Assignment submitted by : P. Hrithik Roshan

Email Id : hrithikroshan5620@gmail.com

College : CBIT, Hyderabad

Assignment Solutions

1. Design an ALU with 3 inputs i.e two 16-bit signed values and a 4-bit control signal which perform ALU Operation? The ALU has 16 bit signed output?

Control Signal	**ALU Operation**
0000	Out = A xor b
0001	Out =A and b
0010	Out =A or B
0011	Out = A xnor B
0100	Out = A nand B
0101	Out = A nor B
0110	Out = A + B
0111	Out = A - B
1000	Out = A*B
1001	Out = A/B
1010	Out = A << 1
1011	Out = A>>1
1100	Out = A/B rotated left by 1
1101	Out = A/B rotated right by 1
1110	Out = A > B
1111	Out = A < B

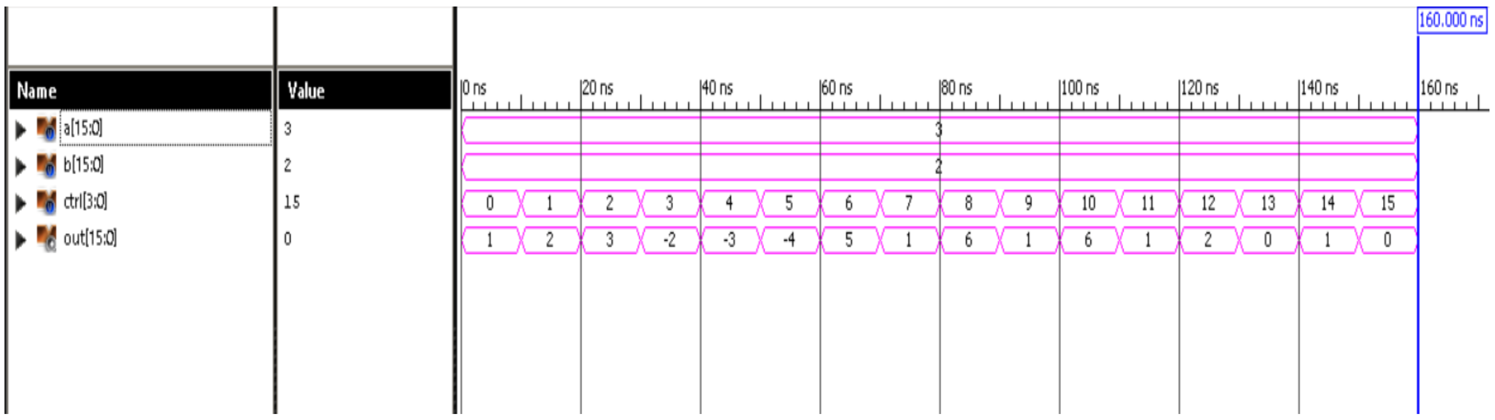
Source Code

```
`timescale 1ns / 1ps

module alu( a,b,ctrl,out );

input signed [15:0]a,b;
input [3:0]ctrl;

output reg signed [15:0]out;

always@(*)

   begin

          case(ctrl)
          0:out=a^b;
          1:out=a&b;
          2:out=a|b;
          3:out= (a~^b) ;
          4:out= ~(a&b);
          5:out= ~(a|b);
          6:out=a+b;
          7:out=a-b;
          8:out=a*b;
          9:out=a/b;
          10:out=a<<1;
          11:out=a>>1;
          12:out=(a/b)<<1;
          13:out=(a/b)>>1;
          14:out=a>b;
          15:out=a<b;


          default: $display("_Invalid_");


   endcase
end


endmodule
```

