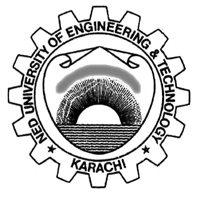
COMBINATIONAL AND SEQUENTIAL LOGIC

ASSIGNMENT

Second Year



* **SUBMITTED TO :** MISS SANA FATIMA
* **SUBJECT :** LOGIC DESIGN AND SWITCHING THEORY
* **PROJECT SUBMISSION DATE :** 20-DECEMBER-2019
* **GROUP LEADER’S NAME** : MUHAMMAD FARAZ ANSAR
* **GROUP LEADER’S ROLL NUMBER** : SE-061 (B)
* **OTHER GROUP MEMBERS :**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **ROLL NUMBER** | **NAME** |
| 1 | SE-062(B) | HASSAN-UR-REHMAN |
| 2 | SE-005(B) | ABDUL AHAD |
| 3 | SE-103(B) | TANVEER AHMED |
| 4 | SE-095(B) | JAHANZAIB RASHID |

CONTENT

|  |  |  |
| --- | --- | --- |
| **QUESTION NUMBERS** | **PAGE NUMBERS** | **DONE BY** |
| 1 AND 2 |  | SE-062 |
| 3 AND 4 |  | SE-005 |
| 5 , 6 AND 7 |  | SE-061 |
| 8 AND 9 |  | SE-103 |
| 10 , 11 , 12 AND 13 |  | SE-095 |

**Question # 1(a):**

* **Combination Circuit:**

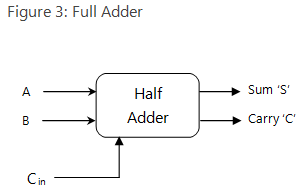
Logic gate are connected to produce specific output for certain specific combination of input variables with no storage involved the resulting circuit is called combinational logic.It is made up of different type of logic gates.

Examples of combination circuit:

* **Full adder:**
* **Truth Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Inputs** | | | **Outputs** | |
| **A** | **B** | **C-IN** | **SUM** | **C-OUT** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **0** | **1** | **0** |
| **1** | **0** | **1** | **0** | **1** |
| **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** |

* **Circuit Diagram:**



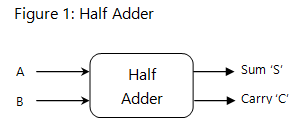
* **Explanation:**

Full Adder takes three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is S which is SUM.It is used to perform addition and is found in ALU(Arthimatic logic unit).It is used to compute larger numbers.

* **Half adder**
* **TruthTable:**

|  |  |  |  |
| --- | --- | --- | --- |
| **INPUTS** | | **OUTPUTS** | |
| **A** | **B** | **SUM** | **CARRY** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** |
| **1** | **0** | **1** | **0** |
| **1** | **1** | **0** | **1** |

* **Circuit Diagram:**

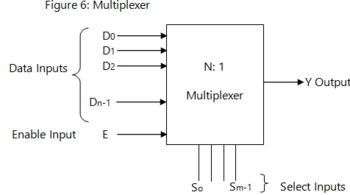


* **Explanation:**

It is used to compute single digit. The circuit takes two inputs and generate sum and carry.The carry is then passed to next input or according to truth table to next column.

**Multiplexer**

* **Circuit Diagram:**



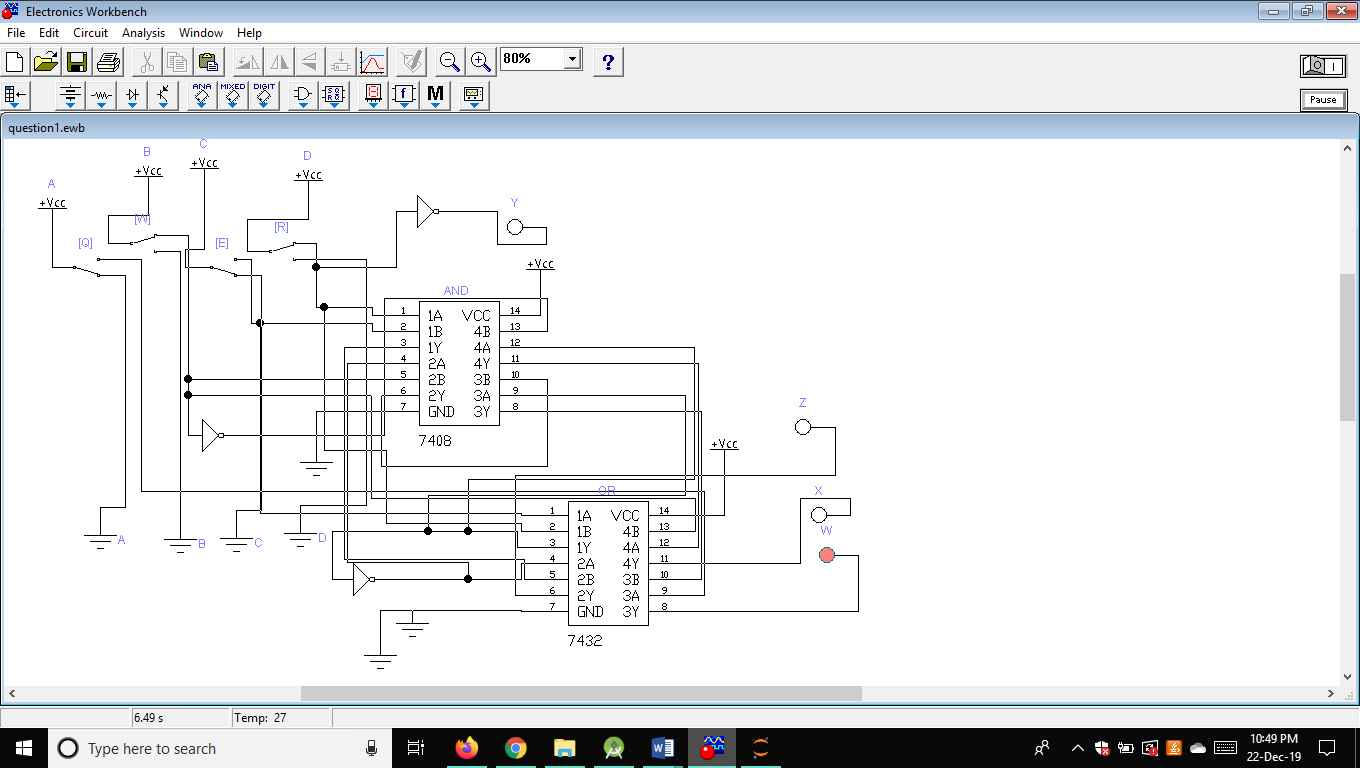
* **Explanation:**

It is designed to choose one input from multiple inputs and produce a single output.

**Question # 1(b)**

* **Truth Table:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** |  | **W** | **X** | **Y** | **Z** |
| **0** | **0** | **0** | **0** |  | **0** | **0** | **1** | **1** |
| **0** | **0** | **0** | **1** |  | **0** | **1** | **0** | **0** |
| **0** | **0** | **1** | **0** |  | **0** | **1** | **0** | **1** |
| **0** | **0** | **1** | **1** |  | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **0** |  | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **1** |  | **1** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** |  | **1** | **0** | **0** | **1** |
| **0** | **1** | **1** | **1** |  | **1** | **0** | **1** | **0** |
| **1** | **0** | **0** | **0** |  | **1** | **0** | **1** | **1** |
| **1** | **0** | **0** | **1** |  | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **0** |  | **X** | **X** | **X** | **X** |
| **1** | **0** | **1** | **1** |  | **X** | **X** | **X** | **X** |
| **1** | **1** | **0** | **0** |  | **X** | **X** | **X** | **X** |
| **1** | **1** | **0** | **1** |  | **X** | **X** | **X** | **X** |
| **1** | **1** | **1** | **0** |  | **X** | **X** | **X** | **X** |
| **1** | **1** | **1** | **1** |  | **X** | **X** | **X** | **X** |

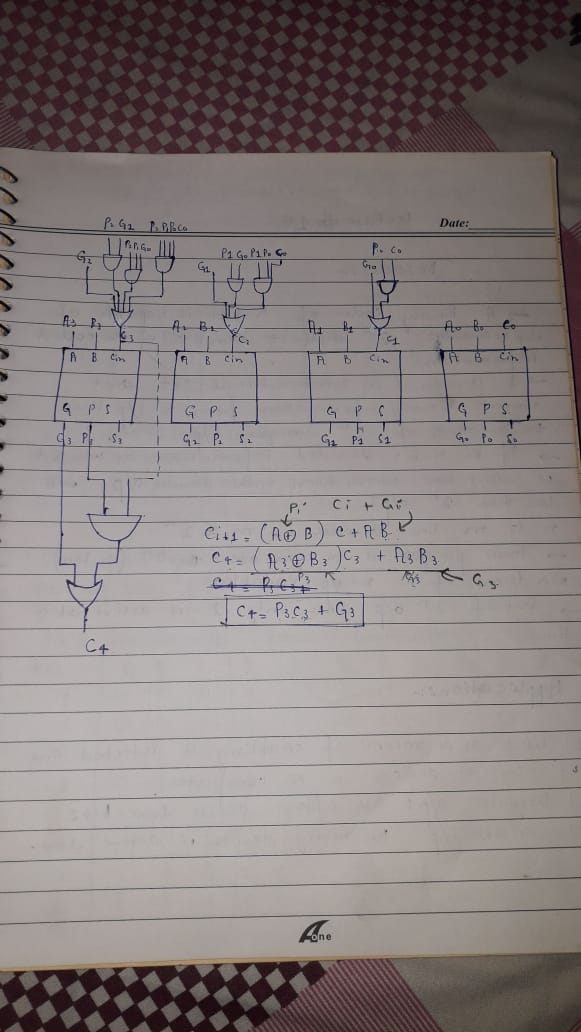
* **Logic Circuit Diagram:**
* **Stimulated:**

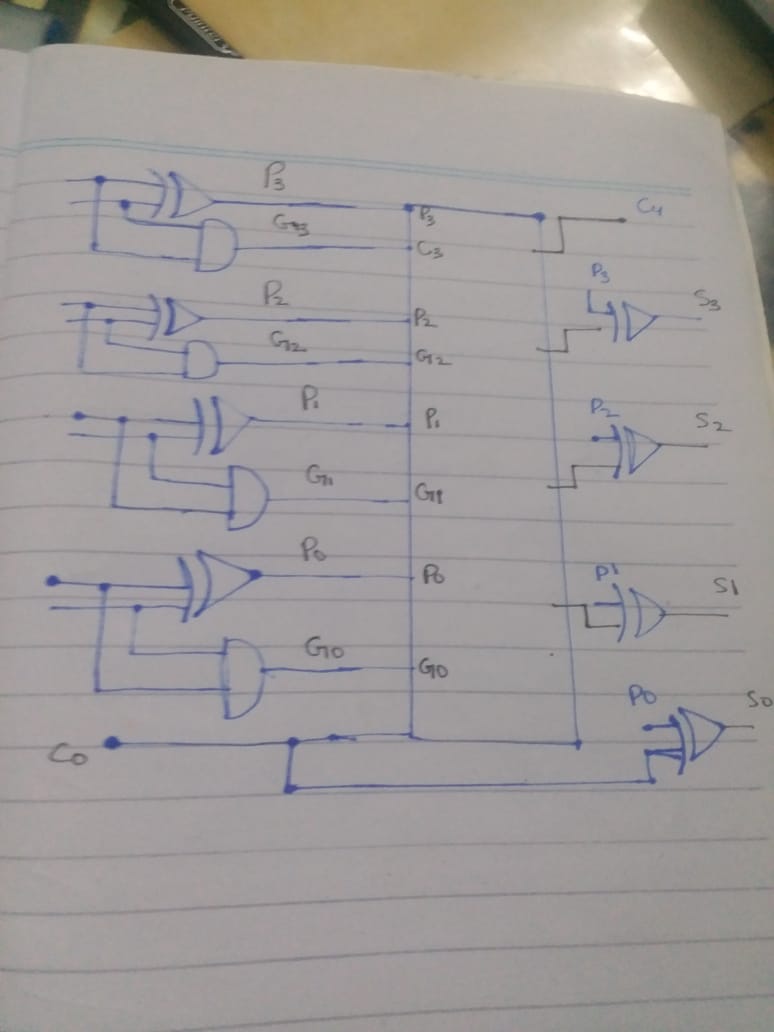
**Question # 2(a)**

* **Truth Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **S** | **Cout** |  |
| **0** | **0** | **0** | **0** | **0** | **No Carry Generation** |
| **0** | **0** | **1** | **1** | **0** | **No Carry Generation** |
| **0** | **1** | **0** | **1** | **0** | **Carry Propagation** |
| **0** | **1** | **1** | **0** | **1** | **Carry Propagation** |
| **1** | **0** | **0** | **1** | **0** | **Carry Propagation** |
| **1** | **0** | **1** | **0** | **1** | **Carry Propagation** |
| **1** | **1** | **0** | **0** | **1** | **Carry Generation** |
| **1** | **1** | **1** | **1** | **1** | **Carry Generation** |

* **Logic Circuit:**





* **Explanation:**

The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added. The most widely used technique to reduce the carry propogation time is carry look ahead logic.

Consider:

**P(i)=Ai ⊕ Bi**

**G(i)=AiBi**

Gi=Carry generate.It produces a carry of 1 when Both Ai and Bi are 1,regardless of i/p carry

Pi=Carry propogate.It determines whether a carry into stage I will propogate into stage i+1

The sum and carry can now be expressed as:

**Si=Pi⊕ Ci**

**C (i+1)=Gi and PiCi**

Now substituting the values of each Ci

So=initial carry

**C1=Go + PoCo**

**C2=G1+P1C1**

**C2=G1+P1(Go+PoCo)**

**C2=G1+P1Go+P1PoCo**

**C3=G2+P2C2.**

**C3=G2+P2(G1+P1Go+P1PoCo)**

**C3= G2+P2G1+P1P2Go+P1P2PoCo**

**C4=G3+P3C3=G3+P3G2+P3P2G1+P3P2P1Go+P3P2P1PoCin.**

The sum of A&B is given by

**S=C4S3S2S1So**

Where Si=Ai⊕Bi ⊕ Ci-1 for i=0,1,2,3

**So=Ao⊕BO⊕Cin=P(o)⊕Cin**

**S1=A1⊕B1⊕C0 =P(1)⊕C0**

**S2=A2⊕B2⊕C1=P2⊕C1**

**S3=A3⊕B3⊕C2=P3⊕C**

From the above Boolean equations we can observe that does not have to wait for and to propagate but actually is propagated at the same time as and . Since the Boolean expression for each carry output is the sum of products so these can be implemented with one level of AND gates followed by an OR gate.

