## Design and implement logic gates

**AIM:**

1. To design and implement basic logic gates.
2. To simulate basic logic gates VHDL using Xilinx ISE simulator.

## EQUIPMENT REQUIRED:

|  |  |  |
| --- | --- | --- |
| Sl. No | Components | Quantity |
| 1 | NOT gate IC7404 | 1 |
| 2 | 2 input AND gate IC7408 | 2 |
| 3 | 2 input OR gate IC7432 | 1 |
| 4 | Digital Trainer Kit | 1 |
| 5 | Patch Cords | As required |

**THEORY:**

1. The *AND gate* is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false." In other words, the output is 1 only when both inputs one AND two are 1.



**AND gate**

|  |  |  |
| --- | --- | --- |
| **Input 1** | **Input 2** | **Output** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. The *OR gate* gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1.



**OR gate**

|  |  |  |
| --- | --- | --- |
| **Input 1** | **Input 2** | **Output** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

1. A logical *inverter*, sometimes called a *NOT gate* to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.



**Inverter or NOT gate**

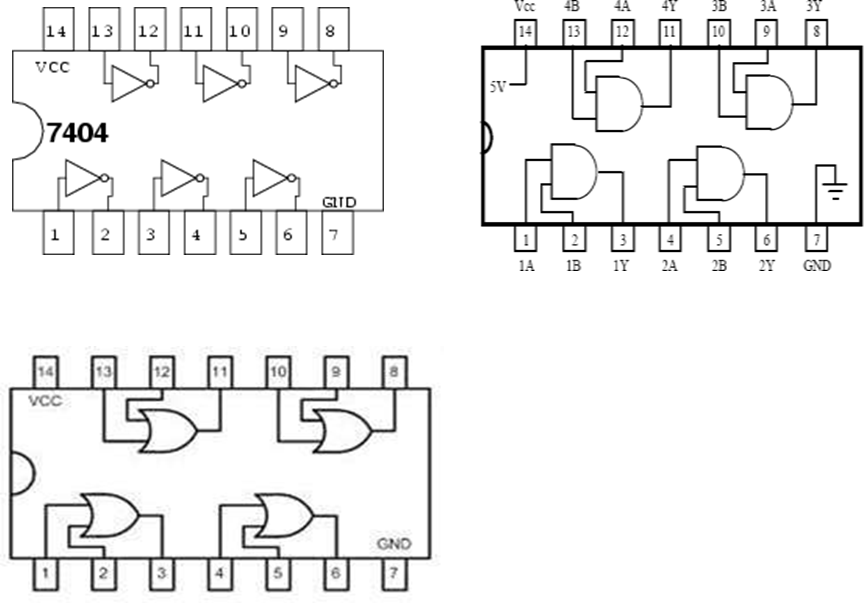
|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | 0 |
| 0 | 1 |

1. The *OR gate* gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1.
2. A logical *inverter*, sometimes called a *NOT gate* to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.

## PROCEDURE:

1. Check all the components for their working.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in the circuit diagram.
4. Give supply to the trainer kit.
5. Provide input data to the circuit via switches.
6. Observe the outputs and verify the Truth Table.

## BLOCK / CIRCUIT DIAGRAM:



## OBSERVATION TABLE:

Write the obtained truth table using kit and logical expression

## SIMULATION:

Write the program and their respective output wave form

## RESULTS & CONCLUSIONS:

1. The truth table of logic gates is verified.
2. The output waveform of logic gates is simulated and verified.

## 2) Design and implement Half adder, Full Adder, Half Subtractor, Full Subtractor using basic gates. And implement the same in HDL.

**AIM:**

1. To design and implement half adder, full adder, half subtractor and full subtractor using basic logic gates.
2. To simulate half adder, full adder, half subtractor and full subtractor in VHDL using Xilinx ISE simulator.