Test Bench Code

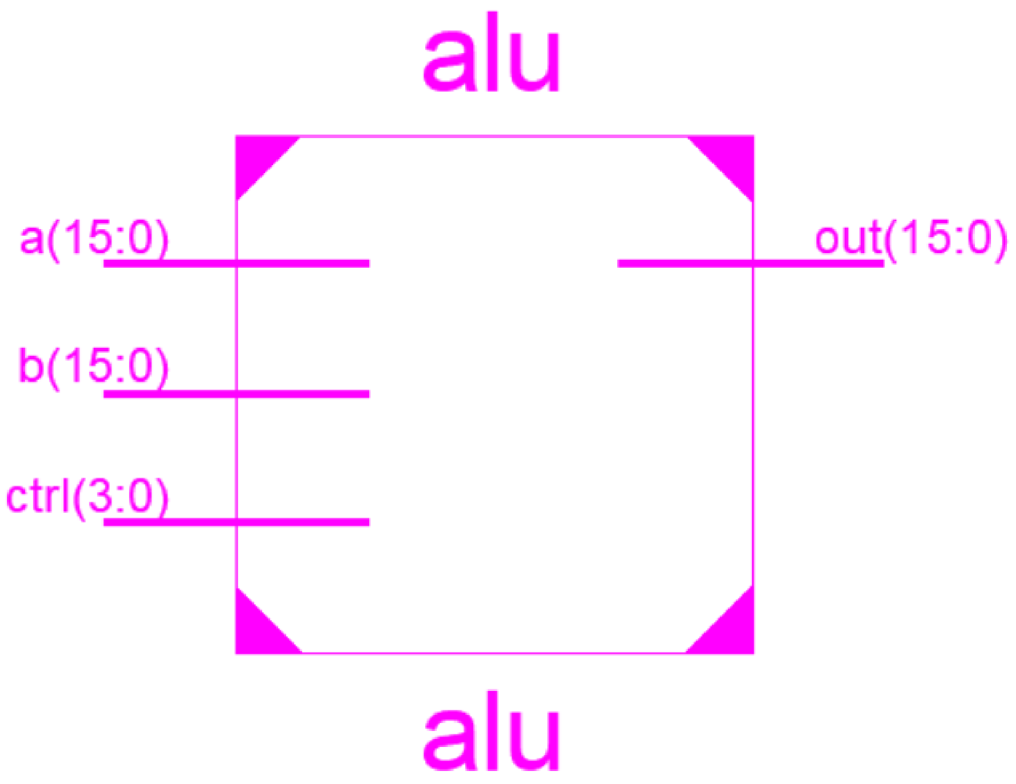
```
`timescale 1ns / 1ps

module tb_alu;

  // Inputs
  reg [15:0] a;
  reg [15:0] b;
  reg [3:0] ctrl;

  // Outputs
  wire [15:0] out;

  // Instantiate the Unit Under Test (UUT)

  alu uut (
            .a(a),
            .b(b),
            .ctrl(ctrl),
            .out(out)
  );



  initial begin
            // Initialize Inputs
            a = 3;
            b = 2;
            ctrl = 0;
            ctrl = 0;  #160;
            $stop;

  end


            always
            #10 ctrl=ctrl+1;

Endmodule
```

Simulation Output



```
a[15:0]     3    3
b[15:0]     2    2
ctrl[3:0]   15   0  1  2  3   4   5   6  7  8  9  10  11  12  13  14  15
out[15:0]   0    1  2  3  -2  -3  -4  5  1  6  1  6   1   2   0   1   0
```

RTL Schematic



2. Simulate and implement binary “101001/101000” sequence detector for the following:

(a) Moore FSM without sequence overlap

(b) Moore FSM with sequence overlap

(c) Mealy FSM without sequence overlap

(d) Mealy FSM with sequence overlap

A. Moore FSM without sequence overlap

Source Code

```
`timescale 1ns / 1ps
module moore_without (sequence_in, clock, reset, out );
input clock;
input reset;
input sequence_in;
output reg out;
parameter  A=3'b000,
  B=3'b001,
  C=3'b010,
  D=3'b011,
  E=3'b100,
  F=3'b101,
  G=3'b110;
reg [2:0] current_state, next_state;

always @(posedge clock, posedge reset)
begin
 if(reset==1)
 current_state <= A;
 else
 current_state <= next_state;
end


always @(current_state,sequence_in)
begin
```

