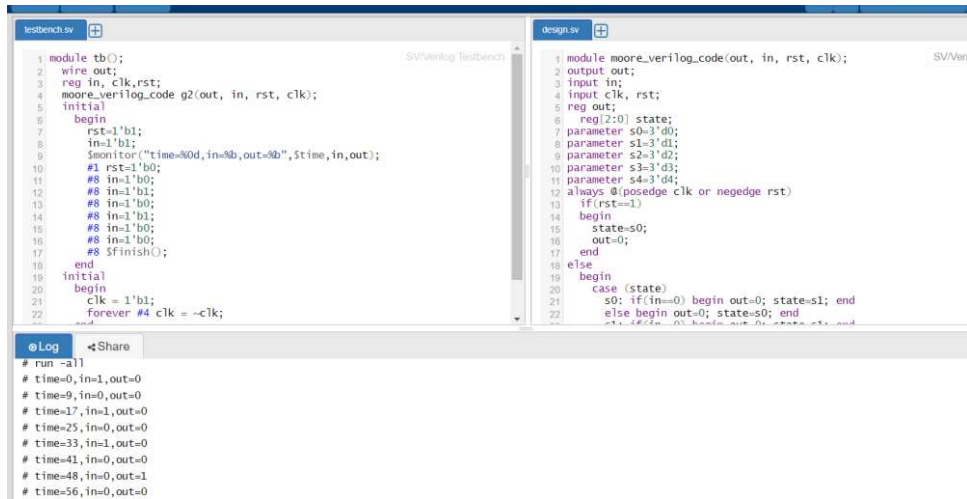


## 1.)verilog code for 4-bit fsm moore and mealy machine



The screenshot shows a Verilog IDE with two tabs: 'testbench sv' and 'design sv'. The 'testbench sv' tab is active, displaying a testbench module 'tb()' that instantiates the 'moore\_verilog\_code' module. It includes an initial block for setting initial values and a monitor for output. The 'design sv' tab shows the 'moore\_verilog\_code' module, which is a Moore FSM for detecting the sequence '0101'. It uses a 3-bit state register and four parameters (s0, s1, s2, s3, s4) to represent states. The logic is implemented using an 'always' block with a reset condition and a case statement for state transitions.

```
1 module tb();
2   wire out;
3   reg in, clk, rst;
4   moore_verilog_code g2(out, in, rst, clk);
5   initial
6   begin
7     rst=1'b1;
8     in=1'b1;
9     $monitor("time=%0d, in=%b, out=%b", $time, in, out);
10    #1 rst=1'b0;
11    #8 in=1'b0;
12    #8 in=1'b1;
13    #8 in=1'b0;
14    #8 in=1'b1;
15    #8 in=1'b0;
16    #8 in=1'b1;
17    #8 $finish();
18  end
19  initial
20  begin
21    clk = 1'b1;
22    forever #4 clk = ~clk;
23  end
endmodule
```

```
1 module moore_verilog_code(out, in, rst, clk);
2   output out;
3   input in;
4   input clk, rst;
5   reg out;
6   reg[2:0] state;
7   parameter s0=3'd0;
8   parameter s1=3'd1;
9   parameter s2=3'd2;
10  parameter s3=3'd3;
11  parameter s4=3'd4;
12  always @(posedge clk or negedge rst)
13  if(rst==1)
14  begin
15    state=s0;
16    out=0;
17  end
18  else
19  begin
20    case (state)
21    s0: if(in==0) begin out=0; state=s1; end
22    else begin out=0; state=s0; end
23    s1: if(in==1) begin out=0; state=s2; end
24    else begin out=0; state=s0; end
25    s2: if(in==0) begin out=0; state=s3; end
26    else begin out=0; state=s0; end
27    s3: if(in==1) begin out=0; state=s4; end
28    else begin out=0; state=s0; end
29    s4: if(in==0) begin out=1; state=s0; end
30    else begin out=0; state=s0; end
31  endcase
32  end
endmodule
```

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=49, in=1, out=1
# time=57, 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

```

The screenshot shows a Verilog IDE with two main windows: 'testbench.v' and 'design.v'. The 'testbench.v' window contains a testbench module that instantiates a mealy machine and applies a sequence of inputs. The 'design.v' window contains the mealy machine logic. A third window at the bottom shows the simulation log, which displays the output of the \$monitor statement at various time intervals.