## EQUIPMENT REQUIRED:

|  |  |  |
| --- | --- | --- |
| Sl. No | Components | Quantity |
| 1 | NOT gate IC7404 | 1 |
| 2 | 2 input AND gate IC7408 | 2 |
| 3 | 2 input OR gate IC7432 | 1 |
| 4 | Digital Trainer Kit | 1 |
| 5 | Patch Cords | As required |

**THEORY:**

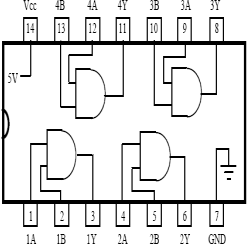
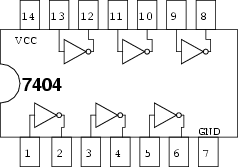
* Adder circuit is a combinational digital circuit that is used for adding numbers. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output. Adders are used in the [arithmetic logic units,](https://en.wikipedia.org/wiki/Arithmetic_logic_unit) in other parts of the processor, where they are used to calculate [addresses,](https://en.wikipedia.org/wiki/Address_%28computing%29) table indices, [increment and decrement operators.](https://en.wikipedia.org/wiki/Increment_and_decrement_operators)
* Half-Adder: A combinational logic circuit that adds two single binary digits *A* and *B*. It has two outputs, sum (*S*) and carry (*C*). The carry signal represents an [overflow](https://en.wikipedia.org/w/index.php?title=Overflow_%28computing%29&action=edit&redlink=1) into the next digit of a multi-digit addition.
* Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit (Cin). A combinational logic circuit that adds two data bits A, B, and a carry-in bit Cin, is called a full-adder. It has two outputs, sum (*S*) and carry (*C*).
* Subtractor circuit is a combinational digital circuit that is used for subtracting numbers. A typical subtractor circuit produces a diffrence bit (denoted by D) and a brrow bit (denoted by B) as the output.
* Half Subtractor: The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B (borrow).
* Full Subtractor: A combinational circuit of full-subtractor performs the operation of subtraction of three bits—the minuend, subtrahend, and borrow generated from the subtraction operation of previous significant digits and produces the outputs difference and borrow.

## PROCEDURE:

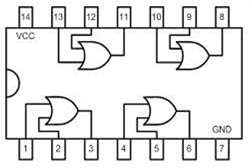
1. Check all the components for their working.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in the circuit diagram.
4. Give supply to the trainer kit.
5. Provide input data to the circuit via switches.
6. Observe the outputs and verify the Truth Table.

## BLOCK / CIRCUIT DIAGRAM:

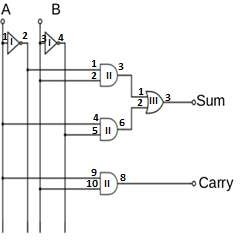
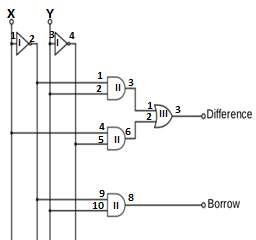
IC 7404 Pin Diagram IC 7408 Pin Diagram



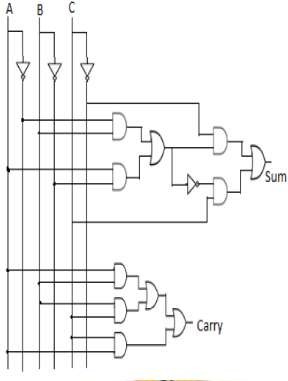
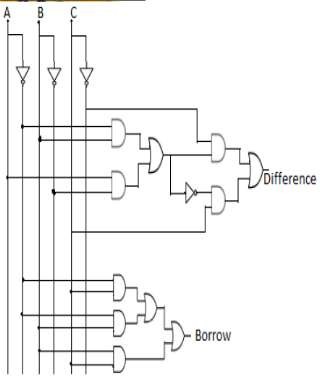
IC 7432 Pin Diagram



Half Adder Circuit Half Subtractor Circuit

Full Adder Circuit Full Subtractor Circuit

## OBSERVATION TABLE:

Truth Table for Half Adder: Truth Table for Half Subtractor:

|  |  |  |  |
| --- | --- | --- | --- |
| INPUT | | OUTPUT | |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| INPUT | | OUTPUT | |
| X | Y | Difference | Barrow |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

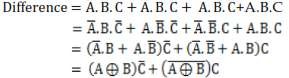
SUM = A̅ ∙ B + A ∙ B̅ CARRY = A ∙ B

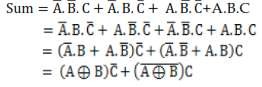
Difference = A̅ ∙ B + A ∙ B̅ Borrow = A̅ ∙ B

Truth Table for Full Adder: Truth Table for Full Subtractor:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INPUT | | | OUTPUT | |
| A | B | Cin | 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 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INPUT | | | OUTPUT | |
| A | B | Bin(C) | Difference | Barrow |
| 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 |