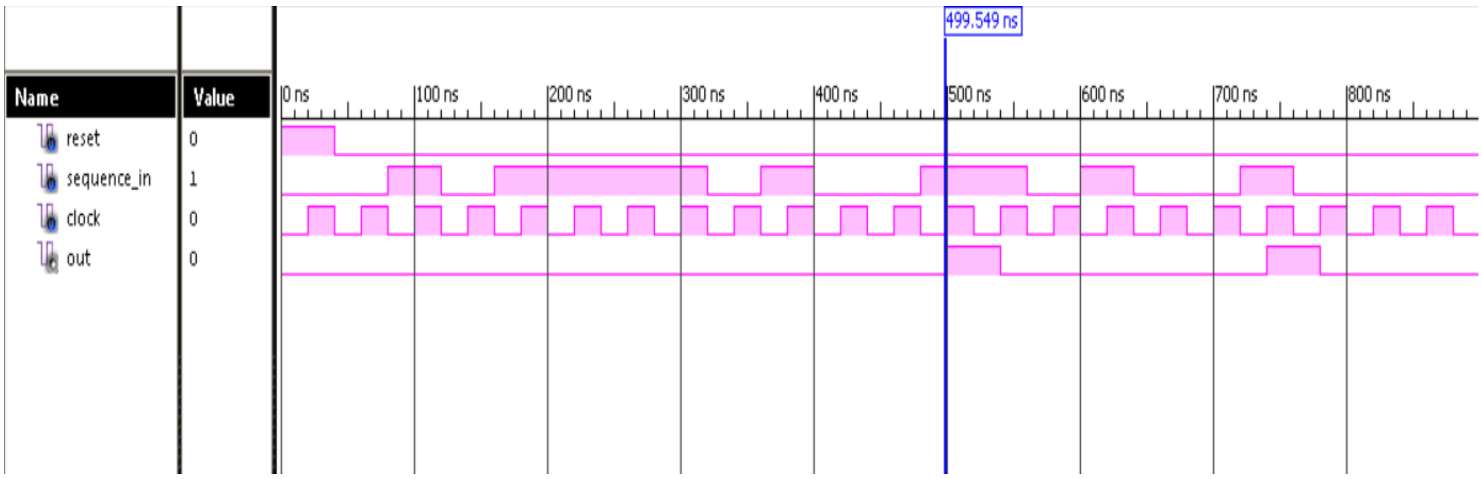
```
case(current_state)
A:begin
 if(sequence_in==1)
  next_state = B;
 else
  next_state = A;
end

B:begin
 if(sequence_in==1)
  next_state = B;
 else
  next_state = C;
end

C:begin
 if(sequence_in==1)
  next_state = D;
 else
  next_state = A;
end

D:begin
 if(sequence_in==1)
  next_state = B;
 else
  next_state = E;
end

E:begin
 if(sequence_in==1)
  next_state = D;
 else
  next_state = F;
end

F:begin
 if(sequence_in==1)
  next_state = G;
 else
  next_state = A;
end

G:begin
 if(sequence_in==1)
  next_state = B;
 else
```

```
   next_state = A;
 end




 default:next_state = A;
 endcase
end




always @(current_state)
begin
 case(current_state)
A : out = 0;
B : out = 0;
C : out = 0;
D : out = 0;
E : out = 0;
F : out = 0;
G : out = 1;

 default:  out = 0;
 endcase
end

endmodule
```

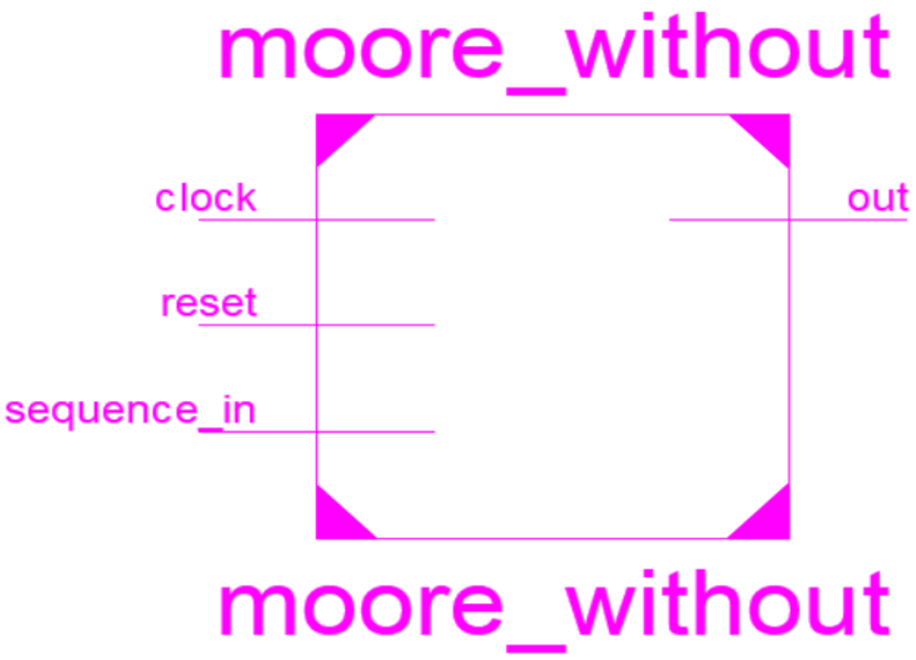
Test Bench Code

```
`timescale 1ns / 1ps
module tb_moore_without;

        reg sequence_in;
        reg clock;
        reg reset;
        wire out;
        moore_without uut (
                .sequence_in(sequence_in),
                .clock(clock),
                .reset(reset),
                .out(out)
        );

        initial begin

                clock = 0;
                forever #20 clock = ~clock;
        end
        initial begin

                sequence_in = 0;
                reset = 1;
                #40;

                reset = 0;
                #40;

                // 101001
                sequence_in = 1; #40;
                sequence_in = 0; #40;
                sequence_in = 1; #40;
                sequence_in = 1; #40;
                sequence_in = 1; #40;

                sequence_in = 1; #40;
                sequence_in = 0; #40;
                sequence_in = 1; #40;
                sequence_in = 0; #40;
                sequence_in = 0; #40;
                sequence_in = 1; #40; //done

                sequence_in = 1; #40;
                sequence_in = 0; #40;
                sequence_in = 1; #40;
                sequence_in = 0; #40;
                sequence_in = 0; #40;
                sequence_in = 1; #40; //done
                sequence_in = 0; #40;
                sequence_in = 0; #40;
        end
endmodule
```

