

# LABORATORY MANUAL

**18CSL37-ANALOG AND DIGITAL ELECTRONICS LABORATORY**

2019-20



DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING  
ATRIA INSTITUTE OF TECHNOLOGY  
Adjacent to Bangalore Baptist Hospital  
Hebbal, Bengaluru-560024

<b>ANALOG AND DIGITAL ELECTRONICS LABORATORY</b> (Effective from the academic year 2018 -2019) <b>SEMESTER – III</b>			
<b>Course Code</b>	18CSL37	<b>CIE Marks</b>	40
<b>Number of Contact Hours/Week</b>	0:2:2	<b>SEE Marks</b>	60
<b>Total Number of Lab Contact Hours</b>	36	<b>Exam Hours</b>	3 Hrs
<b>Credits – 2</b>			
<b>Course Learning Objectives:</b> This course (18CSL37) will enable students to:			
This laboratory course enable students to get practical experience in design, assembly and evaluation/testing of <ul style="list-style-type: none"> <li>• Analog components and circuits including Operational Amplifier, Timer, etc.</li> <li>• Combinational logic circuits.</li> <li>• Flip - Flops and their operations</li> <li>• Counters and registers using flip-flops.</li> <li>• Synchronous and Asynchronous sequential circuits.</li> <li>• A/D and D/A converters</li> </ul>			
<b>Descriptions (if any):</b>			
<ul style="list-style-type: none"> <li>• Simulation packages preferred: Multisim, Modelsim, PSpice or any other relevant.</li> <li>• For Part A (Analog Electronic Circuits) students must trace the wave form on Tracing sheet / Graph sheet and label trace.</li> <li>• Continuous evaluation by the faculty must be carried by including performance of a student in both hardware implementation and simulation (if any) for the given circuit.</li> <li>• A batch not exceeding 4 must be formed for conducting the experiment. For simulation individual student must execute the program.</li> </ul>			
<b>Laboratory Programs:</b>			
<b>PART A (Analog Electronic Circuits)</b>			
1.	Design an astable multivibrator circuit for three cases of duty cycle (50%, <50% and >50%) using NE 555 timer IC. Simulate the same for any one duty cycle.		
2.	Using ua 741 Opamp, design a 1 kHz Relaxation Oscillator with 50% duty cycle. And simulate the same.		
3.	Using ua 741 opamp, design a window comparator for any given UTP and LTP. And simulate the same.		
<b>PART B (Digital Electronic Circuits)</b>			
4.	Design and implement Half adder, Full Adder, Half Subtractor, Full Subtractor using basic gates. And implement the same in HDL.		
5.	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.		
6.	Realize a J-K Master / Slave Flip-Flop using NAND gates and verify its truth table. And implement the same in HDL.		
7.	Design and implement code converter I) Binary to Gray (II) Gray to Binary Code using basic gates.		
8.	Design and implement a mod-n ( $n < 8$ ) synchronous up counter using J-K Flip-Flop ICs and demonstrate its working.		
9.	Design and implement an asynchronous counter using decade counter IC to count up from 0 to n ( $n \leq 9$ ) and demonstrate on 7-segment display (using IC-7447)		
<b>Laboratory Outcomes:</b> The student should be able to:			
<ul style="list-style-type: none"> <li>• Use appropriate design equations / methods to design the given circuit.</li> </ul>			

- Examine and verify the design of both analog and digital circuits using simulators.
- Make use of electronic components, ICs, instruments and tools for design and testing of circuits for the given the appropriate inputs.
- Compile a laboratory journal which includes; aim, tool/instruments/software/components used, design equations used and designs, schematics, program listing, procedure followed, relevant theory, results as graphs and tables, interpreting and concluding the findings.

**Conduct of Practical Examination:**

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (*Courseed to change in accordance with university regulations*)
  - a) For laboratories having only one part – Procedure + Execution + Viva-Voce:  $15+70+15 = 100$  Marks
  - b) For laboratories having PART A and PART B
    - i. Part A – Procedure + Execution + Viva =  $6 + 28 + 6 = 40$  Marks
    - ii. Part B – Procedure + Execution + Viva =  $9 + 42 + 9 = 60$  Marks

**CONTENTS**

<b>Sl. No.</b>	<b>Name of Experiments</b>	<b>Page No.</b>
<b>PART A (Analog Electronic Circuits)</b>		
<b>1</b>	Astable Multivibrator (Hardware and Software)	<b>1</b>
<b>2</b>	Relaxation Oscillator (Hardware and Software)	<b>5</b>
<b>3</b>	Window Comparator (Hardware and Software)	<b>9</b>
<b>PART B (Digital Electronic Circuits)</b>		
<b>4</b>	Half adder, Full Adder, Half Subtractor, Full Subtractor using basic Gates (Hardware and Software)	<b>15</b>
<b>5</b>	8:1 multiplexer IC (Hardware and Software)	<b>23</b>
<b>6</b>	J-K Master / Slave Flip-Flop using NAND gates (Hardware and Software)	<b>27</b>
<b>7</b>	Code converter I) Binary to Gray (II) Gray to Binary Code using basic gates	<b>30</b>
<b>8</b>	mod-n ( $n < 8$ ) synchronous up counter using J-K Flip-Flop ICs	<b>36</b>
<b>9</b>	asynchronous counter using decade counter IC to count up from 0 to n ( $n \leq 9$ ) and demonstrate on 7-segment display (using IC-7447)	<b>42</b>
<b>10</b>	User Manual	<b>45</b>
<b>11</b>	Sample Viva Questions	<b>65</b>

**PART A (Analog Electronic Circuits)**

**Experiment No.1: Design an astable multivibrator circuit for three cases of duty cycle (50%, <50% and >50%) using NE 555 timer IC. Simulate the same for any one duty cycle.**

**a. Hardware Part:****Description:**

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 multivibrators:** It has one stable state and one quasi stable. The application of an input pulse triggers the circuit time constants and the output goes to the quasi stable state, 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.

**Components Required:**

555 Timer IC, Resistors of 3.3K $\Omega$ , 6.8K $\Omega$ , Capacitors of 0.1  $\mu$ F, 0.01  $\mu$ F, digital trainer kit(used to give +5v power supply to 555 IC),CRO.

**Design:****Case 1: Given Duty cycle =50%, Frequency, f=1kHz**

For an Astable multivibrator using 555 Timer we have

$$t_L = 0.693 R_B C \text{ ----- (1)}$$

$$t_H = 0.693 (R_A + R_B)C \text{ ----- (2)}$$

$$T = t_H + t_L = 0.693 (R_A + 2 R_B) C$$

Where  $t_H$  is the time the output is high and  $t_L$  is the time the output is low

Since Frequency,  $f=1\text{kHz}$ ,

$$T = 1/f = 1\text{ms}$$

$$\text{Duty cycle} = t_H / T = 0.5.$$

Hence  $t_H = 0.5T = 0.5\text{ms}$  and

$T = t_H + t_L$  (Where  $t_H$  is the time the output is high and  $t_L$  is the time the output is low)

$$t_L = T - t_H = 0.5\text{ms}.$$

Let  $C=0.1\mu\text{F}$  and substituting in the above equations, So,

Therefore,  $R_B = t_L / 0.693 \times C$

$$= 0.5 \times 10^{-3} / 0.693 \times 0.1 \times 10^{-6}$$

$$R_B = 7.2\text{k}\Omega$$

$$R_A = (t_H - 0.693 \times R_B \times C) / 0.693 \times C$$

$$= (0.5 \times 10^{-3} - 0.693 \times 7.2 \times 10^3 \times 0.1 \times 10^{-6}) / (0.693 \times 0.1 \times 10^{-6})$$

$$R_A = 0.015 \times 10^3 = 15\Omega$$

Use  $R_A = 7.2\text{k}\Omega$  and

$$R_B = 15\Omega.$$

**Note:**

1. The duty cycle determined by  $R_A$  &  $R_B$  can vary only between 50 & 100%. If  $R_A$  is much smaller than  $R_B$ , the duty cycle approaches 50%.
2.  $V_{UT} = (2/3 \times V_{CC})$
3.  $V_{LT} = (1/3 \times V_{CC})$

**Case 2: Given Frequency,  $f=1\text{kHz}$ , Duty Cycle=60% (>50%)**

$$T = 1/f = 1\text{ms},$$

$$\text{Duty cycle} = 60\% (=0.6)$$

$$\text{Time period } T = 1/f = 1\text{ms} = t_H + t_L$$

For an Astable multivibrator using 555 Timer we have

$$t_L = 0.693 R_B C \text{-----} (1)$$

$$t_H = 0.693 (R_A + R_B) C \text{-----} (2)$$

$$T = t_H + t_L = 0.693 (R_A + 2 R_B) C \text{-----} (3)$$

$$\text{Duty cycle} = t_H / T = 0.6.$$

$$\text{Hence } t_H = 0.6 T = 0.6\text{ms and}$$

$$t_L = T - t_H = 0.4\text{ms}.$$

Let  $C=0.1\mu\text{F}$  and substituting in the above equations,

$$\text{So } R_B = t_L / 0.693 \times C$$

$$= 0.4 \times 10^{-3} / 0.693 \times 0.1 \times 10^{-6}$$

$$R_B = 5.8\text{k}\Omega$$

$$R_A = t_H - 0.693 \times C \times R_B / 0.693 \times C$$

$$= ((0.6 \times 10^{-3}) - (0.693 \times 5.8 \times 10^3 \times 0.1 \times 10^{-6})) / (0.693 \times 0.1 \times 10^{-6})$$

$$R_A = 2.9\text{k}\Omega$$

Use  $R_A = 3.3\text{k}\Omega$  and  $R_B = 5.6\text{k}\Omega$

The  $V_{CC}$  determines the upper and lower threshold voltages (observed from the capacitor voltage waveform) as  $V_{UT}$  and  $V_{LT}$

**Case 3: Given Duty cycle =40% (<50%), and Frequency (f)=1kHz**

For an Astable multivibrator using 555 Timer we have

$$t_L = 0.693 R_B C \text{-----} (1)$$

$$t_H = 0.693 (R_A + R_B) C \text{-----} (2)$$

$$T = t_H + t_L = 0.693 (R_A + 2 R_B) C$$

Where  $t_H$  is the time the output is high and  $t_L$  is the time the output is low

Since Frequency (f) =1kHz,

$$T = 1/f = 1\text{ms}$$

$$\text{Duty cycle} = t_H / T = 0.4$$

$$\text{Hence } t_H = 0.4T = 0.4\text{ms and}$$

$$T = t_H + t_L$$

$$t_L = T - t_H = 0.6\text{ms}.$$

Let  $C=0.1\mu\text{F}$  and substituting in the above equations, So,

$$\text{Therefore, } R_B = t_L / 0.693 \times C$$

$$= 0.6 \times 10^{-3} / 0.693 \times 0.1 \times 10^{-6}$$

$$R_B = 8.6\text{k}\Omega$$

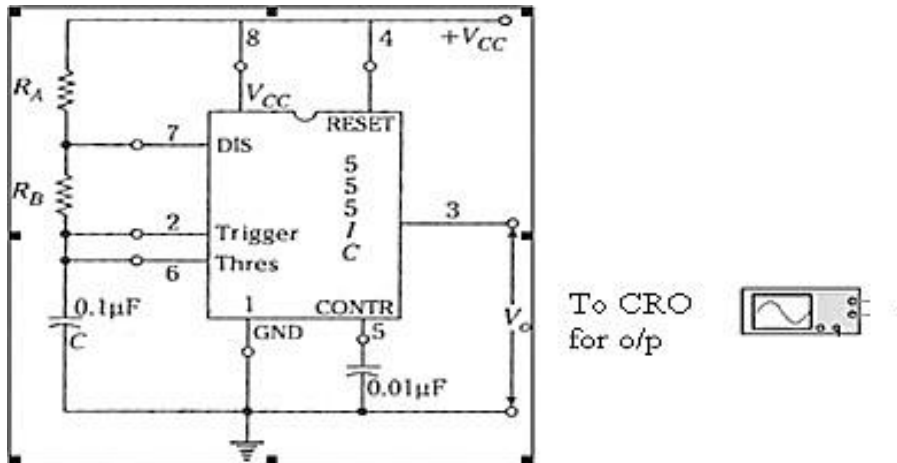
$$R_A = (t_H - 0.693 \times R_B \times C) / 0.693 \times C$$

$$= (0.4 \times 10^{-3} - 0.693 \times 8.6 \times 10^3 \times 0.1 \times 10^{-6}) / (0.693 \times 0.1 \times 10^{-6})$$

$R_A = -2.82 \text{ k}\Omega$  (As we have got **-ve sign**,  $R_A$  can not be used hence we make the following statement)

**The duty cycle determined by  $R_A$  &  $R_B$  can vary only between 50 & 100%. If  $R_A$  is much smaller than  $R_B$ , the duty cycle approaches 50%.**

**Circuit Diagram:**



Connect the pin 2 to the CRO to get the capacitor waveform check the amplitude from the waveform to get the UTP and LTP values.

Connect pin 3 to CRO to get the output. Find out the  $T_H$  and  $T_L$  values.

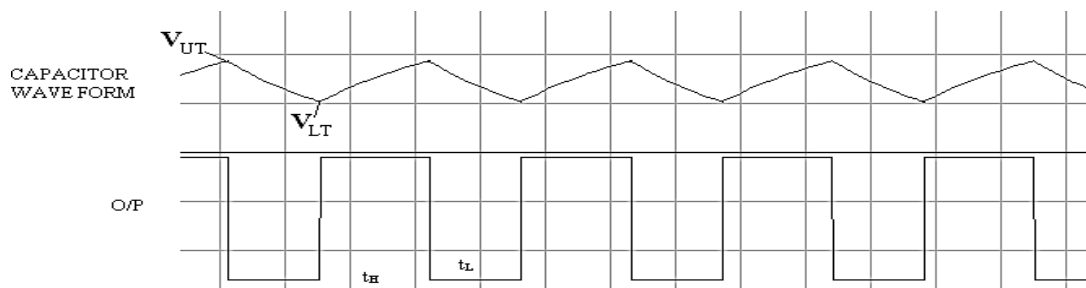
**Procedure:**

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

**Result:**

The frequency of the oscillations = 1KHz.

**Waveforms**



**Result:**

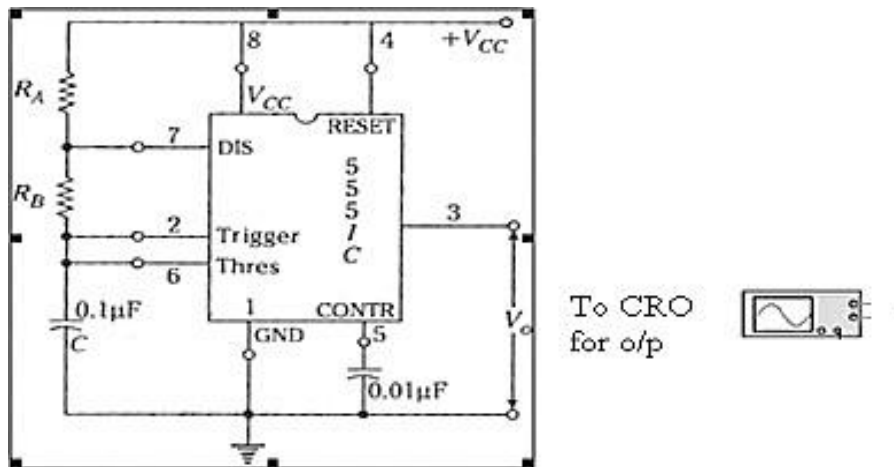
Threshold voltage	$V_{UT}$	$V_{LT}$
theoretical	$(2/3 \times V_{CC})=3.3V$	$(1/3 \times V_{CC})=1.6V$
Practical		

**Note:**

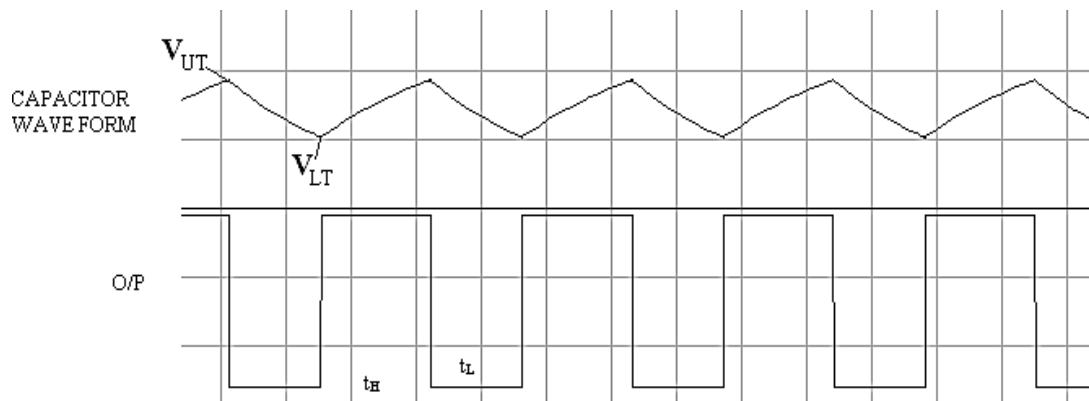
Each division in oscilloscope is 0.2  
 Time=no of div in x-axis x time base  
 Amplitude= no of div in y-axis x volt/div  
 Duty cycle=  $(T_{on}/T_{on} + T_{off}) * 100$

Duty cycle	Duty cycle	Ton (t	Toff
Theoretical	50%	0.5ms	0.5ms
Practical			
Theoretical	60%	0.6ms	0.4ms
Practical			
Theoretical	40%	0.4ms	0.6ms
Practical			

**b. Software Part:**



**Type of analysis:** TIME DOMAIN (TRANSIENT)      **Run to time:** 100msec  
**step size:** 0.1msec





**Experiment No.2: Using ua 741 Opamp, design a 1 kHz Relaxation Oscillator with 50% duty cycle. And simulate the same.**

**a. Hardware Part:**

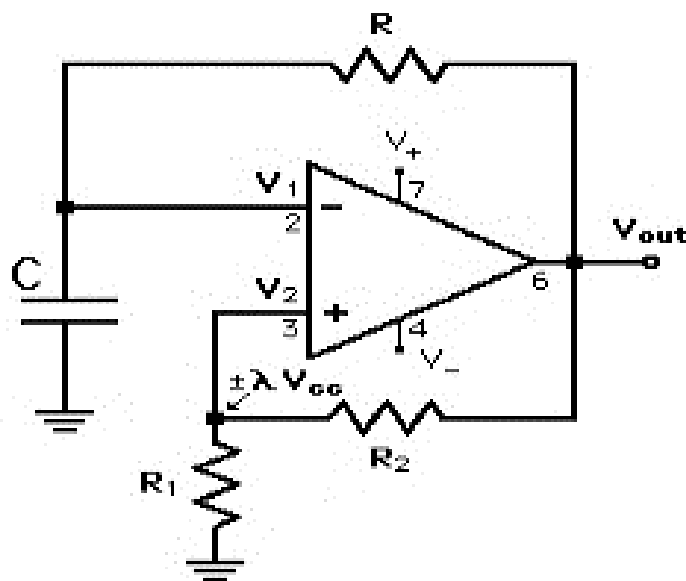
**Description:**

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  $(R_1 / (R_1 + R_2))$  of output is fed back to the noninverting input terminal. Thus reference voltage is  $(R_1 / (R_1 + R_2)) V_o$ . And may take values as  $+(R_1 / (R_1 + R_2)) V_{sat}$  or  $-(R_1 / (R_1 + R_2)) V_{sat}$ . 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.

