

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

```
1 `timescale 1ns / 1ps
2 module SequenceDetector_tb;
3   reg clk;
4   reg reset;
5   reg data_in;
6   wire data_out;
7   SequenceDetector uut (
8     .clk(clk),
9     .reset(reset),
10    .data_in(data_in),
11    .data_out(data_out)
12  );
13  initial begin
14    reset = 1;
15    data_in = 0;
16    $monitor("time=%0d,din=%b,data_out=%b", $time, data_in, data_out);
17    #10 reset = 0;
18    #20 data_in = 1;
19    #10 data_in = 0;
20    #20 data_in = 1;
21    #10 data_in = 0;
22    #20 data_in = 1;
23    #10 data_in = 0;
24    #20 data_in = 1;
25    #10 data_in = 0;
26  end
27  Log
28  < Share
29  :ime=70,din=0,data_out=0
30  :ime=90,din=1,data_out=0
31  :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;

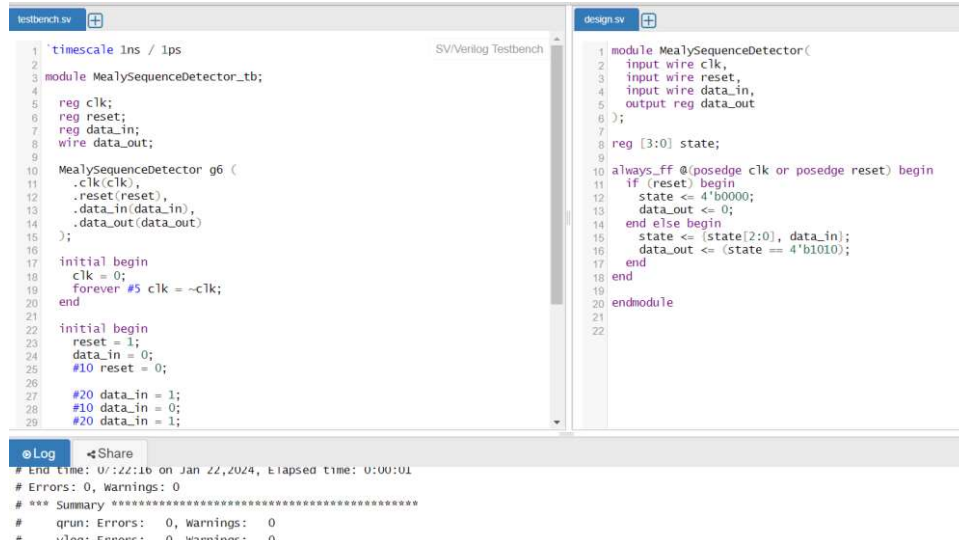
    #10 data_in = 0;

```

```
#10 $finish;
```

```
end
```

```
Endmodule
```



The screenshot shows a Verilog IDE with two files open: 'testbench.sv' and 'design.sv'. The 'testbench.sv' file contains a testbench for the 'MealySequenceDetector' module, including clock and reset generation, and data input sequences. The 'design.sv' file contains the implementation of the 'MealySequenceDetector' module, which uses a 4-bit state register to detect the sequence 1010. The bottom status bar shows the simulation results: 'End time: 07:22:16 on Jan 22, 2024, Elapsed time: 0:00:01', 'Errors: 0, Warnings: 0', and a summary of the run.

```
testbench.sv
1 `timescale 1ns / 1ps
2
3 module MealySequenceDetector_tb;
4
5 reg clk;
6 reg reset;
7 reg data_in;
8 wire data_out;
9
10 MealySequenceDetector g6 (
11     .clk(clk),
12     .reset(reset),
13     .data_in(data_in),
14     .data_out(data_out)
15 );
16
17 initial begin
18     clk = 0;
19     forever #5 clk = ~clk;
20 end
21
22 initial begin
23     reset = 1;
24     data_in = 0;
25     #10 reset = 0;
26
27     #20 data_in = 1;
28     #10 data_in = 0;
29     #20 data_in = 1;
30 end
31
32 endmodule
```

```
design.sv
1 module MealySequenceDetector(
2     input wire clk,
3     input wire reset,
4     input wire data_in,
5     output reg data_out
6 );
7
8 reg [3:0] state;
9
10 always_ff @(posedge clk or posedge reset) begin
11     if (reset) begin
12         state <= 4'b0000;
13         data_out <= 0;
14     end else begin
15         state <= {state[2:0], data_in};
16         data_out <= (state == 4'b1010);
17     end
18 end
19
20 endmodule
```