Simulation Output



RTL Schematic



B. Moore FSM with sequence overlap

Source Code

```
`timescale 1ns / 1ps
module moore_with(sequence_in,clock,reset,out
    );

input clock;
input reset;
input sequence_in;
output reg out;
parameter  A=3'b000,
  B=3'b001,
  C=3'b010,
  D=3'b011,
  E=3'b100,
  F=3'b101,
  G=3'b110;

reg [2:0] current_state, next_state;

always @(posedge clock, posedge reset)
begin
 if(reset==1)
 current_state <= A;
 else
 current_state <= next_state;
end

always @(current_state,sequence_in)
begin
 case(current_state)
 A:begin
  if(sequence_in==1)
   next_state = B;
  else
   next_state = A;
 end
 B:begin
  if(sequence_in==1)
   next_state = B;
  else
   next_state = C;
 end

 C:begin
  if(sequence_in==1)
   next_state = D;
  else
   next_state = A;
 end

 D:begin
  if(sequence_in==1)
   next_state = B;
  else
   next_state = E;
 end

 E:begin
  if(sequence_in==1)
   next_state = D;
  else
   next_state = F;
 end

 F:begin
  if(sequence_in==1)
   next_state = G;
  else
   next_state = A;
 end

 G:begin
  if(sequence_in==1)
   next_state = B;
  else
   next_state = C;
 end

 default:next_state = A;
 endcase
end
```

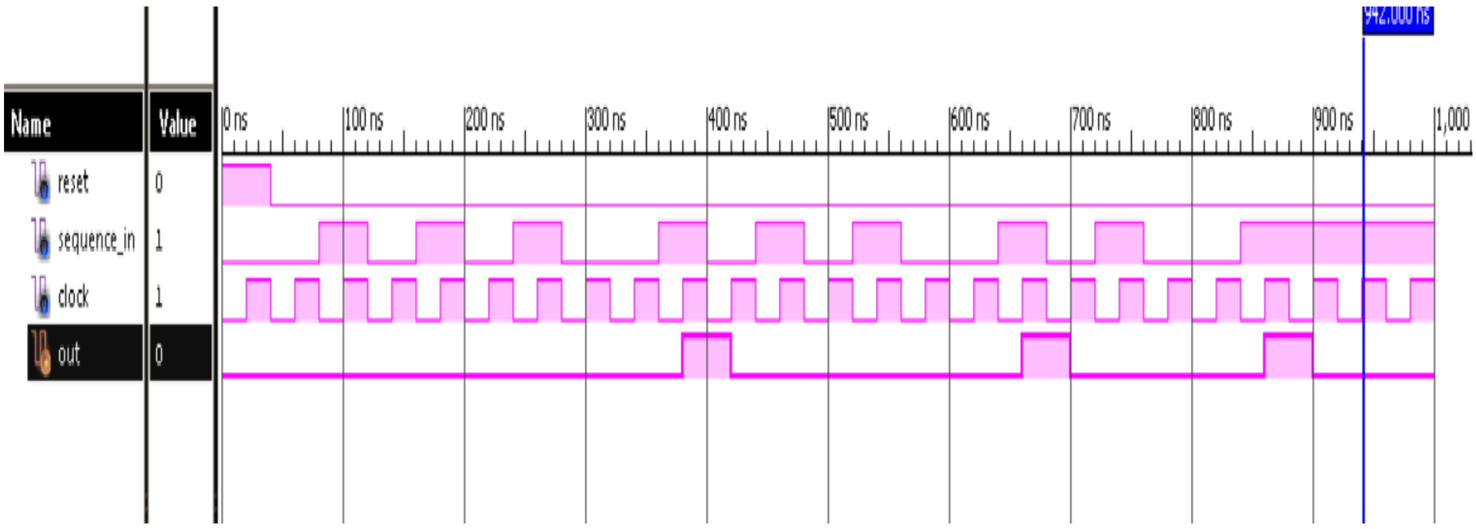
```
always @(current_state)
begin
 case(current_state)
A : out = 0;
B : out = 0;
C : out = 0;
D : out = 0;
E : out = 0;
F : out = 0;
G : out = 1;

 default:  out = 0;
 endcase
end

endmodule
```

