Name	Muhammad Asad
Reg. #	2019-EE-383
Marks	

Experiment # 13

Open Ended Lab

Design a Sequential Circuit to Check Parity for Error Detection in a Communication System

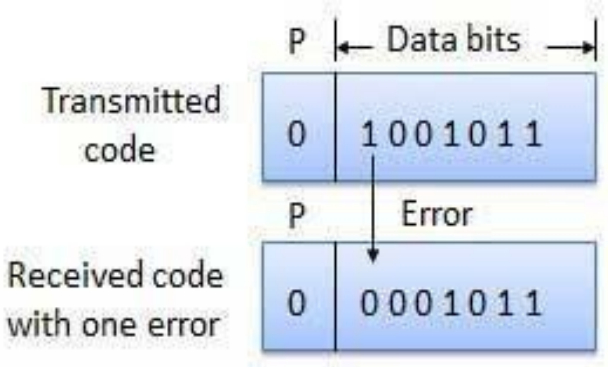
Objective:

- *Designing of a digital system based on background knowledge*

Problem Statement:

In information theory and coding theory with applications in computer science and telecommunication, **error detection** and **correction or error control** are techniques that enable reliable delivery of digital data over unreliable communication channels. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors to avoid false reception of information.

One of the error detection techniques is parity checker. This scheme is used to detect 1-bit error.



As a design engineer, you are required to design a sequential circuit that can check the **odd parity** of input bit stream received at receiver. Total number of 1's should be odd for error-free reception, in this case circuit should generate output **0**. If total number of 1's is even, it will generate output **1**. Your circuit will have one input and one output as shown in the state table below:

Input	Present state (PS)	Next state (NS)/Output
0	0	0
0	1	1
1	0	1
1	1	0

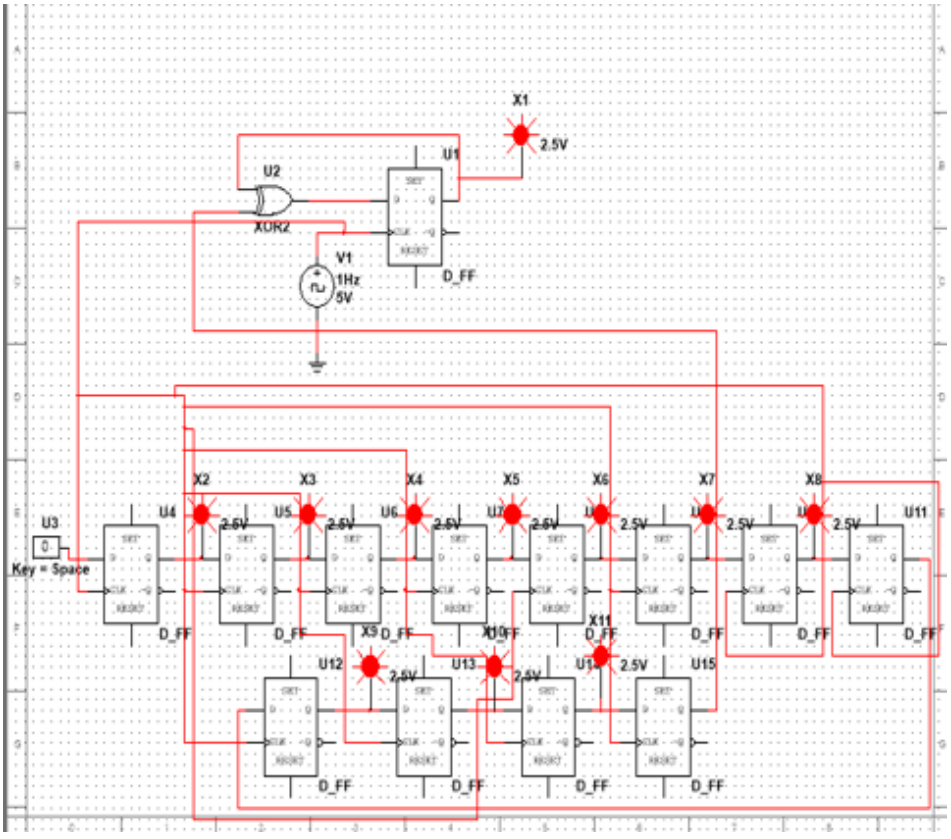
Input stream (*start from right*) **1 0 0 1 1 0 1 0 1 1 1 0** should be provided to the system using serial register.

1. Using D-flipflop, design the circuit diagram *(perform all the calculations and upload scanned copy at the end of the 1st hour)*

2. Write the HDL code for the circuit *(upload Verilog code file at the end of lab)*

3. Implement the circuit and check the results *(upload the circuit and results at the end of the lab)*

4. ***Submit a complete report before dead line***



```
                                    lab13 b2 (B[0],CLK,B[1]);
module labl3 (A,CLK,B}              LaLl3 b3 (B[1)CIE,B[2}:
input A,CLH;                        labl3 b4 (B[2),CIF,B[3)};
outpu€ B:                           Labl3 b5 (B[3),CLF,B[4):
xg  B:                              Latl3 b6 (B4),CLK,B[5):
initial                             Latl3  b7  (B[5)CIE,B[6}:
begin                               LaLl3  b9  (B[6)CIE,B[7):
B=0:                                LaLl3 bS (B[7)CIF,B[9):
end                                 LaLl3 bl0 (B[9)CIX,B[S) *l
always  @(posedge  CLK}  Labl3  bll  (B[S),CLK,B[10)}:
case (A}                            LaLl3 bl2 (B[l0],CIA,B[ll]Q
l'b0  :  B C= 0:                    initial begin
l'bl  :  B <= 1;                    C=0:
endcase                             end
endmodule                           gate  gl  (B[ll),C,B[12)}:
module  gate(C,D,E}:  Labl3  bl3  (B[l2],CLH,B[13]}:
input C,D:                          initial
output E:                           begin
xorxl(E,C,D}:                       CLK= 0:
endmodule                           end
module test_Laasbl3(}:              always €5CLK=-CLK:
reg A,CLK,C:                        initial
win  [0:l3)B:                       begin
laLl3 b1 (A,CLF,B[0);               â=0; #10;
Labl3 b2 (B[0),CIA,B[1)}:           â=l:  10:
LaLl3 b3 (B[1),CIA,B[2)a            â=l:  10:
LaLl3  b4  {B[2)CIX,B[3)}           â=l:  010:
LaLl3  b5  (B[3)CIE,B[4)a           é=0:  010:
LaLl3  b6  (B[4)CIE,B[5):           A=l: #10:
LaLl3  b7  (B[5)CIX,B[6):           A=0: #10:
LaLl3  b9  (B[6)CIX,B[7):           é=l: #10:
LaLl3  bS  (B[7)CIX,B[9):           A=l:  #10:
LaLl3  bl0  (B[9)CIE,B[S}:          â=0:  #10:
labl3 bll (B[S),CIF,B[10)}; é=0: #10:
labl3 b12 (B[l0),CLF,B[11)}; A=l; #10;
initial begin                       end
C=0:                                endmodule
end
```