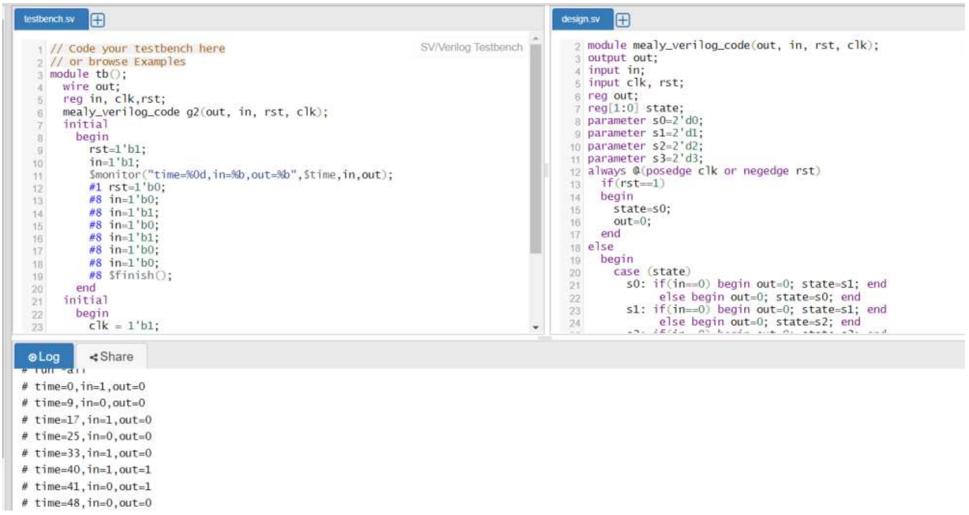
```
# run -all
# time=0,in=1,out=0
# time=9,in=0,out=0
# time=17,in=1,out=0
# time=25,in=0,out=0
# time=33,in=1,out=0
# time=41,in=0,out=0
# time=48,in=0,out=1
# time=56,in=0,out=0
```

Moore sequence detector design code for detecting 0101:

```
module moore_verilog_code(out,  in, rst, clk);

output out;

input in;

input clk, rst;

reg out;

  reg[2:0] state;

parameter s0=3'd0;

parameter s1=3'd1;

parameter s2=3'd2;

parameter s3=3'd3;

parameter s4=3'd4;

always @(posedge clk or negedge rst)

  if(rst==1)

  begin

    state=s0;

    out=0;
  end
else
  begin
    case (state)
      s0: if(in==0) begin out=0; state=s1; end
      else begin out=0; state=s0; end
      s1: if(in==0) begin out=0; state=s1; end
      else begin out=0; state=s2; end
      s2: if(in==0) begin out=0; state=s3; end
      else begin out=0; state=s0; end
      s3: if(in==0) begin out=0; state=s1; end
      else begin out=0; state=s4; end
      s4: if(in==0) begin out=1; state=s3; end
      else begin out=1; state=s0; end
      default: state=s0;
    endcase
end
endmodule
```

Testbench:

```
module tb();
  wire out;
  reg in, clk,rst;
  moore_verilog_code g2(out, in, rst, clk);
  initial
    begin
    rst=1'b1;
    in=1'b1;
    $monitor("time=%0d,in=%b,out=%b",$time,in,out);
    #1 rst=1'b0;
    #8 in=1'b0;
    #8 in=1'b1;
    #8 in=1'b0;
    #8 in=1'b1;
    #8 in=1'b0;
    #8 in=1'b0;
    #8 $finish();
  end
 initial
  begin
    clk = 1'b1;
    forever #4 clk = ~clk;
  end
endmodule
```



