**CE252 DIGITAL ELECTRONICS**

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

**Department of Computer Science & Engineering**

**PRACTICAL - 1**

**AIM:** **Study of logic gates (AND, OR, NOT, NAND, NOR, Ex•OR).**

**APPARATUS:** connecting wires, power supply, bread board, ICs as follow

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Component** | **Specification** | **Quantity** |
| **1** | AND Gate | IC 7408 | 1 |
| **2** | OR Gate | IC 7432 | 1 |
| **3** | NOT Gate | IC 7404 | 1 |
| **4** | NAND Gate | IC 7400 | 1 |
| **5** | NOR Gate | IC 7402 | 1 |
| **6** | X-OR Gate | IC 7486 | 1 |

**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, XOR are known as universal gates. Basic gates can be obtained from all this gate.

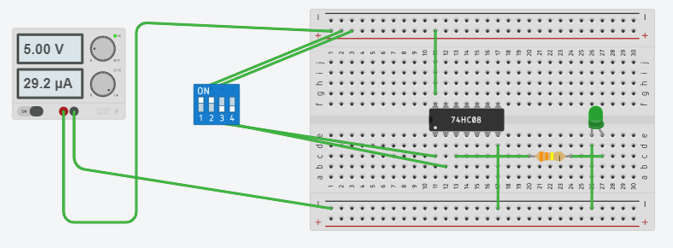
**AND Gate:**

* The AND gate performs a logical multiplication commonly known as AND function. The output is high only when both the input are either one high or one low. When both the input are high the output is low level.

# Symbol:

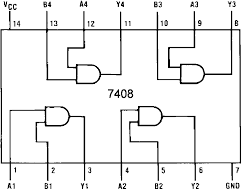


# Tinkercad circuit:



**Observation Table:**

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

**IC Diagram:**

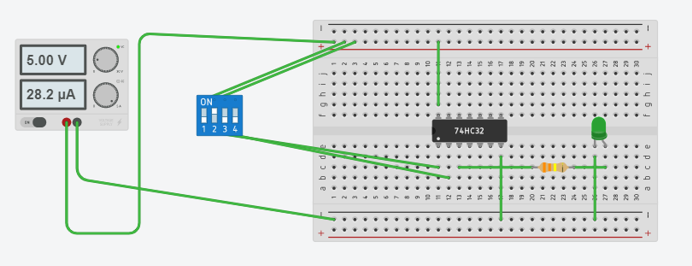
**OR Gate:**

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

# Symbol:

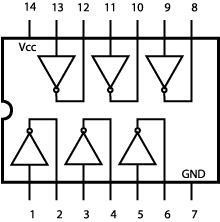
# 

# Tinkercad circuit:



**Observation Table:**

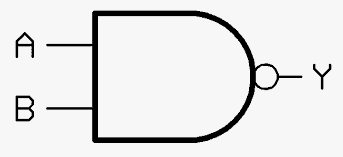
|  |  |
| --- | --- |
| Input | Output |
| 0 | 1 |
| 1 | 0 |

**IC Diagram:**

**NAND Gate:**

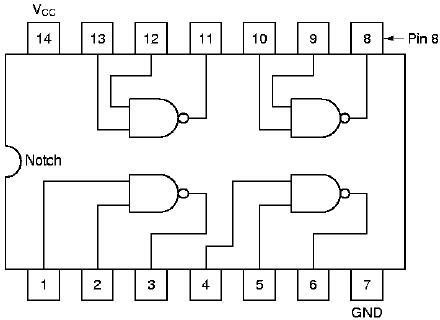
* The NAND gate is a contraction of AND•NOT. The output is high when both inputs are low and any one of the input is low. The output is low level when the input are high.

# Symbol:

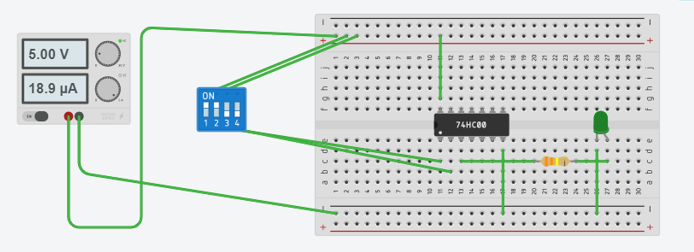


**Observation Table:**

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

**IC Diagram:**

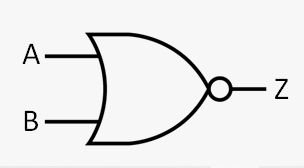
**Tinkercad circuit:**



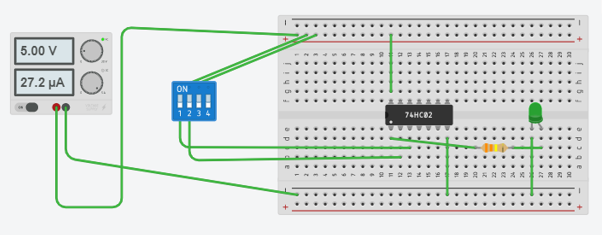
**NOR Gate:**

* The NOR gate is contraction of OR•NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

# Symbol:

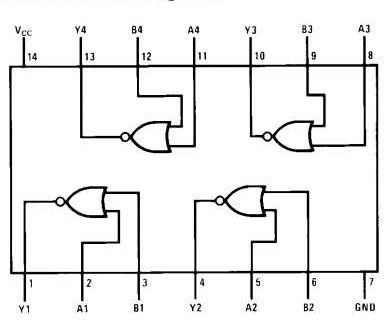


# Tinkercad circuit:



**Observation Table:**

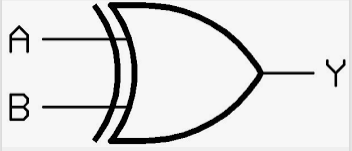
|  |  |  |
| --- | --- | --- |
| Input | Input | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**IC Diagram:**