**Question 2(b)**

* **Truth Table:**

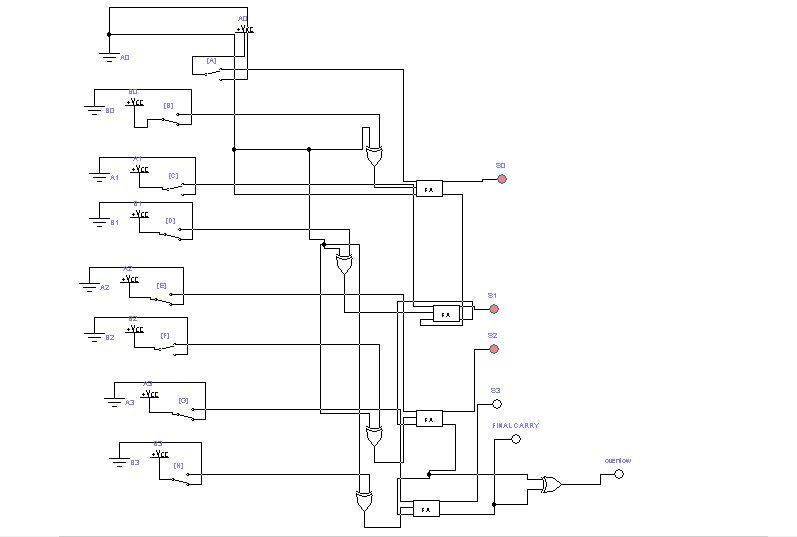
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | | | **Outputs** | |
| **A** | **B** | **C** | **CIN\_0** | **CIN\_1** | **CONTROL** | **S/D** | **C/B** |
| **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** |
| **0** | **0** | **1** | **0** | **1** | **0** | **1** | **0** |
| **0** | **0** | **1** | **0** | **1** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **0** | **0** | **1** | **1** | **1** | **1** |
| **0** | **1** | **1** | **0** | **1** | **0** | **0** | **1** |
| **0** | **1** | **1** | **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **0** | **0** | **1** | **0** | **1** | **0** |
| **1** | **0** | **0** | **0** | **1** | **1** | **1** | **0** |
| **1** | **0** | **1** | **0** | **1** | **0** | **0** | **1** |
| **1** | **0** | **1** | **0** | **1** | **1** | **0** | **0** |
| **1** | **1** | **0** | **0** | **1** | **0** | **0** | **1** |
| **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **0** | **1** | **1** | **1** | **1** |

* **Explanation:**

The mode input M controls the operation. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. - When M = 0, we have B ⊕ 0 = B. The full adders receive the value of B , the input carry is 0, and the circuit performs A plus B .

When M = 1, we have B ⊕ 1 = B’ and C0 = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2’s complement of B . (The exclusive-OR with output V is for detecting an overflow.)

* **Stimulated Diagram:**

****

**QUESTION #3 (a)**

* **Introduction:**

Gray Codes were invented by a Scottish scientist **Frank Gray.**

* **Definition:**

It can be defined as:

**“*A gray code is an encoding of binary numbers so that the adjacent numbers have a single digit differing by 1. It is also called Unit Distance and Reflected Binary Code.” [[1]](#footnote-1)***

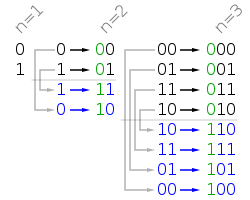
* **Explanation:**
* **Unit Distance Code:**

Gray code is an ordering of the binary numeral system such that two successive values differ each other in only one bit (binary digit) that is why it is also called **Unit Distance Code**.

* **Reflected Binary Code:**

Gray Code is called **Reflected Binary Code** because if we start writing gray codes **0** and **1** and then writing them backwards as **0, 1, 1, 0** and then finally proceed by prepending 0s in the first half as **00, 01** and 1s in the second half as **11, 10** … (we get **00, 01, 11, 10**) there is being seen a reflection (mirror) of the previous binary codes and we also observe that every iteration doubles the number of binary codes.

**A DIAGRAM IS ATTACHED SHOWING GRAY CODES FOR n=1, n=2 and n=3**

[[2]](#footnote-2)

* **APPLICATIONS OF GRAY CODES:**

There are so many applications of Gray Codes. Some of them are briefly defined below:

* **Error Correction:**

In modern [digital communications](https://en.wikipedia.org/wiki/Digital_communications), Gray codes play an important role in [error correction](https://en.wikipedia.org/wiki/Error_correction). For example, in Quadrature Amplitude Modulation where data is typically transmitted in [symbols](https://en.wikipedia.org/wiki/Symbol_rate) of 4 bits or more, the signals are arranged so that the bit patterns conveyed by adjacent inputs differ by only one bit (one digit). This makes the transmission system less susceptible to [noise](https://en.wikipedia.org/wiki/Noise).

* **Mathematical Puzzle:**

The Binary Reflected Gray Code can be used as a solution guide for many mathematical puzzles like Towers of Hanoi, Chinese rings puzzle and Hamiltonian cycle on a hypercube where each bit is seen as one dimension.

* **Genetic Algorithms:**

Due to the some special properties of Gray codes, they are sometimes used in [genetic algorithms](https://en.wikipedia.org/wiki/Genetic_algorithm) for searching in larger datasets in Artificial Intelligence.

* **Altimeters in Aircrafts:**

Altimeter is a device which is used to measure altitudes of aircrafts above ground level. In aircrafts, altimeters are normally mechanical where an encoding disk accompanied with the dials may produce a type of Gray Code output send to the receiver for processing. This specialized code reflects a one-bit change for each 100 foots increment allowing the altitude to be tracked.

**QUESTION #3 (b)**

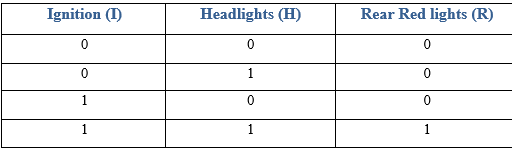
* **Explanation for part 1: (When highlights are ON, R must be ON)**

Let ‘I’ be the ignition of the engine of the car, ‘H’ be the headlights of the car and ‘B’ be the applied brakes of the car and ‘R’ be the rear red lights of the car.

According to given condition, if the ignition is OFF (0), then headlights and rear red lights will be off (H=0 and R=0).

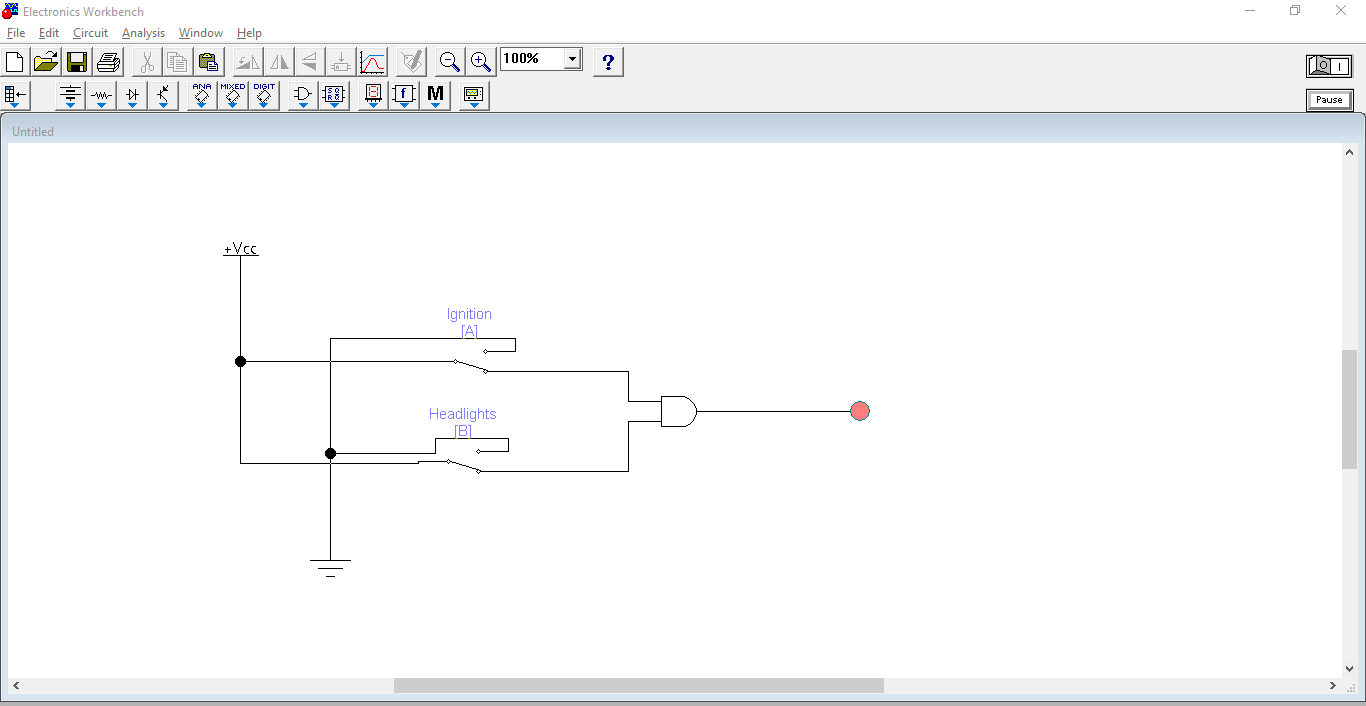
Secondly, when the ignition is on (1), then we will check whether headlight is ON or OFF. If headlights are ON, then rear red lights will be turned ON. It means that rear red lights will only glow when the ignition and headlights both will be turned ON, that is why we used AND gate in this condition.

* **TRUTH TABLE FOR REAR RED LIGHTS OF A CAR:**

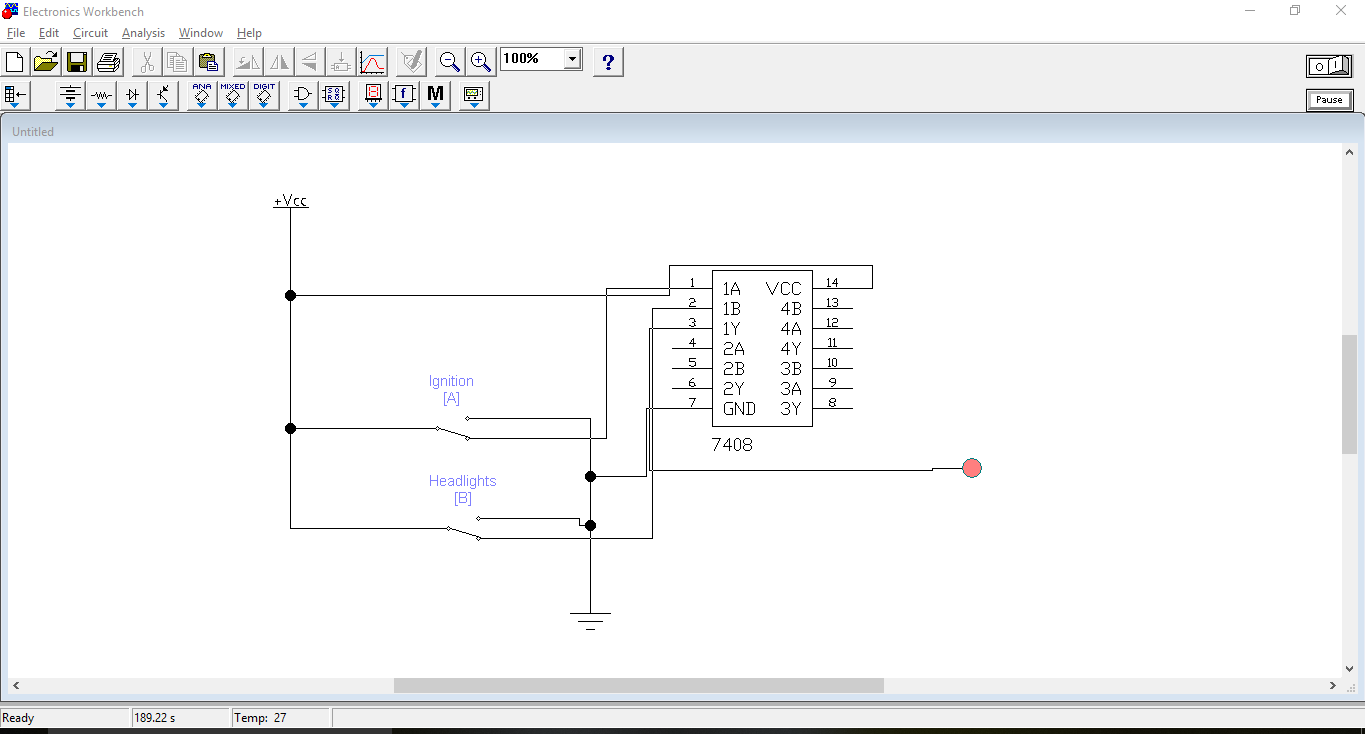


From truth table, we got an equation for our rear red light **R = IH**

* **Logic Circuit:**

****

* **Simulated Circuit Diagram:**



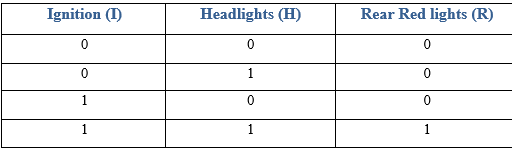
* **Explanation for part 2: (When brakes are applied, R must be ON)**

Let ‘I’ be the ignition of the engine of the car, ‘H’ be the headlights of the car and ‘B’ be the applied brakes of the car and ‘R’ be the rear red lights of the car.

According to given condition, if the ignition is OFF (0), then brakes and rear red lights will be off (R=0 and B=0).

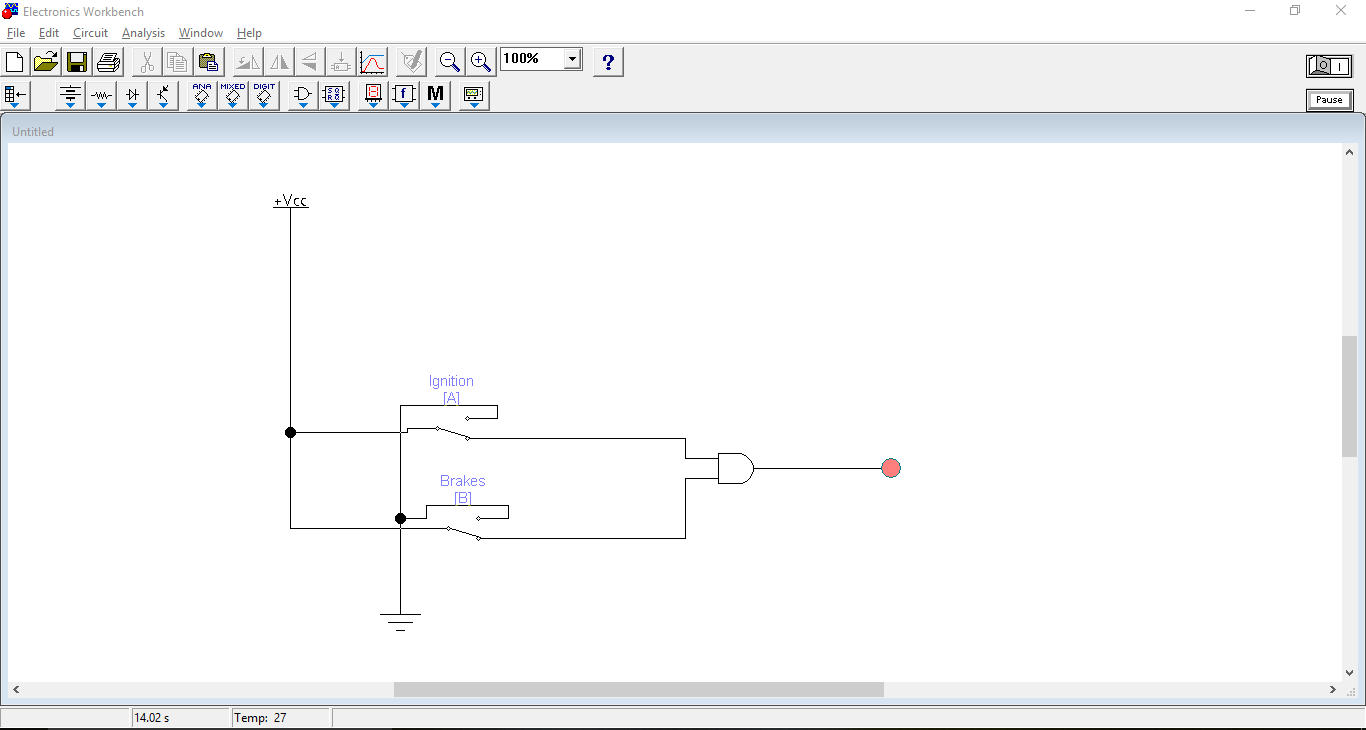
Secondly, when the ignition is on (1) and brakes are being applied (B=1), then rear red lights will be turned ON no matter whether headlights are switched ON or not. It means that rear red lights will only glow when the ignition and brakes both will be applied, that is why we used AND gate in this condition.

