BM T43 - DIGITAL LOGIC THEORY AND DESIGN

UNIT – 1

PART – A

1. State De-Morgan's Theorems. (May 2016, May 2019)

De-Morgan's Theorems:

- Two theorems that are an important part of Boolean algebra were proposed by De-Morgan.
- The first theorem states that the complement of a product is equal to the sum of the complements.

$$(AB)' = A' + B'$$

- The second theorem states that the complement of a sum is equal to the product of the complements.

$$(A+ B)' = A'. B'$$

2. Compare two main features of TTL and CMOS logic gates. (May 2016)

1. For TTL it is 4.75 V to 5.25 V while for CMOS it ranges between 0 to 1/3 VDD at a low level and 2/3VDD to VDD at high levels.
2. CMOS technology is **more economical** and preferred more as compared to the TTL logic.
3. The current requirements of the CMOS are low compared to TTL and thus **power consumption is limited**. Therefore it is easier for the circuits to be designed with the best power management.

3. Write the maxterms corresponding to the logical expression. Y = (A + B + C') (A + B' + C') (A' + B' + C). (Nov. 2016)

Y = (A + B + C')(A + B' + C')(A' + B' + C)

=M1.M3.M6

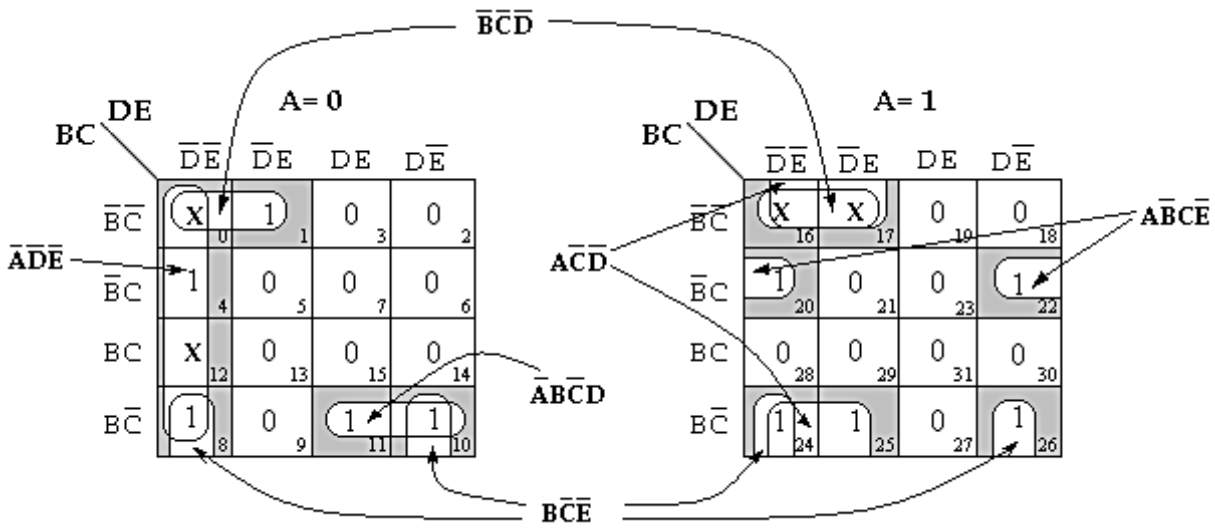
=∏M(1,3,6)

4. Why NAND and NOR gates are called as universal gates? (Nov. 2016)

The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates.

5. Minimize the Boolean expression F (A, B, C, D, E) = ∑m (1, 4, 8, 10, 11, 20, 22, 24, 25, 26)+∑d (0, 12, 16, 17) by k-map method. (May 2017)

Soln:



F (A, B, C, D, E) = B’C’D’+ A’D’E’+ BC’E’+ A’BC’D+ AC’D’+ AB’CE’

6. What is a Hamming code? (May 2017)

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver.

Hamming code can be generated using ‘even parity’ or ‘odd parity’.

7. State duality principle. (Dec. 2017)

Duality property states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

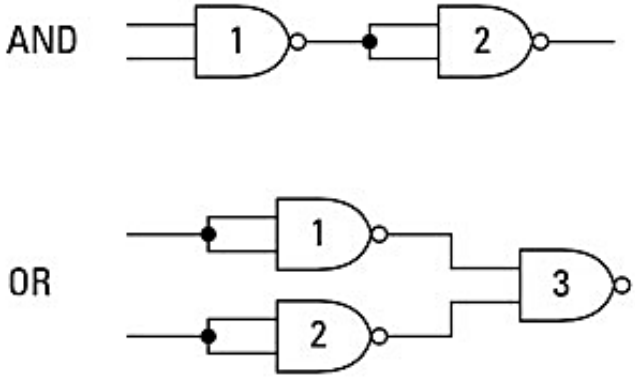
8. What are universal gates? (Dec. 2017)

1. NAND gate
2. NOR gate

9. Convert the following binary code into gray code. 1010111000_2 (May 2018)

Gray Code = 1111100100

10. Construct OR gate and AND gate using NAND gate. (May 2018)



11. Convert decimal 25 to XS3 and to gray. (May 2019)

25 in XS3

According to excess-3 code we need to add 3 to both digit in the decimal number then convert into 4-bit binary number for result of each digit. Therefore, 25 + 33 = 58 = BCD value

XS3 of 25 is 0101 1000

25 in gray code

Divide 25 by 2 = 12
12 divide by 2 = 6
6 divide by 2 = 3
3 divide by 2 = 1
I divide by 2 = 0
The remainder is 10011 rewrite in reverse order 11001
Perform the EX-OR operation for each term in binary

Binary	Gray code
1	1
1	1 1 = 0
0	1 0 = 1
0	0 0 = 0
1	0 1 =1

Gray code 25=10101

12. Determine the decimal value of the fractional binary number 0.10101. (Dec. 2019)

$$0\cdot2^{0}+1\cdot2^{-1}+0\cdot2^{-2}+1\cdot2^{-3}+0\cdot2^{-4}+1\cdot2^{-5} = 0\cdot1+1\cdot0.5+0\cdot0.25+1\cdot0.125+0\cdot0.0625+1\cdot0.03125$$
$$= 0+0.5+0+0.125+0+0.03125$$
$$= 0.65625_{10}$$

$0.10101_2 = 0.65625_{10}$

13. Prove that X(X +Z) =X, using Boolean algebra. (Dec. 2019)

X(X +Z) =X

L.H.S. = X (X + Z)

=X X +XZ By distributive law

=X +XZ, as X.X = X

=X (1 +Z), As 1 +Z =1

= X.1 = X

L.H.S. = R.H.S.

PART - B

1. Generate hamming code for a 4-bit excess-3 message to detect and correct single bit errors. (May 2016)

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver.

Hamming code can be generated using ‘even parity’ or ‘odd parity’.

A detailed explanation of generating a Hamming code is given below.

Given that;

To generate Hamming code for a 4-bit excess 3 code.

An excess 3 code is nothing but the adding of a binary 3 to the binary code.

Decimal	Binary code(BCD)	Excess-3 code
	8 4 2 1	BCD + 011
0	0 0 0 0	0 0 1 1
	0 0 0 0	
	+	
	0 0 1 1	
	_____-	
	0 0 1 1	
	_____-	

1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	.
7	0 1 1 1	.
.	.	.
.	.	.

Picking a 4-bit excess 3 code message;

Let the message m = (0 1 0 1)

<u>Generating a Hamming code (using even parity):</u>

Message (m) = (0 1 0 1)

To determine the parity bits:

2^P > or = P+m+1

Here m = 4 (Given it is 4-bit message)

Let P = 1

2^1 > or = 1+4+1 ? No

Let P = 2

2^2 > or = 2+4+1 ? No

Let P = 3

2^3 > or = 3+4+1 ? Yes

So, the number of parity bits = 3 (P1, P2, P3)

Hence, the Hamming code consists of m+P = 4+3 = 7 bits

<u>Hamming code:</u>

1	2	3	4	5	6	7
P1	P2	m1	P3	m2	m3	m4
P1	P2	0	P3	1	0	1

Parity bits lie in the power of 2 positions (2^0, 2^1, …..)

We simply fill the message bits in the remaining positions.

Finding the parity bits (using even parity):

P1 —> 1 3 5 7 —> P1 0 1 1 => P1 = 0

Explanation:

P1 is the first parity bit.

We need to pick the bits from the positions 1 3 5 7. This is because these digits in binary format contain a 1 in the 1st position.

1. 0 0 0 1

3. 0 0 1 1

5. 0 1 0 1

7. 0 1 1 1

So we get, (P1 0 1 1).

Since, we are considering even parity, P1 = 0, so that there are even number of 1's.

Similarly, we determine other values of P

P2 —> 2 3 6 7 —> P2 0 0 1 => P2 = 1

[2, 3, 6, 7 contain a 1 in the 2nd position]

P3 —> 4 5 6 7 —> P3 1 0 1 => P3 = 0

[4, 5, 6, 7 contain a 1 in the 3rd position]

Now, we have the parity bits and the message bits.

The generated hamming code is;

(P1 P2 m1 P3 m2 m3 m4)

0 1 0 0 1 0 1

2. Simplify the expression Y = ABCD + AB'C'D' + AB'C +AB. Convert this expression into a minterm SOP form and then simplify using Karnaugh Map. (May 2016, May 2019, Sept. 2020)

Y= ABCD+ AB'C'D'+ AB'C+ AB

= ABCD+ AB'C'D'+ AB'C (D+D')+ AB (C+C') (D+D')

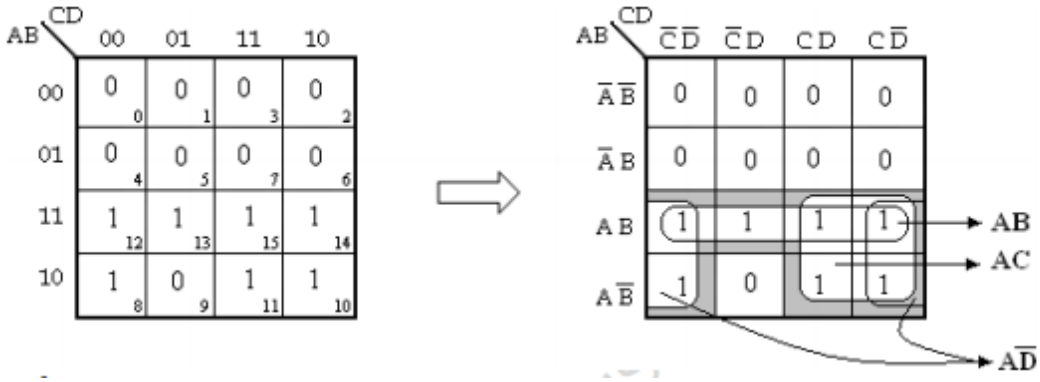
= ABCD+ AB'C'D'+ AB'CD+ AB'CD'+ (ABC+ ABC') (D+ D')

= ABCD+ AB’C’D’+ AB’CD+ AB’CD’+ ABCD+ ABCD’+ ABC’D+ ABC’D’

= ABCD+ AB’C’D’+ AB’CD+ AB’CD’+ ABCD’+ ABC’D+ ABC’D’

= m15+ m8+ m11+ m10+ m14+ m13+ m12

= ∑ m (8, 10, 11, 12, 13, 14, 15)



Therefore, Y= AB+ AC+ AD’

3. Determine the prime implicants of the following function and verify using K map F (A, B, C, D) = ∑(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15). (Nov. 2016)

The prime implicant is essential if it is the only prime implicant that covers the minterm. The minterms of the function are marked with 1’s in the maps of Fig. 3.11 . The partial map (Fig. 3.11(a)) shows two essential prime implicants, each formed by collapsing four cells into a term having only two literals.

One term is essential because there is only one way to include minterm m0 within four adjacent squares. These four squares define the term BD. Similarly, there is only one way that minterm m5 can be combined with four adjacent squares, and this gives the second term B’D’.

The two essential prime implicants cover eight minterms. The three minterms that were omitted from the partial map (m3, m9, and m11) must be considered next. Figure 3.11 (b) shows all possible ways that the three minterms can be covered with prime implicants.

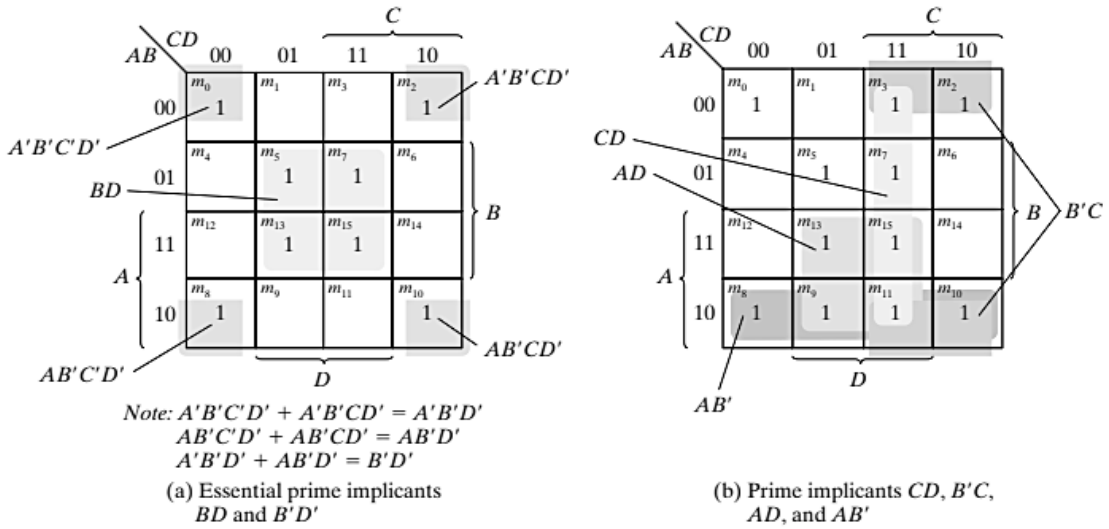


FIGURE 3.11
Simplification using prime implicants

Minterm m3 can be covered with either prime implicant CD or prime implicant B'C. Minterm m9 can be covered with either AD or AB'. Minterm m11 is covered with any one of the four prime implicants. The simplified expression is obtained from the logical sum of the two essential prime implicants and any two prime implicants that cover minterms m3, m9, and m11. There are four possible ways that the function can be expressed with four product terms of two literals each:

$$
\begin{aligned}
F &= BD + B'D' + CD + AD \\
&= BD + B'D' + CD + AB' \\
&= BD + B'D' + B'C + AD \\
&= BD + B'D' + B'C + AB'
\end{aligned}
$$

The previous example has demonstrated that the identification of the prime implicants in the map helps in determining the alternatives that are available for obtaining a simplified expression. The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants. The simplified expression is obtained from the logical sum of all the essential prime implicants, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants. Occasionally, there may be more than one way of combining squares, and each combination may produce an equally simplified expression.

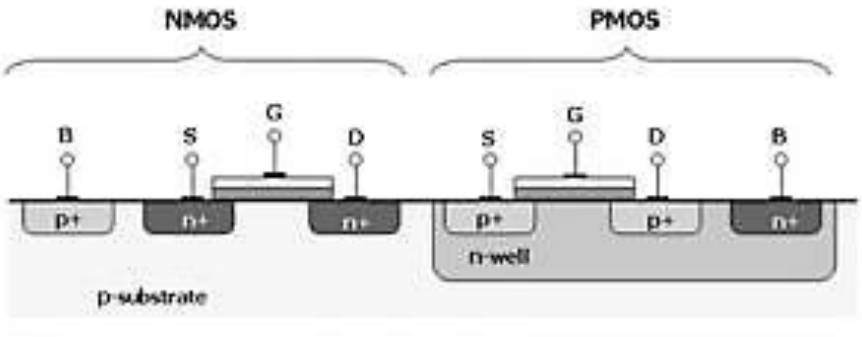
4. Explain the digital logic families: (a) CMOS logics (b) TTL. (Nov. 2016, Dec. 2019)

Digital Logic Family

There are various logic families namely – Diode logic (DL) Resistor-Transistor logic (RTL) Diode-Transistor logic (DTL) Emitter coupled logic (ECL) Logic families can be classified broadly according to the technologies they are built with Emitter coupled logic (ECL) Transistor-Transistor logic (TTL) CMOS logic TTL and CMOS logic family is most widely used IC technologies. Within each family, several subfamilies of logic types are available, with different rating for speed, power consumption, temperature range, voltage level and current level.

(a) CMOS (Complementary Metal Oxide Semiconductor)

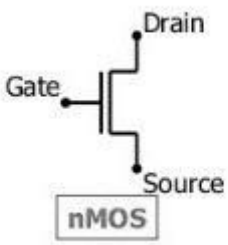
The main advantage of CMOS over NMOS and BIPOLAR technology is the much smaller power dissipation. Unlike NMOS or BIPOLAR circuits, a Complementary MOS circuit has almost no static power dissipation. Power is only dissipated in case the circuit actually switches. This allows integrating more CMOS gates on an IC than in NMOS or bipolar technology, resulting in much better performance. Complementary Metal Oxide Semiconductor transistor consists of P-channel MOS (PMOS) and N-channel MOS (NMOS). Please refer to the link to know more about the fabrication process of CMOS transistor.



CMOS (Complementary Metal Oxide Semiconductor)

NMOS

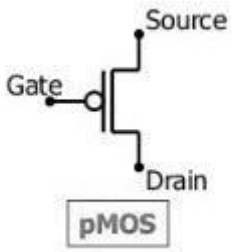
NMOS is built on a p-type substrate with n-type source and drain diffused on it. In NMOS, the majority of carriers are electrons. When a high voltage is applied to the gate, the NMOS will conduct. Similarly, when a low voltage is applied to the gate, NMOS will not conduct. NMOS is considered to be faster than PMOS, since the carriers in NMOS, which are electrons, travel twice as fast as the holes.



NMOS Transistor

PMOS

P- channel MOSFET consists of P-type Source and Drain diffused on an N-type substrate. The majority of carriers are holes. When a high voltage is applied to the gate, the PMOS will not conduct. When a low voltage is applied to the gate, the PMOS will conduct. The PMOS devices are more immune to noise than NMOS devices.

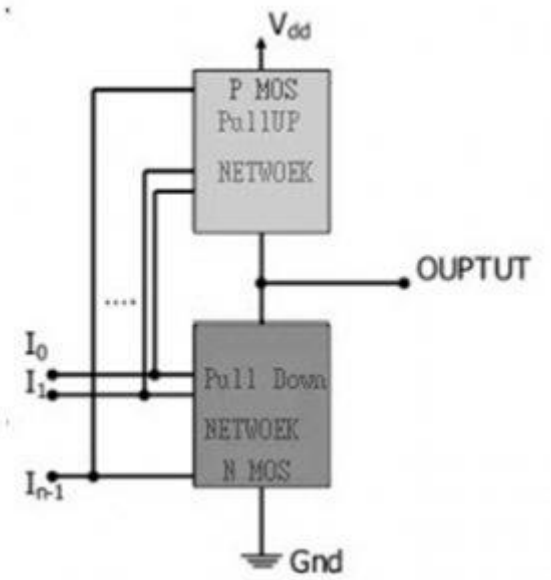


PMOS Transistor

CMOS Working Principle

In CMOS technology, both N-type and P-type transistors are used to design logic functions. The same signal which turns ON a transistor of one type is used to turn OFF a transistor of the other type. This characteristic allows the design of logic devices using only simple switches, without the need for a pull-up resistor.

In CMOS logic gates a collection of n-type MOSFETs is arranged in a pull-down network between the output and the low voltage power supply rail (Vss or quite often ground). Instead of the load resistor of NMOS logic gates, CMOS logic gates have a collection of p-type MOSFETs in a pull-up network between the output and the higher-voltage rail (often named Vdd).



CMOS using Pull Up & Pull Down

Thus, if both a p-type and n-type transistor have their gates connected to the same input, the p-type MOSFET will be ON when the n-type MOSFET is OFF, and vice-versa. The networks are arranged such that one is ON and the other OFF for any input pattern as shown in the figure.

CMOS offers relatively high speed, low power dissipation, high noise margins in both states, and will operate over a wide range of source and input voltages (provided the source voltage is fixed). Furthermore, for a better understanding of the Complementary Metal Oxide Semiconductor working principle, we need to discuss in brief CMOS logic gates as explained below.

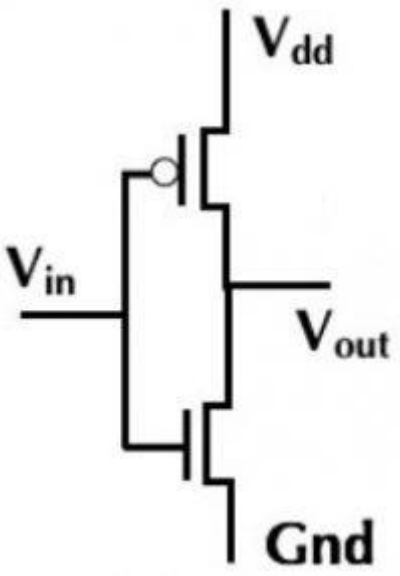
Which Devices use CMOS?

Technology like CMOS is used in different chips like microcontrollers, microprocessors, SRAM (static RAM) & other digital logic circuits. This technology is used in a wide range of analog circuits which includes data converters, image sensors & highly incorporated transceivers for several kinds of communication.

CMOS Inverter

The inverter circuit as shown in the figure below. It consists of PMOS and NMOS FET. The input A serves as the gate voltage for both transistors.

The NMOS transistor has input from Vss (ground) and the PMOS transistor has input from Vdd. The terminal Y is output. When a high voltage (~ Vdd) is given at input terminal (A) of the inverter, the PMOS becomes an open circuit, and NMOS switched OFF so the output will be pulled down to Vss.



CMOS Inverter

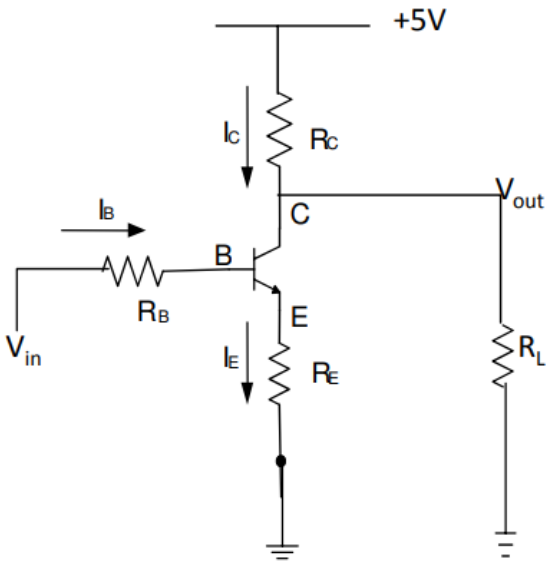
When a low-level voltage (<Vdd, ~0v) applied to the inverter, the NMOS switched OFF and PMOS switched ON. So the output becomes Vdd or the circuit is pulled up to Vdd.

(b) TTL (Transistor-Transistor logic)

Transistor Transistor Logic (TTL) Family One basic function of TTL IC is as a complimenting switch or inverter. When Vin equals 1 (+5V), the transistor is turned on (saturation) and Vout equals 0 (0V). When Vin equals 0 (0V), the transistor is turned off and Vout equals 1 (5V), assuming RL > R C

$$V_{out} = V_{CC} \frac{R_L}{(R_C + R_L)}$$

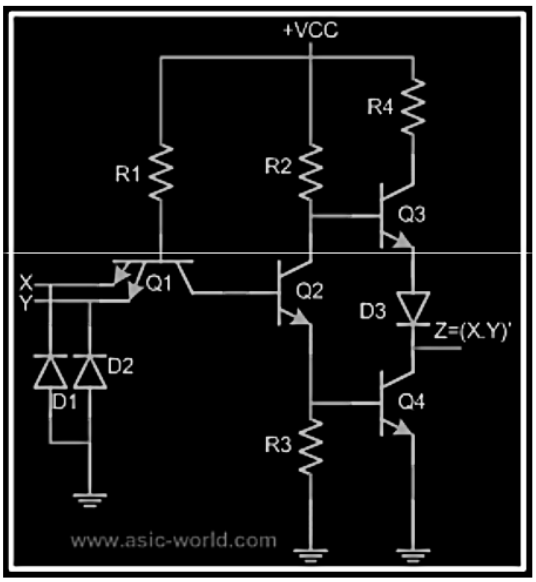
Thus level 1 of inverter output is very much dependent on RL , which can typically vary by factor of 10. Thus we need very small R C compared to RL i.e. RL >> RC .



But when transistor is saturated (Vout = 0V), IC will be very large if RC is very small. Thus we need large RC when transistor is in satuaration and small RC when transistor is off.

The idea of variable R C is accommodated by TTL IC.

Transistor Transistor Logic (TTL) Family Q3 is cutoff (act like a high RC) when output transistor Q4 is saturated and Q3 is saturated (act like a low RC) when output transistor Q4 is cutoff .



Thus one transistor is ON at one time. The combination of Q3 and Q4 is called The idea of variable R C is accommodated by TTL IC. It uses another transistor Q3 in place of R C to act like a varying resistance. The combination of Q3 and Q4 is called totem pole arrangement. Q1 is called input transistor, which is multi-emitter transistor, that drive transistor Q2 which is used to control Q3 and Q4. Diode D1 and D2 is used to protect Q1 from unwanted negative voltages and diode D3 ensures when Q4 is ON, Q3 is OFF. Multi-emitter input transistor is a striking feature of TTL logic family.

5. With the necessary circuit diagram, explain the features and working of TTL NAND gate. (May 2017, Sept. 2020)

TTL gates use a 5V(±0.25V) supply, and are capable of high-speed operation. Over 600 different logic ICs are available, covering a very wide range of digital functions. Due to the use of bipolar transistors, TTL has much higher power consumption than similar CMOS types, when working at relatively low frequencies. As the frequency of signals handled increases however, this difference decreases as the power consumption of CMOS increases and TTL power consumption remains nearly constant.

TTL NAND gate

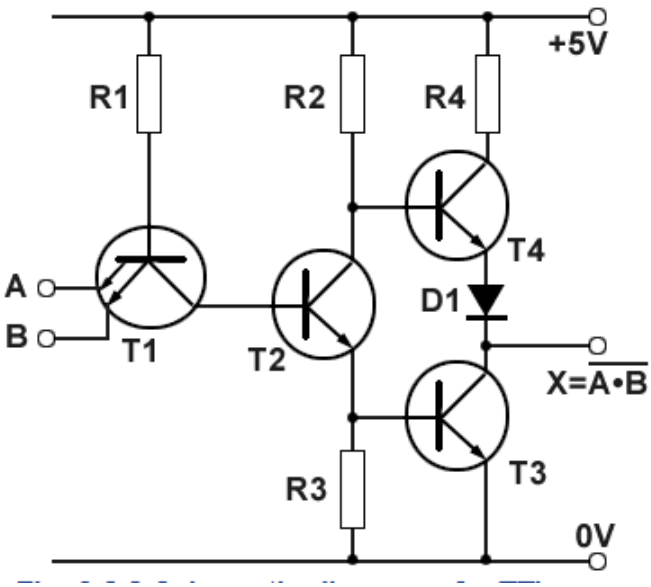
- This circuit looks similar to those found in analogue push pull amplifiers, except that the transistors here are driven either into cut-off or saturation, rather than working in their linear operating condition. Also, being constucted within an IC, it can use a

device not normally found in conventional analogue amplifiers, a multi emitter transistor.

- R1 is a low value resistor (about 4K) and as the base current of T1 is small, the base voltage is about +5V. If both emitters of T1 are at logic 1, (also around +5V), there will be very little potential difference between base and emitter, and T1 will be turned off. As T1 is not conducting, its collector will also be at about 5V, and due to this high potential, T2 base will have a higher potential than its emitter, which will cause T2 to conduct heavily and go into saturation.

- T2 collector will therefore fall to a low potential, and the emitter voltage of T2 will rise due to the current flow through R3. The voltage across R3 will rise to a sufficient level (about 0.7V) to fully turn on T3. As T3 saturates, its collector voltage will fall to about 0.2V, thus giving a logic 0 state at the output terminal.

- T4 emitter voltage is made up of T3 V_{CE} (about 0.2V) plus the forward voltage drop across D1, which will be about 0.7V, giving an emitter potential of 0.2V + 0.7V = 0.9V, the same as its base voltage.



Schematic diagram of a TTL NAND Gate

- The base potential of T4 is made up of T3 base/emitter potential V_{BE} (about 0.7V), plus the collector/emitter, potential (V_{CE}) of T2, (about 0.2V), giving a base voltage for T4 of about 0.9V. Therefore the base and emitter voltages on T4 are approximately equal, so T4 will be turned off.

- With BOTH input terminals are at logic 1 therefore, the output terminal will be at logic 0, the correct operation for a NAND gate.

- If either one of the inputs is taken to logic 0 however, this will make T1 conduct, as the emitter that is at logic 0 will be at a lower voltage than that supplied to the base by R1. This will cause T1 to saturate, taking its collector to a low potential (less than 0.8V) and as this is also connected to T2 base T2 will turn off, making its collector voltage and T4 base voltage, rise to very nearly +Vcc.

- As virtually no current (I_{CE}) is flowing through T2 collector/emitter circuit, practically no voltage is developed across the emitter resistor R3, reducing T3 base voltage to 0V, and so T3 is turned off. However, sufficient current will be flowing out of the output terminal (feeding the next gate input circuit) to cause T4 emitter to be held at about 4.1V. This is 0.9V below +Vcc, made up of the voltage across D1 (0.7V) plus the saturation voltage V_{CE} of T4 (0.2V). This places about 4V or logic 1 (between 2.4V and 5V) on the output terminal.

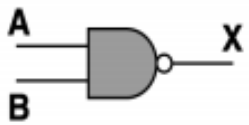
6. (a) Draw the logic symbol and construct the truth table for each of the following gates: (i) Two input NAND gate (ii) Three input OR gate (iii) NOT gate. (May 2017)

NAND GATE

Boolean Expression

$X = (A \cdot B)'$

Logic Diagram Symbol



Truth Table

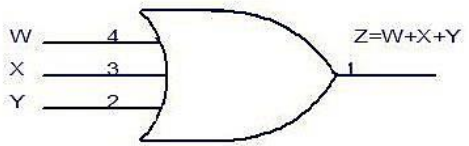
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

OR GATE

Boolean Expression: X = A + B + C

3 Input OR Gate

OR Gate



TRUTH TABLE

INPUTS			OUTPUT
W	X	Y	Z
0	0	0	0
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	1

NOT GATE

Boolean Expression	Logic Diagram Symbol	Truth Table
X = A'		

A	X
0	1
1	0

6. (b) State and explain De-Morgan's Theorem. (May 2017)

De-Morgan's Theorem

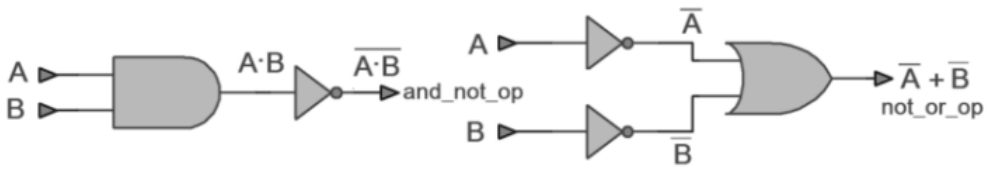
De-Morgan was a great logician and mathematician. He had contributed much to logic. Among his contribution the following two theorems are important.

