And Gate Experiments

Description :

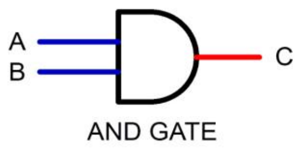
1. An AND gate is a basic digital logic gate.

2. It has multiple inputs of a single output.

3. The output is high (1) if all two inputs is high (1)

4. The output is low (0) if at least one input is low(0)

Logic Diagram



Truth Table

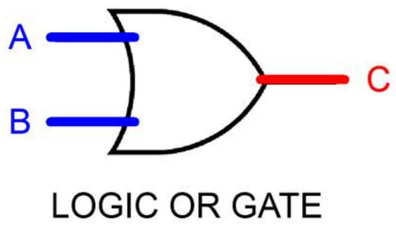
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

# Code:

```
Library IEEE;

Use IEEE.std_logic_1164.all;

Entity and_gate is

Port( A: in std_logic;

      B: in std_logic;

      Y: in std_logic;

end and_gate;

Architecture dataflow of and_gate if

begin

Y<=A and B;

end dataflow
```

OR Gate

Logic diagram



Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Code:

```
Library IEEE;

U+

3.se IEEE.std_logic 1164;

Entity or_gate is

Port( A: is std_logic;

B:  is std_logic;

            Y: is std_logic);

End or_gate;

Architechture dataflow of or_gate is begin

Y<= A or B

End dataflow
```

Description :

- An OR gate is a basic digital logic gate.
- It was multiple inputs and a single output.
- The output its high (1) is at least one input is high.
- The output is low (0) if all input is low (0)

Not Gate

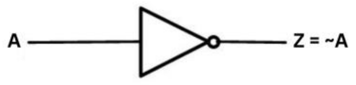
Description :

* A Not gate is basic digital logic gate.

* It has single input and a single output.

- The input to the NOT Gate is low, the output will be high.
- The input to the NOT Gate is high, the output will be low.

Logic Diagram:



Truth Table

A	B
0	1
1	0

Code:

```
Use IEEE.std_logic__all;

Entity not_gate is

Port( A: in std_logic;

Y: out std_logic);

End not_gate;

Architecture dataflow of not_gate is

Begin

Y<= not A;
```

```
End dataflow;
```

XOR Gate

Description:

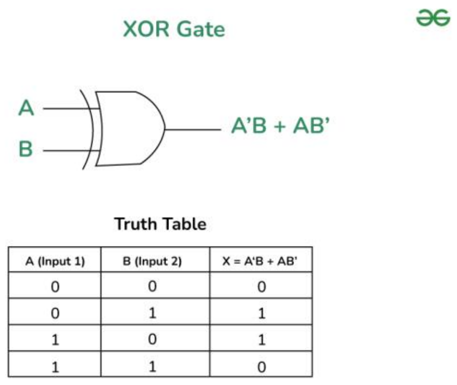
 * An XOR Gate is a basic digital logic gate.

* It was two input and a single output.

* The output of an XOR gate is high if and only if the input are different.

* When both input to the XOR gate are low or both are high, the output will be low.

Diagram:



A (Input 1)	B (Input 2)	X = A'B + AB'
0	0	0
0	1	1
1	0	1
1	1	0

Code:

```
Library IEEE;

Use IEEE.std_logic_1164.all;

Entit XOR_gate is

        port( A: in std_logic;

        B: in std_logic;

        Y: out std_logic);

End XOR_gate;

Architecture dataflow of XOR-gate is
```

```
Begin

Y<= A XOR B;

End dataflow;
```

NOR Gate

Description:

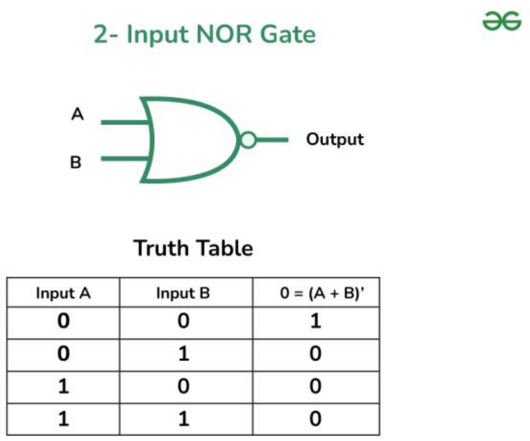
* A NOR gate is a universal Gate.

* It has two inputs and a single output.

* It all input to the NOR gate are low, the output will be high.

* If any input to the NOR gate if high, the output will be low.

