RNS Institute of Technology Vishnuvardhan Road, Channasandra, Bengaluru – 560098



### Department of Information Science and Engineering

Laboratory Manual On

ANALOG AND DIGITAL ELECTRONICS LABORATORY

[18CSL37]

## By

**Ms. Sowmya S K Mrs. Tejashwini P**

**Assistant Professors, Dept of ISE**

**Software used Analog - Multisim Digital - Xilinx**

**1. ASTABLE MULTIVIBRATOR USING 555 TIMER AIM** : To design and implement an astable multivibrator using 555 Timer for a given frequency and duty cycle.

**COMPONENTS REQUIRED**: 555 Timer IC, Resistors 3.3KΩ , 6.8KΩ, Capacitors of 0.1μF, 0.01 μF, Regulated power supply, CRO

#### THEORY:

Multivibrator is a form of oscillator, which has a non-sinusoidal output. The output waveform is rectangular. The multivibrators are classified as:

i. **Astable or free running multivibrator**: It alternates automatically between two states (low and high for a rectangular output) and remains in each state for a time dependent upon the circuit constants. It is just an oscillator as it requires no external pulse for its operation.

ii. **Monostable or one shot multivibrator**: It has one stable state and one quasi stable. The application of an input pulse triggers the circuit time constants. After a period of time determined by the time constant, the circuit returns to its initial stable state. The process is repeated upon the application of each trigger pulse.

iii. **Bistable Multivibrators**: It has both stable states. It requires the application of an external triggering pulse to change the output from one state to other. After the output has changed its state, it remains in that state until the application of next trigger pulse. Flip flop is an example.

#### CIRCUIT DIAGRAM:

**Fig 1.1: Circuit Diagram and actual connections**

**DESIGN** :

Given frequency (f) = 1KHz and duty cycle = 75% (=0.75) The time period T =1/f = 1ms = tH + tL

Where tH is the time the output is high and tL is the time the output is low.

From the theory of astable multivibrator using 555 Timer(refer Malvino), we have

|  |  |
| --- | --- |
| tL = 0.693 RB C | ------(1) |
| tH = 0.693 (RA + RB)C  T = tH + tL = 0.693 (RA +2 RB) C | ------(2) |

Duty cycle = tH / T = 0.75.

Hence tH = 0.75T = 0.75ms and

tL = T – tH = 1ms-0.75ms=0.25ms.

Let C=0.1μF and substituting in the above equations,

RB = 7.2 KΩ (6.8KΩ) (from equation 1) and

RA = 3.6 KΩ (3.3KΩ) (from equation 2 & RB values).

The Vcc determines the upper and lower threshold voltages (observed from the capacitor voltage waveform) as

VUT=2/3VCC and VLT=1/3VCC

#### PROCEDURE:

1. Before making the connections, check the components using multimeter.
2. Make the connections as shown in figure and switch on the power supply **Fig 1.1**.
3. Observe the capacitor voltage waveform at 6th pin of 555 timer on CRO.
4. Observe the output waveform at 3rd pin of 555 timer on CRO (shown below) **Fig 3.2**.
5. Note down the amplitude levels, time period and hence calculate duty cycle.

#### WAVEFORMS

**Fig 3.2: Waveform of Astable multivibrator 555 timer**

**RESULT**:

tH =\_\_\_\_\_\_(0.73ms)

tL = \_\_\_\_\_\_(0.27ms)

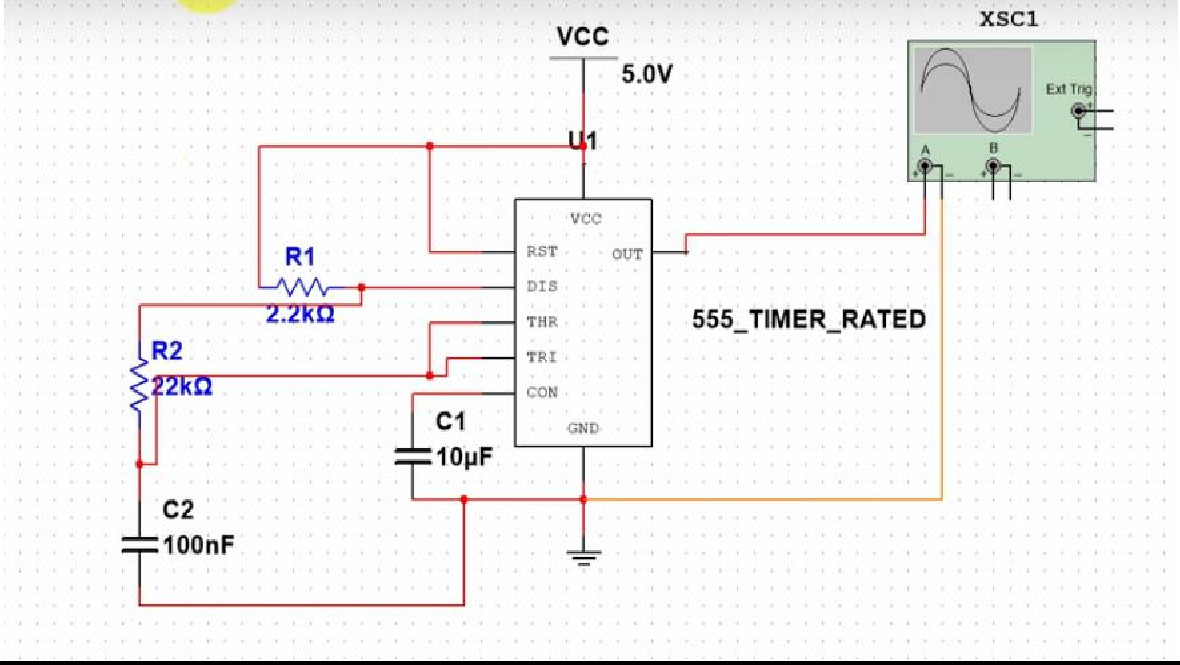
T = \_\_\_\_\_\_\_(1ms)