**Components Required:**

Op-amp  $\mu A 741$ , Resistor of  $10K\Omega, 5.1K\Omega$ , Capacitor of  $0.1 \mu F$ , digital trainer kit (+12v & -12v is given to Op amp from this), CRO.

**Circuit Diagram**



**Design:**

The period of the output rectangular wave is given as

$$T = 2RC \ln \left( \frac{1 + \beta}{1 - \beta} \right) \text{ ----- (1)}$$

Where,

$\beta = R_1 / (R_1 + R_2)$  is the feedback fraction

If  $R_1 = R_2$ , then from equation (1), we have

$$T = 2RC \ln(3) \text{ ----- (2)}$$

Design for a frequency of 1 kHz (implies  $T = 1 \text{ ms}$ )

Let  $C = 0.1 \mu F$

Then calculating R as

$$R = T/2 C \ln(3)$$

$$= 1 \times 10^{-3} / 2 \times 0.1 \times 10^{-6} \times 1.099$$

$$= 5 \times 10^3$$

$$R = 5 \text{K}\Omega$$

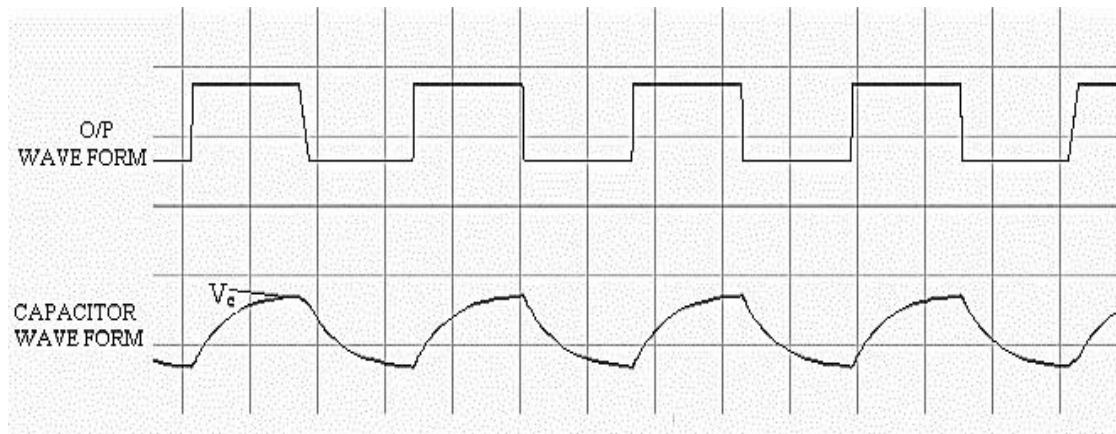
Use  $R = 5.1 \text{K}\Omega$

The voltage across the capacitor has a peak voltage of  $V_c = (R_1/R_1 + R_2) V_{sat}$

### Procedure:

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

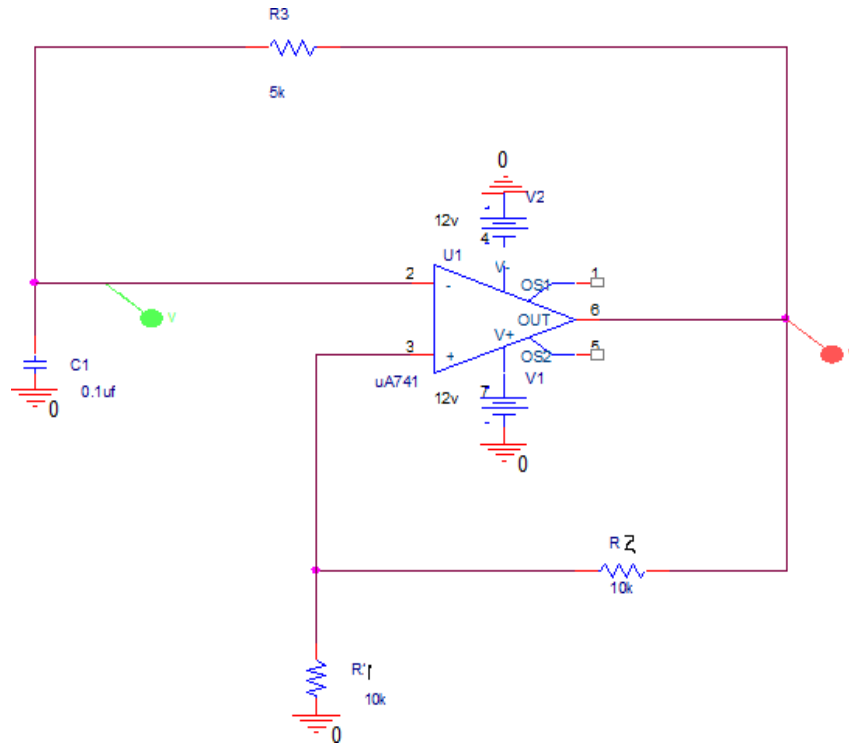
### Waveforms



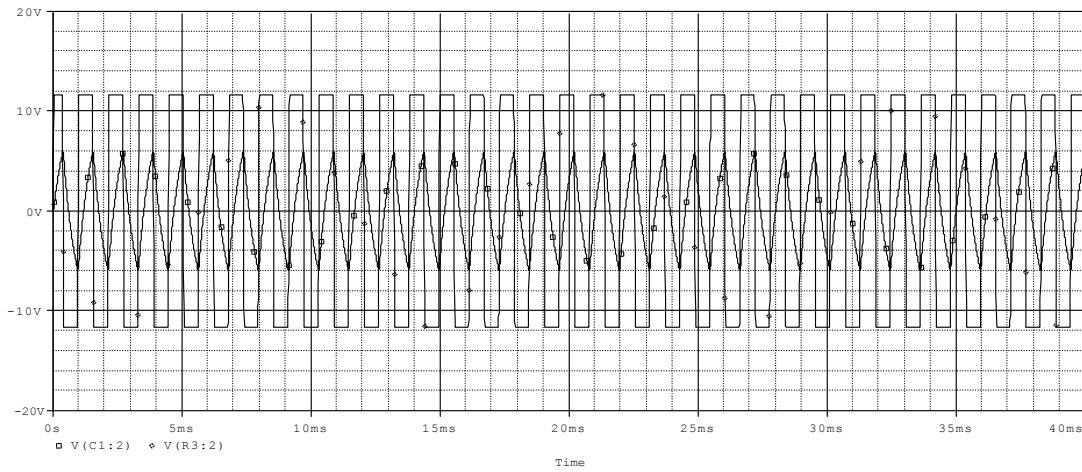
### Result:

The frequency of the oscillations = \_\_\_\_ Hz.

**b. Software Part**



**Waveforms from simulation  $T= 1ms$   $f=1kHz$**

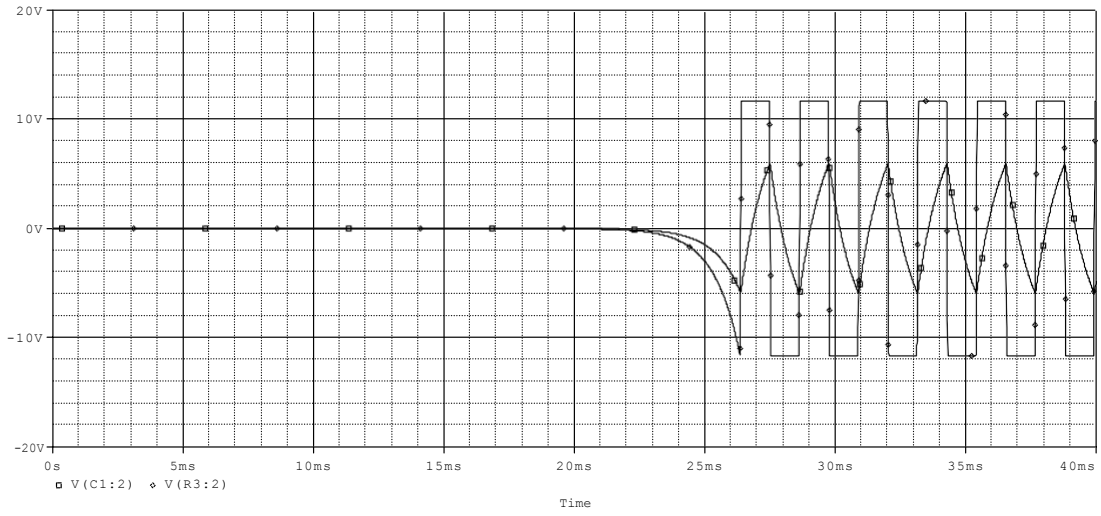


**Type of analysis: TIME DOMAIN (TRANSIENT)**

**Run to time: 10ms**

**Step size: 0.01ms**

Waveforms with resistor values doubled



$T=2ms = \text{frequency} = 500\text{hz}$

**Experiment No.3: Using ua 741 opamp, design a window comparator for any given UTP and LTP. And simulate the same.**

**a. Hardware Part**

**Theory:**

When  $V_{IN}$  is below the lower voltage level,  $V_{REF}(LOWER)$  which equates to  $1/3V_{cc}$ , the output will be LOW. When  $V_{IN}$  exceeds this  $1/3V_{cc}$  lower voltage level, the first op-amp comparator detects this and switches the output HIGH to  $V_{cc}$ .

As  $V_{IN}$  continues to increase it passes the upper voltage level,  $V_{REF}(UPPER)$  at  $2/3V_{cc}$  and the second op-amp comparator detects this and switches the output back LOW. Then the difference between  $V_{REF}(UPPER)$  and  $V_{REF}(LOWER)$  (which is  $2/3V_{cc} - 1/3V_{cc}$  in this example) creates the switching window for the positive going signal.

Lets now assume that  $V_{IN}$  is at its maximum value and equal to  $V_{cc}$ . As  $V_{IN}$  decreases it passes the upper voltage level  $V_{REF}(UPPER)$  of the second op-amp comparator which switches the output HIGH. As  $V_{IN}$  continues to decrease it passes the lower voltage level,  $V_{REF}(LOWER)$  of the first op-amp comparator once again switching the output LOW.

Then the difference between  $V_{REF}(UPPER)$  and  $V_{REF}(LOWER)$  creates the window for the negative going signal. So we can see that as  $V_{IN}$  passes above or passes below the upper and lower reference levels set by the two op-amp comparators, the output signal  $V_{OUT}$  will be HIGH or LOW.

In this simple example we have set the upper trip level at  $2/3V_{cc}$  and the lower trip level at  $1/3V_{cc}$  (because we used three equal value resistors), but can be any values we choose by adjusting the input thresholds. As a result, the window width can be customized for a given application.

If we used a dual power supply and set the upper and lower trip levels to say  $\pm 10$  volts and  $V_{IN}$  was a sinusoidal waveform, then we could use this window comparator circuit as a zero crossing detector of the sine wave which would produce an output, HIGH or LOW every time the sine wave crossed the zero volts line from positive to negative or negative to positive.

**Components Required:** Two LM741 Op-amp Chip, Three  $10K\Omega$  Resistor,  $5.1K\Omega$  Resistor (Optional: Two 1N4006 Diodes, 4049 inverter chip, LED)

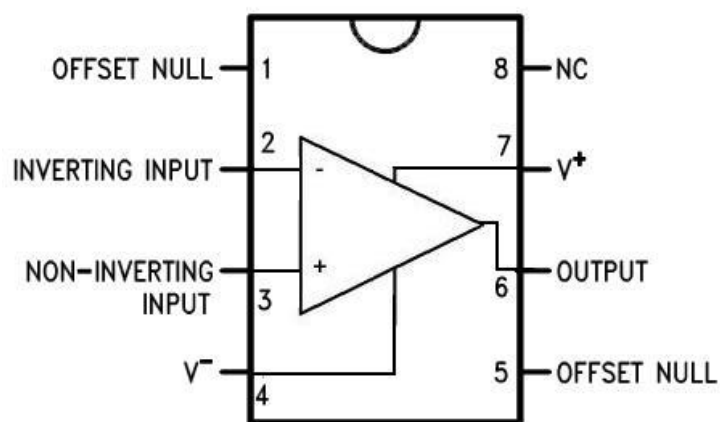
**Note:**

1. The LM741 is a general-purpose operational amplifier IC.
2. The LM741 is an 8-pin chip.
3. As a quick runthrough, we will not be using pins 1, 5, and 8 on this chip.
4. Pin 2 is the inverting terminal and pin 3 is the noninverting terminal. These are the input terminals of the chip.

5. Pins 4 and 7 are the power pins of the LM741 in order to power it on. Pin 4 is  $V^-$  and pin 7 is  $V^+$ . We connect pin 7 to positive voltage and pin 4 to either ground or negative voltage. In this circuit, we connect it to negative voltage.
6. And, lastly, pin 6 is the output. This is the pin which the output sine wave will come out of.
7. The 4049 is a hex inverter chip, meaning it is composed of 6 inverters.
8. In this circuit, we will simply be using 1 inverter. The first inverter's input is pin 3. The inverter's output is pin 2. In order to power the chip, we supply 5V to pin 1 and we ground pin 8. So all we are using are 4 pins on the inverter chip.
9. As far as the diodes go, any 1N400x diode should be sufficient or even any diode, period.

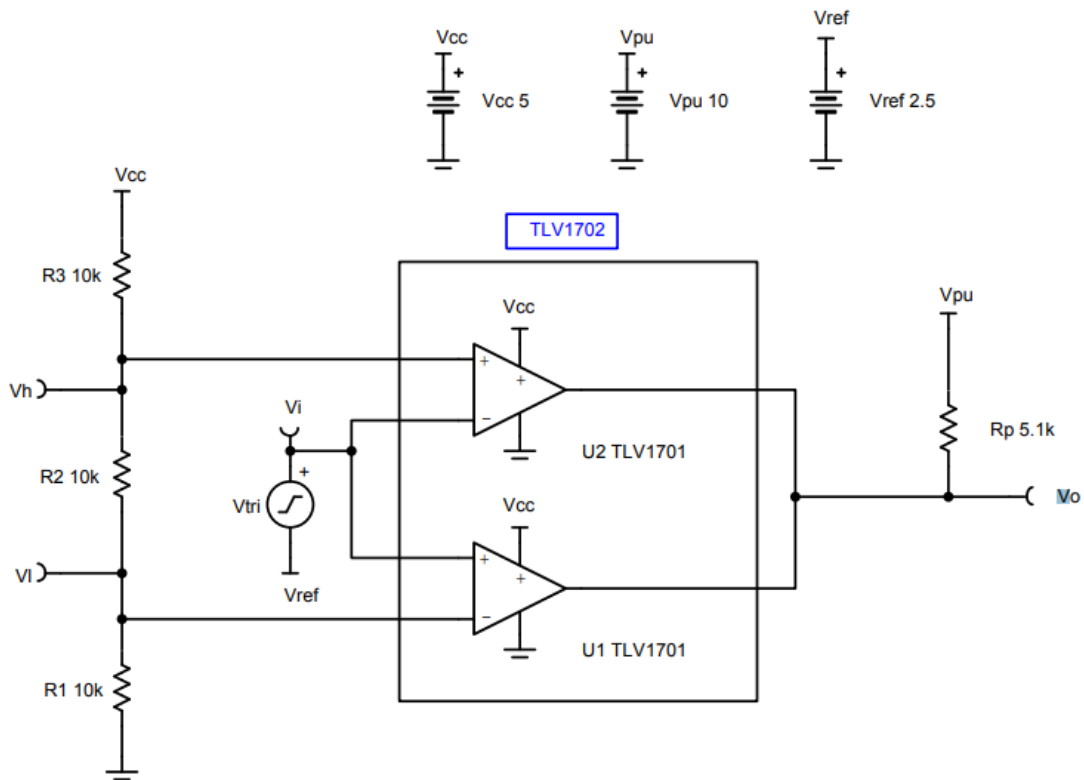
### 10. Pin Diagram:

**LM741 Pinout Diagram**



### Basic Window Comparator and its Output waveform

#### Design:



### Procedure

1. The input should not exceed the common mode limitations of the comparators.
2. If higher pullup voltages are used, Rp should be sized accordingly to prevent large current draw. The TLV1701 supports pullup voltages up to 36V.
3. Comparator must be open-drain or open-collector to allow for the ORed output.

### Design Steps:

$$V_H = V_{cc} \times \frac{R_1 + R_2}{R_1 + R_2 + R_3} = 3.33 \text{ V}$$

1. Define the upper ( $V_H$ ) and lower ( $V_L$ ) window voltages.

$$\frac{V_H}{V_L} = 1 + \frac{R_2}{R_1} = \frac{3.33\text{V}}{1.66\text{V}} = 2$$

$$V_L = V_{cc} \times \frac{R_1}{R_1 + R_2 + R_3} = 1.66 \text{ V}$$

2. Choose resistor values to achieve the desired window voltages.

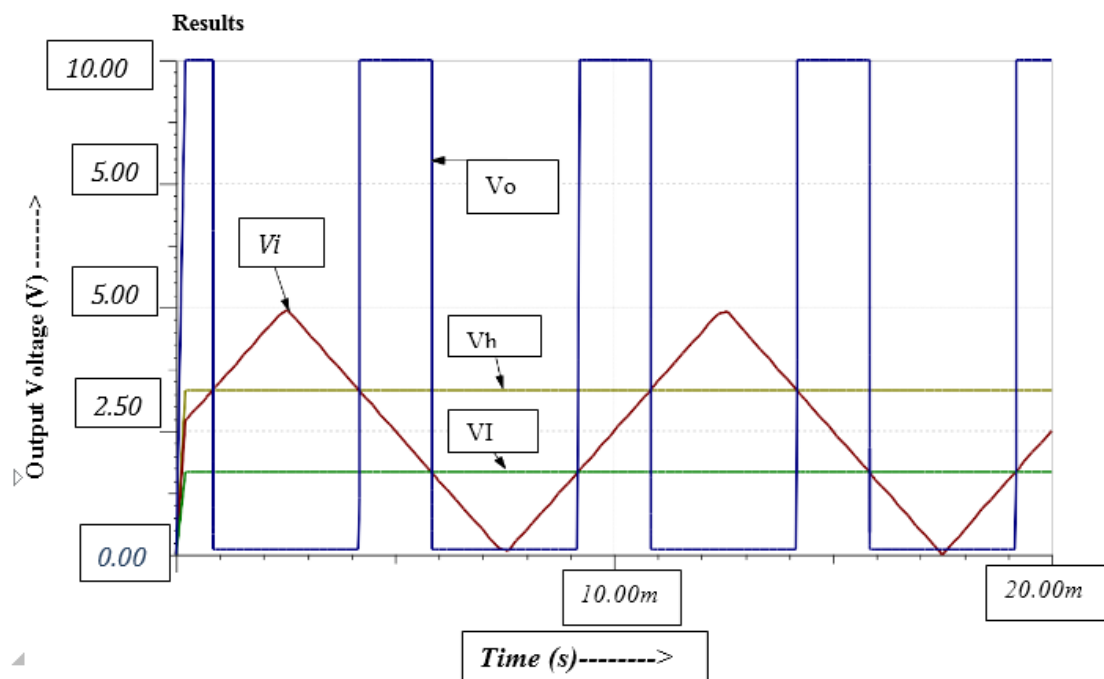
$$\frac{V_H}{V_L} = 1 + \frac{R_2}{R_1} = 2, \text{ so } R_2 = R_1$$

