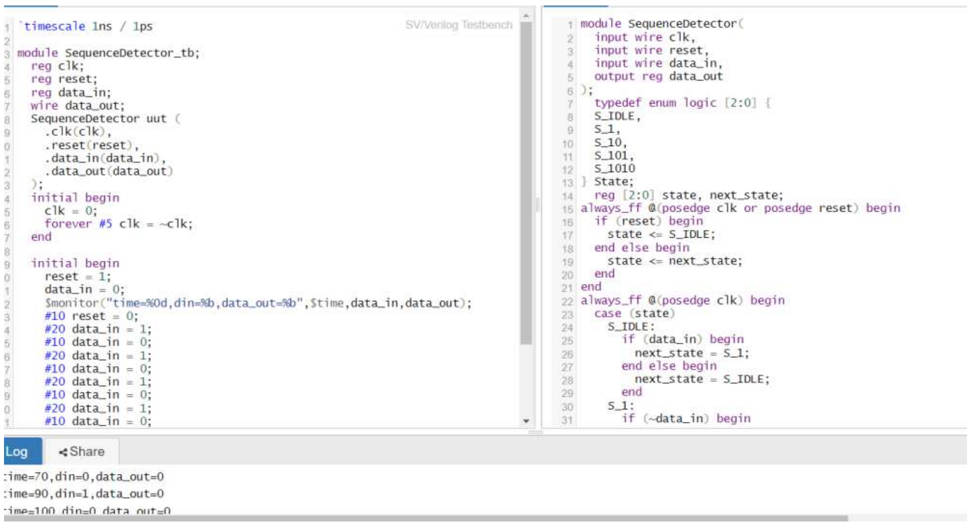
1.)verilog code for 4-bit fsm moore and mealy machine


SV/Verilog Testbench

```
`timescale 1ns / 1ps

module SequenceDetector_tb;
  reg clk;
  reg reset;
  reg data_in;
  wire data_out;
  SequenceDetector uut (
    .clk(clk),
    .reset(reset),
    .data_in(data_in),
    .data_out(data_out)
  );
  initial begin
    clk = 0;
    forever #5 clk = ~clk;
  end

  initial begin
    reset = 1;
    data_in = 0;
    $monitor("time=%0d,din=%b,data_out=%b",$time,data_in,data_out);
    #10 reset = 0;
    #20 data_in = 1;
    #10 data_in = 0;
    #20 data_in = 1;
    #10 data_in = 0;
    #20 data_in = 1;
    #10 data_in = 0;
    #20 data_in = 1;
    #10 data_in = 0;
```

```
module SequenceDetector(
  input wire clk,
  input wire reset,
  input wire data_in,
  output reg data_out
);
  typedef enum logic [2:0] {
  S_IDLE,
  S_1,
  S_10,
  S_101,
  S_1010
} State;
  reg [2:0] state, next_state;
always_ff @(posedge clk or posedge reset) begin
  if (reset) begin
    state <= S_IDLE;
  end else begin
    state <= next_state;
  end
end
always_ff @(posedge clk) begin
  case (state)
    S_IDLE:
      if (data_in) begin
        next_state = S_1;
      end else begin
        next_state = S_IDLE;
      end
    S_1:
      if (~data_in) begin
```

```
:ime=70,din=0,data_out=0
:ime=90,din=1,data_out=0
:ime=100 din=0 data out=0
```

Moore sequence detector design code for detecting 1010:

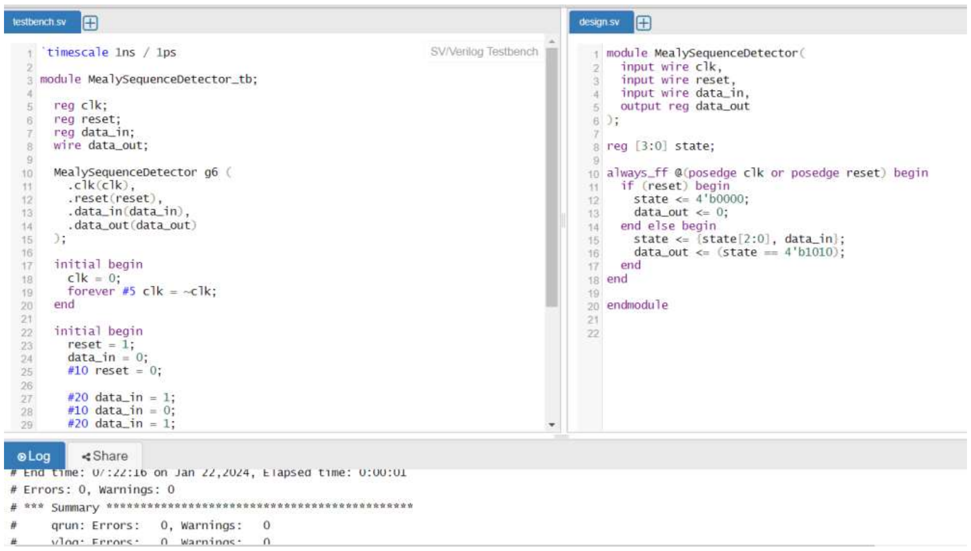
```
module SequenceDetector(
  input wire clk,
  input wire reset,
  input wire data_in,
  output reg data_out
);
  typedef enum logic [2:0] {
  S_IDLE,
  S_1,
  S_10,
  S_101,
  S_1010
} State;
  reg [2:0] state, next_state;
always_ff @(posedge clk or posedge reset) begin
  if (reset) begin
```

```
    state <= S_IDLE;
  end else begin
    state <= next_state;
  end
end
always_ff @(posedge clk) begin
  case (state)
    S_IDLE:
      if (data_in) begin
        next_state = S_1;
      end else begin
        next_state = S_IDLE;
      end
    S_1:
      if (~data_in) begin
        next_state = S_10;
      end else begin
        next_state = S_1;
      end
    S_10:
      if (data_in) begin
        next_state = S_101;
      end else begin
        next_state = S_IDLE;
      end
    S_101:
      if (~data_in) begin
       next_state = S_1010;
      end else begin
       next_state = S_1;
      end
    S_1010:
      if (data_in) begin
       next_state = S_101;
      end else begin
       next_state = S_IDLE;
      end
    default:
      next_state = S_IDLE;
  endcase
  if (state == S_1010) begin
    data_out = 1;
  end else begin
    data_out = 0;
  end
end
endmodule
```

Testbench:

```
`timescale 1ns / 1ps
module SequenceDetector_tb;
  reg clk;
reg reset;

reg data_in;

wire data_out;

SequenceDetector  g3(

  .clk(clk),

  .reset(reset),

  .data_in(data_in),

  .data_out(data_out)

);

initial begin

  clk = 0;

  forever #5 clk = ~clk;

end

initial begin

  reset = 1;

  data_in = 0;

  $monitor("time=%0d,din=%b,data_out=%b",$time,data_in,data_out);

  #10 reset = 0;

  #20 data_in = 1;

  #10 data_in = 0;

  #20 data_in = 1;

  #10 data_in = 0;

  #20 data_in = 1;

  #10 data_in = 0;

  #20 data_in = 1;

  #10 data_in = 0;
```

```
    #10 $finish;
  end
Endmodule
```


testbench.sv

```
`timescale 1ns / 1ps

module MealySequenceDetector_tb;

  reg clk;
  reg reset;
  reg data_in;
  wire data_out;

  MealySequenceDetector g6 (
    .clk(clk),
    .reset(reset),
    .data_in(data_in),
    .data_out(data_out)
  );

  initial begin
    clk = 0;
    forever #5 clk = ~clk;
  end

  initial begin
    reset = 1;
    data_in = 0;
    #10 reset = 0;

    #20 data_in = 1;
    #10 data_in = 0;
    #20 data_in = 1;
```

design.sv

```
module MealySequenceDetector(
  input wire clk,
  input wire reset,
  input wire data_in,
  output reg data_out
);

reg [3:0] state;

always_ff @(posedge clk or posedge reset) begin
  if (reset) begin
    state <= 4'b0000;
    data_out <= 0;
  end else begin
    state <= {state[2:0], data_in};
    data_out <= (state == 4'b1010);
  end
end

endmodule
```

```
# End time: 07:22:16 on Jan 22,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
# *** Summary *********************************************
#     qrun: Errors:   0, Warnings:   0
#     vlog: Errors:   0, Warnings:   0
```