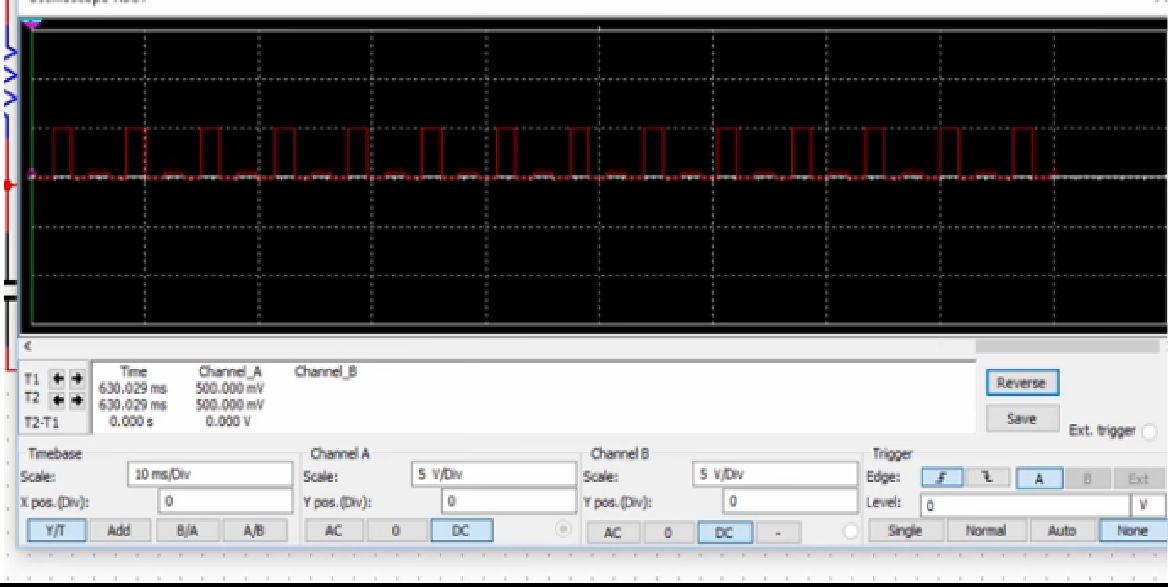
The frequency of the oscillations = 1/T=1/1ms=1Khz.

%Duty cycle (DC) = tH/T\*100 = 0.75/(1ms)\*100 = 75%

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl.no** | **f KHz** | **DUTY CYCLE(%)** | **RA (**KΩ**)** | **RB (**KΩ**)** |
| 1 | 1KHz | 75 % |  |  |
| 2 | 1KHz | 50% |  |  |
| 3 | 1KHz | 25% |  |  |

**1B. 555 TIMER USING MULTISIM**





**2a. OP-AMP AS A RELAXATION OSCILLATOR**

**AIM** : To design and implement a rectangular waveform generator (op-amp relaxation oscillator) for a given frequency.

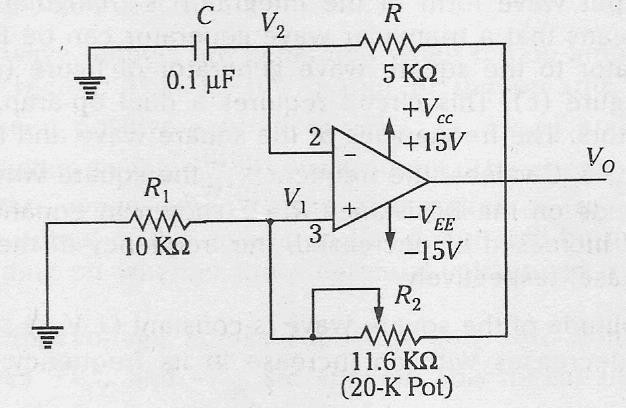
**COMPONENTS REQUIRED**: Op-amp μA 741, Resistor of 1KΩ, 10KΩ, 20 kΩ,