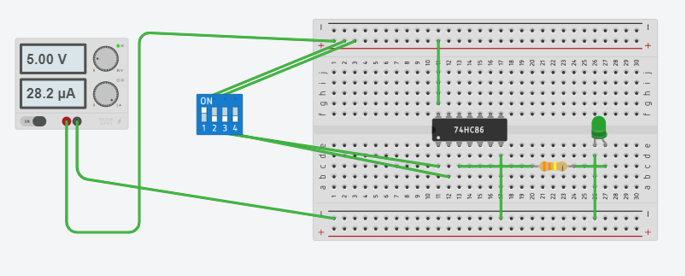
**X•OR Gate:**

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

# Symbol:

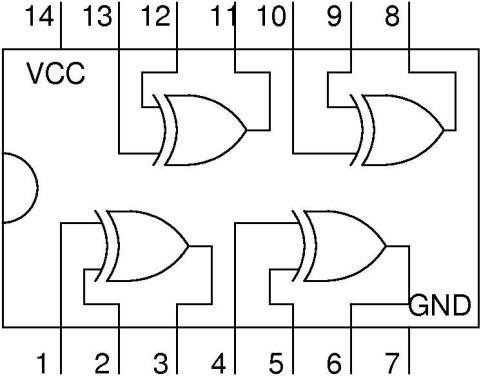


**Tinkercad circuit:**



**Observation Table:**

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

**IC Diagram:**

**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.

# CONCLUSION:

# In this lab we learnt any Boolean expression can be realized using NOT, AND, OR, NAND, NOR, EXOR gate.

# I explored the function of basic logic gates.

# I learned how to implement them on integrated circuits.

# I tested the output voltage of the breadboard circuits using multimeter and ensured the results with the truth table of the logic gate tested.

**PRACTICAL – 2**

**AIM:** Draw a circuit diagram and Verify the truth table of Ex•OR & Ex•NOR gates.

**APPARATUS:** connection wires, power supply, power project board, LED, ICs

# THEORY:

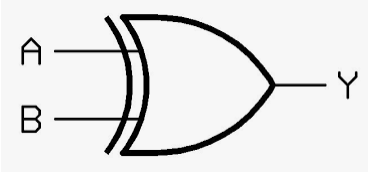
**EX-OR**

* The Ex•OR gate is a two input, one output logic circuit whose output assumes a logic 1 state when one and only one of its two inputs assumes a logic 1 state. Under the conditions when both the inputs assume the logic 0 state, or when both the inputs assume the logic 1 state, the output assumes a logic 0 state. Since an Ex•OR gate produces an output 1 only when the inputs are not equal, it is called an anti•coincidence gate or inequality detector.



2 Input XOR gate using AOI logic:

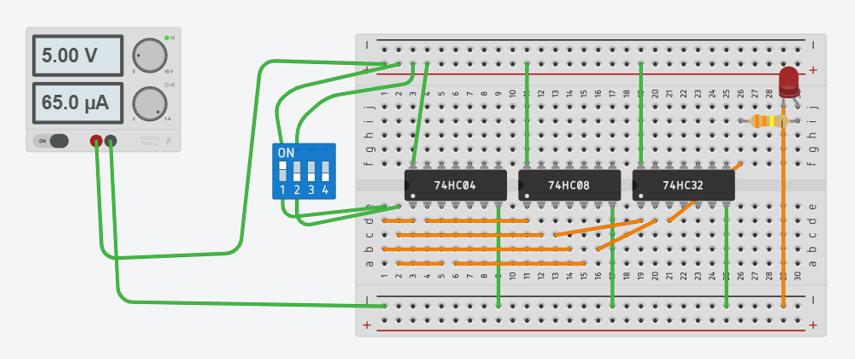
**Symbol:**



**Observation Table:**

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

**Tinkercad circuit:**



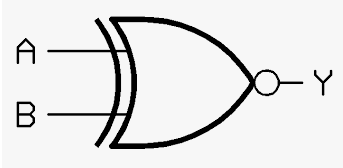
**Ex – NOR**

* The Ex•NOR gate is a two input, one output logic circuit whose output assumes a logic 0 state or when both the inputs assume a logic 1 state. The output assumes a logic 0 state, when one of the inputs assumes a 0 state and the other a 1 state. It is also known as a coincidence gate or equality detector.

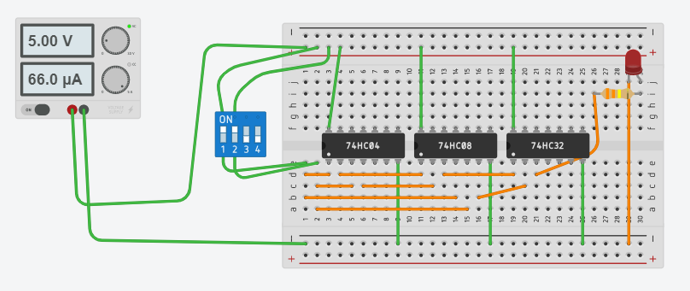
A ʘ B = AB + A'B'

2 Input Ex•NOR gate using AOI logic:

**Symbol:**



**Tinkercad circuit:**



**Observation Table:**

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

**Procedure:**

1. Connect the circuit according to circuit diagram.
2. Apply different input combination at the input pin of ICs.
3. Verify the truth table of Ex•OR and Ex•NOR gate for different input combinations.

**CONCLUSION:**

* In this Practical we learnt about EX-OR and EX-NOR Gate. In EX-OR gate input are not equal then an output is 1. In EX-NOR gate input are equal then an output is 1.
* I tested the output voltage of the breadboard circuits using multimeter and ensured the results with the truth table of the X-OR gate and X-NOR gate tested.

**PRACTICAL – 3**

**AIM:** Verify the operation of NAND & NOR gate as universal gates.

**APPARATUS:** Connection wires, power supply, power project board, LED, ICs.

# THEORY:

# The AND, OR and NOT gates are the basic building blocks of digital system. These gates are known as basic gates. Any digital circuit of any complexity can be built using only these three gates. A universal gate is a gate which alone can be used to build any logic circuit. As the basic gates can be realized using only NAND gates or using only NAND gates or using only NOR gates. So NAND gate and NOR gate are also known as Universal gates. The following design diagrams are shown the realization of AND, OR and NOT function using either only NAND gates or only NOR gates.

# Procedure:

# 1) Mount ICs 7400, 7402 on the power project board.

# 2) Connect pin number 7 and 14 of all ICs to ground and +5V supply

# respectively.

# 3) Make the connection as shown in the logic diagram.

# 4) Verify the truth table of all the data.

# CONCLUSION:

# In this practical we learned how NAND and NOR gate can be used as any gate/universal. And we also learned that why NAND gate and NOR gates are called universal gates.

**PRACTICAL – 4**

**AIM:** Study of 4 variable K-map and verify using example.

**APPARATUS:** Connection wires, power supply, power project board.

# THEORY: A map method provides a simple straight procedure for minimizing boolean functions. This method may be regulated either as a pictorial form of a truth or as an extension of the venn•diagram. The map method, first propose by veitch and slightly modified by Karnaugh. So it also known as the “Veitch diagram” or the “Karnaugh map”. The map is a diagram made up of squares. Each square represents one min•term. Since any Boolean function can be expressed as a sum of min terms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose min•terms are included in the function.