De-Morgan's First Theorem

It States that —The complement of the sum of the variables is equal to the product of the complement of each variable. This theorem may be expressed by the following Boolean expression.

$$(AB)' = A' + B'$$

Digital Circuit



Truth table

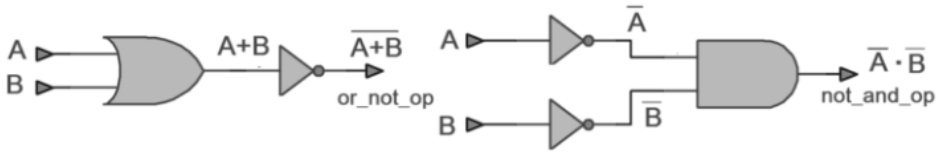
A	B	$\overline{A}$	$\overline{B}$	A·B	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

De-Morgan‘s Second Theorem

It states that the ―Complement of the product of variables is equal to the sum of complements of each individual variable. Boolean expression for this theorem is

$$(A+B)' = A'.B'$$

Digital Circuit



Truth table

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

7. Using K-map reduce the following Boolean expression and realize it in both POS and SOP form. F (A, B, C, D) = (0, 1, 2, 5, 8, 9, 10).(Dec. 2017)

The 1's marked in the map represent all the minterms of the function. The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F. Combining the squares with 1's gives the simplified function in sum-of-products form:

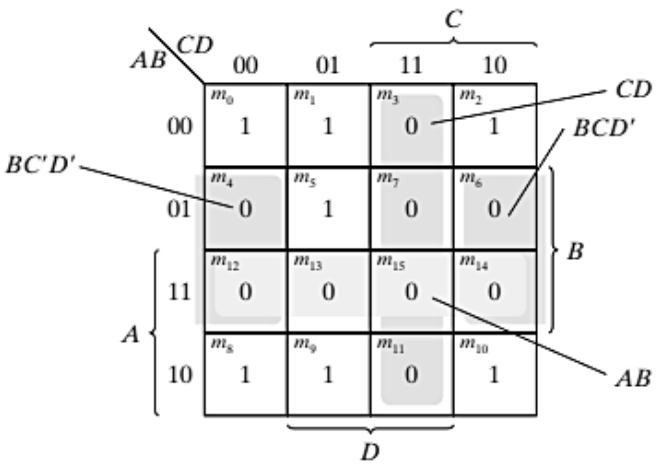
(a) F = B'D' + B'C' + A'C'D

If the squares marked with 0's are combined, as shown in the diagram, we obtain the simplified complemented function:

F' = AB + CD + BD'

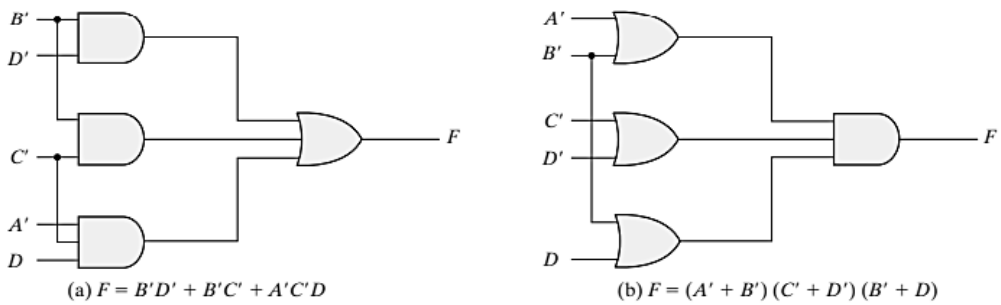
Applying DeMorgan's theorem, we obtain the simplified function in product of-sums form:

(b) F = (A' + B') (C' + D') (B' + D)



Note: BC'D' + BCD' = BD'

F (A, B, C, D) = (0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D = (A' + B') (C' + D') (B' + D)



(a) $F = B'D' + B'C' + A'C'D$ (b) $F = (A' + B')(C' + D')(B' + D)$

FIGURE 3.13
Gate implementations of the function of Example 3.7

Table 3.1
Truth Table of Function F

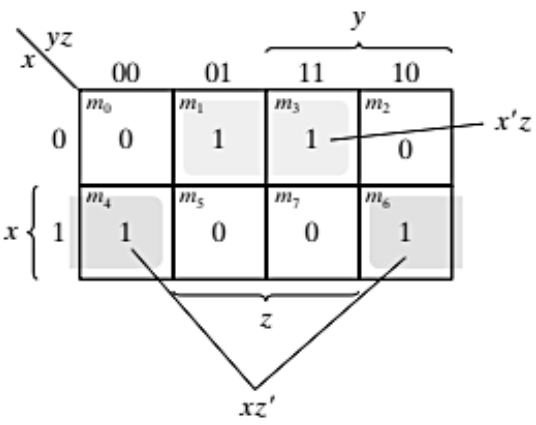
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

The gate-level implementation of the simplified expressions are obtained. The sum-of-products expression is implemented in (a) with a group of AND gates, one for each AND term. The outputs of the AND gates are connected to the inputs of a single OR gate. The same function is implemented in (b) in its product-of-sums form with a group of OR gates, one for each OR term. The outputs of the OR gates are connected to the inputs of a single AND gate.

In each case, it is assumed that the input variables are directly available in their complement, so inverters are not needed. The configuration pattern established in Fig is the general form by which any Boolean function is implemented when expressed in one of the standard forms. AND gates are connected to a single OR gate when in sum-of-products form; OR gates are connected to a single AND gate when in product-of-sums form.

Either configuration forms two levels of gates. Thus, the implementation of a function in a standard form is said to be a two-level implementation. The two-level implementation may not be practical, depending on the number of inputs to the gates. The procedure is also valid when the function is originally expressed in the product-of-maxterms canonical form. Consider, for example, the truth table that defines the function F in Table.

In sum-of-minterms form, this function is expressed as **F (x, y, z) = (1, 3, 4, 6)**



K – map for the above truth table

In product-of-maxterms form, it is expressed as **F (x, y, z) = (0, 2, 5, 7)**

In other words, the 1's of the function represent the minterms and the 0's represent the maxterms. The map for this function is shown in fig .

One can start simplifying the function by first marking the 1's for each minterm that the function is a 1. The remaining squares are marked by 0's. If, instead, the product of maxterms is initially given, one can start marking 0's in those squares listed in the function; the remaining squares are then marked by 1's.

Once the 1's and 0's are marked, the function can be simplified in either one of the standard forms.

For the **sum of products**, we combine the **1's** to obtain

F = x'z + xz'

For the **product of sums**, we combine the **0's** to obtain the simplified complemented function

F = xz + x'z'

which shows that the exclusive-OR function is the complement of the equivalence function.

Taking the complement of F, we obtain the simplified function in product-of-sums form:

F = (x' + z')(x + z)

To enter a function expressed in product-of-sums form into the map, use the complement of the function to find the squares that are to be marked by 0's.

For example, the function

F = (A' + B' + C')(B + D)

can be entered into the map by first taking its complement, namely,

F = ABC + B'D'

and then marking 0's in the squares representing the minterms of F. The remaining squares are marked with 1's.

8. Reduce the following logic function using Quine – McClusky method. f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15).(Dec. 2017)

Procedure of Quine-McCluskey Tabular Method

Follow these steps for simplifying Boolean functions using Quine-McClukey tabular method.

Step 1 − Arrange the given min terms in an **ascending order** and make the groups based on the number of ones present in their binary representations. So, there will be **at most 'n+1' groups** if there are 'n' Boolean variables in a Boolean function or 'n' bits in the binary equivalent of min terms.

Step 2 − Compare the min terms present in **successive groups**. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol '_' in the differed bit position and keep the remaining bits as it is.

Step 3 − Repeat step2 with newly formed terms till we get all **prime implicants**.

Step 4 − Formulate the **prime implicant table**. It consists of set of rows and columns. Prime implicants can be placed in row wise and min terms can be placed in column wise. Place '1' in the cells corresponding to the min terms that are covered in each prime implicant.

Step 5 − Find the essential prime implicants by observing each column. If the min term is covered only by one prime implicant, then it is **essential prime implicant**. Those essential prime implicants will be part of the simplified Boolean function.

Step 6 − Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for Reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15) using Quine-McClukey tabular method.

The given Boolean function is in **sum of min terms** form. It is having 4 variables W, X, Y & Z. The given min terms are 2, 6, 8, 9, 10, 11, 14 and 15. The ascending order of these min terms based on the number of ones present in their binary equivalent is 2, 8, 6, 9, 10, 11, 14 and 15. The following table shows these **min terms and their equivalent binary** representations.

Group Name	Min terms	W	X	Y	Z
GA1	2	0	0	1	0

	8	1	0	0	0
GA2	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
GA3	11	1	0	1	1
	14	1	1	1	0
GA4	15	1	1	1	1

The given min terms are arranged into 4 groups based on the number of ones present in their binary equivalents. The following table shows the possible **merging of min terms** from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6	0	-	1	0
	2,10	-	0	1	0
	8,9	1	0	0	-
	8,10	1	0	-	0
GB2	6,14	-	1	1	0
	9,11	1	0	-	1

	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-

The min terms, which are differed in only one-bit position from adjacent groups are merged. That differed bit is represented with this symbol, '-'. In this case, there are three groups and each group contains combinations of two min terms. The following table shows the possible **merging of min term pairs** from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

The successive groups of min term pairs, which are differed in only one-bit position are merged. That differed bit is represented with this symbol, '-'. In this case, there are two groups and each group contains combinations of four min terms. Here, these combinations of 4 min terms are available in two rows. So, we can remove the repeated rows. The reduced table after removing the redundant rows is shown below.

Group Name	Min terms	W	X	Y	Z
GC1	2,6,10,14	-	-	1	0
	8,9,10,11	1	0	-	-
GC2	10,11,14,15	1	-	1	-

Further merging of the combinations of min terms from adjacent groups is not possible, since they are differed in more than one-bit position. There are three rows in the above table. So, each row will give one prime implicant. Therefore, the **prime implicants** are YZ', WX' & WY.

The **prime implicant table** is shown below.

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ'	1	1			1		1	
WX'			1	1	1	1		
WY					1	1	1	1

The prime implicants are placed in row wise and min terms are placed in column wise. 1s are placed in the common cells of prime implicant rows and the corresponding min term columns.

The min terms 2 and 6 are covered only by one prime implicant **YZ'**. So, it is an **essential prime implicant**. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	8	9	11	15

WX'	1	1	1	
WY			1	1

The min terms 8 and 9 are covered only by one prime implicant **WX'**. So, it is an **essential prime implicant**. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	**15**
WY	1

The min term 15 is covered only by one prime implicant **WY**. So, it is an **essential prime implicant**. This will be part of simplified Boolean function.

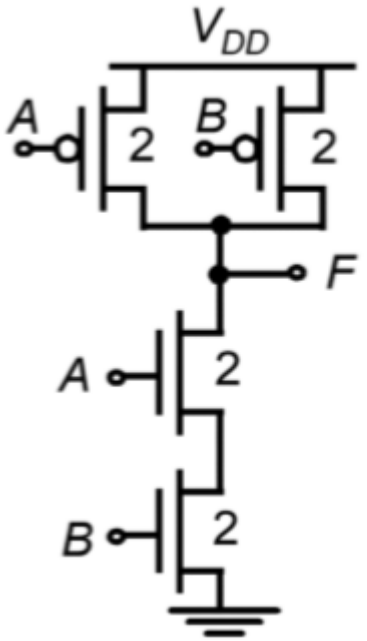
In this example problem, we got three prime implicants and all the three are essential. Therefore, the **simplified Boolean function** is

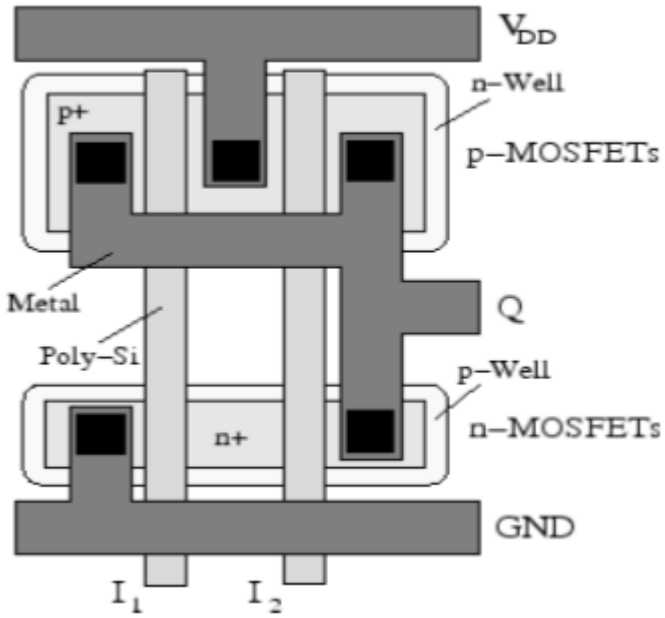
$$\mathbf{F}(W,X,Y,Z) = \mathbf{YZ' + WX' + WY.}$$

9. (a) Explain with an aid of circuit diagram the operation of 2 input CMOS NAND gate and list out its advantages over other logic families. (May 2018, Dec.2019)

CMOS Two-input NAND Gate

The circuit diagram of the two input CMOS NAND gate is given in the figure below.





The principle of operation of the circuit is exact dual of the CMOS two input NOR operation. The n – net consisting of two series connected nMOS transistor creates a conducting path between the output node and the ground, if both input voltages are logic high. Both of the parallelly connected pMOS transistor in p-net will be off.

For all other input combination, either one or both of the pMOS transistor will be turn ON, while p – net is cut off, thus, creating a current path between the output node and the power supply voltage. The switching threshold for this gate is obtained as

$$V_{th}\,(NAND2) = \frac{V_{T,n} + 2\sqrt{\frac{k_p}{k_n}(V_{DD} - |V_{T,p}|)}}{1 + 2\sqrt{\frac{k_p}{k_n}}}$$

The features of this layout are as follows –

- Single polysilicon lines for inputs run vertically across both N and P active regions.
- Single active shapes are used for building both nMOS devices and both pMOS devices.
- Power bussing is running horizontal across top and bottom of layout.
- Output wires runs horizontal for easy connection to neighboring circuit.

Advantages

- The static power dissipation is more than double in case of NOR gate to that of NAND. This also depicts a slight increase in the dynamic power dissipation in case of NOR gate.
- The comparison of rise time, fall time and delay show better results with NAND gate.
- It is clearly shown that the delay is quite low for NAND gate.

- The area requirement is also lesser for NAND out of the other gates.

9. (b) Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction. (May 2018)

(a) X -Y and

(b) Y - X using 2's complements.

a) X = 1010100

2's complement of Y = + 0111101

Sum = 10010001

Discard end carry

Answer: X - Y = 0010001

b) Y = 1000011

2's complement of X = + 0101100

Sum = 1101111

There is no end carry,

Therefore the answer is Y-X = - (2's complement of 1101111) = -0010001

10. (a) Draw the CMOS logic circuit for NOR gate and explain its operation. (May 2017, May 2018, Dec. 2019, Sept. 2020)

CMOS Logic Circuits

CMOS Two input NOR Gate

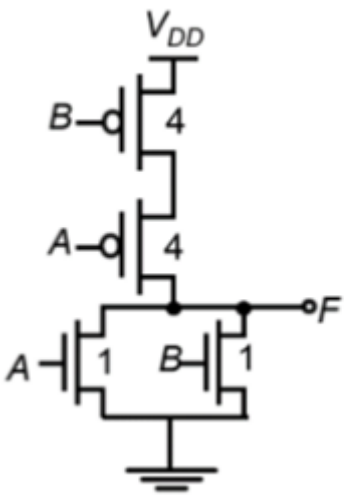
The circuit consists of a parallel-connected n-net and a series-connected complementary p-net. The input voltages V_X and V_Y are applied to the gates of one nMOS and one pMOS transistor.

When either one or both inputs are high, i.e., when the n-net creates a conducting path between the output node and the ground, the p-net is cut—off. If both input voltages are low, i.e., the n-net is cut-off, then the p-net creates a conducting path between the output node and the supply voltage.

For any given input combination, the complementary circuit structure is such that the output is connected either to V_{DD} or to ground via a low-resistance path and a DC current

path between the V_{DD} and ground is not established for any input combinations. The output voltage of the CMOS, two input NOR gate will get a logic-low voltage of $V_{OL} = 0$ and a logic-high voltage of $V_{OH} = V_{DD}$. The equation of the switching threshold voltage V_{th} is given by

$$V_{th}\,(NOR2) = \frac{V_{T,n} + \frac{1}{2}\sqrt{\frac{k_p}{k_n}(V_{DD} - |V_{T,p}|)}}{1 + \frac{1}{2}\sqrt{\frac{k_p}{k_n}}}$$

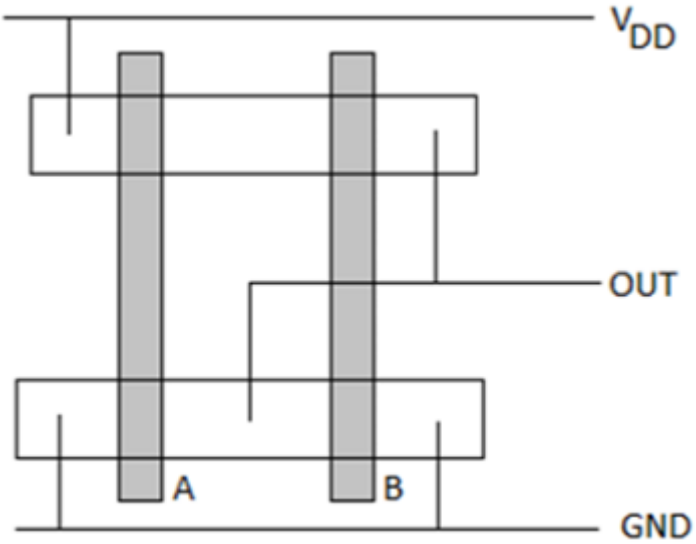


Layout of CMOS 2-input NOR Gate

The figure shows a sample layout of CMOS 2-input NOR gate, using single-layer metal and single-layer polysilicon. The features of this layout are −

- Single vertical polylines for each input
- Single active shapes for N and P devices, respectively
- Metal buses running horizontal

The stick diagram for the CMOS N0R2 gate is shown in the figure given below; which corresponds directly to the layout, but does not contain W and L information. The diffusion areas are depicted by rectangles, the metal connections and solid lines and circles, respectively represent contacts, and the crosshatched strips represent the polysilicon columns. Stick diagram is useful for planning optimum layout topology.



10. (b) Briefly discuss weighted binary code. (May 2018)

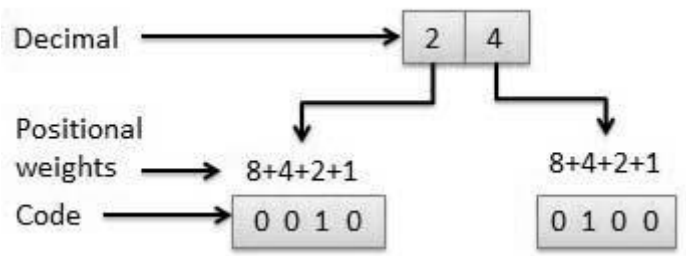
Classification of binary codes

The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes

Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.



1. The main characteristic of a weighted code is, each binary bit is assigned by a "weight" and values depend on the position of the binary bit.

2. The sum of the weights of these binary bits, whose value is 1 is equal to the decimal digit which they represent.
3. In other words, if w1, w2, w3 and w4 are the weights of the binary digits, and x1, x2, x3 and x4 are the corresponding bit values, then the decimal digit N=w4x4 + w3x3+w2x2+w1x1 is represented by the binary sequence x4x3x2x1.
4. A sequence of binary bits which represents a decimal digit is called a "code word".
5. Thus x4x3x2x1 is a code word of N.
6. Example of these codes is: BCD, 8421, 6421, 4221, 5211, 3321 etc.
7. Weighted codes are used in:

   a) Data manipulation during arithmetic operation.

   b) For input/output operations in digital circuits.

   c) To represent the decimal digits in calculators, volt meters etc.

In the example **4782 = number 3210 = positional values**, the positional assignments 0 through 3 can be the weighted values of their assigned digits. So the weight of the 4 is 3 and the weight of the 7 is 2. The weight of a number comes into play when converting from any base numbering system to the decimal (base 10) numbering system.

One formula for converting a weighted number is to multiply each digit by its base to the power of its position, and then add all the resulting digits. In the example below, 100101, which is a binary base 2 number, is converted to a decimal (base 10) number.

100101 = Binary (base 2) number 543210 = positional weights (1 x 2^5) + (0 x 2^4) + (0 x 2^3) + (1 x 2^2) + (0 x 2^1) + (1 x 2^0) = 32 + 0 + 0 + 4 + 0 + 1 = 37 37base10 = decimal conversion

Other weighted methods include BCD and 2421, each of which uses a similar formula to assign weights and convert to decimal.

11. Use Karnaugh maps to find the minimum-cost SOP and POS expressions for the function

f(x1,...,x4) = x1x3x4 + x3x4 + x1x2x4 + x1x2x3x4 assuming that there are also don't-cares defined as D = (9, 12, 14). (May 2019)

Solution:

The Karnaugh map is derived by placing 1s that correspond to each product term in the expression used to specify f. The term x1x3x4 corresponds to minterms 0 and 4. The term x3x4 represents the third row in the map, comprising minterms 3, 7, 11, and 15.

The term x1x2x4 specifies minterms 1 and 3. The fourth product term represents the minterm 13. The map also includes the three don't-care conditions. To find the desired SOP expression, we must find the least-expensive set of prime implicants that covers all 1s in the map.

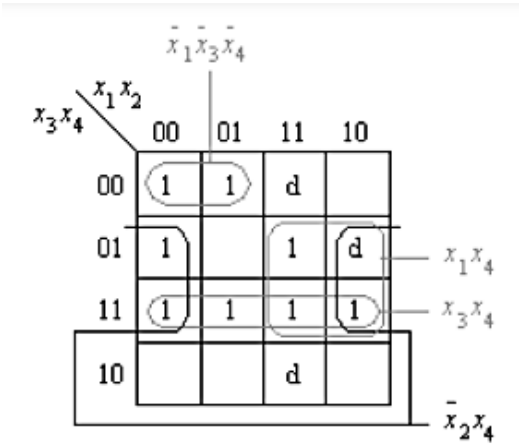
The term x3x4 is a prime implicant which must be included because it is the only prime implicant that covers the minterm 7; it also covers minterms 3, 11, and 15. Minterm 4 can be covered with either x1x3x4 or x2x3x4.

Both of these terms have the same cost; we will choose x1x3x4 because it also covers the minterm 0. Minterm 1 may be covered with either x1x2x3 or x2x4; we should choose the latter because its cost is lower. This leaves only the minterm 13 to be covered, which can be done with either x1x4 or x1x2 at equal costs. Choosing x1x4, **the minimum-cost SOP expression is**

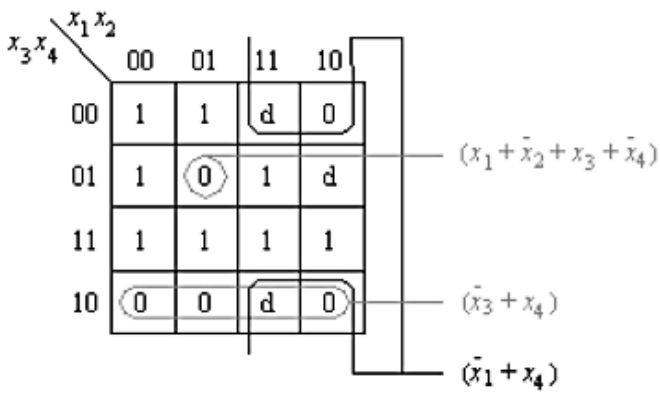
f = x3x4 + x1x3x4 + x2x4 + x1x4

The sum term (x3 + x4) covers the 0s in the bottom row. To cover the 0 in square 8 we must include (x1 + x4). The remaining 0, in square 5, must be covered with (x1 + x2 + x3 + x4). Thus, **the minimum-cost POS expression is**

f = (x3 + x4)(x1 + x4)(x1 + x2 + x3 + x4)



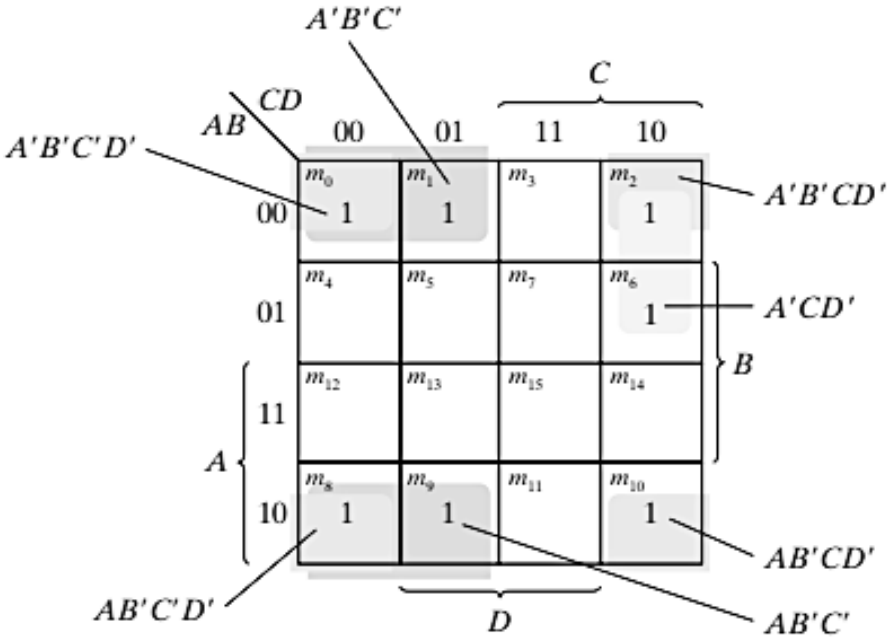
SOP expression



POSexpression

12. Reduce the following function using K map F = A'B'C' + B'CD' + A'BCD' + AB'C'. (Dec. 2019)

F = A'B'C' + B'CD' + A'BCD' + AB'C'



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

From k - map figure: A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'

The area in the map covered by this function consists of the squares marked with 1's. The function has four variables and, as expressed, consists of three terms with three literals each and one term with four literals. Each term with three literals is represented in the map by two squares. For example, A'B'C' is represented in squares 0000 and 0001. The function can be simplified in the map by taking the 1's in the four corners to give the term B'D'. This is possible because these four squares are adjacent when the map is drawn in a surface with top and bottom edges, as well as left and right edges, touching one another. The two left-hand 1's in the top row are combined with the two 1's in the bottom row to give the term B'C'. The remaining 1 may be combined in a two square area to give the term A'CD'.

The simplified function is **F = B'D' + B'C' + A'CD'**

BM T43–DIGITAL LOGIC THEORY AND DESIGN

UNIT 2

PART – A

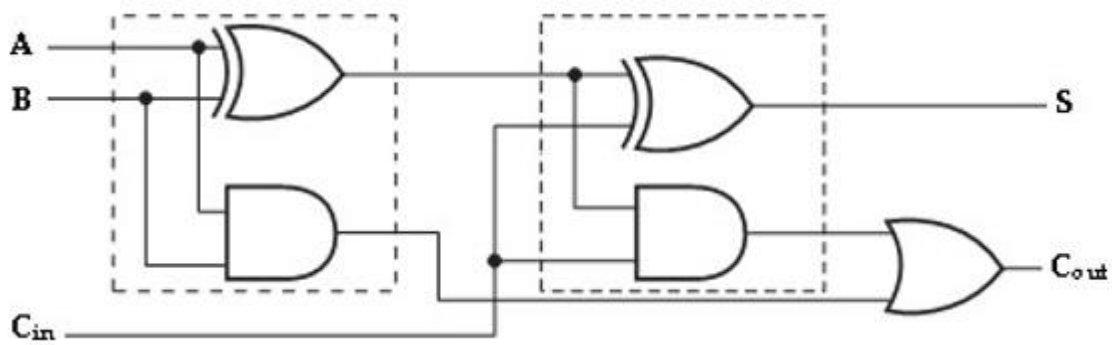
1. Mention the types of adders. (May 2016)

- Adder
- Half adder
- Full adder
- Ripple-carry adder
- Carry-look ahead adder
- Brent-Kung adder
- Kogge-Stone adder
- Carry-save adder

2. Which gate is suitable for building a comparator and why? (May 2016)

An EX-NOR gate is suitable for building a comparator because, an ex-nor outputs high for an even number of 1's or all 0's and outputs low for otherwise. Thus, an ex-nor gate is a basic comparator, because its output is "1" only if its two input bits are equal.

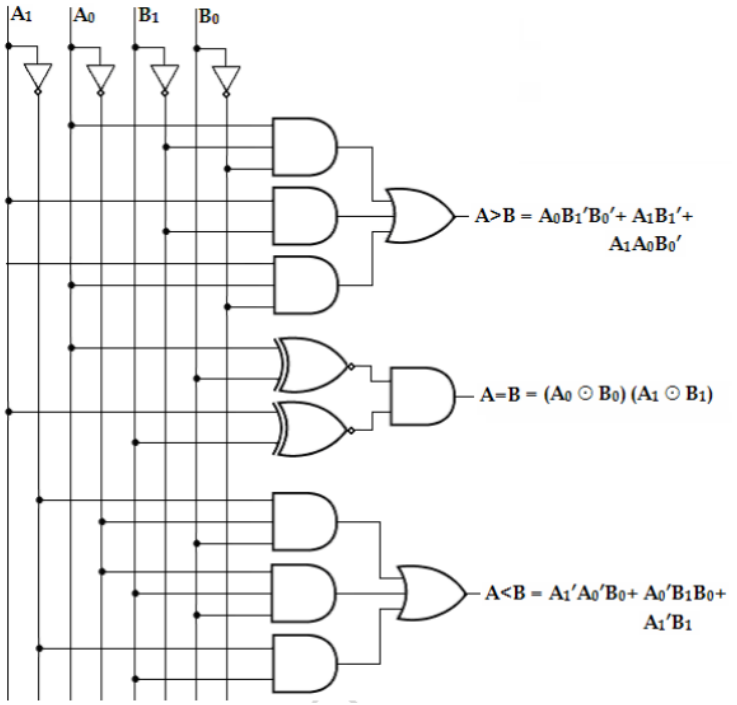
3. Draw the structure of a full adder using two half adders (Nov. 2016)



4. Mention the difference between demultiplexer and multiplexer. (Nov. 2016)

The major factor that differentiates multiplexer and demultiplexer is their ability to accept multiple input and single input respectively. The multiplexer also known as **MUX** operates on several inputs but provide a single output. As against demultiplexer also known as **DEMUX** simply reverses the operation of MUX and operates on single input but transmits the data on multiple outputs.