Capacitor of 0.1 μF, Potentiometer, Regulated DC power supply, CRO

#### THEORY

Op-Amp Relaxation Oscillator is a simple Square wave generator which is also called as a Free running oscillator or Astable multivibrator or Relaxation oscillator. In this figure the op-amp operates in the saturation region. Here, a fraction (R2/(R1+R2)) of output is fed back to the noninverting input terminal. Thus reference voltage is (R2/(R1+R2)) Vo and may take values as +(R2/(R1+R2)) Vsat or - (R2/(R1+R2)) Vsat. The output is also fed back to the inverting input terminal after integrating by means of a low-pass RC combination. Thus whenever the voltage at inverting input terminal just exceeds reference voltage, switching takes place resulting in a square wave output.

#### CIRCUIT DIAGRAM



**Fig 2a.1: Circuit Diagram & actual connections**

**DESIGN**

1  ** 

The period of the output rectangular wave is given as *T*  2*RC* ln **** ****(1)

1  **

Where,

**  *R*1 *R*1  *R*2

1  ** 

is the feedback fraction

case i. If R1 = R2, then from equation (1) we have T = 2RC ln(3)

case ii. if R2=1.16 R1, then T = 2RC.........(2)

Example: Design for a frequency of 1 kHz (implies *T*  1

*f*

 1

103

 103  1*ms)*

figure)

Use R2=1.16 R1, for equation (2) to be applied.

Let R1 = 10kΩ, then R2 = 11.6kΩ (use 20kΩ potentiometer as shown in circuit

Choose next a value of C and then calculate value of R from equation (2).

-7 *T*

103

Let C=0.1µF (i.e., 10

), then

*R*  2*C*  2 107

 5*K*

The voltage across the capacitor has a peak voltage of *Vc*

Vsat= 90% \* Vcc

=90/100\* 15=13.5=12V Vc= 10/10+11.6\*12=5V

#### PROCEDURE

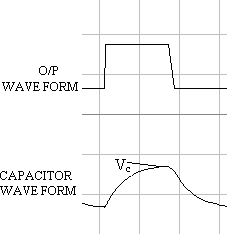
 *R*1

*R*1  *R*2

*Vsat*

1. Before making the connections check all the components using multimeter.
2. Make the connections as shown in **Fig 2a.1** and switch on the power supply.
3. Observe the voltage waveform across the capacitor on CRO.
4. Also observe the output waveform on CRO. Measure its amplitude and frequency. Shown in **Fig 2a.2**

#### WAVEFORMS



**Fig 2a.2: Waveform**

**RESULT:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.no** | **f KHz** | **R** | **Vc** |
| **1** | **1 KHz** | **5kΩ** | **5V** |
| **2** | **1.5 KHz** |  |  |
| **3** | **2 KHz** |  |  |

The frequency of the oscillations = Hz. Vc= Voltage across the capacitors = V

#### 2B. OP-AMP AS A RELAXATION OSCILLATOR

**Aim :** Design and implement a rectangular waveform generator (Op- Amp relaxation oscillator) using a simulation package and demonstrate the change in frequency when all resistor values are doubled.



**R2**

**10kΩ**

**R1 20kΩ**

**Key=A**

**50%**

**XSC1**

**V2**

**12 V**

Ext Trig

+

\_

A

B

+ +

\_ \_

7 1 5 **U1**

3

**C1**

6

2

**100nF**

4

**741**

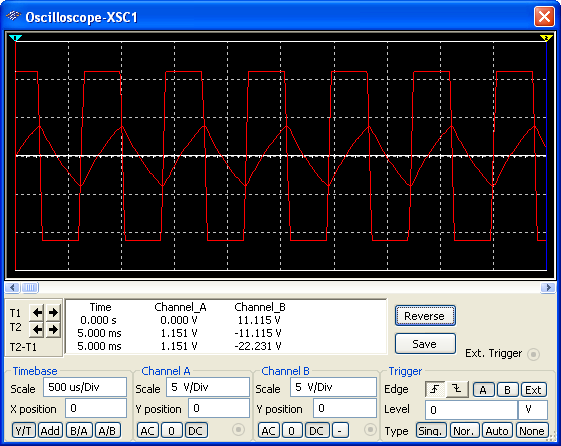
**V1**

**12 V**

**R3**

**5kΩ**

#### Fig 2b.1 Circuit Daigram relaxation Oscillator



**Fig 2b.2 Input and output waveform of relaxation oscillator**

**STUDY OF LOGIC GATES**

**AIM:** To study about logic gates and verify their truth tables

**COMPONENTS REQUIRED:**

|  |  |  |
| --- | --- | --- |
| **SL No.** | **COMPONENT** | **SPECIFICATION** |
| 1. | AND GATE | IC 7408 |
| 2. | OR GATE | IC 7432 |
| 3. | NOT GATE | IC 7404 |
| 4. | NAND GATE 2 I/P | IC 7400 |
| 5. | NOR GATE | IC 7402 |
| 6. | X-OR GATE | IC 7486 |
| 7. | NAND GATE 3 I/P | IC 7410 |
| 8. | IC TRAINER KIT | - |
| 9. | PATCH CORD | - |

**THEORY:**

Circuit that takes the logical decision and the process are called logic gates. Each gate has one or more input and only one output. OR, AND ,NOT are basic gates. NAND, NOR and X-OR are known as universal gates. Basic gates form these gates.