* **TRUTH TABLE FOR REAR RED LIGHTS OF A CAR:**

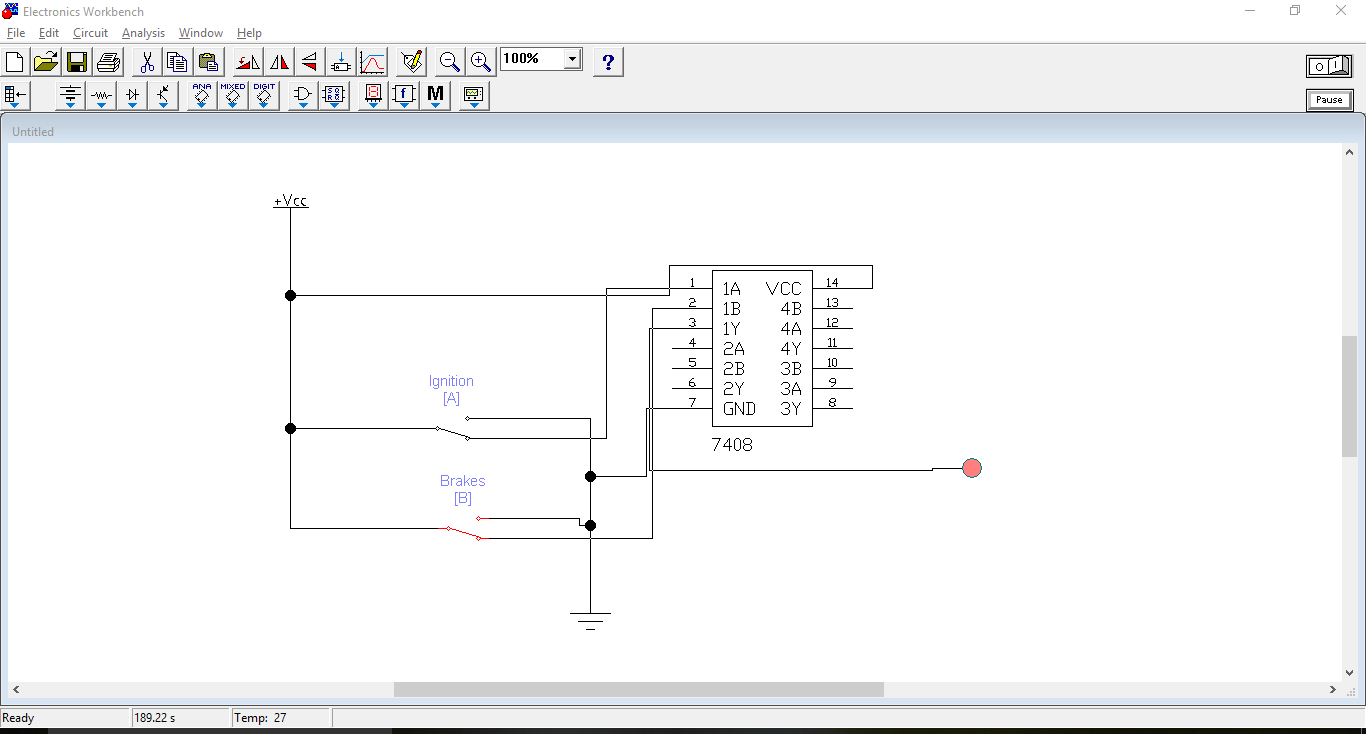


From truth table, we got an equation for our rear red light **R = IB**

* **Logic Circuit:**

****

* **Simulated Circuit Diagram:**



**QUESTION #4 (a)**

* **ERROR CODES:**
* **Definition:**

*“****During transmission of data (0 and 1), there may be a chance of transmitting wrong codes (0 as 1 and its vice versa), this error is called Error Codes.”***

* **Explanation:**

When digital signals (groups of 0s and 1s) are transmitted from one circuit or system to another, error may occur during transmission that is 1 can be changed to 0 and 0 can be changed to 1. It is necessary to detect and correct these codes by using different methods. Parity bit detector is one of them.

* **PARITY BIT DETECTION:**

In order to detect error in code, we use an extra bit called parity bit. If there is any change in code, parity bit is turned ON otherwise it is OFF.

* **EVEN PARITY:**

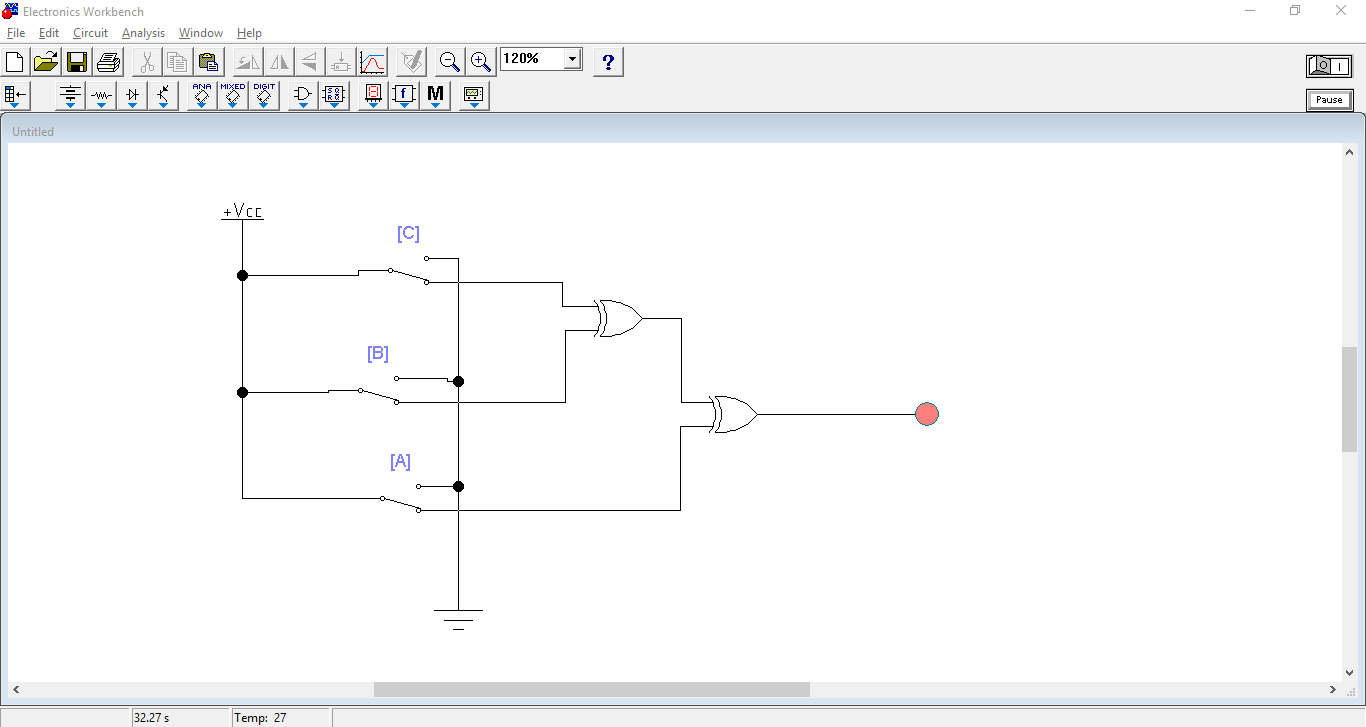
Parity bit is said to be even when the number of 1s present in code are even. For example, consider the following truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | Parity bit |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

If there were even number of 1s in the input, we turned OFF the parity bit, otherwise parity bit was turned ON, this is called **Even Parity.**

Using K-map, we can get the logic expression **P = A (xor) B (xor) C.**

* **Logic Circuit:**

****

* **ODD PARITY:**

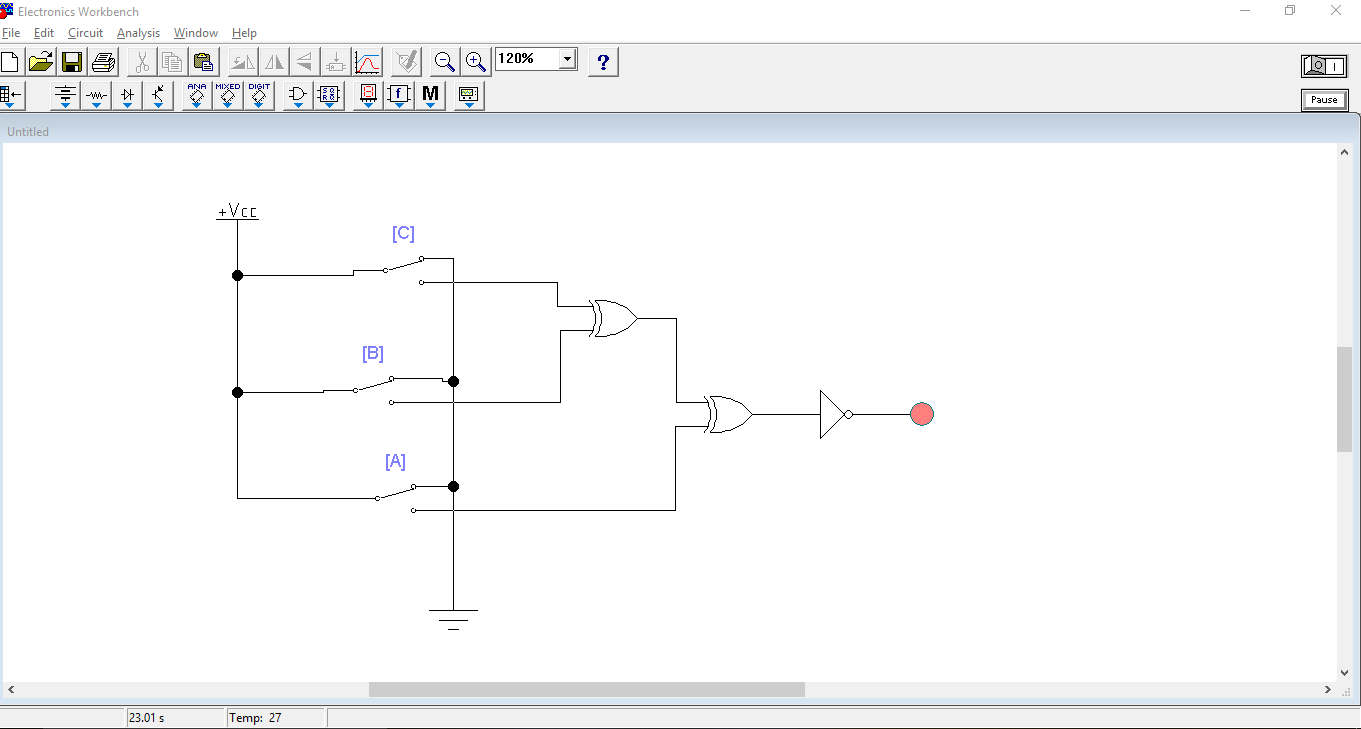
Parity bit is said to be odd when the number of 1s present in code are odd. For example, consider at the following truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **Parity bit** |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

If there were odd number of 1s in the input, we turned OFF the parity bit, otherwise parity bit was turned ON, this is called **Odd Parity.**

Since it is exactly opposite to even parity, that is why we will apply an inverter in even parity to make it odd parity.

* **Logic Circuit:**

****

**QUESTION#4 (b)**

First, we will make a truth table for 4 inputs A, B, C and D and we will have 7 outputs that is a, b, c, d, e, f and g.

* **TRUTH TABLE FOR DECIMALTO BCD 7 SEGMENT DISPLAY:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Decimal** | **A** | **B** | **C** | **D** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

From the above truth table, the Boolean expressions of each output functions can be written as:

**a = F1 (A, B, C, D) = ∑m (0, 2, 3, 5, 7, 8, 9)**

**b = F2 (A, B, C, D) = ∑m (0, 1, 2, 3, 4, 7, 8, 9)**

**c = F3 (A, B, C, D) = ∑m (0, 1, 3, 4, 5, 6, 7, 8, 9)**

**d = F4 (A, B, C, D) = ∑m (0, 2, 3, 5, 6, 8)**

**e = F5 (A, B, C, D) = ∑m (0, 2, 6, 8)**

**f = F6 (A, B, C, D) = ∑m (0, 4, 5, 6, 8, 9)**

**g = F7 (A, B, C, D) = ∑m (2, 3, 4, 5, 6, 8, 9)**

* **K-maps for a, b, c, d, e, f and g:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 1 | 0 | 1 | 1 |
|  | **01** | 1 | 0 | 1 | 0 |
|  | **11** | x | x | x | x |
|  | **10** | 1 | 1 | x | x |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 1 | 0 | 1 | 1 |
|  | **01** | 0 | 1 | 1 | 1 |
|  | **11** | x | x | x | X |
|  | **10** | 1 | 1 | x | X |

**a = A + C + BD + B’D’ b = B’ + C’D’ + CD**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 1 | 0 | 1 | 1 |
|  | **01** | 0 | 1 | 0 | 1 |
|  | **11** | x | x | x | x |
|  | **10** | 1 | 1 | x | x |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 1 | 1 | 1 | 0 |
|  | **01** | 1 | 1 | 1 | 1 |
|  | **11** | x | x | x | x |
|  | **10** | 1 | 1 | x | x |

**c = B + C’ + D d = B’D’ + CD’ + BC’D + B’C + A**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 1 | 0 | 0 | 0 |
|  | **01** | 1 | 1 | 0 | 1 |
|  | **11** | x | X | x | x |
|  | **10** | 1 | 1 | x | x |

