CS220 Assignment 3

1. Sequence Detector (1010):

A sequential detector is a sequential state machine that takes an input string of bits and generates an output 1 whenever the target sequence (1010) is detected. It is implemented by constructing a Finite State Machine (FSM) which allows for overlapping too, i.e, the last bit of one sequence becomes first bit of the next sequence.

We are implementing it through a Mealy machine.

In a Mealy machine, output depends on the present state as well as the external input (X).

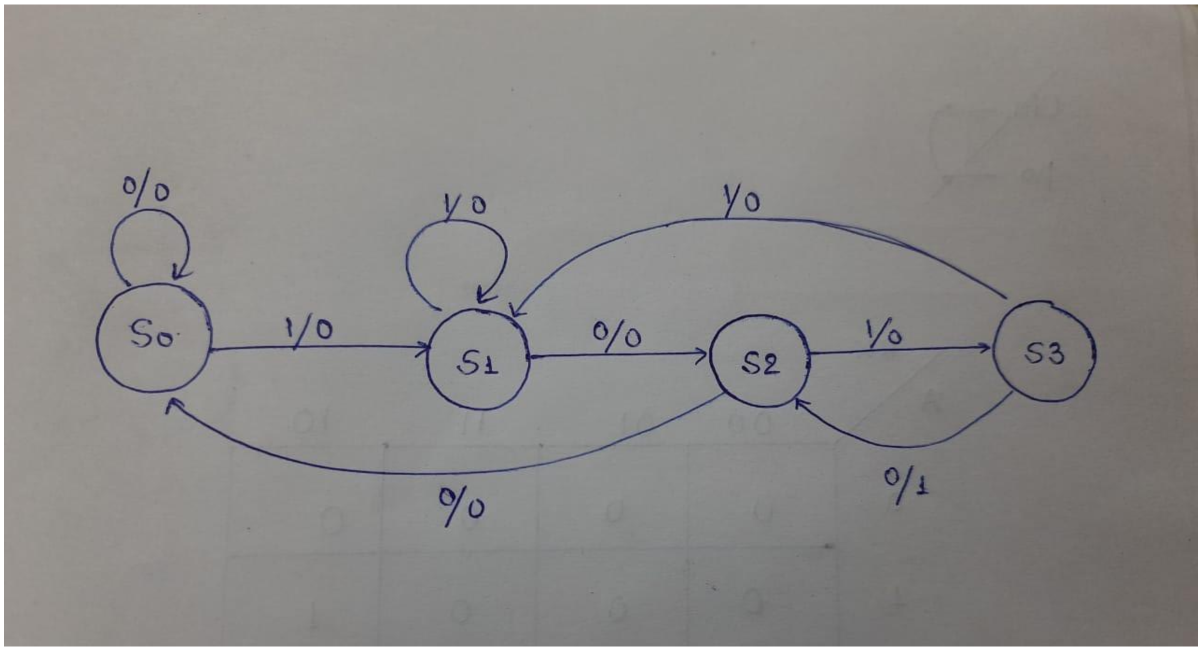
FSM construction of Sequence Detector :

- The FSM is implemented using three always blocks, one block for changing the state on getting input (combinational logic), one for changing the current state whenever the clock (clk) has positive edge (sequential logic) and one for giving output (output logic).
- Four states are defined, IDLE, state1, state2, and state3.

```
parameter IDLE = 2'b00,//initially
state1 = 2'b01, //state when it gets 10 from idle state
state2= 2'b10, //state when it gets 101
state3= 2'b11; //state when it gets 1010
```

- Initially, the clock is set 0 and reset is also set 0. Then 8-bit input is given which sends one-bit to the module seq_detector each time the edge becomes positive in the clock.
- Initially, the state is defined as IDLE and depending upon the input the next state is determined.
- As the state changes, an one-bit output is given. If sequence 1010 gets detected, output is given as 1 else 0.
- Again, when posedge arrives, the next 1-bit input is send, the state changes and the sequence is determined.

❖ State Diagram :



Here S0 is IDLE as stated above.

