**Parity Detector**

**Lab no# 10**

****

Spring 2022

CSE-308L Digital Systems Design lab

Submitted by: **Ashfaq Ahmad**

Registration No: **19PWCSE1795**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Dr: Ma’am Madeha sheer**

**July** 24, 2022

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Objective:**

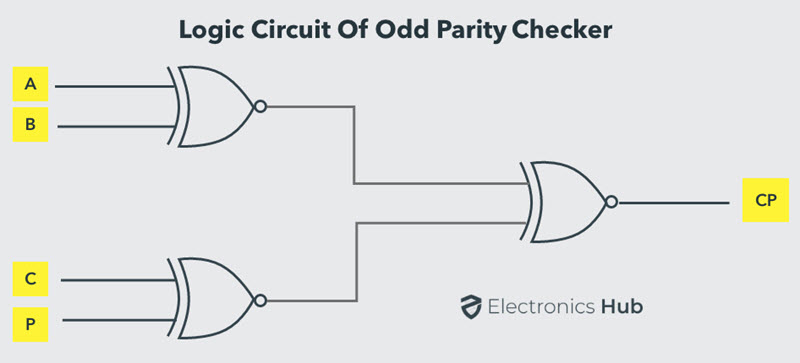
* Built a parity detector.

**Parity Detector:**

It is a logic circuit that checks for possible errors in the transmission. This circuit can be an even parity checker or odd parity checker depending on the type of parity generated at the transmission end.

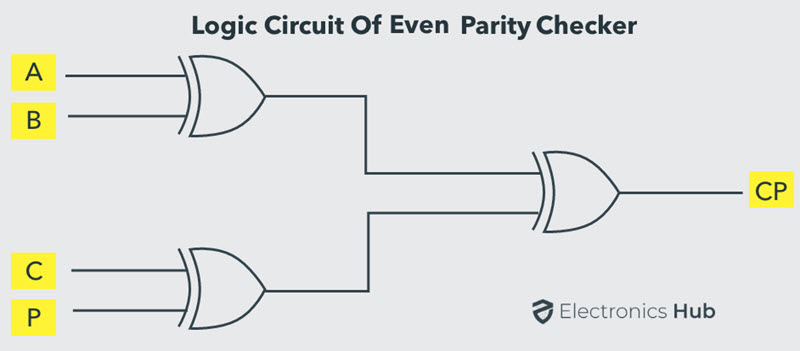
**Odd Parity detector:**

If the total number of 1s in the data is odd, then it indicates no error, whereas if the total number of 1s is even then it indicates the error since the data is transmitted with odd parity at transmitting end.



**Even Parity detector:**

If the total number of 1s in the data is even, then it indicates no error, whereas if the total number of 1s is odd then it indicates the error since the data is transmitted with even parity at transmitting end.



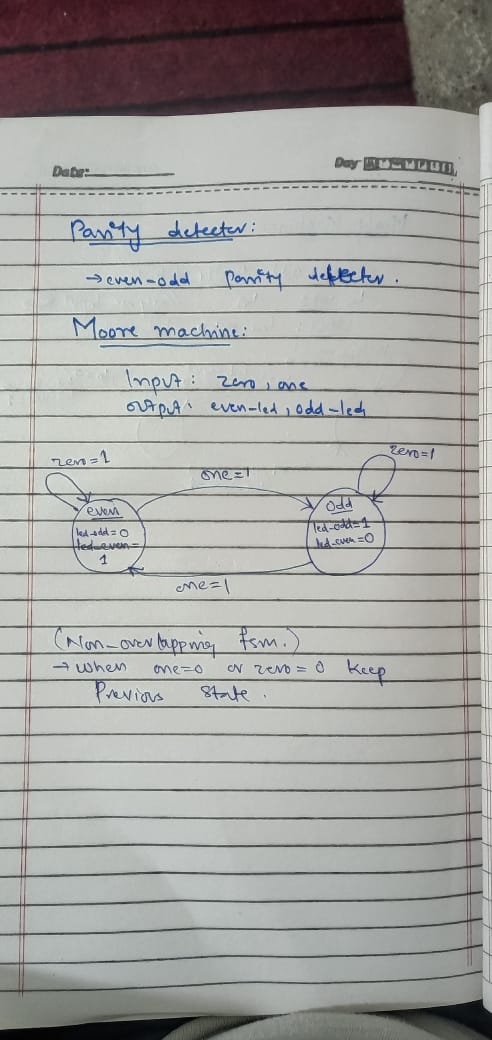
**Task01: Design a Parity Detector.**

**Note:** in this task we designed a code for both even and odd parity. In even case even led on and

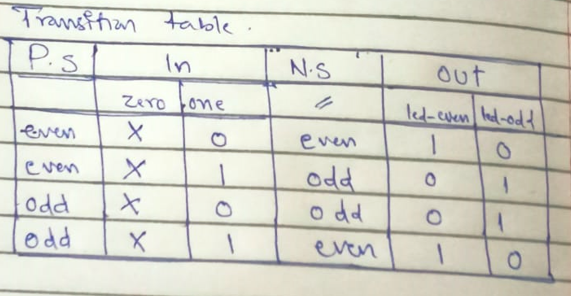
In odd case odd led on. We can also design individually even or odd parity detector.