5. Design a single-bit magnitude comparator. (Nov. 2017)



6. Write the expression for sum and carry for half adder (May 2017, Sep. 2020)

For Carry

A \ B	0	1
0	0	0
1	0	1

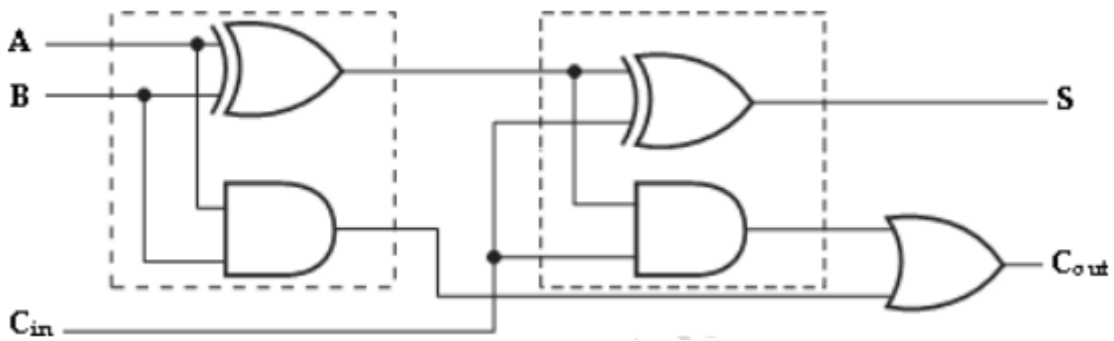
Carry = A . B

For Sum

A \ B	0	1
0	0	1
1	1	0

Sum = AB'+ A'B
= A⊕B

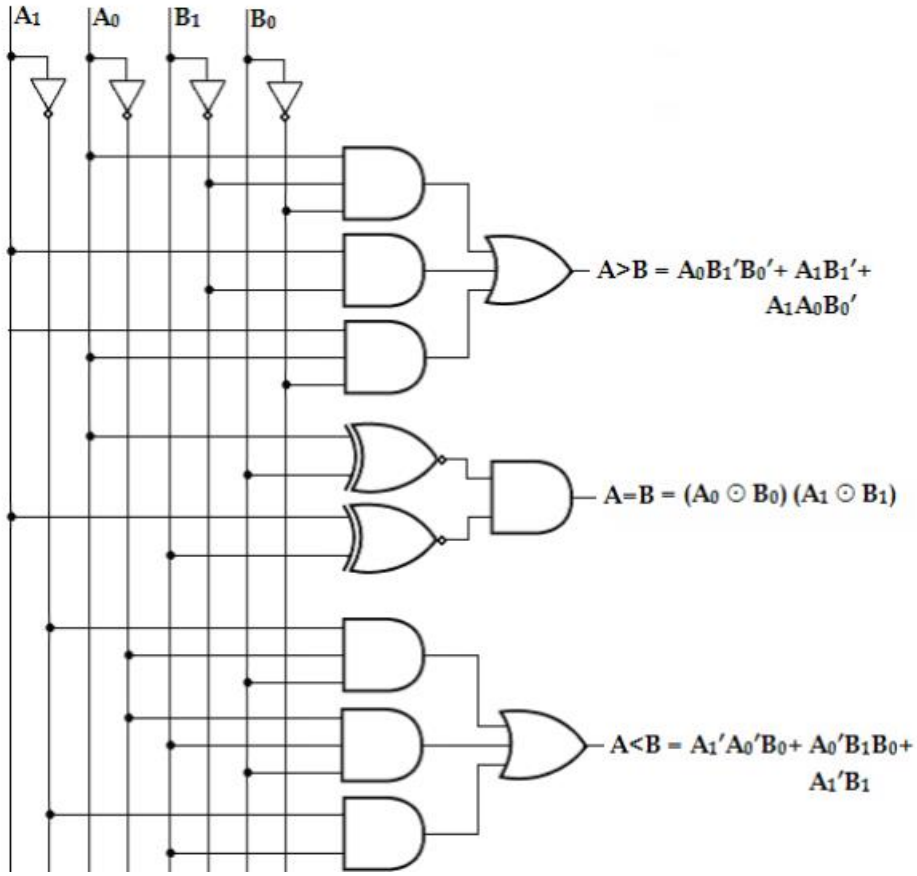
7. Implement a full adder using two half adders. (Nov. 2017)



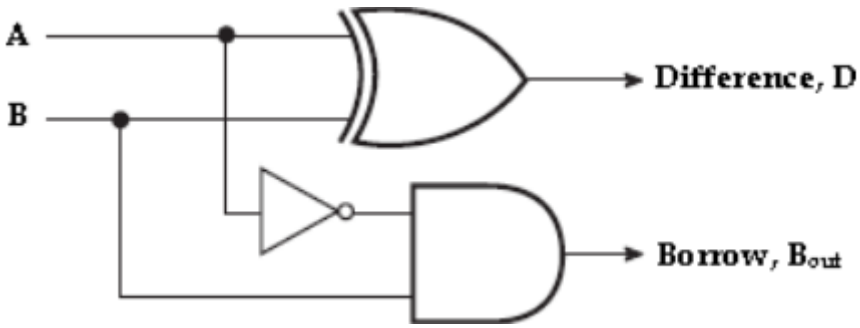
8. How does an encoder differ from a decoder (Nov. 2018, Nov. 2019)

Encoder circuit basically converts the applied information signal into a coded digital bit stream. Decoder performs reverse operation and recovers the original information signal from the coded bits. In case f encoder, the applied signal is the active signal input. Decoder accepts coded binary data as its input.

9. Draw 2-bit comparator using logic gates. (Nov. 2018)

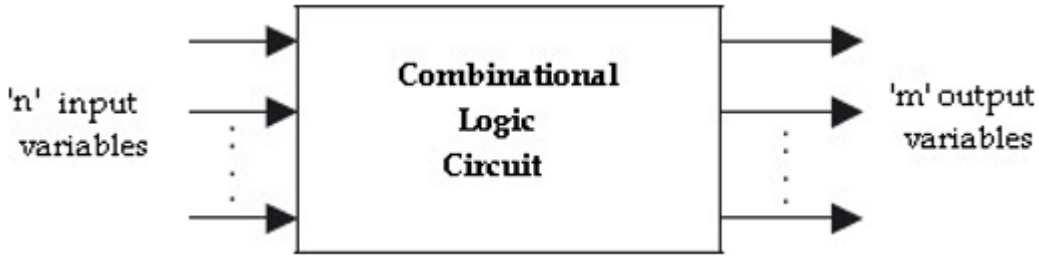


10. Design half subtractor using basic gates. (May 2019)



11. What do you mean by combinational circuits? Give example (Nov. 2019)

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from inputs and output signals are generated according to the logic circuits employed in it. Binary information from the given data transforms to desired output data in this process. Both input and output are obviously the binary signals, i.e., both the input and output signals are of two possible states, logic 1 and logic 0.



For n number of input variables to a combinational circuit, 2n possible combinations of binary input states are possible. For each possible combination, there is one and only one possible output combination. A combinational logic circuit can be described by m Boolean functions and each output can be expressed in terms of n input variables.

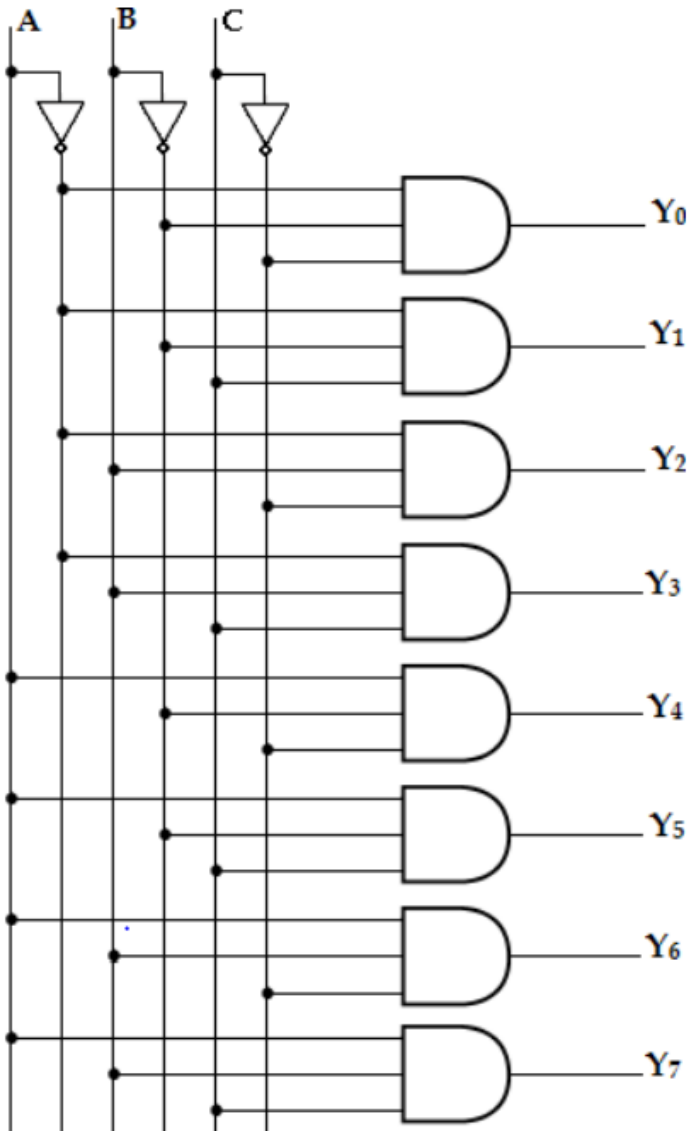
12. Convert the given expression in canonical SOP form Y = AC+AB+BC (Nov. 2018)

$$
\begin{aligned}
Y(A, B, C) &= AC + AB + BC \\
&= AC(B + B') + AB(C + C') + BC(A + A') \\
&= \underline{ABC} + AB'C + \underline{ABC} + ABC' + \underline{ABC} + A'BC \\
&= ABC + AB'C + ABC' + A'BC \\
&= \sum m(3, 5, 6, 7).
\end{aligned}
$$

PART – B

1. Design a 3 to 8 decoder. (May 2016, Sep. 2020)

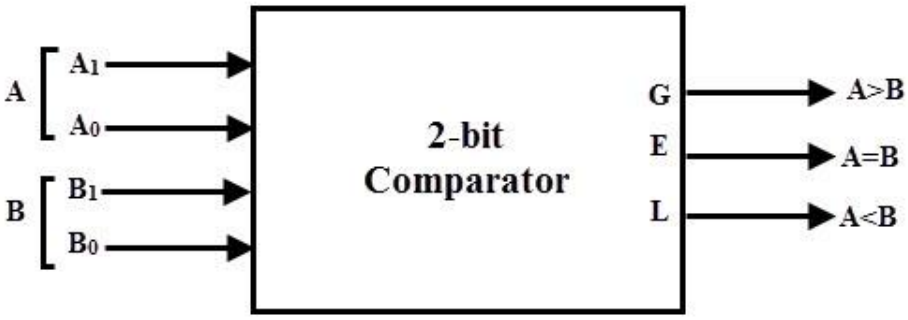
A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y0- Y7). Based on the 3 inputs one of the eight outputs is selected. The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion. The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterms equivalent of the binary number presently available in the input lines.



Inputs			Outputs							
A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

2. Design a 2-bit magnitude comparator (May 2016)

Magnitude Comparator: A magnitude comparator is a combinational circuit that compares two given numbers (A and B) and determines whether one is equal to, less than or greater than the other. The output is in the form of three binary variables representing the conditions A = B, A>B and A<B, if A and B are the two numbers being compared.



For comparison of two n-bit numbers, the classical method to achieve the Boolean expressions requires a truth table of 2^{2n} entries and becomes too lengthy and cumbersome.

2- Bit Magnitude Comparator:

The truth table of 2-bit comparator is given in table below:

Control (pump) signal				Output signals at different ports		
A_1	A_0	B_1	B_0	Port Y1 (A=B)	Port Y2 (A>B)	Port Y3 (A<B)
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

K-MAP SIMPLIFICATION

For A>B

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$$

For A=B

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

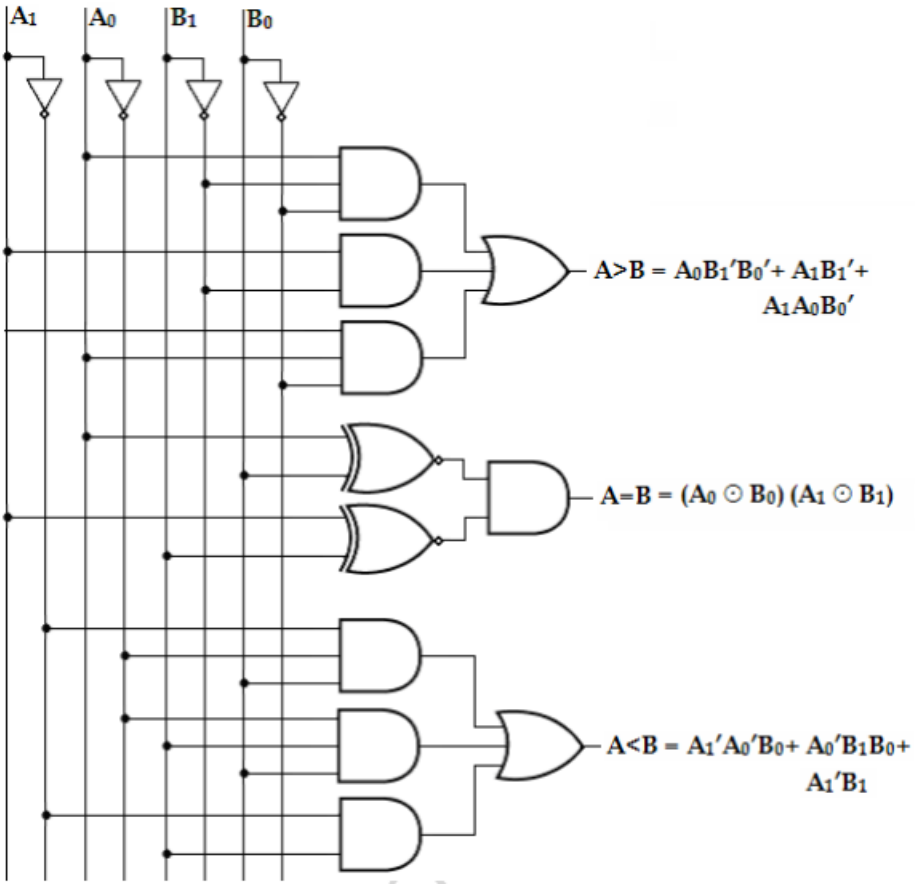
$$\begin{aligned} A=B &= A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + \\ &\quad A_1A_0B_1B_0 + A_1A_0'B_1B_0' \\ &= A_1'B_1'(A_0'B_0' + A_0B_0) + A_1B_1(A_0B_0 + A_0'B_0') \\ &= (A_0 \odot B_0)(A_1 \odot B_1) \end{aligned}$$

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

For A<B

$$Y_1 = \overline{A_1}\,\overline{A_0}\,B_0 + \overline{A_1}\,B_1 + \overline{A_0}\,B_1\,B_0$$

LOGIC DIAGRAM:



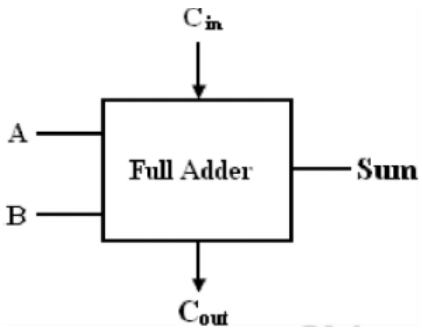
EXAMPLE INPUTS

Inputs		Outputs		
A	B	A<B	A=B	A>B
00001100	00001100	0	1	0
00001010	00010001	1	0	0
00001111	00000101	0	0	1
00011000	00011000	0	1	0

3. Design a full adder and a half subtractor (Nov. 2016)

Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of 3 inputs and 2 outputs. Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position. The block diagram of full adder is given by,



The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. As there are three input variables, eight different input combinations are possible.

Truth Table:

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

To derive the simplified Boolean expression from the truth table, the Karnaugh map method is adopted as,

For Carry

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Carry, C_{out} = AB+ AC_{in} + BC_{in}

For Sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

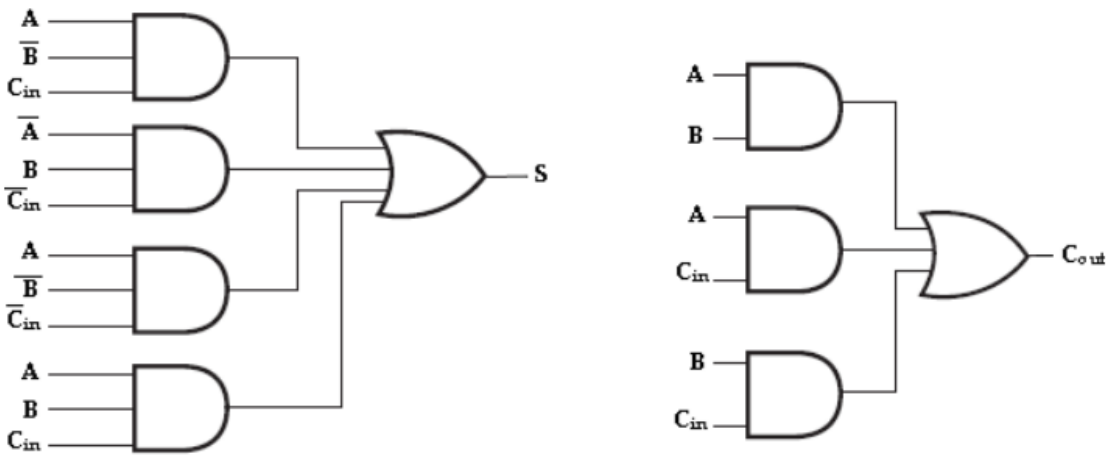
Sum, S = A′B′C_{in}+ A′BC′$_{in}$+ AB′C′$_{in}$+ ABC_{in}

The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A’B’C_{in}+ A’BC’$_{in}$ + AB’C’$_{in}$ +ABC_{in}

Carry, C_{out} = AB+ AC_{in} + B$Ci_{n.}$

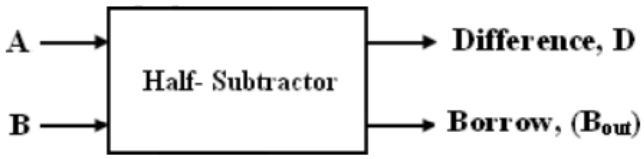
The logic diagram for the above functions is shown as,



Implementation of full-adder in Sum of Products

Half Subtractor:

A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a _1' has be en borrowed to perform the subtraction.



Block schematic of half-subtractor

The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown below.

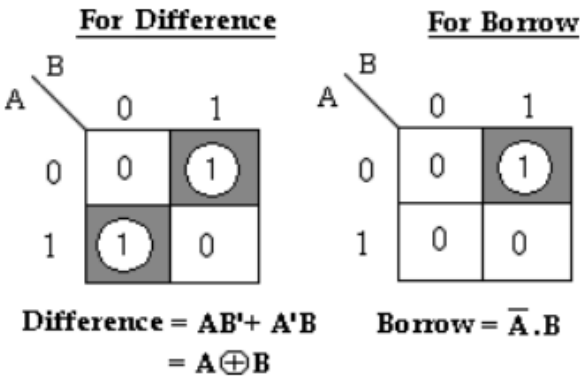
Input		Output	
A	B	Difference (D)	Borrow (B_{out})
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map Simplification:

The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

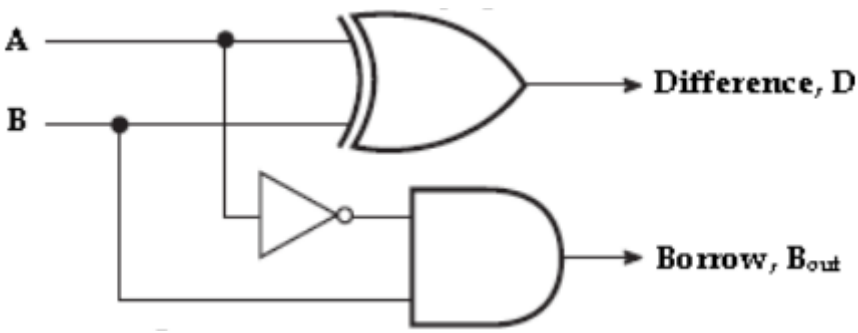
Difference, D = A'B+ AB'= A ⊕ B

Borrow, Bout = A'. B



The first one representing the DIFFERENCE (D)output is that of an exclusive-OR gate, the expression for the BORROW output (Bout) is that of an AND gate with input A complemented before it is fed to the gate.

Logic Diagram:



Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, ie., the minuend is complemented, an AND gate can be used to implement the BORROW output.

4. Explain the working principle of following combinational circuits(May 2017)

a) BCD to binary converter

The steps involved in the BCD-to-binary conversion process are as follows:

1. The value of each bit in the BCD number is represented by a binary equivalent or weight.

2. All the binary weights of the bits that are 1‘s in the BCD are added.

3. The result of this addition is the binary equivalent of the BCD number.

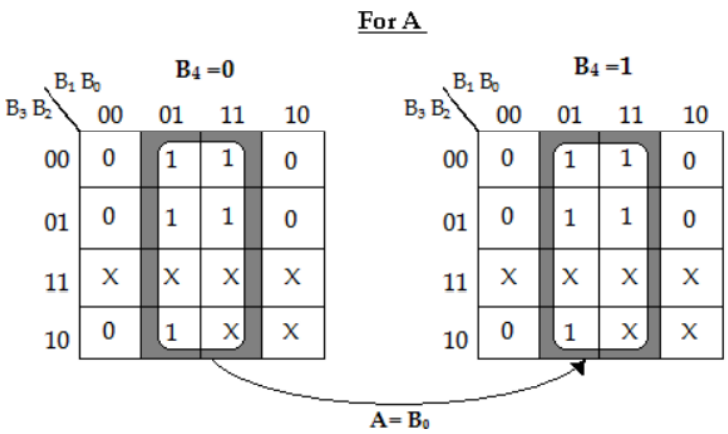
Two-digit decimal values ranging from 00 to 99 can be represented in BCD by two 4-bit code groups. For example, 1910 is represented as,

$$\underbrace{1}_{0001}\quad\underbrace{9}_{1001}$$

The left-most four-bit group represents 10 and right-most four-bit group represents 9. The binary representation for decimal 19 is $19_{10} = 1100_{12}$.

BCD Code					Binary				
B_4	B_3	B_2	B_1	B_0	E	D	C	B	A
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	0	1	0	1	0	1	1

K-map simplification:



For B

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	X	X	X	X
10	0	0	X	X

$B_4 = 1$

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	X	X	X	X
10	1	1	X	X

$$B = B_1B_4' + B_1'B_4$$
$$= B_1 \oplus B_4$$

For C

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	0	0	X	X

$B_4 = 1$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$$C = B_4'B_2 + B_2B_1' + B_4B_2'B_1$$

For D

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

$B_4 = 1$

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$$D = B_4'B_3 + B_4B_3'B_2' + B_4B_3'B_1'$$

For E

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	0	0	X	X

$B_4 = 1$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	X	X	X	X
10	1	1	X	X

$E = B_4B_3 + B_4B_2B_1$

From the above K-map,

$A = B_0$

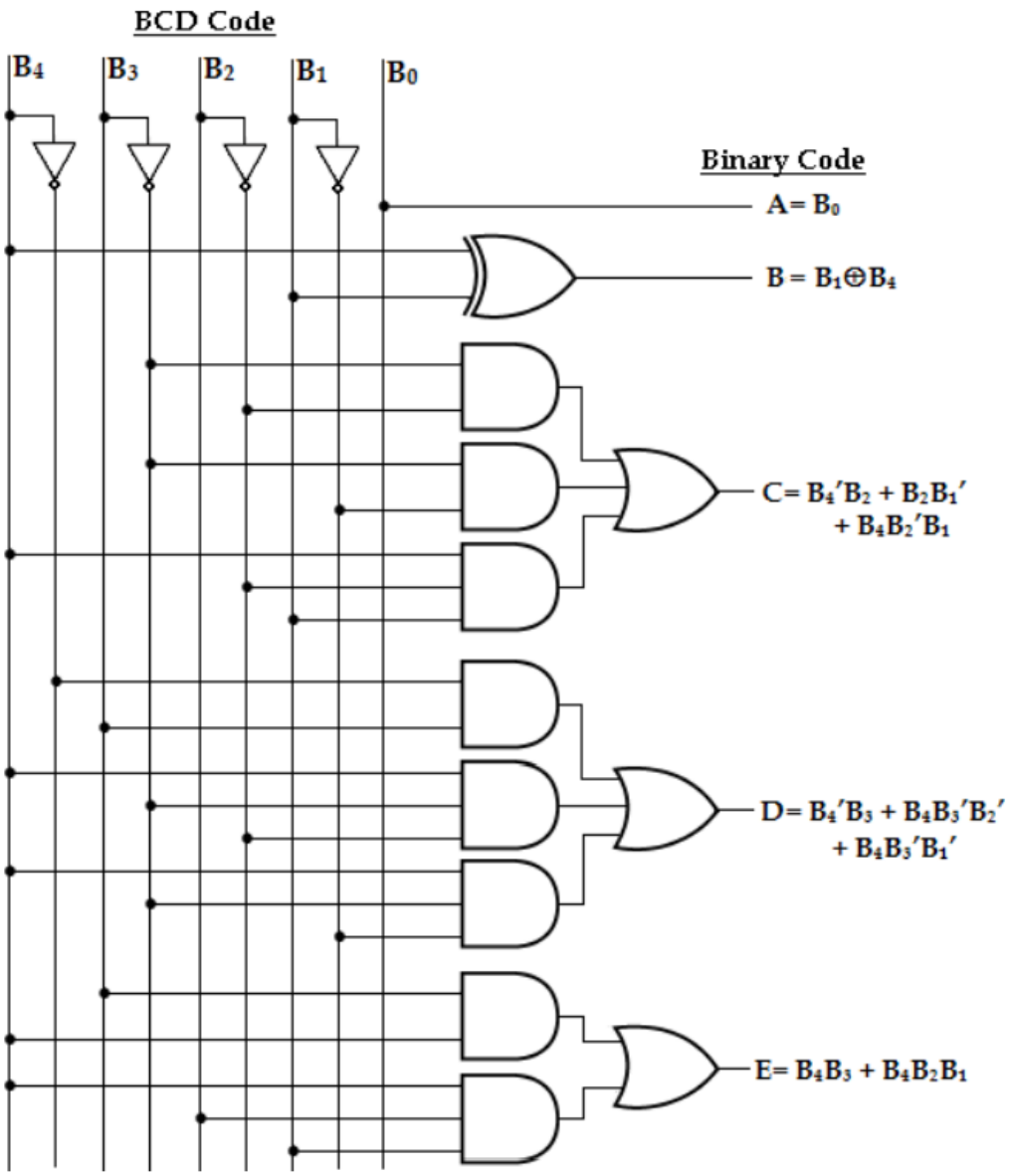
$B = B_1B_4' + B_1'B_4$
$= B_1 \oplus B_4$

$C = B_4'B_2 + B_2B_1' + B_4B_2'B_1$

$D = B_4'B_3 + B_4B_3'B_2' + B_4B_3'B_1'$

$E = B_4B_3 + B_4B_2B_1$

Logic Diagram:



5.With necessary diagram explain the working principle of the following comninational circuits

a) Binary multiplier using shift method(May 2017)

Multiplication of binary numbers is performed in the same way as in decimal numbers. The multiplicand is multiplied by each bit of the multiplier starting from the least significant bit. Each such multiplication forms a partial product. Such partial products are shifted one position to the left. The final product is obtained from the sum of partial products.

Consider the multiplication of two 2-bit numbers. The multiplicand bits are B1 and B0, the multiplier bits are A1 and A0, and the product is C3, C2, C1 and C0. The first partial product is formed by multiplying A0 by B1B0. The multiplication of two bits such as A0 and

B0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation. Therefore, the partial product can be implemented with AND gates as shown in the diagram below.

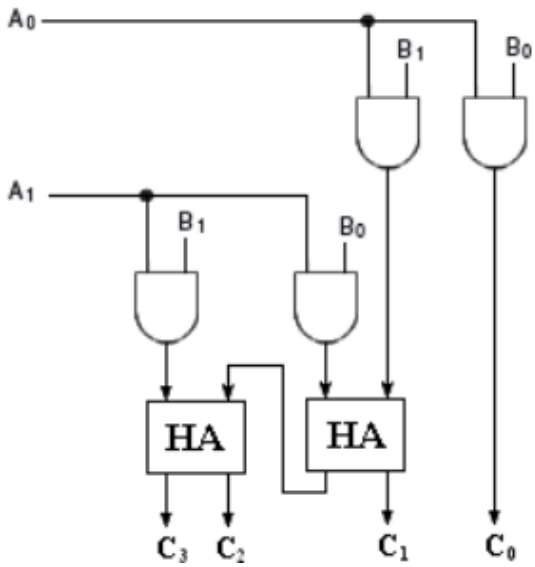
The second partial product is formed by multiplying A1 by B1B0 and shifted one position to the left. The two partial products are added with two half adder (HA) circuits.

Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products. The least significant bit of the product does not have to go through an adder since it is formed by the output of the first AND gate.

A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.

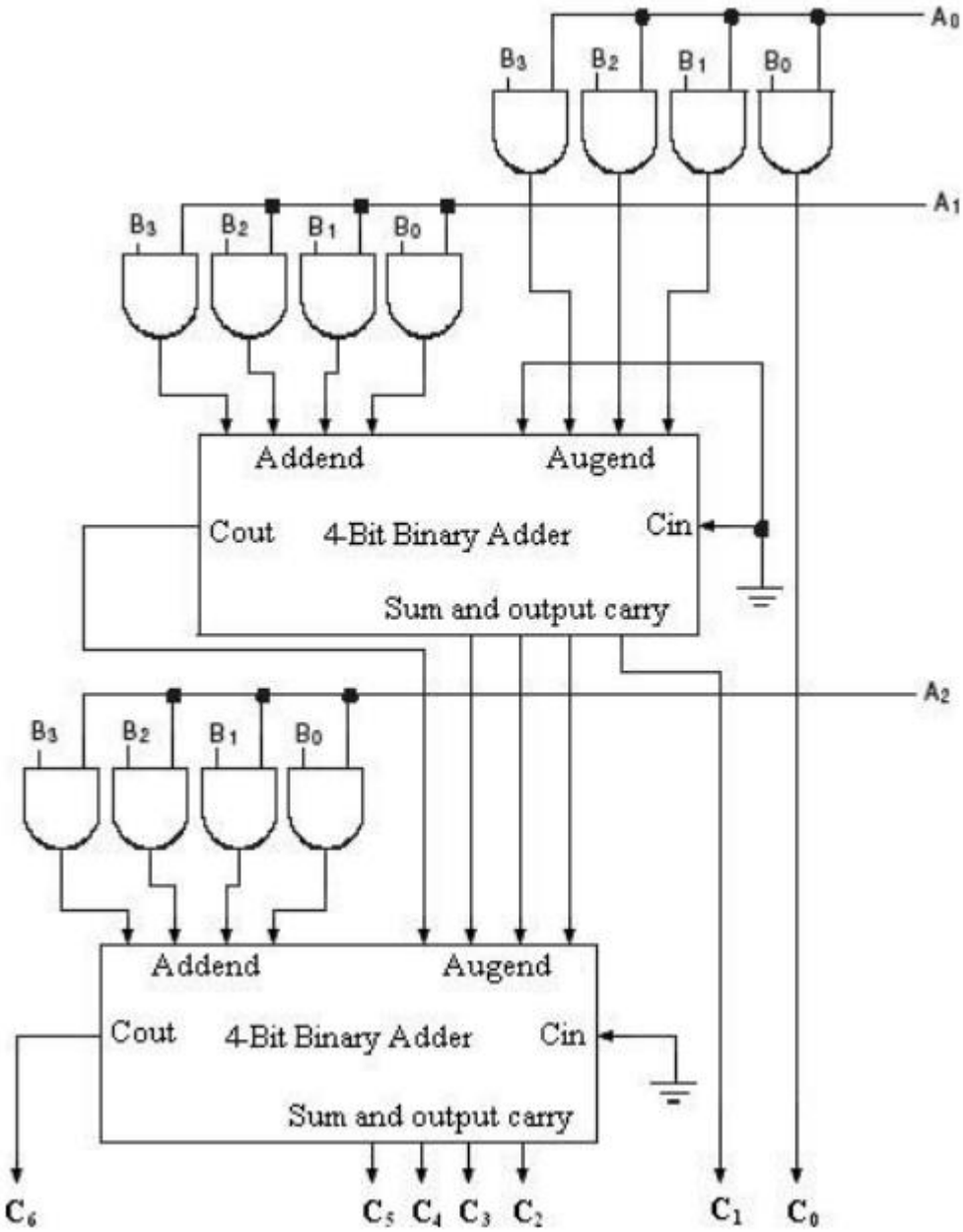
The binary output in each level of AND gates are added with the partial product of the

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



previous level to form a new partial product. The last level produces the product. For J multiplier bits and K multiplicand bits we need (J x K) AND gates and (J-1) k-bit adders to produce a product of J+K bits.