**AND GATE:**

The AND gate performs a logical multiplication commonly known as AND function. The output is high when both the inputs are high. The output is low level when any one of the inputs is low.

**OR GATE:**

The OR gate performs a logical addition commonly known as OR function. The output is high when any one of the inputs is high. The output is low level when both the inputs are low.

**NOT GATE:**

The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high.

**NAND GATE:**

The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the input is low .The output is low level when both inputs are high.

**NOR GATE:**

The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low.

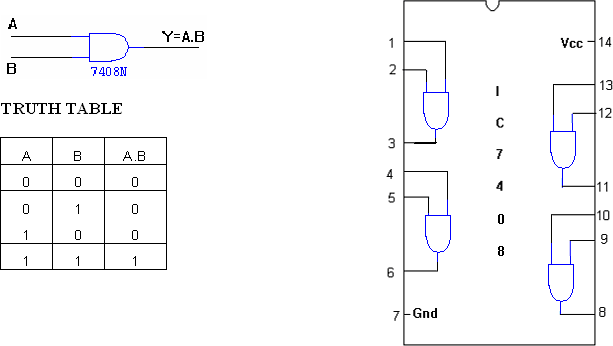
The output is low when one or both inputs are high.

**X-OR GATE:**

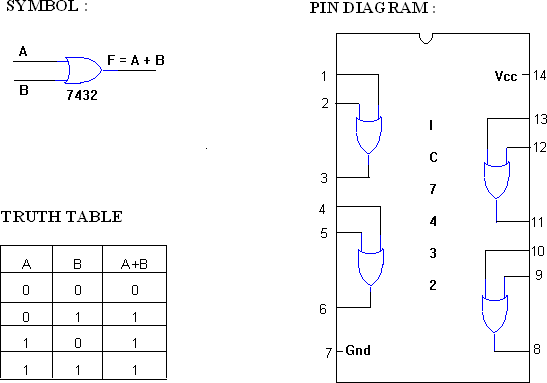
The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