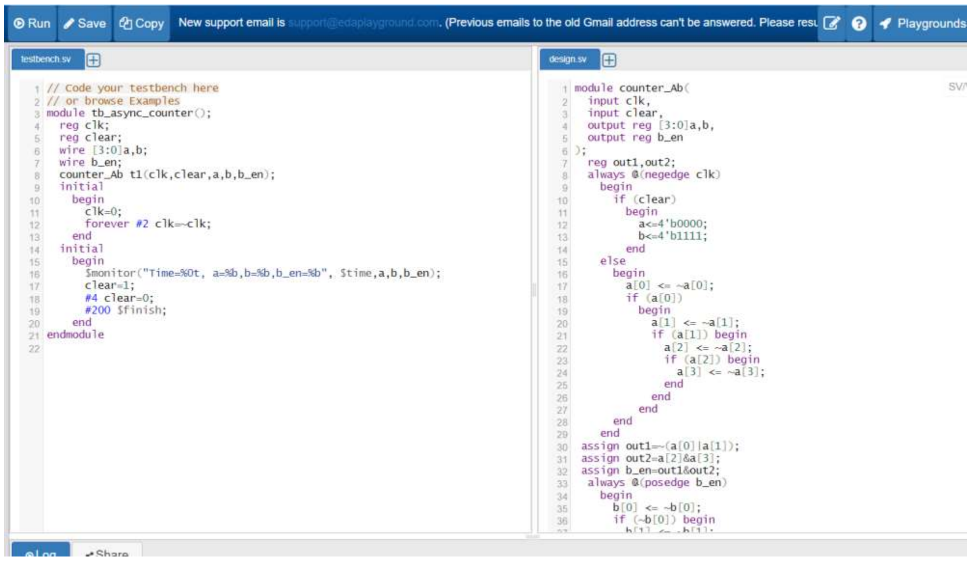
Mealy code for detecting sequence 1010:

```
module MealySequenceDetector(
  input wire clk,
  input wire reset,
  input wire data_in,
  output reg data_out
);
reg [3:0] state;
always_ff @(posedge clk or posedge reset) begin
  if (reset) begin
    state <= 4'b0000;
    data_out <= 0;
  end else begin
    state <= {state[2:0], data_in};
    data_out <= (state == 4'b1010);
  end
end
endmodule
```

Testbench:

```
 `timescale 1ns / 1ps
module MealySequenceDetector_tb;
  reg clk;
  reg reset;
  reg data_in;
  wire data_out;
  MealySequenceDetector  g6 (
    .clk(clk),
    .reset(reset),
    .data_in(data_in),
    .data_out(data_out)
  );
  initial begin
    clk = 0;
    forever #5 clk = ~clk;
  end
  initial begin
    reset = 1;
    data_in = 0;
    #10 reset = 0;
  #20 data_in = 1;

  #10 data_in = 0;

  #20 data_in = 1;

  #10 data_in = 0;

  #20 data_in = 1;

  #10 data_in = 0;

  #20 data_in = 1;

  #10 data_in = 0;

  #10 $finish;

 end

Endmodule
```

2)assignment question of counter



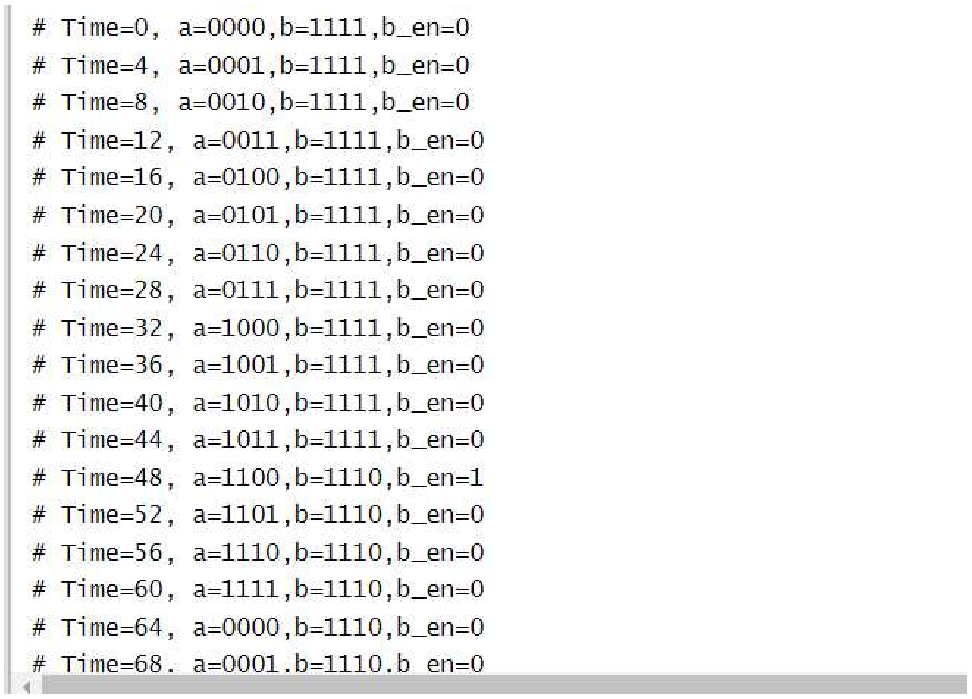
New support email is support@edaplayground.com. (Previous emails to the old Gmail address can't be answered. Please resu