$R_1 = R_2 = 10\text{k}\Omega$  (Selected standard values)

$$R_3 = \frac{R_1 \times V_{cc}}{V_L} - (R_1 + R_2)$$

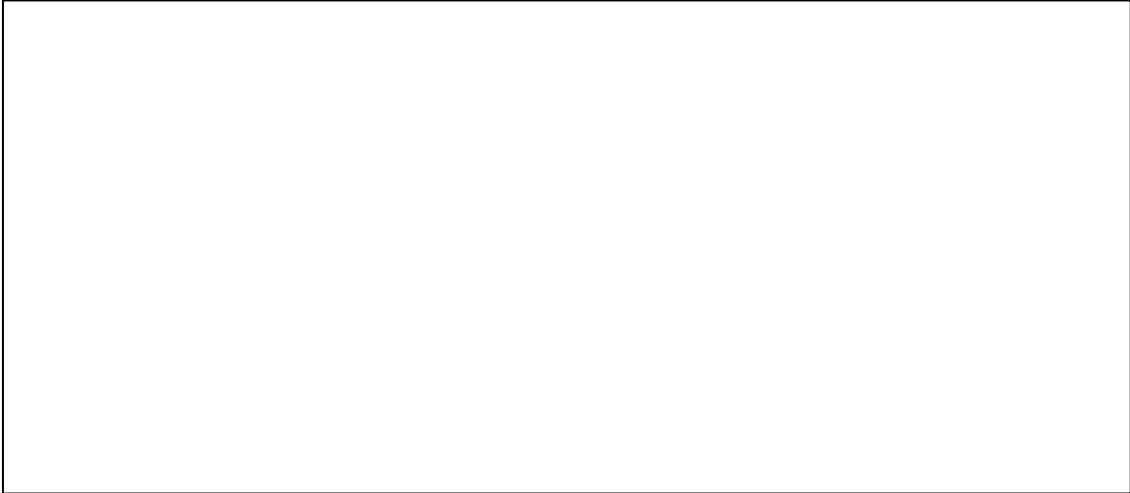
$$R_3 = \frac{10\text{k}\Omega \times 5\text{V}}{1.66\text{V}} - 20\text{k}\Omega = 10.12\text{ k}\Omega \approx 10\text{k}\Omega \text{ (Standard Value)}$$

### Design Simulations:

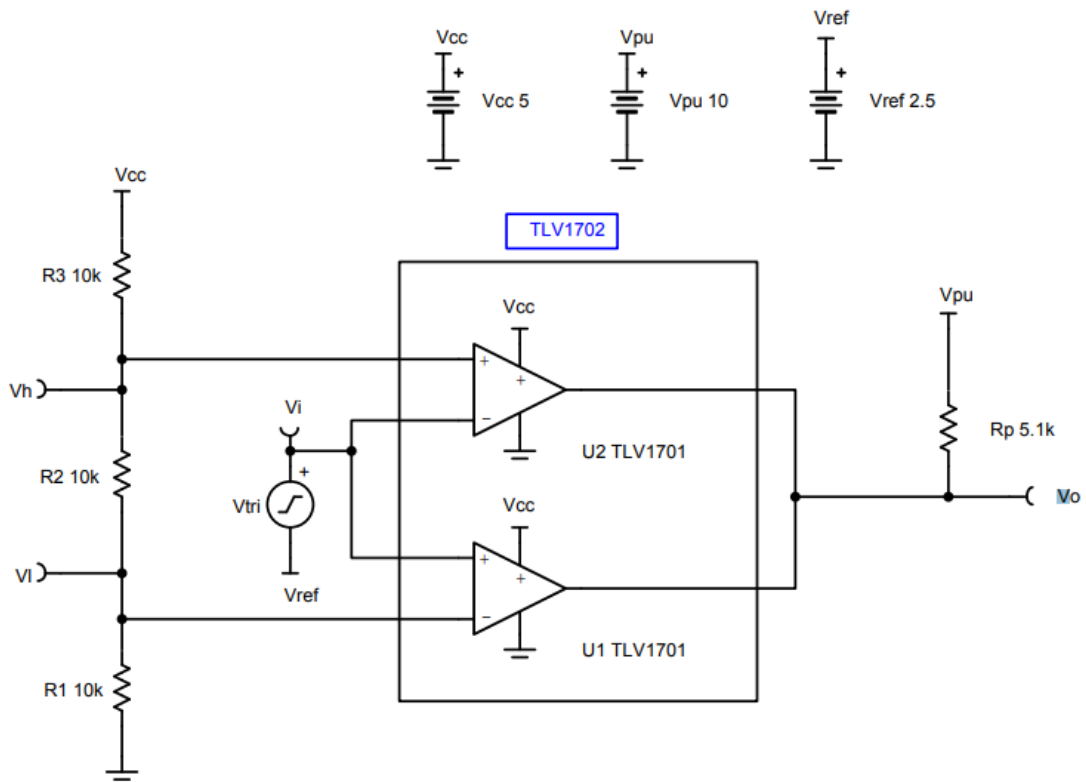




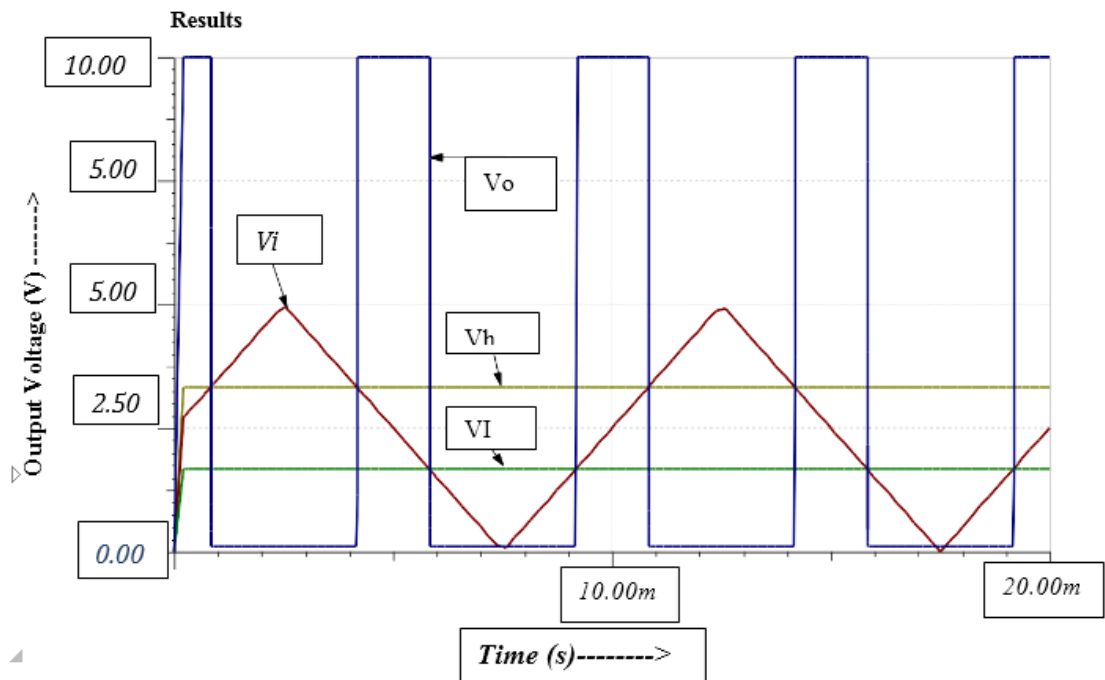
**Results:**



**b. Software Part (Simulation)**



**Waveform:**



**Type of analysis:** TIME DOMAIN (TRANSIENT)

**Run to time:** 10ms

**Step size:** 0.01ms

**PART B (Digital Electronics Circuits)**

**Experiment No 4. Design and implement Half adder, Full Adder, Half Subtractor, Full Subtractor using basic gates. And implement the same in HDL.**

**a. Hardware Part****Description:**

**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:

$$\text{Sum} = A \oplus B \quad \text{Cout} = A B$$

**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,  $C_{in}$ , is called a full-adder. The Boolean functions describing the full-adder are:

$$\text{Sum} = A \oplus B \oplus C_{in} \quad \text{Cout} = AB + C_{in}(A \oplus B)$$

**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:

$$D = A \oplus B \quad \text{Bout} = A'' B$$

**Full Subtractor:** Subtracting two single-bit binary values, B,  $C_{in}$  from a single-bit value A produces a difference bit D and a borrow out  $B_r$  bit. This is called full subtraction. The Boolean functions describing the full-subtractor are:

$$D = A \oplus B \oplus C_{in} \quad B_r = A''B + A''(C_{in}) + B(C_{in})$$

**Components required:**

IC 7400, IC 7408, IC 7486, IC 7432, Patch Cords & IC Trainer Kit.

**I) TO REALIZE HALF ADDER****TRUTH TABLE**

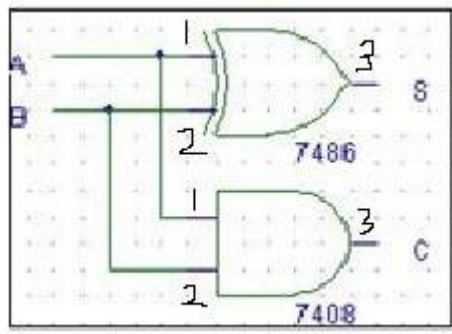
INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

**BOOLEAN EXPRESSIONS:**

$$S = A \oplus B$$

$$C = A B$$

Circuit diagram:



II) FULL ADDER

TRUTH TABLE:

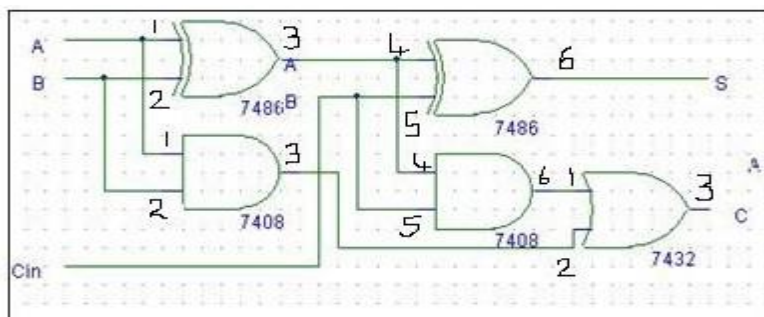
INPUTS			OUTPUTS	
A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

BOOLEAN EXPRESSIONS:

$$S = A \oplus B \oplus C$$

$$C = A B + B C_{in} + A C_{in}$$

Circuit diagram:



**III) HALF SUBTRACTOR**

**TRUTH TABLE**

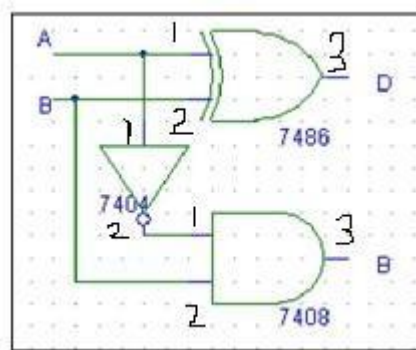
INPUTS		OUTPUTS	
A	B	D	Br
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

**BOOLEAN EXPRESSIONS:**

$$D = A \oplus B$$

$$Br = A \bar{B}$$

**Circuit diagram:**



**IV) FULL SUBTRACTOR**

**TRUTH TABLE**

INPUTS			OUTPUTS	
A	B	Cin	D	Br
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

**BOOLEAN EXPRESSIONS:**

$$D = A \oplus B \oplus C$$

$$Br = \bar{A} B + B Cin + A \bar{Cin}$$



```

else
    SC<="01";

end if;
end process;
end Behavioral;

```

### Truth Table for Half Adder:

INPUTS		OUTPUTS	
A(1)	A(0)	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

### Results of Simulation:



Modelsim Simulation of Half Adder

### VHDL Code for Full Adder using Behavioral Model:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FULLADDER_BEHAVIORAL_SOURCE is
    Port ( A : in  STD_LOGIC_VECTOR (2 downto 0);
          SC : out STD_LOGIC_VECTOR (1 downto 0));
end FULLADDER_BEHAVIORAL_SOURCE;

```

architecture Behavioral of FULLADDER\_BEHAVIORAL\_SOURCE is  
begin

```

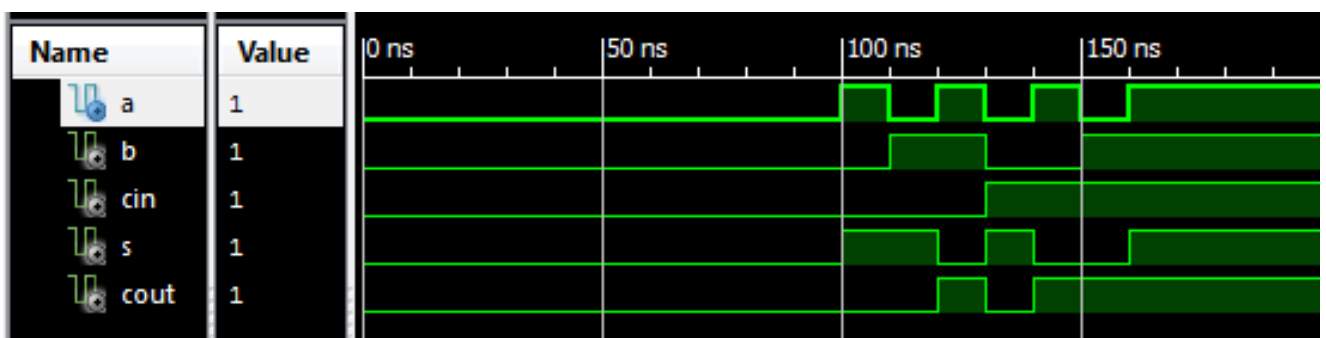
process(A)
begin
  if (A = "001" or A = "010" or A="100") then
    SC<="10";
  elsif (A = "011" or A="101" or "110") then
    SC<="01";
  elsif (A="000") then
    SC<="00"
  else
    SC<="11";
  end if;
end process;
end Behavioral;

```

### Truth Table for Full Adder:

INPUTS			OUTPUTS	
A(2)	A(1)	A(0)	S	C
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

### Simulation Waveform (Full Adder):





**VHDL Code for Half Subtractor using Behavioral Model:**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity HALFSUBTRACTOR_BEHAVIORAL_SOURCE is
    Port ( A : in  STD_LOGIC_VECTOR (1 downto 0);
          DB : out STD_LOGIC_VECTOR (1 downto 0));
end HALFSUBTRACTOR_BEHAVIORAL_SOURCE;
    
```

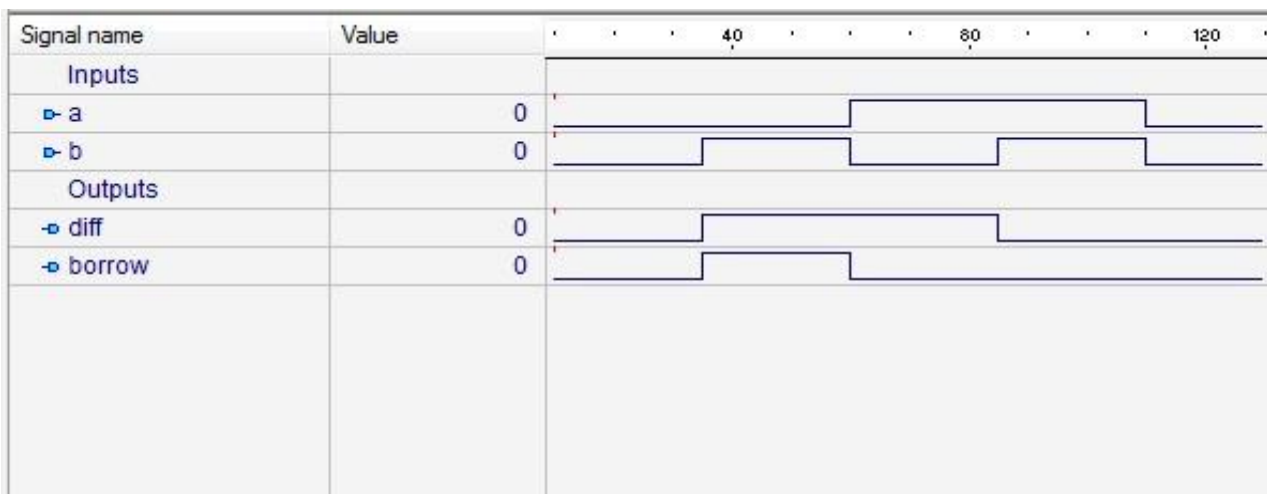
architecture Behavioral of HALFSUBTRACTOR\_BEHAVIORAL\_SOURCE is  
begin

```

    process(A)
    begin
        if (A = "00" or A = "11") then
            DB<="00";
        elsif (A = "01") then
            DB<="11";
        else
            DB<="10";
        end if;
    end process;
end Behavioral;
    
```