**AND GATE:**

**SYMBOL: PIN DIAGRAM:**

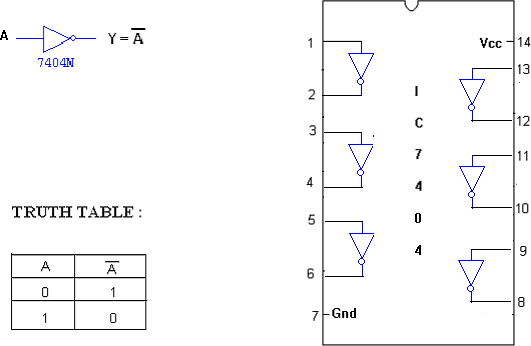


**OR GATE:**



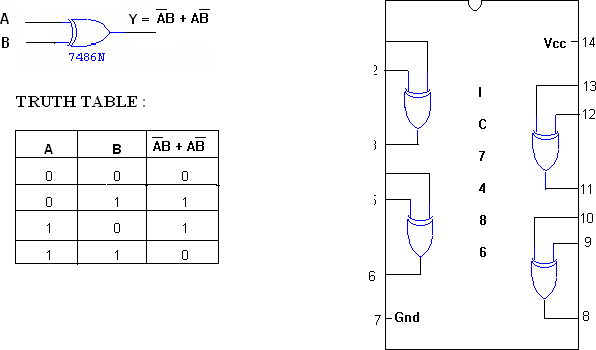
**NOT GATE:**

**SYMBOL: PIN DIAGRAM:**



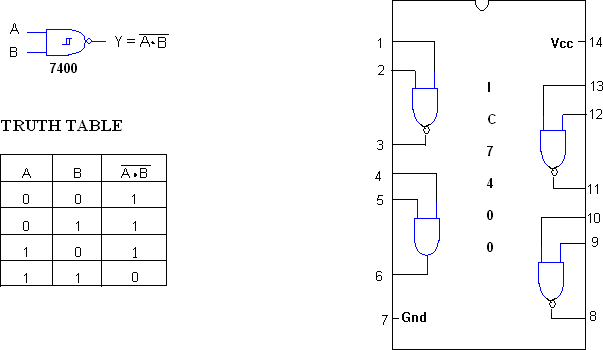
**XOR GATE:**

**SYMBOL : PIN DIAGRAM :**

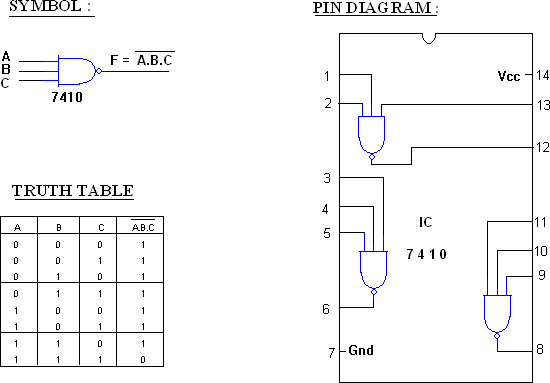


**NAND GATE:**

**SYMBOL: PIN DIAGRAM:**



**3 INPUT NAND GATE:**



#### NOR GATE

**PROCEDURE:**

1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram
3. Observe the output and verify the truth table.

**RESULT: All basic gates are verified**

**4. Design and implement Half adder, Full Adder, Half Subtractor, Full Subtractor using basic gates. And implement the same in HDL.**

**AIM:** To realize

1. Half Adder and Full Adder
2. Half Subtractor and Full Subtractor by using Basic gates.

**COMPONENTS REQUIRED:** IC 7400, IC 7408, IC 7486, IC 7432, Patch Cords & IC Trainer

Kit.

#### THEORY:

***Half-Adder:*** A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C. The Boolean functions describing the half-adder are:

**S =A  B C = A B**

***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. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit, Cin , is called a full-adder. The Boolean functions describing the full-adder are:

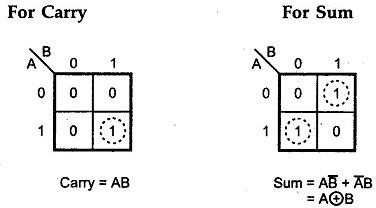
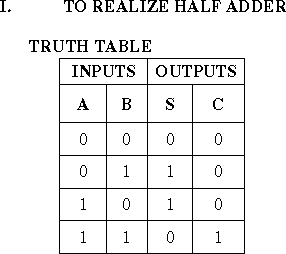
**S = (A  B)  Cin C = AB + Cin (A  B)**

***Half Subtractor:*** Subtracting a single-bit binary value B from another A (i.e. A -B ) produces a difference bit D and a borrow out bit B-out. This operation is called half subtraction and the circuit to realize it is called a half subtractor. The Boolean functions describing the half- Subtractor are:

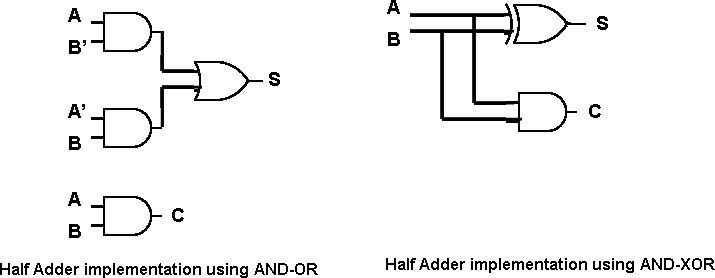
**S =A  B C = A’ B**

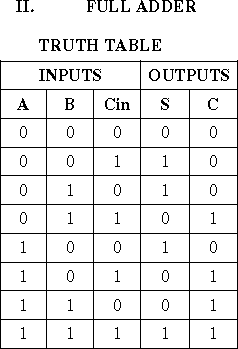
***Full Subtractor:*** Subtracting two single-bit binary values, B, Cin from a single-bit value A produces a difference bit D and a borrow out Br bit. This is called full subtraction. The Boolean functions describing the full-subtracter are:

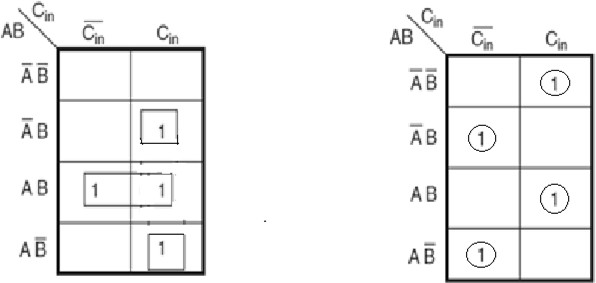
**D = (A  B)  Cin Br= A’B + A’(Cin) + B(Cin)**



**K-maps**



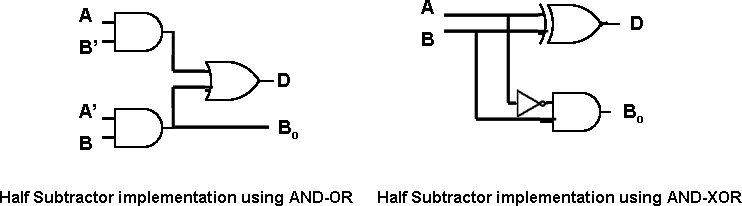
**k-maps**

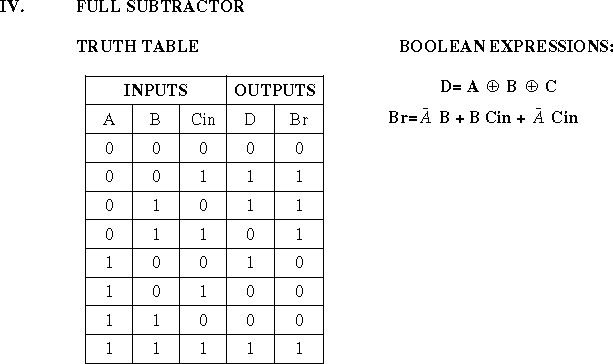


**C= AB+ABCin + ABCin S = A  B  Cin C = AB + Cin(AB)**

Draw using basic gates….