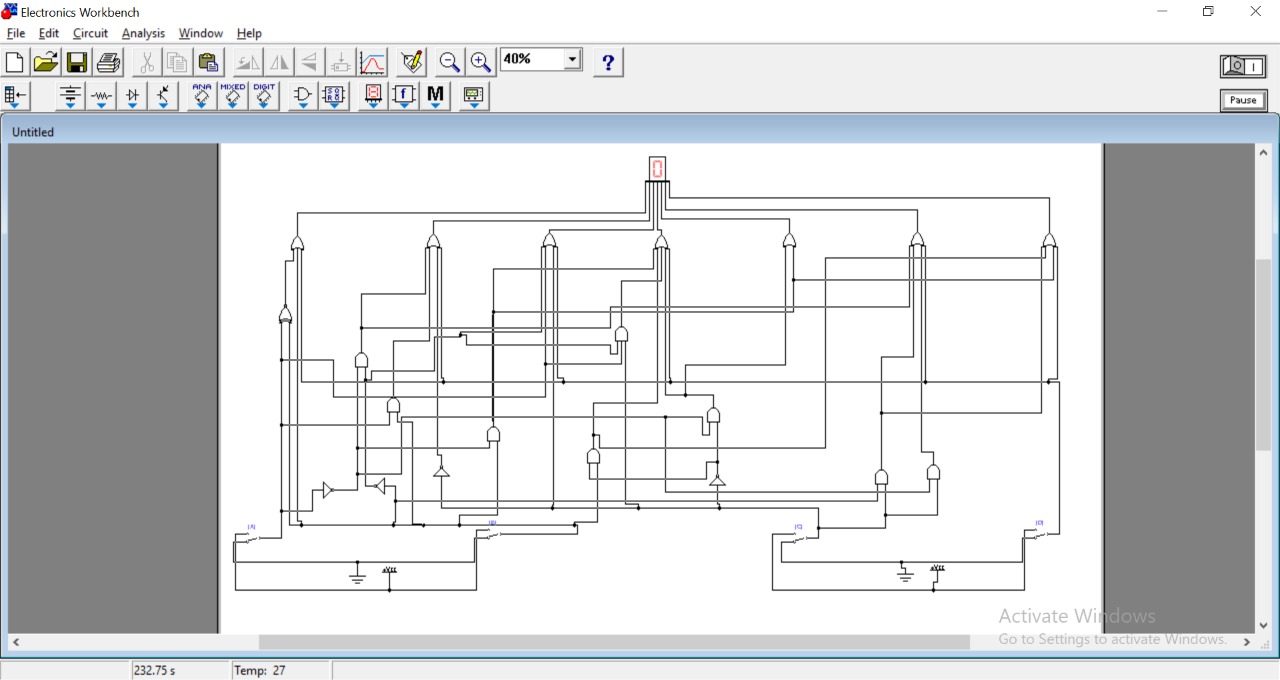
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 1 | 0 | 0 | 1 |
|  | **01** | 0 | 0 | 0 | 1 |
|  | **11** | x | x | x | X |
|  | **10** | 1 | 0 | x | x |

**e = B’D’ + CD’ f = A + C’D’ + BC’ + BD’**

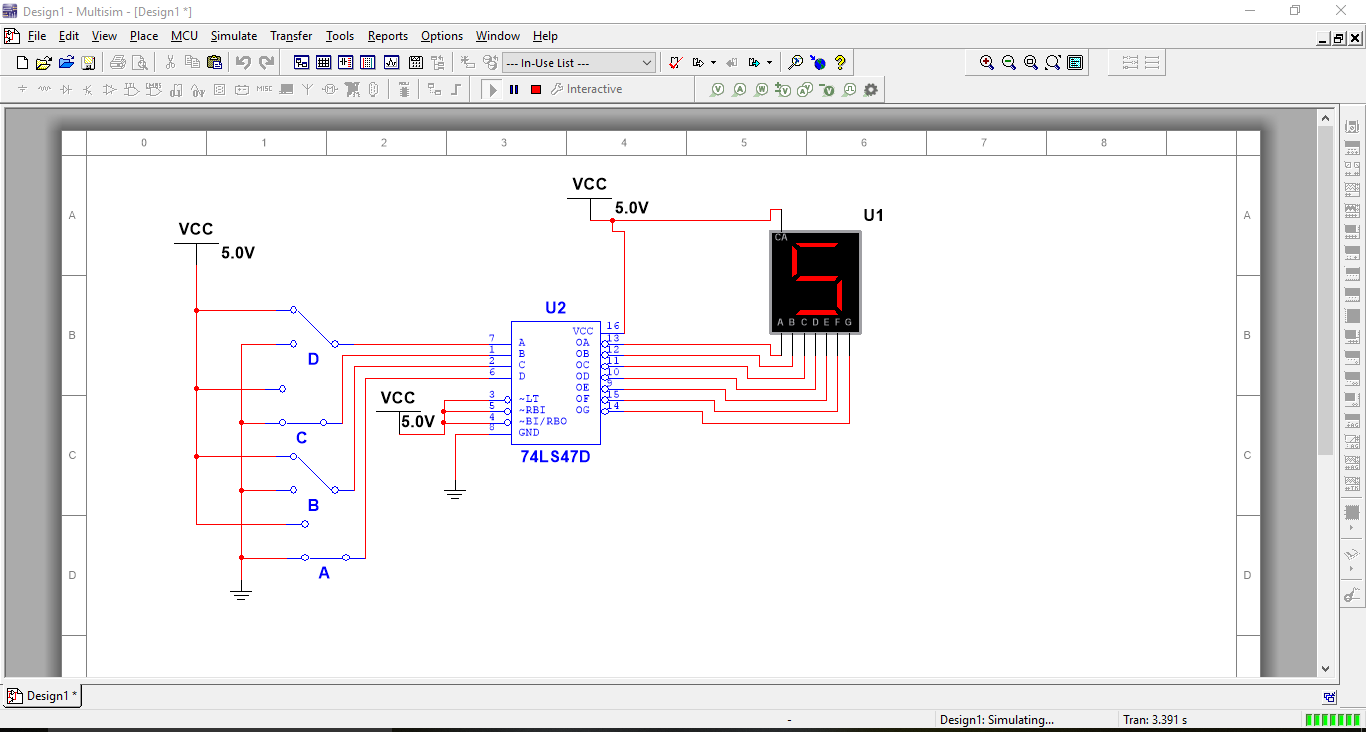
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CD** |  |  |  |  |
| **AB** |  | **00** | **01** | **11** | **10** |
|  | **00** | 0 | 0 | 1 | 1 |
|  | **01** | 1 | 1 | 0 | 1 |
|  | **11** | x | x | x | x |
|  | **10** | 1 | 1 | x | x |

**g = B’C + CD’ + BC’ + A**

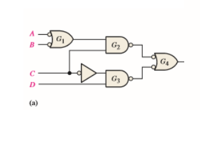
* **Logic Circuit:**

****

* **Simulated Circuit Diagram:**

****

**Write down the Boolean expression of the following circuit .Give the output wave form of each gate in the diagram.**



* **BOOLEAN EXPRESSION:**

**[(A’+B’) . C] + [C’ . D]**

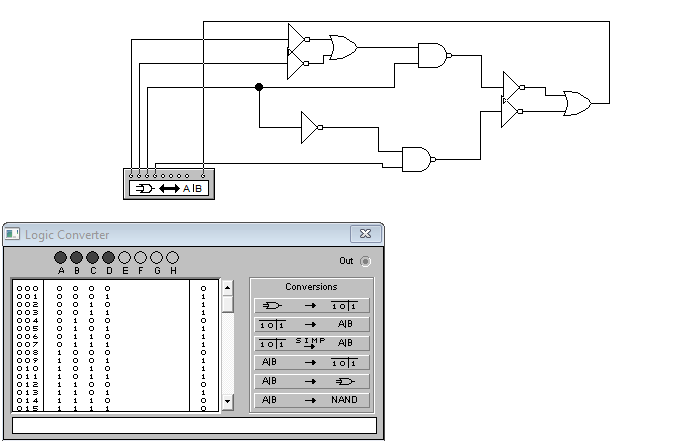
* **TRUTH TABLE :**

**LET ,**

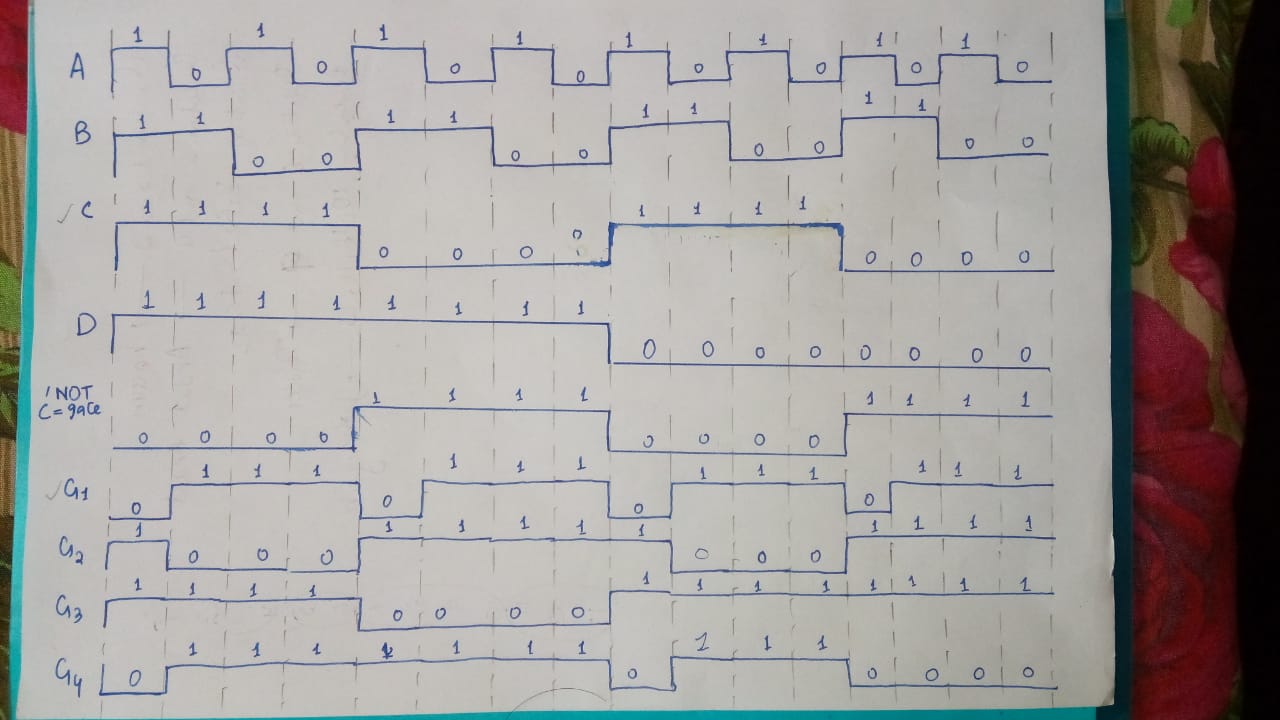
**F=[(A’+B’) . C] + [C’ . D] , P=[(A’+B’) . C] , Q=[C’ . D]**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **A’** | **B’** | **C’** | **A’ + B’** | **P** | **Q** | **F** |
| **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **1** |
| **0** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **0** |
| **0** | **1** | **0** | **1** | **1** | **0** | **1** | **1** | **0** | **1** | **1** |
| **0** | **0** | **1** | **0** | **1** | **1** | **0** | **1** | **1** | **0** | **1** |
| **0** | **0** | **1** | **1** | **1** | **1** | **0** | **1** | **1** | **0** | **1** |
| **0** | **1** | **1** | **0** | **1** | **0** | **0** | **1** | **1** | **0** | **1** |
| **0** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **0** | **0** | **0** |
| **1** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **1** | **1** | **0** | **1** | **0** | **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

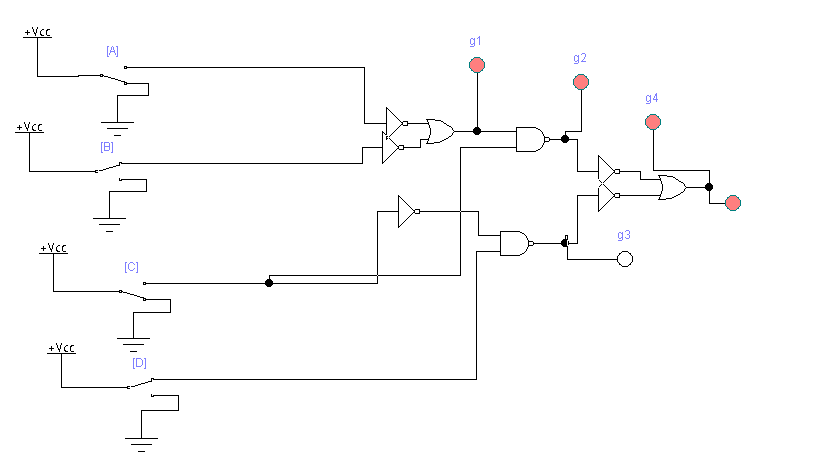
* **VERIFIACTION THROUGH LOGIC CONVERTER:**



* **OUTPUT WAVEFORM FOR EACH MENTIONED GATE :**

****

* **VERIFIACTION OF WAVEFORM THROUGH CIRCUIT SIMULATION:**



**QUESTION NO : 5 (b)**

**Explore a logic circuit with four input variables that will only produce a 1 output when**

**exactly three input variables are 1s. Also draw logic circuit diagram. (Attach Simulated**

**circuit diagram as well).**

* **TRUTH TABLE :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **OUTPUT** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** |
| **0** | **1** | **0** | **0** | **0** |
| **0** | **1** | **0** | **1** | **0** |
| **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **1** | **1** | **0** |
| **0** | **1** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **1** |
| **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **1** | **0** |
| **1** | **1** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** | **0** |

* **BOOLEAN EXPRESSION:**

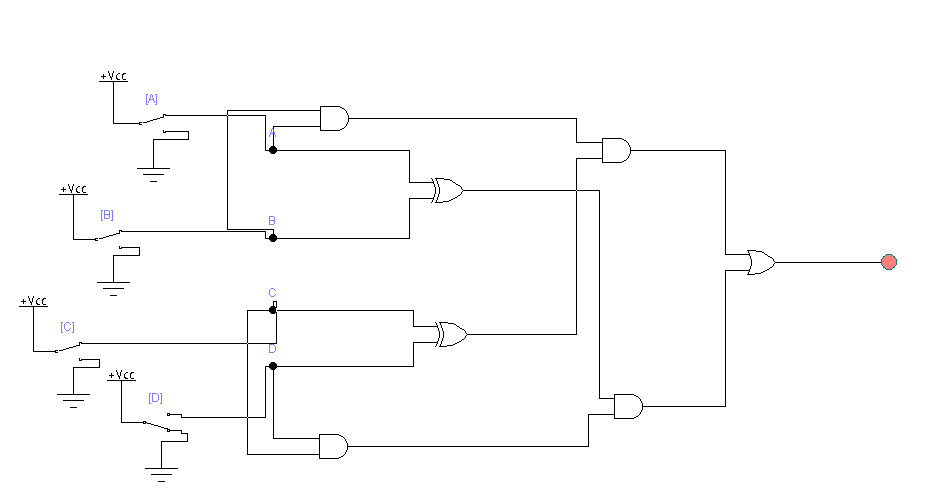
**A’BCD + AB’CD + ABC’D + ABCD’**

* **REDUCED EXPRESSION:**

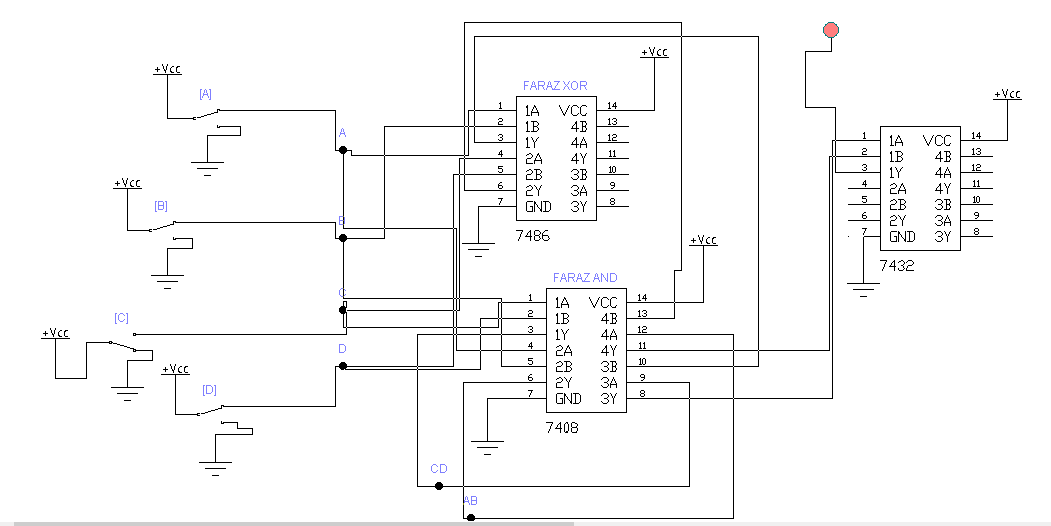
By applying Boolean algebra laws, the above expression simplified as;

**CD(A ⊕ B) + AB(C ⊕ D)**

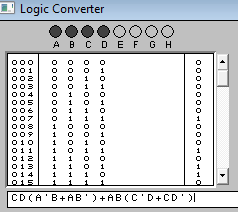
* **GATE-LEVEL SIMULATION CIRCUIT:**



* **IC-LEVEL SIMULATION CIRCUIT:**



* **VERIFICATION OF BOOLEAN EXPRESSION USING LOGIC CONVERTER:**



**QUESTION NO : 6 (a)**

Elaborate logic of CLC that takes 2-bit input value and produces the square of the value in

binary form with the help of truth table. Draw neat & clean logic diagram. (Attach Simulated

circuit diagram as well).