INPUTS		OUTPUTS	
A(1)	A(0)	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

**Simulation Waveform (Half Subtractor)**



**VHDL Code for Full Subtractor using Behavioral Model**

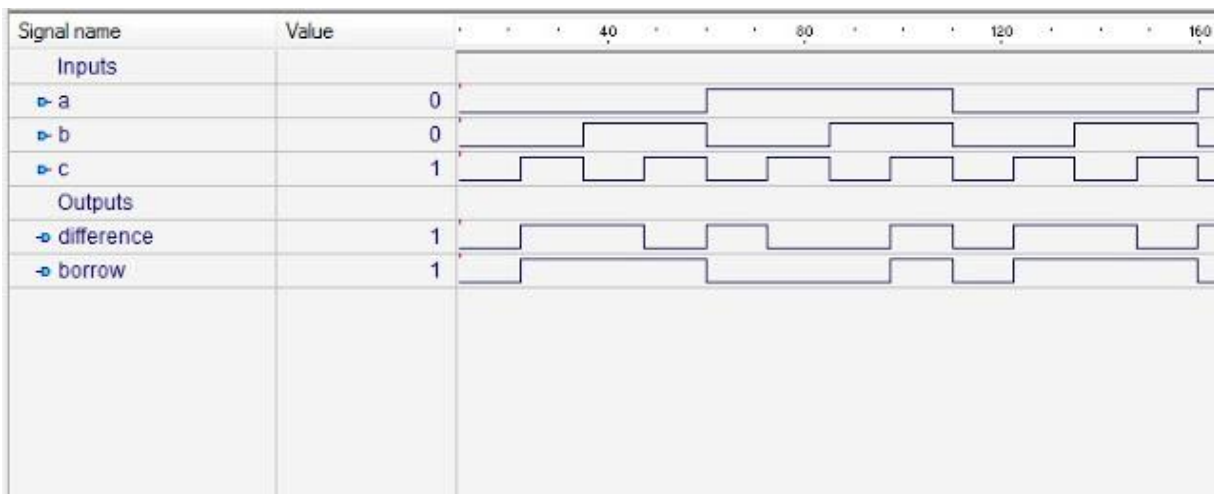
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
    Port ( A : in  STD_LOGIC_VECTOR (2 downto 0);
          DB : out STD_LOGIC_VECTOR (1 downto 0));
end FULLSUBTRACTOR_BEHAVIORAL_SOURCE;

architecture Behavioral of FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
begin
    process (A)
    begin
        if (A = "001" or A = "010" or A = "111") then
            DB <= "11";
        elsif (A = "011") then
            DB <= "01";
        elsif (A = "100") then
            DB <= "10";
        else
            DB <= "00";
        end if;
    end process;
end Behavioral;

```

**Waveform (Full Subtractor) :**

**Experiment No.5:** 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.

**a. Hardware Part:**

**Description:**

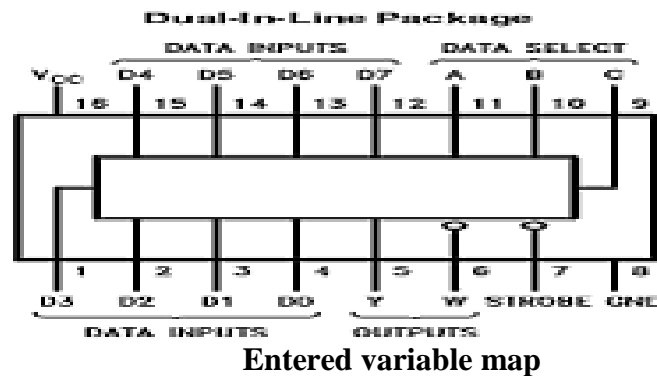
The term multiplex means “many to one”. A multiplexer (MUX) has  $n$  inputs. Each line is used to shift digital data serially. There is a single output line. One of the data stored in the  $n$  input line is transferred to the output based on the value of control bits. An  $n$  to 1 multiplexer requires  $m$  control bits where  $n \leq 2^m$ .

To construct an 4 variable function we require a  $16(2^4)$  to 1 multiplexer, whereas using an entered variable map method a 4 variable expression can be realized using  $8(2^3)$  to 1 multiplexer

**Components Used:**

IC 74 LS151, patch chords, power chords, trainer kit.

**Pin Diagrams: IC 74LS151**



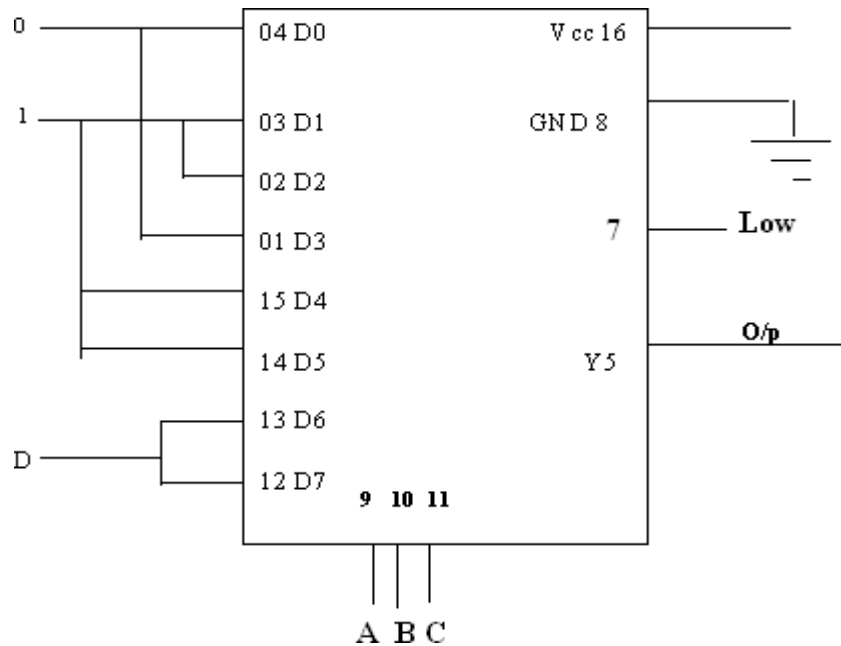
**Example:**

Simplify the following function using EVM technique

$$f(a,b,c,d)=\sum m(2,3,4,5,13,15)+dc(8,9,10,11)$$

Decimal	ABCD	f	MEV map entry	Data
0	0000	0	0	D0
1	0001	0		
2	0010	1	1	D1
3	0011	1		
4	0100	1	1	D2
5	0101	1		
6	0110	0	0	D3
7	0111	0		
8	1000	X	X	D4
9	1001	X		
10	1010	X	X	D5
11	1011	X		
12	1100	0	D	D6
13	1101	1		
14	1110	0	D	D7
15	1111	1		

**Circuit Diagram**



**Procedure:**

1. Verify all components & patch chords whether they are in good condition or not.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit.
4. Provide input data to circuit via switches
5. Verify truth table sequence & observe outputs.

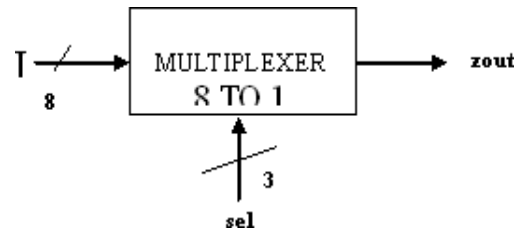
**Result:**

The truth table is verified

## b. Software Part

### Description:

An 8:1 multiplexer has 8 inputs and one output. The data stored in one of these 8 input lines is transferred serially to the output based on the value of the selection bits



### Truth table:

INPUTS			OUTPUTS
SEL (2)	SEL (1)	SEL (0)	Zout
0	0	0	I(0)
0	0	1	I(1)
0	1	0	I(2)
0	1	1	I(3)
1	0	0	I(4)
1	0	1	I(5)
1	1	0	I(6)
1	1	1	I(7)

### Algorithm for 8 to1 multiplexer:

#### Input/output

I is a one dimensional array of size 8

Sel is a one dimensional array of size 3

Zout is output

#### Method

If the value of sel is 000 zout = I[0]

If the value of sel is 001 zout = I[1]

If the value of sel is 010 zout = I[2]

If the value of sel is 011 zout = I[3]

If the value of sel is 100 zout = I[4]

If the value of sel is 101 zout = I[5]

If the value of sel is 110 zout = I[6]

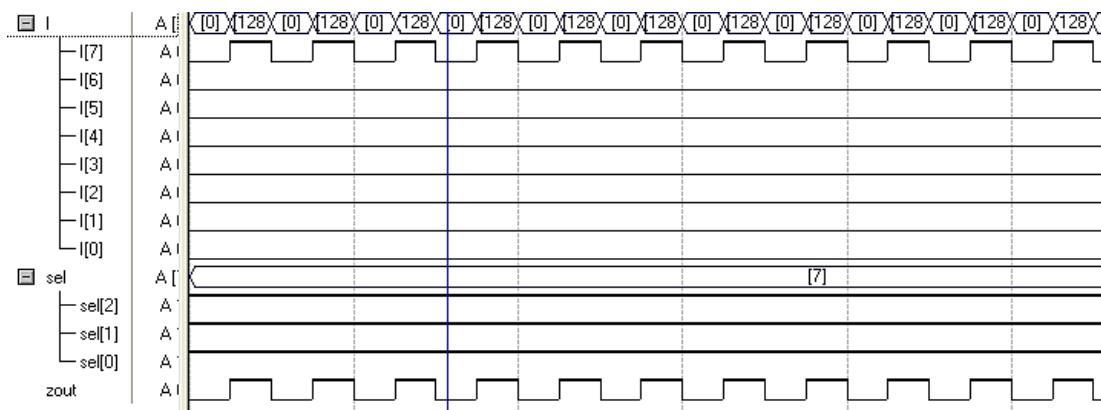
Else the value of zout is I[7]

**VHDL code for 8 to 1 MUX (behavioral modeling):**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;           // includes the standard library
entity mux1 is
  Port ( I : in std_logic_vector(7 downto 0);
        sel : in std_logic_vector(2 downto 0); //Input and output is declared as ports
        zout : out std_logic);
end mux1;
architecture Behavioral of mux1 is
begin
  zout<= I(0) when sel="000" else // Based on the value of selection the value of data
    I(1) when sel="001" else //stored in the array I is stored in zout
    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;

```

**OUTPUT:**

**Experiment No.6: Realize a J-K Master / Slave Flip-Flop using NAND gates and verify its truth table. And implement the same in HDL.**

**a. Hardware Part**

**Description:**

A **flip-flop** is a device very much like a latch in that it is a bistable multivibrator, having two states and a feedback path that allows it to store a bit of information. The difference between a latch and a flip-flop is that a latch is asynchronous, and the outputs can change as soon as the inputs do (or at least after a small propagation delay). A flip-flop, on the other hand, is edge-triggered and only changes state when a control signal goes from high to low or low to high.

**Master Slave Flip Flop:**

The control inputs to a clocked flip flop will be making a transition at approximately the same times as triggering edge of the clock input occurs. This can lead to unpredictable triggering.

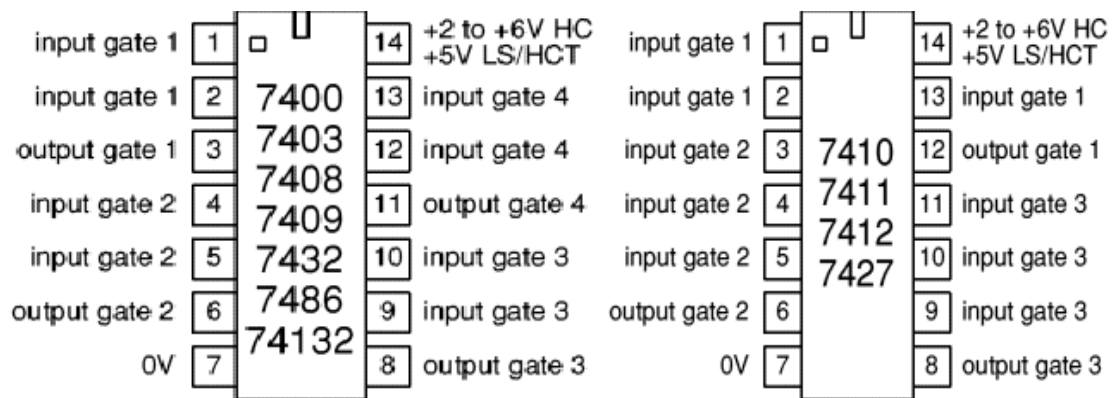
A JK master flip flop is positive edge triggered, whereas slave is negative edge triggered. Therefore master first responds to J and K inputs and then slave. If J=0 and K=1, master resets on arrival of positive clock edge. High output of the master drives the K input of the slave.

For the trailing edge of the clock pulse the slave is forced to reset. If both the inputs are high, it changes the state or toggles on the arrival of the positive clock edge and the slave toggles on the negative clock edge. The slave does exactly what the master does.

**Components used:**

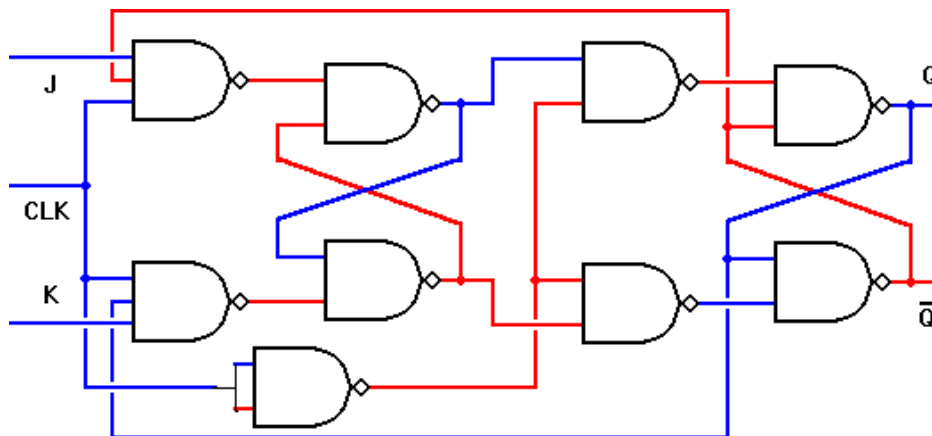
IC 74 LS00, IC 74LS10, patch chords, trainer kit.

**Pin Diagrams:74LS10,74LS00**



**Truth table:**

Clk	J	K	Q	Q bar	comment
	0	0	Q <sub>0</sub>	Q bar	No change
	0	1	0	1	Reset
	1	0	1	0	Set
	1	1	Q <sub>0</sub>	Q <sub>0</sub>	toggle

**CIRCUIT DIAGRAM:****Procedure:**

1. Verify all components & patch chords whether they are in good condition or not.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit.
4. Provide input data to circuit via switches.
5. Verify truth table sequence & observe outputs.

**Conclusion:**

Truth table is verified



**b. Software Part (VHDL Code for JK Master Slave Flip Flop using Behavioral Model)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity jk_flip_flop is
  port(
    j : in STD_LOGIC;
    k : in STD_LOGIC;
    clk : in STD_LOGIC;
    reset : in STD_LOGIC;
    q : inout STD_LOGIC;
    qb : out STD_LOGIC
  );
end jk_flip_flop;