## K-maps







**D =A  B  Cin Br =AB+ABCin + ABCin**

**AB+ Cin (AB)**

Draw using basic gates………

**VHDL**

**Procedure:**

1. Click on Xilinx ise 7.1i.
2. Go to file and if any project is opened close that.
3. File🡪New project→project name(multix)
4. Device family(spartan2),Generated simulation language(VHDL),Modelsim(ISE simulator)
5. Click next till finish.
6. Select multix.ise file→right click→New source→VHDL module→File name→next
7. Give port names,input variables and output variables→next→finish.
8. Write VHDL code and save it.
9. Goto process view→synthesize-XST(double click).If any error clear it.
10. Select filename.behavioral→right click→new source→Testbenchwaveforms Next→next→finish.
11. Select combinatorial for multiplexer and single clock for other experiments, click OK.
12. Give the input waveform and save.
13. Select filename.tbw→click process view→double click simulate behavioral model.
14. The output waveform is shown.

**AIM:** Design and develop the VHDL code for **Half adder, Full Adder, Half Subtractor, Full Subtractor**. Simulate and verify its working.

#### CODE:Half adder

library IEEE;

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

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity adr is

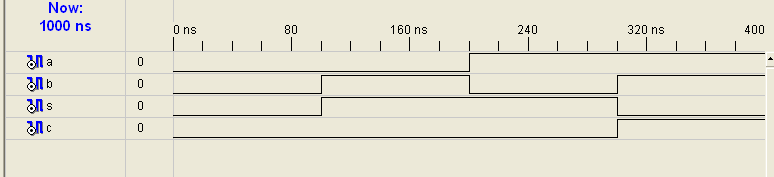
Port ( A,B: in std\_logic; sum,carry : out std\_logic;

end adr;

architecture Behavioral of adr is begin

sum <= A xor B; carry <= A and B;

end Behavioral;



#### CODE: Full adder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity adr is

port ( a,b,c: in std\_logic; sum,carry : out std\_logic;

end adr;

architecture behavioral of adr is begin

sum <= a xor b xor c;

carry <= (a and b) or (c and (a xor b));

end behavioral;

#### CODE: Half subtractor

library IEEE;

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

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity adr is

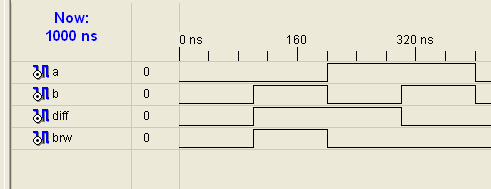
port ( a,b: in std\_logic; diff,brw : out std\_logic;

end adr;

architecture behavioral of adr is begin

diff <= a xor b

brw <= (not a) and b; end behavioral;



#### CODE: Full subtractor

library ieee;

use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity adr is

port ( a,b: in std\_logic; sum,carry : out std\_logic;

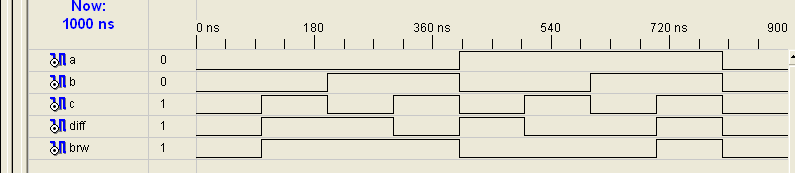
end adr;

architecture behavioral of adr is begin

diff <=a xor b xor c

brw <= ((not A) and B) or (c and not(a xor b));

end behavioral;



|  |  |
| --- | --- |
| Half Adder | sum <= a xor b; carry <= a and b; |
| Half Subtractor | diff <= a xor b  brw <= (not a) and b; |
| Full Adder | sum <= a xor b xor c;  carry <= (a and b) or (c and (a xor b)); |
| Full Subtractor | diff <= a xor b xor c  brw <= ((not A) and B) or (c and not(a xor b)); |

**Result:** Verified Half adder, Full adder, Half subtractor and Full subtarctor using Basic gates and also simulated using Xilinx.