```

testbench.v
1 // Code your testbench here
2 // or browse Examples
3 module tb();
4     wire out;
5     reg in, clk,rst;
6     mealy_verilog_code g2(out, in, rst, clk);
7     initial
8     begin
9         rst=1'b1;
10        in=1'b1;
11        $monitor("time=%0d,in=%b,out=%b",$time,in,out);
12        #1 rst=1'b0;
13        #8 in=1'b0;
14        #8 in=1'b1;
15        #8 in=1'b0;
16        #8 in=1'b1;
17        #8 in=1'b0;
18        #8 in=1'b0;
19        #8 $finish();
20    end
21    initial
22    begin
23        clk = 1'b1;

```

```

design.v
1 module mealy_verilog_code(out, in, rst, clk);
2     output out;
3     input in;
4     input clk, rst;
5     reg out;
6     reg[1:0] state;
7     parameter s0=2'd0;
8     parameter s1=2'd1;
9     parameter s2=2'd2;
10    parameter s3=2'd3;
11    always @(posedge clk or negedge rst)
12    if(rst==1)
13    begin
14        state=s0;
15        out=0;
16    end
17    else
18    begin
19        case (state)
20        s0: if(in==0) begin out=0; state=s1; end
21            else begin out=0; state=s0; end
22        s1: if(in==0) begin out=0; state=s1; end
23            else begin out=0; state=s2; end
24

```

```

@Log
# 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

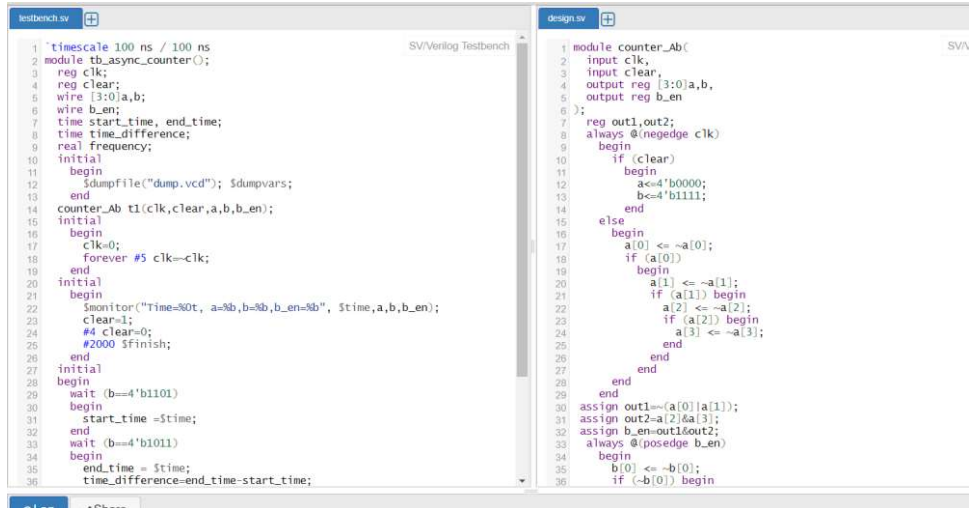
clk = 1'b1;

forever #4 clk = ~clk;

