Aim:-

To design and simulate the behaviour of half adder and full adder using logic gates in VHDL

Objective:-

- To understand the working of half adder and full adder.
- To simulate both the adders and verify the truth table.

Theory:-

Half-adder:-

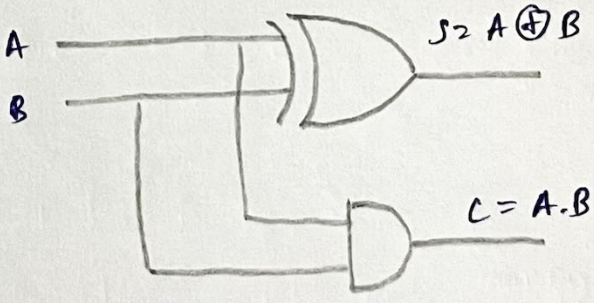
Half adder is a combinational circuit that adds two single bit binary number and produces a sum and carry as output.

sum = A XOR B

carry = A AND B

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

truth table



circuit diagram

Full-adder :-

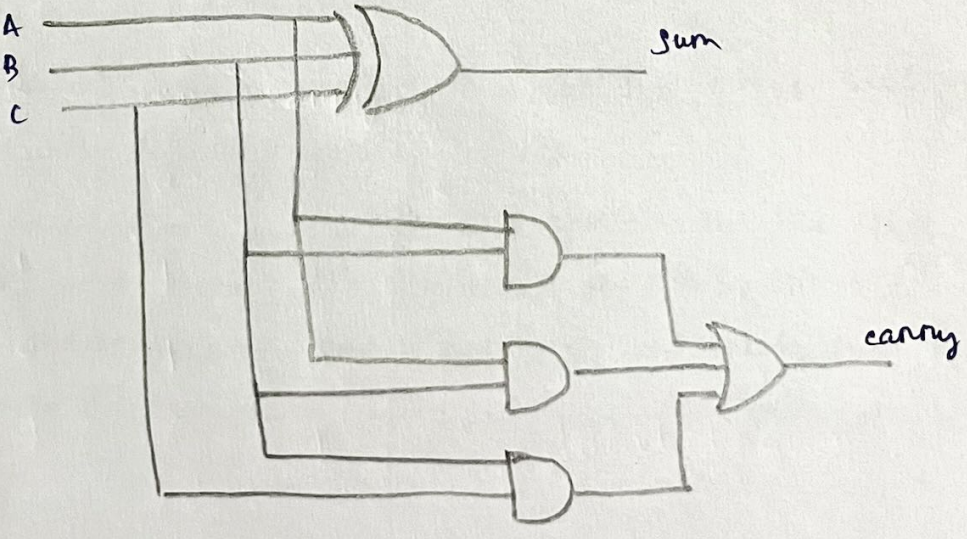
A full adder is a combinational circuit that adds three single bit binary numbers and produces a sum along with a carry.

→ Sum = (A XOR B) XOR C

→ Carry = (A . AND B) OR (B AND C) OR (A AND C)

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table



circuit diagram

HALF ADDER

DESIGN SOURCE CODE

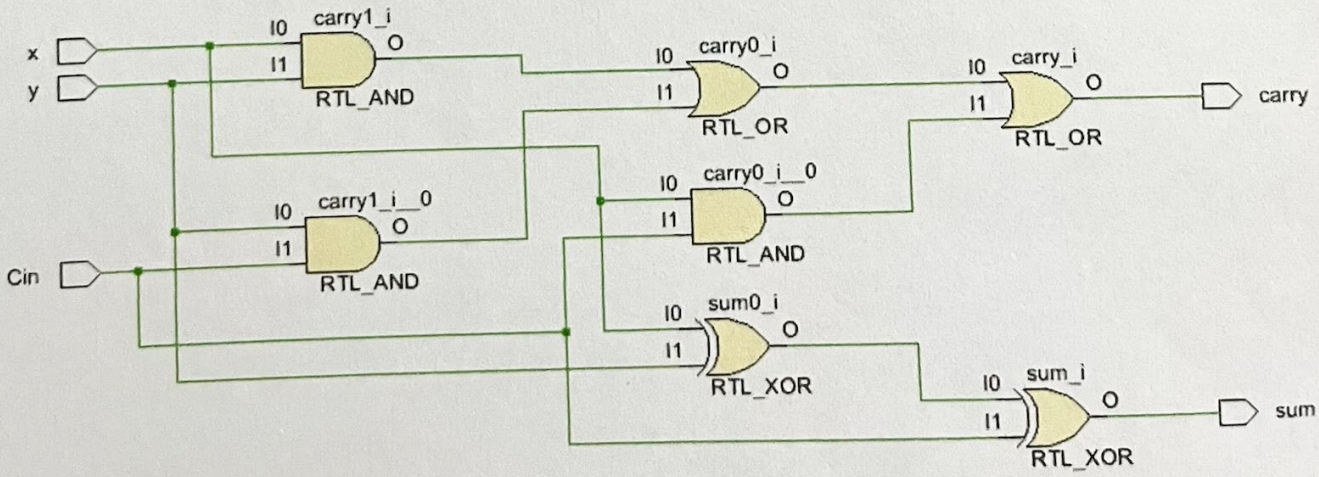
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity half_adder is
    Port ( a,b : in STD_LOGIC;
           sum,carry : out STD_LOGIC);
end half_adder;