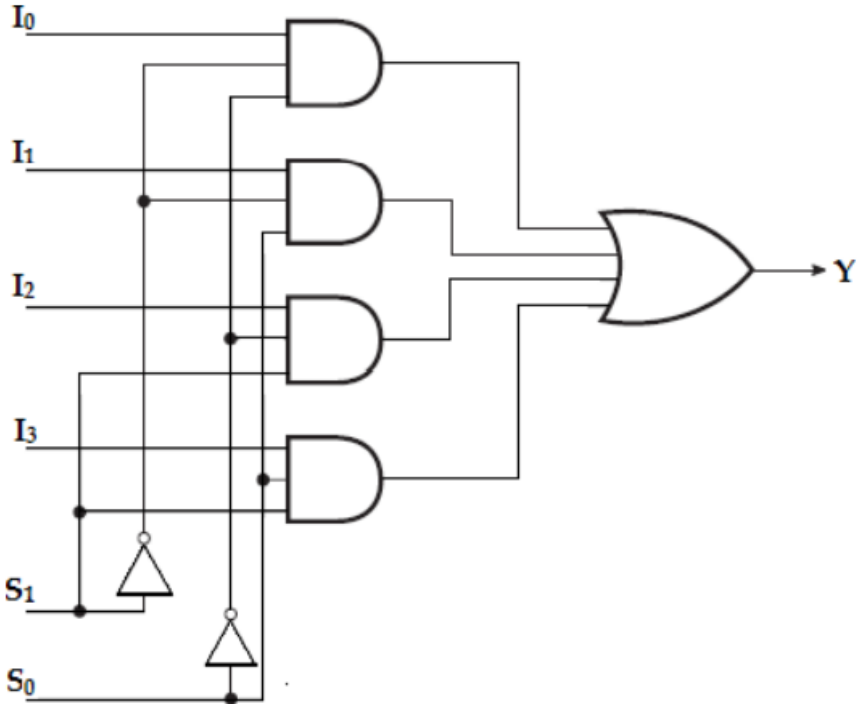
Consider a multiplier circuit that multiplies a binary number of four bits by a number of three bits. Let the multiplicand be represented by B3, B2, B1, B0 and the multiplier by A2, A1, and A0. Since K= 4 and J= 3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits. The logic diagram of the multiplier is shown below.



5. b) 4 to 1 multiplexer

A 4-to-1-line multiplexer has four (2n) input lines, two (n) select lines and one output line. It is the multiplexer consisting of four input channels and information of one of the channels can be selected and transmitted to an output line according to the select inputs combinations.

Selection of one of the four input channel is possible by two selection inputs. Each of the four inputs I0 through I3, is applied to one input of AND gate. Selection lines S1 and S0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.



Function table:

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

To demonstrate the circuit operation, consider the case when S1S0= 10. The AND gate associated with input I2 has two of its inputs equal to 1 and the third input connected to I2. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I2, providing a path from the selected input to the output.

The data output is equal to I0 only if S1= 0 and S0= 0; Y= I0S1‘S0‘.

The data output is equal to I1 only if S1= 0 and S0= 1; Y= I1S1‘S0.

The data output is equal to I2 only if S1= 1 and S0= 0; Y= I2S1S0‘.

The data output is equal to I3 only if S1= 1 and S0= 1; Y= I3S1S0.

When these terms are ORed, the total expression for the data output is,

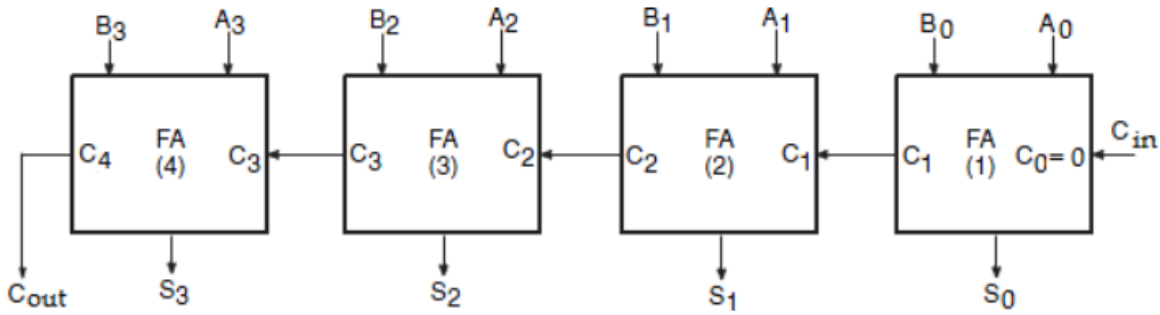
Y= I0S1’S0’+ I1S1’S0 +I2S1S0’+ I3S1S0.

As in decoder, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer.

6. Design the following circuit (Nov. 2017)

a) 4-bit parallel adder

The 4-bit binary adder using full adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in figure below.



4-bit binary parallel Adder

Since all the bits of augend and addend are fed into the adder circuits simultaneously and the additions in each position are taking place at the same time, this circuit is known as parallel adder.

Let the 4-bit words to be added be represented by,
A3A2A1A0= 1111 and B3B2B1B0= 0011.

Significant place		4	3	2	1
Input carry		1	1	1	0
Augend word A :		1	1	1	1
Addend word B :		0	0	1	1
	1	0	0	1	0 ← Sum
	↑ Output Carry				

The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry C0 in the least significant position must be 0. The carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple-carry adder.

In the least significant stage, A0, B0 and C0 (which is 0) are added resulting in sum S0 and carry C1. This carry C1 becomes the carry input to the second stage. Similarly, in the second stage, A1, B1 and C1 are added resulting in sum S1 and carry C2, in the third stage, A2, B2 and C2 are added resulting in sum S2 and carry C3, in the third stage, A3, B3 and C3

are added resulting in sum S3 and C4, which is the output carry. Thus, the circuit results in a sum (S3S2S1S0) and a carry output (Cout).

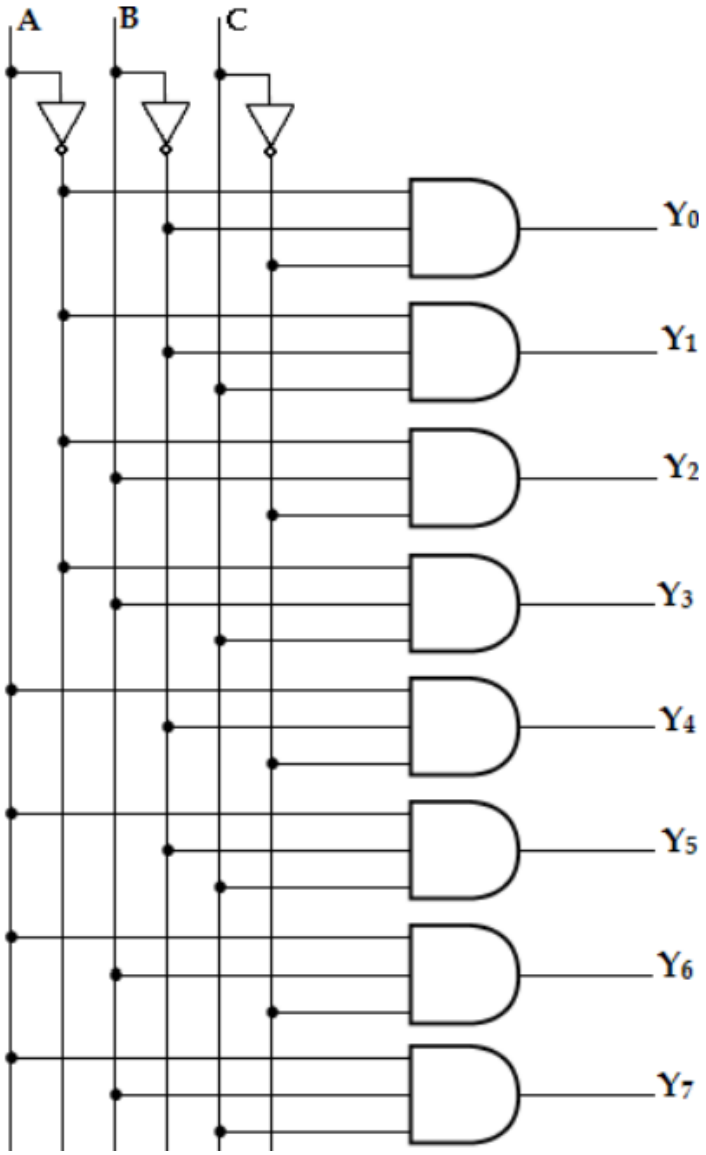
Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages. However, there are several methods to reduce this delay. One of the methods of speeding up this process is look-ahead carry addition which eliminates the ripple-carry delay.

6. b) binary to octal decoder (Nov. 2017)

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y0- Y7). Based on the 3 inputs one of the eight outputs is selected. The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion.

The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

Inputs			Outputs							
A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



7. Develop a code converter circuit that converts four-bit data into gray code (Nov. 2017)

The truth table for bcd to gray code converter is,

BCD Code (8421)				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0

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

K-map simplification:

For G_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

$G_3 = B_3$

For G_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$G_2 = B_3 + B_2$

For G_1

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

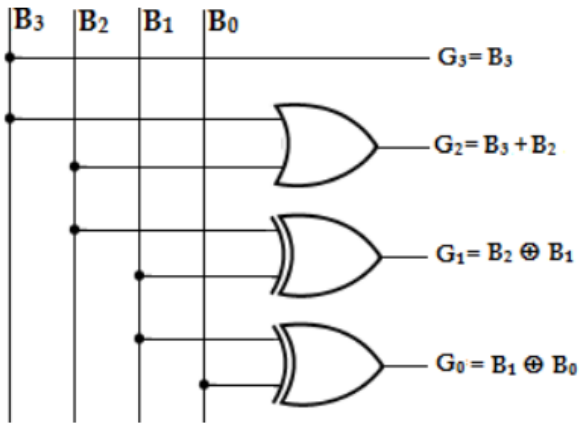
$G_1 = B_2'B_1 + B_2B_1'$
$= B_2 \oplus B_1$

For G_0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	X	X

$G_0 = B_1'B_0 + B_1B_0'$
$= B_1 \oplus B_0$

Logic diagram:



8. Draw the circuit of a BCD adder and explain it. (May 2018)

(or)

Design a 4-bit decimal adder using a 4-bit binary adder (Nov. 2019)

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD bits and produces a sum digit also in BCD.

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9+ 9+1 = 19; the 1 is the sum being an input carry. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

These binary numbers are labeled by symbols K, Z8, Z4, Z2, Z1, K is the carry. The columns under the binary sum list the binary values that appear in the outputs of the 4- bit binary adder. The output sum of the two decimal digits must be represented in BCD.

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9

0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 9 (1001), we obtain a nonvalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of the given truth table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

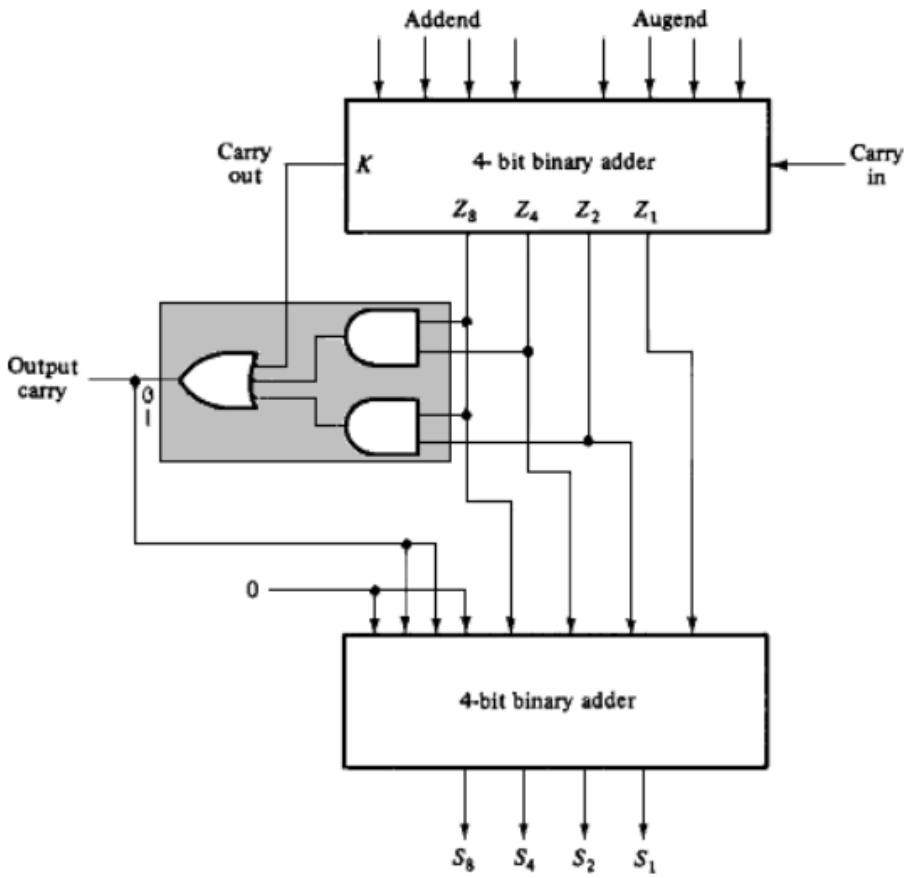
$S_3 S_2$ \ $S_1 S_0$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$Y = S_3S_2 + S_3S_1$

To implement BCD adder we require:

- 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add 01102 in the sum if the sum is greater than 9 or carry is 1.

The two decimal digits, together with the input carry, are first added in the top4- bit binary adder to provide the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary 0110 is added to the binary sum through the bottom 4-bit adder. The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. The output carry from one stage must be connected to the input carry of the next higher-order stage.



Block diagram of BCD adder

9. Design a BCD to Excess 3 code converter (May 2018)

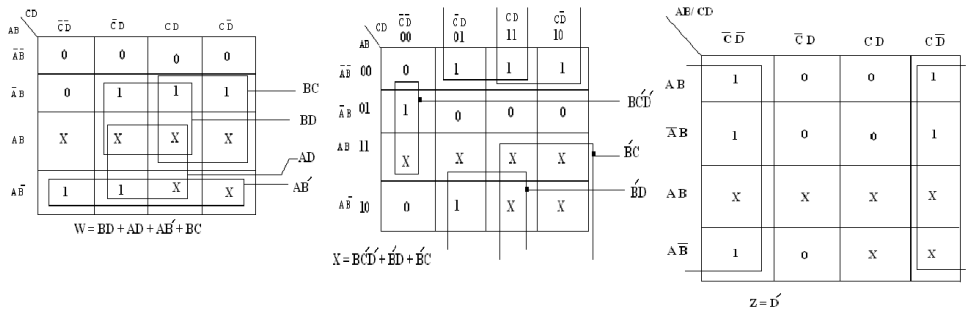
(or)

Illustrate BCD to Excess code converter using minimum number of NAND gates. (May 2019, Nov. 2019)

Excess-3 is a modified form of a BCD number. The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as,

BCD no	EXCESS-3 NO
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

After that by using K maps we can get simplified functions for w, x, y, z shown below as:



NAND gate implementation for simplified function

W = BD + AD + AB' + BC

through complementing twice we find

W = ((BD + AD + AB' + BC)')'

= ((BD)' . (AD)' . (AB')' . (BC)')' X = BC'D + B'D + B'C

Through complementing twice we find

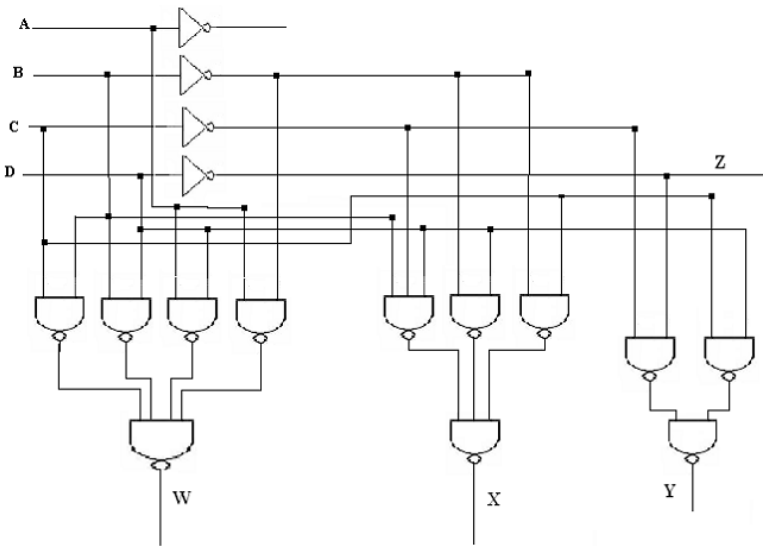
X = BC'D + B'D + B'C

= ((BC'D)' . (B'D)' . (B'C)')'

Y = C'D' + CD

= ((C'D')' + (CD)')'

Z = D'



Logic diagram for BCD to excess 3 code converter using minimum number of NAND gates

Truth table:

Decimal	Binay (BCD) 8 4 2 1	Excess-3 Code
0	0 0 0 0	0011
1	0 0 0 1	0100
2	0 0 1 0	0101
3	0 0 1 1	0110
4	0 1 0 0	0111
5	0 1 0 1	1000
6	0 1 1 0	1001
7	0 1 1 1	1010
8	1 0 0 0	1011
9	1 0 0 1	1100

From the truth table, the logic expression for the Excess-3 code outputs can be written as,

$E_3 = \sum_m (5, 6, 7, 8, 9) + \sum_d (10, 11, 12, 13, 14, 15)$
$E_2 = \sum_m (1, 2, 3, 4, 9) + \sum_d (10, 11, 12, 13, 14, 15)$
$E_1 = \sum_m (0, 3, 4, 7, 8) + \sum_d (10, 11, 12, 13, 14, 15)$
$E_0 = \sum_m (0, 2, 4, 6, 8) + \sum_d (10, 11, 12, 13, 14, 15)$

K-map simplification:

For E_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$E_3 = B_3 + B_2(B_0 + B_1)$

For E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$E_2 = B_2B_1'B_0' + B_2'(B_0 + B_1)$

For E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

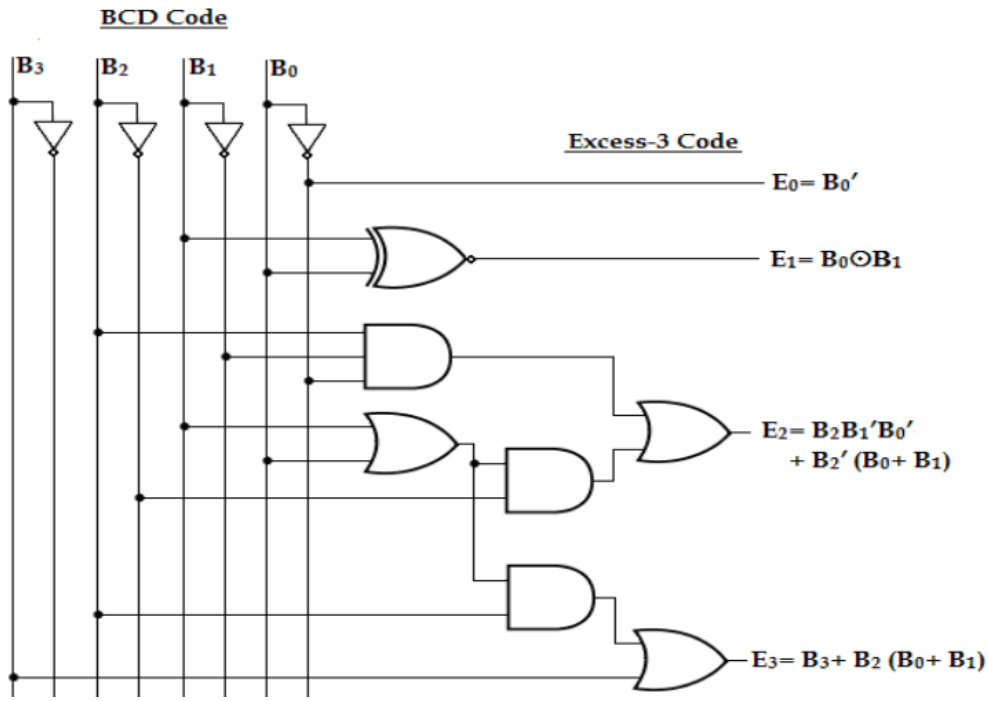
$E_1 = B_1'B_0' + B_1B_0$

For E_0

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

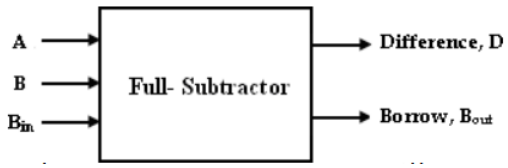
$E_0 = B_0'$

Logic diagram:



10. a) Design a full subtractor and implement using the logic gates. Also implement the same using half subtractor. (Nov. 2018)

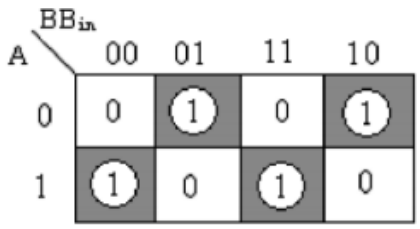
A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a _1‘ has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}.



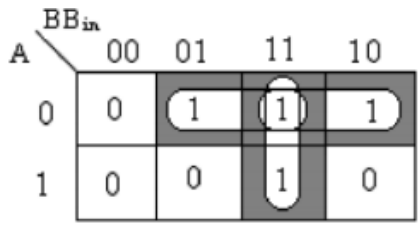
There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The BORROW output bit tells whether the minuend bit needs to borrow a _1‘ from the next possible higher minuend bit. The truth table is given below,

Inputs			Outputs	
A	B	B_{in}	Difference(D)	Borrow(B_{out})
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0

K-map simplification:



Difference, D = $A'B'B_{in}$+ $A'BB'_{in}$+ $AB'B'_{in}$+ ABB_{in}

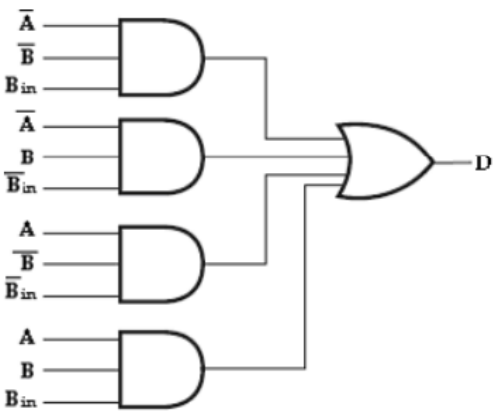


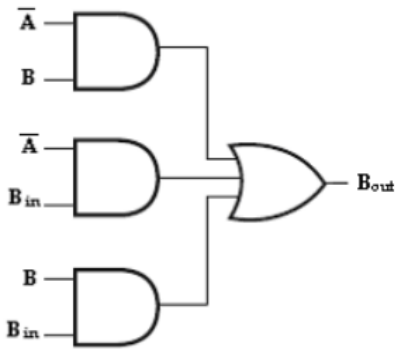
Borrow, B_{out} = $A'B$+ $A'B_{in}$ + BB_{in}

The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = $A'B'B_{in}$+ $A'BB'_{in}$ + $AB'B'_{in}$ + ABB_{in}
Borrow, B_{out} = $A'B$+ $A'C_{in}$ + BB_{in}.

Logic diagram:





The logic diagram of the full-subtractor can also be implemented with two half-subtractors and one OR gate. The difference,D output from the second half subtractor is the exclusive-OR of B_{in} and the output of the first half-subtractor, giving

Difference,D= $B_{in} \oplus (A \oplus B)$ $[x \oplus y = x'y + xy']$

$= B_{in} \oplus (A'B+AB')$

$= B'_{in}\ (A'B+AB') + B_{in}\ (A'B+AB')'$ $[(x'y+xy')' = (xy+x'y')]$

$= B'_{in}\ (A'B+AB') + B_{in}\ (AB+A'B')$

$= A'BB'_{in} + AB'B'_{in} + ABB_{in} + A'B'B_{in}$.

and the borrow output is,

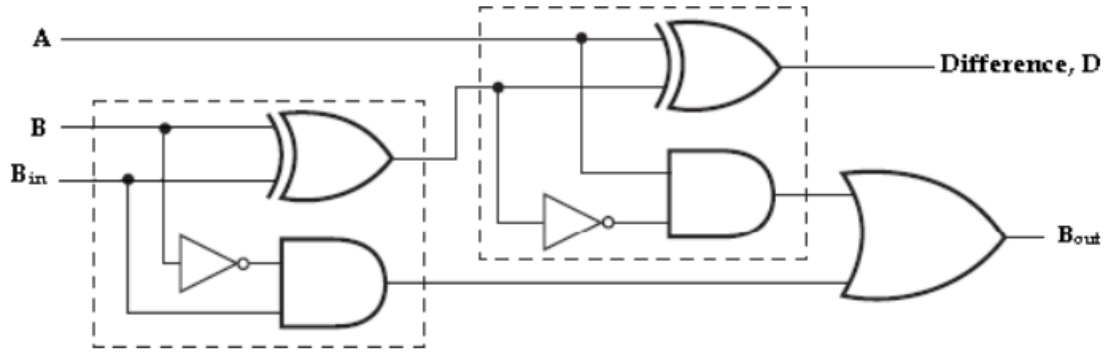
Borrow, B_{out} = $A'B$+ B_{in} $(A'B+AB')'$

$= A'B + B_{in}\ (AB+A'B')$

$= A'B + ABB_{in} + A'B'B_{in}$

Therefore,

we can implement full-subtractor using two half-subtractors and OR gate as,



Implementation of full-subtractor with two half-subtractors and an OR gate

11. What are magnitude comparators? Explain the design of magnitude comparators with the help of a suitable example. Construct 16-bit comparator using 4-bit comparator as a building block. (Nov. 2018)

Let us consider the two binary numbers A and B with four digits each. Write the coefficient of the numbers in descending order as,

$$A = A_3A_2A_1A_0$$
$$B = B_3 B_2 B_1 B_0,$$

Each subscripted letter represents one of the digits in the number. It is observed from the bit contents of two numbers that A = B when A3 = B3, A2 = B2, A1 = B1 and A0 = B0. When the numbers are binary they possess the value of either 1 or 0, the equality relation of each pair can be expressed logically by the equivalence function as,

$$X_i = A_iB_i + A_i'B_i' \qquad \text{for } i = 1, 2, 3, 4.$$

Or, $X_i = (A \oplus B)'$. or, $X_i' = A \oplus B$

Or, $X_i = (A_iB_i' + A_i'B_i)'$.

where, Xi =1 only if the pair of bits in position i are equal (ie., if both are 1 or both are 0). To satisfy the equality condition of two numbers A and B, it is necessary that all Xi must be equal to logic 1. This indicates the AND operation of all Xi variables. In other words, we can write the Boolean expression for two equal 4-bit numbers.

$$(A = B) = X_3X_2X_1 X_0.$$

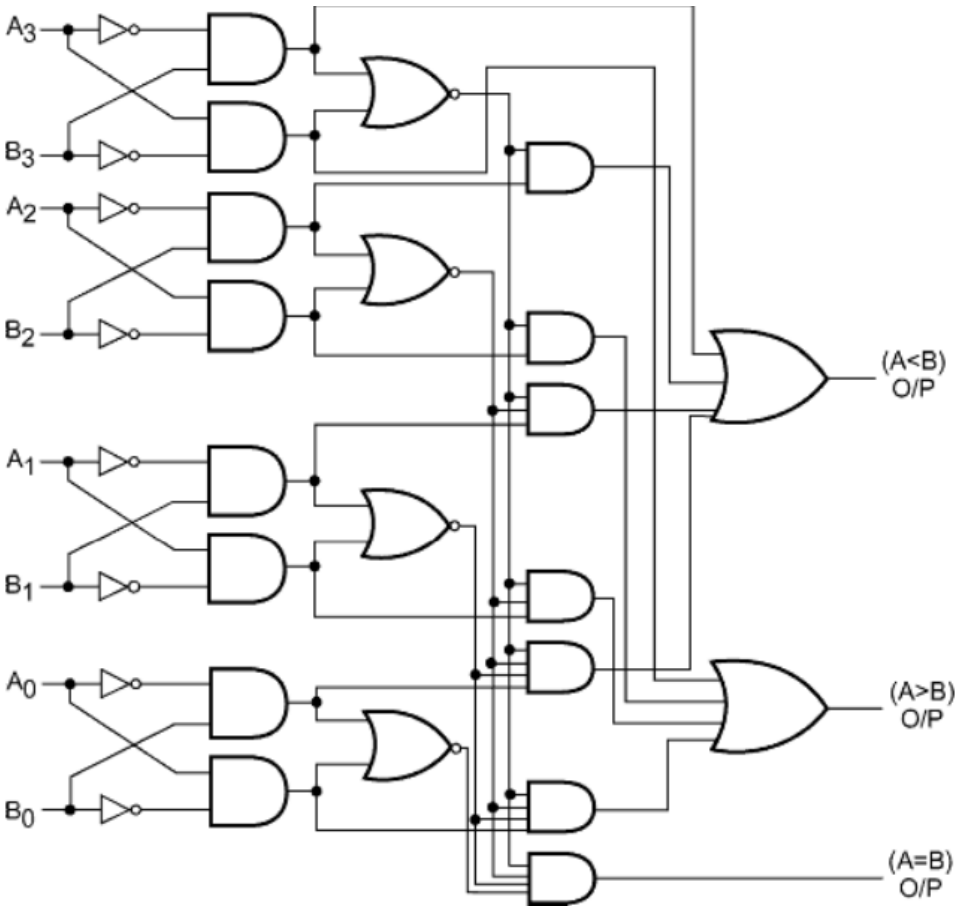
The binary variable (A=B) is equal to 1 only if all pairs of digits of the two numbers are equal.

To determine if A is greater than or less than B, we inspect the relative magnitudes of pairs of significant bits starting from the most significant bit. If the two digits of the most significant position are equal, the next significant pair of digits is compared. The comparison process is continued until a pair of unequal digits is found. It may be concluded that A>B, if the corresponding digit of A is 1 and B is 0. If the corresponding digit of A is 0 and B is 1, we conclude that A<B. Therefore, we can derive the logical expression of such sequential comparison by the following two Boolean functions,

$$(A>B) = A_3B_3' + X_3A_2B_2' + X_3X_2A_1B_1' + X_3X_2X_1A_0B_0'$$
$$(A<B) = A_3'B_3 + X_3A_2'B_2 + X_3X_2A_1'B_1 + X_3X_2X_1A_0'B_0$$

The symbols (A>B) and (A<B) are binary output variables that are equal to 1 when A>B or A<B, respectively.

The gate implementation of the three output variables just derived is simpler than it seems because it involves a certain amount of repetition. The unequal outputs can use the same gates that are needed to generate the equal output. The logic diagram of the 4-bit magnitude comparator is shown below,

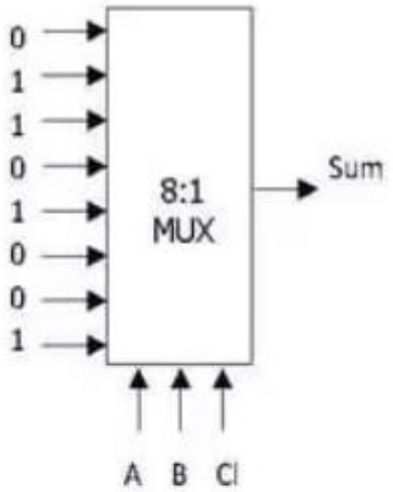


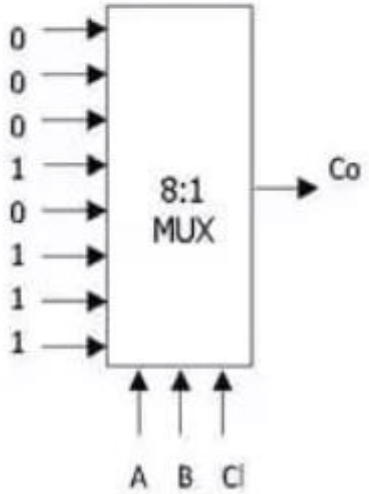
The four x outputs are generated with exclusive-NOR circuits and applied to an AND gate to give the binary output variable (A=B). The other two outputs use the x variables to

generate the Boolean functions listed above. This is a multilevel implementation and has a regular pattern.