testbench.sv

SV/Verilog Testbench

```
// Code your testbench here
// or browse Examples
module tb();
  wire out;
  reg in, clk,rst;
  mealy_verilog_code g2(out, in, rst, clk);
  initial
    begin
      rst=1'b1;
      in=1'b1;
      $monitor("time=%0d,in=%b,out=%b",$time,in,out);
      #1 rst=1'b0;
      #8 in=1'b0;
      #8 in=1'b1;
      #8 in=1'b0;
      #8 in=1'b1;
      #8 in=1'b0;
      #8 in=1'b0;
      #8 $finish();
    end
  initial
    begin
      clk = 1'b1;
```

design.sv

```
module mealy_verilog_code(out, in, rst, clk);
output out;
input in;
input clk, rst;
reg out;
reg[1:0] state;
parameter s0=2'd0;
parameter s1=2'd1;
parameter s2=2'd2;
parameter s3=2'd3;
always @(posedge clk or negedge rst)
  if(rst==1)
  begin
    state=s0;
    out=0;
  end
else
  begin
    case (state)
      s0: if(in==0) begin out=0; state=s1; end
           else begin out=0; state=s0; end
      s1: if(in==0) begin out=0; state=s1; end
           else begin out=0; state=s2; end
```

Log Share

```
# time=0,in=1,out=0
# time=9,in=0,out=0
# time=17,in=1,out=0
# time=25,in=0,out=0
# time=33,in=1,out=0
# time=40,in=1,out=1
# time=41,in=0,out=1
# time=48,in=0,out=0
```

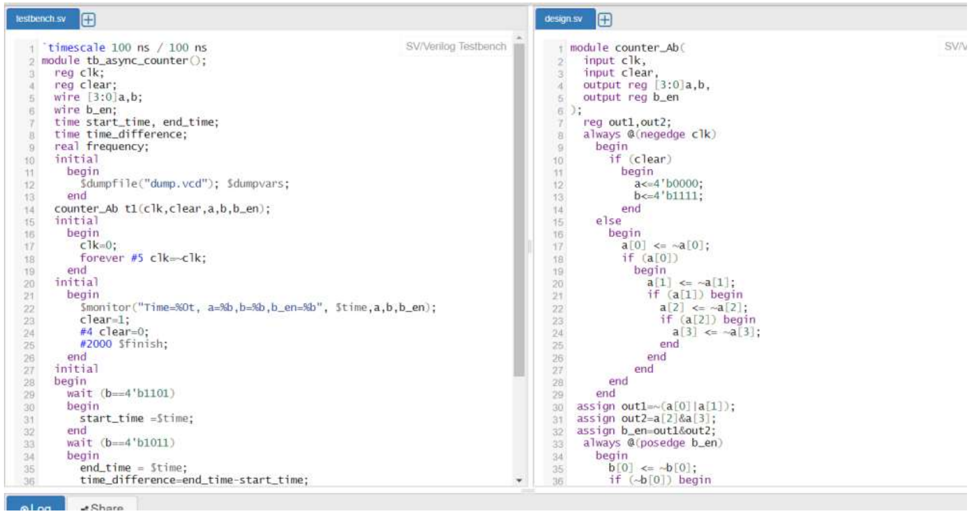
Mealy code for detecting sequence 0101:

```
module mealy_verilog_code(out,  in, rst, clk);
output out;
input in;
input clk, rst;
reg out;
reg[1:0] state;
parameter s0=2'd0;
parameter s1=2'd1;
parameter s2=2'd2;
parameter s3=2'd3;
always @(posedge clk or negedge rst)
  if(rst==1)
  begin
    state=s0;
    out=0;
  end
else
  begin
    case (state)
      s0: if(in==0) begin out=0; state=s1; end
          else begin out=0; state=s0; end
      s1: if(in==0) begin out=0; state=s1; end
          else begin out=0; state=s2; end
      s2: if(in==0) begin out=0; state=s3; end
          else begin out=0; state=s0; end
      s3: if(in==0) begin out=0; state=s1; end
          else begin out=1; state=s2; end
      default: state=s0;
    endcase
end
endmodule
```