architecture dataflow of half_adder is


begin
sum <= a xor b;
carry <= a and b;


end dataflow;
```

Full adder

Design Source Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder is
   Port ( x,y,Cin : in STD_LOGIC;
          sum,carry : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is

begin
sum<= x xor y xor Cin;
carry<=(x and y) or (y and Cin)or (x and Cin);

end Behavioral;
```

RTL Schematic:



Conclusion:-

Half adders and full adders are basic circuits used in binary addition. The half adder adds two binary numbers but cannot handle a carry, while the full adders adds three numbers, including a carry. Together, they are key components in building arithmetic operations in digital systems like computers and calculators.

EXPERIMENT-3

Aim:- Study, design and simulation of 4-bit parallel adder using full-adder.

Objective:- To understand the working of 4 bit parallel adder To stimulate the 4-bit parallel adder and verify the truth table.

Theory:-

4-bit parallel adder:-

A 4-bit parallel adder is a digital circuit that adds two 4-bit binary numbers and produces a 4-bit sum and a carry out bit. It uses four full-adders connected in series where each full-adder add the corresponding bits from the two input numbers and includes a carry input from the previous less significant stage.

* inputs:- Two 4-bit binary numbers, A(A3, A2, A1, A0) and B(B3, B2, B1, B0).
* Outputs:- A 4-bit sum S(S3, S2, S1, S0) and carry out (cout).
* Full adder:- Each stage of adder has a carry input and produces a sum and carry output

Truth table:-

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

4-Bit Parallal Adder :-

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity para4bit is
  Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
       b : in STD_LOGIC_VECTOR (3 downto 0);
        cin : in STD_LOGIC;
        s : out STD_LOGIC_VECTOR (3 downto 0);
        cout : out STD_LOGIC);
end para4bit; architecture


Behavioral of para4bit is


component fulladder is
Port ( a : in STD_LOGIC;
      b : in STD_LOGIC;
      cin : in STD_LOGIC;
      s : out STD_LOGIC;
       cout : out STD_LOGIC);
end component; signal p :


STD_LOGIC_VECTOR (3 downto 1); begin

u0:fulladder port map (a(0),b(0),cin,s(0),p(1)); u1:fulladder
port map (a(1),b(1),p(1),s(1),p(2)); u2:fulladder port map
(a(2),b(2),p(2),s(2),p(3)); u3:fulladder port map
(a(3),b(3),p(3),s(3),cout);


end Behavioral;
```

Conclusion:-

The 4-bit parallel adder efficiently adds 4-bit binary numbers using 4 Full adders producing a 4-bit sum and a carry-out it is fundamental building block in digital systems, forming the basis for arithmetic operations in processors and ALU's. It 's simplicity and scalability make it a critical component in logic design.

EXPERIMENT-4

Aim:- Design and simulate a 2bit magnite comparator using VHDL.

Objective:- to understand the working of a 2bit magnitude comparator.

2 bit magnitude comparator:-

A 2-bit magnitude comparator is a combinational logic circuit used to compare two 2-bit binary numbers A(A0,A1) and B(B0,B1). It determines the relationship between the two numbers and produces three outputs.

1. A = B (A is equal to B)
2. A > B (A is greater than B)
3. A < B (A is less than B)

Inputs:-

Two 2-bit numbers

A = A0, A1 B = B0, B1.

Outputs:-

F0 = A = B, F1 = A > B, F2 = A < B

Truth table:-

A0	A1	B0	B1	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

2-BIT COMPARATOR:-

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity twobitcomparator is
   Port ( a0 : in STD_LOGIC;
          a1 : in STD_LOGIC;
          b0 : in STD_LOGIC;
          b1 : in STD_LOGIC;
          E : out STD_LOGIC;
          G : out STD_LOGIC;
          L : out STD_LOGIC);
end twobitcomparator;

architecture Behavioral of twobitcomparator is

begin
E<=((a0 xnor b0) and (a1 xnor b1));
G<=(a1 and (not b1)) or ((a1 xnor b1) and a0 and (not b0));
L<=((not a1) and b1) or ((a1 xnor b1) and ((not a0) and b0));


end Behavioral;
```

Conclusion:- The 2-bit magnitude comparator compares two 2-bit binary numbers and determines their relationships (A = B, A > B, A < B). It is a vital component in decision-making circuits and digital systems.