* **EXPECTED TRUTH TABLE FOR 2-bit CLC FOR SQUARE DETERMINATION:**
* The number of inputs should be **2 i-e A and B**
* The range of input will be **0-3**
* The highest square value will be **3­­2 = 9**
* The binary form of 9(max range) is **1001 i-e 4-bit output**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **P0** | **P1** | **P2** | **P3** |
| **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** | **0** | **1** |
| **1** | **0** | **0** | **1** | **0** | **0** |
| **1** | **1** | **1** | **0** | **0** | **1** |

* **BOOLEAN EQUATION FOR P0:**

**AB**

* **BOOLEAN EQUATION FOR P1:**

**AB’**

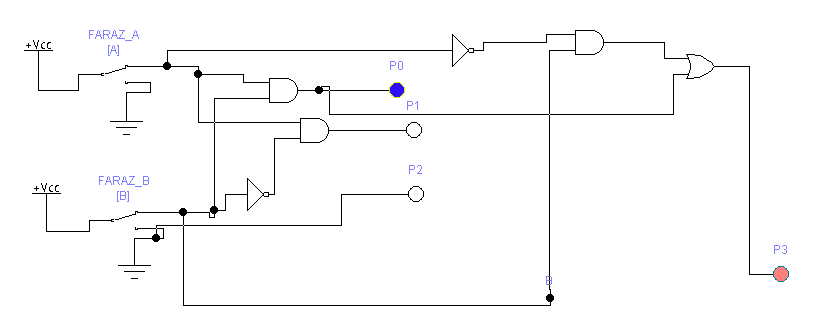
* **BOOLEAN EQUATION FOR P2:**

**0**

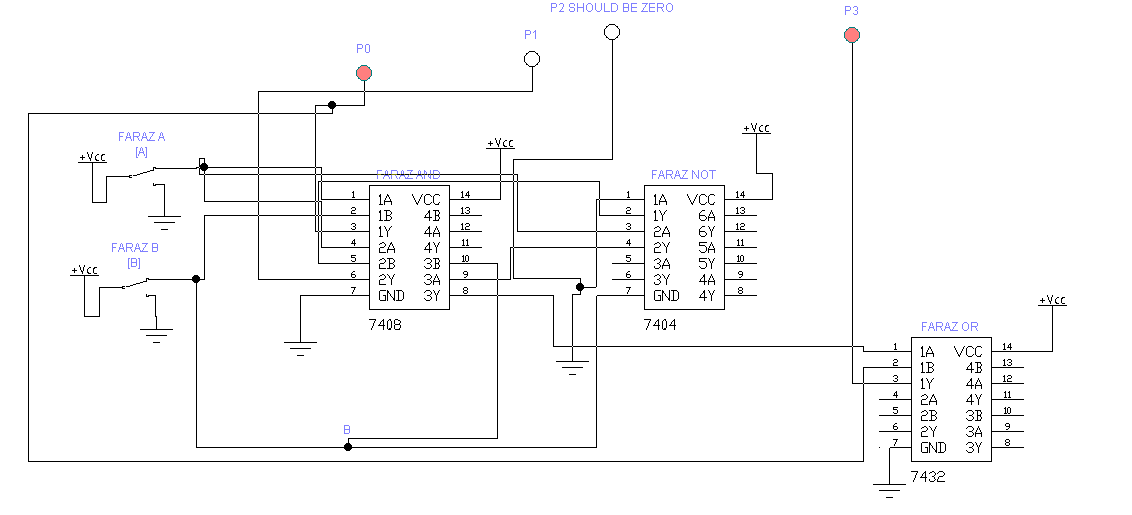
* **BOOLEAN EQUATION FOR P3:**

**A’B+AB**

* **GATE LEVEL DIAGRAM FOR CLC-SQUARE CIRCUIT:**



* **IC-LEVEL DIAGRAM FOR CLC-SQUARE CIRCUIT:**



* **EXPLAINATION:**
* First we have to identify that there are ‘2-bit’ input so that the numbers in range are 0-3.
* Then we have to take the square of highest possible number i-e ‘3’ which is ‘9’(1001) i-e maximum 4-bit output possible.
* After that we have to make the truth table and pick Boolean equation from each bit output and draw the circuit accordingly.

**QUESTION NO : 6 (b)**

1. **Cascade two 8 \*1 and one 2 \*1 multiplexers to implement 16 \*1 multiplexer logic.**

* **TRUTH TABLE OF EXPECTED 16\*1 MUX:**
* **Where S0 , S1 , S2 and S3  are selecting input lines and S3 being MSB.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S3** | **S2** | **S1** | **S0** | **OUTPUT FROM** |
| **0** | **0** | **0** | **0** | **I0** |
| **0** | **0** | **0** | **1** | **I1** |
| **0** | **0** | **1** | **0** | **I2** |
| **0** | **0** | **1** | **1** | **I3** |
| **0** | **1** | **0** | **0** | **I4** |
| **0** | **1** | **0** | **1** | **I5** |
| **0** | **1** | **1** | **0** | **I6** |
| **0** | **1** | **1** | **1** | **I7** |
| **1** | **0** | **0** | **0** | **I8** |
| **1** | **0** | **0** | **1** | **I9** |
| **1** | **0** | **1** | **0** | **I10** |
| **1** | **0** | **1** | **1** | **I11** |
| **1** | **1** | **0** | **0** | **I12** |
| **1** | **1** | **0** | **1** | **I13** |
| **1** | **1** | **1** | **0** | **I14** |
| **1** | **1** | **1** | **1** | **I15** |

* **ROUGH SKETCH FOR ABOVE CASCADING:**

**SELECT INPUT LINE(MSB)**

**SELECT INPUT LINES**

**SELECT INPUT LINES**

**16 INPUT LINES**

**1 OUTPUT LINE**

**FARAZ**

**2\*1 MUX**

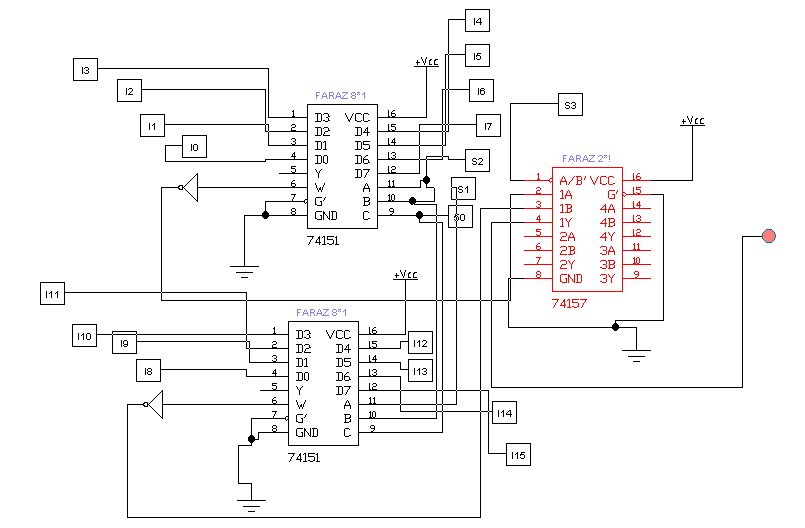
**FARAZ**

**8\*1 MUX**

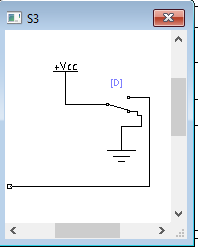
**FARAZ**

**8\*1 MUX**

* **IC-LEVEL IMPLEMENTATION FOR 16\*1 MUX CASCADING:**



* **SUB-CIRCUIT FOR EACH SWITCH:**
* **Where toggle key for each switch is different**

* **EXPLAINATION:**

In 16\*1 multiplexer , there are 16 input lines and one output line with 4 selecting input lines. The selecting input lines decides which input line transmit the signal to output.

To cascade the above from two 8\*1 and one 2\*1 mux, we have 3 select input lines in both 8\*1 mux. So connect the selecting inputs of both 8\*1 mux with eachother which are S0 , S1 , S2. The 2\*1 MUX has only one selecting input line which become the 4th one which is S3. In the circuit **S3  will be MSB.**

This S3 will be responsible to switch two 8\*1 mux to generate desired output.

For example, if the input is 1000 .Since in this example S3 is ‘1’ then output will be from second 8\*1 mux then later bits are ‘000’ hence output will be from ‘D0(from IC)’ which is ‘I8’.

**ii. Draw the logic diagram of a 2-to-4 line decoder using NOR gates only. Include an enable**

**input. (Attach Simulated circuit diagram as well).**

* **TRUTH TABLE FOR 2\*4 DECODER:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **E** | **A** | **B** | **P0** | **P1** | **P2** | **P3** |
| **1** | **0** | **0** | **1** | **0** | **0** | **0** |
| **1** | **0** | **1** | **0** | **1** | **0** | **0** |
| **1** | **1** | **0** | **0** | **0** | **1** | **0** |
| **1** | **1** | **1** | **0** | **0** | **0** | **1** |
| **0** | **DON’T CARE** | **DON’T CARE** | **0** | **0** | **0** | **0** |

* **BOOLAEN EQUATION FOR P0:**

**E A’ B’**

* **BOOLAEN EQUATION FOR P1:**

**E A’ B**

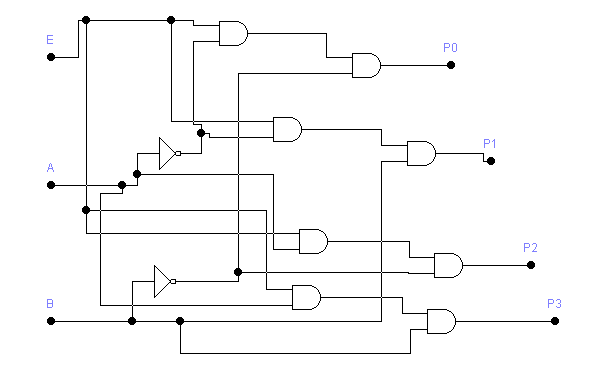
* **BOOLAEN EQUATION FOR P2:**

**E A B’**

* **BOOLAEN EQUATION FOR P3:**

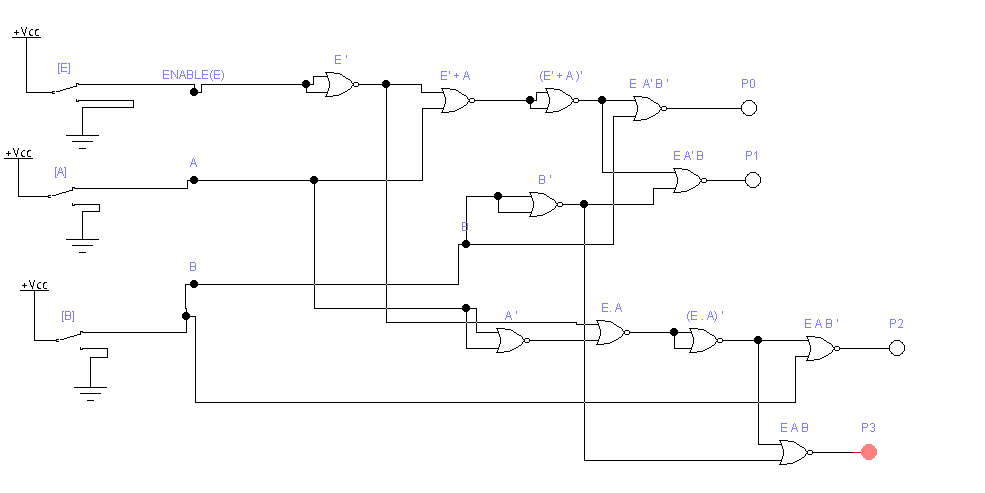
**E A B**

* **CIRCUIT FOR 2\*4 LINE DECODER WITHOUT USING NOR-GATES:**

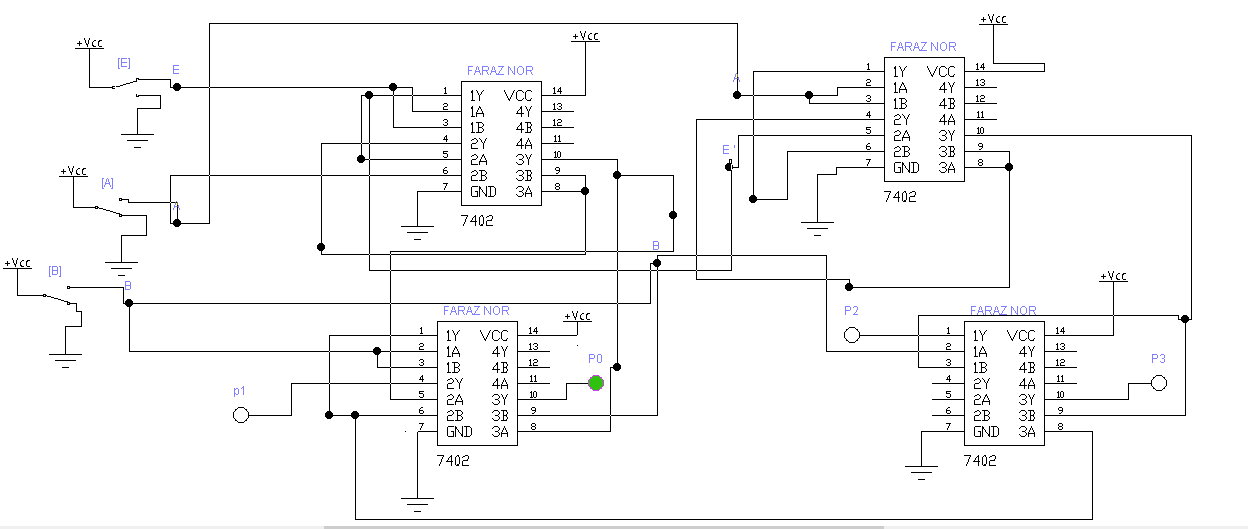


* **NOR – REALIZATION OF ABOVE CIRCUIT:**

By applying NOR-REALIZATION to above simple circuit, we got 2\*4 line decoder WITH only NOR-GATES.



