East West University

Fall – 2020

Name : Bijoy Basak

ID No : 2018-2-60-033

Section : 02

Course Instructor : Musharrat Khan
Senior Lecturer, Department of
Computer Science and Engineering

Course Title : Digital Logic Design

Course Code : CSE345

Project Title : 4-bit Binary to Odd Parity Code Converter

Date of Submission : 02th January, 2021

Problem Statement

A 4-bit code converter for odd parity converts the 4-bit input to 5-bit output so that odd parity is ensured.

Design Details

The parity generating technique is one of the most widely used error detection techniques for the data transmission. In digital systems, when binary data is transmitted and processed, data may be subjected to noise so that such noise can alter 0s (of data bits) to 1s and 1s to 0s.

The sum of the data bits and parity bits can be even or odd. In even parity, the added parity bit will make the total number of 1s an even amount whereas in odd parity the added parity bit will make the total number of 1s odd amount. The basic principle involved in the implementation of parity circuits is that sum of odd number of 1s is always 1 and sum of even number of 1s is always zero.

In odd parity bit scheme, the parity bit is '1' if there are even number of 1s in the data stream and the parity bit is '0' if there are odd number of 1s in the data stream.

According the theory, Then I constructed the truth table and determined the equation by K-map. After that, I build the logic diagram in Quartus II Software.

Truth Table:

4-BITS INPUTS				**5-BITS OUTPUTS**				
i0	**i1**	**i2**	**i3**	**o0**	**o1**	**o2**	**o3**	**o4**
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	1	1
0	1	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1	1
0	1	1	0	0	1	1	0	1
0	1	1	1	0	1	1	1	0
1	0	0	0	1	0	0	0	0
1	0	0	1	1	0	0	1	1
1	0	1	0	1	0	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	0	1	1	0	0	1
1	1	0	1	1	1	0	1	0
1	1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	1	1

K-Maps:

K-Map for o0:

io i1\ i2 i3	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

K-Map expression:

o0 = i0

K-Map for o1:

io i1\ i2 i3	00	01	11	10
00				
01	1	1	1	1
11	1	1	1	1
10				

K-Map expression:

o1 = i1

K-Map for o2:

io i1\ i2 i3	00	01	11	10
00			1	1
01			1	1
11			1	1
10			1	1

K-Map expression:

o2 = i2

K-Map for o3:

io i1\ i2 i3	00	01	11	10
00		1	1	
01		1	1	
11		1	1	
10		1	1	

K-Map expression:

o3 = i3

K-Map for o4(Parity Bit):

io i1\ i2 i3	00	01	11	10
00	1		1	
01		1		1
11	1		1	
10		1		1

K-Map expression:

o4 = io`i1`i2`i3` + io`i1`i2i3 + io`i1i2`i3 + io`i1i2i3` +

ioi1`i2`i3 + ioi1`i2i3` + ioi1i2`i3` + ioi1i2i3

$= i0 \otimes i1 \otimes i2 \otimes i3$

Circuit Diagram:

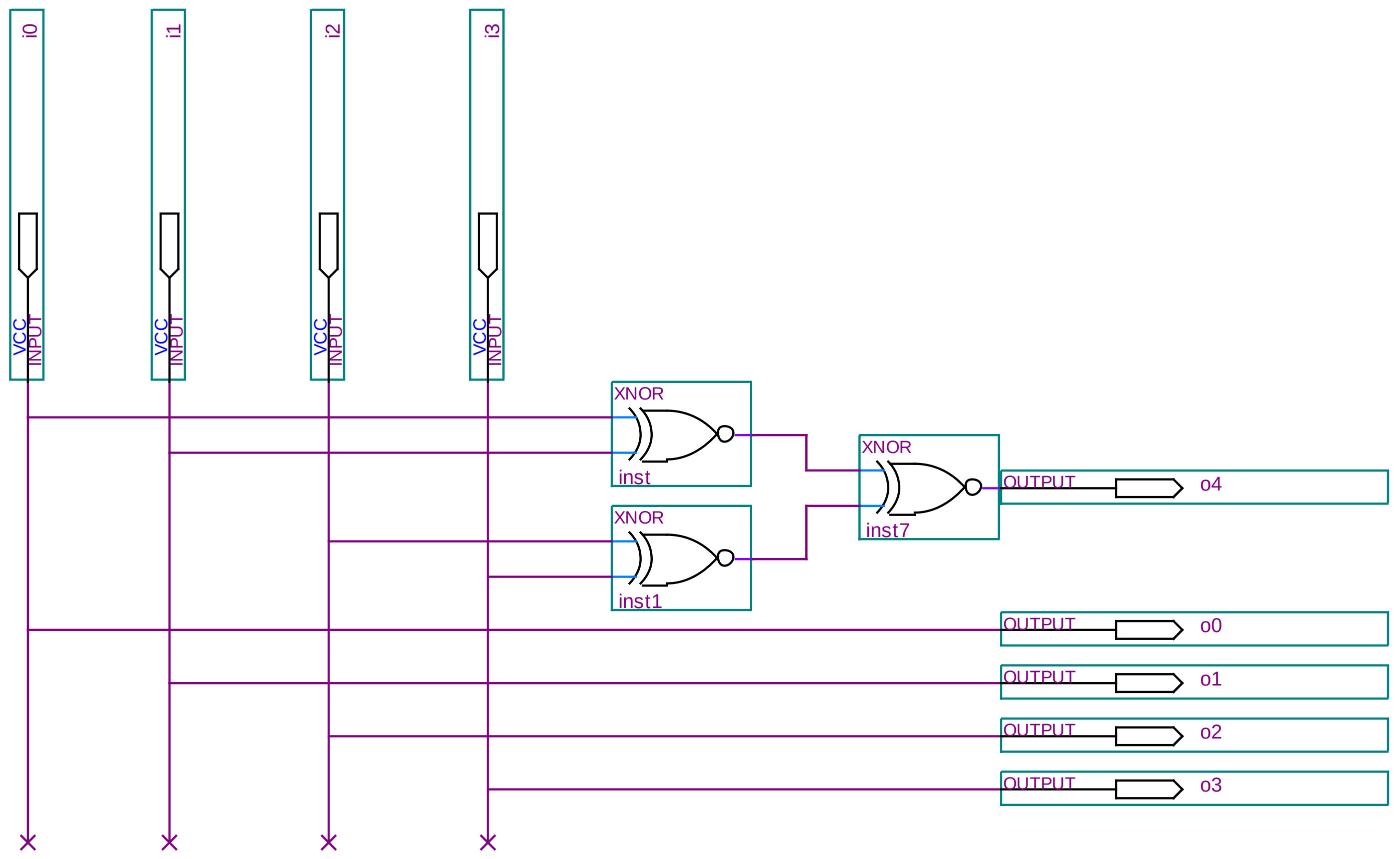


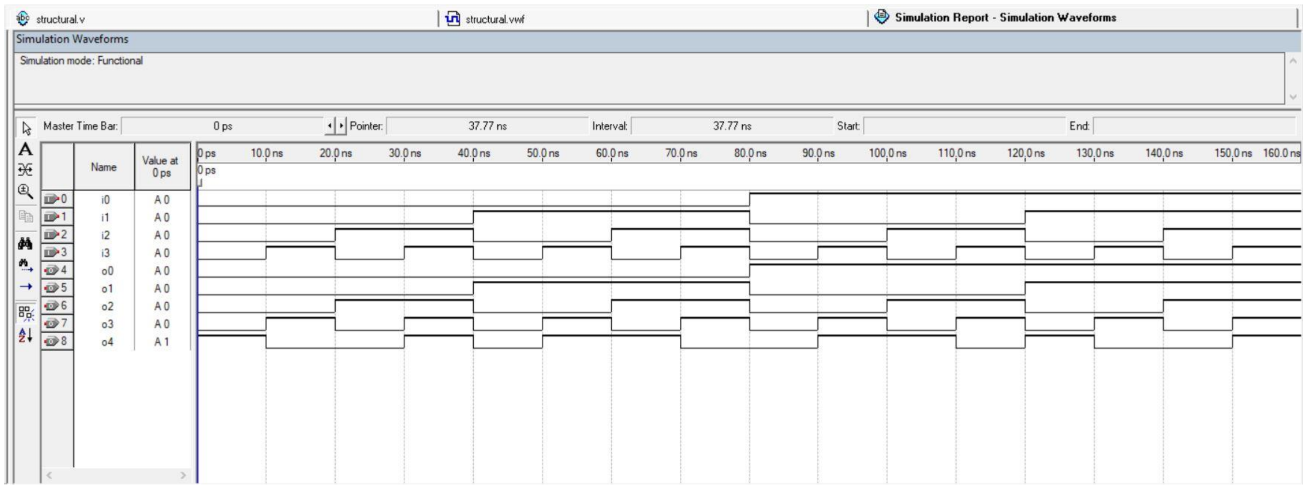
Fig 1: Circuit Diagram

Structural and Behavioral Verilog Codes and Simulation Results:

Structural Code:

```
module structural(input i0, i1, i2, i3, output o0, o1, o2, o3, o4);
wire w1, w2;
        assign o0 = i0;
        assign o1 = i1;
        assign o2 = i2;
        assign o3 = i3;
        xnor g1(w1,i0,i1),
                g2(w2,i2,i3);
        xnor g3(o4,w1,w2);
endmodule
```

Structural Simulation Result:


structural.v | structural.vwf | Simulation Report - Simulation Waveforms

Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 0 ps | Pointer: 37.77 ns | Interval: 37.77 ns | Start: | End:

	Name	Value at 0 ps
0	i0	A 0
1	i1	A 0
2	i2	A 0
3	i3	A 0
4	o0	A 0
5	o1	A 0
6	o2	A 0
7	o3	A 0
8	o4	A 1

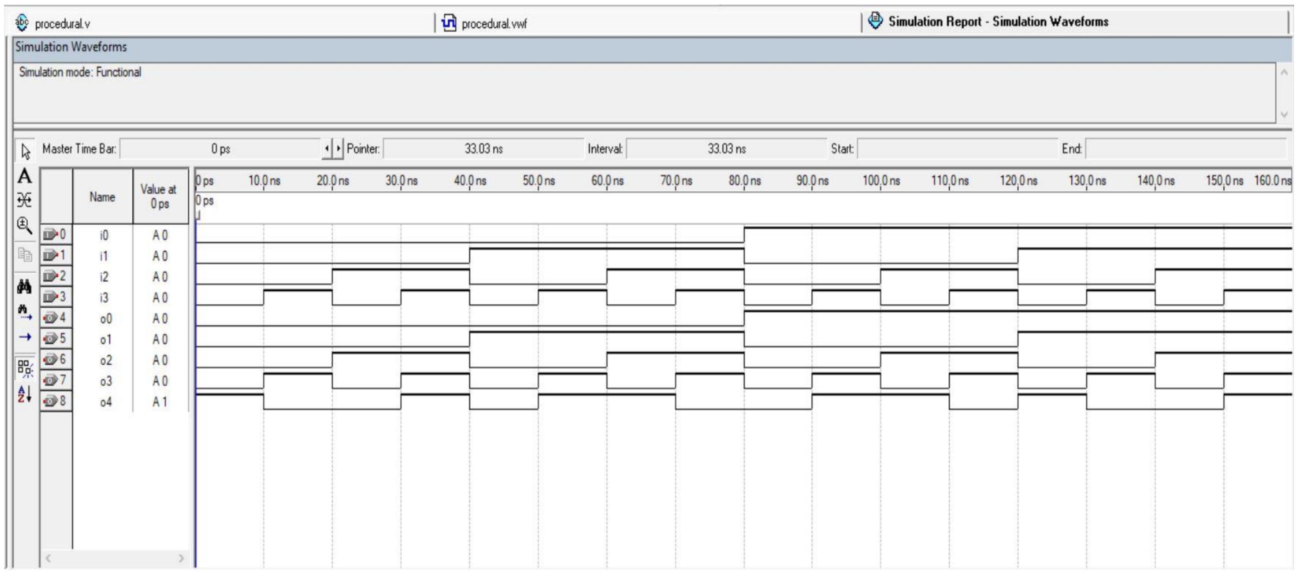
Fig 2: Structural Simulation Result

Behavioral Verilog Codes and simulation results:

Procedural Verilog Code:

```
module procedural(input i0, i1, i2, i3, output reg o0, o1, o2, o3, o4);
    always@(i0, i1, i2, i3) begin
        o0 = i0;
        o1 = i1;
        o2 = i2;
        o3 = i3;
        o4 = 0;
        if(~i0&~i1&~i2&~i3) o4 =1;
        if(~i0&~i1&i2&i3) o4 =1;
        if(~i0&i1&~i2&i3) o4 =1;
        if(~i0&i1&i2&~i3) o4 =1;
        if(i0&~i1&~i2&i3) o4 =1;
        if(i0&~i1&i2&~i3) o4 =1;
        if(i0&i1&~i2&~i3) o4 =1;
        if(i0&i1&i2&i3) o4 =1;
    end
endmodule
```

Procedural Verilog Simulation Result:



Simulation mode: Functional

Master Time Bar: 0 ps   Pointer: 33.03 ns   Interval: 33.03 ns   Start: End:

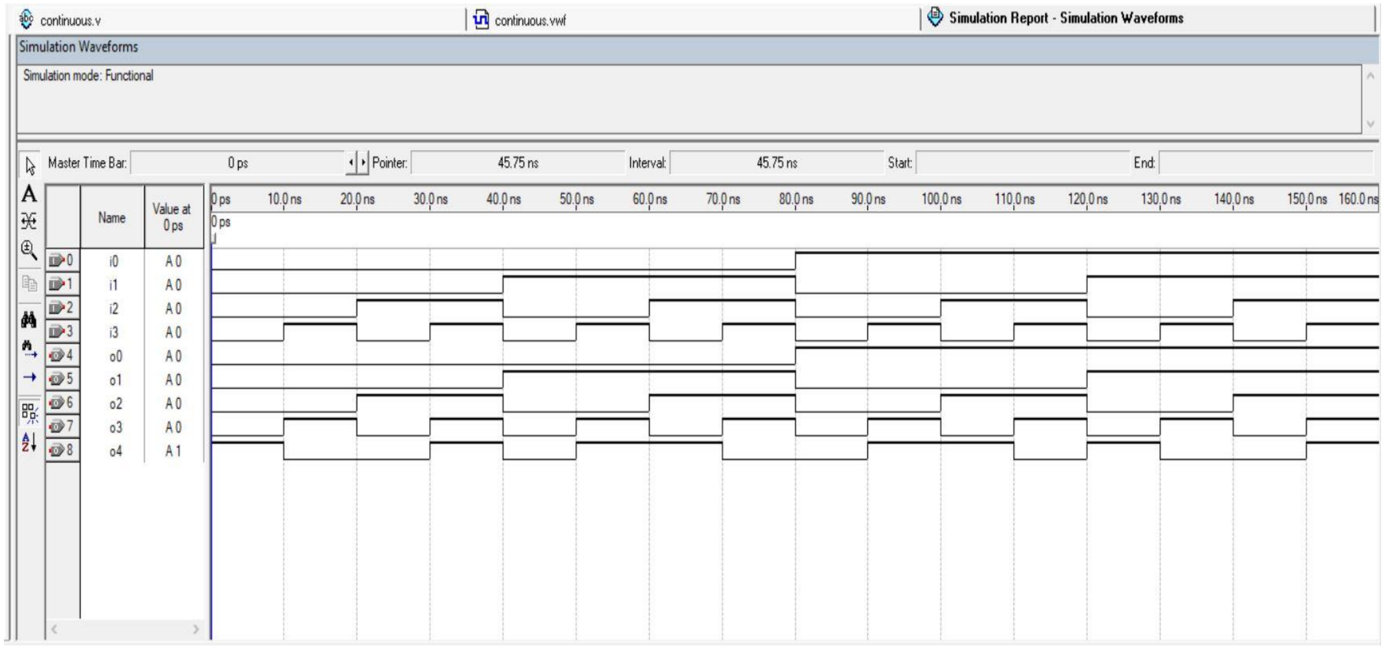
	Name	Value at 0 ps
0	i0	A 0
1	i1	A 0
2	i2	A 0
3	i3	A 0
4	o0	A 0
5	o1	A 0
6	o2	A 0
7	o3	A 0
8	o4	A 1

Fig 3: Procedural Verilog Simulation Result

Continuous Assign Verilog Code:

```
module continuous(input i0, i1, i2, i3, output o0, o1, o2, o3, o4);

assign o0 = i0;

assign o1 = i1;

assign o2 = i2;

assign o3 = i3;

assign o4 =
(~i0&~i1&~i2&~i3)|(~i0&~i1&i2&i3)|(~i0&i1&~i2&i3)|(~i0&i1&i2&
~i3)|(i0&~i1&~i2&i3)|(i0&~i1&i2&~i3)|(i0&i1&~i2&~i3)|(i0&i1&i2&
i3);

endmodule
```

Continuous Assign Verilog Simulation Result:



Simulation mode: Functional

Master Time Bar: 0 ps   Pointer: 45.75 ns   Interval: 45.75 ns   Start: End:

	Name	Value at 0 ps
0	i0	A 0
1	i1	A 0
2	i2	A 0
3	i3	A 0
4	o0	A 0
5	o1	A 0
6	o2	A 0
7	o3	A 0
8	o4	A 1

Fig 4: Continuous Assign Verilog Simulation Result