12. Design:

a) Full adder using demultiplexer (Nov may 2019





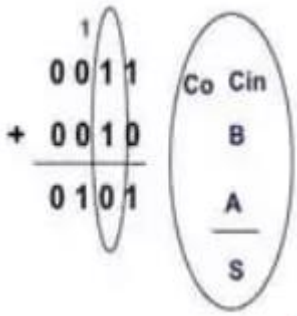
CO \ CI \ A B	00	01	11	10
0	0	0	1	0
1	0	1	1	1

S \ CI \ A B	00	01	11	10
0	0	1	0	1
1	1	0	1	0

S = CI xor A xor B

CO = B CI + A CI + A B = CI (A + B) + A B

A	B	CI	S	CO
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

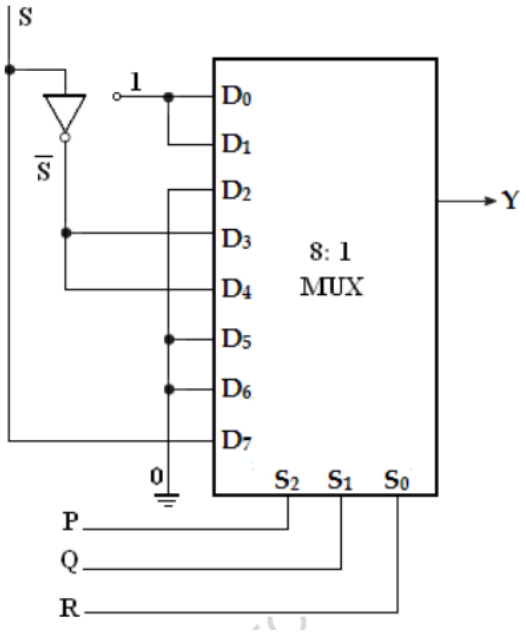


13. Simplify the function using multiplexer F (Q,P,R,S) = ∑ (0,1,3,4,8,9,15) (Nov. 2019)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{S}$	0	1	2	3	4	5	6	7
S	8	9	10	11	12	13	14	15
	1	**1**	**0**	$\overline{S}$	$\overline{S}$	**0**	**0**	**S**

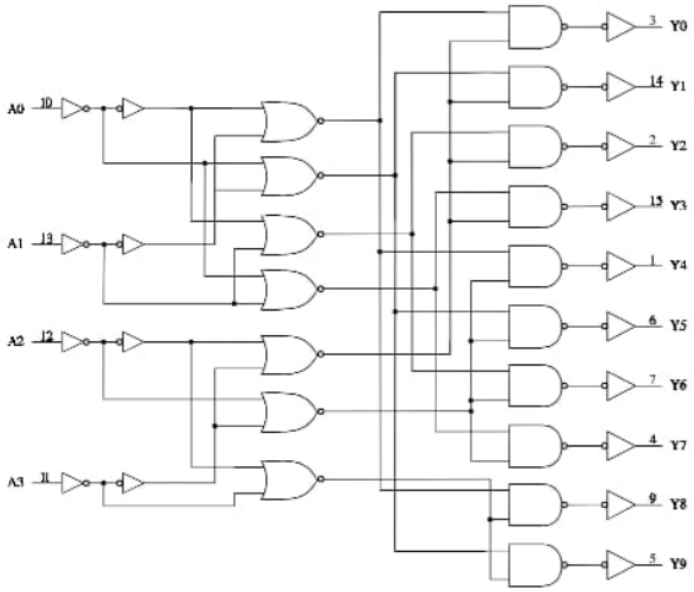
Multiplexer Implementation:



13. Explain the operation of 4 to 10 Decoder (May 2016, Nov. 2016)

The BCD to decimal decoders accept four active HIGH BCD inputs and provide 10 mutually exclusive active LOW outputs. The active LOW outputs facilitate addressing other MSI circuits with active LOW input enables.

Logic Diagram:

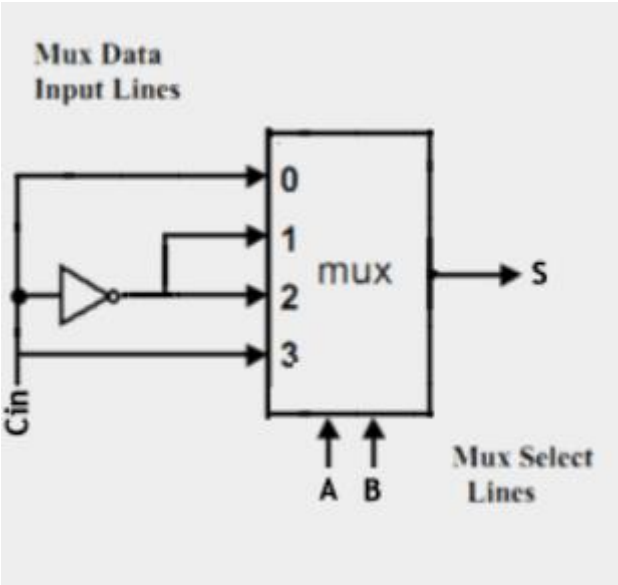


Truth table:

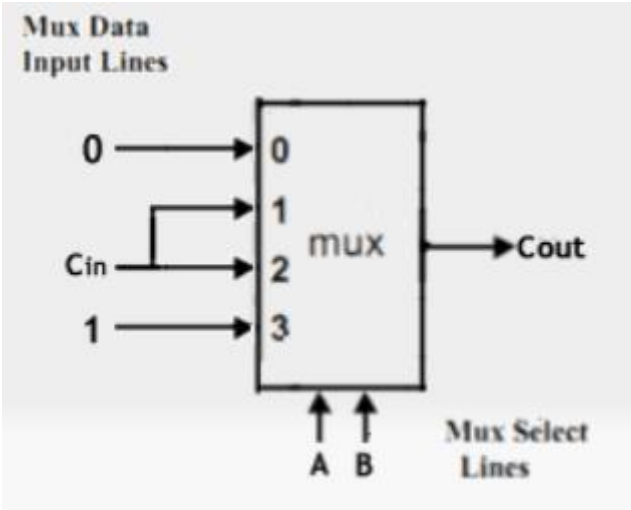
Inputs				Outputs									
A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9
L	L	L	L	H	L	L	L	L	L	L	L	L	L
L	L	L	H	L	H	L	L	L	L	L	L	L	L
L	L	H	L	L	L	H	L	L	L	L	L	L	L
L	L	H	H	L	L	L	H	L	L	L	L	L	L
L	H	L	L	L	L	L	L	H	L	L	L	L	L
L	H	L	H	L	L	L	L	L	H	L	L	L	L
L	H	H	L	L	L	L	L	L	L	H	L	L	L
L	H	H	H	L	L	L	L	L	L	L	H	L	L
H	L	L	L	L	L	L	L	L	L	L	L	H	L
H	L	L	H	L	L	L	L	L	L	L	L	L	H
H	L	H	L	L	L	L	L	L	L	L	L	L	L
H	L	H	H	L	L	L	L	L	L	L	L	L	L
H	H	L	L	L	L	L	L	L	L	L	L	L	L
H	H	L	H	L	L	L	L	L	L	L	L	L	L
H	H	H	L	L	L	L	L	L	L	L	L	L	L
H	H	H	H	L	L	L	L	L	L	L	L	L	L

14. Design a Full adder and implement it using suitlable multiplexer. (May 2018)

Logic diagram for sum out:



Logic diagram for carry out:



For sum out:

Cin \ AB	00	01	10	11
0	0	1	1	0
1	1	0	0	1
	Cin	C'in	C'in	Cin

For carry out:

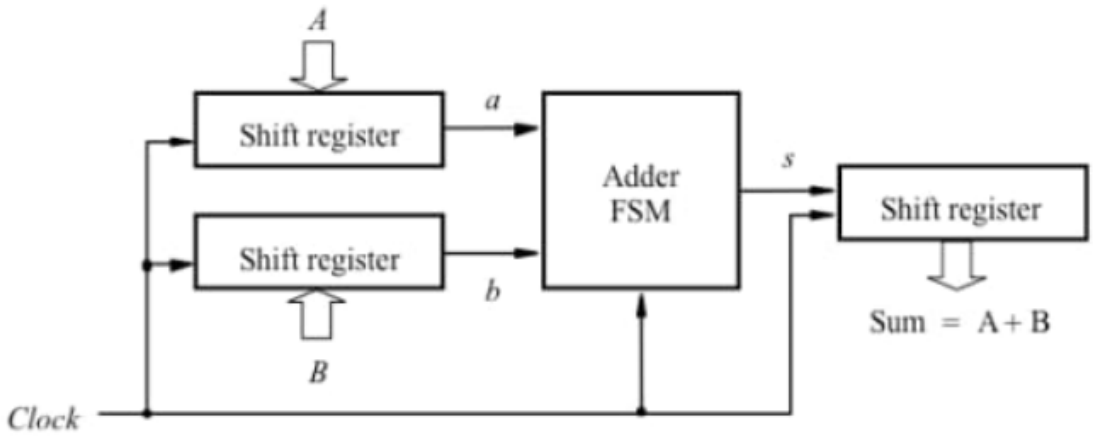
Cin \ AB	00	01	10	11
0	0	0	0	1
1	0	1	1	1
	0	Cin	Cin	1

S(A,B,Cin)= ∑(1,2,4,7)

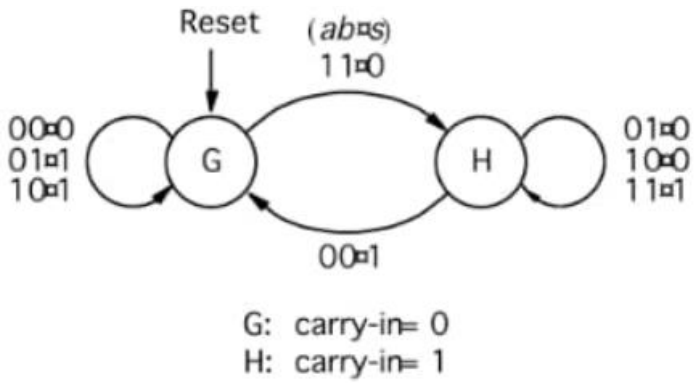
Cout(A,B,Cin)= ∑(3,5,6,7)

15. Explain the working principle of the following combinational circuit.

a) Serial adder (May 2017, May 2019)



Block diagram of a serial adder



State diagram of a serial adder

State table for the serial address:

Present state	Next state				Output s			
	ab =00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

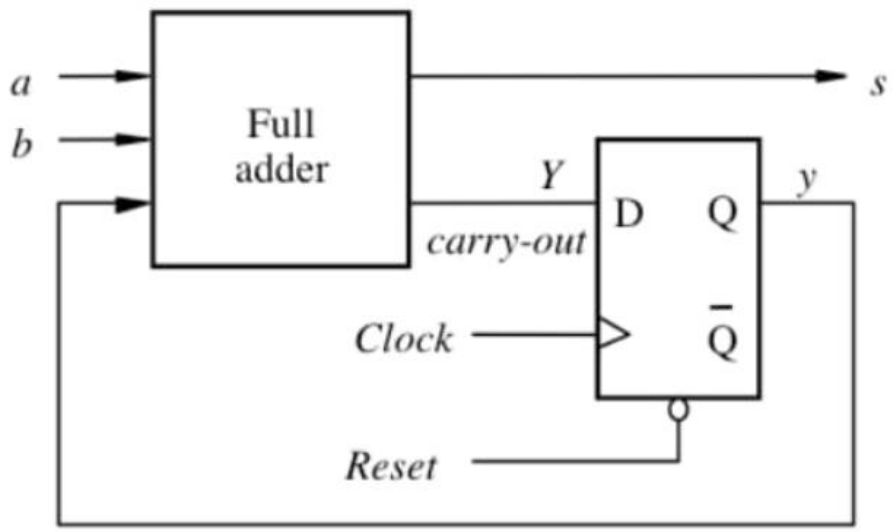
State assigned table:

Present state	Next state				Output			
	ab =00	01	10	11	00	01	10	11
y	Y				s			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

$$Y = ab + ay + by$$

$$s = a \oplus b \oplus c$$

Circuit for the adder:



BM T43–DIGITAL LOGIC THEORY AND DESIGN

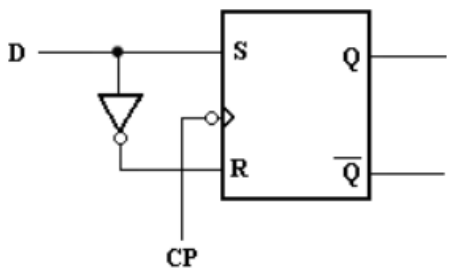
UNIT 3

PART – A

1. Build a D flip-flop using SR flip-flop IC. (May 2016, Nov. 2019)

Required Flip-Flop (D)			Given Flip-Flop (SR)	
Input	Present state	Next state	Flip-Flop Inputs	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0

Logic diagram



D Flip-Flop

2. Compare combinational circuits and sequential circuits. (May 2016, Sep. 2020)

S.No	Combinational logic	Sequential logic
1	The output variable, at all times depends on the combination of input variables.	The output variable depends not only on the present input but also depend upon the past history of inputs.
2	Memory unit is not required	Memory unit is required to store the past history of input variables.
3	Faster in speed	Slower than combinational circuits.
4	Easy to design	Comparatively harder to design.
5	Eg. Parallel adder	Eg. Serial adder

3. Write the excitation table for JK flip-flop (Nov. 2016, Nov. 2017)

Qn	Jn	Kn	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

State Table

Jn	Kn	Qn+1
0	0	Qn
0	1	0
1	0	1
1	1	$\overline{Qn}$

Characteristic Table

Qn \ JnKn	00	01	11	10
0	0	0	1	1
1	1	0	0	1

K-Map

$$Qn+1 = Jn.\overline{Qn} + \overline{Kn}.Qn$$

Characteristic Equation

Qn	Qn+1	Jn	Kn
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Excitation Table

The d indicates a "don't care" term. That is, the signal value can be either a 0 or a 1 as it doesn't affect the next state.

4. Differentiate between Synchronous and Asynchronous counters. (May 2017)

S.No	Synchronous sequential circuits	Asynchronous sequential circuits
1	Memory elements are clocked Flip-Flops	Memory elements are either unclocked Flip-Flops or time delay elements.
2	The change in input signals can affect memory element upon activation of clock signal.	The change in input signals can affect memory element at any instant of time.
3	The maximum operating speed of clock depends on time delays involved.	Because of the absence of clock, it can operate faster than synchronous circuits.
4	Easier to design	More difficult to design

5. State the application of Shift register. (May 2017, Sep. 2020)

- Converting between serial and parallel data
- Time delay devices
- Temporary storage in a processor
- Arithmetic operations- Multiply , Divide

6. What is meant by shift register? Name the different types of shift register. (Nov. 2017)

A register is simply a group of Flip-Flops that can be used to store a binary number. There must be one Flip-Flop for each bit in the binary number. For instance, a register used to store an 8-bit binary number must have 8 Flip-Flops. The Flip-Flops must be connected such that

the binary number can be entered (shifted) into the register and possibly shifted out. A group of Flip-Flops connected to provide either or both of these functions is called a shift register.

Types:

- Serial in- serial out,
- Serial in- parallel out,
- Parallel in- serial out,
- Parallel in- parallel out.

7. Define flip-flop. Name the different types of flip-flop. (May 2018)

Flip-Flops are synchronous bistable devices (has two outputs Q and Q'). In this case, the term synchronous means that the output changes state only at a specified point on the triggering input called the clock (CLK), i.e., changes in the output occur in synchronization with the clock.

Types:

- S-R Flip-Flop,
- J-K Flip-Flop,
- D Flip-Flop,
- T Flip-Flop.

8. Why gated D latch are called transparent latch? (May 2018)

As shown in the figure, D input goes directly to the S input, and its complement is applied to the R input. Therefore, only two input conditions exists, either S=0 and R=1 or S=1 and R=0. The truth table for D latch is shown below.

EN	D	Q_n	Q_{n+1}	State
1	0	x	0	Reset
1	1	x	1	Set
0	x	x	Q_n	No Change (NC)

As shown in the truth table, the Q output follows the D input. For this reason, D latch is called ***transparent latch***.

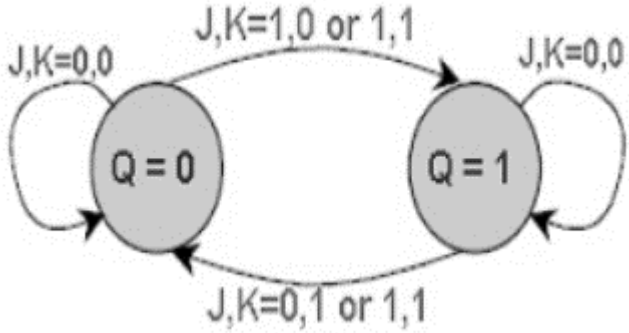
9. Explain Excitation table for RS flip-flop. (Nov. 2018)

Input	Present state	Next state	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	0
1	1	0	0	1

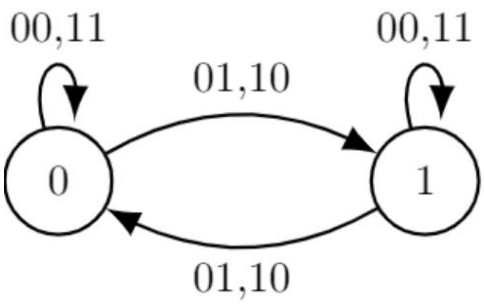
The above table presents the excitation table for SR Flip-Flop. It consists of present state (Q_n), next state (Q_{n+1}) and a column for each input to show how the required transition is achieved.

There are 4 possible transitions from present state to next state. The required Input conditions for each of the four transitions are derived from the information available in the characteristic table. The symbol 'x' denotes the don't care condition, it does not matter whether the input is 0 or 1.

10. Draw the state diagram of JK flip flop. (Nov. 2018)



11. Draw the state diagram of SR flip-flop. (May 2019)



12. Generalize how Jk flip-flop differs from SR flip-flop. (May 2019)

The only difference between JK flip-flop and SR flip-flop is tthat when both inputs of SR flip-flop is set to 1, the circuit produces the invalid states as outputs, but in the case of JK flip-flop, there are no invalid states even if both J and K flip-flops are set to 1.

13. How are clocked sequential circuits used? Give example. (Nov. 2019)

The clocked sequential circuits have flip-flops or gated latches for its memory elements.There is a periodic clock connected to the clock inputs of all the memory elements of the circuit to synchronize all th internal changes of the state.

14. Assume that a 4-bit synchronous counter is holding the count 0100.
What will be the count after its pulse?

Because 4-bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 t0 15, 1111 will be the count after 10 pulses.

PART – B

1.a) Write the excitation tables of SR, JK, T, D Flip-flops. (May 2016)

SR flip-flop:

Input	Present state	Next state	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	0
1	1	0	0	1

JK flip-flop:

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

D flip-flop:

Present State	Next State	Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

T flip-flop:

Present State	Next State	Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

1. b) Realize D and T fli-flop using JK flip-flop. (May 2016, Nov. 2016)

JK to T flip-flop:

Excitation table for above conversion is :

Input	Present state	Next state	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	J	K
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

K-map simplification

For J

T \ Q_n	0	1
0	0	x
1	1	x

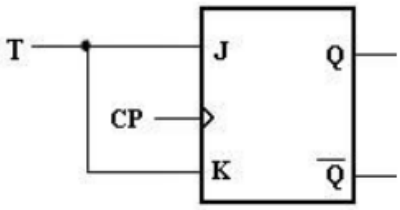
J = T

For K

T \ Q_n	0	1
0	x	0
1	x	1

K = T

Logic diagram



JK to D flip-flop:

The excitation table for the above conversion is

Input	Present state	Next state	Flip-Flop Inputs	
D	Q_n	Q_{n+1}	J	K
0	0	0	0	x 1
0	1	0	x	x
1	0	1	1	0
1	1	1	x	

K-map simplification

For J

D \ Q_n	0	1
0	0	x
1	1	x

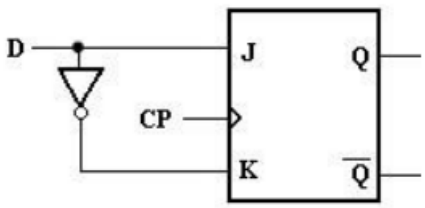
J = D

For K

D \ Q_n	0	1
0	x	1
1	x	0

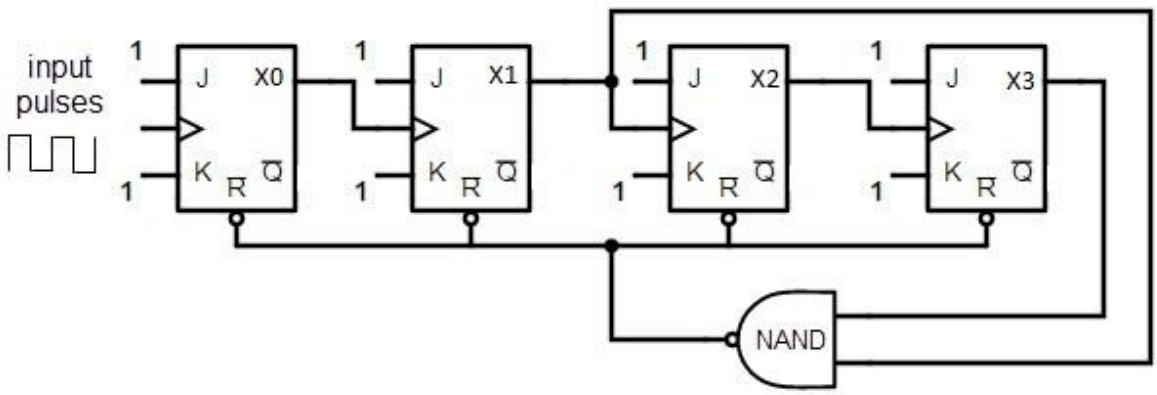
K = $\overline{D}$

Logic diagram



2. Design a decade counter using JK flip-flop. (May 2017, Sep.2020)

A binary coded decimal (BCD) is a serial digital counter that counts ten digits. And it resets for every new clock input. As it can go through 10 unique combinations of output, it is also called as “Decade counter”. A BCD counter can count 0000, 0001, 0010, 1000, 1001, 1010, 1011, 1110, 1111, 0000, and 0001 and so on.



The above figure shows a decade counter constructed with JK flip flop. The J output and K outputs are connected to logic 1. The clock input of every flip flop is connected to the output of next flip flop, except the last one.

Decade Counter Operation

When the Decade counter is at REST, the count is equal to 0000. This is first stage of the counter cycle. When we connect a clock signal input to the counter circuit, then the circuit .

Truth table:

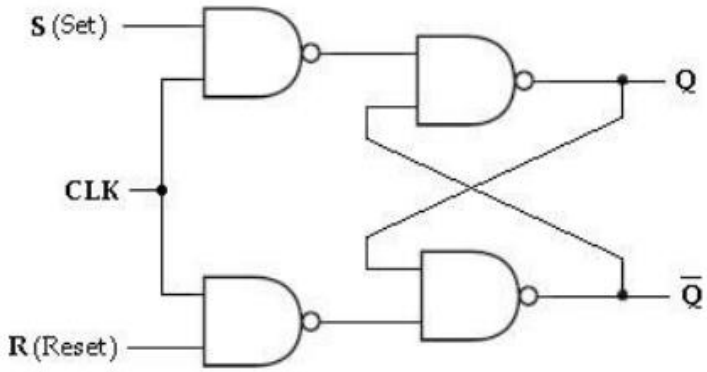
Input Pulses	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
0	0	0	0	0 (resets)

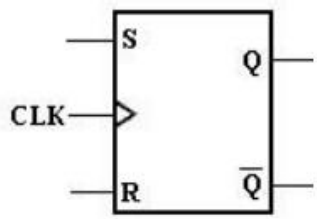
3. With necessary diagram and truth table, explain about different types of flip-flops. (May 2017, Sep. 2020)

SR flip-flop:

The S and R inputs of the S-R Flip-Flop are called synchronous inputs because data on these inputs are transferred to the Flip-Flop's output only on the triggering edge of the clock pulse. The circuit is similar to SR latch except enable signal is replaced by clock pulse (CLK). On the positive edge of the clock pulse, the circuit responds to the S and R inputs

When S is HIGH and R is LOW, the Q output goes HIGH on the triggering edge of the clock pulse, and the Flip-Flop is SET. When S is LOW and R is HIGH, the Q output goes LOW on the triggering edge of the clock pulse, and the Flip-Flop is RESET. When both S and R are LOW, the output does not change from its prior state. An invalid condition exists when both S and R are HIGH.





Logic Symbol

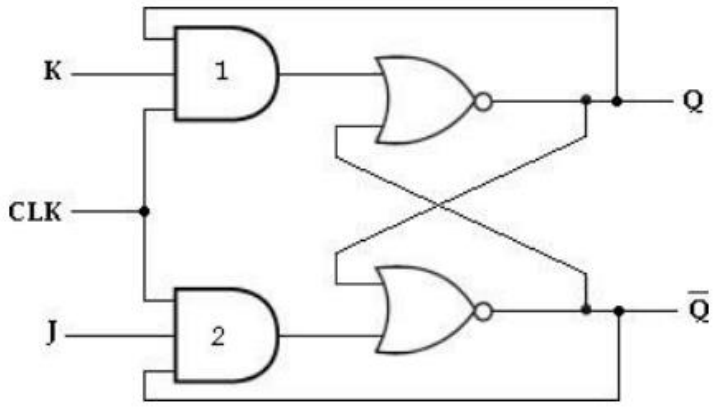
SR Flip-Flop

1 1	1 1	1 1	0 1	x x	Indeterminate *
0 0	x x	x x	0 1	0 1	No Change (NC)

Truth table for SR Flip-Flop

JK flip-flop:

JK means Jack Kilby, Texas Instrument (TI) Engineer, who invented IC in 1958. JK Flip-Flop has two inputs J(set) and K(reset). A JK Flip-Flop can be obtained from the clocked SR Flip-Flop by augmenting two AND gates as shown below.



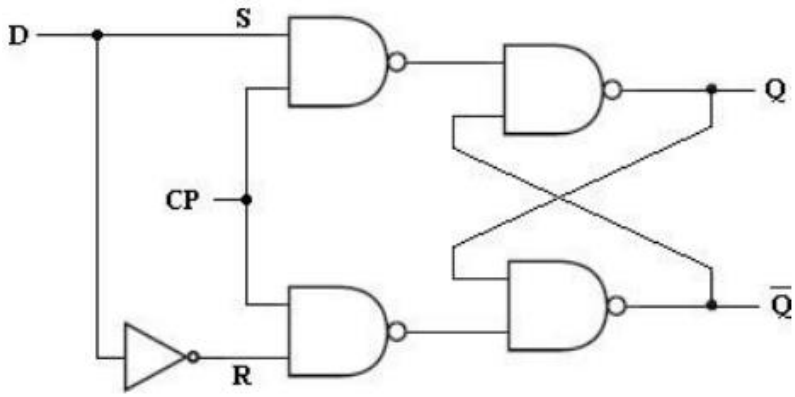
JK Flip Flop

Truth table:

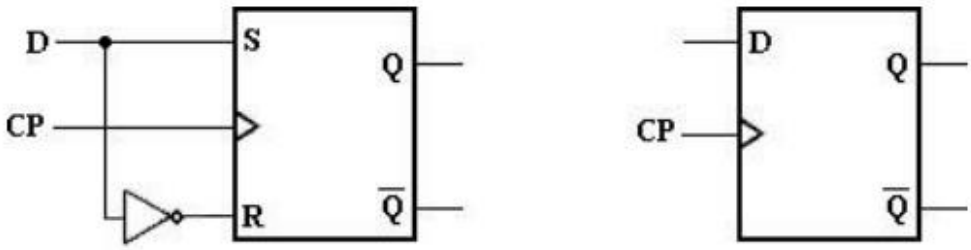
CLK	Inputs		Output	State
	J	K	Q_{n+1}	
1	0	0	Q_n	No Change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	Q_n'	Toggle

D flip-flop:

Like in D latch, in D Flip-Flop the basic SR Flip-Flop is used with complemented inputs. The D Flip-Flop is similar to D-latch except clock pulse is used instead of enable input.



To eliminate the undesirable condition of the indeterminate state in the RS Flip-Flop is to ensure that inputs S and R are never equal to 1 at the same time. This is done by D Flip-Flop. The D (delay) Flip-Flop has one input called delay input and clock pulse input. The D Flip-Flop using SR Flip-Flop is shown below.



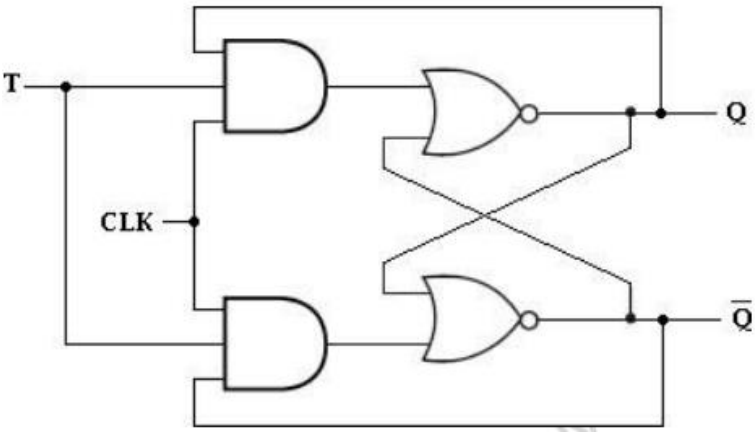
(a) Using SR flipflop (b) Graphic symbol

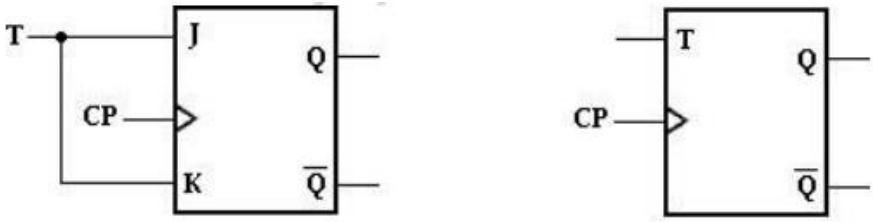
Clock	D	Q_{n+1}	State
1	0	0	Reset
1	1	1	Set
0	x	Q_n	No Change

Truth table for D Flip-Flop

T flip-flop:

The T (Toggle) Flip-Flop is a modification of the JK Flip-Flop. It is obtained from JK Flip-Flop by connecting both inputs J and K together, i.e., single input. Regardless of the present state, the Flip-Flop complements its output when the clock pulse occurs while input 1.