* **IC-LEVEL IMPLEMENTATION FOR ABOVE DECODER:**



* **EXPLAINATION:**

From truth table of 2\*4 line decoder, we pick the Boolean equations for each output lines i-e P0 , P1 , P2 and P3.

Then we draw the simple circuit without using NOR GATES.

After all we applied NOR-REALIZATION METHOD to convert that circuit to NOR-GATES.

In NOR-REALIZATION we all know that AND gate can be replace by three NOR gates and NOT gate can be replace by one NOR gate.

**QUESTION NO: 7(a)**

**How well you understand Multiplexer, De- multiplexer, and Encoder? Explain with suitable applications**

* **MULTIPLEXER:**
* It is a device which accepts ‘2n’ number of signals and output only one signal of desired frequency at a time.
* The lines which decides whether which input line’s signal go towards the output are called selecting input lines which are ‘n’ in number.
* For example, consider 2\*1 MUX . It has 2 inputs i-e ‘21’ and 1 selecting input line and 1 output.

**2N INPUT LINES**

**MUX**

**1 OUTPUT**

**.**

**.**

**.**

**. . ….**

**N-SELECT LINES**

* **APPLICATIONS OF MULTIPLEXER:**
* **COMPUTER COMMUNICATION SYSTEMS:**

The purpose of communication systems is to receive the signals from transmitter(input) and select the desired frequency signal and move it towards the receiver(output). To get the things done, multiplexers are used.

* **COMPUTER MEMORY:**

Multiplexers are used in computer memory used to fetch desired data from memory. Our data whole data is connected to its input line and computer has just to give the selecting input to pick the desired data.

* **DE-MULTIPLEXER:**
* It is a device which accept ‘only 1’ signal as input and output ‘2n’ signals of desired frequency at a time. It has ‘n’ selecting inputs.
* For example, consider 1\*8 line decoder. It has 1 input and ‘23’ i-e ‘8’ outputs. And 3 select inputs.
* It is the reverse of multiplexer.
* It receive only one signal and output the signal at the desired stations(outputs).

**1 INPUT LINE**

**DE-MUX**

**2­N OUTPUT LINES**

* **. .**
  + - **.**

**N SELECT INPUTS**

* **APPLICATIONS OF DE-MULTIPLEXER:**
* **COMPUTER COMMUNICATION SYSTEMS:**

The purpose of communication systems is to receive the signals from transmitter(input) and select the desired frequency signal and move it towards the receiver(output). To get the things done, DE-multiplexers are also used.

* **ARITHMATIC AND LOGIC UNIT:**

The purpose of demultiplexers in ALU is to receive the arithmetic and logical operations output from ALU and move it to the required stations through multiple output lines.

* **ENCODER:**
* An encoder gives the opposite output as that of a decoder.
* Encoder has ‘**2n**’ input lines and ‘**n**’ output lines which is the reverse of decoder.
* For example, consider 4\*2 encoder. It has 4 inputs and 2 outputs. If the ‘I0’ of the decoder input will give signal i-e ‘on’ then the output will be its subscript code i-e ‘00’(0). And so on.

**2N INPUT LINES**

**ENCODER**

**N OUTPUT**

**.**

**.**

**.**

* **. .**

**.**

* **APPLICATIONS OF ENCODER:**
* Analog to digital converter
* robotics

**QUESTION NO: 7(b)**

Examine the logic of full adder circuit and implement with:

i. A decoder. Draw neat and clean diagram. (Attach Simulated circuit diagram as well).

* **TRUTH TABLE OF FULL ADDER:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A(MSB)** | **B** | **CIN** | **SUM** | **COUT** | **DECODER** |
| **0** | **0** | **0** | **0** | **0** | **Y0** |
| **0** | **0** | **1** | **1** | **0** | **Y1** |
| **0** | **1** | **0** | **1** | **0** | **Y2** |
| **0** | **1** | **1** | **0** | **1** | **Y3** |
| **1** | **0** | **0** | **1** | **0** | **Y4** |
| **1** | **0** | **1** | **0** | **1** | **Y5** |
| **1** | **1** | **0** | **0** | **1** | **Y6** |
| **1** | **1** | **1** | **1** | **1** | **Y7** |

* **BOOLEAN EQUATION FOR SUM:**

**A’B’CIN + AB’C’IN + A’BC’IN + ABCIN**

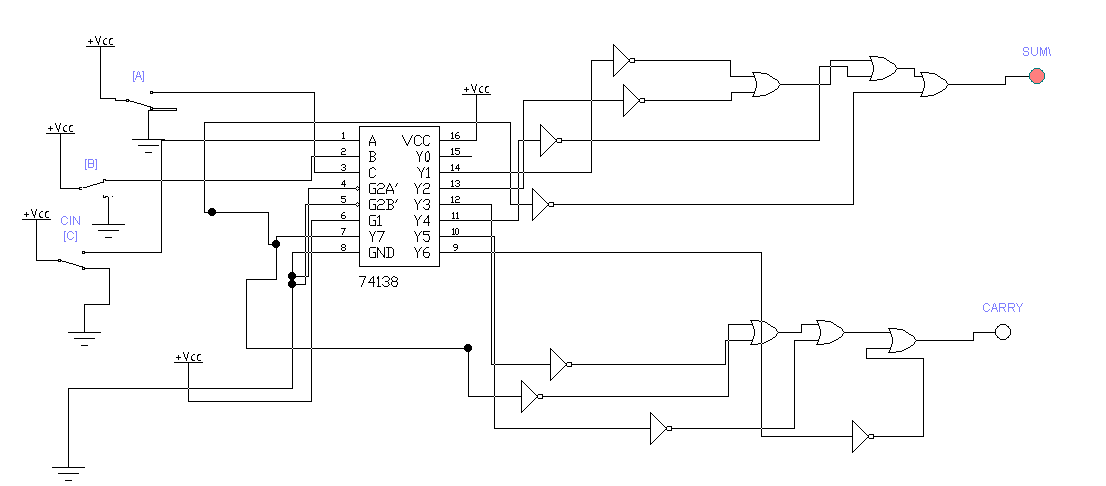
**SUM= Σ (Y1 , Y2 , Y4 , Y7)**

* **BOOLEAN EQUATION FOR CARRY:**

**AB’CIN + A’BCIN + ABC’IN + ABCIN**

**CARRY= Σ (Y3 , Y5 , Y6 , Y7)**

* **SIMULATED CIRCUIT:**
* C being MSB in simulated circuit
* Outputs are active low that’s why connecting not-gate



* **EXPLAINATION:**

In 3\*8 line Decoder, we have 3 input lines and 8 output lines. It converts the binary signal into decimal code from Y0 to y7 i-e 8 numbers. So according to the truth table, for sum, note the number of ones in the SUM-CLOUMN and sum(ORed) the related min-terms. similarly, for carry, note the number of ones in CARRY-COLUMN and sum(ORed) the related min-terms.

**ii. two 4 \*1 multiplexers. (Attach Simulated circuit diagram as well).**

* **TRUTH TABLE OF FULL ADDER:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A(MSB)** | **B** | **CIN** | **SUM** | **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** |

* **BOOLEAN EQUATION FOR SUM:**

**A’B’CIN + AB’C’IN + A’BC’IN + ABCIN**

**SUM= Σ (1,2,4,7)**

* **K-MAP FOR SUM:**

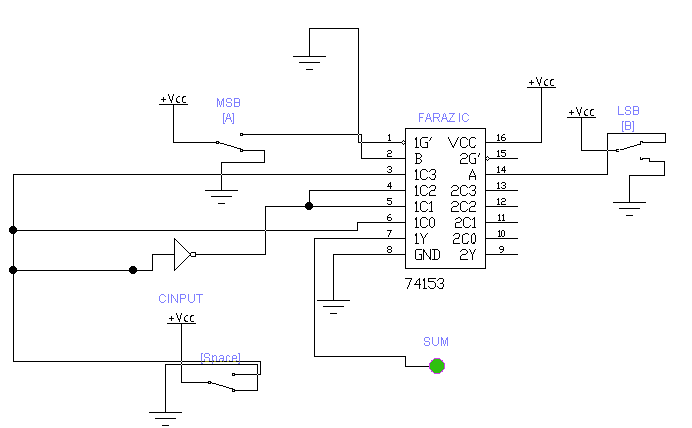
**AB CIN 0 1**

|  |  |  |
| --- | --- | --- |
| **I0**  **00** | **0** | **1** |
| **I1**  **01** | **1** | **0** |
| **I3**  **11** | **0** | **1** |
| **I2**  **10** | **1** | **0** |

* **EXPLAINATION:**

**From above K-MAP it is clear that in multiplexer I0 , I3 should be CIN and I1 , I2 should be C’IN(NOT).**

* **CIRCUIT FOR SUM USING 4\*1 MUX:**



* **EQUATION FOR CARRY:**

**CARRY=Σ(3,5,6,7)**

* **K-MAP FOR CARRY:**

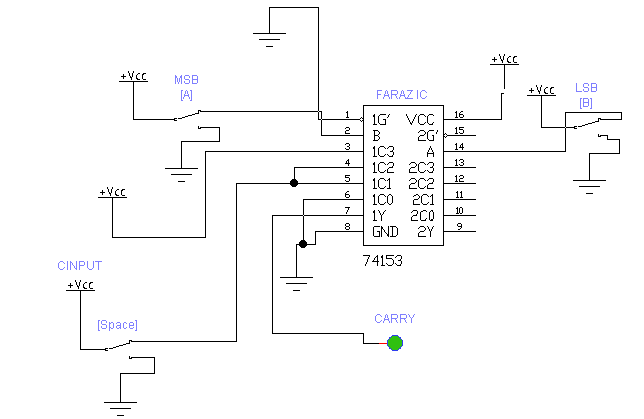
**AB CIN 0 1**

|  |  |  |
| --- | --- | --- |
| **I0**  **00** | **0** | **0** |
| **I1**  **01** | **0** | **1** |
| **I3**  **11** | **1** | **1** |
| **I2**  **10** | **0** | **1** |

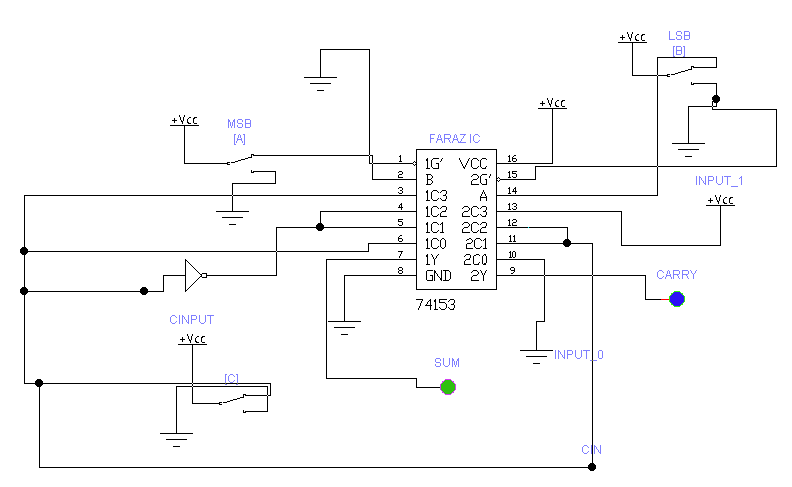
* **EXPLAINATION:**

From above K-MAP, it is clear that I0 will remain ‘0’ so it is connected to ground. Similarly, I3 will remain ‘1’ so It is connected to ‘vcc’ and I1 and I2 will remain which is CIN.

* **CIRCUIT FOR CARRY USING 4\*1 MUX:**



* **COMPLETE CIRCUIT OF FULL ADDER USING TWO 4\*1 MUX:**



**Question No 8(a**)

Why NAND and NOR gates are called universal gates? Discuss in detail with suitable examples.

The NAND and NOR gates are called universal gates because the combination of them can be used to carry out any of basic operation. By using NAND and NOR gates we can produce an inverter, an OR gate, or an AND gate. NAND and NOR gates are called universal because all other gates can be created by using these gates. The examples are given below.

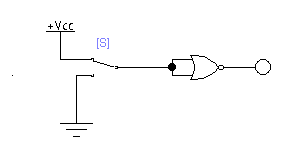
**Use of NOR to implement other gates**

* **1-NOR as NOT**
* **TRUTH TABLE**

|  |  |  |
| --- | --- | --- |
| S | T | NOR |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| S | T | (ST)’ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

* **Logic Circuit Diagram**



As NOR=NOT, so we can use NOR as NOT in some cases.

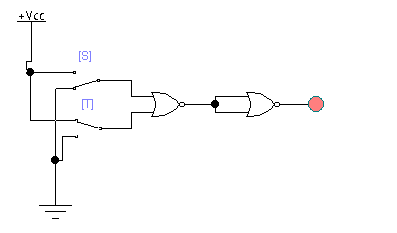
This NOT can be made by just taking the NOR of 2-inputs. As NOR gate is equivalent to OR gate by applying inverter on it.

* **2-NOR as OR**
* **Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| S | T | NOR | (ST)’ |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| S | T | OR |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* **Logic circuit Diagram**



We can replace OR gate by just taking the inverter of NOR gate. Or, by joining two NOR gate we can also make OR gate.

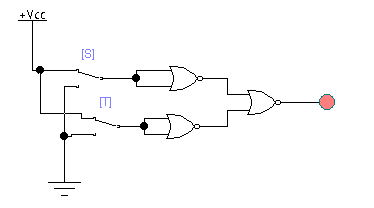
* **3-NOR as AND**

|  |  |  |
| --- | --- | --- |
| S | T | AND |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* **Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| S | T | NOR | (ST)’ |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |

* **Logic Circuit Diagram**



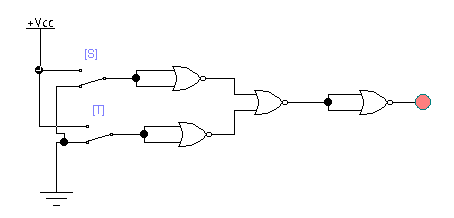
An AND gate give output “1” when both inputs are “1” The NOR gate give output “1” when both inputs are “0”. Therefore, an AND gate is made by inverting the inputs to a NOR gate.

* **4-NOR as NAND**
* **Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S | T | S’ | T’ | S’+T’ | S.T | (S.T)’ |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| S | T | (S.T)’ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* **Logic circuit Diagram**

  
The NAND gate can be replaced by using four NOR gate.

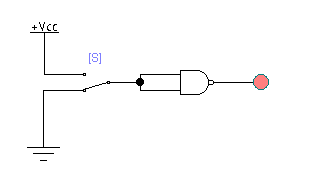
**USE OF NAND GATE**

* **1-NAND As INVERTER**
* **Truth Table**

|  |  |  |
| --- | --- | --- |
| S | T | NAND |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| S | T | (ST)’ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* **Logic circuit Diagram**



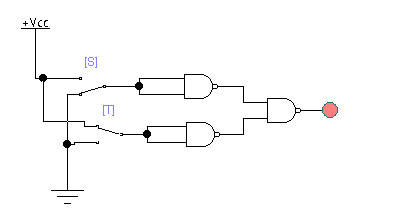
The NOT gate can be replaced by just taking the NAND of two inputs. As NAND gate is equal to AND gate by just applying inverter on it.

* **2-NAND as OR**
* **Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | T | S’ | T’ | S’.T’ | S+T |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| S | T | S+T |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* **Logic Diagram**



An OR gate can be replaced by NAND gate with all of its inputs complemented by NAND gate inverter.