Test Bench Code

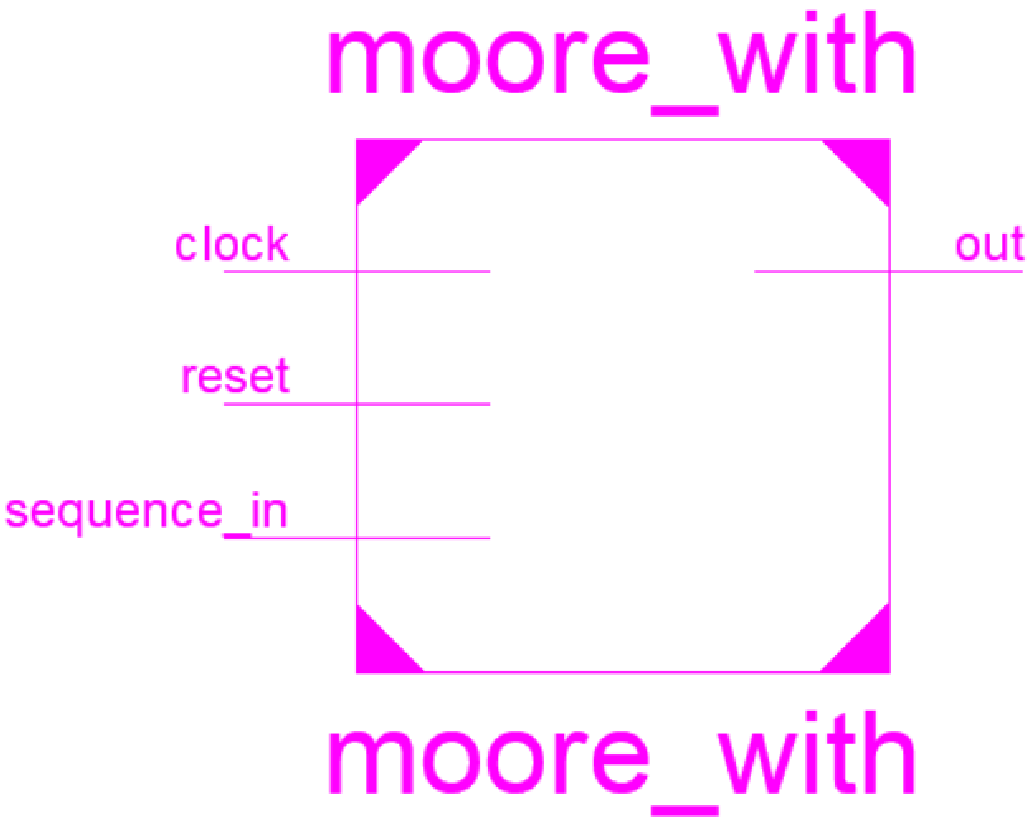
```
`timescale 1ns / 1ps
module tb_moore_with;
        reg sequence_in;
        reg clock;
        reg reset;
        wire out;


        moore_with uut (
                .sequence_in(sequence_in),
                .clock(clock),
                .reset(reset),
                .out(out)
        );


        initial begin
                clock = 0;
                forever #20 clock = ~clock;
        end


        initial begin
```

```
                        sequence_in = 0;
                        reset = 1;
                        #40;

                        reset = 0;
                        #40;

                        // 101001

                        sequence_in = 1; #40;
                        sequence_in = 0; #40;

                        sequence_in = 1; #40;
                        sequence_in = 0; #40;
                        sequence_in = 1; #40;
                        sequence_in = 0; #40;
                        sequence_in = 0; #40;
                        sequence_in = 1; #40;//done

                        sequence_in = 0; #40; // break the seq

                        sequence_in = 1; #40;
                        sequence_in = 0; #40;
                        sequence_in = 1; #40;
                        sequence_in = 0; #40;
                        sequence_in = 0; #40;
                        sequence_in = 1; #40; //done

                        sequence_in = 0; #40;
                        sequence_in = 1; #40;
                        sequence_in = 0; #40;
                        sequence_in = 0; #40;
                        sequence_in = 1; #40; //done
                  end
            endmodule
```