**State Transition Diagram:**

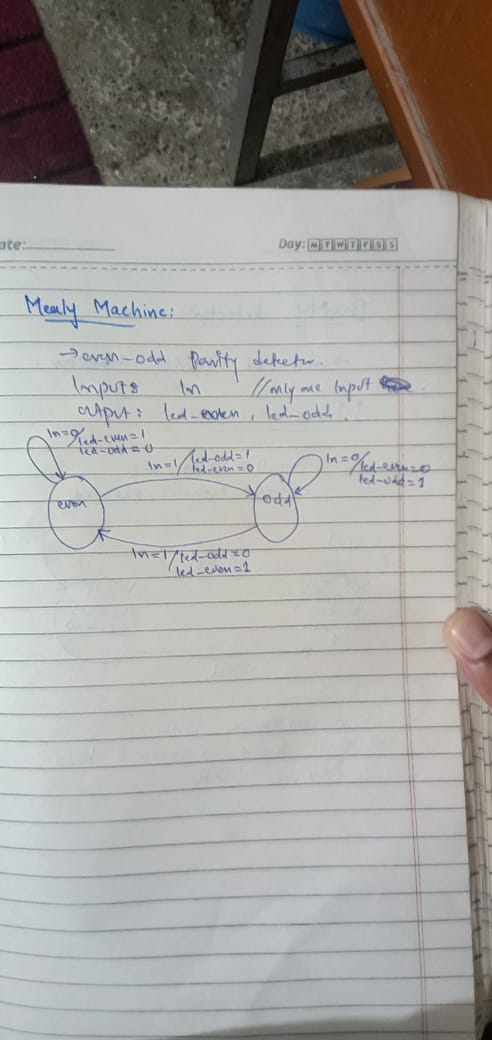
**Moore Machine with two input, zero for 0 and one for 1:**

****

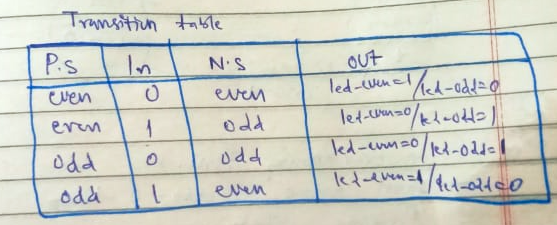
**Transition table**

****

**Mealy Machine with one input set for 1 and reset for 0:**



**Transition table:**

****

**Source Code: (Two always block and two inputs button (one for 0 and other for 1))**

**Moore Machine:**

module parity\_detector(led\_even,led\_odd,zero,one,clk\_100Mhz,reset);

input zero,one,clk\_100Mhz,reset;  //two inputs button

output reg led\_even,led\_odd;

wire clk\_1hz,sync\_zero,sync\_one,l2p\_zero,l2p\_one;

parameter even=1'b0,odd=1'b1;

reg state,next\_state;

clk\_divider cd(clk\_1hz,clk\_100Mhz,reset);  //all module take clk\_ihz.

synchronizer syn1(sync\_zero,zero,clk\_1hz,reset);

synchronizer syn2(sync\_one,one,clk\_1hz,reset);

l2p l2p1(l2p\_zero,sync\_zero,clk\_1hz,reset);

l2p l2p2(l2p\_one,sync\_one,clk\_1hz,reset);

always @(posedge clk\_1hz,posedge reset)

        if(reset)

                  state=even;

           else

                   state=next\_state;

always @(\*)          //moore machine.

        case(state)

          even:

          begin

               led\_even=1;

                 led\_odd=0;

                 if(l2p\_zero) //if we have two inputs then we can't use ternary operator like (next\_state=in?even:odd)

                        next\_state=even;

                 else if(l2p\_one)

                        next\_state=odd;

                 else

                        next\_state = state;

          end

          odd:

          begin

               led\_odd=1;

                 led\_even=0;

                 if(l2p\_zero)

                    next\_state=odd;

                 else if(l2p\_one)

                    next\_state=even;

                 else

                    next\_state = state;

          end

        default:  //if initially not reset then default statement will execute.