- ❖ Output Table :

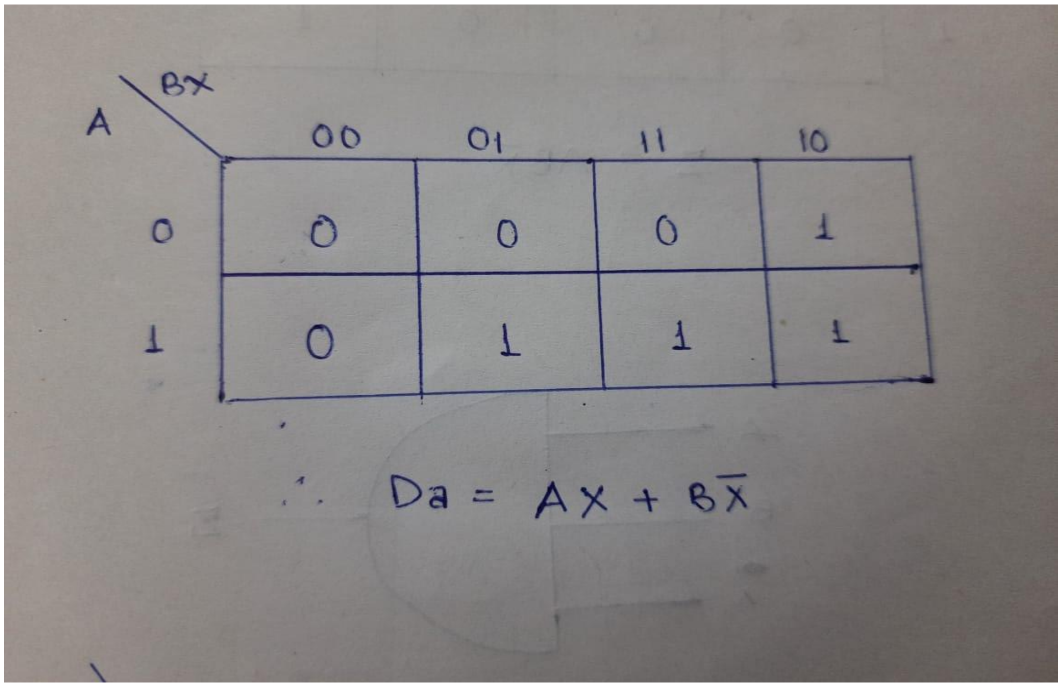
Present State	Next State		Output	
	X=0	X=1	X=0	X=1
00	00	01	0	0
01	10	01	0	0
10	00	11	0	0
11	10	11	1	0

- ❖ Excitation Table :

Present State		Input	Next State		Flip-flop excitations		Output
A	B	X	A’	B’	Da	Db	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0

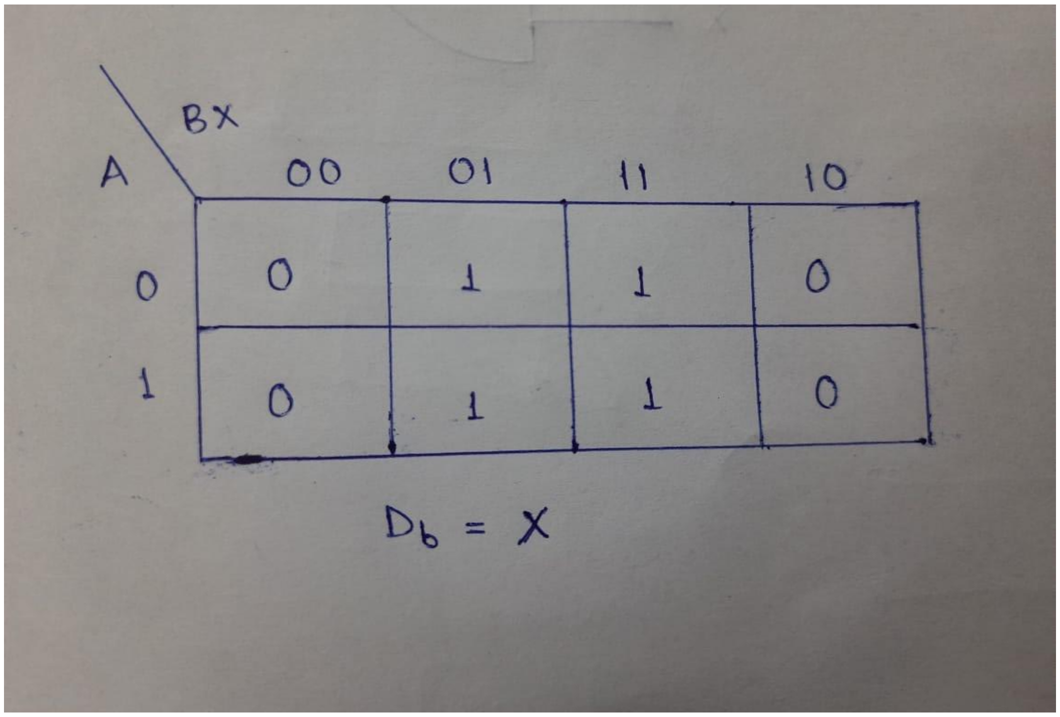
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1
1	1	1	1	1	1	1	0