(a) Using JK flipflop (b) Graphic symbol

T	Q_{n+1}	State
0	Q_n	No Change
1	Q_n'	Toggle

Truth table for T Flip-Flop

4. a) Realize a JK flip-flop using SR flip-flop. (Nov. 2017, Nov. 2019)

Excitation table:

JK inputs		Outputs		SR inputs	
J	K	Q_p	Q_{p+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

K-map simplification:

For S

J \ KQ_n	00	01	11	10
0	0	x	0	0
1	1	x	0	1

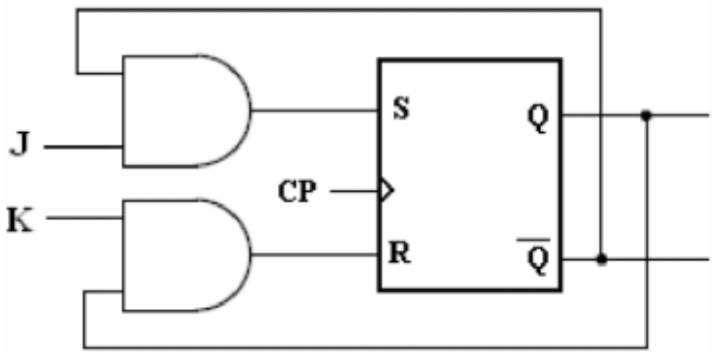
$S = J\overline{Q}_n$

For R

J \ KQ_n	00	01	11	10
0	x	0	1	x
1	0	0	1	0

$R = KQ_n$

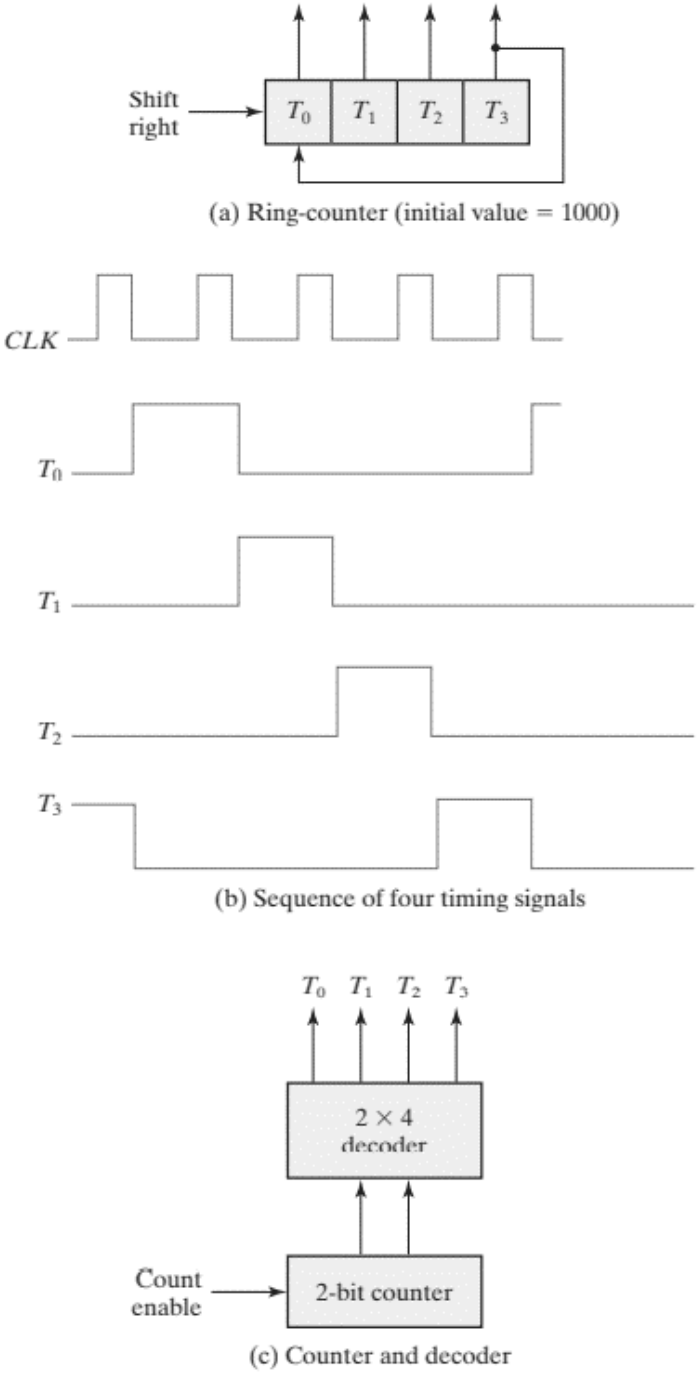
Logic diagram



JK Flip-Flop

4. b) Discuss the working of ring counter with necessary logic circuit and timing diagram. (Nov 2017)

Timing signals that control the sequence of operations in a digital system can be generated by a shift register or by a counter with a decoder. A ring counter is a circular shift register with only one flip-flop being set at any particular time; all others are cleared. The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals. Figure shows a four-bit shift register connected as a ring counter. The initial value of the register is 1000 and requires Preset/Clear flip-flops.



The single bit is shifted right with every clock pulse and circulates back from T3 to T0. Each flip-flop is in the 1 state once every four clock cycles and produces one of the four timing signals shown in Figure Each output becomes a 1 after the negative-edge transition of a clock pulse and remains 1 during the next clock cycle.

For an alternative design, the timing signals can be generated by a two-bit counter that goes through four distinct states. The decoder shown in Figure decodes the four states of the counter and generates the required sequence of timing signals.

To generate 2n timing signals, we need either a shift register with 2^n flip-flops or an n -bit binary counter together with an n -to-2 n -line decoder. For example, 16 timing signals can be generated with a 16-bit shift register connected as a ring counter or with a 4-bit binary counter and a 4-to-16-line decoder. In the first case, we need 16 flip-flops. In the second, we need 4 flip-flops and 16 four-input AND gates for the decoder. It is also possible to generate the timing signals with a combination of a shift register and a decoder. That way, the number of flip-flops is less than that in a ring counter, and the decoder requires only two-input gates. This combination is called a Johnson counter.

5. Design a synchronous counter to count in the following sequence 0,1,2,3,4,5,6,0,1,2 using JK flip-flop. (May 2016, Nov. 2017)

Step1: Determine the desired number of FFs From the given sequence the number of FFs is equal to 3.

Step2: Write the excitation table and circuit excitation table Table1 shows the excitation table for JK flip flop.

Excitation Table:

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

Circuit excitation table:

Present State			Next State			Flip Flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	X	0	X	X	1
1	1	0	0	0	0	X	X	X	1	0	X
1	1	1	0	0	0	X	X	X	1	X	1

k-map simplification:

For J_C

C\BA	00	01	11	10
0	0	0	1	0
1	X	X	X	X

$J_C = BA$

For J_B

C\BA	00	01	11	10
0	0	1	X	X
1	0	0	X	X

$J_B = \overline{C}\overline{A}$

For J_A

C\BA	00	01	11	10
0	X	X	1	0
1	1	X	X	0

$J_A = \overline{B}\overline{C}$

For K_C

C\BA	00	01	11	10
0	X	X	X	X
1	0	X	X	X

$K_C = \overline{B}\overline{A}$

For K_B

C\BA	00	01	11	10
0	X	X	0	X
1	X	X	1	1

$K_B = C\,B$

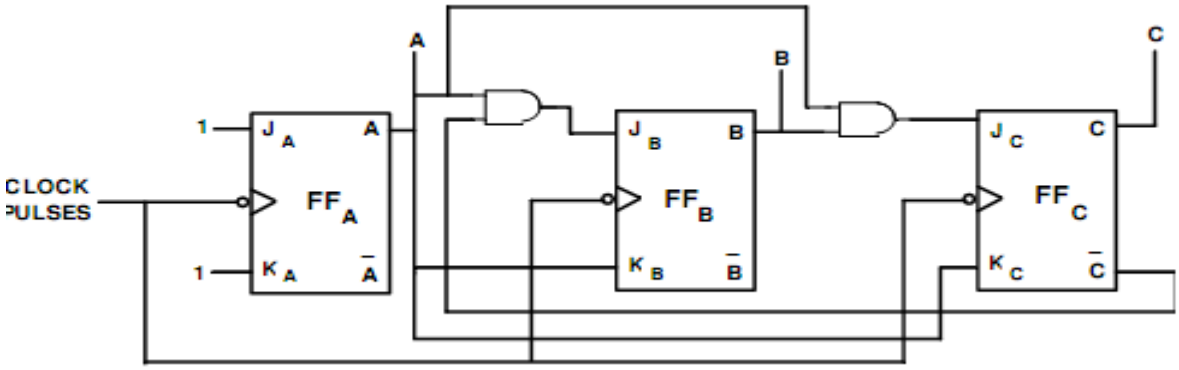
For K_A

C\BA	00	01	11	10
0	X	1	1	X
1	X	1	1	X

$K_A = A$

66

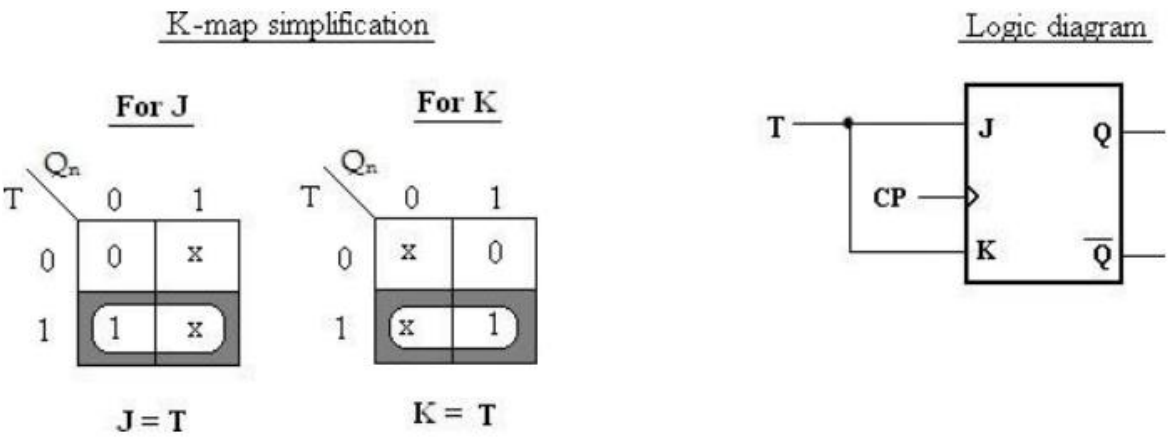
Logic diagram:



6. a) Realize a JK flip-flop from T flip-flop. (May 2018)

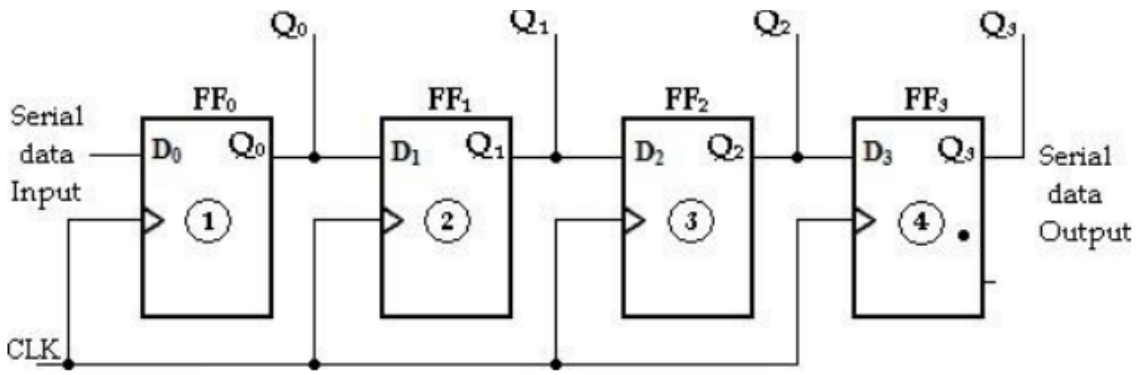
Excitation table:

Input	Present state	Next state	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	J	K
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

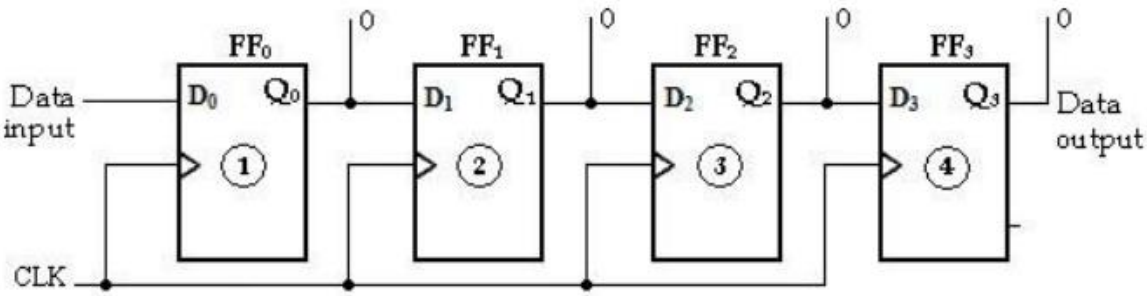


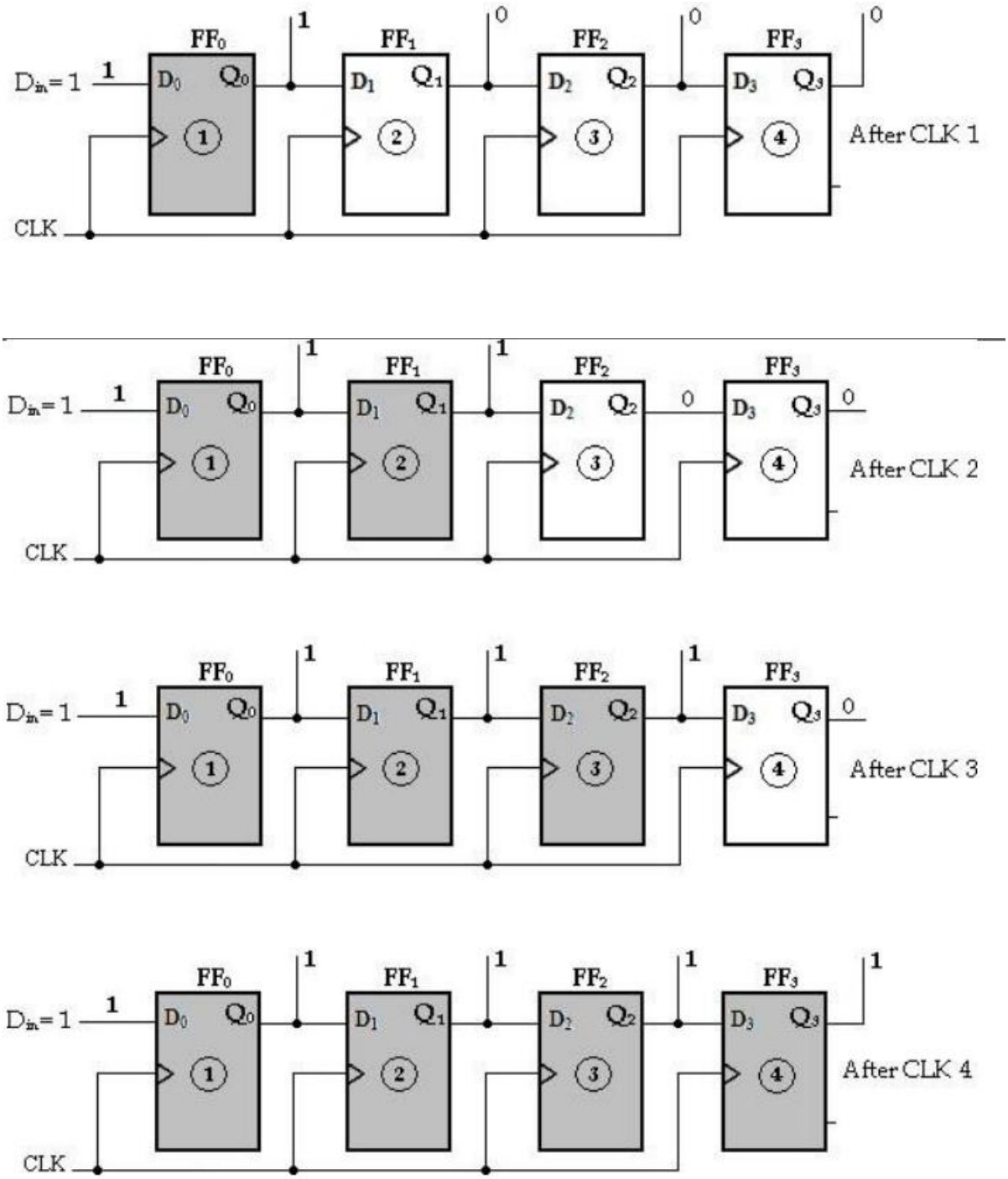
6. b) Write short notes on SIPO and draw output waveforms. (May 2018)

In this shift register, data bits are entered into the register in the same as serialin serial-out shift register. But the output is taken in parallel. Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously instead of on a bit-by-bit.



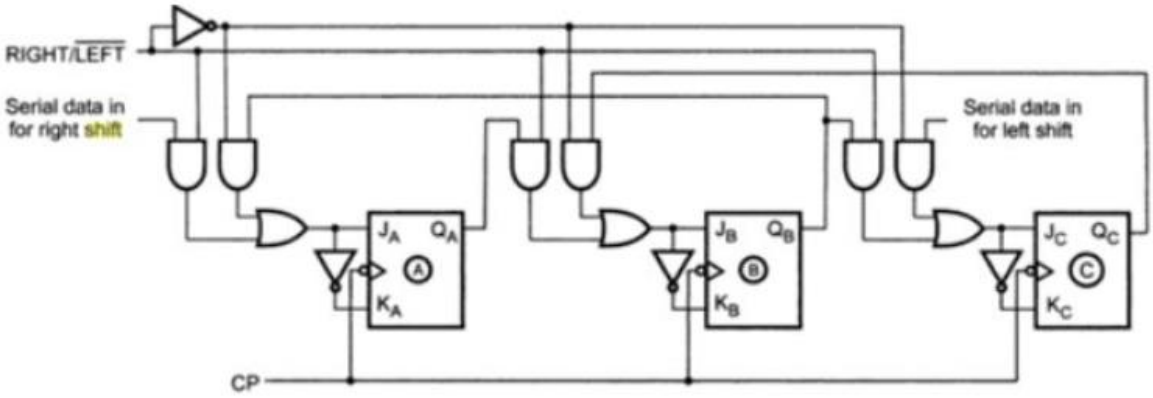
Serial-In parallel-Out Shift Register





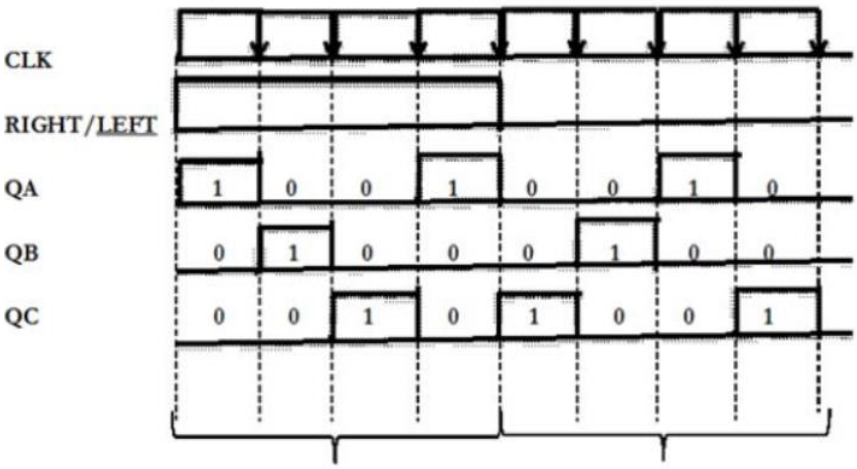
Four bits (1111) being serially entered into the register

7. Design a 3-bit bidirectional shift register. (May 2018)



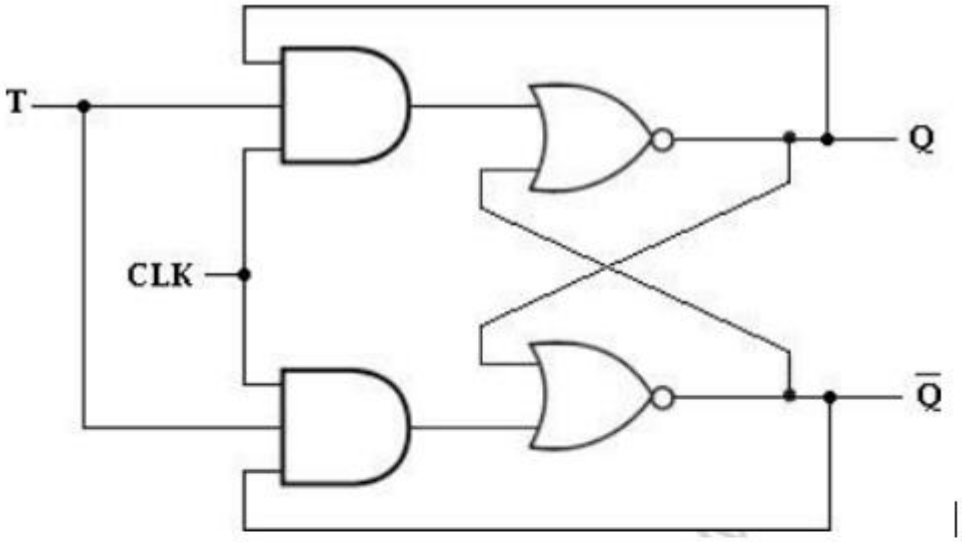
- The above diagram shows 3-bit Bi-directional Shift register, this type of register allows shifting of data either to the left side or to the right side.
- It is implemented by using logic gate circuitry that enables the transfer of data from one stage to the next stage to left or to the right, depending on the level of a control line.
- The RIGHT/LEFT is the control input signal which allows data shifting either towards right or towards left. A high on this line enables the shifting of data towards right and a low enables it towards left.
- When RIGHT/LEFT signal is high, out of the pair of each 2 AND gates the first AND gate from each of the pair is enabled.
- The state of the Q output of each Flip Flop is passed through the input of the following flip flop. When the clock pulse arrives, the data are shifted one place to the right.
- When RIGHT/LEFT signal is low, out of the pair of each 2 AND gates the second AND gate from each of the pair is enabled.
- The state of the Q output of each Flip Flop is passed through the input of the preceding flip flop. When the clock pulse arrives, the data are shifted one place to the left.

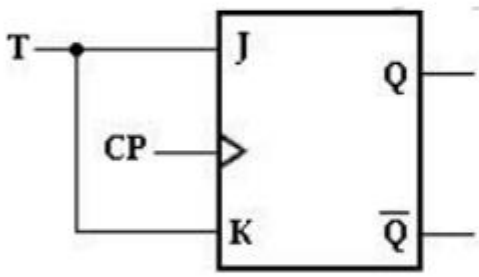
Waveform:



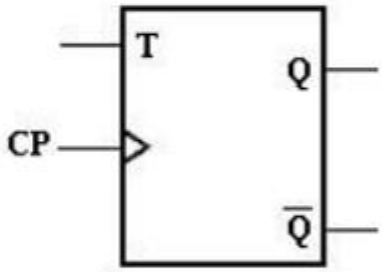
8. a) Explain T flip-flop with suitable internal structure (Nov. 2018)

The T (Toggle) Flip-Flop is a modification of the JK Flip-Flop. It is obtained from JK Flip-Flop by connecting both inputs J and K together, i.e., single input. Regardless of the present state, the Flip-Flop complements its output when the clock pulse occurs while input 1.





(a) Using JK flipflop



(b) Graphic symbol

T	Q_{n+1}	State
0	Q_n	No Change
1	Q_n'	Toggle

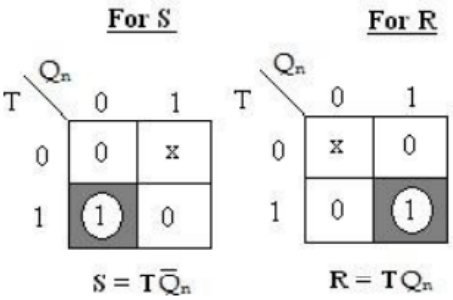
Truth table for T Flip-Flop

8. b) Convert SR flip-flop to T flip-flop. (Nov. 2018)

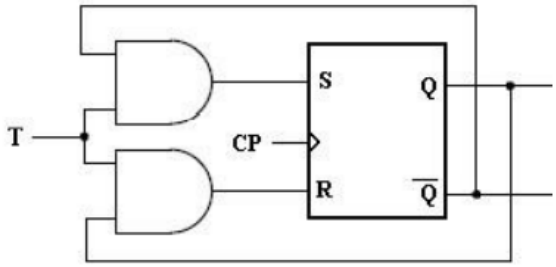
The excitation table for the above conversion is

Input	Present state	Next state	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	0
1	1	0	0	1

K-map simplification



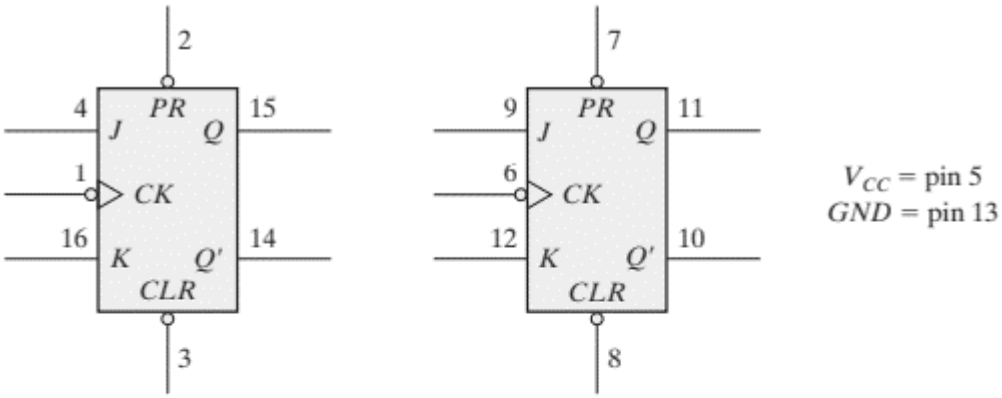
Logic diagram



9. a) With a neat sketch of circuit describe JK master- slave flip-flop. (May 2019)

Connect a master–slave D flip-flop using two D latches and an inverter. Connect the D input to a switch and the clock input to a pulser. Connect the output of the master latch to one indicator lamp and the output of the slave latch to another indicator lamp. Set the value of the input to the complement value of the output. Press the push button in the pulser and then release it to produce a single pulse. Observe that the master changes when the pulse goes positive and the slave follows the change when the pulse goes negative. Press the push button again a few times while observing the two indicator lamps. Explain the transfer sequence from input to master and from master to slave.

Connect the complement output of the flip-flop to the D input. This causes the flip-flop to be complemented with each clock pulse. Using a dual-trace oscilloscope, observe the waveforms of the clock and the master and slave outputs. Verify that the delay between the master and the slave outputs is equal to the positive half of the clock cycle. Obtain a timing diagram showing the relationship between the clock waveform and the master and slave outputs.

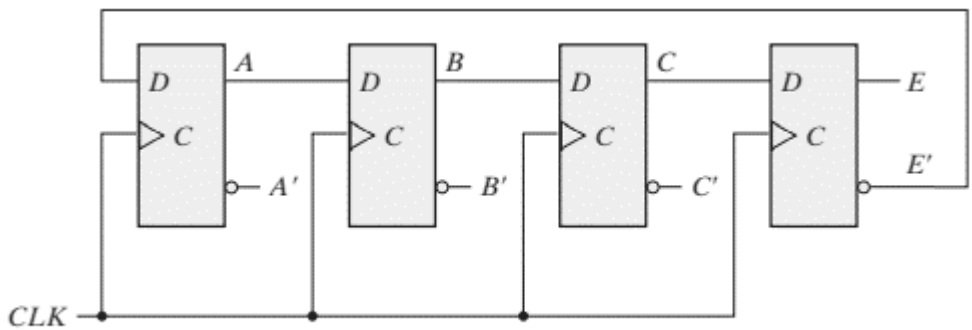


Function table

Inputs					Outputs	
Preset	Clear	Clock	J	K	Q	Q'
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	1	1
1	1	↑	0	0	No change	
1	1	↑	0	1	0	1
1	1	↑	1	0	1	0
1	1	↑	1	1	Toggle	

9. b) Draw the 4-bit Johnson counter. (May 2019)

A k -bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states. The number of states can be doubled if the shift register is connected as a switch-tail ring counter. A switch-tail ring counter is a circular shift register with the complemented output of the last flip-flop connected to the input of the first flip-flop. Figure shows such a shift register.



(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

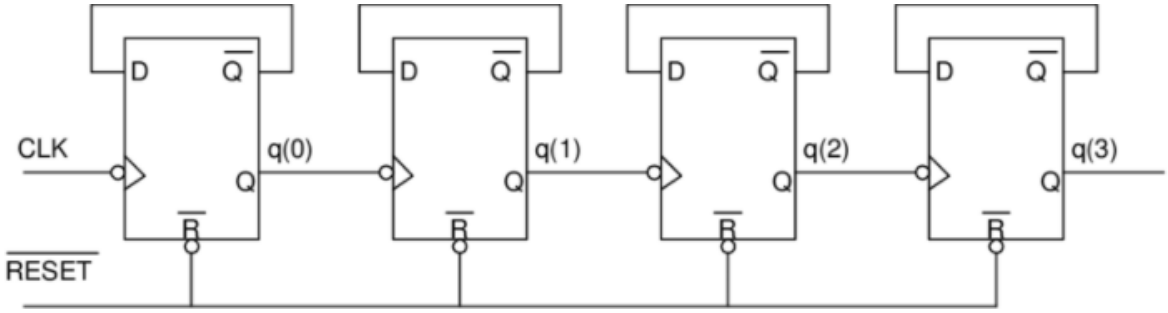
(b) Count sequence and required decoding

The register shifts its contents once to the right with every clock pulse, and at the same time, the complemented value of the E flip-flop is transferred into the A flip-flop. Starting from a cleared state, the switch-tail ring counter goes through a sequence of eight states, as listed. In general, a k -bit switch-tail ring counter will go through a sequence of 2 k states. Starting from all 0's, each shift operation inserts 1's from the left until the register is filled with all 1's. In the next sequences, 0's are inserted from the left until the register is again filled with all 0's. A Johnson counter is a k -bit switch-tail ring counter with 2 k decoding gates to provide outputs for 2 k timing signals.

The decoding gates are not shown, but are specified in the last column of the table. The eight AND gates listed in the table, when connected to the circuit, will complete the construction of the Johnson counter. Since each gate is enabled during one particular state sequence, the outputs of the gates generate eight timing signals in succession. The decoding of a k -bit switch-tail ring counter to obtain 2 k timing signals follows a regular pattern. The all-0's state is decoded by taking the complement of the two extreme flip-flop outputs. The all-1's state is decoded by taking the normal outputs of the two extreme flip-flops. All other states are decoded from an adjacent 1, 0 or 0, 1 pattern in the sequence.

For example, sequence 7 has an adjacent 0, 1 pattern in flip-flops B and C. The decoded output is then obtained by taking the complement of B and the normal output of C, or BC. One disadvantage of the circuitis that if it finds itself in an unused state, it will persist in moving from one invalid state to another and never find its way to a valid state.

The difficulty can be corrected by modifying the circuit to avoid this undesirable condition. One correcting procedure is to disconnect the output from flip-flop B that goes to the D input of flip-flop C and instead enable the input of flip-flop C by the function DC = (A + C)B where DC is the flip-flop input equation for the D input of flip-flop C. Johnson counters can be constructed for any number of timing sequences. The number of flip-flops needed is one-half the number of timing signals. The number of decoding gates is equal to the number of timing signals, and only two-input gates are needed.

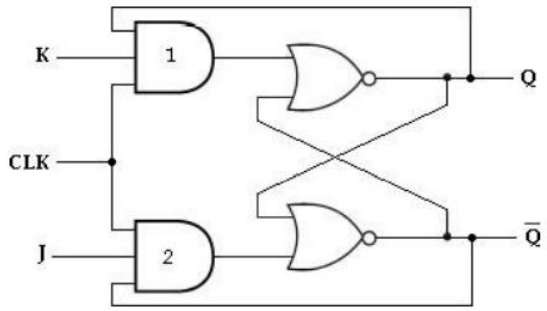


The structural description of a ripple counter is shown in HDL Example 6.4. The first module instantiates four internally complementing flip-flops defined in the second module as Comp_D_flip_flop (Q, CLK, Reset) . The clock (input CLK) of the first flip-flop is connected to the external control signal Count . (Count replaces CLK in the port list of instance F0 .) The clock input of the second flip-flop is connected to the output of the first. (A0 replaces CLK in instance F1 .) Similarly, the clock of each of the other flip-flops is connected to the output of the previous flip-flop. In this way, the flip-flops are chained together to create a ripple counter as shown.

The second module describes a complementing flip-flop with delay. The circuit of a complementing flip-flop is constructed by connecting the complement output to the D input. A reset input is included with the flip-flop in order to be able to initialize the counter; otherwise the simulator would assign the unknown value (x) to the output of the flip-flop and produce useless results. The flip-flop is assigned a delay of two time units from the time that the clock is applied to the time that the flip-flop complements its output. The delay is specified by the statement Q 6 = #2 <Q.

Notice that the delay operator is placed to the right of the nonblocking assignment operator. This form of delay, called intra-assignment delay,has the effect of postponing the assignment of the complemented value of Q to Q. The effect of modeling the delay will be apparent in the simulation results. This style of modeling might be useful in simulation, but it is to be avoided when the model is to be synthesized. The results of synthesis depend on the ASIC cell library that is accessed by the tool, not on any propagation delays that might appear within the model that is to be synthesized.