EXPERIMENT-5

Aim:- Design and Stimulate the behavioral, Dataflow 2×1 mux and Behavioral, Dataflow and Structural model of 4×1 mux.

Objective:- To understand the working of 2×1 and 4×1 mux. To simulate the 2×1 and 4×1 mux and verify the truth table.

Theory:-

Multiplexor:-

A multiplexor is a combinational circuit that selects one of several inputs signals and forwards it to a single output line. The selection of the input is controlled by selection lines.

2×1 mux:-

A 2×1 multiplexor has two data inputs (A,B), one selection line (S) and one output (F). The selection line determines which input is connected to the output.

if $S=0, Y=A$

if $S=1, Y=B$

Truth table:-

S	I/P	F
0	A	1
1	B	1

4×1 mux:- A 4×1 multiplexor has 4 inputs (A,B,C,D) and two selection line (S0, S1) and one output (F). The combination of selection lines determine which input is connected to the output.

If, $S1S0 = 00 = A$

$S1S0 = 01 = B$

$S1S0 = 10 = C$

$S1S0 = 11 = D$

Truth table:-

S1	S0	I/P	O/P
0	0	A	1
0	1	B	1
1	0	C	1
1	1	D	1

Procedure:-

1) Open vivado software.
2) Click on create a new project and enter name with location to save.
3) Select the file type as "RTL Project" and click on next and select the language as VHDL.
4) Now click on the module "Add Sources" and click on "create file" give a file name..
5) Now on module defination add a entity name and add the architecture name.
6) Now in I/O ports defination select and add the inputs and output ports.
7) Now goto design sources and click on that file i.e created above.
8) Now write the logic for the 2x1 and 4x1 multiplexer.
9) Now goto "RTL Schematic" and click on open "elaborated design" and click on Schematic to view the Schematic diagram.
10) Now click on "Simulation" and then "Run - Simulation" and then click on "Run behavioral simulation".

2X1 MUX Behavioural :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_2X1 is
   Port ( A , B , S : in STD_LOGIC;
          F : out STD_LOGIC);
end MUX_2X1;

architecture Behavioral of MUX_2X1 is
begin
process(A , B , S)
begin

if S = '0' then

F <= A;
else
F <= B;

end if;
end process;
end Behavioral;
```

2X1 MUX Dataflow :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity DataFlow_2X1_MUX is
    Port ( A,B,S : in STD_LOGIC;
           F : out STD_LOGIC);
end DataFlow_2X1_MUX;

architecture Dataflow of DataFlow_2X1_MUX is

begin
F <= (A and not S) or (B and S);

end Dataflow;
```

4X1 MUX Dataflow:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_4X1_Dataflow is
   Port ( A,B,C,D,S0,S1 : in STD_LOGIC;
          F : out STD_LOGIC);
end MUX_4X1_Dataflow;

architecture Dataflow of MUX_4X1_Dataflow is

begin
F <= (A and not S0 and not S1) or (B and not S0 and  S1) or (C and S0
and not S1) or (D and  S0 and  S1);

end Dataflow;
```

4x1 Behavioural :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_4X1_Behavioural is
   Port ( A,B,C,D,S0,S1 : in STD_LOGIC;
          F : out STD_LOGIC);
end MUX_4X1_Behavioural;

architecture Behavioral of MUX_4X1_Behavioural is

begin
process(A, B ,C ,D , S0, S1)
begin
if (S0 = '0' and S1='0') then
F <= A;
elsif (S0 = '0' and S1='1') then
F <= B;
elsif (S0 = '1' and S1='0') then
F <= C;
else
F <= D;
end if;
```

4X1 MUX using 2X1 MUX :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_4X1_USING_2X1 is
   Port ( A,B,C,D,S0,S1 : in STD_LOGIC;
          F : out STD_LOGIC);
end MUX_4X1_USING_2X1;

architecture Structural of MUX_4X1_USING_2X1 is

Component MUX_2X1 is
   Port ( A , B , S : in STD_LOGIC;
          F : out STD_LOGIC);
end Component;

Signal T1, T2: STD_LOGIC;

begin
a0:MUX_2X1 port map(A , B , S1 , T1);
a1:MUX_2X1 port map(C , D , S0 , T2);
a2:MUX_2X1 port map(T1 , T2 , S0 , F);
end Structural;
```

Conclusion:-

Multiplexer (mux) are essential digital components that effectively route one of the several input signals to a single output based on selection lines. The 2×1 mux selects between 2 inputs, while the 4×1 mux selects between four inputs, demonstrating scalability in design the vesatile circuits are widely used in data routing, control systems, and digital communication, making them fundamental in modern electronics.