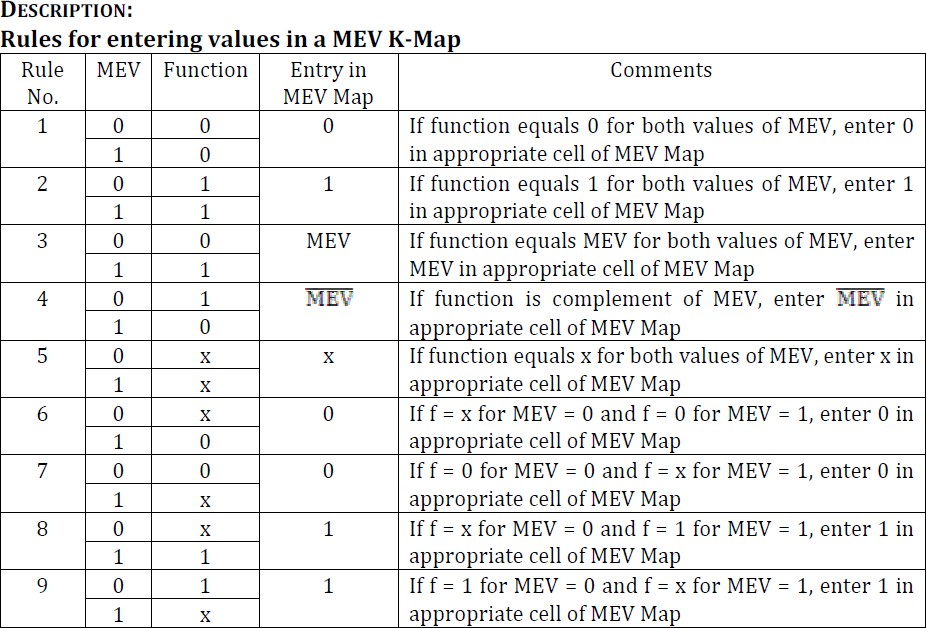
## 5 a. Given a 4-variable logic expression, simplify it using appropriate Technique and realize the simplified logic expression using 8:1 multiplexer IC. And implement the same in HDL.

**AIM:** To simplify 4 variables Boolean expression using Entered Variable Map method and realize the simplified logic using 8:1 MUX

**COMPONENTS REQUIRED:** IC74151,patch chords, trainer kit

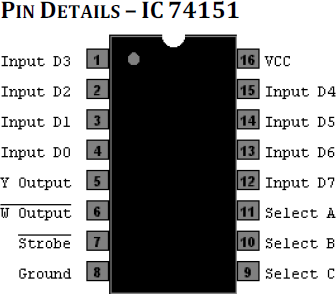
#### THEORY:

Multiplexer sometimes is called universal logic circuit because a 2n to 1multiplexer can be used as a design solution for any n variable truth table. Let’s consider A Band C variables to be fed as select inputs, the fourth variable D as data input. We write all the combinations of 3 select inputs in first row along different columns. Now corresponding to each value of 4th variable D Truth table output Y is written. The 4th column Y as a function of D.

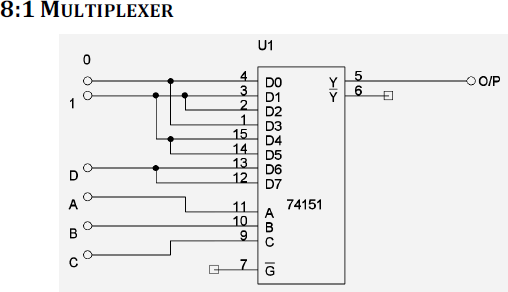


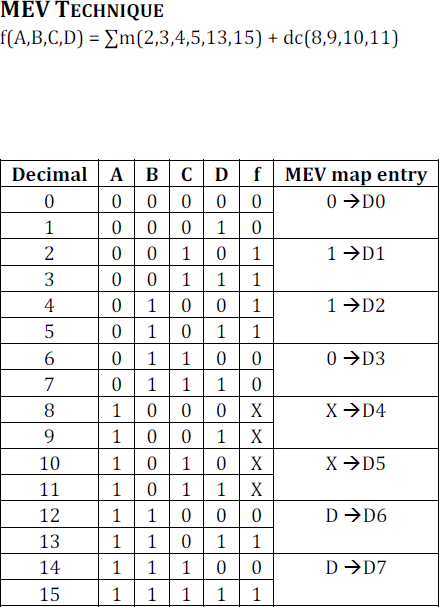
#### PROCEDURE:

1. Verify all the components and patch chords whether they are in good condition or not.
2. Make connections as shown in the circuit diagram (fig 1.1).
3. Give power supply to the trainer kit.
4. Provide input data to the circuit via switches.
5. Record and verify the output sequence for each combination of the select lines.