Log | Share
End time: 07:22:16 on Jan 22, 2024, Elapsed time: 0:00:01
Errors: 0, Warnings: 0
*** Summary *****
run: 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;
```

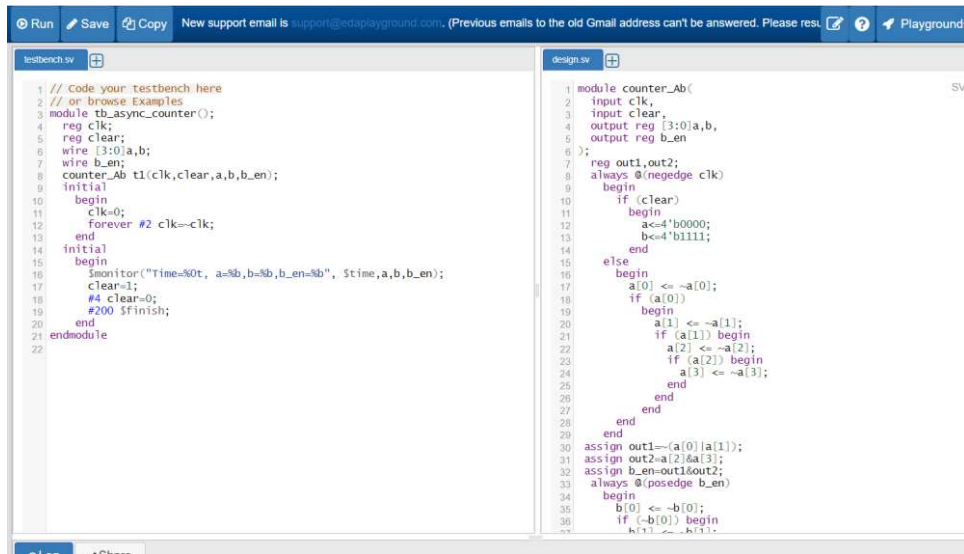
```
#10 data_in = 0;
```

```
#10 $finish;
```

```
end
```

```
Endmodule
```

2)assignment question of counter



The screenshot shows a Verilog code editor with two files: 'testbench.sv' and 'design.sv'. The 'testbench.sv' file contains a testbench for a counter module, and the 'design.sv' file contains the counter module code.

```
testbench.sv
1 // Code your testbench here
2 // or browse Examples
3 module tb_async_counter();
4   reg clk;
5   reg clear;
6   wire [3:0]a,b;
7   wire b_en;
8   counter_Ab t1(clk,clear,a,b,b_en);
9   initial
10    begin
11      clk=0;
12      forever #2 clk=clk;
13    end
14   initial
15    begin
16      $monitor("Time=%0t, a=%b,b=%b,b_en=%b", $time,a,b,b_en);
17      clear=1;
18      #4 clear=0;
19      #200 $finish;
20    end
21 endmodule
22
```

```
design.sv
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
36        b[1] <= ~b[1];
37      end
38    end
39  end
40 endmodule
```

```

# 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
    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

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);