- K-map



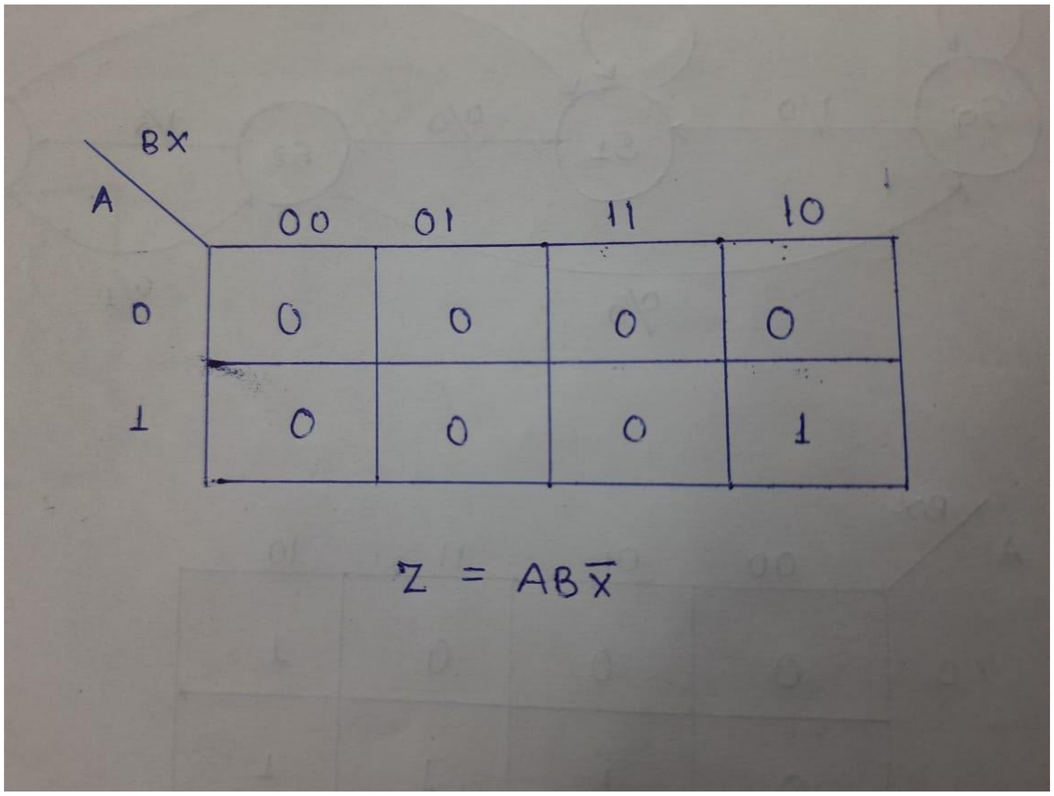
$$\begin{array}{c|cccc} A \backslash BX & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{array}$$