Simulation Output



RTL Schematic



C. Mealy FSM without sequence overlap

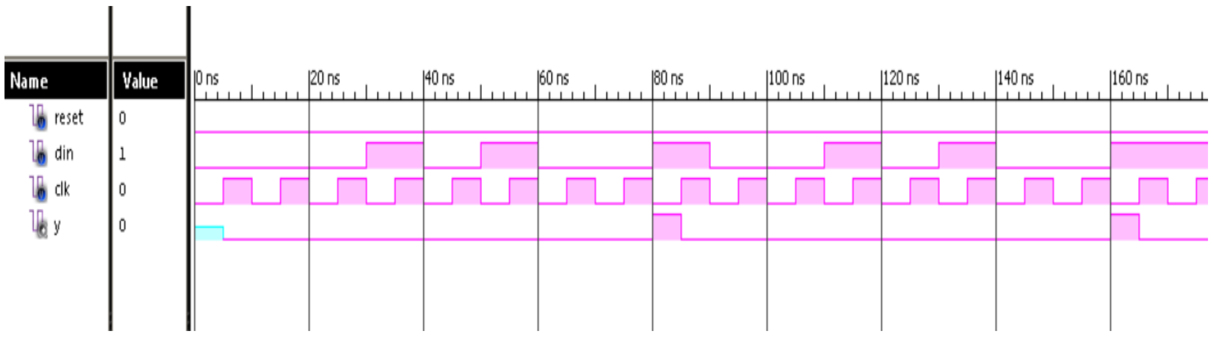
Source Code

```
`timescale 1ns / 1ps
module mealy_without(din, reset, clk, y);
input din;
input clk;
input reset;
output reg y;
reg [2:0] cst, nst;
parameter A = 3'b000, B = 3'b001,C = 3'b010, D = 3'b011,E = 3'b100, F = 3'b101;
always @(cst or din)
 begin
 case (cst)
   A: if (din == 1'b1)
          begin
         nst = B;
          y=1'b0;
          end
      else
          begin
          nst = A;
          y=1'b0;
          end

   B: if (din == 1'b1)
          begin
         nst = B;
          y=1'b0;
          end
      else
          begin
          nst = C;
          y=1'b0;
          end

   C: if (din == 1'b1)
          begin
         nst = D;
          y=1'b0;
          end
      else
          begin
          nst = A;
          y=1'b0;
          end

   D: if (din == 1'b1)
          begin
         nst = B;
          y=1'b0;
          end
      else
          begin
          nst = E;
          y=1'b0;
          end

   E: if (din == 1'b1)
          begin
         nst = D;
          y=1'b0;
          end
      else
          begin
          nst = F;
          y=1'b0;
          end

   F: if (din == 1'b1)
          begin
         nst = A;
          y=1'b1;
          end
      else
          begin
          nst = A;
          y=1'b0;
          end
   default: nst = A;
  endcase
end

always@(posedge clk)
          begin
           if (reset)
             cst <= A;
           else
             cst <= nst;
          end
endmodule
```