          begin

             led\_odd=0;

             led\_even=0;

             next\_state=even;

        end

        endcase

endmodule

module clk\_divider(clk\_1hz,clk\_100Mhz,reset);

input clk\_100Mhz,reset;

output reg clk\_1hz;

integer c;

always @(posedge clk\_100Mhz)

       if(reset)

         begin

                 clk\_1hz=1;

                    c=0;

         end

         else

         begin

             c=c+1'b1;

              if(c==10000000)

              begin

              clk\_1hz=~clk\_1hz;

              c=0;

              end

         end

endmodule

module synchronizer(out,in,clk\_1hz,reset);  //we can also design synchronizer using D\_FFs.

input in,clk\_1hz,reset;

output reg out;

reg FF1\_out;

always @(posedge clk\_1hz)

        if(reset)

                  out=0;

          else

          begin

                  FF1\_out=in;

                     out=FF1\_out;

          end

endmodule

module l2p(l2p\_out,sync\_in,clk\_1hz,reset);

input sync\_in,clk\_1hz,reset;

output reg l2p\_out;

parameter s0=1'b0,s1=1'b1;

reg state,next\_state;

always @(posedge clk\_1hz)

        if(reset)

          state=s0;

          else

          state=next\_state;

always @(\*)          //Mealy machine.

        case(state)

          s0:

          begin

             l2p\_out=sync\_in?1'b1:1'b0;   //as there is only one input so we can use ternary operator.

              next\_state=sync\_in?s1:s0;

          end

          s1:

          begin

             l2p\_out=sync\_in?0:0;

              next\_state=sync\_in?s1:s0;

          end

       default:

         begin

              l2p\_out=0;

                next\_state=s0;

         end

       endcase

endmodule

**Output: (Moore output)**

* The input Pins are Dip switch no 1 for 0 input and Dip switch no 2 for 1 input and reset switch is dip switch no 8.
* States are Even and odd.
* Led no 1 is even output and led no 2 is odd output.

**State:** even;

**Output:** even\_led=1, odd\_led=0;

**Input:**  zero=0, one=1;



**State:** odd;

**Output:** even\_led=0, odd\_led=1;

**Input:**  zero=0, one=1;



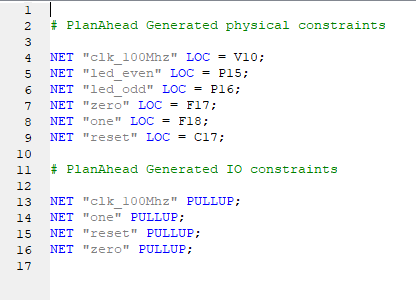
**Reset case:**

**Output:** even led=1;

**State:**  even;



**UCF file:**

****

**Mealy Code: (one always block and one inputs button (set for 1 and reset for 0))**

module parity\_detector(led\_even,led\_odd,in,clk\_100Mhz,reset);  //one input

input in,clk\_100Mhz,reset;

output reg led\_even,led\_odd;

wire clk\_1hz,sync\_out,l2p\_out;

parameter even=1'b0,odd=1'b1;

reg state;

clk\_divider cd(clk\_1hz,clk\_100Mhz,reset);

synchronizer sync(in,sync\_out,clk\_1hz,reset);

l2p lp(l2p\_out,sync\_out,clk\_1hz,reset);

always @(posedge clk\_1hz,posedge reset)  //one always block .....Mealy Machine

        if(reset)

          begin

                  state=even;

                     led\_even=1'b1;

                     led\_odd=1'b0;

          end

          else

        case(state)

          even:

          begin

               led\_even=l2p\_out?1'b0:1'b1;   //as one input so ternary operator.

                 led\_odd=l2p\_out?1'b1:1'b0;

                 state=l2p\_out?odd:even;

          end

          odd:

          begin

               led\_even=l2p\_out?1'b1:1'b0;

                 led\_odd=l2p\_out?1'b0:1'b1;

                 state=l2p\_out?even:odd;

          end

          default:

          begin

               state=even;

                 led\_even=1'b1;

                 led\_odd=1'b0;

          end

        endcase

endmodule

module clk\_divider(clk\_1hz,clk\_100Mhz,reset);  //clock divider

input clk\_100Mhz,reset;

output reg clk\_1hz;

integer c;

always @(posedge clk\_100Mhz)

       if(reset)

         begin

                 clk\_1hz=1;

                    c=0;

         end

         else

         begin

             c=c+1'b1;

              if(c==10000000)

              begin

              clk\_1hz=~clk\_1hz;

              c=0;

              end

         end

endmodule

module D\_FF(in,out,clk\_1hz,reset);   //D\_FF

input in,clk\_1hz,reset;

output reg out;

always @(posedge clk\_1hz)

        if(reset)

                out=0;

        else

            out=in;

endmodule

module synchronizer(in,sync\_out,clk\_1hz,reset);   //synchronizer

input in,clk\_1hz,reset;

output sync\_out;

wire FF1\_out;