10. Explain the JK flip-flop with logic diagram, characteristic table and equation, present state, next state table, state diagram and application/excitation table (Nov. 2019)



JK Flip Flop

Truth table:

CLK	Inputs		Output	State
	J	K	Q_{n+1}	
1	0	0	Q_n	No Change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	Q_n'	Toggle

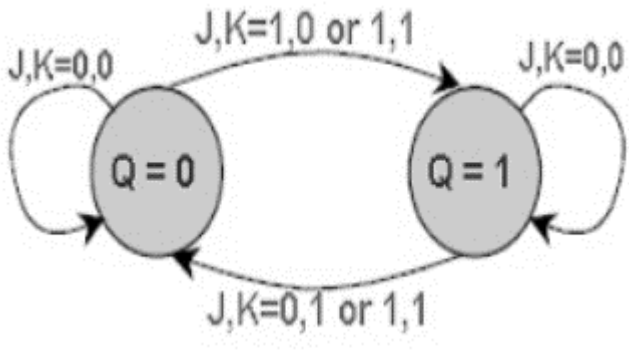
Characteristic table:

Present State	Inputs		Next State
Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Excitation table:

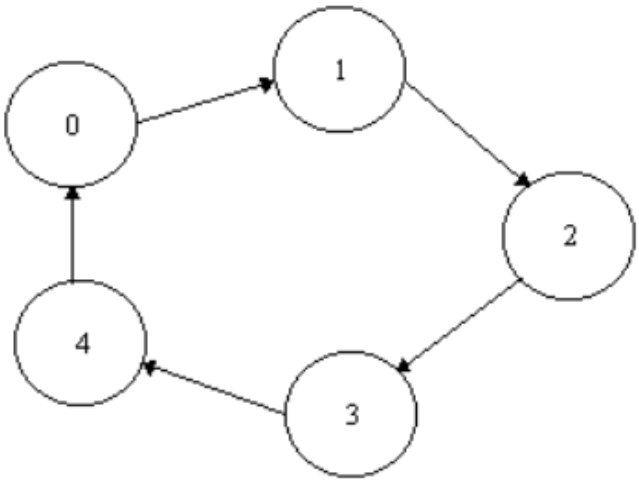
Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

State diagram:



11. Design a mod – 5 synchronous counters. (Nov. 2019)

State Diagram:



State Table

	CP
0	1
1	2
2	3
3	4
4	0
5	0
6	0
7	0

Next State

Secondary State Assignment

	CP		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
	y2	y1	y0

Transition Table

y2	y1	y0	CP		
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0
			y2	y1	y0

Modified Transition Table

y2	y1	y0	CP		
0	0	0	0	0	I
0	0	1	0	I	ϕ
0	1	0	0	1	I
0	1	1	I	ϕ	ϕ
1	0	0	ϕ	0	0
1	0	1	ϕ	0	ϕ
1	1	0	ϕ	ϕ	0
1	1	1	ϕ	ϕ	ϕ
			y2	y1	y0

Excitation Equations

$y_2 \backslash y_1y_0$	00	01	11	10
0	0	0	I	0
1	ϕ	ϕ	ϕ	ϕ

y_2

$y_2 \backslash y_1y_0$	00	01	11	10
0	0	I	ϕ	1
1	0	0	ϕ	ϕ

y_1

$y_2 \backslash y_1y_0$	00	01	11	10
0	I	ϕ	ϕ	I
1	0	ϕ	ϕ	0

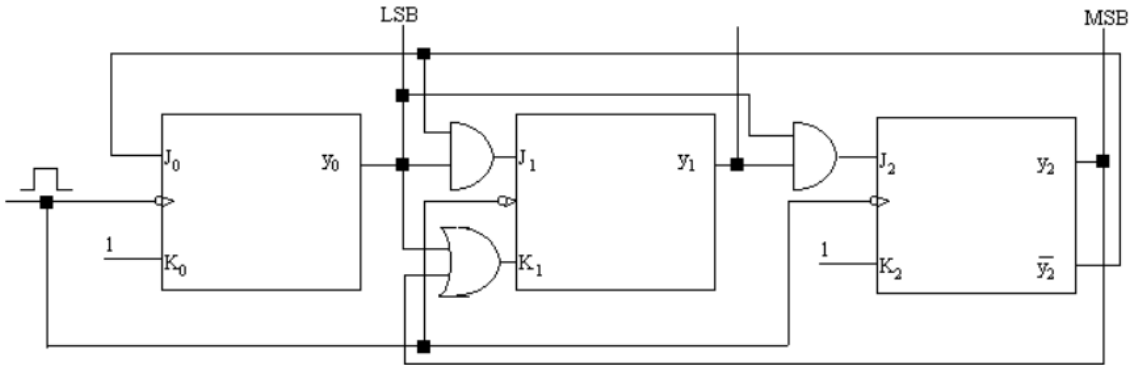
y_0

Using J-K Flip-Flops

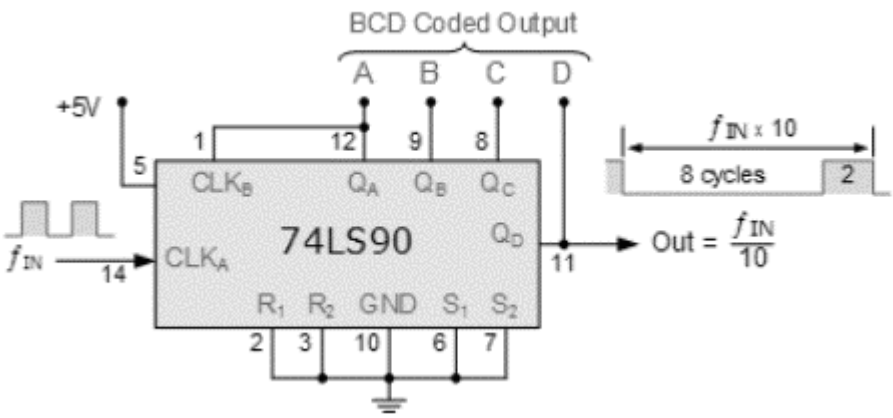
$J_2 = y_1 \cdot y_0$ $\quad$ $J_1 = \overline{y_2} \cdot y_0$ $\quad$ $J_0 = \overline{y_2}$

$K_2 = 1$ $\quad$ $K_1 = y_2 + y_0$ $\quad$ $K_0 = 1$

Logic Schematic

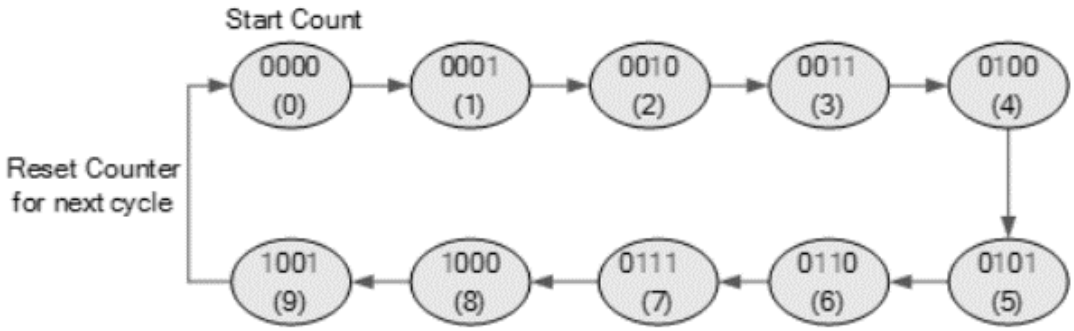


11. a) Design a synchronous decade counter using T flip-flop and construct the timing diagram. (Nov. 2018)



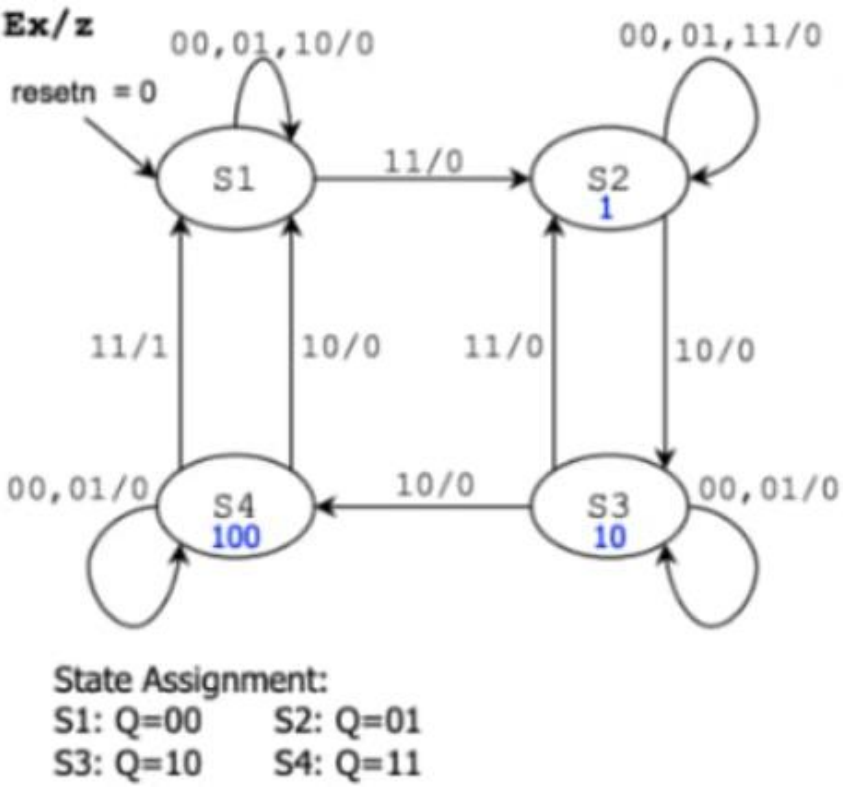
Clocked T-type flip-flops act as a binary divide-by-two counter and in asynchronous counters, the output of one counting stage provides the clock pulse for the next stage. Then a flip-flop counter has two possible output states and by adding more flip-flop stages, we can make a divide-by-2^N counter. But the problem with 4-bit binary counters is that they count from 0000 to 1111. That is from 0 to 15 in decimal.

State diagram:



11. b) Design a Mealy model of sequence detector to detect the pattern 1001. (Nov. 2019)

State diagram:



State table and Excitation table:

E	x	PRESENT STATE	NEXT STATE	z
0	0	S1	S1	0
0	0	S2	S2	0
0	0	S3	S3	0
0	0	S4	S4	0
0	1	S1	S1	0
0	1	S2	S2	0
0	1	S3	S3	0
0	1	S4	S4	0
1	0	S1	S1	0
1	0	S2	S3	0
1	0	S3	S4	0
1	0	S4	S1	0
1	1	S1	S2	0
1	1	S2	S2	0
1	1	S3	S2	0
1	1	S4	S1	1

		PRESENT STATE		NEXTSTATE		
E	x	Q_1	Q_0 (t)	Q_1	Q_0 (t+1)	z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	1	1	0
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	0	0	1

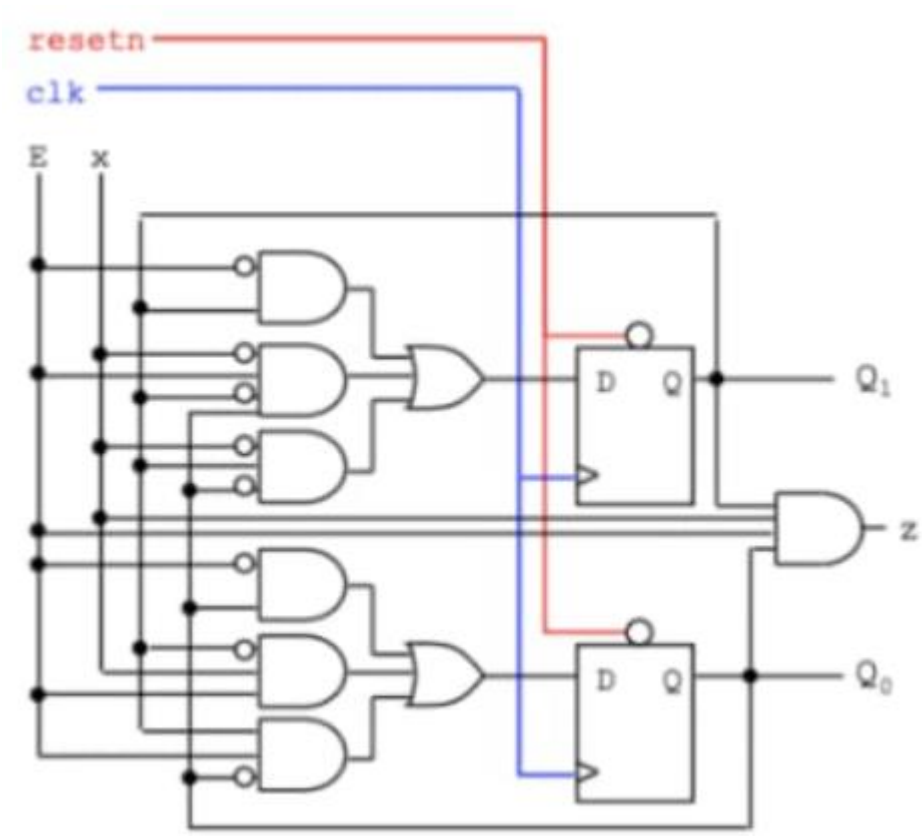
$Q_1(t+1)$

Q_1Q_0 \ Ex	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	1	1	0	0
10	1	1	0	1

$Q_0(t+1)$

Q_1Q_0 \ Ex	00	01	11	10
00	0	0	1	0
01	1	1	1	0
11	1	1	0	0
10	0	0	1	1

Circuit implementation:



DIGITAL LOGIC THEORY AND DESIGN

UNIT IV

PART – A (2 MARKS)

1. State the problems in asynchronous circuits. (MAY 2016)

- Some asynchronous circuits may require extra power for certain operations.
- More difficult to design and subject to problems like sensitivity to the relative arrival times of inputs at gates. If transitions on two inputs arrive at almost the same time, the circuit can go into the wrong state depending on slight differences in the propagation delays of the gates which are known as race condition.
- The number of circuit elements (transistors) maybe double that of synchronous circuits. Fewer people are trained in this style compared to synchronous design. Difficult to test and debug. Their output is uncertain.

2. Give classification of synchronous sequential circuits. (MAY 2016)

There are two types of sequential circuit, synchronous and asynchronous. Synchronous types use pulsed or level inputs and a clock input to drive the circuit (with restrictions on pulse width and circuit propagation). Asynchronous sequential circuits do not use a clock signal as synchronous circuits do.

3. Define bit, byte and word. (NOV 2016)

A bit is a binary digit, the smallest increment of data on a computer. A bit can hold only one of two values: 0 or 1, corresponding to the electrical values of off or on, respectively.A byte is eight bits, a word is 2 bytes (16 bits), a doubleword is 4 bytes (32 bits), and a quadword is 8 bytes (64 bits).

4. List the basic types of programmable logic devices. (NOV 2016)

- Programmable Logic Array (PLA)
- Programmable Array Logic (PAL)
- Generic Logic Array (GLA)

- Complex Programmable Logic Device (CPLD)
- Field Programmable Gate Array (FPGA)

5. Why synchronous sequential machines are called as deterministic machines? (MAY 2017)

A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.

6. What do you mean by Hazards in Digital circuits / Define the term hazard? (MAY 2017/ DEC 2017/ SEP 2020)

A hazard, if exists, in a digital circuit causes a temporary fluctuation in output of the circuit. In other words, a hazard in a digital circuit is a temporary disturbance in ideal operation of the circuit which if given some time, gets resolved itself.

7. Distinguish between Moore and Mealy machine. (DEC 2017)

	Mealy	Moore
(1)	O/Ps depend on the present state and present I/Ps	O/Ps depend only on the present state
(2)	The O/P change asyn -chronously with the enabling clock edge	Since the O/Ps change when the state changes, and the state change is synchronous with the enabling clock edge, O/Ps change synchronously with this clock edge
(3)	A counter is not a Mealy machine	A counter is a Moore machine
(4)	A Mealy machine will have the same # or fewer states than a Moore machine	

8. What is a dead lock condition? (MAY 2018)

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).

9. What is static 1 hazard and static 0 hazard? (MAY 2018)

There are two types of static hazards: Static-1 Hazard: the output is currently 1 and after the inputs change, the output momentarily changes to 0, 1 before settling on 1. Static-0 Hazard: the output is currently 0 and after the inputs change, the output momentarily changes to 1, 0 before settling on 0.

10. Write a short note on fundamental mode asynchronous circuit? (DEC 2018)

The inputs (I) to the synchronous circuits change only when the circuit is stable, that means when the state variables (S) are not in their transition state. Another assumption is that the inputs are levels and not pulses.

11. What is static and dynamic hazards? (DEC 2018)

Static hazard occur when an input changes and it causes the output to change at the same moment before output becomes stable. Dynamic hazard occur when output changes for two adjacent inputs while the output should change only once. Dynamic hazard is complex to resolve. It can be eliminated by using redundant gates.

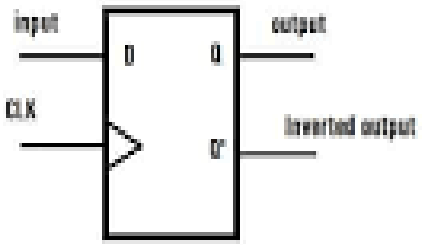
12. Classification of asynchronous sequential circuits. (MAY 2019)

Their memory elements are either un-clocked flip-flops or time-delay elements. They are similar to combinational circuits with feedback. No clock signal, hence no waiting for a clock pulse to begin processing inputs, therefore fast.

13. What are pulse mode sequential circuits? (DEC 2019)

A pulse -mode circuit is designed to respond to pulses of certain duration; the constant signals between the pulses are “null” or “spacer” signals, which do not affect the circuit's behavior. More than one input can be change at a time after stable state.

14. Model a D flip flop using Verilog. (MAY 2019)



15. Define equivalence of two states in Asynchronous sequential circuits. (DEC 2019)

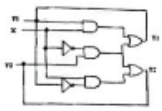
An asynchronous circuit, clockless, or self-timed circuit, is a sequential digital logic circuit which is not governed by a clock circuit or global clock signal. Instead it often uses signals that indicate completion of instructions and operations, specified by simple data transfer protocols.

16. State problems in asynchronous circuits. (SEP 2020)

- Some asynchronous circuits may require extra power for certain operations.
- More difficult to design and subject to problems like sensitivity to the relative arrival times of inputs at gates.
- Number of circuit elements (transistors) maybe double that of synchronous circuits.

PART- B (11MARKS)

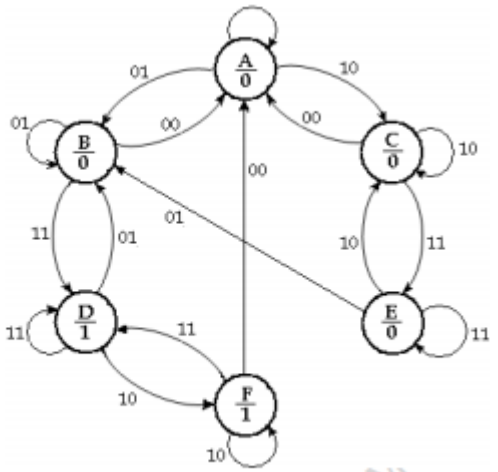
1. **Design an asynchronous sequential circuit with 2 inputs T and C. The output attains a value of 1 when T = 1 AND C moves from 1 to 0. Otherwise the output is 0 (OR) Design an asynchronous sequential circuit that the inputs $X_2$and X_1 and one output Z. The output is to remain a '0', as a long as X_1 is a '0'. The first change in X_2 that occurs while X_1 is a '1' will cause Z to be a '1'. Z is to remain a '1' until Z returns to zero. (OR)**



(May 2016/MAY 2017/DEC 2017/MAY 2018)

STEP 1:

The state diagram is given as:



State diagram

STEP 2: PRIMITIVE FLOW TABLE

	AB 00	01	11	10
A	(A), 0	B, –	–, –	C, –
B	A, –	(B), 0	D, –	–, –
C	A, –	–, –	E, –	(C), 0
D	–, –	B, –	(D), 1	F, –
E	–, –	B, –	(E), 0	C, –
F	A, –	–, –	D, –	(F), 1

REDUCEDFLOWTABLE:

	X2 X1 00	01	11	10
A	(A), 0	(A), 0	D, –	C, –
C	A, –	D, –	(C), 0	(C), 0
D	A, –	(D), 1	(D), 1	C, –

Reduced Flow table

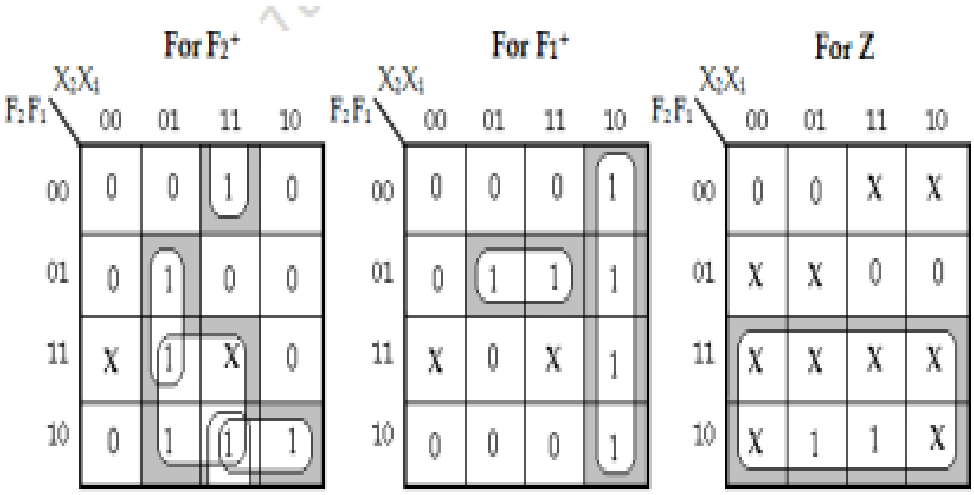
Present State	Next state for Inputs X_2X_1, Output			
F_2F_1	00	01	11	10
$S_0 \rightarrow 00$	S_0, 0	S_0, 0	S_2, –	S_1, –
$S_1 \rightarrow 01$	S_0, –	S_1, –	S_3, 0	S_3, 0
$S_2 \rightarrow 10$	S_0, –	S_3, 1	S_3, 1	S_0, –
$S_3 \rightarrow 11$	–, –	S_2, –	–, –	S_1, –

Flow table with state assignment

Substituting the binary assignment into the reduced flow table, the transition table is obtained. The output map is obtained from the reduced flow table.

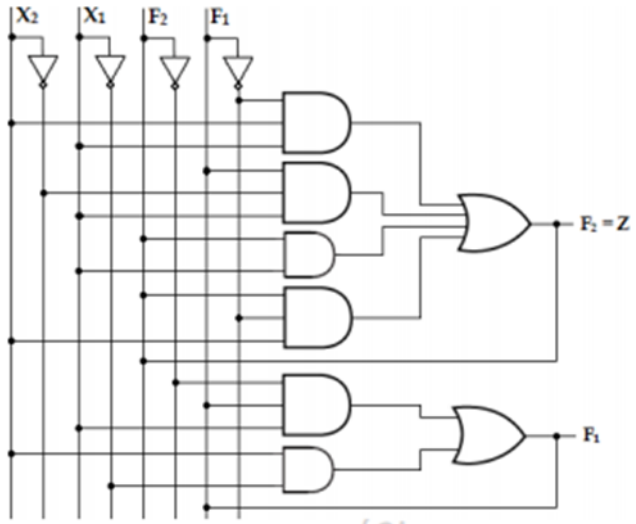
Present State	Next state for Inputs X_2X_1, Output			
F_2 F_1	00	01	11	10
0 0	00, 0	00, 0	10, –	01, –
0 1	00, –	11, –	01, 0	01, 0
1 0	00, –	10, 1	10, 1	11, –
1 1	–, –	10, –	–, –	01, –

Kmap Simplification:



$F_2^+ = \overline{F_1}X_2X_1 + F_1\overline{X_2}X_1 + F_2X_1 + F_2\overline{F_1}X_2$ $\quad F_1^+ = \overline{F_2}F_1X_1 + X_2\overline{X_1}$ $\quad Z = F_2$

Logic Diagram:



2. (a) Explain the difference between synchronous and asynchronous sequential circuits. (MAY 2016/ SEP 2020)

S.No	Synchronous sequential circuits	Asynchronous sequential circuits
1	Memory elements are clocked flip-flops	Memory elements are either unclocked flip-flops or time delay elements.
2	The change in input signals can affect memory element upon activation of clock signal.	The change in input signals can affect memory element at any instant of time.
3	The maximum operating speed of clock depends on time delays involved. Therefore synchronous circuits can operate slower than asynchronous.	Because of the absence of clock, it can operate faster than synchronous circuits.
4	Easier to design	More difficult to design

(b) Determine the sequence of internal states Y1Y2 for the following sequence of inputs x1x2: 00, 10, 11, 01, 11, 10, 00

y \ x_1x_2	00	01	11	10	00	01	11	10
0	0	0	1	0	0	1	0	1
1	0	1	1	1	1	0	1	0

Each input x1x2, a pair of values for J1K1 and J2K2 is shown. These pairs are determined for the first and the second bits in the binary code for the present and the next states for the corresponding inputs. For example, consider the present state q1q2 = 00 for the input x1x2 = 01. Then, from the encoded state table, the next states are Y1Y2 = 01. For the first pair q1Y1 = 00, from the application table for the JK-flip-flops, the inputs J1K1 = 0. For the second pair q2Y2 = 01, and the same input x1x2 = 01, the inputs J2K2 = 1. Therefore, in the excitation table, for the input x1x2 = 01 the pair 0−, 1− is written. In the same way, the complete excitation table is determined. Fig. 13.20 shows the separated tables for each input, which when considered as the Karnaugh maps with variables x1x2 for inputs and y1y2 for the present states, yield the following functions excitation functions.

J1 = x1x2,

K1 = x1x2,

J2 = x1x2y1 + x1x2y1 + x1x2y1 + x1x2y1,

K2 = x1x2y1 + x1x2y1 + x1x2y1 + x1x2y1.

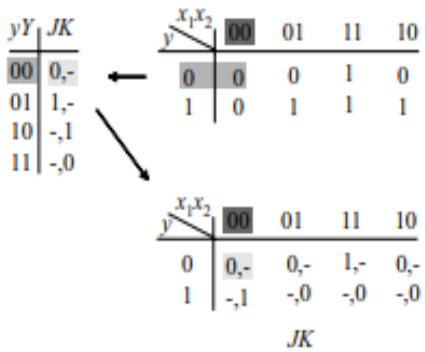
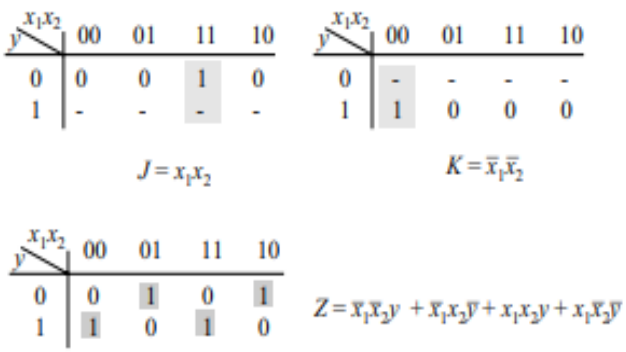


Figure 13.16. Excitation table for the binary adder with JK-flip-flop.



3. Describe the working and applications of the following memories: (a) PLD (b) EPROM. (NOV 2016)

PLDs are ICs with a large number of gates and flip flops that can be configured with basic software to perform a specific logic function or to perform the logic for a complex circuit. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.

Types of Programmable Logic:

Programmable logic devices are available in many different types. The current range of devices span from small devices capable of implementing only a handful of logic equations to huge FPGAs that can hold an entire processor core and peripherals. In addition to this incredible difference in size, there are many variations in the architecture.

Programmable logic devices can be divided into three distinct architectural groups.

- Simple Programmable Logic Devices – SPLDs
- Complex Programmable Logic Devices – CPLDs
- Field Programmable Gate Arrays – FPGAs

1. Simple Programmable Devices:

SPLDs are the simplest, smallest and least-expensive type of programmable logic device. These devices typically have logic gates laid out in arrays where the interconnection between these arrays is configurable by the user.

The term SPLD covers several types of device:

- Programmable Logic Array (PLA) – This device has both programmable AND and OR planes.
- Field Programmable Logic Array (FPLA) – Same as PLA but can be erased and reprogrammed.
- Programmable Array Logic (PAL) – This device has a programmable AND plane and a fixed OR plane.
- GAL – This device has the same logical properties as the PAL but can be erased and reprogrammed.

Advantages of Programmable Logic Devices:

Programmable logic devices offer a number of important advantages over fixed logic devices, including:

- Design Flexibility: PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.
- Improved Reliability: Lower power plus fewer interconnections and packages translate into greatly improved system reliability.

Applications of Programmable Logic Devices:

- Glue Logic: Glue logic is the Simple logic circuits used to connect together more complex circuits which are not perfectly compatible. For example, an ASIC chip may contain large functions, such as a microprocessor, memory block or communications block, which are tied together via small amounts of glue logic. At the printed circuit board (PCB) level, glue logic may be implemented with simple "jelly bean" chips ("glue chips") that contain a few gates all the way to programmable logic devices.

(B) EPROM

The Erasable Programmable Read Only Memory is a memory chip that does not lose data even when the power is switched off. This is a non-volatile memory type i.e. it retains data even when the power is switched off. Each EPROM is individually programmed by an electronic device. After that, the data can be erased by exposing the EPROM to strong ultraviolet light.An EPROM contains a transparent fused quartz window at the top of the package which allows exposure to ultraviolet light. The silicon chip is visible from this window.

Applications of EPROM

- It was assumed that EPROM was too expensive for mass production and would be used in development only. However, EPROM was found to be economical as a part for small volume production.
- On-chip EPROM was used by some microcontrollers such as Intel 8048, Freescale 68HC11, PIC microcontroller (C version) etc. These microcontrollers were available in windowed versions that were primarily used for program development and program debugging.