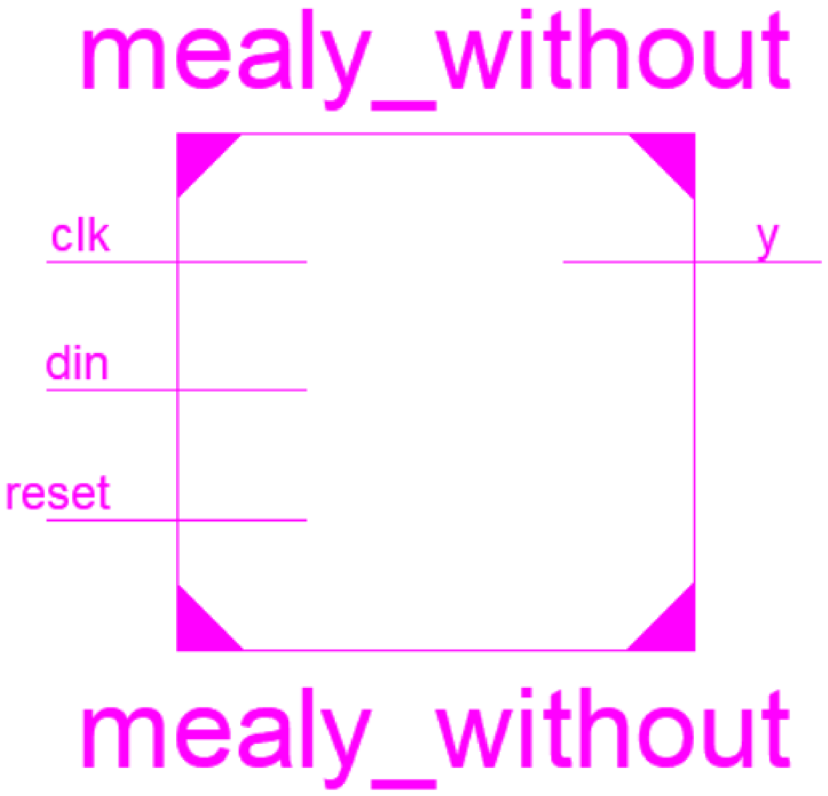
Test Bench Code

```
`timescale 1ns / 1ps
module tb_mealy_without;
	reg din;
	reg reset;
	reg clk;
	wire y;
	mealy_without uut (
		.din(din),
		.reset(reset),
		.clk(clk),
		.y(y)
	);
	initial begi
		din = 0;
		reset = 0;
		clk = 0;   #20;
		// 101001 is our sequence
		#10 din=1;
		#10 din=0;
		#10 din=1;
		#10 din=0;
		#10 din=0;
		#10 din=1; //done

		#10 din=0;
		#10 din=0;

		#10 din=1;
		#10 din=0;
		#10 din=1;
		#10 din=0;
		#10 din=0;
		#10 din=1; //done
	end
	always
	#5 clk=~clk;

endmodule
```

Simulation Output



RTL Schematic



D. Mealy FSM with sequence overlap

Source Code

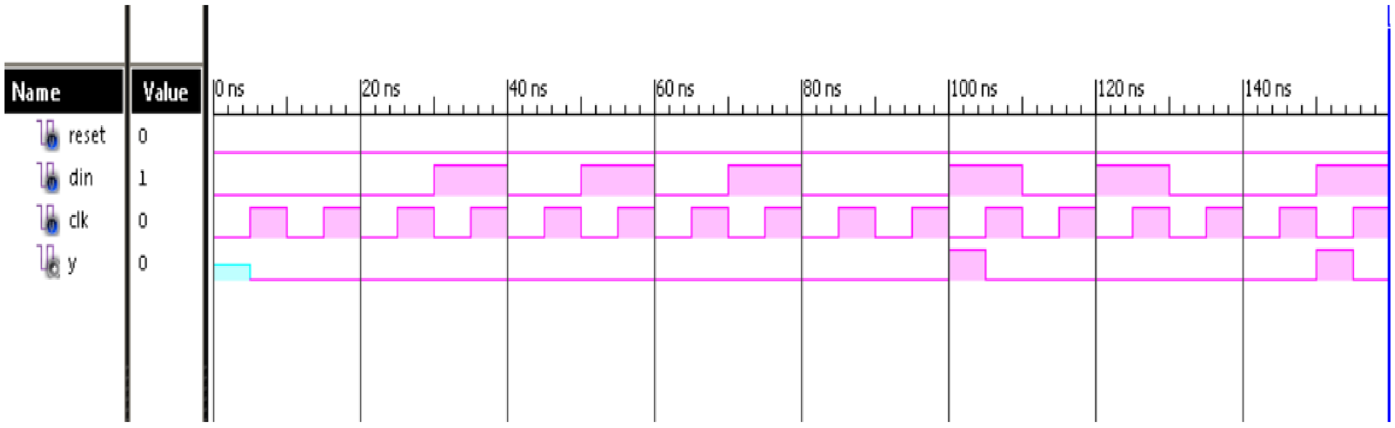
```
`timescale 1ns / 1ps

module mealy_with(din, reset, clk, y);
input din;
input clk;
input reset;
output reg y;
reg [2:0] cst, nst;
parameter A = 3'b000,B = 3'b001,C = 3'b010,D = 3'b011, E = 3'b100,F = 3'b101;
always @(cst or din)
 begin
 case (cst)
   A: if (din == 1'b1)
        begin
       nst = B;
        y=1'b0;
        end
     else
        begin
        nst = A;
        y=1'b0;
        end

   B: if (din == 1'b1)
        begin
       nst = B;
        y=1'b0;
        end
     else
        begin
        nst = C;
        y=1'b0;
        end

   C: if (din == 1'b1)
        begin
       nst = D;
        y=1'b0;
        end
     else
        begin
        nst = A;
```

```
         y=1'b0;
         end

   D: if (din == 1'b1)
         begin
        nst = B;
         y=1'b0;
         end
     else
         begin
         nst = E;
         y=1'b0;
         end

   E: if (din == 1'b1)
         begin
        nst = D;
         y=1'b0;
         end
     else
         begin
         nst = F;
         y=1'b0;
         end

   F: if (din == 1'b1)
         begin
        nst = B;
         y=1'b1;
         end
     else
         begin
         nst = A;
         y=1'b0;
         end

   default: nst = A;
  endcase
end
always@(posedge clk)
         begin
          if (reset)
            cst <= A;
          else
            cst <= nst;
         end
endmodule
```