# Procedure:

# Solve the problem given by lab instructor with help of basic logic Gates.

# Find out the proper input and output.

# Use pin diagram and make proper connection as per requirement.

# Measure the various output in LED on/off condition with different input condition. Note down the observation table.

# Compare truth table with observation table and write conclusion.

# Example:

# 

# CONCLUSION:

# In this practical we learned how to draw k-map and Derive equation from it and further creating circuit from the equation.

**PRACTICAL – 5**

**AIM:** Implement half adder and half subtractor circuit.

**APPARATUS:** Connecting wires, power project board, LED, power supply IC [7408•AND, 7404•NOT, 7402•OR]

# THEORY:

# Logic Circuits for digital system may be combinational or sequential. The output of a combinational circuit depends on its present inputs only. Combinational circuit perform a specific information processing operation fully specified logically by a set of boolean functions. The example of combinational circuit is Adder, Subtractor, Multiplexer, Demultiplexer, Encoder, Decoder, Magnitude Comparator etc.

# A half adder can add two bits. It has two inputs, generally labelled A and B, and two outputs, the sum S and carry C.

# S is the two•bit XOR of A and B, and C is the AND of A and B. Essentially the output of a half adder is the sum of two one•bit numbers, with C being the most significant of these two outputs.

# A half adder is a logical circuit that performs an addition operation on two one•bit binary numbers. The half adder outputs a sum of the two inputs and a carry value. The drawback of this circuit is that in case of a multibit addition, it cannot include a carry.

# A half subtractor is a logical circuit that performs a subtractor operation on two one bit binary numbers. The half subtractor outputs a difference of the two inputs and a borrow value. In half subtractor circuit difference is the XOR operation of two inputs and borrow is the AND operation of A’B

# 

# 

# Procedure:

# i) Do the connection as per Combinational logic diagram for various input data.

# ii) Apply proper input condition and observe the output information of using DMM.

# iii) Compare theoretical data with observation and write conclusion.

**OBSERVATION TABLE:**

* **Half Adder**

|  |  |  |  |
| --- | --- | --- | --- |
| Input A | Input B | Sum | Carry (Cout) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

* **Half Substractor**

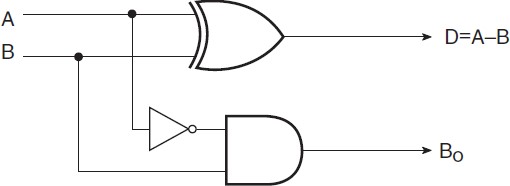
|  |  |  |  |
| --- | --- | --- | --- |
| Input A | Input B | Difference | Borrow (Cout) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

# CONCLUSION:

# Half Adder:

# 

* **Half Substractor:**



**PRACTICAL – 6**

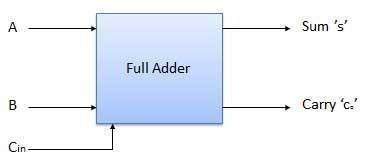
**AIM:** Implement full adder combinational circuit.

**APPARATUS:** Logic Gate ICs, connecting wires, Bread Board, Power supply, LED, DMM

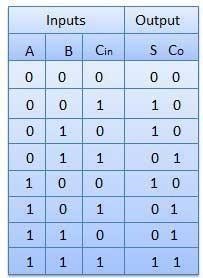
# THEORY:

* Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

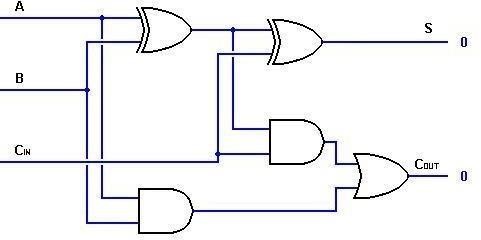
**BLOCK DIAGRAM:**



**TRUTH TABLE:**



**Combinational Logic:**



**Procedure:**

1. Do the connection as per Combinational logic diagram for various input data.
2. Apply proper input condition and observe the output information of using DMM.

iii) Compare theoretical data with observation and write conclusion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input A | Input B | Cin | Sum | Carry(Cout) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

# CONCLUSION: In this practical we learned how to make full adder combinational circuit.

**PRACTICAL – 7**

**AIM:** Show the conversion from BCD to Excess•3.

**APPARATUS:** Logic Gate ICs, connecting wires, Bread Board, Power supply, LED, DMM.

# THEORY:

* We will complete this experiment to code converters by designing an Excess•3 Binary Coded Decimal (BCD) circuit. The term BCD refers to representing the ten decimal digits in binary forms; which simply means to count in binary; see Table below. The Excess•3 system simply adds 3 to each number to make the codes look different. We will not venture to discuss the importance of the Excess•3 BCD system because the discussion would serve too great a distraction from our present purpose and the cost would outweigh the benefit. Suffice it to say that the Excess•3 BCD system has some properties that made it useful in early computers.
* The Excess•3 BCD system is formed by adding 0011 to each BCD value as in Table. For example, the decimal number 7, which is coded as 0111 in BCD, is coded as 0111+0011=1010 in Excess•3 BCD.