D\_FF dff1(in,FF1\_out,clk\_1hz,reset);

D\_FF dff2(FF1\_out,sync\_out,clk\_1hz,reset);

endmodule

module l2p(l2p\_out,sync\_in,clk\_1hz,reset);    //l2p

input sync\_in,clk\_1hz,reset;

output reg l2p\_out;

parameter s0=1'b0,s1=1'b1;

reg state,next\_state;

always @(posedge clk\_1hz)

        if(reset)

          state=s0;

          else

          state=next\_state;

always @(\*)          //Mealy machine.

        case(state)

          s0:

          begin

             l2p\_out=sync\_in?1'b1:1'b0;   //as there is only one input so we can use ternary operator.

              next\_state=sync\_in?s1:s0;

          end

          s1:

          begin

             l2p\_out=sync\_in?0:0;

              next\_state=sync\_in?s1:s0;

          end

       default:

         begin

                l2p\_out=0;

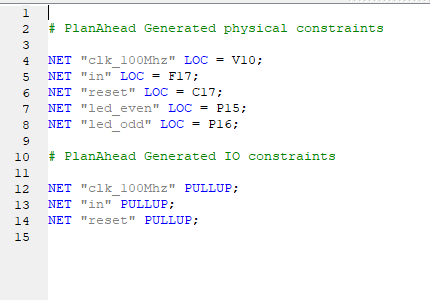
                next\_state=s0;

         end

       endcase

endmodule

**UCF File:**



**Output: (Mealy output)**

* The Dip switch no 1 is input switch and reset switch is dip switch no 8.
* States are even and odd