architecture jk_flip_flop_arc of jk_flip_flop is
begin
  process (j,k,clk,reset) is
    variable m : std_logic := '0';
  begin
    if (reset='1') then
      m := '0';
    elsif (rising_edge (clk)) then
      if (j/=k) then
        m := j;
      elsif (j='1' and k='1') then
        m := not m;
      end if;
    end if;
    q <= m;
    qb <= not m;
  end process;
end jk_flip_flop_arc;
```

**Experiment No 7. Design and implement code converter I) Binary to Gray II) Gray to Binary Code using basic gates.**

**Description:**

**Gray Code** is one of the most important codes. It is a non-weighted code which belongs to a class of codes called minimum change codes. In this codes while traversing from one step to another step only one bit in the code group changes. In case of **Gray Code** two adjacent code numbers differs from each other by only one bit.

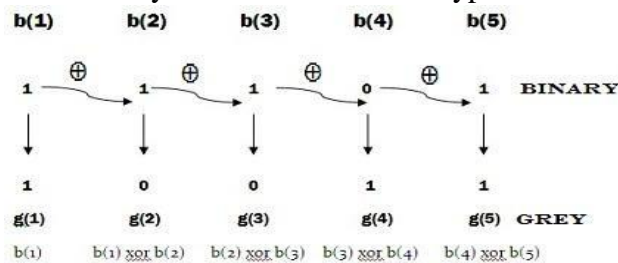
**Binary to gray code conversion** is a very simple process. There are several steps to do this types of conversions.

Steps given below elaborate on the idea on this type of conversion.

(1) The M.S.B. of the gray code will be exactly equal to the first bit of the given binary number.

(2) Now the second bit of the code will be exclusive-or of the first and second bit of the given binary number, i.e if both the bits are same the result will be 0 and if they are different the result will be 1.

(3)The third bit of gray code will be equal to the exclusive-or of the second and third bit of the given binary number. Thus the **Binary to gray code conversion** goes on. One example given below can make your idea clear on this type of conversion.

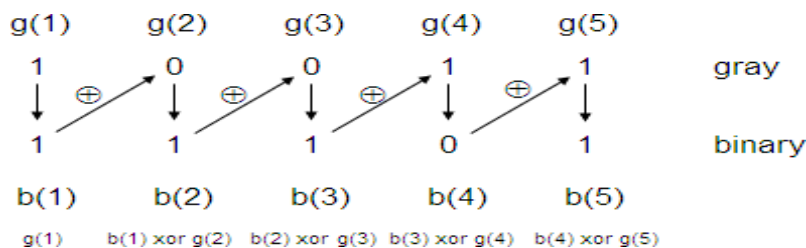


**Gray code to binary conversion** is again very simple and easy process. Following steps can make your idea clear on this type of conversions.

(1) The M.S.B of the binary number will be equal to the M.S.B of the given gray code.

(2) Now if the second gray bit is 0 the second binary bit will be same as the previous or the first bit. If the gray bit is 1 the second binary bit will alter. If it was 1 it will be 0 and if it was 0 it will be 1.

(3) This step is continued for all the bits to do **Gray code to binary conversion**. One example given below will make your idea clear.



**Components required:**

IC 7486, Patch Cords & digital trainer Kit.

**II) BINARY TO GRAY CONVERSION**

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

0	1	0	1
0	1	0	1
0	1	0	1
0	1	0	1

0	1	1	0
0	1	1	0
1	0	0	1
1	0	0	1

0	0	0	0
1	1	1	1
0	0	0	0
1	1	1	1

BOOLEAN EXPRESSIONS:

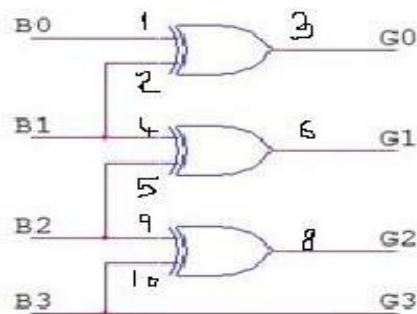
$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_1 \oplus B_2$

$G_0 = B_1 \oplus B_0$

**Circuit Diagram:** BINARY TO GRAY CODE



**TRUTH TABLE:**

Binary				Gray			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

**Circuit diagram of Binary to Gray code using only Basic Gates**



III) GRAY TO BINARY CONVERSION

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

$B_3 = G_3$

0	1	0	1
0	1	0	1
0	1	0	1
0	1	0	1

$B_2 = G_3 \oplus G_2$

0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

$B_1 = G_3 \oplus G_2 \oplus G_1$

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$

**BOOLEAN EXPRESSIONS:**

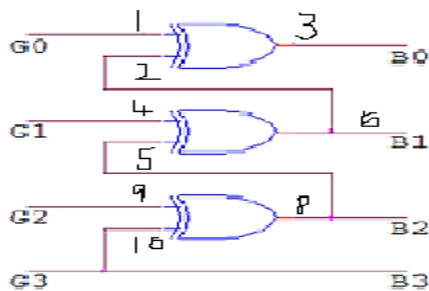
$B_3 = G_3$

$B_2 = G_3 \oplus G_2$

$B_1 = G_3 \oplus G_2 \oplus G_1$

$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$

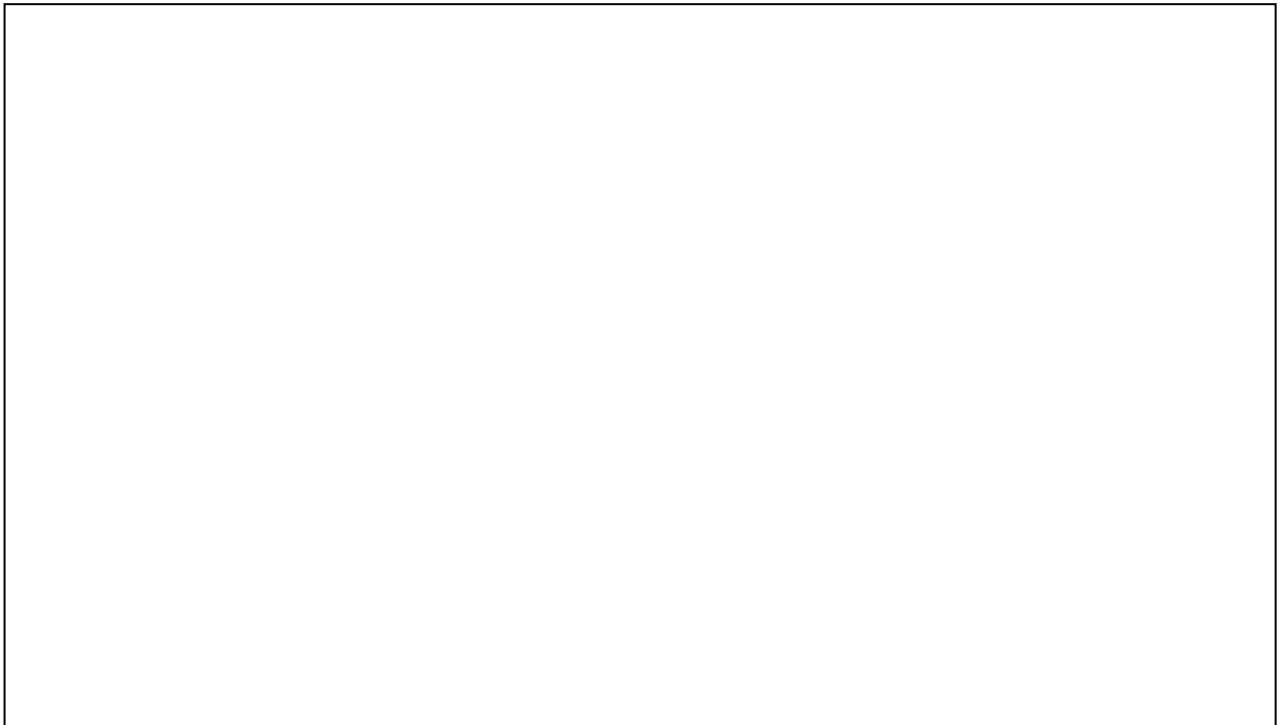
Circuit Diagram: Gray to Binary



**TRUTH TABLE:**

Gray				Binary			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

**Circuit diagram of Gray to Binary code using only Basic Gates**



**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. Verify the Truth Table and observe the outputs.

**RESULT:** Binary to gray code conversion and vice versa is realized using EX-OR gates

### Experiment No.8: Design and implement a mod n ( $n < 8$ ) synchronous up counter using JK Flip Flop ICs and demonstrate its working.

#### a. Hardware Part

##### Description:

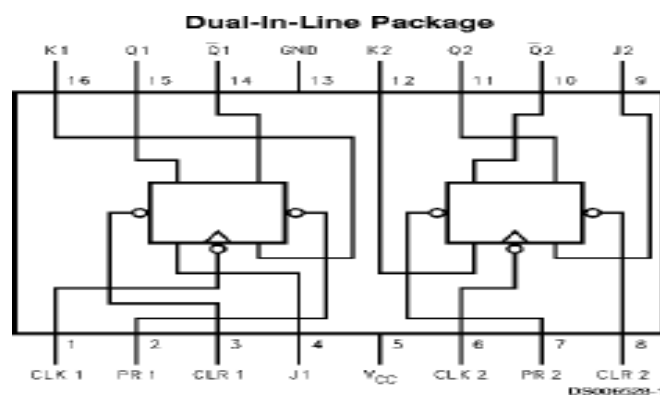
The ripple counter requires a finite amount of time for each flip flop to change state. This problem can be solved by using a synchronous parallel counter where every flip flop is triggered in synchronism with the clock, and all the output which are scheduled to change do so simultaneously.

The counter progresses counting upwards in a natural binary sequence from count 000 to count 100 advancing count with every negative clock transition and get back to 000 after this cycle.

##### Components Used:

IC 74 LS76, IC 74LS08, patch chords, trainer kit.

##### Pin Diagrams:74LS76



##### Synchronous counter design:

To successfully design synchronous counters we may employ the following six basic steps:

1. Create the state transition diagram.
2. Create a present state-next state table (often referred to as the **next state** table).
3. Expand the table to form the transition table for each flip-flop in the circuit. The transition table shows the flip-flop inputs required to make the counter go from present state to the desired next state. This is also referred to as the excitation table.
4. Determine the logic functions of the J and K inputs as a function of the present states.
5. Analyse the counter to verify the design.
6. Construct and test the counter.



**Function Table:**

Inputs					Outputs	
PR	CLR	CLK	J	K	Q	$\bar{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H	H
H	H	$\square$	L	L	$Q_0$	$\bar{Q}_0$
H	H	$\square$	H	L	H	L
H	H	$\square$	L	H	L	H
H	H	$\square$	H	H	Toggle	Toggle

for initialization

for normal

**Functional Truth Table for J-K Flip Flop:**

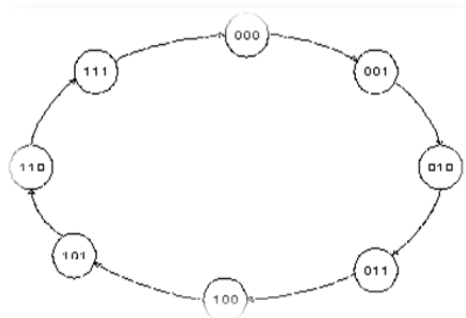
J	K	$Q_n$	$Q_{n+1}$
0	0	0	0
		1	1
0	1	0	0
		1	0
1	0	0	1
		1	1
1	1	0	1
		1	0

**State Synthesis Table for JK Flip Flop**

Present state	Next state	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Employ these techniques to design a MOD-8 counter to count in the following sequence: 0, 1, 2, 3, 4, 5, 6, 7.

**Step1:** Creating state transition diagram.



**Step 2:** Creating present state-next state table

Present State			Next State		
$Q_c$	$Q_b$	$Q_a$	$Q_c$	$Q_b$	$Q_a$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

**Step 3:** Expand the present state-next state table to form the transition table.

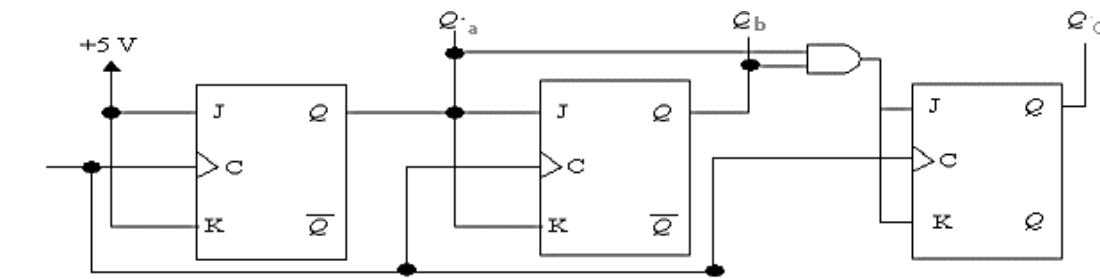
	Present State			Next State			Present inputs					
	$Q_c$	$Q_B$	$Q_A$	$Q_C$	$Q_B$	$Q_A$	$J_C$	$K_c$	$JB$	$K_B$	$JA$	$KA$
0	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	0	0	X	1	X	X	1
2	0	1	0	0	1	1	0	X	X	0	1	X
3	0	1	1	1	0	0	1	X	X	1	X	1
4	1	0	0	1	0	1	X	0	0	X	1	X
5	1	0	1	1	1	0	X	0	1	x	X	1
6	1	1	0	1	1	1	X	0	X	0	1	X
7	1	1	1	0	0	0	X	1	X	1	X	1

**Step 4:** Use Karnaugh maps to identify the present state logic functions for each of the 'X' indicates a "don't care" condition.  
input.

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"><b>C</b> <b>BA</b></td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">x</td> <td style="text-align: center;">x</td> <td style="text-align: center;">x</td> <td style="text-align: center;">x</td> </tr> </table> <p style="text-align: center;"><b>Jc = QbQa</b></p>	<b>C</b> <b>BA</b>	00	01	11	10	0	0	0	1	0	0	x	x	x	x	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"><b>C</b> <b>BA</b></td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">x</td> <td style="text-align: center;">x</td> <td style="text-align: center;">x</td> <td style="text-align: center;">x</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> </table> <p style="text-align: center;"><b>Kc=QbQa</b></p>	<b>C</b> <b>BA</b>	00	01	11	10	0	x	x	x	x	0	0	0	1	0
<b>C</b> <b>BA</b>	00	01	11	10																											
0	0	0	1	0																											
0	x	x	x	x																											
<b>C</b> <b>BA</b>	00	01	11	10																											
0	x	x	x	x																											
0	0	0	1	0																											
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"><b>C</b> <b>BA</b></td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">X</td> <td style="text-align: center;">x</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">X</td> <td style="text-align: center;">x</td> </tr> </table> <p style="text-align: center;"><b>Jb = Qa</b></p>	<b>C</b> <b>BA</b>	00	01	11	10	0	0	1	X	x	0	0	1	X	x	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"><b>C</b> <b>BA</b></td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">x</td> <td style="text-align: center;">X</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">x</td> <td style="text-align: center;">X</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> </table> <p style="text-align: center;"><b>Kb = Qa</b></p>	<b>C</b> <b>BA</b>	00	01	11	10	0	x	X	1	0	0	x	X	1	0
<b>C</b> <b>BA</b>	00	01	11	10																											
0	0	1	X	x																											
0	0	1	X	x																											
<b>C</b> <b>BA</b>	00	01	11	10																											
0	x	X	1	0																											
0	x	X	1	0																											
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"><b>C</b> <b>BA</b></td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">X</td> <td style="text-align: center;">X</td> <td style="text-align: center;">1</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">X</td> <td style="text-align: center;">X</td> <td style="text-align: center;">1</td> </tr> </table> <p style="text-align: center;"><b>Ja = 1</b></p>	<b>C</b> <b>BA</b>	00	01	11	10	0	1	X	X	1	0	1	X	X	1	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"><b>C</b> <b>BA</b></td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">X</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">X</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">X</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">X</td> </tr> </table> <p style="text-align: center;"><b>Ka = 1</b></p>	<b>C</b> <b>BA</b>	00	01	11	10	0	X	1	1	X	0	X	1	1	X
<b>C</b> <b>BA</b>	00	01	11	10																											
0	1	X	X	1																											
0	1	X	X	1																											
<b>C</b> <b>BA</b>	00	01	11	10																											
0	X	1	1	X																											
0	X	1	1	X																											

**Step 5:** Trace through indicates circuit should work correctly.

**Step 6:** Constructing Circuit



A three-bit synchronous counter

**Note:**

Connect preset and clear to high to work on normal operation

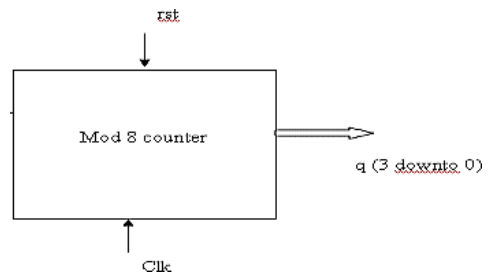
**Conclusion:**

A mod 8 counter was designed using J K Flip Flop and the truth table is verified.

**b. Software Part (VHDL Code for MOD-8 Synchronous Up Counter using Behavioral Model)**

**Description:**

A modulus 8 counter has 8 unique states . The inputs are clock and reset. The operation of counter is summarized in the truth table below



**Truth Table:**

rst	clock	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>
1	x	0	0	0
0	↓	0	0	1
0	↓	0	1	0
0	↓	0	1	1
0	↓	1	0	0
0	↓	1	0	1
0	↓	1	1	0
0	↓	1	1	1
0	↓	0	0	0

**Algorithm:**

Input :clock and reset

Output: Q

Method:

Execute the code repeated if clk or rst changes

begin

1. If rst is high counter remains stable at state 0

2. If rst is low and if clock occurs the counter progresses through state 0 to 7

end

**VHDL Code for MOD-8 Counter:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity mod8 is

Port ( rst,clk: in std\_logic; // rst and clock is declared as input port

q : buffer std\_logic\_vector(2 downto 0):="000"); //Q is declares as a buffer as it is

end mod8; // used for input and as output and it should store a value

architecture Behavioral of mod8 is

process(clk,rst) is // process uses for sequential circuits ececutes the code

// with in if clk or rst

begin

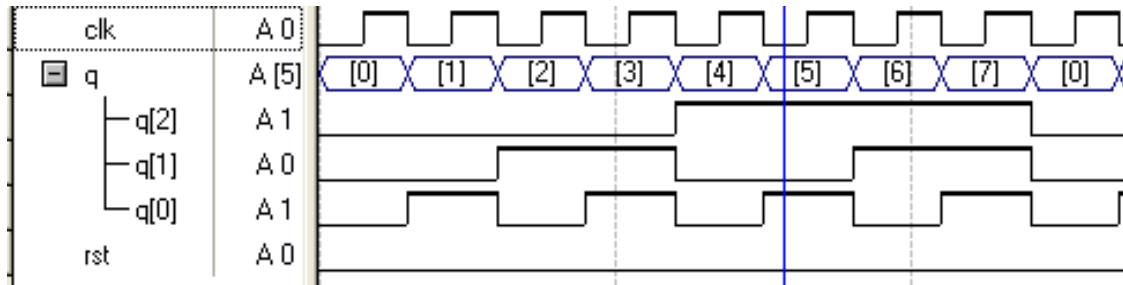
//changes