testbench.sv

```
// Code your testbench here
// or browse Examples
module tb_async_counter();
  reg clk;
  reg clear;
  wire [3:0]a,b;
  wire b_en;
  counter_Ab t1(clk,clear,a,b,b_en);
  initial
    begin
      clk=0;
      forever #2 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;
      #200 $finish;
    end
endmodule
```

design.sv

```
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=0,  a=0000,b=1111,b_en=0
# Time=4,  a=0001,b=1111,b_en=0
# Time=8,  a=0010,b=1111,b_en=0
# Time=12,  a=0011,b=1111,b_en=0
# Time=16,  a=0100,b=1111,b_en=0
# Time=20,  a=0101,b=1111,b_en=0
# Time=24,  a=0110,b=1111,b_en=0
# Time=28,  a=0111,b=1111,b_en=0
# Time=32,  a=1000,b=1111,b_en=0
# Time=36,  a=1001,b=1111,b_en=0
# Time=40,  a=1010,b=1111,b_en=0
# Time=44,  a=1011,b=1111,b_en=0
# Time=48,  a=1100,b=1110,b_en=1
# Time=52,  a=1101,b=1110,b_en=0
# Time=56,  a=1110,b=1110,b_en=0
# Time=60,  a=1111,b=1110,b_en=0
# Time=64,  a=0000,b=1110,b_en=0
# Time=68,  a=0001,b=1110,b_en=0
```

Design code:

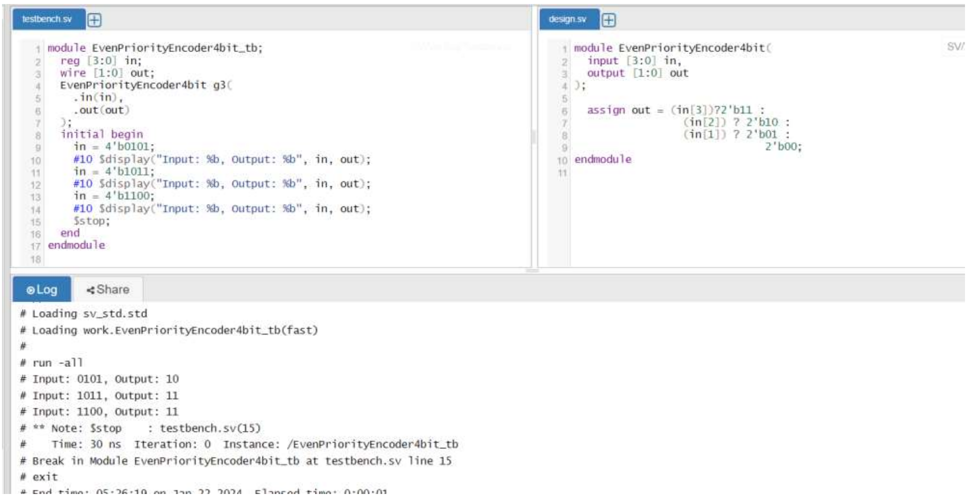
```
module counter_Ab(
 input clk,
 input clear,
 output reg [3:0]a,b,
 output reg b_en
);
 reg out1,out2;
 always @(negedge clk)
  begin
   if (clear)
    begin
     a<=4'b0000;
     b<=4'b1111;
    end
  else
   begin
    a[0] <= ~a[0];
    if (a[0])
     begin
      a[1] <= ~a[1];
      if (a[1]) 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;
 counter_Ab t1(clk,clear,a,b,b_en);
 initial
  begin
   clk=0;
   forever #2 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;
   #200 $finish;
  end
endmodule
```