Diagram:



Input A	Input B	0 = (A + B)'
0	0	1
0	1	0
1	0	0
1	1	0

Code:

```
Library IEEE;

Use IEEE.std_logic_1164.all;

Entity NOR_GATE is

Port( A: in std_logic;

B:  in std_logic;

Y: out std_logic;
```

End NOR_GATE;

Architecture dataflow of NOR_GATE is

Begin

Y<= A nor B;

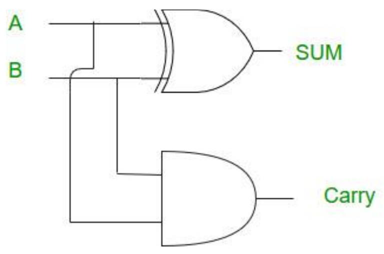
End dataflow;

Half Adder

Description:

- A half adder is a basic digital combination circuit used for adding two single-bit binary numbers.
- It has two inputs (typically labeled as A and B) and two outputs
  - Sum(s): Represents the result of addition
  - Carry(c): Represents the carry-out, which occurs when both inputs are 1.
- If both inputs are low, the sum will be low and the carry will also be low)
- If one input is low and the other is 1, the sum will be high and the carry will be low.
- If both inputs are high, the sum will be low and the carry will be high.

Diagram



Truth Table

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

Code

Lirary IEEE.std_logic_1164.all;

Entity half_adder is

Port(A: in std_logic;

B: is std_logic;

        Sum: out std_logic;

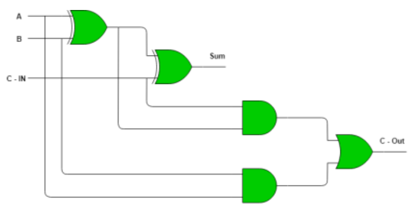
```
            Carry: Out std_logic;

End  half_adder

Architecture dataflow of half_adder is

Begin

Sum <= A XOR B;

Carry<= A and B;

End dataflow;
```

Full Adder: **Aim :-** Full adders are used in digital circuits for binary addition, particularly in arithmetic logic units (ALU) they are the building block for creating multiple address(such as 4 bit, 8 bit etc.) by connecting multiple address in series.

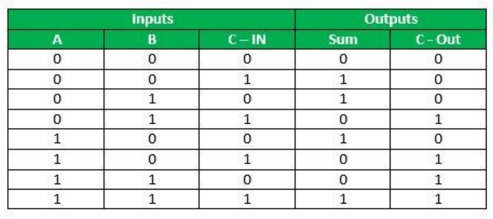
Definition:

A Full Adder is a digital circuit that adds three binary bits: two significant bits and a carry bit form a previous addition. It produces a sum and a carry output.

Diagram:



Truth Table



Inputs			Outputs	
A	B	C – IN	Sum	C - Out
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

Code

```
Library IEEE;

Use IEEE.std_logic_1164.all;

Entity full_adder is

Port ( A: in std_logic;

B: in std_logic;

C: in std_logic;

Sum: out std_logic;
```

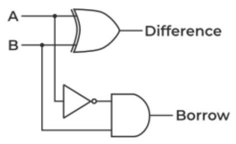
```
Carry-out: Out std_logic);

End full_adder;

Architecture full adder-arch of full_adder is

Begin

Sum<= A X or B X or C;

Carry-out = (a and b) or (c and (a x or b)0;

End full adder_arch;
```

Half subtractor

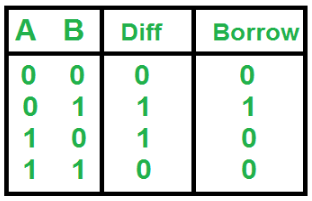
Describe:-

- Half subtractor is a basic digital logic circuit.
- It has two inputs (A & B) and two outputs (Difference and Borrow).
- The Half subtractor performs subtraction by calculating.

Logic Diagram:



Truth Table



A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Code:

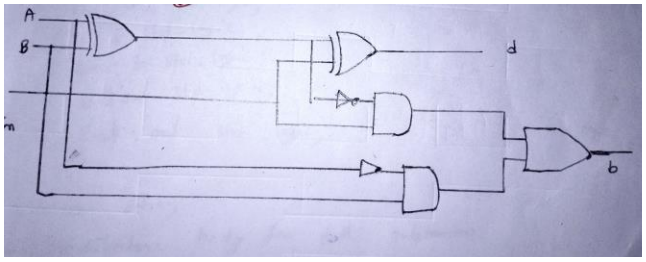
```
Library IEEE.std_logic_1164.all;

Entity Half_Subtractor is

Port(

A: in std_logic;

                    B: in std_logic;

                    Diff: Out std_logic;
                    Borrow: out std-logic;

                    );

End Half_Subtractor;

Architecture beahviour of Half_Subtractor is

Begin

Diff<= A XOR B;

Borrow <= NOT A AND B;

End Behavioral;
```

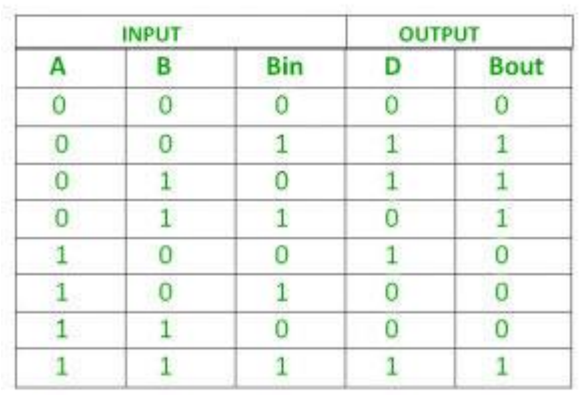
Full subtractor

Description:

1. A full subtractors hars three inputs : A, B, BorrowIn (Bin)
2. It has two outputs: Difference & Borrow out (Bart)
3. Difference output is 1 if the number of 1st the input is odd 0 if it is even, similar to an XOR operation.
4. The Borrow Out (Bout) output is 1 if a borrow is needed for the next subtraction stage.