Testbench:

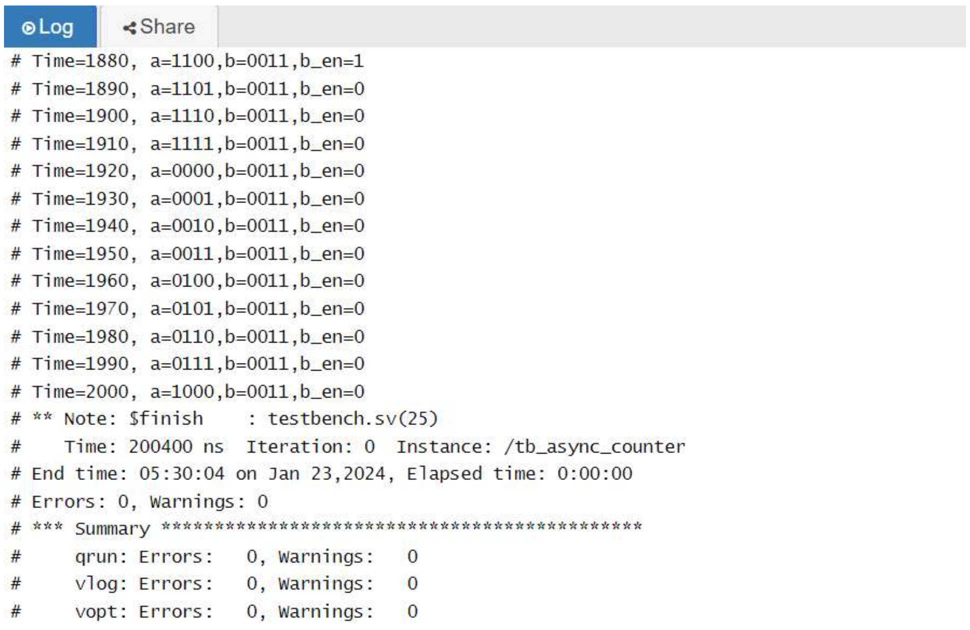
```
module tb();
  wire out;
  reg in, clk,rst;
  mealy_verilog_code g2(out, in, rst, clk);
  initial
    begin
      rst=1'b1;
      in=1'b1;
      $monitor("time=%0d,in=%b,out=%b",$time,in,out);
      #1 rst=1'b0;
      #8 in=1'b0;
      #8 in=1'b1;
      #8 in=1'b0;
      #8 in=1'b1;
      #8 in=1'b0;
      #8 in=1'b0;
      #8 $finish();
    end
  initial
  begin
    clk = 1'b1;
    forever #4 clk = ~clk;
  end
endmodule
```