3. 4bit even priority encoder



testbench.sv

```
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
```

design.sv

```
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
```

Log

```
# 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:

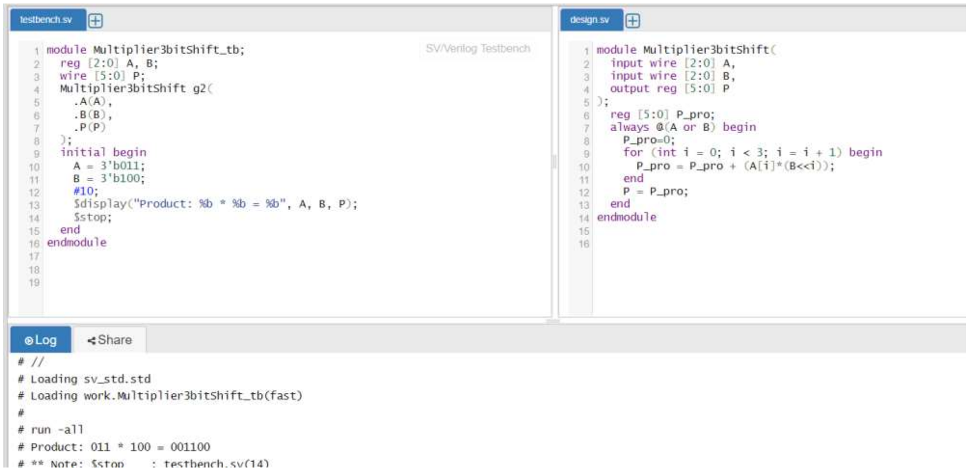
```
module EvenPriorityEncoder4bit(
  input [3:0] in,
  output [1:0] out
);


  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 multipler constructed with shift registers



testbench.sv

SV/Verilog Testbench

```
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
```

design.sv

```
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:

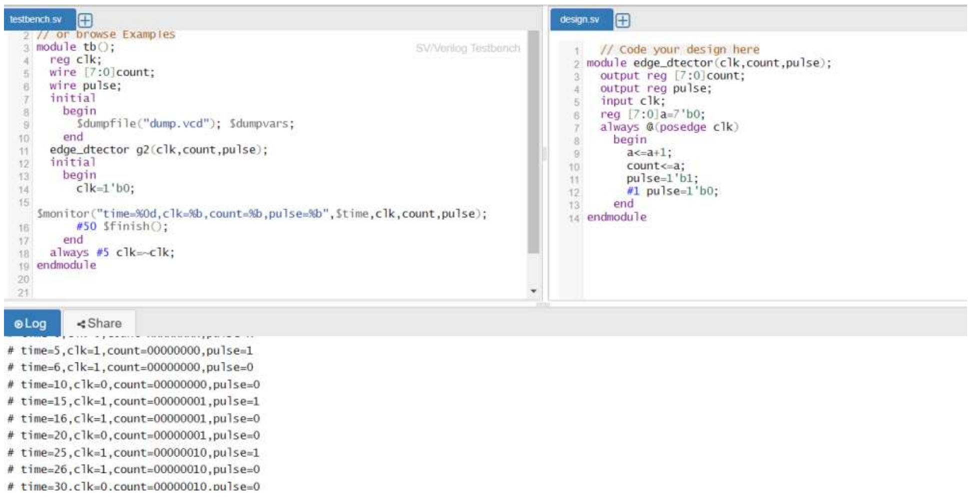
```
module Multiplier3bitShift(
 input wire [2:0] A,
 input wire [2:0] B,
 output reg [5:0] P
);
 reg [5:0] P_pro;
 always @(A or B) begin
  P_pro=0;
  for (int i = 0; i < 3; i = i + 1) begin
   P_pro = P_pro + (A[i]*(B<<i));
  end
  P = P_pro;
 end
Endmodule
```

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
```