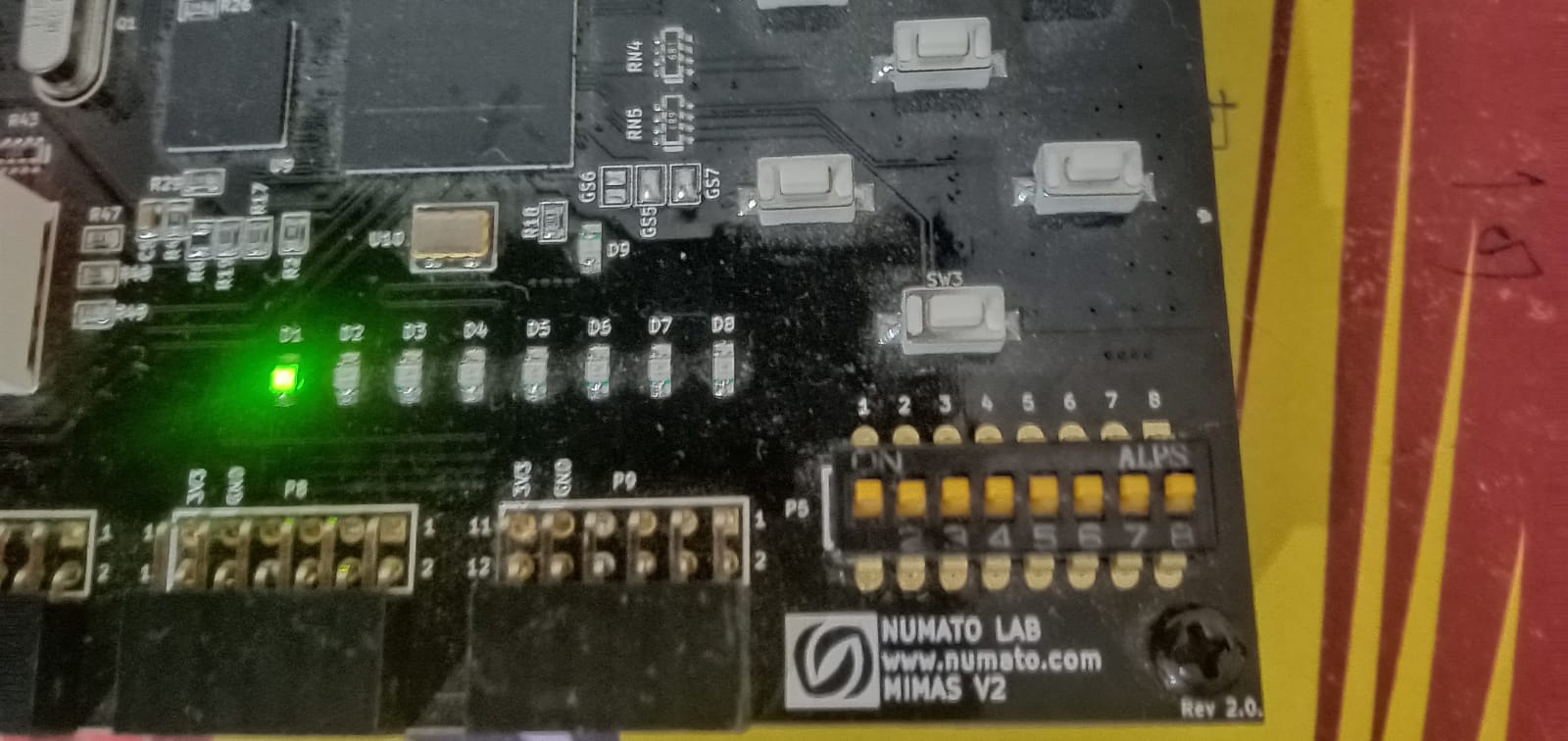
Initially

**State:** even

**Reset=**0

**Input:** in=0;

**Output:** led even=1, led odd=0;

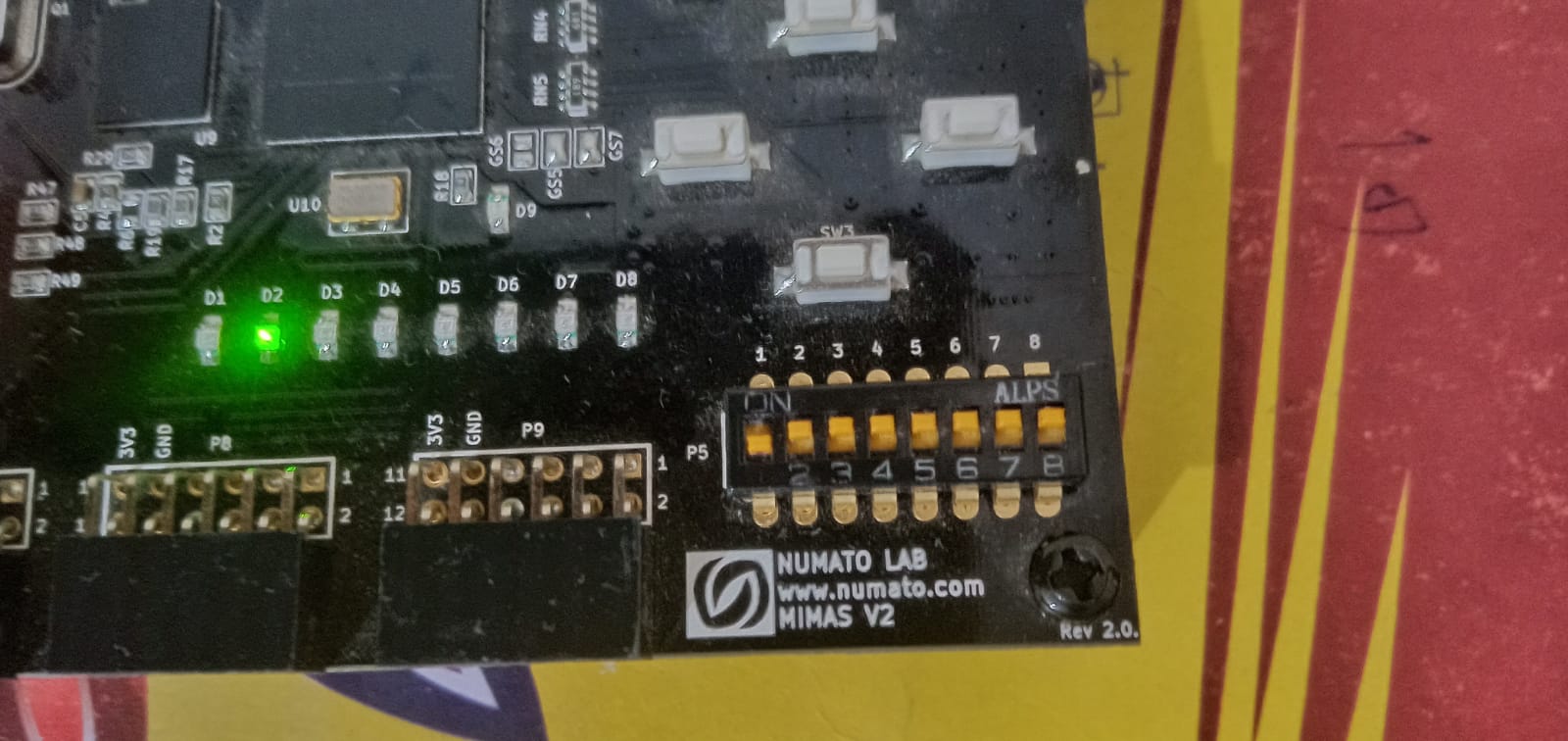


**State:** odd

**Reset=**0

**Input:** in=1;

**Output:** led even=0, led odd=1;

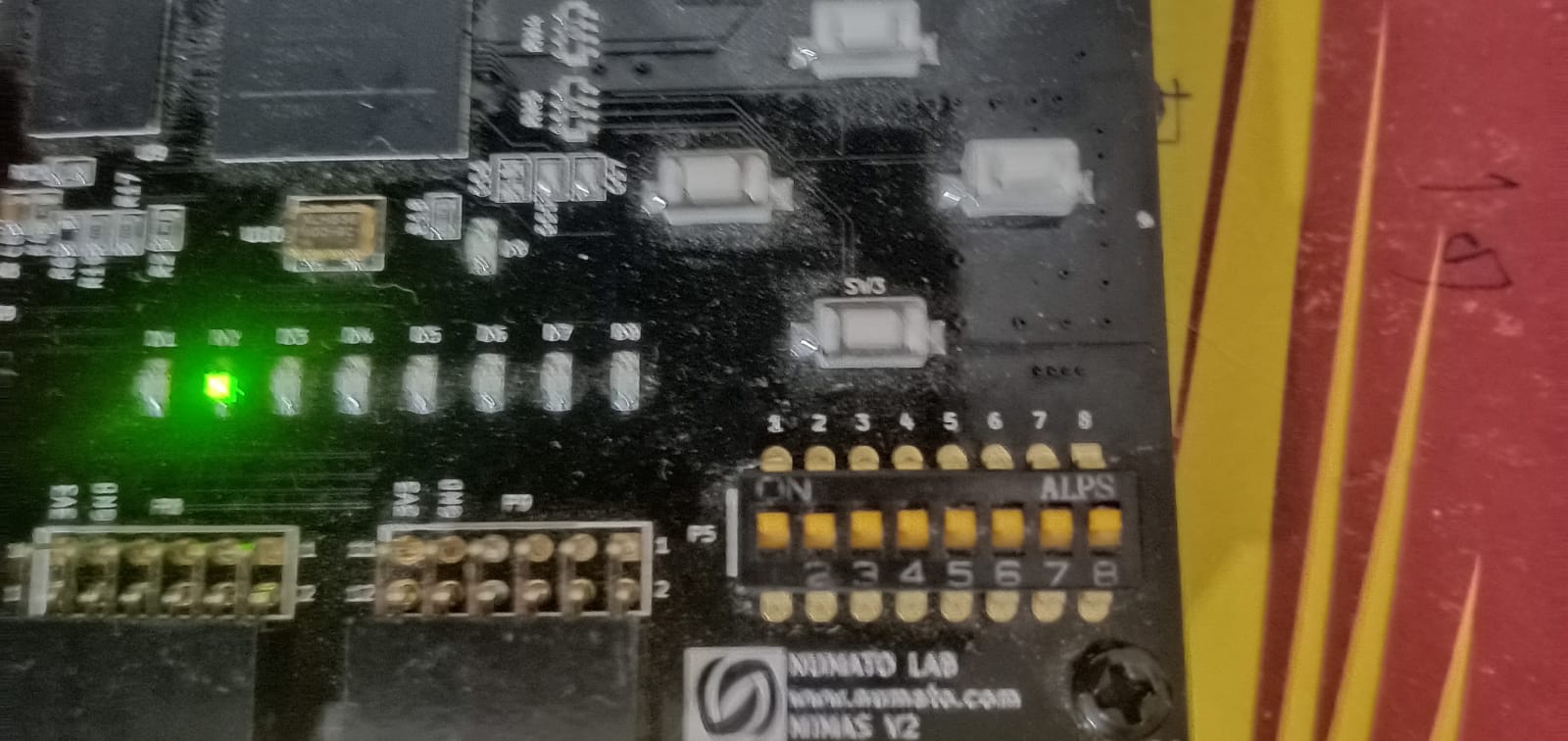


**State:** odd

**Reset=**0

**Input:** in=0;

**Output:** led even=0, led odd=1;

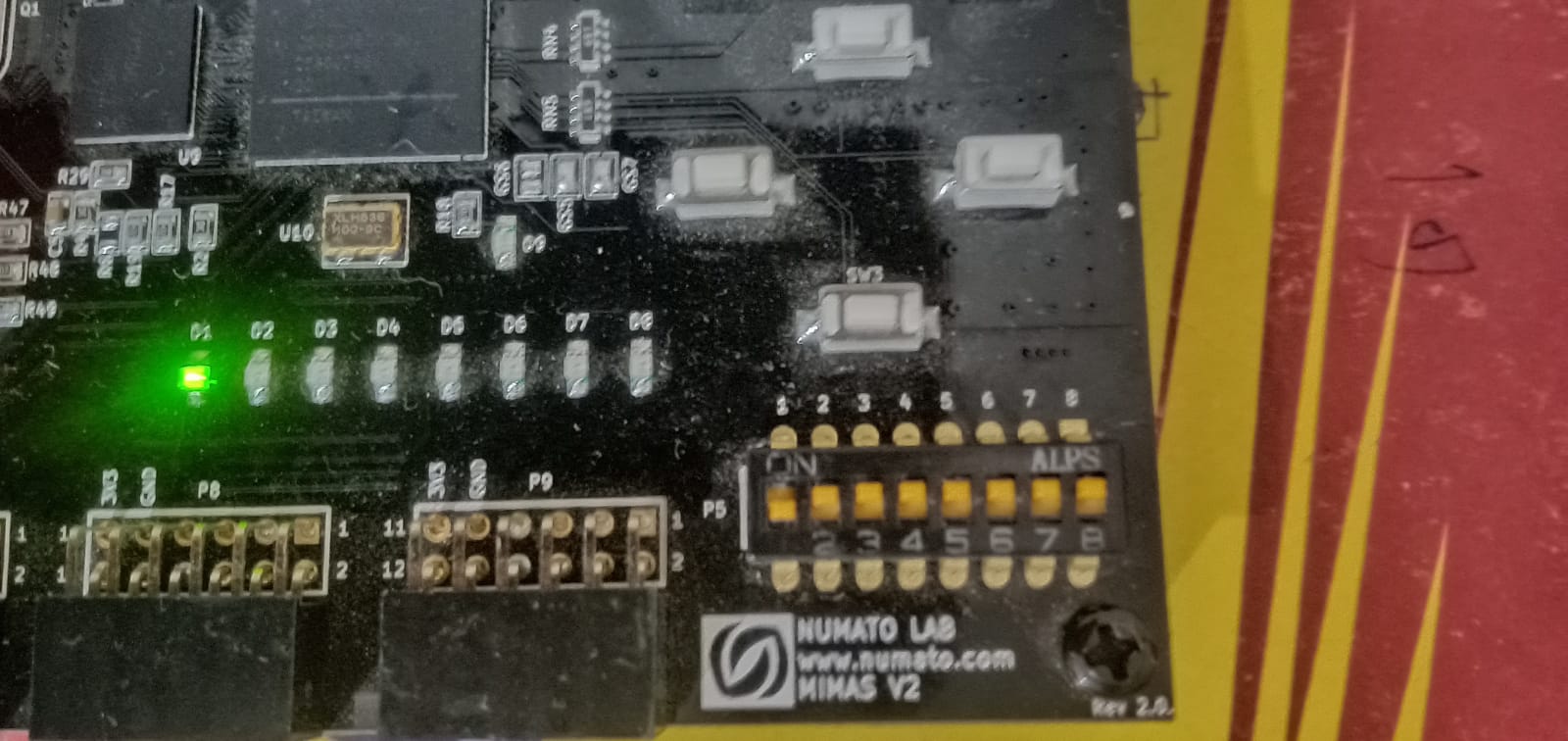


**State:** even

**Reset=**0

**Input:** in=1;

**Output:** led even=1, led odd=0;

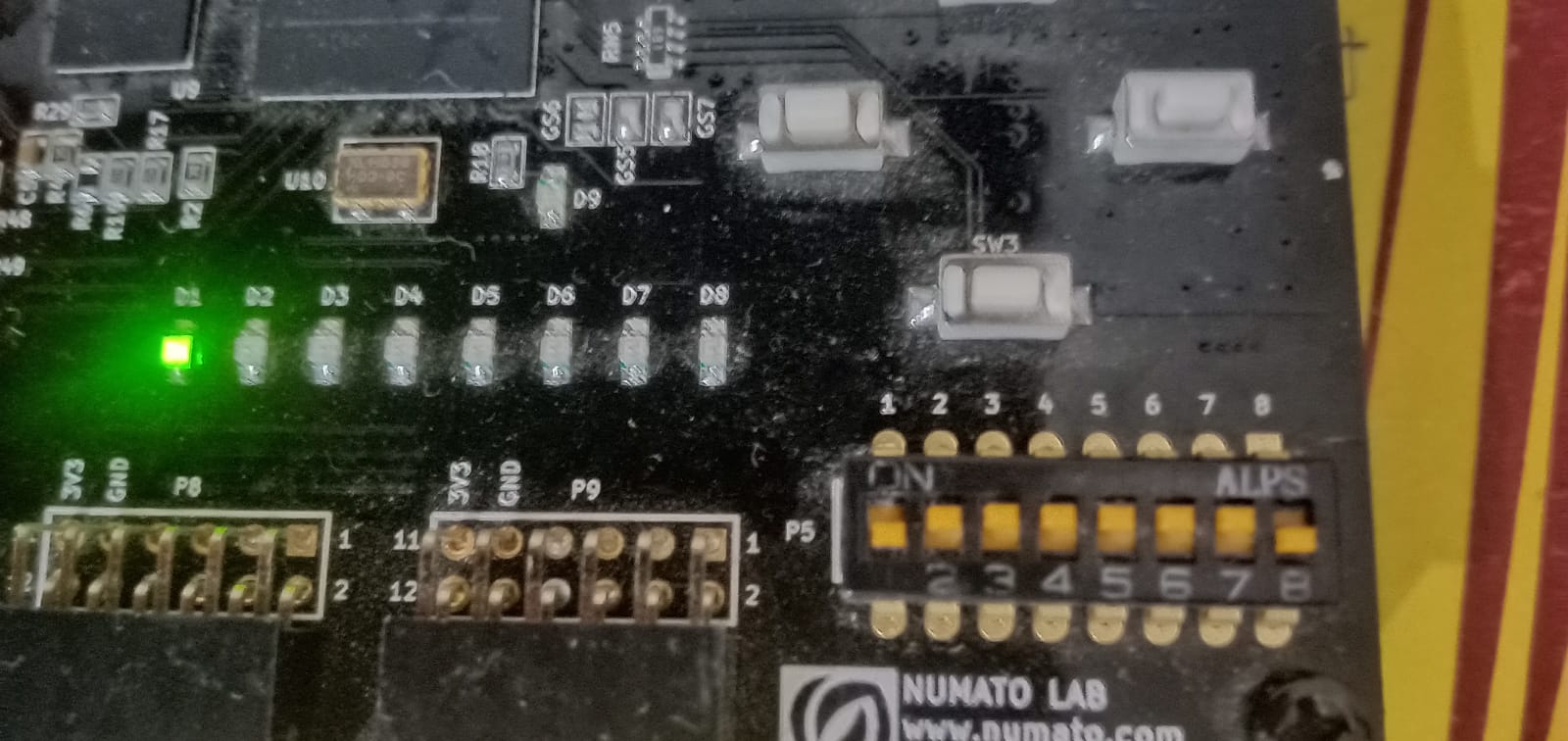