```

if (CLK'event and clk='0') then
    if(rst='1') then
        q<="000";
    else
        q <= q+1;
    end if;
end if;
end process;
end Behavioral;

```

**Result:**



**Experiment No.9: Design and implement asynchronous counter using decade counter IC to count up from 0 to n ( $n \leq 9$ ) and demonstrate on seven segment display (using IC-7447).**

**Description:**

Asynchronous counter is a counter in which the clock signal is connected to the clock input of only first stage flip flop. The clock input of the second stage flip flop is triggered by the output of the first stage flip flop and so on. This introduces an inherent propagation delay time through a flip flop. A transition of input clock pulse and a transition of the output of a flip flop can never occur exactly at the same time. Therefore, the two flip flops are never simultaneously triggered, which results in asynchronous counter operation.

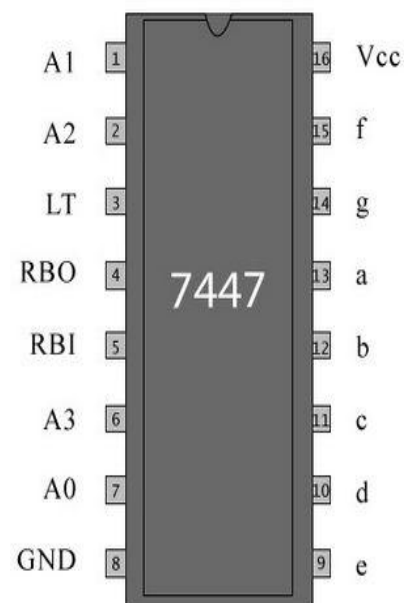
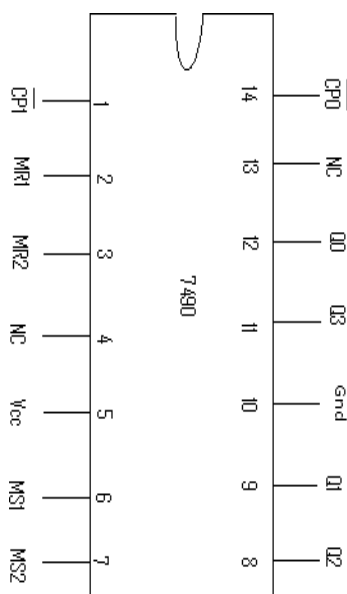
**Components Used:**

IC 74 LS90, IC 7447 (BCD to seven segment decoder), patch chords, trainer kit.

1. Cp0 (pin 14) to be connected to clock

2. Cp1 (pin 1) to be connected to Q0. (The output of the first flip flop drives the second clock)

**Pin Diagram: 74LS90 / 74LS47**



**Pin Names Description of 7447:**

A0–A3 =BCD Inputs

RBI =Ripple Blanking Input (Active LOW)

LT= Lamp Test Input (Active LOW)

RBO =Ripple Blanking Output (Active LOW)

a –g =Segment Outputs (Active LOW)

**Pin Names Description of 7490:**

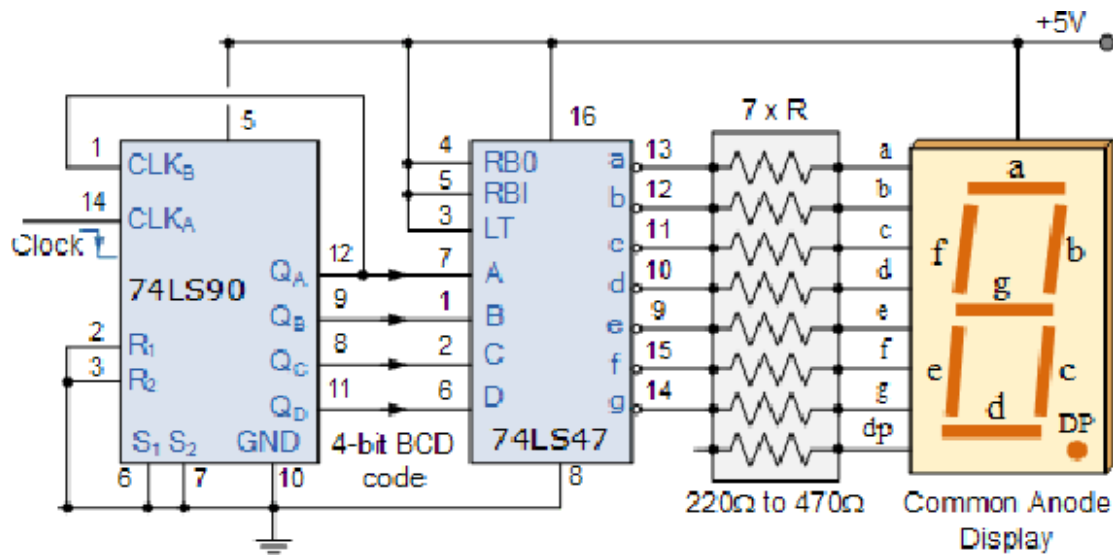
R1 and R2-clear all flipflop (high active and low for not active)

S1 and S2- set all flip flop (high active and low for not active)

CLK<sub>A</sub>-clock pulse to first flip flop

CLK<sub>B</sub>-clock pulse to second flip flop (output of first flip flop clock for second flip flop)

**Circuit Diagram:**



**For mod 9**

connect Q0 and Q3 to reset(clear) through an AND gate. Reset should not be connected to the switch

**For mod8**

Connect Q3 to reset

**For mod7**

Connect Q2, Q1,Q0 to reset through an And Gate

**For Mod 6**

Connect Q2 and Q1 to reset through an AND gate

**For mod 5**

Connect Q0 and Q2 to reset through an AND gate

**For Mod 4**

Connect Q2 to reset

**For mod 3**

Connect Q1 and Q0 to reset through an AND gate

**For mod 2**

Connect Q1 to reset

**Function Table:**

Clock	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

**Procedure:**

1. Verify all components & patch chords whether they are in good condition or not.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit.
4. Provide input data to circuit via switches.
5. Verify truth table sequence & observe outputs.

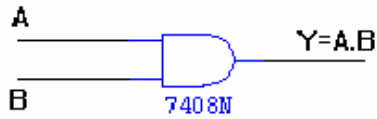
**Result:**

mod  $n \leq 9$  counter implemented using the decade counter IC.



User Manual – 1  
IC Pin Diagrams

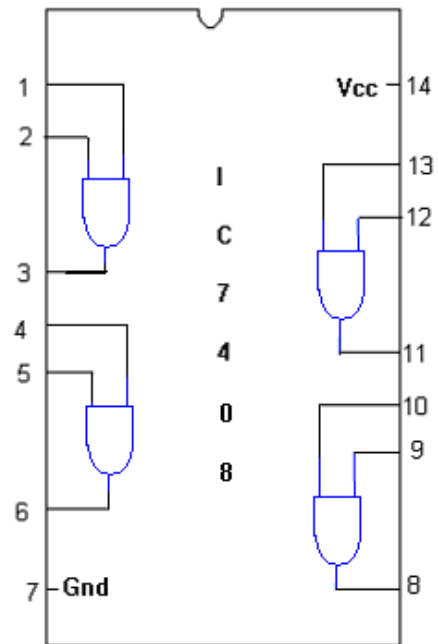
**SYMBOL:**



**TRUTH TABLE**

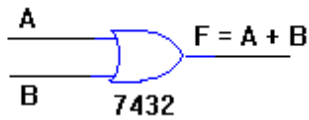
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

**PIN DIAGRAM:**



**OR GATE:**

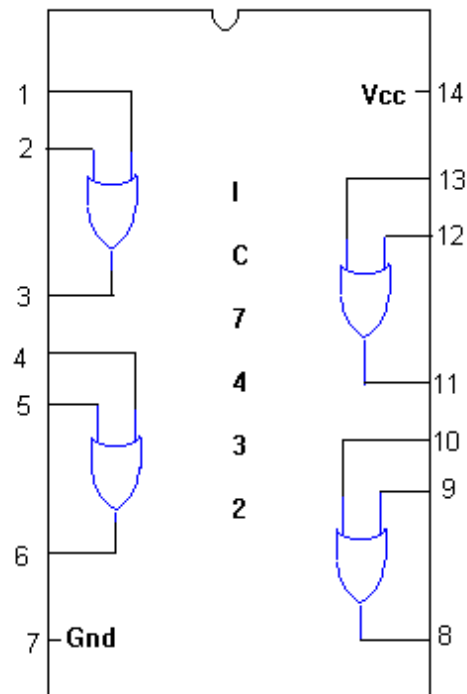
**SYMBOL :**



**TRUTH TABLE**

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

**PIN DIAGRAM :**



**NOT GATE:**

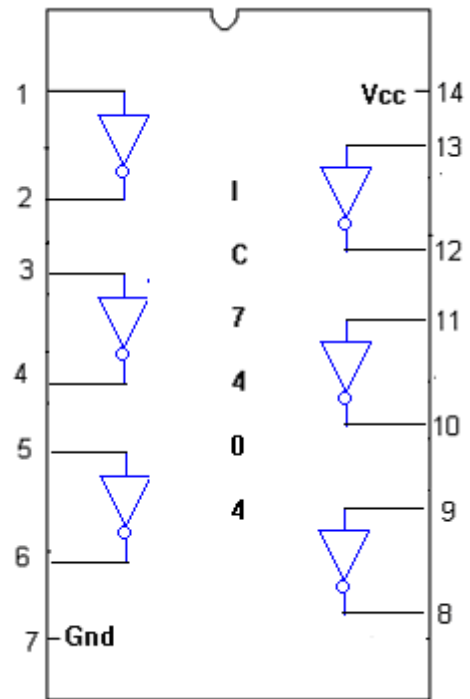
**SYMBOL:**



**TRUTH TABLE :**

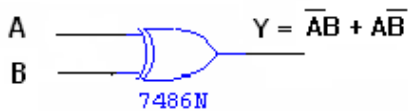
A	$\bar{A}$
0	1
1	0

**PIN DIAGRAM:**



**X-OR GATE :**

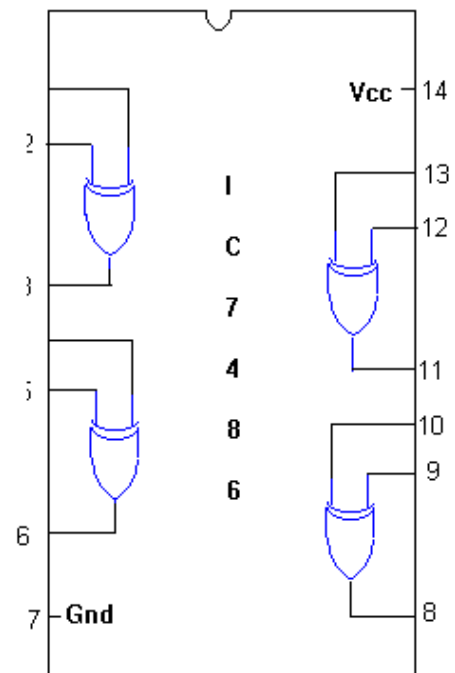
**SYMBOL :**



**TRUTH TABLE :**

A	B	$\bar{A}B + A\bar{B}$
0	0	0
0	1	1
1	0	1
1	1	0

**PIN DIAGRAM :**



**2-INPUT NAND GATE:**

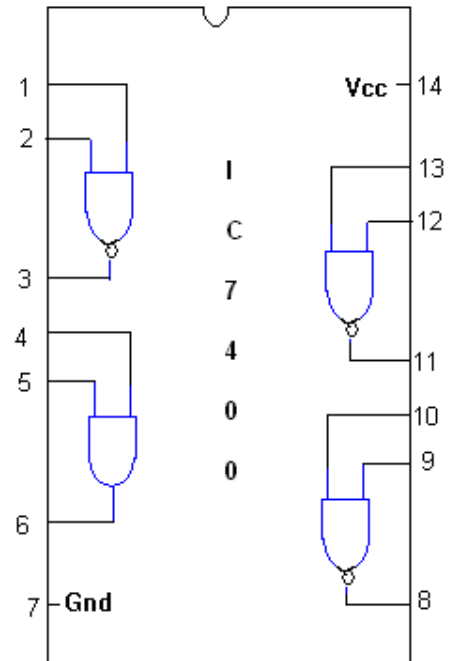
**SYMBOL:**



**TRUTH TABLE**

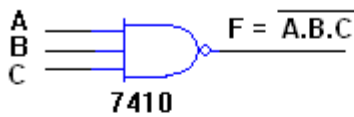
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

**PIN DIAGRAM:**



**3-INPUT NAND GATE :**

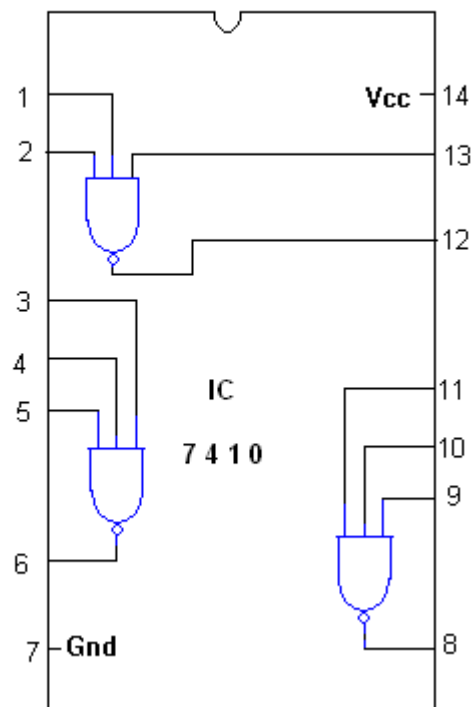
**SYMBOL :**



**TRUTH TABLE**

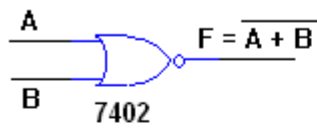
A	B	C	$\overline{A \cdot B \cdot C}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

**PIN DIAGRAM :**

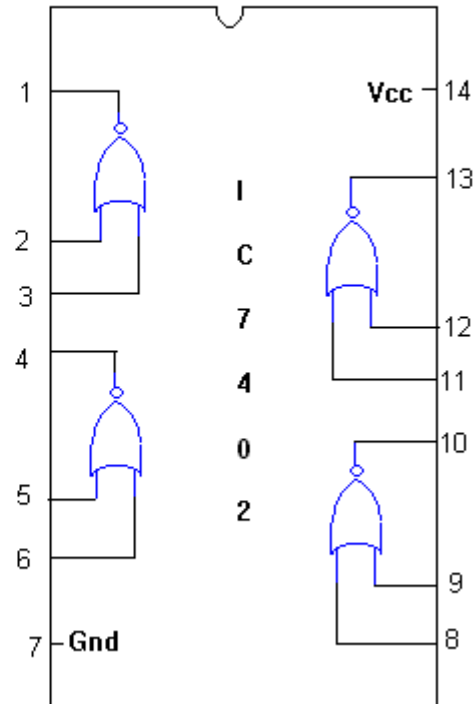


**NOR GATE:**

SYMBOL :



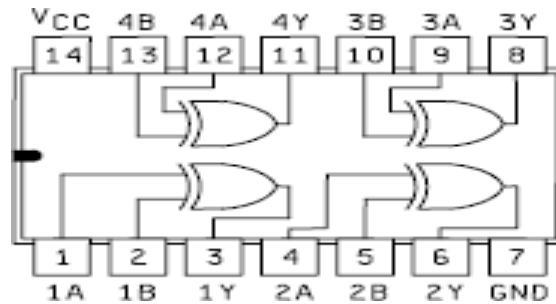
PIN DIAGRAM :



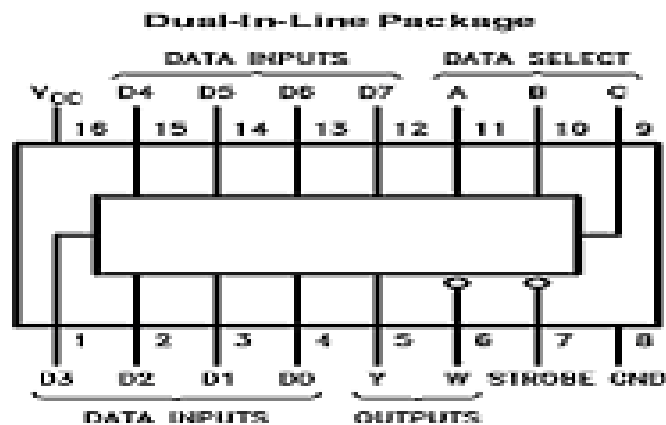
TRUTH TABLE

A	B	$\overline{A+B}$
0	0	1
0	1	1
1	0	1
1	1	0

**IC 74LS86**

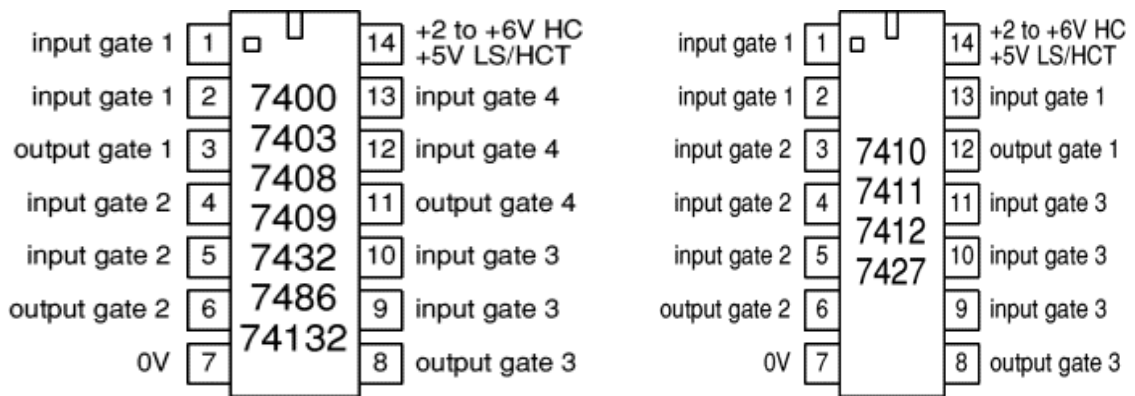


**IC 74LS151**

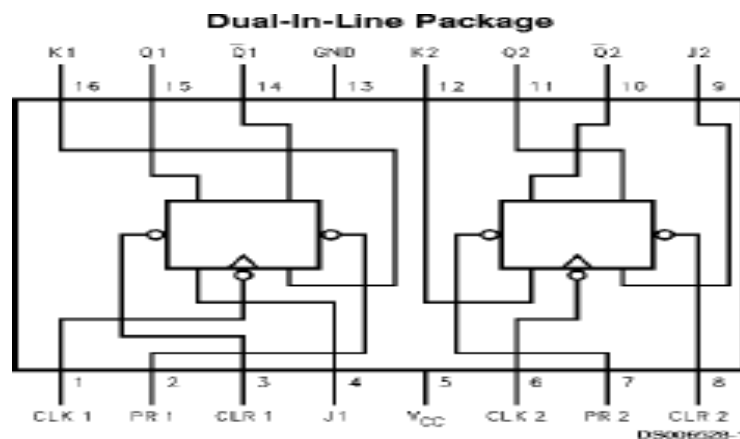


**IC 7400**

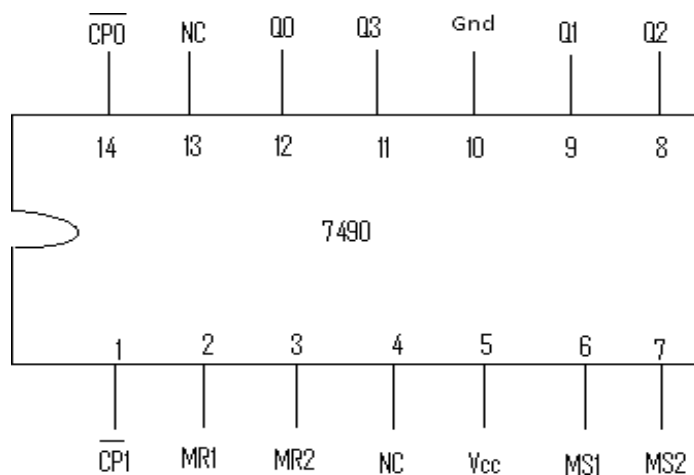
**IC 7410**



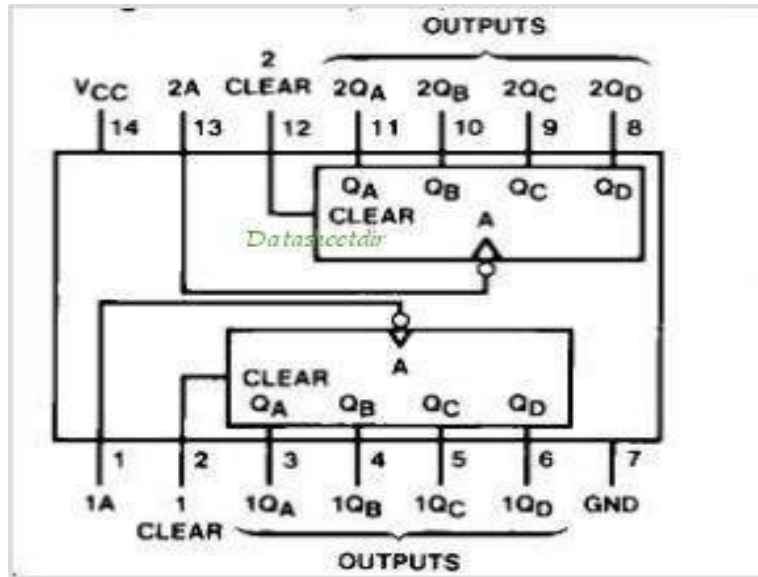
**IC 74LS76 ( two JK flip flop in one IC)**



**IC 7490**

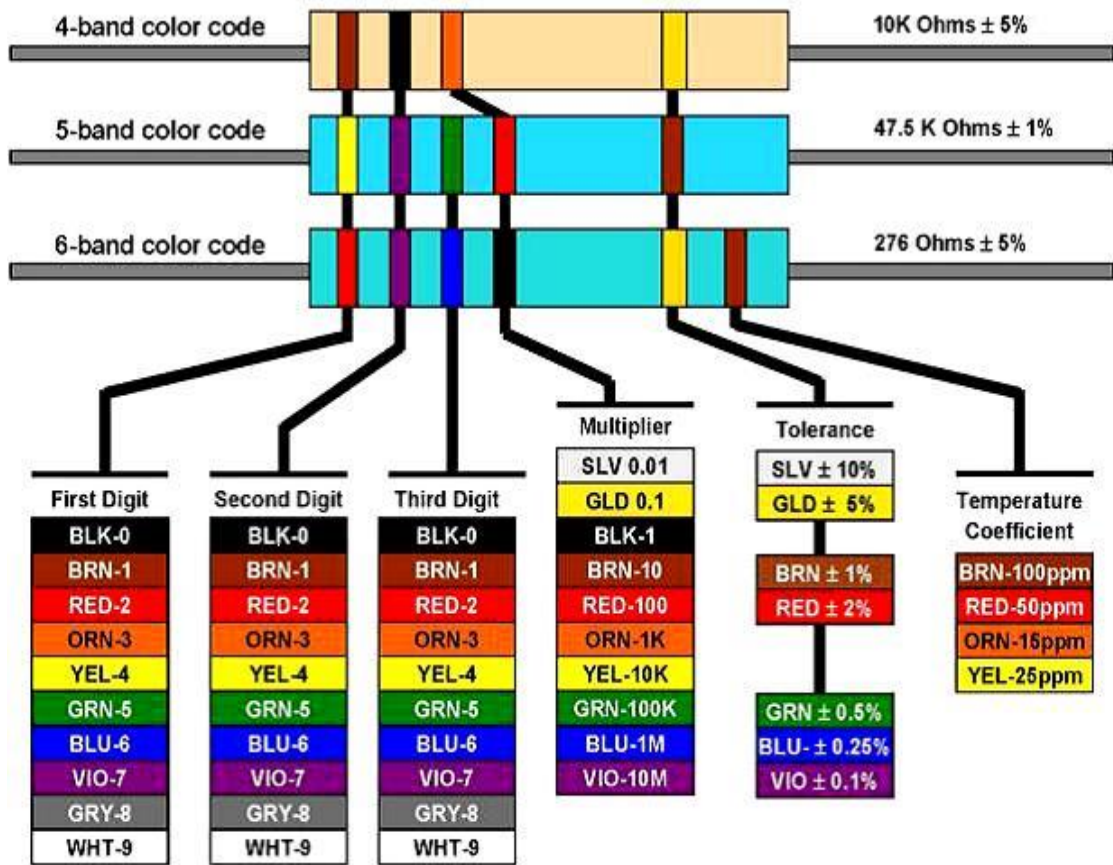


74LS393 binary counter IC:



User Manual – 2  
Resistor value finder

# Resistor Color Code



### User Manual – 3

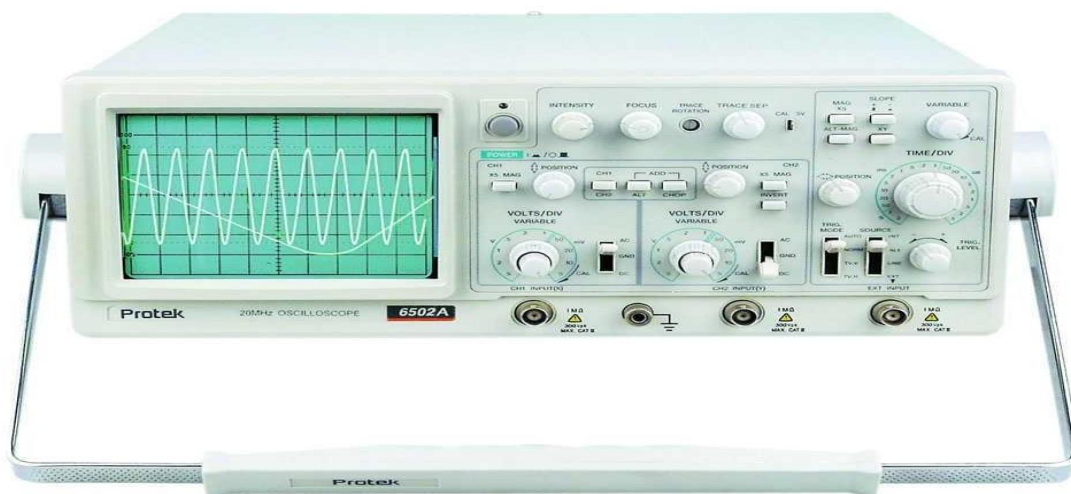
#### Cathode Ray Oscilloscope

An oscilloscope is a test instrument which allows us to look at the 'shape' of electrical signals by displaying a graph of voltage against time on its screen. It is like a voltmeter with the valuable extra function of showing how the voltage varies with time. A graticule with a 1cm grid enables us to take measurements of voltage and time from the screen.

The graph, usually called the trace, is drawn by a beam of electrons striking the phosphor coating of the screen making it emit light, usually green or blue. This is similar to the way a television picture is produced.

Oscilloscopes contain a vacuum tube with a cathode (negative electrode) at one end to emit electrons and an anode (positive electrode) to accelerate them so they move rapidly down the tube to the screen. This arrangement is called an electron gun. The tube also contains electrodes to deflect the electron beam up/down and left/right.

The electrons are called cathode rays because they are emitted by the cathode and this gives the oscilloscope its full name of cathode ray oscilloscope or CRO. A dual trace oscilloscope can display two traces on the screen, allowing us to easily compare the input and output of an amplifier for example. It is well worth paying the modest extra cost to have this facility.



#### Setting up an oscilloscope:

Oscilloscopes are complex instruments with many controls and they require some care to set up and use successfully. It is quite easy to 'lose' the trace off the screen if controls are set wrongly.

There is some variation in the arrangement and labeling of the many controls. So, the following instructions may be adapted for this instrument.

1. Switch on the oscilloscope to warm up (it takes a minute or two).
2. Do not connect the input lead at this stage.
3. Set the AC/GND/DC switch (by the Y INPUT) to DC.



4. Set the SWP/X-Y switch to SWP (sweep).
5. Set Trigger Level to AUTO.
6. Set Trigger Source to INT (internal, the y input).
7. Set the Y AMPLIFIER to 5V/cm (a moderate value).
8. Set the TIMEBASE to 10ms/cm (a moderate speed).
9. Turn the time base VARIABLE control to 1 or CAL.
10. Adjust Y SHIFT (up/down) and X SHIFT (left/right) to give a trace across the middle of the screen, like the picture.
11. Adjust INTENSITY (brightness) and FOCUS to give a bright, sharp trace.

### Connecting an oscilloscope:

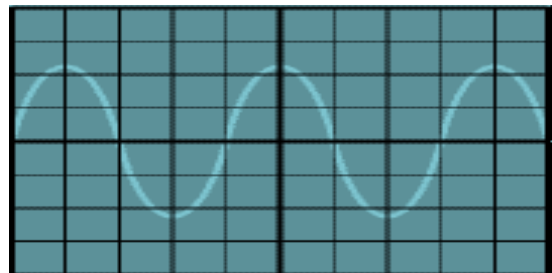
The Y INPUT lead to an oscilloscope should be a co-axial lead and the figure 4 shows its construction. The central wire carries the signal and the screen is connected to earth (0V) to shield the signal from electrical interference (usually called noise).

Most oscilloscopes have a BNC socket for the y input and the lead is connected with a push and twist action, to disconnect we need to twist and pull. Professionals use a specially designed lead and probes kit for best results with high frequency signals and when testing high resistance circuits, but this is not essential for simpler work at audio frequencies (up to 20 kHz).

### Obtaining a clear and stable trace:

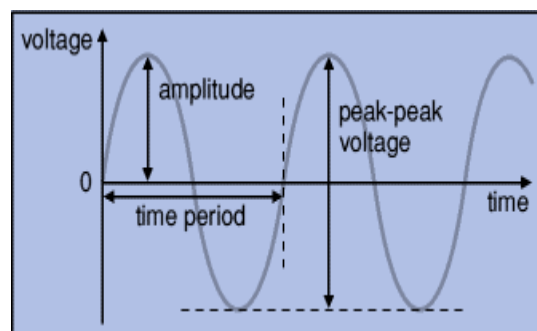
Once if we connect the oscilloscope to the circuit, it is necessary to adjust the controls to obtain a clear and stable trace on the screen in order to test it.

- The Y AMPLIFIER (VOLTS/CM) control determines the height of the trace. Choose a setting so the trace occupies at least half the screen height, but does not disappear off the screen.