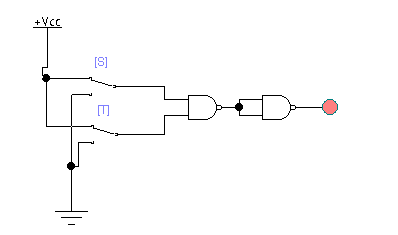
* **3-NAND as AND**

|  |  |  |
| --- | --- | --- |
| S | T | S.T |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* **Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | T | S’ | T’ | (S.T)’ | S.T |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

* **Logic circuit Diagram**



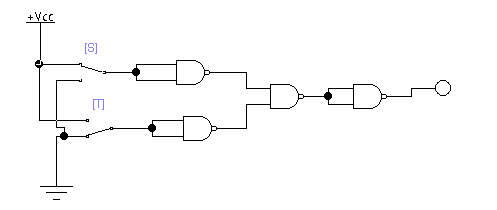
The AND gate can be replaced by NAND gates with all the inputs complemented by NAND gate inverter.

* **4-NAND as NOR**
* **Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S | T | S’ | T’ | S’.T’ | S+T | (S+T)’ |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| S | T | (S+T)’ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

* **Logic Circuit Diagram**



The AND gate can be replaced by NAND gates with all the inputs complemented by NAND gate inverter.

**Question # 08(b)**

Design 4-bit 2’s complementor combinational circuit. (The output generates the 2’s complement of the input binary number) Show that the circuit can be constructed using exclusive-OR gates?

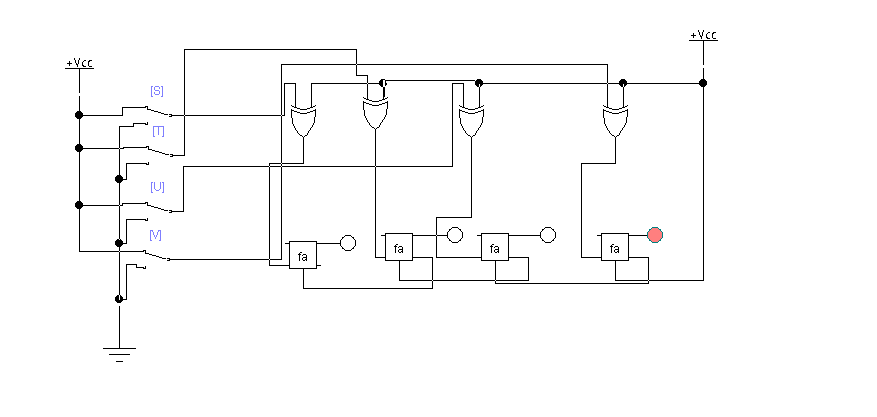
We have to design such a circuit in which the output will be the 2’s complement of the inputs such as truth table given below.

|  |  |  |  |
| --- | --- | --- | --- |
| W | X | Y | Z |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

**INPUTS OUTPUTS**

|  |  |  |  |
| --- | --- | --- | --- |
| S | T | U | V |
| 0 | 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 |
| 1 | 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 |

* **Logic Circuit Diagram:**



* **Explanation:**

Four full adders used in this circuit. As we have to design the circuit in which we give inputs and output should be the 2’s complement of binary number. So, we design the circuit given above in which four full adders are used. We give inputs 1111 then the circuit shows outputs 0001 which is the 2’s complement of the binary number 1111.

**Question # 9(a)**

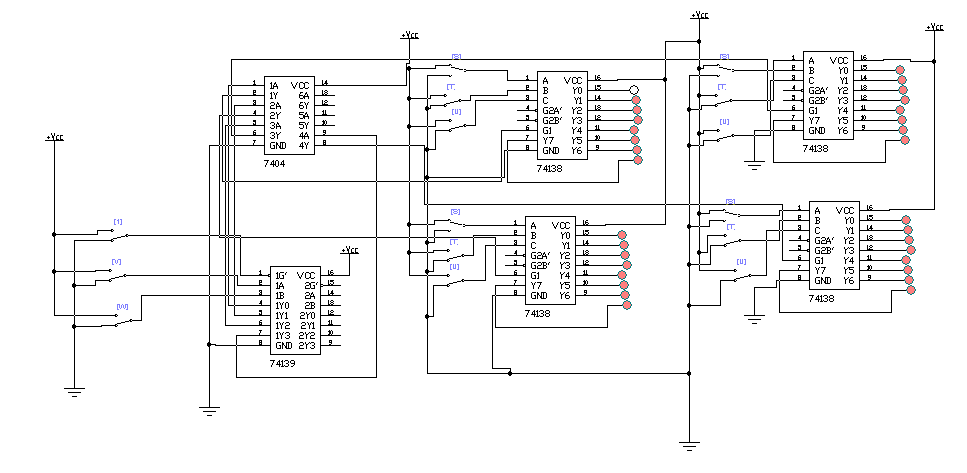
Elaborate the logic of decoder and Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable input use 2-to4- line decoder if required.

* **Decoder:**

The device that transfers binary information from n-input lines to a maximum of 2^n output lines.

A binary code of n-bits is capable of representing up to 2^n distinct element of coded information. The decoder means to decode the coded information from one format into another.

* **Logic Circuit Diagram:**



* **Explanation:**

We used four 3\*8-line decoder, one 2\*4-line decoder and one inverter in this circuit. Not gate used to reverse the output. In given above circuit we give inputs 10000, it is binary code of 17 so, we see that it will turn on all the LED except 17. 10000 turn off only one LED.

**Question # 9(b)**

Design a 3-input majority circuit by discover the circuit’s truth table, Boolean equation and a logic diagram.

* **Majority circuit:**

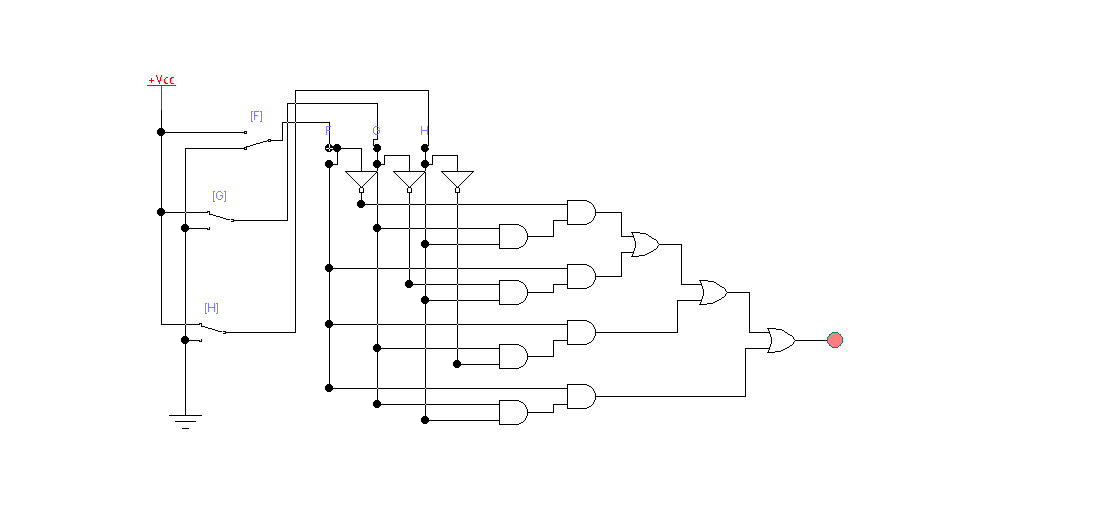
A majority circuit is a combinational circuit which shows outputs “1” when inputs variable has more 1’s then 0’s.

* **Truth Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **F** | **G** | **H** | **Output** |
| **0** | **0** | **0** | **0** |
| **0** | **0** | **1** | **0** |
| **0** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** |
| **1** | **0** | **0** | **0** |
| **1** | **0** | **1** | **1** |
| **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** |

* **Equation:**

**F’GH+FG’H+FGH’+FGH**

* **Logic Circuit Diagram:** 

**QUESTION 10(a)**

How latches are differs from flip flops? Discuss. Give the logic of how two-ve edge triggered D flip-flops cascaded with each other.

|  |  |
| --- | --- |
| **LATCHES** | **FLIP FLOPS** |
| A latch is a device it has 2 stable states high output as well as low-output. includes a feedback lane data is stored with the device it is a memory device it is used to store one bit of data. Latches are like flip-flops, however, they are not synchronous devices. They do not work on edge . | A Flip-Flop is made with couple of latches and it can be made using a NOR gate or a NAND gate. An FlipFlop can have 2-inputs, 2-outputs, a set and a reset. The type of FF is named as SR-FF. the flipflop is used to store binary values. It has an extra CLK signal that makes it different from latch. |
| The latch structure is made with logic gates | These are made with latches by adding an extra clock. |
| A latch does not contain any clock signal | A flip-flop contains a clock signal |
| Latches are fast | FlipFlops are slow. |
| They consume less power. | They consume more power. |

* **EXPLAINATION:**

The two negative edge trigerred flipflops are cascaded together in the manner they share same clock(CLK) using D flip flops, connect the Q output of the first stage to the D input of the second stage and clock them both at the same time. Q of one flop connected to the D of second flip flop. Logic is used with feedback.

**QUESTION NO 10(b)**

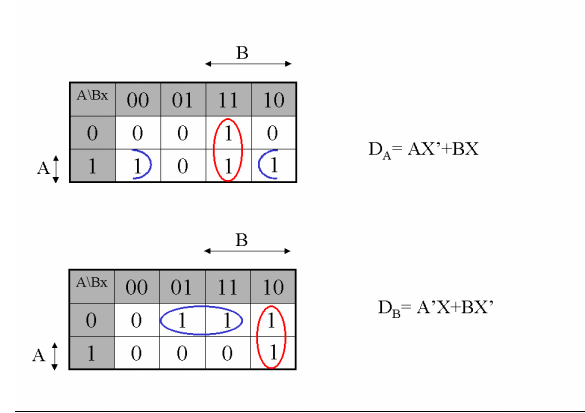
**Design a sequential circuit with two D Flip-Flops, A and B, and one input x. When x =**

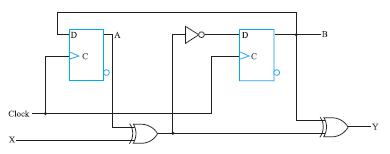
**0, then the state of the circuit remains the same. When x 1, the circuit goes through the**

**state transitions from 00 to 01 to 11 to 10 back to 00, and repeats. Hint: draw sate table which has three columns "present state", "input(x)". "next state" Determine the**

**equations using k- map and draw sequential circuit.**

|  |  |  |
| --- | --- | --- |
| **Present state**  **AB** | **Input** | **Next state**  **AB** |
| **00** | **0** | **00** |
| **00** | **1** | **01** |
| **01** | **0** | **01** |
| **01** | **1** | **11** |
| **10** | **0** | **10** |
| **10** | **1** | **00** |
| **11** | **0** | **11** |
| **11** | **1** | **10** |
|  |  |  |



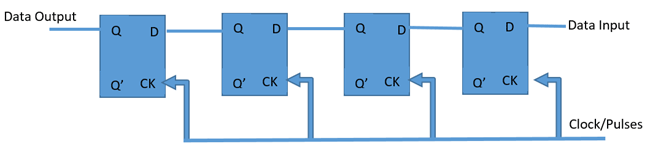
sequential circuit.

**QUESTION NO 11(a)**

Briefly Describe Shift registers. What do you understand by the term Serial-in-serial out Parallel-in-parallel-out, Serial-in-parallel -out, Parallel-in-serial-out shift register. Explain in detail with suitable example.

* **SHIFT REGISTERS:**

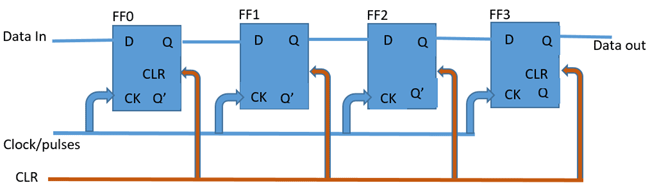
Shift Registers are sequential logic circuits, they can of store and transfer of data. They are constructed with Flip Flops which are connected as output of one flip flop serve as the input of the other flip-flop it depend on the type of shift registers being created.



**D-FlipFlop shift Register**

* **Serial in - Serial out Shift Registers**

Serial in – Serial out shift registers. These shift registers get in data serially (one bit per clock cycle) and out data too in the same way, one after the other.

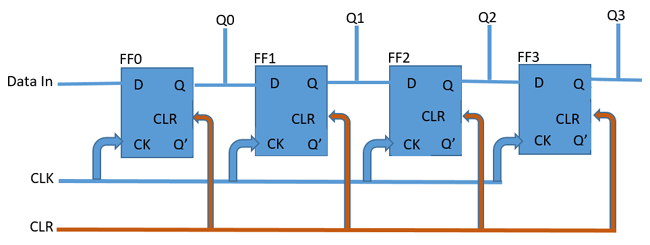


A simple serial in serial out shift register is  is shown above, it consists of 4 flip flops the shift register is first sets outputs of all flip flops to 0, the input data is then applied to the input serially, one bit at a time.

* **Serial in – Parallel out Shift Register**

This is another type of shift register the Serial in – Parallel out shift register. This shift register is used to convert data from serial to parallel. The data comes in one after the other. when the data is read in, each read in bit becomes available on their respective output line.

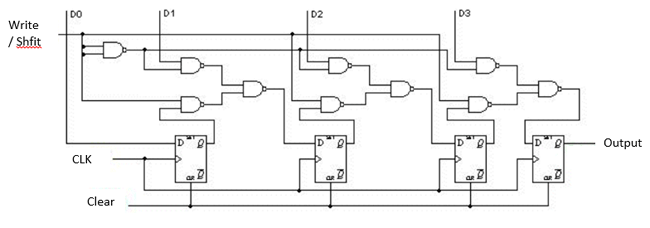
A 4-bits serial in – Parallel out shift register is shown

Serial in – Parallel out Shift Register

A good example of the serial in – parallel out shift register is the 74HC164 shift **register,** which is an 8-bit shift register.

* **Parallel in – Serial out Shift Register**

Its also a type of shift register in this the data is supplied in parallel an example of 4-bit shift register is shown below.

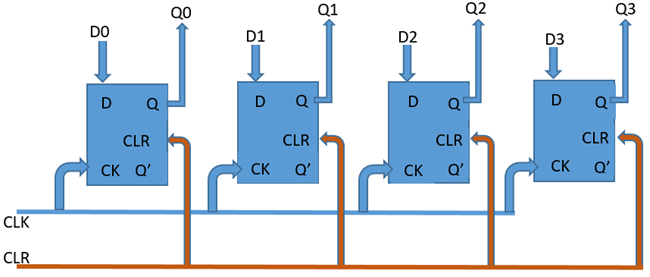


It can be used to store and shift a 4-bit word, register. data can be written and clocked in via D0 to D3. To shift the data the WS control line is high, the register then shifts the data out on clock input.

**example of a parallel in – serial out shift register is the 74HC165 8-bit shift register**

* **Parallel in – Parallel out shift register**

parallel in – parallel out shift register, the output data across the parallel outputs simultaneously as the input data is fed in.

**4 Bits register is show above**

Data input at each of the input pins from D0 to D3 in same time are read the device is clocked and the data read in from each of the inputs is passed out at output (from Q0 to Q3).

The 74HC195 shift register is a shift register which is capable of working in most of the  the modes described by all the types as well as a parallel in – parallel out shift register.