end

endmodule

## 2)assignment question of counter



The screenshot displays a Verilog code editor with two files: 'testbench.sv' and 'design.sv'. The 'testbench.sv' file on the left contains a testbench for a counter module, including time scale settings, module instantiation, and signal monitoring. The 'design.sv' file on the right contains the implementation of the 'counter\_Ab' module, which uses a 4-bit register 'a' and a 1-bit output 'b\_en' to generate a 4-bit output 'out1' and a 3-bit output 'out2'.

```
1 `timescale 100 ns / 100 ns
2 module tb_async_counter();
3   reg clk;
4   reg clear;
5   wire [3:0]a,b;
6   wire b_en;
7   time start_time, end_time;
8   time time_difference;
9   real frequency;
10  initial
11  begin
12    $dumpfile("dump.vcd"); $dumpvars;
13  end
14  counter_Ab t1(clk,clear,a,b,b_en);
15  initial
16  begin
17    clk=0;
18    forever #5 clk=~clk;
19  end
20  initial
21  begin
22    $monitor("Time=%0t, a=%b,b=%b,b_en=%b", $time,a,b,b_en);
23    clear=1;
24    #4 clear=0;
25    #2000 $finish;
26  end
27  initial
28  begin
29    wait (b==4'b1101)
30    begin
31      start_time=$time;
32    end
33    wait (b==4'b1011)
34    begin
35      end_time=$time;
36      time_difference=end_time-start_time;
```

```
1 module counter_Ab(
2   input clk,
3   input clear,
4   output reg [3:0]a,b,
5   output reg b_en
6 );
7   reg out1,out2;
8   always @(negedge clk)
9   begin
10    if (clear)
11    begin
12      a<=4'b0000;
13      b<=4'b1111;
14    end
15    else
16    begin
17      a[0] <= ~a[0];
18      if (a[0])
19      begin
20        a[1] <= ~a[1];
21        if (a[1]) begin
22          a[2] <= ~a[2];
23          if (a[2]) begin
24            a[3] <= ~a[3];
25          end
26        end
27      end
28    end
29    assign out1={a[0]|a[1]};
30    assign out2=a[2]&a[3];
31    assign b_en=out1&out2;
32    always @(posedge b_en)
33    begin
34      b[0] <= ~b[0];
35      if (~b[0]) begin
```

```
Log Share
# 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( decimal 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
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];
    end

```



```

    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
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
endmodule

```

```

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

The screenshot shows a Verilog IDE with two files: testbench.v and design.v. The testbench.v file contains a module EvenPriorityEncoder4bit\_tb with an initial block and a \$stop statement. The design.v file contains the module EvenPriorityEncoder4bit with a 4-bit input, a 3-bit output, and a priority encoder logic block. The log window at the bottom shows the simulation results, including the input and output values for the testbench.

```

testbench.v
1 module EvenPriorityEncoder4bit_tb;
2   reg [3:0] in;
3   wire [1:0] out;
4   EvenPriorityEncoder4bit g3(
5     .in(in),
6     .out(out)
7   );
8   initial begin
9     in = 4'b0101;
10    #10 $display("Input: %b, Output: %b", in, out);
11    in = 4'b1011;
12    #10 $display("Input: %b, Output: %b", in, out);
13    in = 4'b1100;
14    #10 $display("Input: %b, Output: %b", in, out);
15    $stop;
16  end
17 endmodule

design.v
1 module EvenPriorityEncoder4bit(
2   input [3:0] in,
3   output [1:0] out
4 );
5
6   assign out = (in[3])?2'b11 :
7                 (in[2])? 2'b10 :
8                 (in[1])? 2'b01 :
9                 2'b00;
10 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.v(15)
# Time: 30 ns Iteration: 0 Instance: /EvenPriorityEncoder4bit_tb
# Break in Module EvenPriorityEncoder4bit_tb at testbench.v line 15
# exit
# End time: 05:26:10 on Tue 22 2022 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;

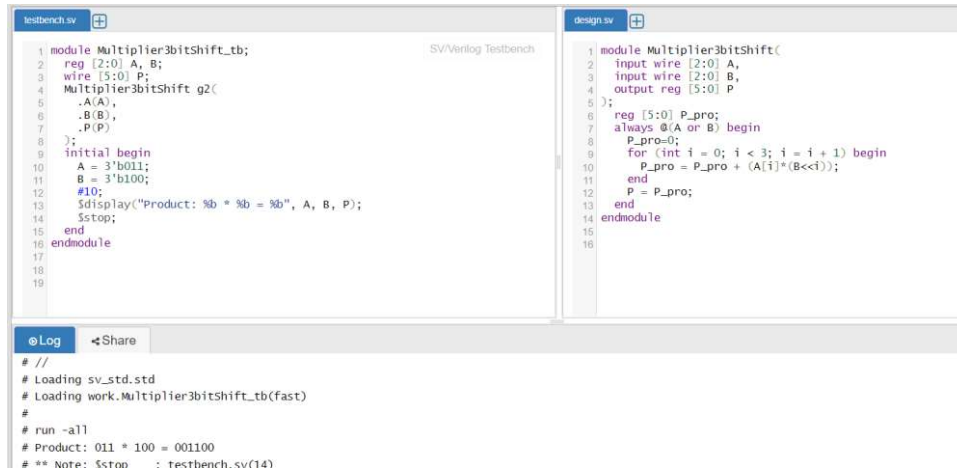
        #10 $display("Input: %b, Output: %b", in, out);