Carry =∑(3, 5, 6)

(after simplifying using K map)

Carry = A. B + B. C + C. A

Borrow =∑(1, 2, 3, 7)

(after simplifying using K map) Borrow= A̅. C + A̅. B + B. C

## SIMULATION:

Half Adder VHDL Code:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity HalfAdder is

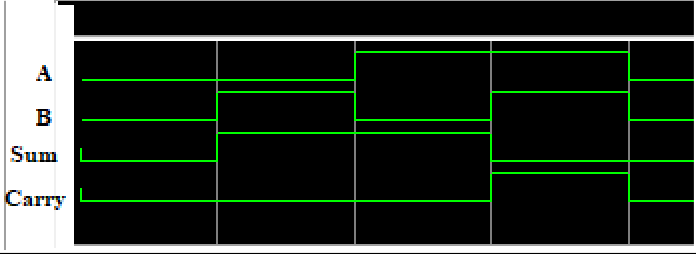
Port ( A, B : in STD\_LOGIC;

Sum, Carry : out STD\_LOGIC); end HalfAdder;

architecture equation of HalfAdder is begin

sum <= ((not A)and B)or(A and(not B)); carry <= A and B;

end equation;

Output Waveform for Half Adder:

Half Subtractor VHDL Code: library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity HalfSub is

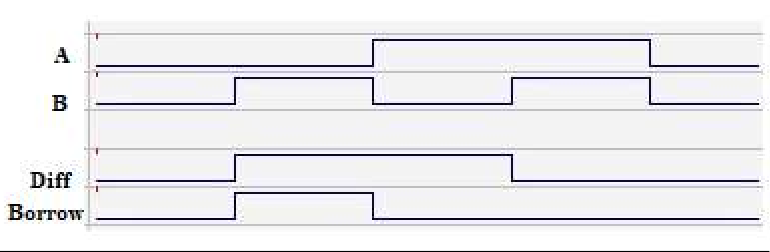
Port ( A,B : in STD\_LOGIC;

Diff,Borrow : out STD\_LOGIC); end HalfSub;

architecture equation of HalfSub is begin

Diff<=((not A)and B)or(A and(not B)); Borrow<=((not A)and B);

end equation;

Output Waveform for Half Subtractor:

Full Adder VHDL Code:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity FullAdder is

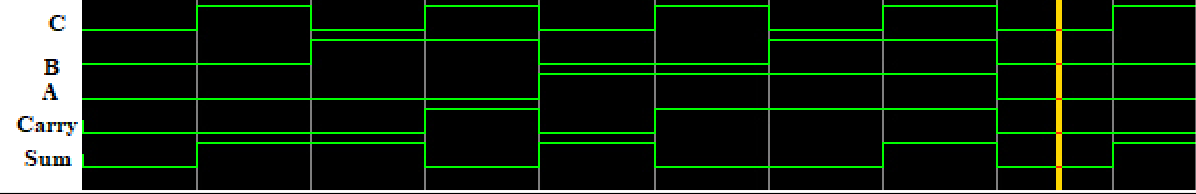
Port ( A,B,C : in STD\_LOGIC;

Sum,Carry : out STD\_LOGIC); end FullAdder;

architecture equation of FullAdder is begin

sum<=(((not A)and(not b)and c)or((not A)and B and(not C))or(a and(not B)and(not C))or(A and B and C));

carry<=(A and B)or(B and C)or(A and C); end equation;

Output Waveform for Full Adder:

Full Subtractor VHDL Code**:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity FullSubtractor is

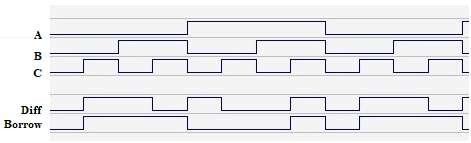
Port ( A,B,C : in STD\_LOGIC;

Diff,Borrow : out STD\_LOGIC); end FullSubtractor;

architecture equation of FullSubtractor is begin

Diff <= (((not A)and(not b)and c)or((not A)and B and(not C))or(a and(not B)and(not C))or(A and B and C));

Borrow <= (A and B)or(B and C)or(A and C); end equation;

Output Waveform for Full Subtractor:

## RESULTS & CONCLUSIONS:

1. The truth table of half adder, half subtractor, full adder and full subtractor is verified.
2. The output waveform of half adder, half subtractor, full adder and full subtractor is simulated and verified.