$\therefore \quad Da = AX + B\overline{X}$



A \ BX	00	01	11	10
0	0	1	1	0
1	0	1	1	0

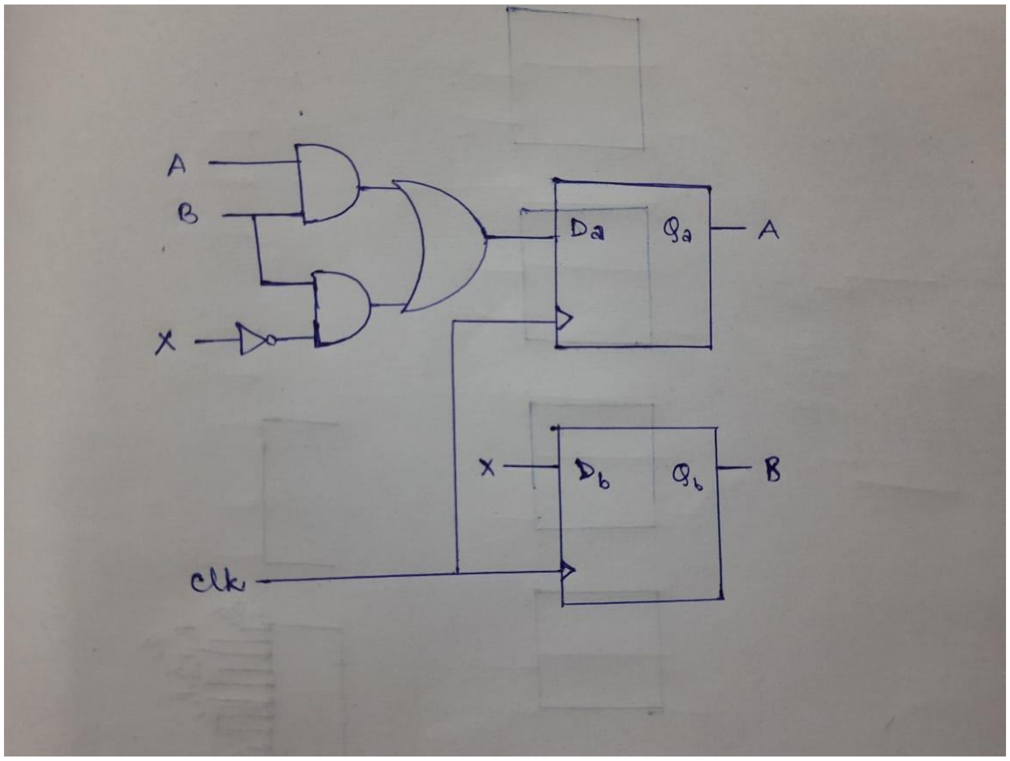
$D_b = X$

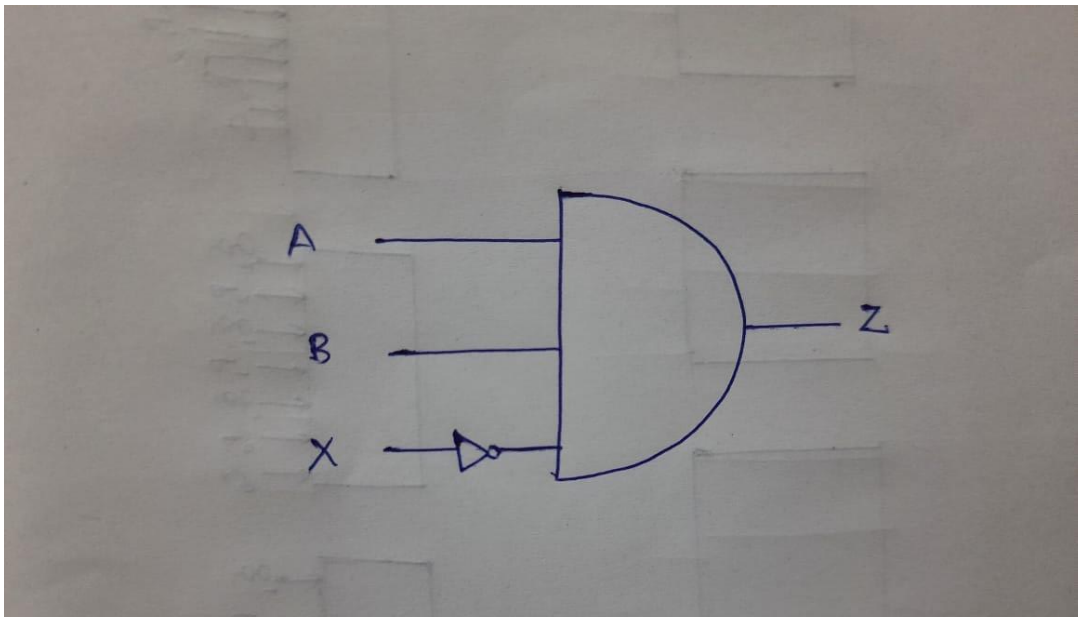


A \ BX	00	01	11	10
0	0	0	0	0
1	0	0	0	1

$Z = AB\overline{X}$

❖ Circuit / Logic Diagram





❖ Verilog Code

```
module seq_detector(clk,reset,in,out);


input clk,reset,in;
output out;
```

```
wire clk,reset;
wire in; //1-bit input
reg out; //1-bit output
parameter SIZE = 2; //setting parameter SIZE as 2

//setting parameters as different states
parameter IDLE = 2'b00,//initially
state1 = 2'b01, //state when it gets 10 from idle state
state2= 2'b10, //state when it gets 101
state3= 2'b11; //state when it gets 1010


reg [SIZE-1:0] state; //current state
reg [SIZE-1:0] next_state; //next state
initial begin
  state = IDLE;
end

always @(state or in) begin
    next_state= IDLE;
    case (state)

        //when it gets input after it is at IDLE state
        IDLE: if (~in) begin

            next_state= IDLE;
        end
        else begin
          next_state = state1;
        end

        //when it gets input after it is at state1
        state1: if(~in) begin

          next_state = state2;
        end
        else begin
          next_state= state1;
        end

        //when it gets input after it is at state2
        state2: if(~in) begin

          next_state = IDLE;
        end
        else begin
          next_state = state3;
        end

        //when it gets input after it is at state3
        state3: if(~in) begin
          next_state= state2;
        end
        else begin
          next_state = state1;
        end

        default:  begin
        next_state = IDLE;
        end


    endcase

end

always @(posedge clk) begin

    if(reset == 1'b1) begin

      state = IDLE; //new (8-bit) input
    end
    else begin

      state = next_state; //continuation of the input
    end

end
```

```
always @(state or in) begin

    if(reset == 1'b1) begin
      out = 1'b0;
    end

    else begin
      if(state == state3) begin
        if(in == 1'b0) begin
          out = 1'b1; //output when it finds the sequence of
1010
        end
        else begin