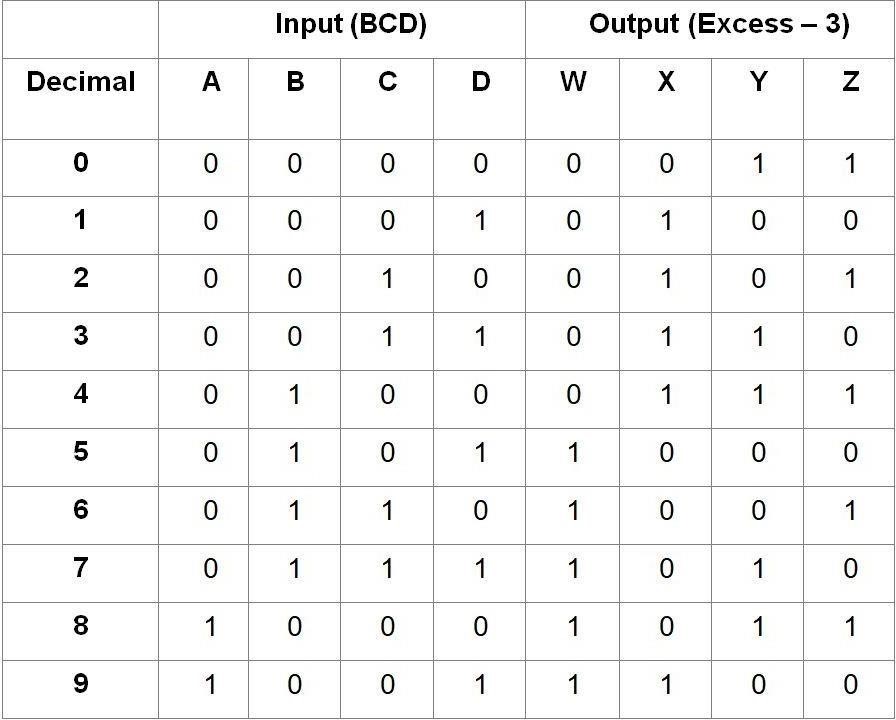
# 

# BCD to Excess•3 Code conversion logic diagram

**PROCEDURE:**

1. Do the connection as per Combinational logic diagram for various input data.
2. Apply proper input condition and observe the output information of using DMM.
3. Compare theoretical data with observation and write conclusion.

**OBSERVATION TABLE:**



# CONCLUSION: In this experiment we learned the conversion from BCD To Excess 3.

**PRACTICAL – 8**

**AIM:** Study 4•bit magnitude comparator and implement it.

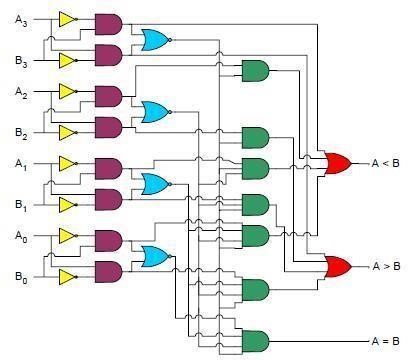
**APPARATUS:** 4 bit magnitude comparator IC (74LS85), Connecting wires, Bread Board, Power supply, LED, DMM.

# THEORY:

# 

# 

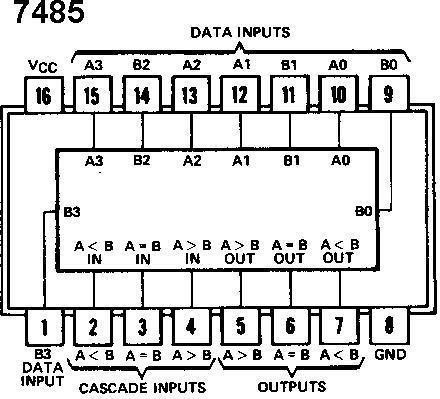
**Logic Diagram:**



**Procedure:**

1. Do the connection as per below pin diagram for various input data or 4 bit number.
2. Apply proper input condition and observe the output information of ledon/off.
3. Compare theoretical data with observation and write conclusion.

**Pin Diagram:**



**Observation:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input A** | | | | **Input B** | | | | **Output** | | |
| **A3** | **A2** | **A1** | **A0** | **B3** | **B2** | **B1** | **B0** | **A>B** | **A=B** | **A<B** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

**CONCLUSION:** We have learned the Implementation of 4-Bit Magnitude Comparator using IC-74LS85.

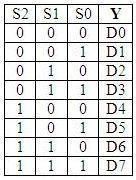
**PRACTICAL – 9**

**AIM:** Implement 8:1 multiplexer.

**APPARATUS:** Multiplexer IC (74LS151), Connecting wires, Bread Board, Power supply, LED, DMM.

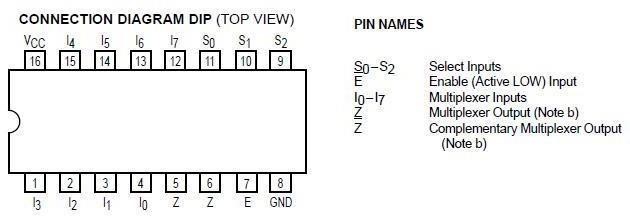
**THEORY:**

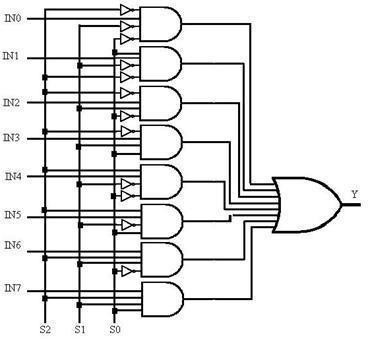
* In electronics , a multiple xeror muxis a device that selects one of several an a log or digital input signal sand for wards the selected input into a single line. A multiplexer of 2n inputs has n select lines, which are used to select which input line to send to the output.
* An electronic multiplexer can be considered as a multiple-input, single-output switch i.e. digitally controlled multi-position switch. The digital code applied at the select inputs determines which data inputs will be switched to output.
* A common example of multiplexing or sharing occurs when several peripheral devices share a single transmission line or bus to communicate with computer. Each device in succession is allocated a brief time to send and receive data. At any given time, one and only one device is using the line. This is an example of time multiplexing since each device is given a specific time interval to use the line.
* In frequency multiplexing, several devices share a common line by transmitting at different frequencies.