Logic Diagram:





INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Code:
```
Library IEEE;Use IEEE.std_logic_1164.all;

Entity subs is:

Port(

A: in std_logic;

B: in std_logic;

Bin: in std_logic;

D: out std_logic;

Bout: out std_logic;
);End subs;

Architecture body for full subtractor

Architecture subs_arch of subs is being

Diff <= A xor B xor Bin

            Bout <= (not A and B) or (Band Bin) or (not A and Bin)


End subs_qreh;
```

1x4 De multiplexer:

→Description:

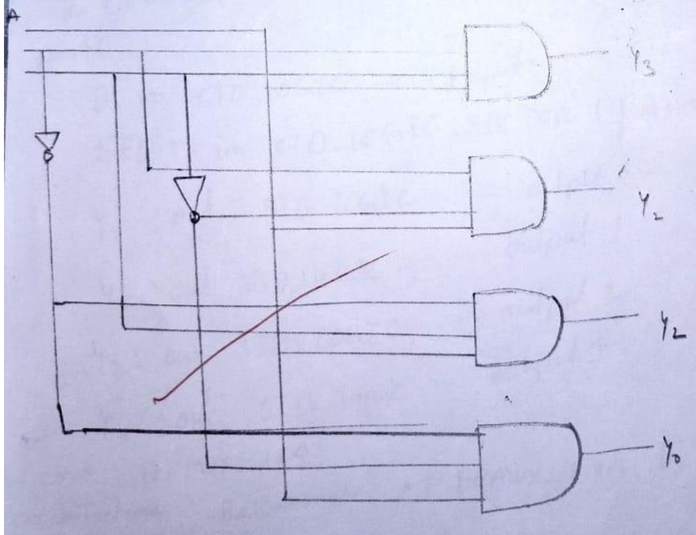
i) A demultiplexer is a digital circuit,

ii) It takes a single input and directs it to mutiple outputs.

(iii) Demultiplexer converts serial data to parallel data.

(iv) Demultiplexers are used for signal routing, duplication, and distribution.

→ Logic Diagram :-



1x4 Demultiplexer

->

Input		output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

-> code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL
entity DEMUX_1x4 is
  port(
     D: in STD_LOGIC; ---Input
     SEL : in STD_LOGIC_VECTOR (1 down to 0); --2-bit
                                               Selection signal
     Y0 : out STD_LOGIC ; -- output 0
     Y1 : out STD_LOGIC ; --- output 1
     Y2 : out STD_LOGIC; --- output 2
     Y3 : out STD_logic --- output 3
 end DEMUX_1x4;
architecture Behavioral of DEMUX_1x4 is
begin
      Process (D, SEL)
  begin --- Default all outputs to '0'
       Y0 <= '0',
       Y1 <= '0'
       Y3 <= '0',
```

4 x 1 Multiplexer :

> Description :-

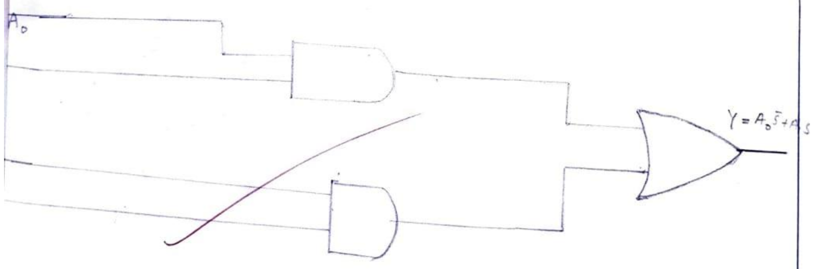
i) A 4 x 1 multiplexer is a digital circuit.

(ii) It has four data inputs, two selection lines, and one output.

(iii) A multiplexer selects input from multiple lines and sends it to a single output line.

(iv) Multiplexer are used for signal transmission, data compression, and channel sharing.

> Logic Diagram :-



ect inputs

4x1 Multiplexer

Truth Table :-

Inputs		output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

Code

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

entity MUX_4X1 is
   port (
     A : in STD_LOGIC ; -- input 0
     B : in STD_LOGIC; -- input 1
     C : in STD_LOGIC; -- input 2
     D : in STD_LOGIC; -- input 3
     SEL : in STD_LOGIC_VECTOR (1 down to 0); -- 2-bit selection signal
     Y : out STD_LOGIC -- output
  );
end MUX_4X1;
```