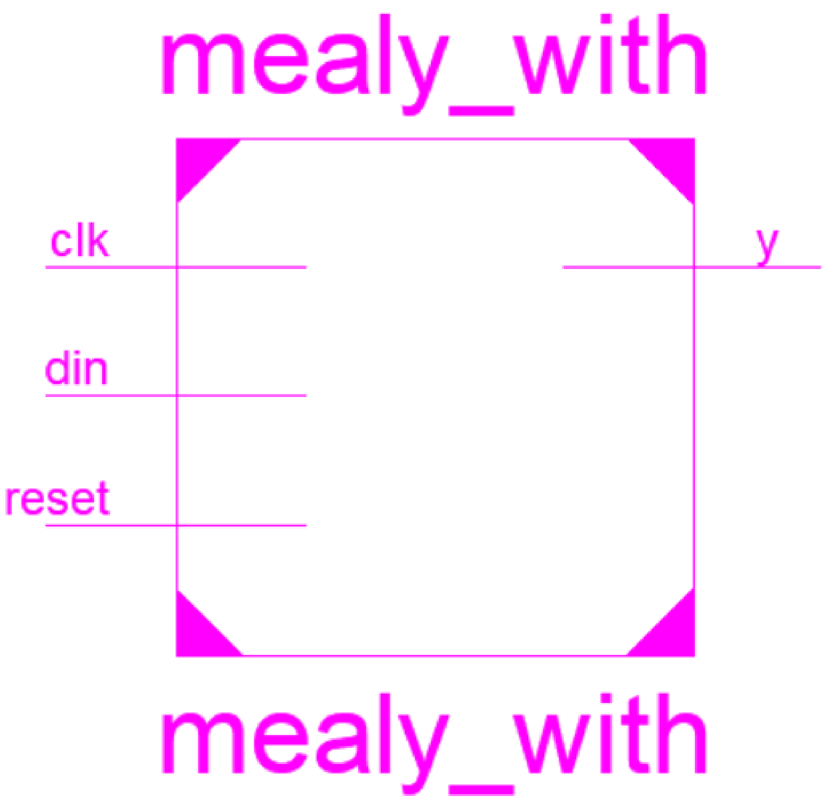
Test Bench Code

```
`timescale 1ns / 1ps
module tb_mealy_with;
    reg din;
    reg reset;
    reg clk;
    wire y;
    mealy_with uut (
              .din(din),
              .reset(reset),
              .clk(clk),
              .y(y)
    );

    initial begin

              din = 0;
              reset = 0;
              clk = 0;
              #20;
              //101001
              #10 din=1;
              #10 din=0;

              #10 din=1;
              #10 din=0;
              #10 din=1;
              #10 din=0;
              #10 din=0;
              #10 din=1; //done

              #10 din=0;
              #10 din=1;
              #10 din=0;
              #10 din=0;
              #10 din=1; //done

    end

    always
    #5 clk=~clk;


endmodule
```

Simulation Output



RTL Schematic


mealy_with
clk
din
reset
y
mealy_with