Truth Table of 8:1 MUX

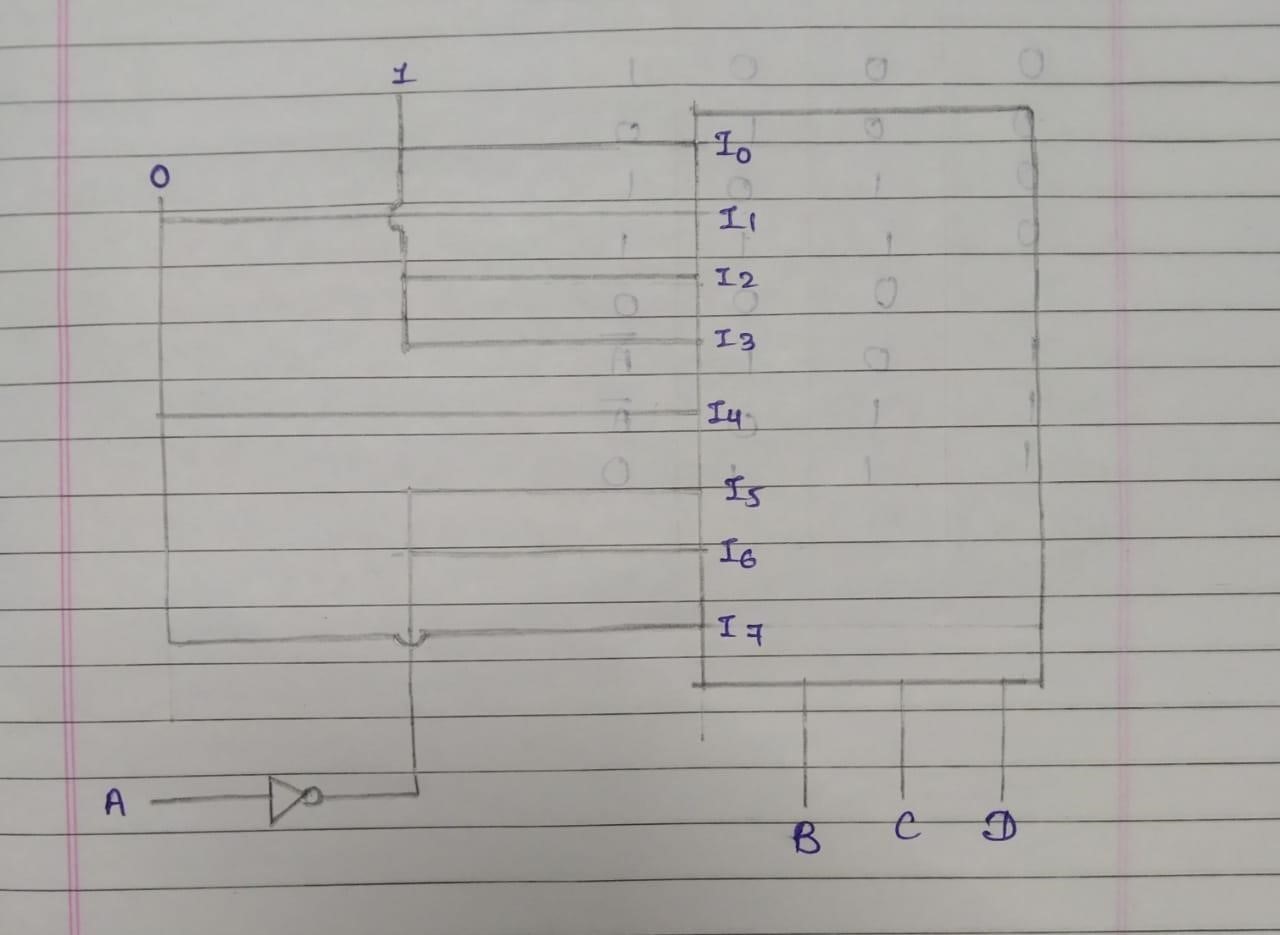
3



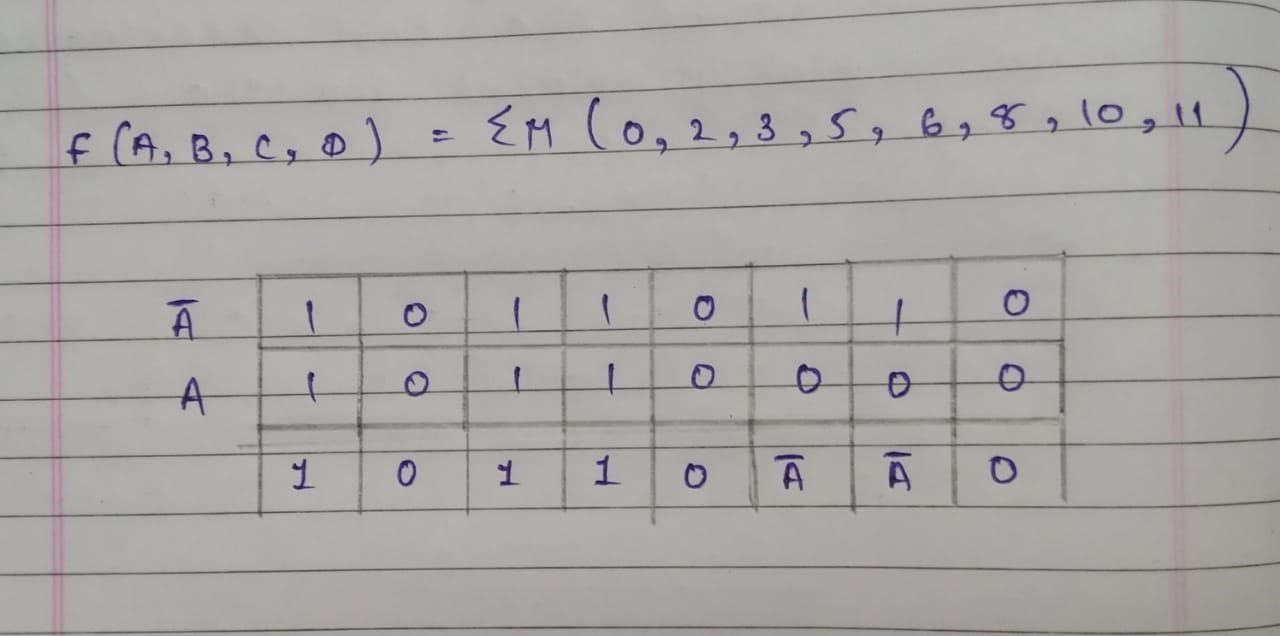


**PROCEDURE:**

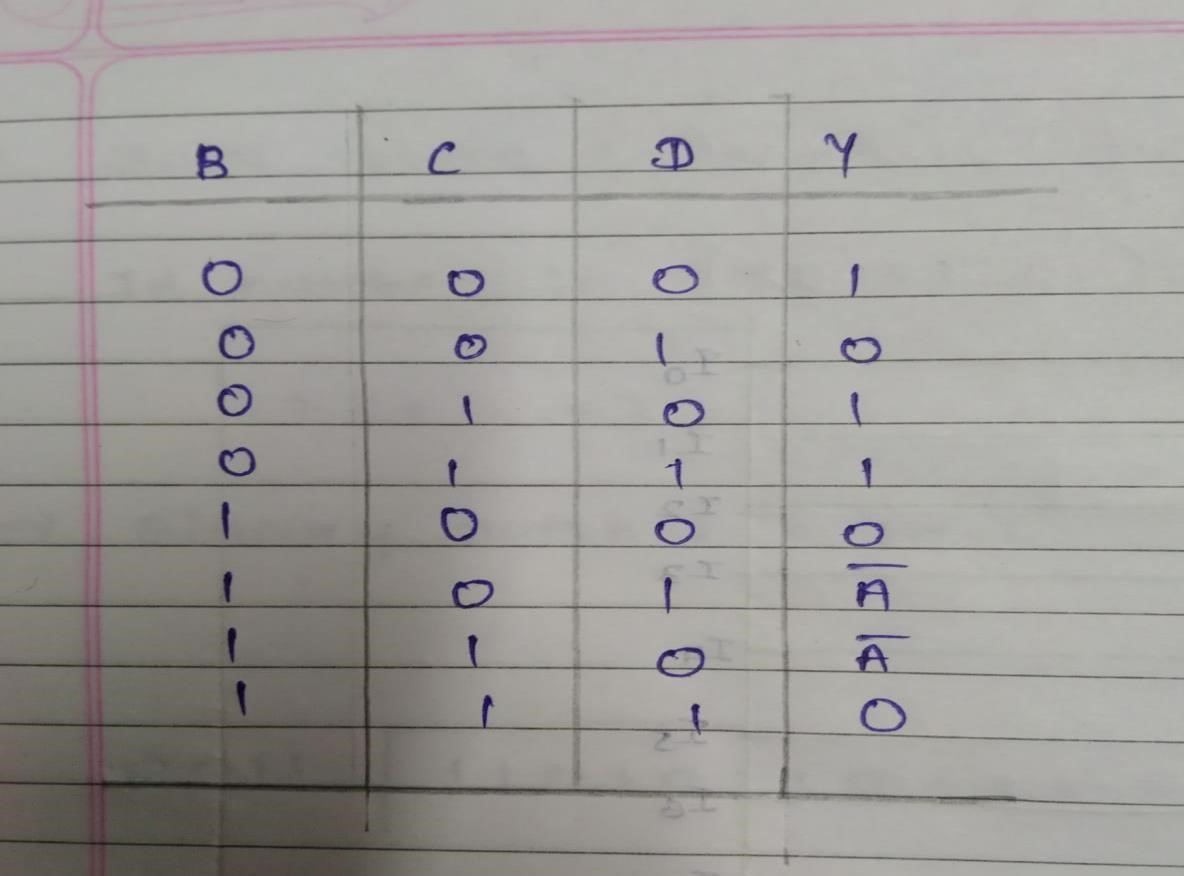
1. Solve the problem given by lab instructor with help of 8x1 Multiplexer using implementation table method.
2. Find out the proper input and output as well as define the select line variable.
3. Use pin diagram and make proper connection as per requirement.
4. Measure the various output in LED on/off condition with different input condition. Note down the observation table.
5. Compare truth table with observation table and write conclusion.



**Example :**



**Observation Table:**



**CONCLUSION:**

* In this practical we learned how to create 8:1 multiplaxer circuit.

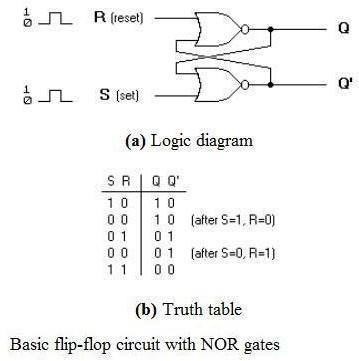
**PRACTICAL – 10**

**AIM:** Demonstrate the operation of RS, JK &D flip•flops. Verify truth table, Excitation table and functional table.

**APPARATUS:** Sigma Flip Flop trainer model•DFF11, Connecting wires

# THEORY:

* Flipflops are actually an application no logic gates.With the help of Boolean logic you can create memory with them. Flip flops can also be considered as the most basic idea of a Random Access Memory [RAM]. When a certain input value is given to them, they will be remembered and executed, if the logic gates are designed correctly. A higher application of flip flops is helpful in designing better electronic circuits.
* The most commonly used application of flipflops is in the implementation of a feedback circuit. As a memory relies on the feedback concept, flip flops can be used to design it.
* There are mainly four types of flip flops that are used in electronic circuits.They are

1. The basic Flip Flop or S•R FlipFlop
2. Delay Flip Flop [D FlipFlop]
3. J•K FlipFlop
4. T Flip Flop

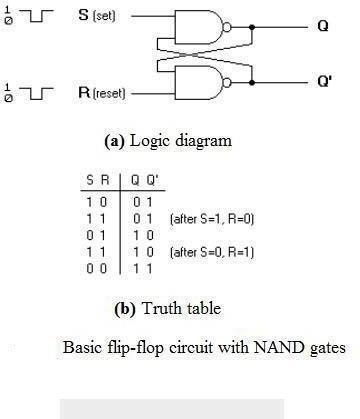