          out = 1'b0; ////output when it is at any other state
        end
      end
      else begin
        out = 1'b0;
      end
    end

end




endmodule
```

★ Testbench :

```
`include "A3_Q1_seq_detector.v"

module seq_tb();

reg clk, reset, inp;
reg [7:0] in; //reg to store 7-bit input
reg [7:0]out; //reg to store 7-bit output
wire outp; //to store 1-bit after each 1-bit input
```

```
integer i;


always #2 clk=~clk; //oscillation in clock

seq_detector x(clk,reset,inp,outp);

initial begin
  clk= 1'b0; //setting clock to 0 initially
  reset = 1'b0; //setting reset to 0 initially and after each input
case changing to 1 to get again IDLE state
  in = 8'b10101010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01110110; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01010010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b10100010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01011010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01010110; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b00010010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b00010110; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b11001010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01110011; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01110100; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b11010010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01010110; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b00010010; #35 $display("input = %b output= %b",in,out);
  reset = 1'b1;
  #5 reset =1'b0;
  in = 8'b01010000; #35 $display("input = %b output= %b",in,out);

  #5 $finish;


end


//giving input bit-wise whenever the 7-bit input changes
always @(in) begin
  for(i=7;i>=0;i-=1) begin
      #2
      inp= in[i];
      #2
      out[i]= outp;
    end

end

endmodule
```

3-bit odd parity bit generator:

A 3-bit odd parity bit generator takes 3 bits input one by one and generates an output 1 if the number of 1s are even or 0 otherwise. It can be implemented by Finite State Machine (FSM) which keeps count as states. It is a type of Moore Machine which gives output on the basis of Current State .