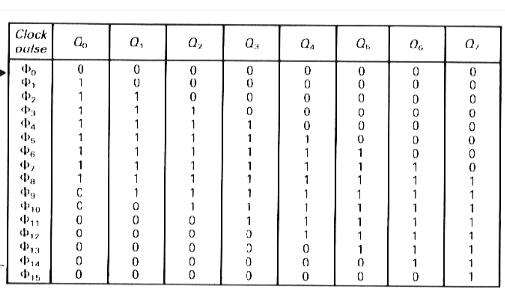
**QUESTION NO 11(b)**

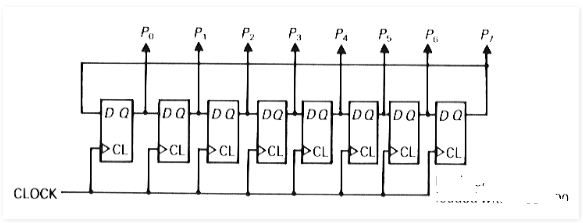
Design 8-bit Ring counter using D Flipflop. Construct truth table and give brief

description of logic implemented. (Attach Simulated circuit diagram as well). Also draw

neat & clean timing diagram.

Answer: the 8 bit ring counter is made by using 8 d flipflops.it counts an 8 bit number .





Circuit diagram

**QUESTION NO 12(a)**

Differentiate "synchronous" and "asynchronous" counters. Design the logic of Mod-n

asynchronous counter where "n" is the last digit of your roll number. (Attach Simulated

circuit diagram as well).

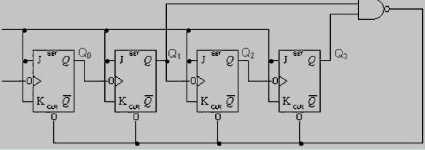
|  |  |
| --- | --- |
| **Synchronous counters** | **Asynchronous counters** |
| In synchronous counters flip flops are triggered with same clock. | In asynchronous counters flip flops are not triggered with same clock. |
| They do not produce decoding errors | Decoding errors can be produced |
| These counters are fast. | These counters are slow compared to synchronous counters. |
| Its also called parallel counter | Its called series counter |
| Its designing and implementation is  Complex | Asynchronous counters designing is easy |
| It can operate in any desired count sequence. | They work in only fixed count sequence |
| For example :Ring counter, Johnson counter. | For example: Ripple up counter , Ripple down counter. |

Logic Of Mode n Asynchronous counters.

**Roll number = 95 so (9+5=14) so, 2^4=16 so n=4**

We require 4 flipflops

states will be 0000 ,0001 …..to 1101 (which is 14th state) .when 15th state arrives 1110 , we need to reset the flops.



Simulated circuit diagram

**QUESTION NO 12(b)**

sequential circuit with two D Flip-Flops, A and B two inputs, x and y; and one

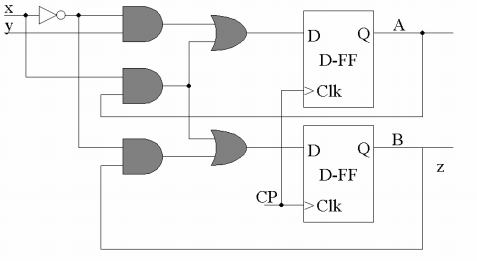
output, z, is specified by the following next-state and output equations:

**A(t+1) x'y + XA**

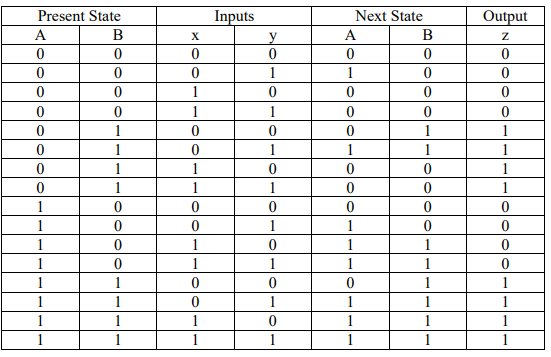
**B(t+1) - X'B + XA**

**Z=B**

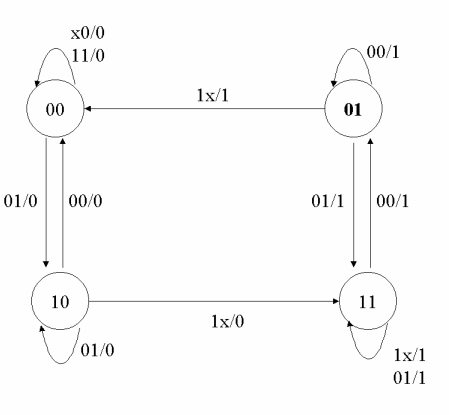
* **Draw the logic diagram of the circuit**



* **List the state table for the sequential circuit.**



* **Draw the corresponding state diagram.**



**QUESTION NO 13(a)**

What do you understand by the term "Simple Programmable Logic Devices (SPLDS)".

Elaborate the working of PAL GAL and PLA.

* **Simple programmable logic device**

The SPLDS are programable logic device . they have complexity less then complex programmable device.

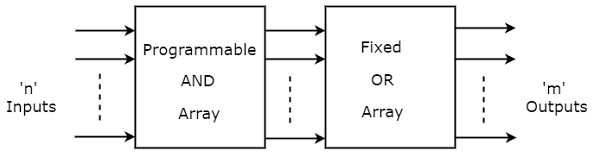
The term commonly refers to devices as ROMS PALS PLA GLAs etc.

These are the simplest smallet and least expensive programmable devices. These are used in bards to replace AND , OR, and NOT gates. These are made of 4 to 22 macrocells. These macro cells are made up of combinational logics and flip flops,

**Working of PAL, GAL and PLA**

* **PAL:**

PaL is a type of PLD it is used to realize a particular logical function. PALs are made of an AND gate array followed by an OR gate .array only the AND gate array is programmable and the OR gate array conatins a fixed logicthe reason is here inputs are sent to AND gate which act as programmable links. The structure of PALs is better then PLSs .

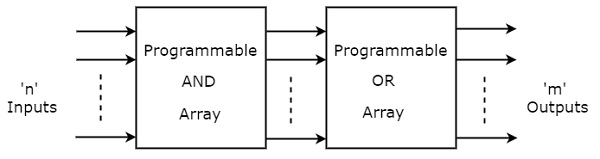


the user to decide the connection between the input lines and the AND gates. We have to connect each and every input line to either single or multiple AND gate(s), depending on the logic . the outputs from AND gate serve as inputs of OR gate through wired connections. All gates don’t have same connections some gates have multiple inputs including the outputs of AND gate .However OR gate has output of AND gates as input. Then there are n output lines of OR gate array.

* **PLA:**

The full form of the PLA is a programmable logic array. The construction of PLA can be done using the programmable collection of AND & fixed collection of OR gates. The availability of PLA is more.its cost is affordable.the number of fuctions implemented is less but speed is high.PLA is used to implement combinational logic circuits . The PLA has a set of programmable it has set of AND gates which connected to set of programmab OR gate It has 2N AND Gates for N input variables, and for M outputs from PLA, there should be M OR Gates, each with programmable inputs from all of the AND gates. This allows logic functions to be in the sum of products form.

PLAs are different from pals and gals because it has both AND and OR gates programmable. PLA Prepares SOP ([sum of products](https://en.wikipedia.org/wiki/Sum_of_products)) form. Then Obtain the minimum SOP form to reduce the number of product terms to a minimum.Decide the input connection of the AND matrix for generating the required product term.Then decide the input connections of OR matrix to generate the sum terms then Decide the connections of invert matrix.and Program the PLA.

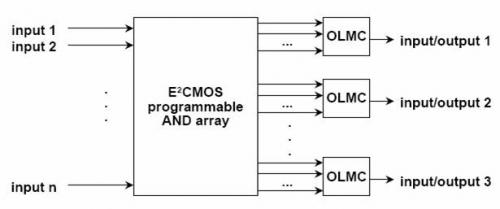


That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required product terms by using these AND gates.

Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of sum of products form.

* **GLA:**

A generic array logic (GAL) same as that of the PAL architecture. The only difference between GAL and PAL is that, the programmable AND array of a GAL device can be erased and reprogrammed. Further, theÂ output logic of GAL device is re-programmable. Because of this facility in GAL device is that the implemented logic can be corrected by reprogramming. Output logic macrocell can be configured either for a combinational output of for a registered output. GAL can be erased and reprogrammed and usually replace a whole set of different PALs.



QUESTION NO 13(b)

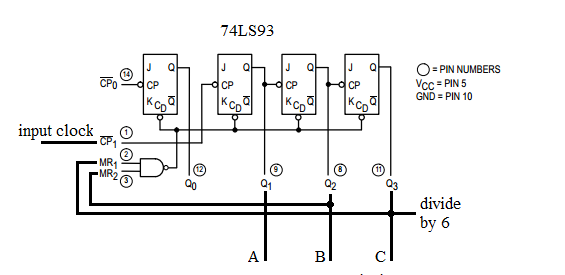
Construct digital clock circuit for Hours and Seconds. (Attach Simulated circuit

diagram as well).

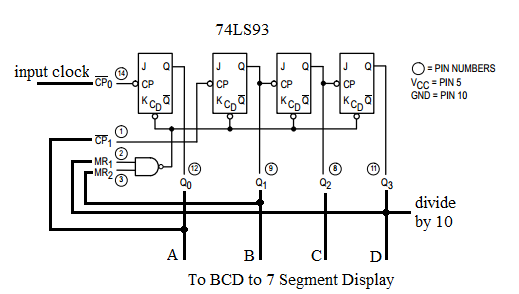
Answer

the 74LS93 is used to implement the divide by 10 and divide by 6 circuits. The 74LS93 is a high-speed 4-bit ripple type counters partitioned into two sections. The counter has a divide-by-two section and divide-by-eight section which are triggered by a HIGH-to-LOW transition on the clock inputs.

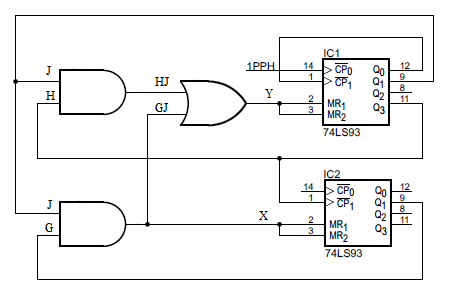
Divide by 10 counter



* **Divide by 6 counter**



* **clock**



**REFRENCES**

**QUESTION NO 1(a)**

* <https://www.geeksforgeeks.org/combinational-circuit/>
* <https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm>
* <https://study.com/academy/lesson/basic-combinational-circuits-types-examples.html>

**QUESTION NO 1(b)**

* <https://faculty.kfupm.edu.sa/COE/abouh/Lesson3_3.pdf>
* <https://www.gatevidyalay.com/carry-look-ahead-adder/>
* <https://users.encs.concordia.ca/~asim/coen312/Lectures/CLA_adder.pdf>

**QUESTION NO 2(a)**

* <https://faculty.kfupm.edu.sa/COE/abouh/Lesson3_3.pdf>
* <https://www.gatevidyalay.com/carry-look-ahead-adder/>
* <https://users.encs.concordia.ca/~asim/coen312/Lectures/CLA_adder.pdf>

**QUESTION NO 2(b)**

* <http://cs.baylor.edu/~maurer/aida/logicdes/logicdes.pdf>
* <https://circuitdigest.com/tutorial/full-adder-circuit-theory-truth-table-construction>
* <http://www.c-jump.com/CIS77/CPU/Overflow/lecture.html>

**QUESTION NO 3(a)**

* <http://mathworld.wolfram.com/GrayCode.html>
* <http://mathworld.wolfram.com/GrayCode.html>
* <https://en.wikipedia.org/wiki/Gray_code#/media/File:Binary-reflected_Gray_code_construction.svg>

**QUESTION NO 3(b)**

* <https://en.wikipedia.org/wiki/Gray_code#History_and_practical_application>
* Book\_Reference\_Digital\_Fundamental\_by\_Thomas\_Floyd\_Eighth\_Edition
* <http://mathworld.wolfram.com/GrayCode.html>

**QUESTION NO 4(a)**

* Book\_Reference\_by\_M\_Morris\_Fifth\_Edition
* <https://www.techopedia.com/definition/1801/even-parity>
* <https://www.techopedia.com/definition/1801/odd-parity>

**QUESTION NO 4(b)**

* <https://www.electronicshub.org/bcd-7-segment-led-display-decoder-circuit/>
* <https://www.elprocus.com/bcd-to-seven-segment-display-decoder-theory/>
* <http://electronics-course.com/bcd-7-segment>

**QUESTION NO 5(a)**

* Digital logic and computer design fundamentals by m . morris mano and Charles r rime ,chapter 3
* [**https://www.tutorialspoint.com/computer\_logical\_organization/logic\_gates.htm**](https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm)
* **https://www.electronics-tutorials.ws/waveforms/waveforms.html**

**QUESTION NO 5(b)**

* **Digital logic and computer design fundamentals by m . morris mano and Charles r rime chapter 4**
* [**https://www.tutorialspoint.com/computer\_logical\_organization/logic\_gates.htm**](https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm)
* [**https://www.tutorialspoint.com/computer\_logical\_organization/logic\_gates.htm**](https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm)

**QUESTION NO 6(a)**

* **Digital logic and computer design fundamentals by m . morris mano and Charles r rime ,chapter no 2**
* **https://stackoverflow.com/questions/30623006/circuit-design-that-outputs-square-of-binary-input**
* Digital logic and computer design fundamentals by m . morris mano and Charles r rime ,chapter 3

**QUESTION NO 6(b)**

* [**https://www.electronics-tutorials.ws/combination/comb\_2.html**](https://www.electronics-tutorials.ws/combination/comb_2.html)
* [**https://www.electronics-tutorials.ws/logic/logic\_6.html**](https://www.electronics-tutorials.ws/logic/logic_6.html)
* Digital logic and computer design fundamentals by m . morris mano and Charles r rime ,chapter 4

**QUESTION NO 7(a)**

* [**https://www.elprocus.com/multiplexer-and-demultiplexer/**](https://www.elprocus.com/multiplexer-and-demultiplexer/)
* [**https://www.watelectronics.com/different-types-encoder-decoder-applications/**](https://www.watelectronics.com/different-types-encoder-decoder-applications/)
* **Digital logic and computer design fundamentals by m . morris mano and Charles r rime ,chapter no 4**

**QUESTION NO 7(b)**

* **Digital logic and computer design fundamentals by m . morris mano and Charles r rime ,chapter no 4**
* **https://www.massey.ac.nz/~mjjohnso/notes/59233/lect2.html**
* **https://www.scribd.com/doc/307897332/Full-Adder-Using-Multiplexer**

**Verification**

This is to verify that the deep analysis has been made to test the document thoroughly and we,

**Muhammad Faraz Ansar,Hassan-Ur-Rehman, Abdul Ahad,Tanveer Ahmed and Jahanzaib Rashid** ensures that document contain NO PLAGARISIM .

Signatures of group members with names:

1. **MUHAMMAD FARAZ ANSAR ………………………………………………… FARAZ**
2. **HASSAN-UR-REHMAN ……………………………………………….HASSAN**
3. **ABDUL AHAD ………………………………………………. AHAD**
4. **TANVEER AHMED ………………………………………….. TANVEER**
5. **JAHANZAIB RASHID ………………………………………….. JAHANZAIB**

1. [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)