**Reset case:**

**Reset=1;**

**Output:** even led=1;

**State:**  even;



**Above Code without synchronizer and L2P converter.**

If we don’t use synchronizer and l2p converter and keep input as high then output (even led odd led) will toggle continuously. Because at each clock cycle the no of 1’s increment. In even case even led on and in odd case odd led on.

**Code:**

module parity\_detector(led\_even,led\_odd,in,clk\_100Mhz,reset);

input in,clk\_100Mhz,reset;

output reg led\_even,led\_odd;

wire clk\_1hz;

parameter even=1'b0,odd=1'b1;

reg state,next\_state;

clk\_divider cd(clk\_1hz,clk\_100Mhz,reset);

always @(posedge clk\_1hz,posedge reset)

        if(reset)

                  state=even;

           else

                   state=next\_state;

always @(\*)

        case(state)

          even:

          begin

               led\_even=1;

                 led\_odd=0;

                 next\_state=in?odd:even;

          end

          odd:

          begin

               led\_odd=1;

                 led\_even=0;

                 next\_state=in?even:odd;

          end

        endcase

endmodule

module clk\_divider(clk\_1hz,clk\_100Mhz,reset);

input clk\_100Mhz,reset;

output reg clk\_1hz;

integer c;

always @(posedge clk\_100Mhz)

       if(reset)

         begin

                 clk\_1hz=1;

                    c=0;

         end

         else

         begin

             c=c+1'b1;

              if(c==10000000)

              begin

              clk\_1hz=~clk\_1hz;

              c=0;

              end

         end

endmodule