# S•R Flip Flop:

* The SET•RESET flip flop is designed with the help of two NOR gates and also two NAND gates. These flipflops are also called S•R Latch.

## **S•R Flip Flop using NOR Gate:**

* The design of such a flip flop includes two in puts, called the SET[S]and RESET [R]. There are also two outputs, Q and Q’. The diagram and truth table is shown next page.
* From the diagram it is evident that the flip flop has mainly four states. They are S=1, R=0— Q=1, Q’=0 This state is also called the SET state.
* S=0, R=1— Q=0, Q’=1 This state is known as the RESET state.
* In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the value of S.
* S=0, R=0—Q & Q’ = Remember If both the values of S and R are switched to 0, then the circuit remembers the value of S and R in their previous state. S=1, R=1—Q=0, Q’=0 [Invalid]
* This is an invalid state because the values of both Q and Q’ are 0. They are supposed to be compliments of each other. Normally, this state must be avoided.
* **S•R FLIP FLOP USING NAND GATE**

* The circuit of the S•R flip flop using NAND Gate and its truth table is shown below.



S•R

Flip

Flop

using

NAND

Gate

* Like the NOR Gate S•R flip flop, this one also has four

### states. They are S=1, R=0—Q=0, Q’=1

* This state is also called the SET state. S=0, R=1—Q=1, Q’=0 This state is known as the RESET state.
* In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the compliment value of S.

S=0, R=0—Q=1, & Q’ =1 [Invalid]

* If both the values of S and R are switched to 0 it is an invalid state because the values of both Q and Q’are1.They are supposed to be compliments of each other. Normally, this state must be avoided.

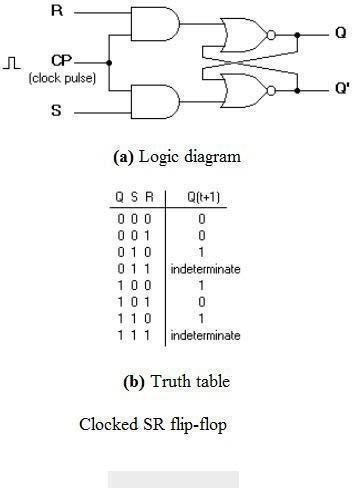
S=1, R=1—Q & Q’= Remember

* If both the values of S and R are switched to 1, then the circuit remembers the value of S and R in their previous state.