2)assignment question of counter



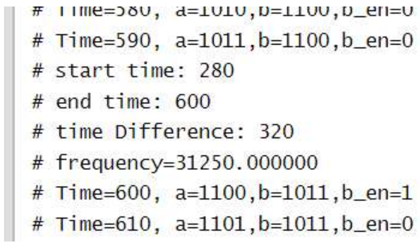
```
`timescale 100 ns / 100 ns
module tb_async_counter();
  reg clk;
  reg clear;
  wire [3:0]a,b;
  wire b_en;
  time start_time, end_time;
  time time_difference;
  real frequency;
  initial
    begin
      $dumpfile("dump.vcd"); $dumpvars;
    end
  counter_Ab t1(clk,clear,a,b,b_en);
  initial
    begin
      clk=0;
      forever #5 clk=~clk;
    end
  initial
    begin
      $monitor("Time=%0t, a=%b,b=%b,b_en=%b", $time,a,b,b_en);
      clear=1;
      #4 clear=0;
      #2000 $finish;
    end
  initial
  begin
    wait (b==4'b1101)
    begin
      start_time =$time;
    end
    wait (b==4'b1011)
    begin
      end_time = $time;
      time_difference=end_time-start_time;
```

```
module counter_Ab(
  input clk,
  input clear,
  output reg [3:0]a,b,
  output reg b_en
);
  reg out1,out2;
  always @(negedge clk)
    begin
      if (clear)
        begin
          a<=4'b0000;
          b<=4'b1111;
        end
    else
      begin
        a[0] <= ~a[0];
        if (a[0])
          begin
            a[1] <= ~a[1];
            if (a[1]) begin
              a[2] <= ~a[2];
              if (a[2]) begin
                a[3] <= ~a[3];
              end
            end
          end
      end
    end
 assign out1=~(a[0]|a[1]);
 assign out2=a[2]&a[3];
 assign b_en=out1&out2;
  always @(posedge b_en)
    begin
      b[0] <= ~b[0];
      if (~b[0]) begin
```



```
# Time=1880, a=1100,b=0011,b_en=1
# Time=1890, a=1101,b=0011,b_en=0
# Time=1900, a=1110,b=0011,b_en=0
# Time=1910, a=1111,b=0011,b_en=0
# Time=1920, a=0000,b=0011,b_en=0
# Time=1930, a=0001,b=0011,b_en=0
# Time=1940, a=0010,b=0011,b_en=0
# Time=1950, a=0011,b=0011,b_en=0
# Time=1960, a=0100,b=0011,b_en=0
# Time=1970, a=0101,b=0011,b_en=0
# Time=1980, a=0110,b=0011,b_en=0
# Time=1990, a=0111,b=0011,b_en=0
# Time=2000, a=1000,b=0011,b_en=0
# ** Note: $finish    : testbench.sv(25)
#    Time: 200400 ns  Iteration: 0  Instance: /tb_async_counter
# End time: 05:30:04 on Jan 23,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# *** Summary *********************************************
#     qrun: Errors:   0, Warnings:   0
#     vlog: Errors:   0, Warnings:   0
#     vopt: Errors:   0, Warnings:   0
```

At the end of simulation for 0.2 milli secs A counter had 1000(demical 8) and b counter had 0011(decimal 3)



```
# Time=580, a=1010,b=1100,b_en=0
# Time=590, a=1011,b=1100,b_en=0
# start time: 280
# end time: 600
# time Difference: 320
# frequency=31250.000000
# Time=600, a=1100,b=1011,b_en=1
# Time=610, a=1101,b=1011,b_en=0
```

Frequency of b[0] bit 31250 hertz or 0.3125 khz

Design code:

```
module counter_Ab(
 input clk,
 input clear,
 output reg [3:0]a,b,
 output reg b_en
);
 reg out1,out2;
 always @(negedge clk)
  begin
```

```
   if (clear)
    begin
     a<=4'b0000;
     b<=4'b1111;
    end
  else
   begin
    a[0] <= ~a[0];
    if (a[0])
     begin
      a[1] <= ~a[1];
      if (a[1]) begin
       a[2] <= ~a[2];
       if (a[2]) begin
        a[3] <= ~a[3];
       end
      end
     end
   end
  end
assign out1=~(a[0]|a[1]);
assign out2=a[2]&a[3];
assign b_en=out1&out2;
 always @(posedge b_en)
  begin
   b[0] <= ~b[0];
```

```
    if (~b[0]) begin
     b[1] <= ~b[1];
     if (~b[1]) begin
      b[2] <= ~b[2];
      if (~b[2]) begin
       b[3] <= ~b[3];
      end
     end
    end
  end
Endmodule
```