- The TIMEBASE (TIME/CM) control determines the rate at which the dot sweeps across the screen. Choose a setting so the trace shows at least one cycle of the signal across the screen. Steady DC input signal gives a horizontal line trace for which the time base setting is not critical.
- The TRIGGER control is usually best left set to AUTO.
- The trace of an AC signal with the oscilloscope controls correctly set is as shown in Figure



### Measuring voltage and time period:

The trace on an oscilloscope screen is a graph of voltage against time. The shape of this graph is determined by the nature of the input signal. In addition to the properties labeled on the graph, there is frequency which is the number of cycles per second. The diagram shows a sine wave but these properties apply to any



signal with a constant shape.

- **Amplitude** is the maximum voltage reached by the signal. It is measured in volts.
- **Peak voltage** is another name for amplitude.
- **Peak-peak voltage** is twice the peak voltage (amplitude). When reading an oscilloscope trace it is usual to measure peak-peak voltage.
- **Time period** is the time taken for the signal to complete one cycle. It is measured in seconds (s), but time periods tend to be short so milliseconds (ms) and microseconds ( $\mu\text{s}$ ) are often used.  $1\text{ms} = 0.001\text{s}$  and  $1\mu\text{s} = 0.000001\text{s}$ .  $\text{Time period} = 1/\text{Frequency}$
- **Frequency** is the number of cycles per second. It is measured in hertz (Hz), but frequencies tend to be high so kilohertz (kHz) and megahertz (MHz) are often used.  $1\text{kHz} = 1000\text{Hz}$  and  $1\text{MHz} = 1000000\text{Hz}$ .  $\text{Frequency} = 1/\text{Time period}$

**Measuring Voltage:** Voltage is shown on the vertical y-axis and the scale is determined by the Y AMPLIFIER (VOLTS/CM) control. Usually peak-peak voltage is measured because it can be read correctly even if the position of 0V is not known. The amplitude is half the peak-peak voltage.  $\text{Voltage} = \text{distance in cm} \times \text{volts/cm}$

**Measuring Time period:** Time is shown on the horizontal x-axis and the scale is determined by the TIMEBASE (TIME/CM) control. The time period (often just called period) is the time for one cycle of the signal. The frequency is the number of cycles per second,  $\text{frequency} = 1/\text{time period}$ .

$\text{Time} = \text{distance in cm} \times \text{time/cm}$

#### User Manual – 4 Function Generator

A function generator is a device that can produce various patterns of voltage at a variety of frequencies and amplitudes. It is used to test the response of circuits to common input signals. The electrical leads from the device are attached to the ground and signal input terminals of the device under test.



General function generators can be used to generate the following waveforms:

- Sinusoidal
- Square
- Triangular
- Pulse

#### Setting the values on the function generator:


- Double click on the function generator to open the function generator panel.
- Select the waveform of your choice by clicking on the waveform selector buttons.
- Select the property associated with the waveform such as frequency, amplitude or offset etc. by clicking on the waveform property selector button.

- Use the arrow keys to move the cursor to the desired position
- Use the number pad to change the value or alternatively you can use the knob
- After changing one property, move on to the next property


**Note:**

After changing properties of a waveform (e.g. frequency, amplitude of Sine), if you wish to select other waveform (e.g. Square), will set all the properties of Sine to their default values.


**Generating Sine wave:**

- Click the  sine button.
- Click and select frequency, change it to the desired value
- Similarly, change amplitude, offset and phase

**Generating Square wave:**

- Click the  square button.
- Click and select frequency, change the value to the desired value
- Similarly, change amplitude, duty cycle and phase

**Generating a pulse train:**

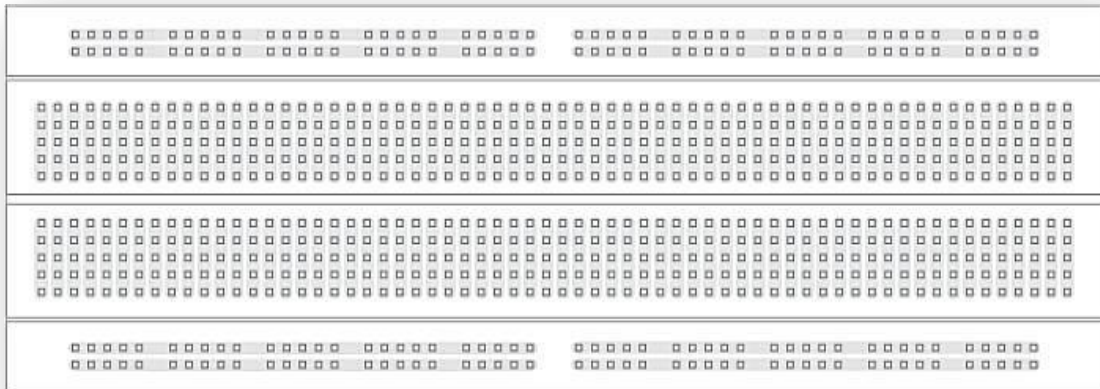
- Click the  pulse button
- Click and select amplitude, change the value to the desired value
- Similarly, change other parameters as well.

The amplitude control on a function generator varies the voltage difference between the high and low voltage of the output signal. The direct current (DC) offset control on a function generator varies the average voltage of a signal relative to the ground.

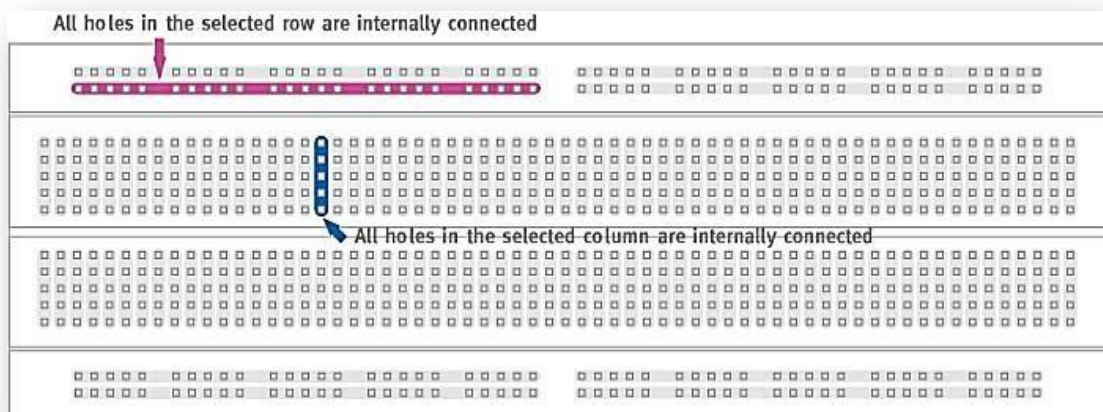
The frequency control of a function generator controls the rate at which output signal oscillates. On some function generators, the frequency control is a combination of different controls. One set of controls chooses the broad frequency range (order of magnitude) and the other selects the precise frequency. This allows the function generator to handle the enormous variation in frequency scale needed for signals.

## User Manual – 5 Breadboards

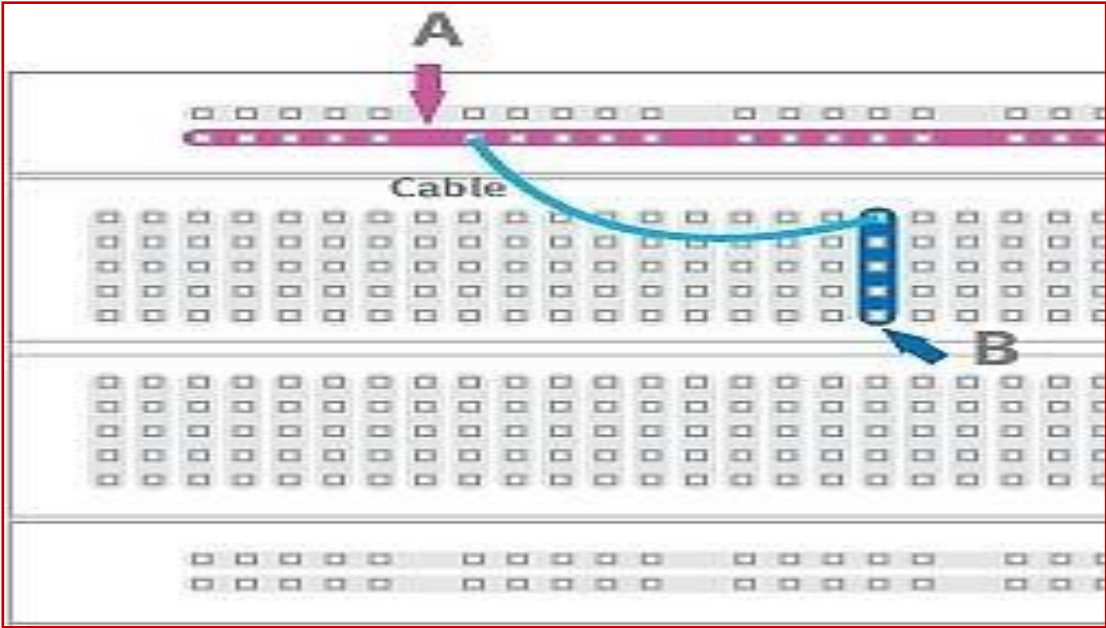
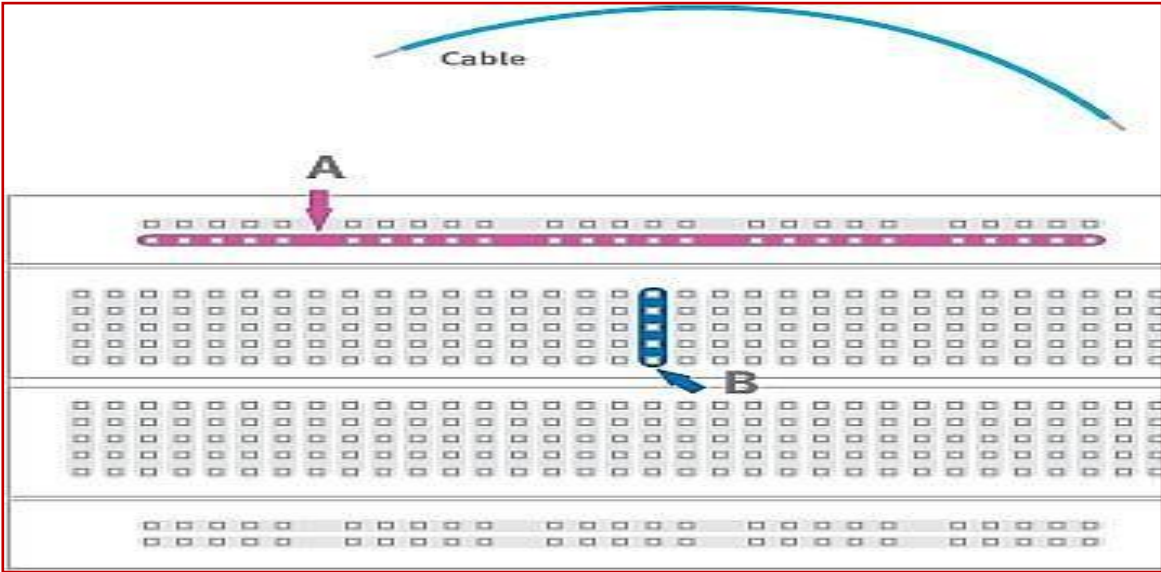
A breadboard is a solder less device for temporary prototype with electronics and test circuit designs. Most electronic components in electronic circuits can be interconnected by inserting their leads or terminals into the holes and then making connections through wires where appropriate. The breadboard has strips of metal underneath the board and connect the holes on the top of the board. The metal strips are laid out as shown below. Note that the top and bottom rows of holes are connected horizontally and split in the middle while the remaining holes are connected vertically.



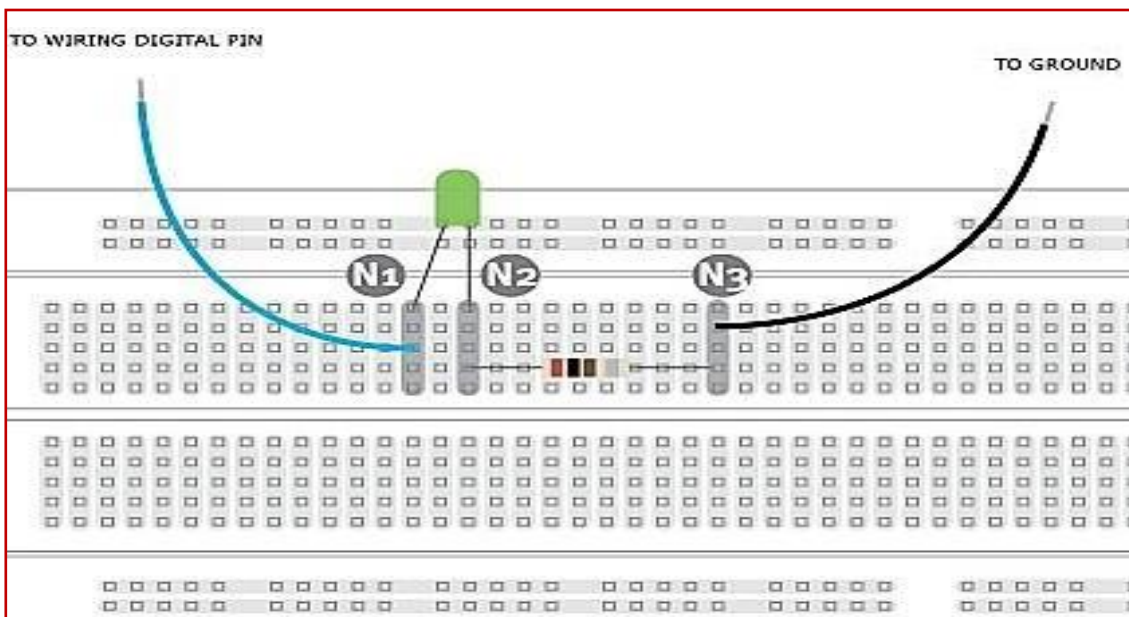
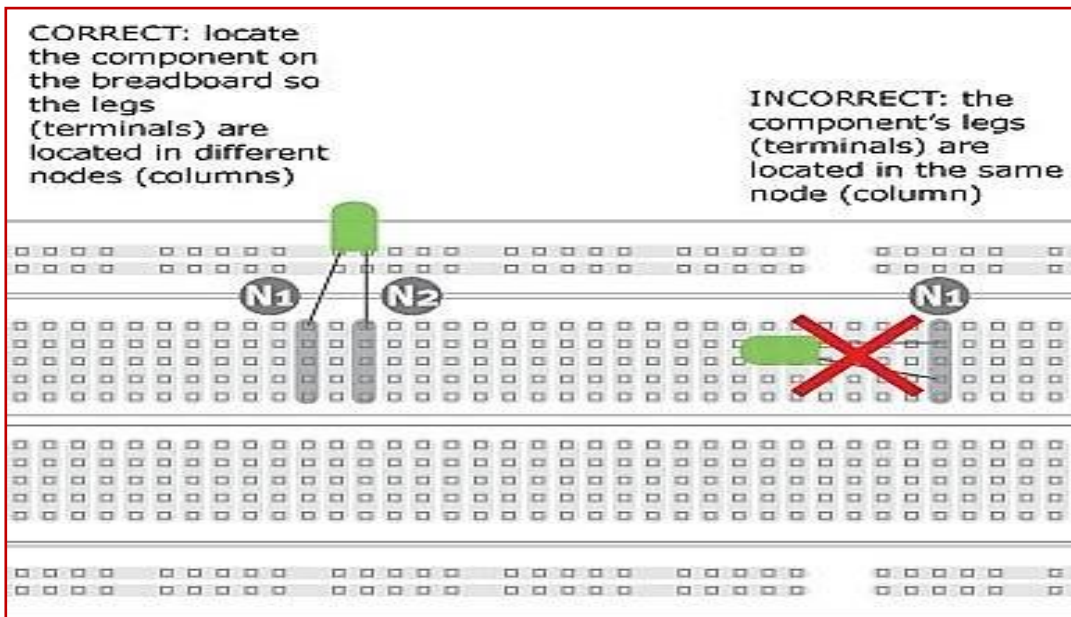
Note how all holes in the selected row are connected together, so the holes in the selected column. The set of connected holes can be called a node:



To interconnect the selected row (node A) and column (node B) a cable going from any hole in the row to any hole in the column is needed:



Now the selected column (node B) and row (node A) are interconnected:



## User Manual – 6

### PSpice simulator tool

In order to ensure a successful circuit design and mitigate costly and potentially dangerous design flaws, careful planning and evaluation must occur at every stage of the circuit design process. Circuit simulation provides a cost-effective and efficient method for identifying faults before moving to the more expensive and time-consuming prototyping stage. Including simulation in the design process reduces design errors and speeds the design cycle by allowing you to predict and better understand circuit behavior. The main purpose of simulation is to predict and understand the behavior of electronic circuits. PSpice is a program that simulates electronic circuits on your PC.

#### Limitations of simulation:

While a prototype helps you to verify and validate your design in the real world, simulation helps you catch design errors before spending money and time on prototyping.

#### Orcad 9.2 Lite Edition Installation:

1. Insert Cadence CD into CD-ROM drive
2. Select Products to install
  - Capture – Schematic entry application – it must be installed
  - Capture CIS – Should be grayed out.
  - PSpice – For conducting mixed-signal analog and digital simulations
  - Layout– For creating PC Board layouts from schematics

Then follow the instructions as it appears on the monitor and complete the installation

#### The steps to simulation:

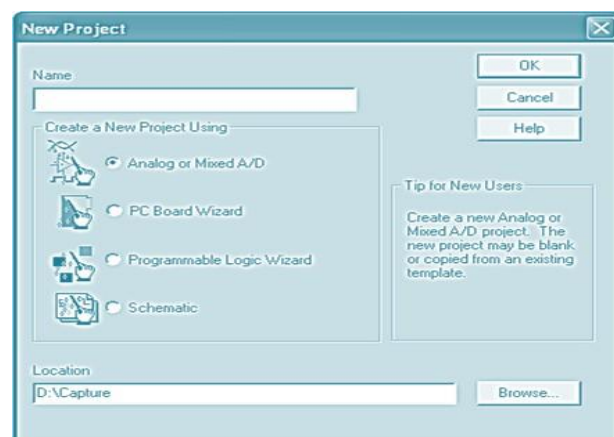
1. Create a simulation project
2. Draw schematic to simulate
3. Establish a simulation profile
4. Set up simulation type
5. Simulate circuit
6. Analyze results in Probe

#### General procedure for all experiments:

1. Select the required components from the menu.
2. Place all the required components in the schematic.
3. Simulate the circuit using RUN option from the menu.
4. Observe the waveforms from the output

#### To start a simulation session:

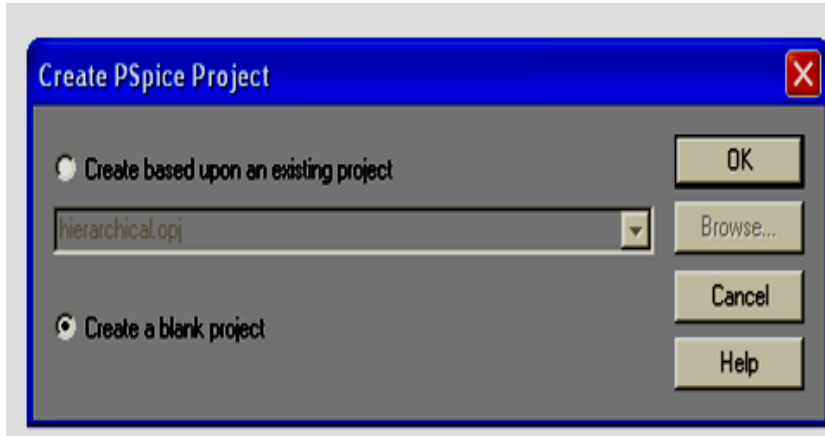