#### CIRCUIT DIAGRAM:





**5B. 8:1 MULTIPLEXER**

**AIM:** Design and develop the Verilog /VHDL code for an 8:1 multiplexer. Simulate And verify its working.

#### CODE:

library IEEE;

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

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity rr is

Port ( i : in std\_logic\_vector(7 downto 0); sel : in std\_logic\_vector(2 downto 0);

z : out std\_logic);

end rr;

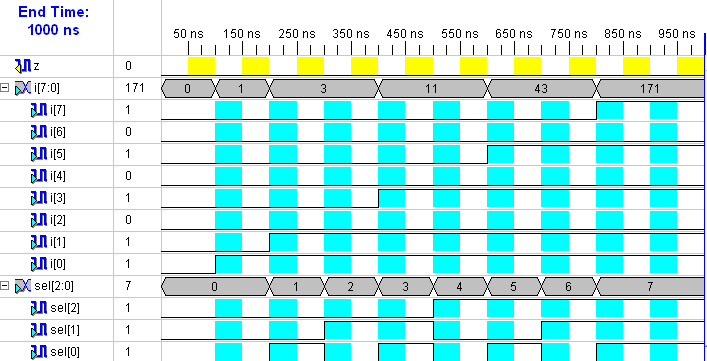
architecture Behavioral of rr is begin

z <= i(0) when sel="000" else i(1) when sel="001" else

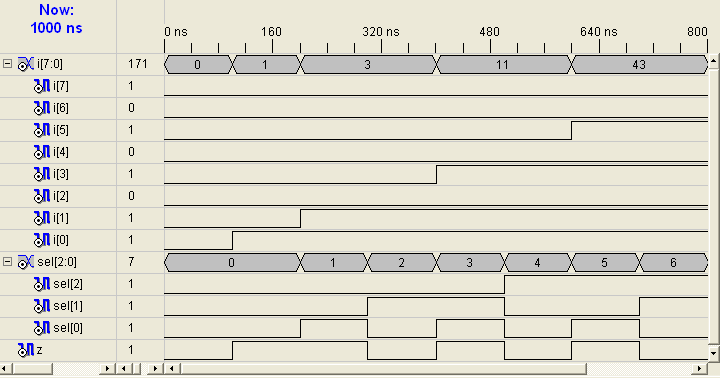
i(2) when sel="010" else i(3) when sel="011" else i(4) when sel="100" else i(5) when sel="101" else i(6) when sel="110" else i(7);

end Behavioral;

**Input Waveform**

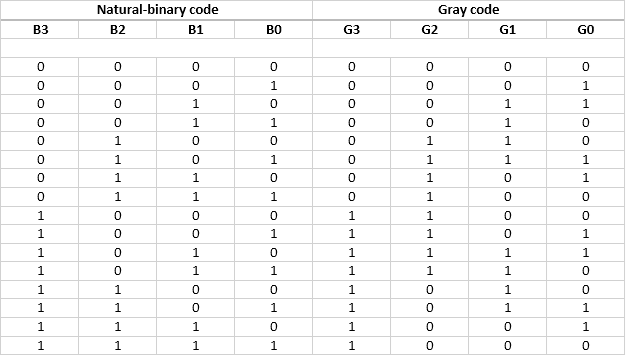


**Output Waveform**



**7. Design and implement code converter I) Binary to Gray II) Gray to Binary Code using basic gates.**

**AIM:** To realize Binary to Gray code converter and vic

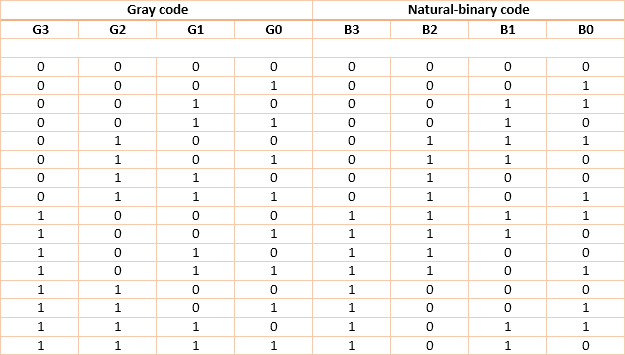


For Kmap and circuit refer observation

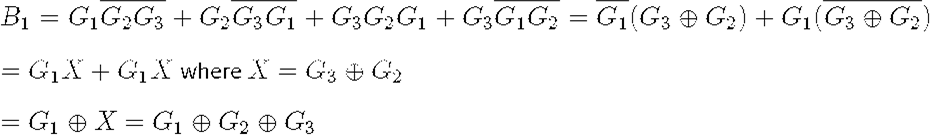
**G3 = B3 G2=B3 B2 + B3B2 G2 = B3  B2**

**G1 =B2 B1+ B2B1 G0 =B1B0 + B1B0 G1= B2  B1 G0 = B1  B0**

**II) GRAY TO BINARY CONVERSION**



**B3 = G3**

**B2 = G3G2 + G2G = G3  G2**

**B0 = G3  G2  G1  G0**

**PROCEDURE:**

Check all the components for their working.

Insert the appropriate IC into the IC base.

Make connections as shown in the circuit diagram.

Verify the Truth Table and observe the outputs.

**RESULT:** Binary to gray code conversion and vice versa is realized using basic gates and truth tables are verified