FSM Construction of 3-bit odd parity generator :

- The FSM is implemented using three always blocks, one block for changing the state on getting input (combinational logic), one for changing the current state whenever the clock (clk) has positive edge (sequential logic) and one for giving output (output logic).
- 11 states are defined, IDLE, odd0_even0, odd0_even1,odd1_even0,odd0_even2,odd1_even1,odd2_even0,odd2_even1,odd1_even2,odd3_even0,odd3_even0

- We are defining the states on the basis of number of 0 and 1 we encounter, for eg odd2_even1 means 2 bits are 1(odd) and 1 bit is 0(even)
- .

States :

```
IDLE = 4'b0000,
odd0_even0 = 4'b1000,
odd0_even1 = 4'b0010,
odd1_even0 = 4'b0001,
odd0_even2 = 4'b0110,
odd1_even1 = 4'b0011,
odd2_even0 = 4'b0101,
odd0_even3 = 4'b1110,
odd1_even2 = 4'b0111,
odd2_even1 = 4'b1011,
odd3_even0 = 4'b1101;
```

- Initially, the clock is set 0 and reset is also set 0. Then 3-bit input is given which sends one-bit to the module parity_gen each time the edge becomes positive in the clock.
- Initially, the state is defined as IDLE and depending upon the input the next state is determined.
- The input changes each time the edge becomes positive in the clock and at the end of the 3-bit input, output is given as 1 in case the number of 1's in the sequence is even, 0 in other case.

Truth Table:-

Inputs (3-bits)			Output
A	B	C	O
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

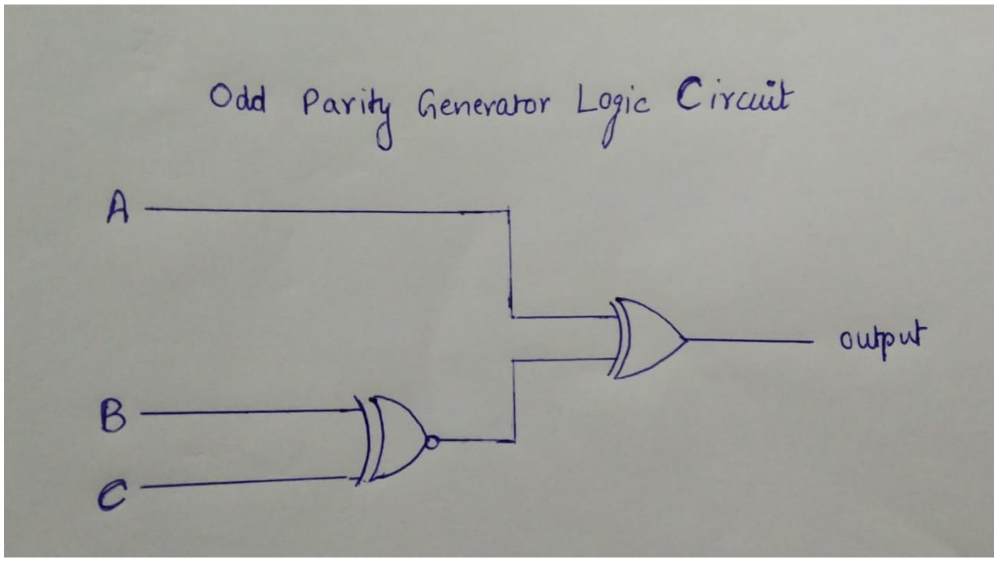
1	1	1	0

K-Map

		BC			
		00	01	11	10
A	0	1	0	1	0
	1	0	1	0	1

Output = A'B'C' + AB'C + A'BC + ABC'
= A ⊕ (B ⊕ C)'

Logic Ciruit:



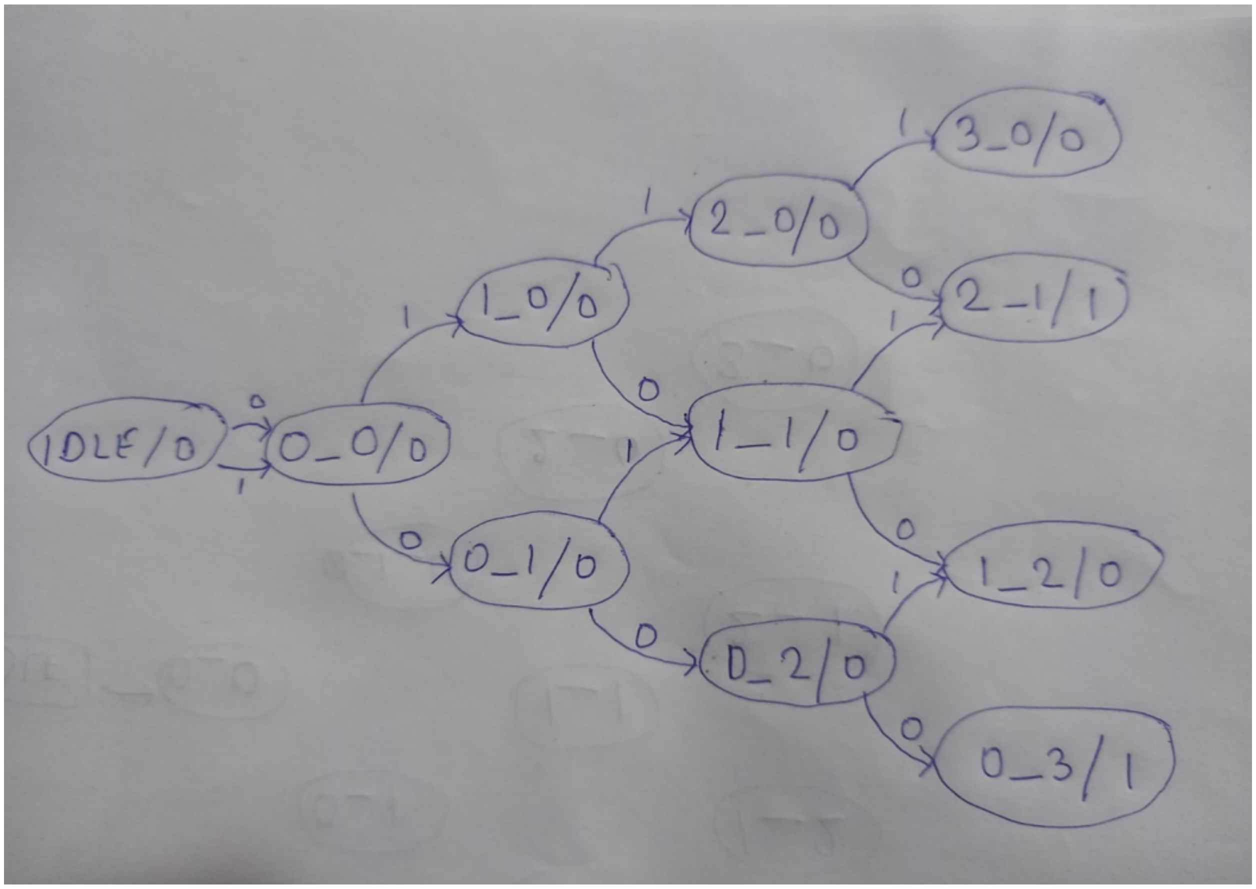
Excitation Table:

For any input with new (8-bit)input the state changes to IDLE_ state

Previous State	Next State	Input	Output
0000	1000	0/1	0
1000	0001	1	0
1000	0010	0	0

0010	0110	0	0
0010	0011	1	0
0001	0011	0	0
0001	0101	1	0
0110	1110	0	0
0110	0111	1	0
0011	0111	0	0
0011	1011	1	0
0101	1011	0	0
0101	1101	1	0
1110	0000	0/1	1
0111	0000	0/1	0
1011	0000	0/1	1
1101	0000	0/1	0

State diagram:



Moore state diagram of 3 bit parity generator

Verilog code:

```
module parity_gen(clk,reset,in,out);

input clk,reset,in;
output out;

wire clk,reset;
wire in; //1-bit input
reg out; //1-bit output
parameter SIZE = 4; //setting parameter SIZE as 4

//setting parameters as different states
parameter IDLE = 4'b0000, //initial state
odd0_even0 = 4'b1000, //state when there are zero 1s and zero 0s
odd0_even1 = 4'b0010, //state when there are zero 1s and one 0s
odd1_even0 = 4'b0001, //state when there are one 1s and zero 0s
odd0_even2 = 4'b0110, //state when there are zero 1s and two 0s
```

```verilog
odd1_even1 = 4'b0011, //state when there are one 1s and one 0s
odd2_even0 = 4'b0101, //state when there are two 1s and zero 0s
odd0_even3 = 4'b1110, //state when there are zero 1s and three 0s
odd1_even2 = 4'b0111, //state when there are one 1s and two 0s
odd2_even1 = 4'b1011, //state when there are two 1s and one 0s
odd3_even0 = 4'b1101; //state when there are three 1s and zero 0s


reg [SIZE-1:0] state; //current state
reg [SIZE-1:0] next_state; //next state
initial begin
 state = IDLE;
end

always @(state or in) begin

   next_state= IDLE;
   case (state)
       IDLE: next_state= odd0_even0;

       //when it gets input after it is at odd0_even0 state
       odd0_even0 :if (in) begin
           next_state= odd1_even0;
       end
       else begin
         next_state= odd0_even1;
       end

       //when it gets input after it is at odd0_even1 state
       odd0_even1 : if (in) begin
           next_state= odd1_even1;
       end
       else begin
         next_state= odd0_even2;
       end

       //when it gets input after it is at odd1_even0 state
       odd1_even0: if(in) begin
         next_state= odd2_even0;
       end
```

```
    else begin
      next_state= odd1_even1;
    end

    //when it gets input after it is at odd0_even2 state
    odd0_even2: if(in) begin
      next_state= odd1_even2;
    end
    else begin
      next_state= odd0_even3;
    end

    //when it gets input after it is at odd1_even1 state
    odd1_even1: if(in) begin
      next_state= odd2_even1;
    end
    else begin
      next_state= odd1_even2;
    end

    //when it gets input after it is at odd2_even0 state
    odd2_even0: if(in) begin
      next_state= odd3_even0;
    end
    else begin
      next_state= odd2_even1;
    end

    //when it gets input after it is at odd0_even3 state
    odd0_even3: if(in) begin
      next_state= IDLE;
    end
    else begin
      next_state= IDLE;
    end

    //when it gets input after it is at odd1_even2 state
    odd1_even2: if(in) begin
      next_state= IDLE;
    end
```

```
       else begin
         next_state= IDLE;
       end

       //when it gets input after it is at odd2_even1 state
       odd2_even1: if(in) begin
         next_state= IDLE;
       end
       else begin
         next_state= IDLE;
       end

       //when it gets input after it is at odd3_even0 state
       odd3_even0: if(in) begin
         next_state= IDLE;
       end
       else begin
         next_state= IDLE;
       end
       default: next_state = IDLE;
   endcase

end

always @(posedge clk ) begin
   if(reset) begin
     state = #1 IDLE; //new (3-bit) input
   end
   else begin
     state = #1 next_state; //continuation of the input
   end
end

always @(in or state ) begin
   //output when it is at state with odd number of ones
   if(state == odd2_even1) begin
     out = #1 1'b1;
   end
   else if(state == odd0_even3) begin
     out = #1 1'b1;
   end
   else  begin
     out = #1 1'b0; //otherwise condition
   end

end



endmodule

```

Testbench:

```
`include "A3_Q2_parity_gen.v"

module tstbnch();

reg clk,reset,inp,out;
reg [2:0] in; //input register
wire outp;  //output wire
integer i;

always #1 clk = ~clk; //for clock oscillation

parity_gen x(clk,reset,inp,outp);

initial begin
   clk = 1'b0;  //setting clock to 0 initially
   reset = 1'b0; //setting reset to 0 initially and after each input case
changing to 1 to get again IDLE state
   in = 3'b000;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b001;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b010;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b011;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b100;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b101;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b110;
   #10
   $display("input=%b parity_bit=%b",in,outp);
   reset=1'b1;
   #5 reset=1'b0;
   in = 3'b111;
   #10
   $display("input=%b parity_bit=%b",in,outp);







   #3$finish;


end
```

```
//giving input bit-wise whenever the 3-bit input changes

always @(in) begin

   for(i=0;i<3;i+=1) begin
       #2
       inp= in[i];

   end







end

always @(outp) begin
   // $display("outpchange %d",$time);

end

endmodule
```