

DLD VT-2 REPORT

AIM- To implement odd parity generator for 4-Bit binary number using VHDL and write the code for design and testbench. (Question 8-B)

THEORY- 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. Hence, a **Parity Bit** is added to the word containing data in order to make number of 1s either even or odd. The message containing the data bits along with parity bit is transmitted from transmitter to the receiver. At receiving end, the number of 1s in the message is counted and if it doesn't match with the transmitted one, it means there is an error in the data. Thus, the Parity Bit is used to detect errors, during the transmission of binary data.

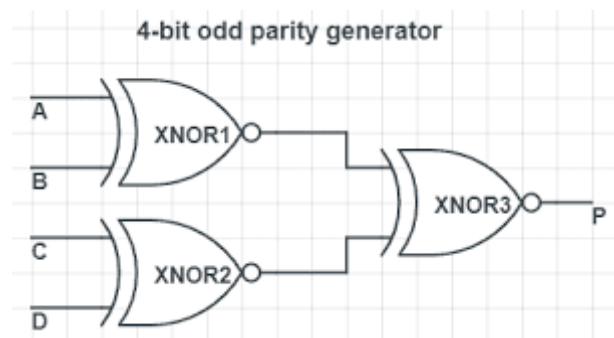
The sum of the data bits and parity bits can be even or odd. In even parity, the added parity bit will make the total number of 1s an even number, whereas in odd parity, the added parity bit will make the total number of 1s an odd number.

If we have 4 bits, then it becomes a 4-bit odd parity generator. Now the output odd parity bit would be decided on the basis of 4 input bits, namely D₃, D₂, D₁, and D₀.

Table 1

D ₃	D ₂	D ₁	D ₀	Even-parity P	Odd-parity P
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

On solving the table-1 using K-Maps we get the expression for odd parity as-
 $\text{NOT}(((D3 \text{ XOR } D2) \text{ XOR } D1) \text{ XOR } D0)$



METHODOLOGY- 1. Study the logic of 4-Bit Odd Parity Generator and drawing the truth table.

2. Solve for odd-parity from the truth table using K-Map.

3. Write the code for designing 4-Bit Odd Parity Generator Model in VHDL using the expression derived from K-Map.

4. Write the code for testbench in VHDL.

5. Run the code and obtain the waveform.


CODE-

Link for the code- <https://www.edaplayground.com/x/A4RL>

A. DESIGN:

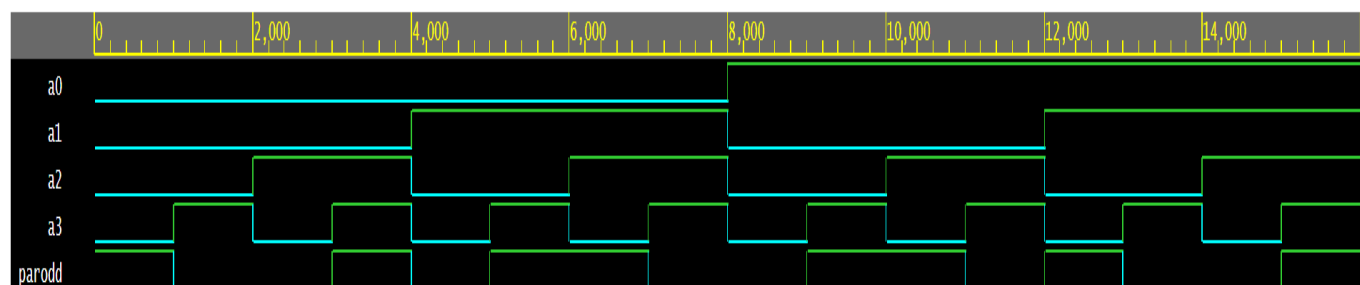
```
design.vhd  +
1  -- Code your design here
2  Library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity oddpargen is
6      port(a0,a1,a2,a3:in bit;
7           parodd:out bit);
8  end oddpargen;
9
10 architecture arc of oddpargen is
11 begin
12     parodd <= not(((a0 xor a1) xor a2) xor a3);
13 end arc;
14
```

B. TESTBENCH:

```
testbench.vhd  VHDL Testbench

1  Library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity testbench is
5  end testbench;
6
7  architecture tb of testbench is
8  component oddpargen is
9      port(a0,a1,a2,a3:in bit;
10         parodd:out bit);
11 end component;
12 signal aS,bS,cS,dS,oS: bit;
13
14 begin
15     dut: oddpargen port map(aS,bS,cS,dS,oS);
16     process
17     begin
18         aS<='0'; bS<='0'; cS<='0'; dS<='0'; wait for 1 ns;
19         aS<='0'; bS<='0'; cS<='0'; dS<='1'; wait for 1 ns;
20         aS<='0'; bS<='0'; cS<='1'; dS<='0'; wait for 1 ns;
21         aS<='0'; bS<='0'; cS<='1'; dS<='1'; wait for 1 ns;
22         aS<='0'; bS<='1'; cS<='0'; dS<='0'; wait for 1 ns;
23         aS<='0'; bS<='1'; cS<='0'; dS<='1'; wait for 1 ns;
24         aS<='0'; bS<='1'; cS<='1'; dS<='0'; wait for 1 ns;
25         aS<='0'; bS<='1'; cS<='1'; dS<='1'; wait for 1 ns;
26         aS<='1'; bS<='0'; cS<='0'; dS<='0'; wait for 1 ns;
27         aS<='1'; bS<='0'; cS<='0'; dS<='1'; wait for 1 ns;
28         aS<='1'; bS<='0'; cS<='1'; dS<='0'; wait for 1 ns;
29         aS<='1'; bS<='0'; cS<='1'; dS<='1'; wait for 1 ns;
30         aS<='1'; bS<='1'; cS<='0'; dS<='0'; wait for 1 ns;
31         aS<='1'; bS<='1'; cS<='0'; dS<='1'; wait for 1 ns;
32         aS<='1'; bS<='1'; cS<='1'; dS<='0'; wait for 1 ns;
33         aS<='1'; bS<='1'; cS<='1'; dS<='1'; wait for 1 ns;
34         wait;
35     end process;
36 end tb;
```

RESULT AND WAVEFORM-



CONCLUSION- From the waveform it is clearly visible that “parodd” is 1 only when the sum of number of 1’s in “a0”, “a1”, “a2” and “a3” is even so as to make the total number of 1’s odd.

Therefore 4-Bit Odd Parity Generator is successfully implemented using VHDL.

PREPARED BY- Amogh Garg (2020UCO1688)

Yash Jindal (2020UCO1689)

Group Number-22