1. On the start menu select ORCAD FAMILY RELEASE 9.2 LITE EDITION - CAPTURE LITE EDITION
2. Once the capture window appears, select FILE - NEW – PROJECT. The following window appears:



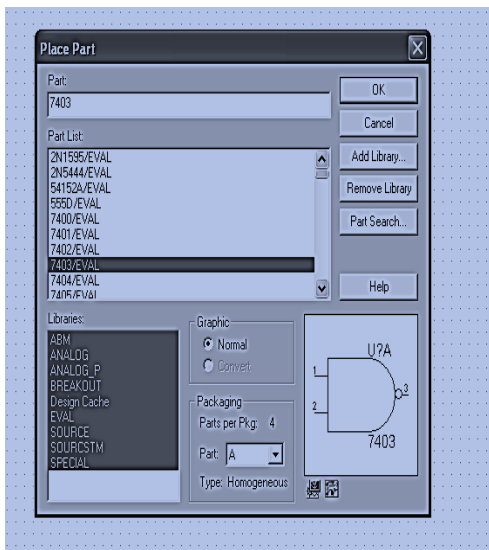
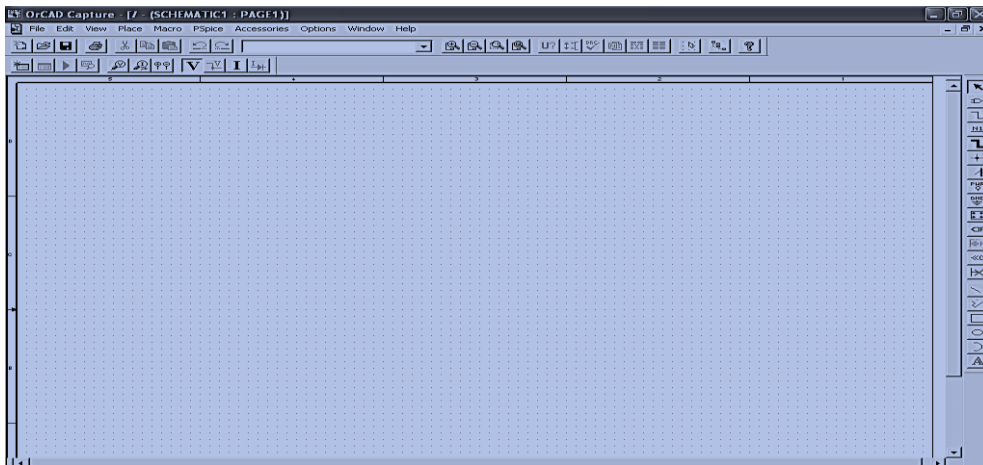
3. Give the project a descriptive name (spaces can be included).
4. Select ANALOG OR MIXED-SIGNAL CIRCUIT WIZARD.
5. Specify a location where the project is to be stored

6. Click OK.

The following window appears:



7. Select Create a blank project and click OK. The following window appears:



8. To place parts, click PLACE PART (Shift+P).

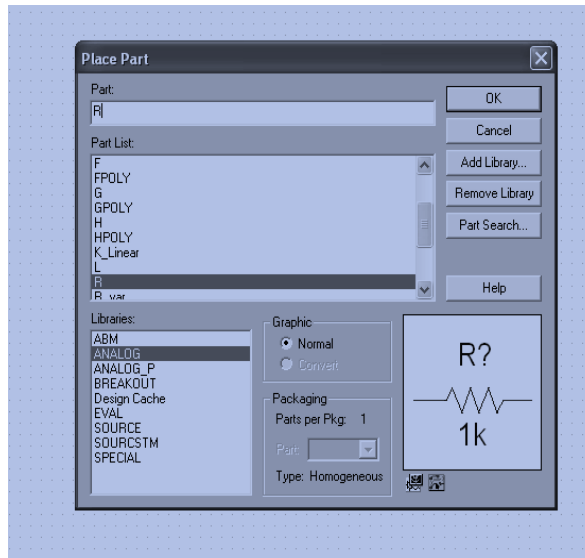
Then the following window appears:

If Libraries are not appearing in the window then click Add Library. The library files will generally be available in the following path by default.

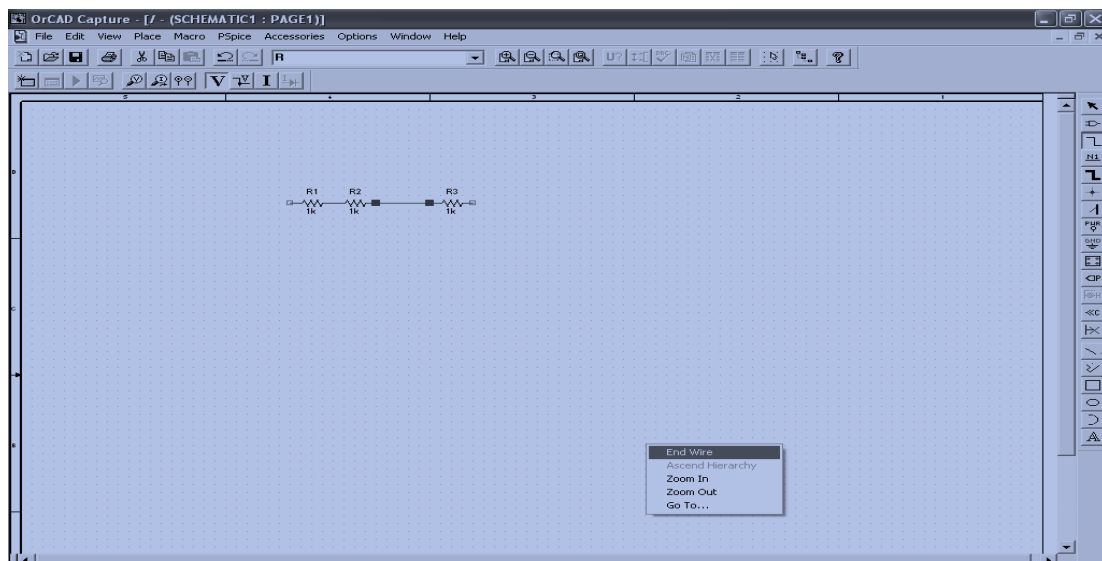
C:\Program Files\Orcad\_Demo \Capture \ Library\Ps spice. Select all Library files by pressing Ctrl A and then press Open. All the Library files will appear in the window.



9. Select the part you wish to place in the schematic. Insert as many as needed

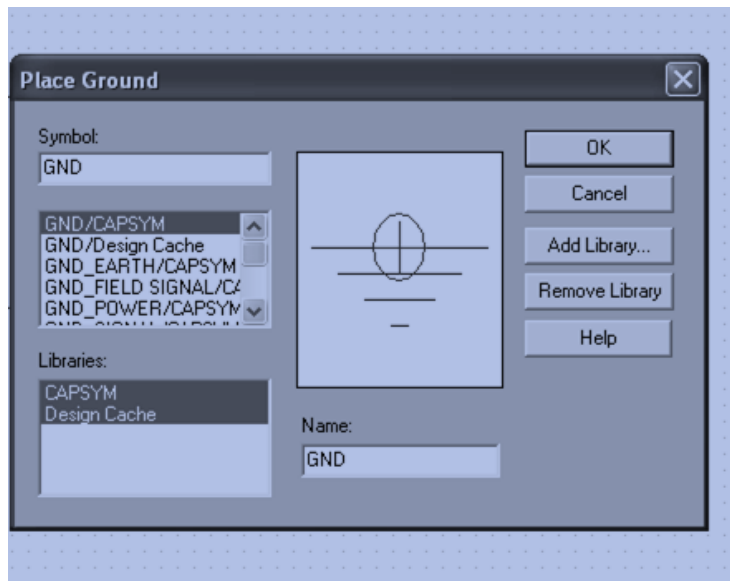


10. Right Click and select END MODE to stop inserting parts



11. To wire parts together, click Place Wire Icon from the Right hand side vertical Icons list (Shift+W). Place cursor over boxes at ends of parts and draw wires connecting parts. When done, right click and select End Wire.

12. To insert a ground node, click Place – Ground Icon. Window appears with caption Place Ground with only ground nodes available for selection.

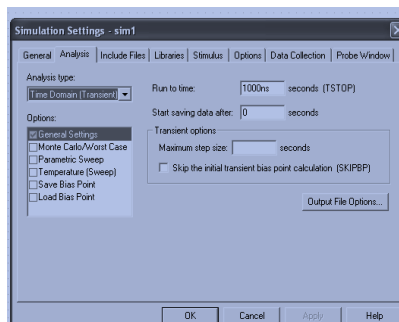


Always select 0/SOURCE for the ground node of an analog circuit – every analog circuit must contain at least one 0 ground. This is not a requirement for digital circuits.

13. To change component values that are displayed. Double click the displayed value. Change the desired value in the dialog box that appears.

#### To set up a simulation profile:

1. Select PSPICE ---NEW SIMULATION PROFILE from the menu.
2. Give a descriptive name to the type of simulation.
3. Select the desired parameters for the particular circuit and then click OK.
4. Place voltage, current, and power markers from the PSPICE --- MARKERS menu where needed.



5. Click PSPICE -- RUN.

## User Manual – 7

## VHDL

VHDL stands for **Very High Speed Integrated Circuit Hardware Description Language**. It describes the behavior of an electronic circuit or system, from which the physical circuit or system can then be implemented.

VHDL was originally intended to serve 2 main purposes-

1. It was used as a **documentation** language for describing the structure of complex digital circuits.
2. VHDL provides features for modeling the **behavior** of a digital circuit.

**General Features of VHDL:**

1. The language can be used as an exchange medium between chip vendors and CAD tool user and can be used as communication medium between CAD and CAE tools.
2. It supports hierarchy.
3. It is not a case sensitive language.
4. It is strongly type checked language.
5. It provides design portability and flexible design methodologies: top down, bottom up or mixed
6. It supports both synchronous and asynchronous timing models.
7. Nominal Propagation delays, min-max delays, setup and hold timing constraint and spike detection can be described in this language.

**Usage of the Tool:**

It is one of most popular software tool used to synthesize VHDL code. This tool includes many steps. To make user feel comfortable with the tool the steps are given below:-

1. Select NEW PROJECT in FILE MENU.
  - a. Enter following details as per your convenience
  - b. Project name : sample (should be same as the entity name in your VHDL code)
  - c. Project location : C:\example( As per convenience use default)
  - d. Top level module : HDL
2. In **NEW PROJECT** dropdown Dialog box, Choose your appropriate device specification. Example is given below:
  - a. Device family : cyclone
  - b. Device : EP1C6Q240
  - c. Package : PQFP
  - d. Pincount :240
  - e. Speed grade 8
3. On File Drop down menu choose new Vhdl file. Type the Vhdl code Under the Processing Drop down menu
4. Choose Start compilation
5. If there are errors go back to the VHDL code and correct it. Once the compilation is successful
6. Under the processing drop down box select simulator tool select the simulator mode to functional and click on generate functional simulation netlist. we Get the success message.
7. Under simulator tool click on open. In the empty location right click . Click on insert on NODE or BUS. Then click on NODE finder. In the window opened select pins to unassigned. Click on List . IT will list all the Net list select all and click ok.

8. The input and output appears give appropriate input.
9. On the simulator tool click on start simulation. Simulation success message will be prompted.
10. On simulator tool click on report to see the output.

**Note:**

Create a new project for every new VHDL code.

The primary data type `std_logic` (standard unresolved logic) consists of nine character literals in the following order:

'U' - uninitialized

'X' - strong drive, unknown logic value

'0' - strong drive, logic zero

'1' - strong drive, logic one

'Z' - high impedance

'W' - weak drive, unknown logic value

'L' - weak drive, logic zero

'H' - weak drive, logic one

'-' - don't care

**Sample Viva Questions**

1. Why operational amplifier is called by its name?
2. Explain the advantages of OPAMP over transistor amplifiers.
3. List the OPAMP ideal characteristics.
4. Give the symbol of OPAMP
5. Explain the various applications of OPAMP
6. Define UTP and LTP
7. Mention the applications of schmitt trigger
8. What is a square wave generator/ Regenerative comparator?
9. Give the hysteresis curve of a schmitt trigger
10. What is a bipolar and unipolar devices? Give examples
11. Define resolution
12. Explain the need of D/A and A/D converters.
13. List the different types of A/D and D/ A converters
14. What is a multivibrators?
15. What is a bistable multivibrators?
16. Give the applications of monostable and astable multivibrators
17. Explain the working of 555 timer as astable and monostable multivibrator
18. Why astable multivibrator is called as free running multivibrato
19. Define duty cycle.
20. List the applications of 555 timer
21. Explain 555 timer as astable multivibrator to generate a rectangular wave of duty cycle of less than 0.5
22. Define a logic gate.
23. What are basic gates?
24. Why NAND and NOR gates are called as universal gates?
25. State De morgans theorem
26. Give examples for SOP and POS
27. Explain how transistor can be used as NOT gate
28. Realize logic gates using NAND and NOR gates only
29. List the applications of EX-OR and EX~NOR gates
30. What is a half adder?
31. What is a full adder?
32. Differentiate between combinational and sequential circuits. Give examples
33. Give the applications of combinational and sequential circuits
34. Define flip flop
35. What is an excitation table?
36. What is race around condition?
37. How do you eliminate race around condition?
38. What is minterm and max term?
39. Define multiplexer/ data selector
40. What is a demultiplexer?
41. Give the applications of mux and demux
42. What is a encoder and decoder?
43. Compare mux and encoder
44. Compare demux and decoder
45. What is a priority encoder?
46. What are counters? Give their applications.
47. Compare synchronous and asynchronous counters
48. What is modulus of a number?

49. What is a shift register?
50. What does LS stand for, in 74LS00?
51. What is positive logic and negative logic?
52. What are code converters?
53. What is the necessity of code conversions?
54. What is gray code?
55. Realize the Boolean expressions for
  - a Binary to gray code conversion
  - b Gray to binary code conversion

**Note:**

All the above questions are the most commonly asked and the depth of it may vary based on the answers which you give during the viva voice procedure.

*All the very best!*