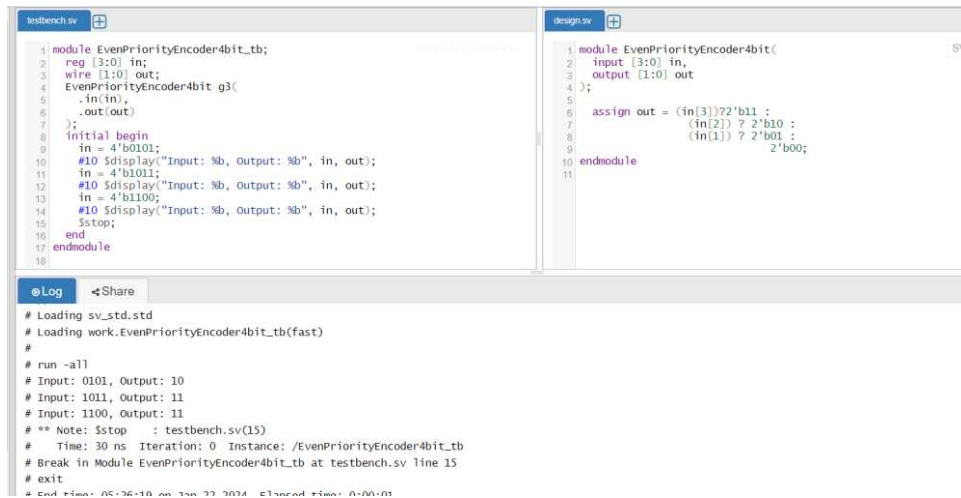
    initial
    begin
        clk=0;
        forever #2 clk=~clk;
    end

    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
```

end

endmodule

3. 4bit even priority encoder



The screenshot shows a Verilog IDE with two panels. The left panel, titled 'testbench.sv', contains a testbench for the 'EvenPriorityEncoder4bit' module. It defines a 4-bit input 'in' as a register, connects it to an instance 'g3' of the module, and includes an initial block with three test cases: '0101', '1011', and '1100'. The right panel, titled 'design.sv', contains the module definition for 'EvenPriorityEncoder4bit'. It takes a 4-bit input 'in' and produces a 2-bit output 'out'. The logic uses a case statement to assign 'out' based on the input: '0101' results in '2'b11', '1011' in '2'b10', '1100' in '2'b01', and all other inputs result in '2'b00'.

```
testbench.sv
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
18

design.sv
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
11
```

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: 05:26:10 on Jan 27 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);

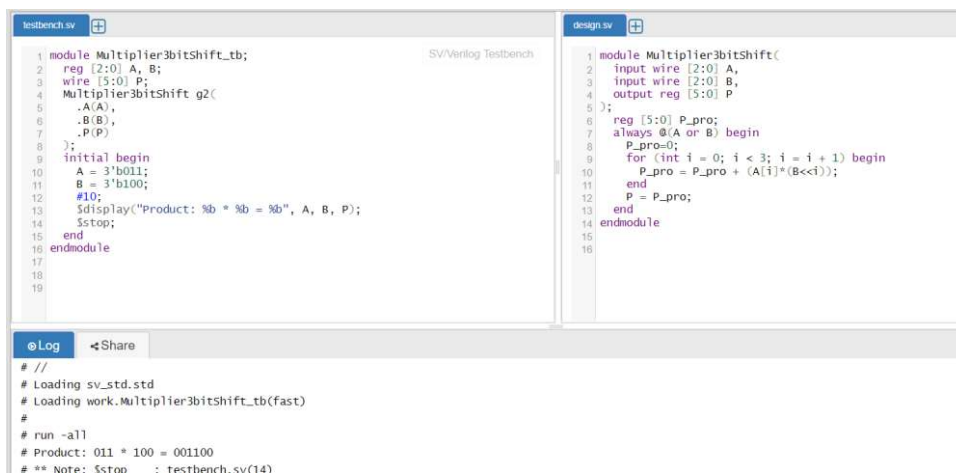
$stop;

end

Endmodule

```

4. 3 bit multiplier constructed with shift registers



```

testbench.sv
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
17
18
19
design.sv
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
15
16
Log
Share
# //
# Loading sv_std.std
# Loading work.Multiplier3bitShift_tb(Fast)
#
# run -all
# Product: 011 * 100 = 001100
# ** Note: $<root : 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 files: `testbench.sv` and `design.sv`.

testbench.sv:

```

1 // or browse Examples
2 module tb();
3   reg clk;
4   wire [7:0] count;
5   wire pulse;
6   initial
7   begin
8     $dumpfile("dump.vcd"); $dumpvars;
9   end
10  edge_detector g2(clk, count, pulse);
11  initial
12  begin
13    clk = 1'b0;
14  end
15  $monitor("time=%0d, clk=%b, count=%b, pulse=%b", $time, clk, count, pulse);
16  #50 $finish();
17 end
18 always #5 clk = ~clk;
19 endmodule
20
21

```

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
```