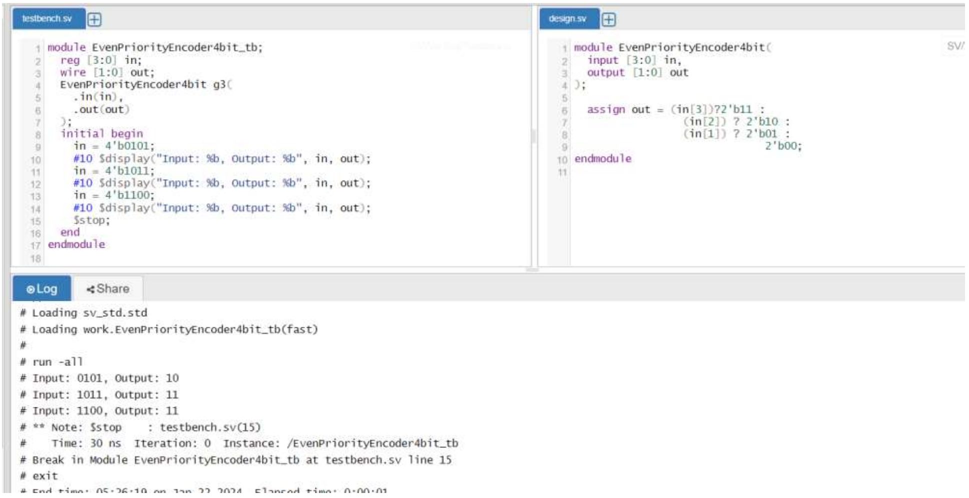
Testbench code:

```
module tb_async_counter();
 reg clk;
 reg clear;
 wire [3:0]a,b;
 wire b_en;
 time start_time, end_time;
 time time_difference;
 real frequency;
 initial
  begin
   $dumpfile("dump.vcd"); $dumpvars;
  end
```

```
counter_Ab t1(clk,clear,a,b,b_en);
initial
 begin
  clk=0;
  forever #5 clk=~clk;
 end
initial
 begin
  $monitor("Time=%0t, a=%b,b=%b,b_en=%b", $time,a,b,b_en);
  clear=1;
  #4 clear=0;
  #2000 $finish;
 end
initial
begin
 wait (b==4'b1101)
 begin
  start_time =$time;
 end
 wait (b==4'b1011)
 begin
  end_time = $time;
  time_difference=end_time-start_time;
  $display("start time: %0t",start_time);
  $display("end time: %0t",end_time);
  $display("time Difference: %0t",time_difference);
```

```
    frequency=10**7/time_difference;

    $display("frequency=%f",frequency);

  end

 end

endmodule
```

3. 4bit even priority encoder



```
module EvenPriorityEncoder4bit_tb;
  reg [3:0] in;
  wire [1:0] out;
  EvenPriorityEncoder4bit g3(
    .in(in),
    .out(out)
  );
  initial begin
    in = 4'b0101;
    #10 $display("Input: %b, Output: %b", in, out);
    in = 4'b1011;
    #10 $display("Input: %b, Output: %b", in, out);
    in = 4'b1100;
    #10 $display("Input: %b, Output: %b", in, out);
    $stop;
  end
endmodule
```

```
module EvenPriorityEncoder4bit(
  input [3:0] in,
  output [1:0] out
);

  assign out = (in[3])?2'b11 :
               (in[2]) ? 2'b10 :
               (in[1]) ? 2'b01 :
                         2'b00;
endmodule
```

```
# Loading sv_std.std
# Loading work.EvenPriorityEncoder4bit_tb(fast)
#
# run -all
# Input: 0101, Output: 10
# Input: 1011, Output: 11
# Input: 1100, Output: 11
# ** Note: $stop    : testbench.sv(15)
#    Time: 30 ns  Iteration: 0  Instance: /EvenPriorityEncoder4bit_tb
# Break in Module EvenPriorityEncoder4bit_tb at testbench.sv line 15
# exit
# End time: 05:26:19 on Jan 22 2024, Elapsed time: 0:00:01
```

Design code:

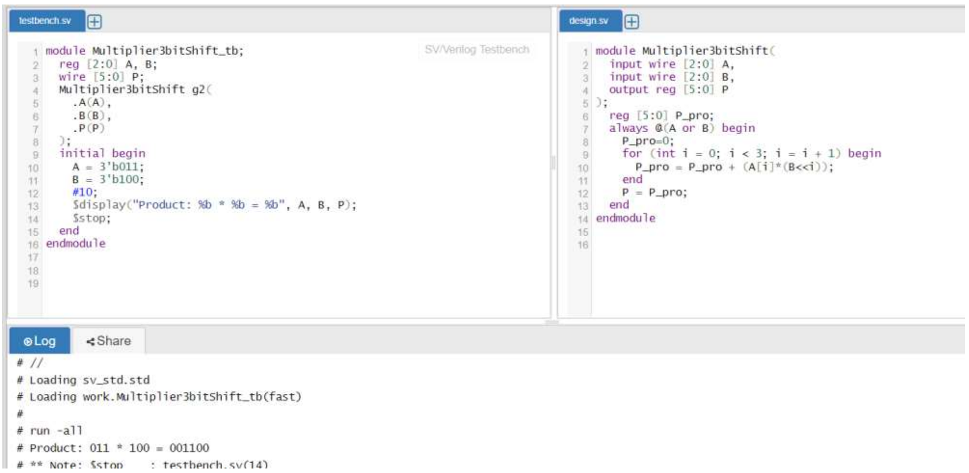
```
module EvenPriorityEncoder4bit(

 input [3:0] in,

 output [1:0] out

);



 assign out = (in[3])?2'b11 :

          (in[2]) ? 2'b10 :

          (in[1]) ? 2'b01 :

                2'b00;
```

```
Endmodule
```

Testbench code:

```
module EvenPriorityEncoder4bit_tb;
 reg [3:0] in;
 wire [1:0] out;
 EvenPriorityEncoder4bit g3(
  .in(in),
  .out(out)
 );
 initial begin
  in = 4'b0101;
  #10 $display("Input: %b, Output: %b", in, out);
  in = 4'b1011;
  #10 $display("Input: %b, Output: %b", in, out);
  in = 4'b1100;
  #10 $display("Input: %b, Output: %b", in, out);
  $stop;
 end
Endmodule
```

4. 3 bit multiplier using shift registers



```
module Multiplier3bitShift_tb;
  reg [2:0] A, B;
  wire [5:0] P;
  Multiplier3bitShift g2(
    .A(A),
    .B(B),
    .P(P)
  );
  initial begin
    A = 3'b011;
    B = 3'b100;
    #10;
    $display("Product: %b * %b = %b", A, B, P);
    $stop;
  end
endmodule
```

```
module Multiplier3bitShift(
  input wire [2:0] A,
  input wire [2:0] B,
  output reg [5:0] P
);
  reg [5:0] P_pro;
  always @(A or B) begin
    P_pro=0;
    for (int i = 0; i < 3; i = i + 1) begin
      P_pro = P_pro + (A[i]*(B<<i));
    end
    P = P_pro;
  end
endmodule
```

```
# //
# Loading sv_std.std
# Loading work.Multiplier3bitShift_tb(fast)
#
# run -all
# Product: 011 * 100 = 001100
# ** Note: $stop    : testbench.sv(14)
```

Design code:

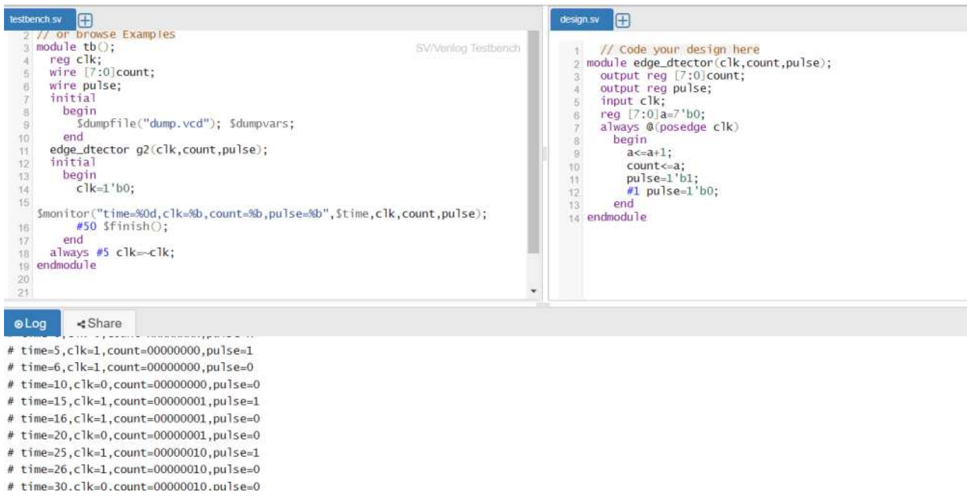
```
module Multiplier3bitShift(
  input wire [2:0] A,
  input wire [2:0] B,
  output reg [5:0] P
);
  reg [5:0] P_pro;
  always @(A or B) begin
    P_pro=0;
    for (int i = 0; i < 3; i = i + 1) begin
      P_pro = P_pro + (A[i]*(B<<i));
    end
    P = P_pro;
  end
```

```
Endmodule
```

Testbench code:

```
module Multiplier3bitShift_tb;
 reg [2:0] A, B;
 wire [5:0] P;
 Multiplier3bitShift g2(
  .A(A),
  .B(B),
  .P(P)
 );
 initial begin
  A = 3'b011;
  B = 3'b100;
  #10;
  $display("Product: %b * %b = %b", A, B, P);
  $stop;
 end
endmodule
```

5. rising and falling edge detector



```
// or browse Examples
module tb();
  reg clk;
  wire [7:0]count;
  wire pulse;
  initial
    begin
      $dumpfile("dump.vcd"); $dumpvars;
    end
  edge_dtector g2(clk,count,pulse);
  initial
    begin
      clk=1'b0;
$monitor("time=%0d,clk=%b,count=%b,pulse=%b",$time,clk,count,pulse);
      #50 $finish();
    end
  always #5 clk=~clk;
endmodule
```

```
  // Code your design here
module edge_dtector(clk,count,pulse);
  output reg [7:0]count;
  output reg pulse;
  input clk;
  reg [7:0]a=7'b0;
  always @(posedge clk)
    begin
      a<=a+1;
      count<=a;
      pulse=1'b1;
      #1 pulse=1'b0;
    end
endmodule
```

```
# time=5,clk=1,count=00000000,pulse=1
# time=6,clk=1,count=00000000,pulse=0
# time=10,clk=0,count=00000000,pulse=0
# time=15,clk=1,count=00000001,pulse=1
# time=16,clk=1,count=00000001,pulse=0
# time=20,clk=0,count=00000001,pulse=0
# time=25,clk=1,count=00000010,pulse=1
# time=26,clk=1,count=00000010,pulse=0
# time=30,clk=0,count=00000010,pulse=0
```

Design code:

```
module edge_dtector(clk,count,pulse);
 output reg [7:0]count;
 output reg pulse;
 input clk;
 reg [7:0]a=7'b0;
 always @(posedge clk)
  begin
   a<=a+1;
   count<=a;
   pulse=1'b1;
   #1 pulse=1'b0;
  end
Endmodule
```

testbench code:

```
module tb();
 reg clk;
 wire [7:0]count;
 wire pulse;
 initial
  begin
   $dumpfile("dump.vcd"); $dumpvars;
  end
 edge_dtector g2(clk,count,pulse);
 initial
  begin
   clk=1'b0;
   $monitor("time=%0d,clk=%b,count=%b,pulse=%b",$time,clk,count,pulse);
   #50 $finish();
  end
 always #5 clk=~clk;
Endmodule
```