        $stop;

    end

Endmodule
```

## 4. 3 bit multiplier using shift registers



The screenshot shows a Verilog IDE with two panels. The left panel, titled 'testbench.sv', contains a testbench for a 3-bit multiplier. It defines a module 'Multiplier3bitShift\_tb' with a 3-bit register 'A', a 5-bit register 'P', and an instance of the multiplier module. It sets initial values for 'A' (3'b011) and 'B' (3'b100), and includes a display statement to show the product. The right panel, titled 'design.sv', contains the implementation of the 'Multiplier3bitShift' module. It takes two 3-bit inputs 'A' and 'B', and produces a 5-bit output 'P'. The logic uses a 5-bit register 'P\_pro' and an 'always' block triggered by changes in 'A' or 'B'. Inside the block, 'P\_pro' is initialized to 0, and a loop iterates over the bits of 'B' (from 0 to 2), shifting 'A' left by the current bit index and adding it to 'P\_pro'.

```
1 module Multiplier3bitShift_tb;
2   reg [2:0] A, B;
3   wire [5:0] P;
4   Multiplier3bitShift g2(
5     .A(A),
6     .B(B),
7     .P(P)
8   );
9   initial begin
10    A = 3'b011;
11    B = 3'b100;
12    #10;
13    $display("Product: %b * %b = %b", A, B, P);
14    $stop;
15  end
16 endmodule

1 module Multiplier3bitShift(
2   input wire [2:0] A,
3   input wire [2:0] B,
4   output reg [5:0] P
5 );
6   reg [5:0] P_pro;
7   always @(A or B) begin
8     P_pro=0;
9     for (int i = 0; i < 3; i = i + 1) begin
10      P_pro = P_pro + (A[i]*(B<<i));
11    end
12    P = P_pro;
13  end
14 endmodule
```

Log

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

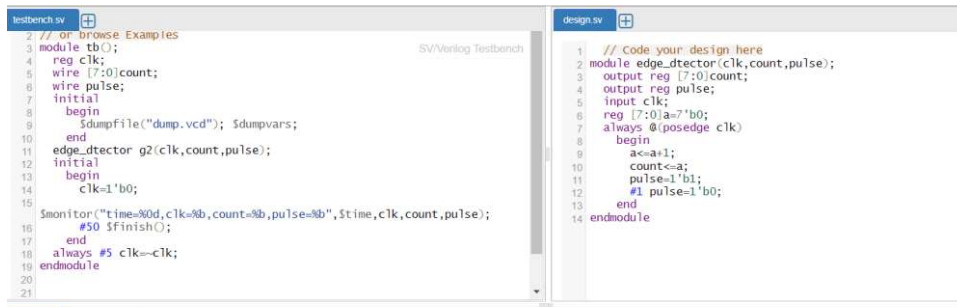
        $display("Product: %b * %b = %b", A, B, P);

        $stop;

    end

endmodule
```

## 5. rising and falling edge detector



The screenshot shows a Verilog IDE with two panels. The left panel, titled 'testbench.sv', contains a testbench module 'tb()' that instantiates the 'edge\_detector' module and uses '\$monitor' to track its outputs. The right panel, titled 'design.sv', contains the 'edge\_detector' module code. Below the code panels, a 'Log' window displays the simulation results, showing the values of 'clk', 'count', and 'pulse' at various time intervals.

```
testbench.sv
2 // or browse Examples
3 module tb();
4   reg clk;
5   wire [7:0] count;
6   wire pulse;
7   initial
8   begin
9     $dumpfile("dump.vcd"); $dumpvars;
10  end
11  edge_detector g2(clk, count, pulse);
12  initial
13  begin
14    clk=1'b0;
15  end
16  $monitor("time=%0d, clk=%b, count=%b, pulse=%b", $time, clk, count, pulse);
17  #50 $finish();
18  always #5 clk=~clk;
19 endmodule

design.sv
1 // Code your design here
2 module edge_detector(clk, count, pulse);
3   output reg [7:0] count;
4   output reg pulse;
5   input clk;
6   reg [7:0] a=7'b0;
7   always @(posedge clk)
8   begin
9     a<=a+1;
10    count<=a;
11    pulse=1'b1;
12    #1 pulse=1'b0;
13  end
14 endmodule

Log
# 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_detector(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_detector 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
```