## **CLOCKED S•R FLIP-FLOP**

* It is also called a Gated S•R flipflop. The problems with S•R flipflops using NOR and NAND gate is the invalid state. This problem can be overcome by using a b is table SR flip•flop that can change outputs when certain invalid states are met, regardless of the condition of either the Set or the Reset inputs. For this, a clocked S•R flip flop is designed by adding two AND gates to a basic NOR Gate flip flop. The circuit diagram and truth table is shown below.

3



**PRACTICAL – 11**

**AIM:** Study and Implement different types of shift

**APPARATUS:** Sigma shiftregisters•DSR08, connecting wires

# THEORY:

# A universal shift register is an integrated logic circuit that can transfer data in three different modes. Like a parallel register it can load and transmit data in parallel. Like shift registers it can load and transmit data in serial fashions, through left shifts or right shifts. In addition, the universal shift register can combine the capabilities of both parallel and shift registers to accomplish tasks that neither basic type of register can perform on its own. For instance, on a particular job a universal register can load data in series (e.g. through a sequence of left shifts) and then transmit/output data in parallel

# Universal shift registers ,as all other types of registers, are used in computers as memory elements.Althoughothertypesofmemorydevicesareusedfortheefficientstorageo fverylarge volume of data, from a digital system perspective when we say computer memory we mean registers. In fact, all the operations in a digital system are performed on registers. Examples of such operations include multiplication, division, and data transfer.

# In order for the universal shift register to operate in a specific mode, it must first select the mode. To accomplish mode selection the universal register uses a set of two selector switches, S1 and S0. As shown in Table 1, each permutation of the switches corresponds to a loading/input mode.

|  |  |  |
| --- | --- | --- |
| Operating Mode | S1 | S0 |
| Locked | 0 | 0 |
| Shift-Right | 0 | 1 |
| Shift-Left | 1 | 0 |
| Parallel Loading | 1 | 1 |

* In the locked mode (S1S0 = 00) the register is not admitting any data; so that the content of the register is not affected by whatever is happening at the inputs. You may verify this detail by playing around with the main interactive circuit. For example, Set L3L2L1L0=1010 and then cycle the clock to see that nothing changes at the out puts as long as S1S0 = 00. See Table2.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clock Cycle | L3 | L2 | L1 | L0 |  | Q3 | Q2 | Q1 | Q0 |
| Initial Value | 1 | 0 | 1 | 0 |  | 0 | 0 | 0 | 0 |
| Cycle 1 | 1 | 0 | 1 | 0 |  | 0 | 0 | 0 | 0 |

***Table 2***

* In the shift•right mode(S1S0=01) serial inputs
* Are admitted from Q3 to Q0. You can confirm this aspect by setting the value of the shift•right switch according to the sequence 1100100 as you cycle the clock; see Table3. Watch as the signals move from Q3 to Q0. In the shift•left mode (S1S0 = 10) the register works in a similar fashion, except that the signals move from Q0 to Q3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Clock Cycle | Shift•Right Switch | Q3 | Q2 | Q1 | Q0 |
| Initial Value | Initial Value | 0 | 0 | 0 | 0 |
| Cycle 1 | 1 | 1 | 0 | 0 | 0 |
| Cycle 2 | 1 | 1 | 1 | 0 | 0 |
| Cycle 3 | 0 | 0 | 1 | 1 | 0 |
| Cycle 4 | 0 | 0 | 0 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cycle 5 | 1 | 1 | 0 | 0 | 1 |
| Cycle 6 | 0 | 0 | 1 | 0 | 0 |
| Cycle 7 | 0 | 0 | 0 | 1 | 0 |

# Table 3

* Finally,in the parallel loading mode(S1S0=11) data is read from the lines L0, L1, L2, and L3 simultaneously.
* Here, setting L3L2L1L0 = 1010 will cause Q3Q2Q1Q0 = 1010 after cycling the clock as depicted in Table4.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clock Cycle | L3 | L2 | L1 | L0 |  | Q3 | Q2 | Q1 | Q0 |
| Initial Value | 1 | 0 | 1 | 0 |  | 0 | 0 | 0 | 0 |
| Cycle 1 | 1 | 0 | 1 | 0 |  | 1 | 0 | 1 | 0 |

***Table 4***

**PROCEDURE:**

1. Do the connection as per below kit connection diagram for various shift register.
2. Apply proper input condition and observe the output information of ledon/off.
3. Compare theoretical data with observation and write conclusion.

**OBSERVATION:**

**Right Shift**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Clock Cycle | Shift•Right Switch | Q3 | Q2 | Q1 | Q0 |
| Initial Value | Initial Value | 0 | 0 | 0 | 0 |
| Cycle 1 | 1 | 1 | 0 | 0 | 0 |
| Cycle 2 | 0 | 0 | 1 | 0 | 0 |
| Cycle 3 | 0 | 0 | 0 | 1 | 0 |
| Cycle 4 | 1 | 1 | 0 | 0 | 1 |
| Cycle 5 | 1 | 1 | 1 | 0 | 0 |
| Cycle 6 | 0 | 0 | 1 | 1 | 0 |
| Cycle 7 | 1 | 1 | 0 | 1 | 1 |

**Left Shift**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Clock Cycle |  | Shift•Left Switch |  | Q3 | Q2 | Q1 | Q0 |
| Initial Value |  | Initial Value | 0 |  | 0 | 0 | 0 |
| Cycle 1 | 1 |  | 0 |  | 0 | 0 | 1 |
| Cycle 2 | 0 |  | 0 |  | 0 | 1 | 0 |
| Cycle 3 | 0 |  | 0 |  | 1 | 0 | 0 |
| Cycle 4 | 1 |  | 1 |  | 0 | 0 | 1 |
| Cycle 5 | 1 |  | 0 |  | 0 | 1 | 1 |
| Cycle 6 | 0 |  | 0 |  | 1 | 1 | 0 |
| Cycle 7 | 1 |  | 1 |  | 1 | 0 | 1 |

**Parallel Loading**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clock Cycle | L3 | L2 | L1 | L0 |  | Q3 | Q2 | Q1 | Q0 |
| Initial Value | 1 | 0 | 1 | 0 |  | 0 | 0 | 0 | 0 |
| Cycle 1 | 1 | 0 | 1 | 0 |  | 1 | 0 | 1 | 0 |

# CONCLUSION:

* In this practical we learn that how to study and implement the different shift.

**PRACTICAL – 12**

**AIM:** Implement the operation of Binary and Decade counter.

**APPARATUS:** Sigma Counters Trainer ∙DCO13, connecting wires.

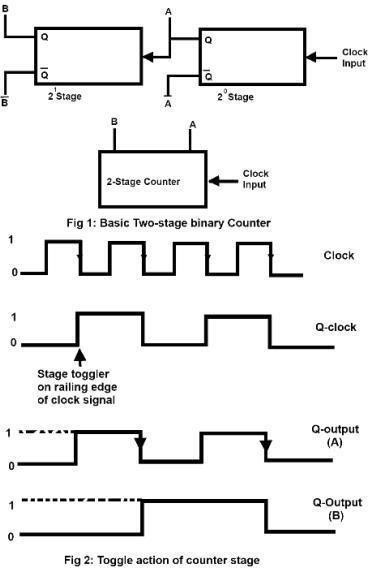
# THEORY:

* Counting is frequently required in digital computers and other digital systems to record the number of events occurring in a specified interval of time. Normally an electronic counter is used for counting the number of pulses coming at the input line in a specified time period. The counter must possess memory since it has to remember its past states. As with other sequential logic circuits counters can be synchronous or asynchronous. As the name suggests, it is a circuit which counts. The main purpose of the counter is to record the number of occurrence of some input. There are many types of counter both binary and decimal. Commonly used counters are

1. Binary Ripple Counter
2. Ring Counter
3. BCD Counter
4. Decade counter
5. Up down Counter
6. Frequency Counter

**Binary Ripple Counter**

* A binary ripple counter is generally using bistable multivibrator circuits so that cache input applied to the counter causes the count to advance or decrease. A basic counter circuit is shown in Figure 1 using two triggered (T•type) flip flop stages. Each clock pulse applied to the T•input causes the stage to toggle. The Q and output terminals are always logically opposite. If the Q output is logical 1 (SET), the output is then logical 0. If the Q output is logical 0 (REST), then the output is logical1.
* The clock input causes the flip flop to toggle or change stage once clock pulse Figure 2 (a)shows the clock input signal and Q output signal. Notice that the circuit used in this case toggles on the trailing edge of the clock signal (when logic signal goes from 1 to 0). Referring back to Figure 1 the Q output of the first stage (called the 2 o stage or units position stage) is used here as the toggle in put to the second stage s(called the 21 or two’s position stage). The Q output from the two successive stage are marked A and B, respectively, to differentiate them. Notice that the  output of each stage is marked with a negative bar over the letter designation, so that whatever logical stage A is at, is the opposite logical state. Since the Q output (A signal) from the first stage triggers the second stage, the second stage changes state only when the Q output of first stage goes from logical 1 to logical 0 as shown in Figure2 (b).



Input Pulses 2n Output (B) 2n Output (A)

|  |  |
| --- | --- |
| 0 0 | 0 |
| 1 0 | 1 |
| 2 1 | 0 |
| 3 1 | 1 |

4 or 0 0 0

* To see how a counter is made using more stage considers the 4 stage counter of Figure 3. The counter is simply made with the Q output of each state connected as the toggle input to the succeeding state. With four stages the counter cycle will repeat every sixteen clock pulses. In general there are 2n counts with an n•stage counter. For the four stages used here the count goes 24 or 16 steps as a rule, for a binary counter.
* Number of counts = N = 2n
* Where, n=number of counter stage. A six stage counter = 6 would be provide a count that repeat sever y N=26= 64 counts. A ten•stage counter (n=10) would recycle every N = 210 = 1024 counts.

**Decade Counter**

* A decade counter is the one which goes through 10 unique combinations of outputs and then resets as the clock proceeds. We may use some sort of a feedback in a 4•bit binary counter to skip any six of the sixteen possible output states from 0000 to 1111 to get to a decade counter. A decade counter does not necessarily count from 0000 to 1001 it could count as 0000,0001, 0010, 1000, 1001, 1010, 1011, 1110, 1111,0000, 0001 and so on. Figure 6 shows a decade counter having a binary count that is always equivalent to the input pulse count. The circuit is essentially, a ripple counter which count up to 16. We desire however, a circuit operation in which the count advance from 0 to 9 and then reset to 0 for a new cycle. This reset is a accomplished at the desired count as follows.

1. With counter REST count = 0000 the counter is ready to stage countercycle.
2. Input pulses advance counter in binary sequence up to count of a (count =1001)
3. The next count pulse advance the count to 10 count = 1010. A logic NAND gate decodes the count of 10 providing a level change at that time to trigger the one shot unit which then resets all counter stages. Thus, the pulse after the counter is at count = 9, effectively results in the counter going to count =0.

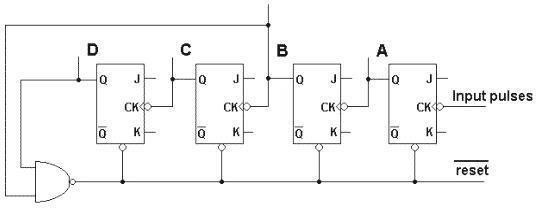


Figure 6 : Decade Counter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Pulses | D | C | B | A |
| 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 |
| 10 | 1 | 0 | 1 | 0 |
|  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 |

* Table provides a count table showing the binary count equivalent to the decimal count input pulses. The table also shows that the count goes momentarily count from nine (1001) to ten (1010) before resetting to zero(0000). The NAND gate provides an output of 1 until the count reach ten. The count of ten is decoded (or sensed in this case) by using logic inputs that area.
* The Q signal is used since it is normally high and goes low during the one shot timing period the flip flop in this circuit being reset by a low signal level (active low clearing). The one shot pulse need only be long enough so that slowest counter stage resets. Actually, at this time only the 21and 23 stage need be reset, but all stages are reset to insure that a new cycle at the count 0000.

**PROCEDURE:**

* Do the connection as per below kit connection diagram for various shift register.
* Apply proper input condition and observe the output information of ledon/off.

# Compare theoretical data with observation and write conclusion

**OBSERVATION:**

* **4 -Bit Binary Counter**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input pulses | D | C | B | A |
| 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 |
| 10 | 1 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 |

**Decade Counter:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input pulses | D | C | B | A |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

# CONCLUSION:

* In this practical we learn that how to count the binary and decade Counter.