4. A combinational circuit is defined by the function

(a) F_1 (a, b, c) = $\sum_m$ (3,5,6,7)

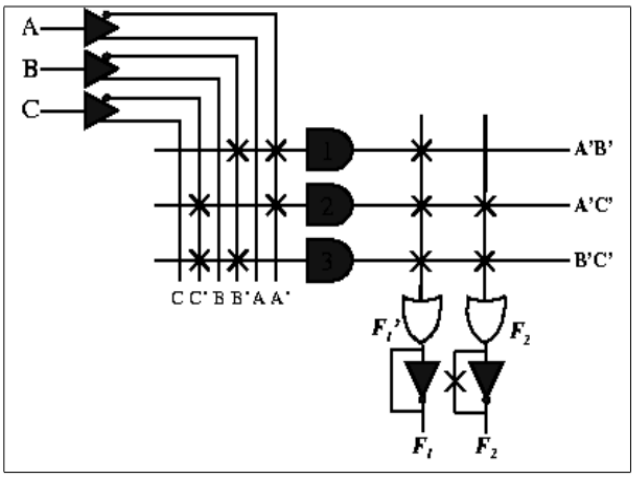
(b) F_2 (a, b, c) = $\sum_m$(0,2,4,7)

Implement the circuit with PLA. (NOV 2016)

$$f_1(a,b,c) = ab + ac + bc \qquad f_1, f_2 \rightarrow 5 \text{ terms}$$
$$\bar{f}_1(a,b,c) = \bar{a}\bar{b} + \bar{a}\bar{c} + \bar{b}\bar{c} \qquad f_1, \bar{f}_2 \rightarrow 4 \text{ terms}$$
$$f_2(a,b,c) = \bar{a}\bar{c} + \bar{b}\bar{c} \qquad \bar{f}_1, f_2 \rightarrow 3 \text{ terms}$$
$$\bar{f}_2(a,b,c) = ab + c \qquad \bar{f}_1, \bar{f}_2 \rightarrow 5 \text{ terms}$$

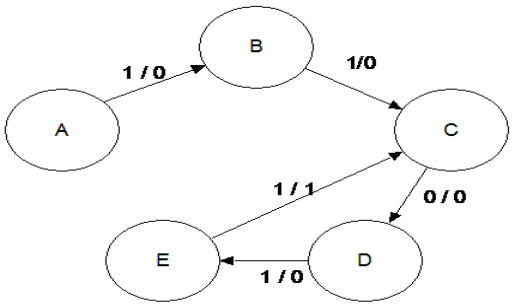
		Inputs			Outputs	
Term	Term#	a	b	c	f_1	f_2
$\bar{a}\bar{b}$	1	0	0	_	1	_
$\bar{a}\bar{c}$	2	0	_	0	1	1
$\bar{b}\bar{c}$	3	_	0	0	1	1
					C	T

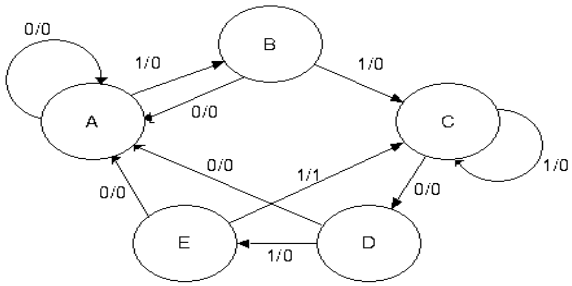
PLA FLOW DIAGRAM:



5. Using state diagram and state table, design sequence detector, which produces an output 1 every time when the sequence 11011 is detected and an output 0 at all other time. (MAY 2017)

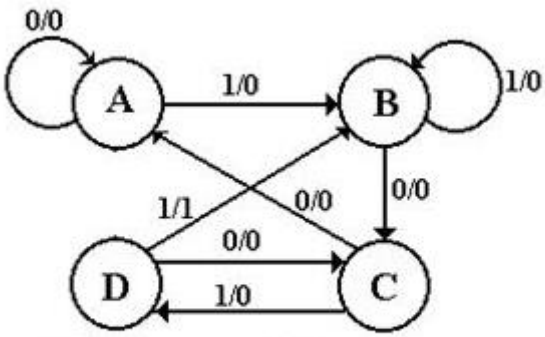
Derive the State Diagram and State Table for the Problem:





Present State	Next State / Output	
	X = 0	X = 1
A	A / 0	B / 0
B	A / 0	C / 0
C	D / 0	C / 0
D	A / 0	E / 0
E	A / 0	C / 1

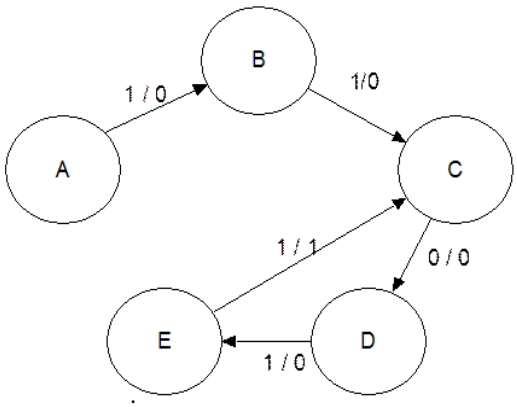
Present State		Next State / Output	
		X = 0	X = 1
	$Y_2Y_1Y_0$	$Y_2Y_1Y_0$ / Z	$Y_2Y_1Y_0$ / Z
A	0 0 0	0 0 0 / 0	0 0 1 / 0
B	0 0 1	0 0 0 / 0	0 1 1 / 0
C	0 1 1	1 0 0 / 0	0 1 1 / 0
D	1 0 0	0 0 0 / 0	1 0 1 / 0
E	1 0 1	0 0 0 / 0	0 1 1 / 1

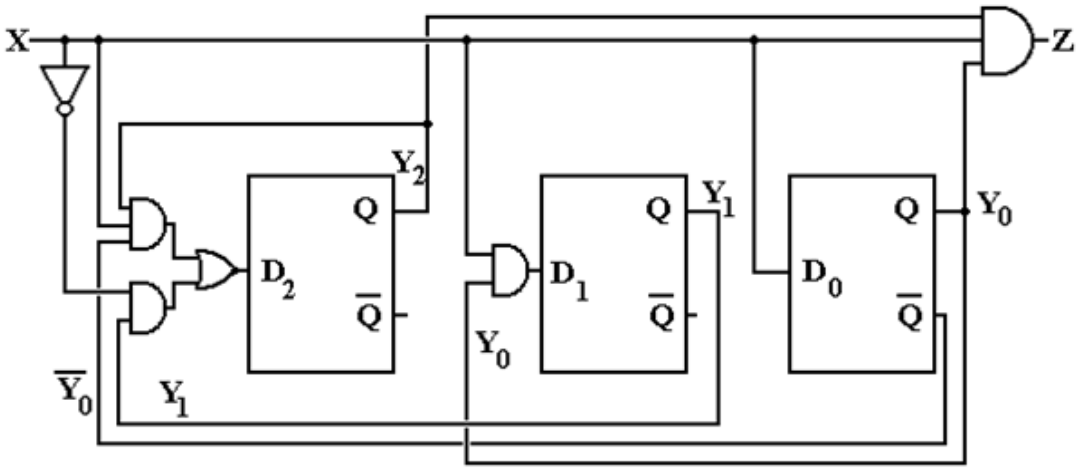


1011 Sequence Detector With Overlap

6. Design a sequence detector that detects a sequence of 1011101 in the given binary input stream in non-overlapped manner. (DEC 2017)

A	**--**	**11011**
B	**1**	**1011**
C	**11**	
D	**110**	**011**
E	**1101**	**11**
		1



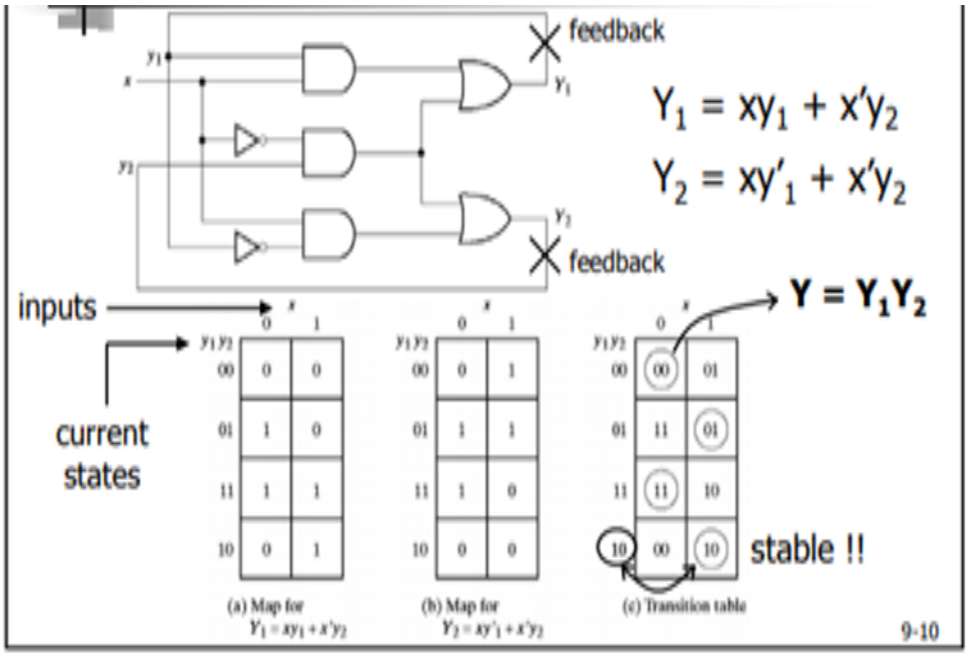


7.(a) Sketch the transitions table and state table for an asynchronous sequential circuit described by the following Boolean expressions:

y1 + x1x2 + x1y2 + x2y1,

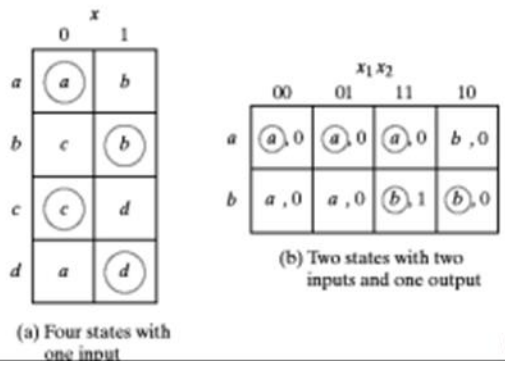
y1 + x2 + x2y1 + y2 + x1y1,

z = x2 + y1



- Similar to a transition table except the states are represented by ***letter symbols***
- Can also include the output values
- Suitable to obtain the logic diagram from it
- Primitive flow table: ***only one*** stable state in each row (ex: 9-4(a))

Equivalent to 9-3(c) if a=00, b=01, c=11, d=10



(a) Four states with one input

(b) Two states with two inputs and one output

9-12

(b) Discuss the steps involved in the designing asynchronous sequential circuit with suitable example. (MAY 2019)

There are a number of steps that must be carried out in order to minimize the circuit complexity and to produce a stable circuit without critical races. Briefly, the design steps are as follows:

- Obtain a primitive flow table from the given specification.
- Reduce the flow table by merging rows in the primitive flow table.
- Assign binary states variables to each row of the reduced flow table to obtain the transition table.
- Assign output values to the dashes associated with the unstable states to obtain the output maps.
- Simplify the Boolean functions of the excitation and output variables and draw the logic diagram.

The design process will be demonstrated by going through a specific example:

Design Example – Specification

Design a gated latch circuit with two inputs, G (gate) and D (data), and one output Q. The gated latch is a memory element that accepts the value of D when G = 1 and retains this value after G goes to 0. Once G = 0, a change in D does not change the value of the output Q.

Step 1: Primitive Flow Table

A primitive flow table is a flow table with only one stable total state in each row. The total state consists of the internal state combined with the input.

To derive the primitive flow table, first a table with all possible total states in the system is needed:

	Inputs		Output	
State	**D**	**G**	**Q**	**Comments**
a	0	1	0	$D = Q$ because $G = 1$
b	1	1	1	$D = Q$ because $G = 1$
c	0	0	0	After state *a* or *d*
d	1	0	0	After state *c*
e	1	0	1	After state *b* or *f*
f	0	0	1	After state *e*

Each row in the above table specifies a total state; the resulting primitive table for the gated latch is shown below:

States	Inputs *DG* 00	01	11	10
a	*c*, –	(*a*), 0	*b*, –	–, –
b	–, –	*a*, –	(*b*), 1	*e*, –
c	(*c*), 0	*a*, –	–, –	*d*, –
d	*c*, –	–, –	*b*, –	(*d*), 0
e	*f*, –	–, –	*b*, –	(*e*), 1
f	(*f*), 1	*a*, –	–, –	*e*, –

First, we fill in one square in each row belonging to the stable state in that row. Next recalling that both inputs are not allowed to change at the same time, we enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state. Next we find values for twomore squares in each row. The comments listed in the previous table may help in deriving the necessary information. A dash indicates don't care conditions.

Step 2: Reduction of the Primitive Flow Table

The primitive flow table can be reduced to a smaller number of rows if two or more stable states are placed in the same row of the flow table. The simplified merging rules are as follows:

Ø Two or more rows in the primitive flow table can be merged into one if there are non-conflicting states and outputs in each of the columns.

Ø Whenever, one state symbol and don't care entries are encountered in thethesamestate column, is listed in the merged row.

Ø If the state is circled in one of the rows, it is also circled in the merged row.

Ø The output state is included with each stable state in the merged row.

Now apply these rules to the primitive flow table shown previously. To see how this is done the primitive flow table is separated into two parts of three rows each:

States \ DG	00	01	11	10
a	c, –	a, 0	b, –	–, –
c	c, 0	a, –	–, –	d, –
d	c, –	–, –	b, –	d, 0

States \ DG	00	01	11	10
b	–, –	a, –	b, 1	e, –
e	f, –	–, –	b, –	e, 1
f	f, 1	a, –	–, –	e, –

(a) States that are candidates for merging

States \ DG	00	01	11	10
a, c, d	c, 0	a, 0	b, –	d, 0
b, e, f	f, 1	a, –	b, 1	e, 1

States \ DG	00	01	11	10
a	a, 0	a, 0	b, –	a, 0
b	b, 1	a, –	b, 1	b, 1

(b) Reduced table (two alternatives)

Step 3: Transition table and logic diagram

The assignment of distinct binary value to each state converts the flow table into a transition table. In the general case, a binary state assignment must be made to ensure that the circuit will be free of critical races. There can be no critical races in a two-row flow table; therefore, we can finish the design of the gated latch. Assigning 0 to state a and 1 to state b in the reduced flow table, we obtain the transition table. The transition table is, in effect, a map for the excitation variable Y. The simplified Boolean function for Y is then obtained from the map.

$$Y = DG + \overline{G}y$$

There are two don't-care outputs in the final reduced flow table. Alternatively, if we replace the don'tcare by 1 when y = 1 and DG= 01, the map reduces to Q = Y. If we assign the other possible values to the don't -care outputs. We can make output Q equal to y.

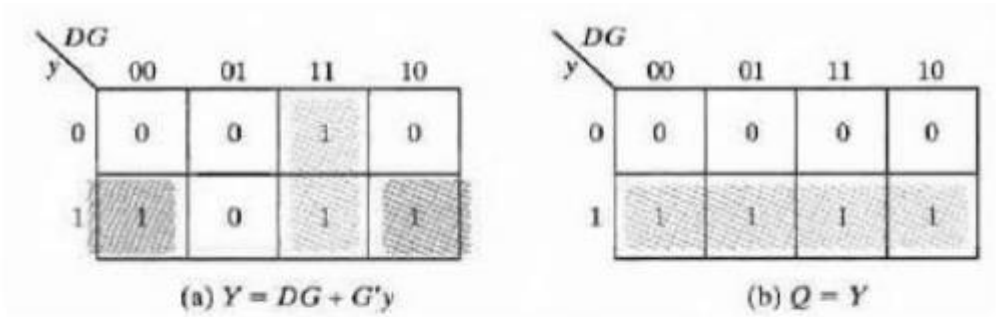
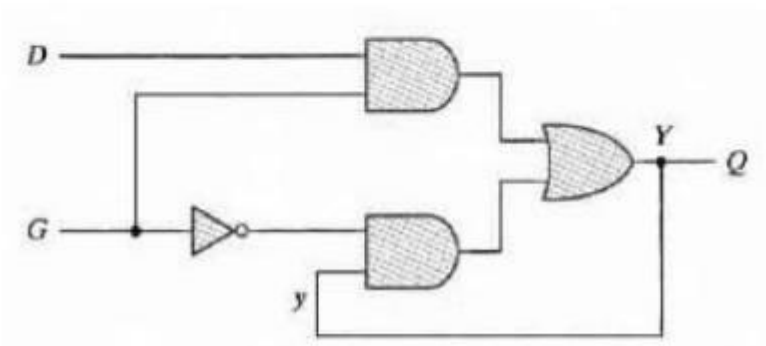


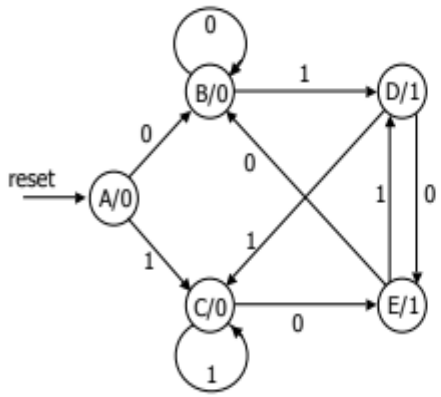
Fig: Gated- latch logic diagram



8. Describe the design procedure for Mealy and Moore machine. Give the design procedure for asynchronous sequential circuit. (DEC 2018)

Specifying Outputs for a Moore Machine

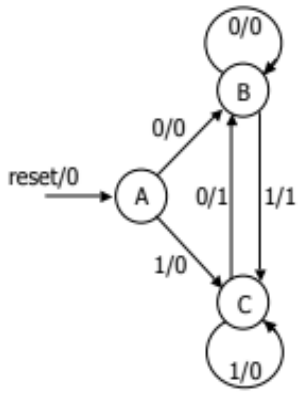
- Output is only function of state
  - Specify in state bubble in state diagram
  - Example: sequence detector for 01 or 10



reset	input	current state	next state	output
1	–	–	A	
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

Specifying Outputs for a Mealy Machine

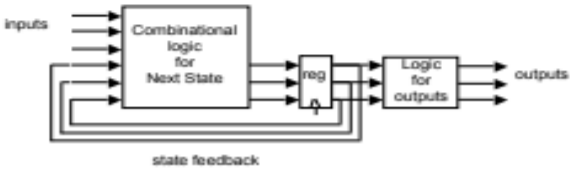
- Output is function of state and inputs
  - Specify output on transition arc between states
  - Example: sequence detector for 01 or 10

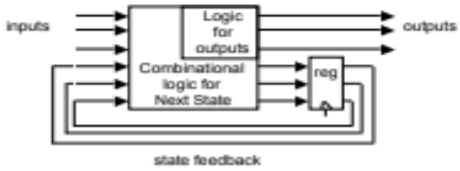


reset	input	current state	next state	output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Comparison of Mealy and Moore Machines

- Mealy Machines tend to have less states
  - Different outputs on arcs (n^2) rather than states (n)
- Moore Machines are safer to use
  - Outputs change at clock edge (always one cycle later)
  - In Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback
- Mealy Machines react faster to inputs
  - React in same cycle – don't need to wait for clock
  - In Moore machines, more logic may be necessary to decode state into outputs – more gate delays after





9. Explain the various types of hazards in sequential circuit design and the method to eliminate them.Give suitable examples. (DEC 2018/ MAY 2019)

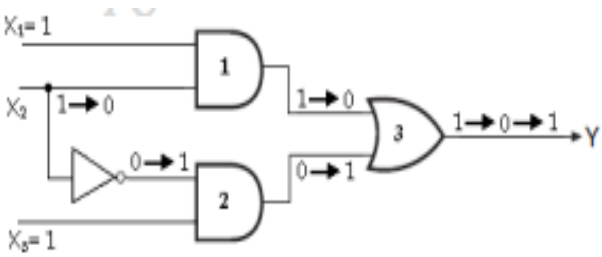
Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays. Hazards occur in combinational circuits, where they may cause a temporary false-output value. When this condition occurs in asynchronous sequential circuits, it may result in a transition to a wrong stable state. Hazards in Combinational Circuits: A hazard is a condition where a single variable change produces a momentary output change when no output change should occur.

Types of Hazards: Static hazard and Dynamic hazard

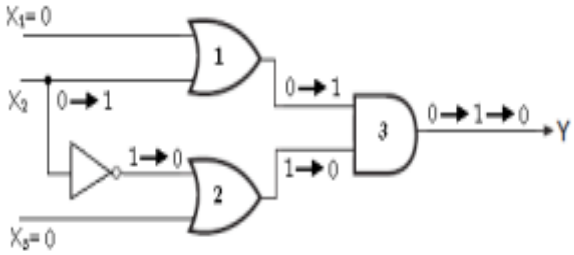
Static Hazard: In digital systems, there are only two possible outputs, a ‘0’ or a ‘1’. The hazard may produce a wrong ‘0’ or a wrong ‘1’. Based on these observations, there are three types,

Static- 0 hazard, Static- 1 hazard, Static- 0 hazard: When the output of the circuit is to remain at 0, and a momentary 1 output is possible during the transmission between the two then the hazard is called a static 0-hazard.

Static- 1 hazard: When the output of the circuit is to remain at 1, and a momentary 0 output is possible during the transmission between the two inputs, then the hazard is called a static 1- hazard.

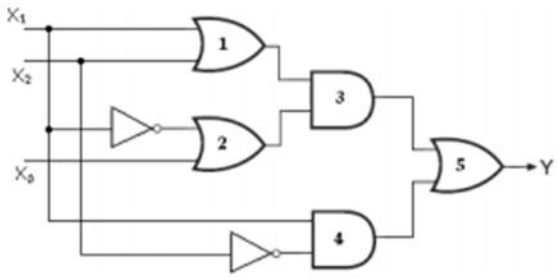


Circuit with static-1 hazard

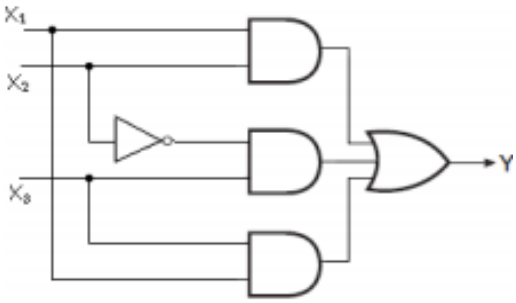


Circuit with static-0 hazard

Dynamic Hazard: A dynamic hazard is defined as a transient change occurring three or more times at an output terminal of a logic network when the output is supposed to change only once during a transition between two input states differing in the value of one variable. Now consider the input states X1X2X3= 000 and X1X2X3= 100. For the first input state, the steady state output is 0; while for the second input state, the steady state output is 1. To facilitate the discussion of the transient behavior of this network, assume there are no propagation delays through gates G3 and G5 and that the propagation delays of the other three gates are such that G1 can switch faster than G2 and G2 can switch faster than G4.



Overcoming HAZARD:



Hazard-free Circuit

A hazard can be detected by inspection of the map of the particular circuit. To illustrate, consider the map in the circuit with static 0-hazard, which is a plot of the function implemented. The change in X2 from 1 to 0 moves the circuit from minterm 111 to minterm 101. The hazard exists because the change in input results in a different product term covering the two minterrns. Maps demonstrating a Hazard and its Removal The minterm 111 is covered by the product term implemented in gate 1 and minterm 101 is covered by the product term implemented in gate 2. Whenever the circuit must move from one product term to another, there is a possibility of a momentary interval when neither term is equal to 1, giving rise to an undesirable 0 output. The remedy for eliminating a hazard is to enclose the two minterms in question with another product term that overlaps both groupings.

10. Define sequential circuit and illustrate operating mode sequential circuit model. (DEC 2019)

A Sequential logic circuits is a form of the binary circuit; its design employs one or more inputs and one or more outputs, whose states are related to some definite rules that depend on previous states. Both the inputs and outputs can reach either of the two states: logic 0

Types of Sequential Circuits

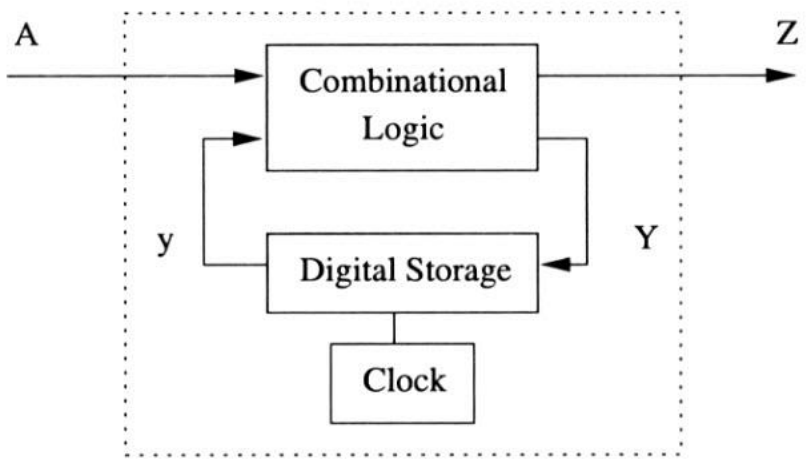
The sequentialcircuits are classified into two types

- Synchronous Circuit
- Asynchronous Circuit

In synchronous sequential circuits, the state of device changes at discrete times in response to a clock signal. In asynchronous circuits, the state of the device changes in response to changing inputs.

Synchronous Circuits

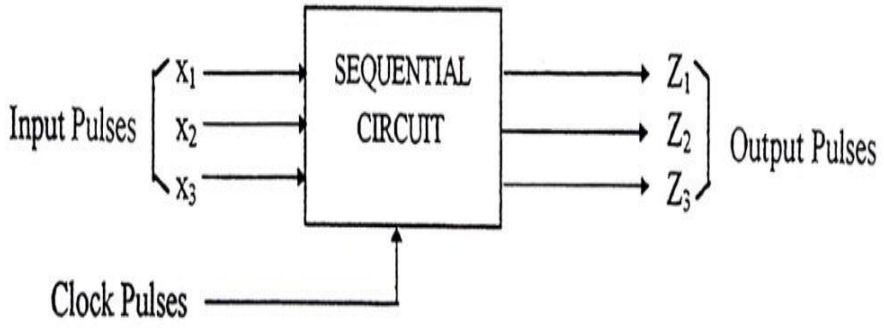
In synchronous circuits, the inputs are pulses with certain restrictions on pulse width and propagation delay. Thus synchronous circuits can be divided into clocked and un-clocked or pulsed sequential circuits.



S

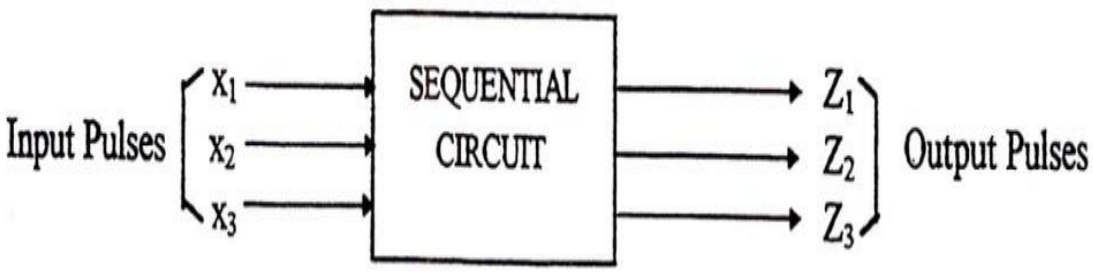
Clocked Sequential Circuit

The clocked sequential circuits have flip-flops or gated latches for its memory elements. There is a periodic clock connected to the clock inputs of all the memory elements of the circuit to synchronize all the internal changes of state. Hence the operation of the circuit is controlled and synchronized by the periodic pulse of the clock.



Unclocked Sequential Circuit

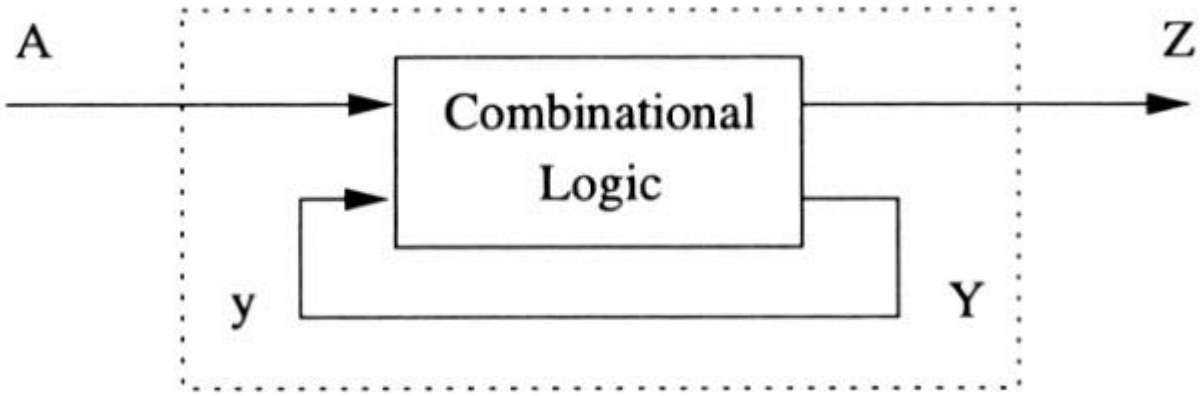
In an unclocked sequential circuit requires two consecutive transitions between 0 and 1 to alternate the state of the circuit. An unclocked mode circuit is designed to respond to pulses of certain durations which do not affect the circuit's behaviour.



The synchronous logic circuit is very simple. The logic gates which perform the operations on the data, require a finite amount of time to respond to the changes in the input.

Asynchronous Circuits

An asynchronous circuit does not have a clock signal to synchronize its internal changes of the state. Hence the state change occurs in direct response to changes that occur in primary input lines. An asynchronous circuit does not require the precise timing control from flip-flops.



Asynchronous Circuit

Asynchronous logic is more difficult to design and it has some problems compared to synchronous logic. The main problem is that the digital memory is sensitive to the order that their input signals arrive them, like, if two signals arrive at a flip-flop at the same time, which state the circuit goes into can depend on which signal gets to the logic gate first.

Asynchronous circuits are used in critical parts of synchronous systems where the speed of the system is a priority, like as in microprocessors and digital signal processing circuits.

DIGITAL LOGIC AND CIRCUIT THEORY

UNIT V

PART – A (2 MARKS)

1. Difference between RAM&ROM. [MAY 2016, NOV2017, SEP 2020]

- RAM stores new information for later use. The process of storing new information into memory is referred to as a memory write operation. The process of transferring the stored information out of memory is referred to as a memory read operation.
- ROM can perform only the read operation. This means that suitable binary information is already stored inside memory and can be retrieved or read at any time. However, that information cannot be altered by writing.

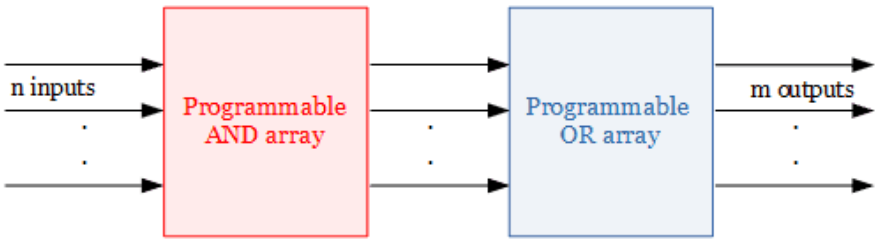
2. Difference between PAL&PLA. [MAY 2016, NOV 2018]

- The PLA is similar in concept to the PROM, except that the PLA does not provide full decoding of the variables and does not generate all the minterms. The decoder is replaced by an array of AND gates that can be programmed to generate any product term of the input variables. The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.
- The PAL is a programmable logic device with a fixed OR array and a programmable AND array. Because only the AND gates are programmable, the PAL is easier to program than, but is not as flexible as, the PLA.

3. What are the two types of programmable logic devices? Give example for each type. [MAY 2017, NOV 2016]

- Programmable Logic Array (PLA)
  Eg: PROM
- Programmable Array Logic (PAL)
  Eg: SRAM

4.Draw the block diagram of PLA.[MAY 2018]



5. What is the difference between PROM&EPROM. [MAY 2018]

- A PROM uses some type of fusing process to store bits, in which a memory link is burned open or left intact to represent a 0 or a 1. The fusing process is irreversible; once a PROM is programmed, it cannot be changed.
- An EPROM uses an NMOSFET array with an isolated-gate structure. The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time.

6. What is the advantages of PLA over ROM [NOV2018]

1. There is no needed for the time-consuming logic design of random-logic gate networks and even more time-consuming layout.
2. Design checking is easy, and design change is also easy.
3. Layout is far simpler than that for random-logic gate networks, and thus is far less time-consuming.
4. When new IC fabrication technology is introduced, we can use previous design information with ease but without change, making adoption of the new technology quick and easy.

7. Compare static RAM and dynamic RAM? [NOV 2019]

S.No	Static RAM	Dynamic RAM
1	It contains less memory cells per unit area.	It contains more memory cells per unit area.
2	Its access time is less, hence faster memories.	Its access time is greater than static RAM
3	It consists of number of flip-flops. Each flip-flop stores one bit.	It stores the data as a charge on the capacitor. It consists of MOSFET and capacitor for each cell.
4	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors every time after every few milliseconds. Extra hardware is required to control refreshing.
5	Cost is more	Cost is less.

8.Define synchronous sequential circuit. [NOV 2016]

Synchronous sequential circuit is defined as **digital sequential circuits** in which the feedback to the input for next output generation is governed by clock signals. On other hand Asynchronous sequential circuits are digital sequential circuits in which the feedback to the input for next output generation is not governed by clock signals.

<u>PART – B (11 MARKS)</u>

1. **Discuss on the concept of working and application: -**
   [MAY 2016, MAY 2017, NOV 2018]
   a) **ROM**
   b) **EPROM**
   c) **PLA**

a) ROM

ROM (read-only memory) is a type of memory in which data are stored permanently or semi permanently. Data can be read from a ROM, but there is no write operation as in the RAM. The ROM, like the RAM, is a random-access memory but the term RAM traditionally means a random-access read/write memory. Because ROMs retain stored data even if power is turned off, they are nonvolatile memories.

A ROM contains permanently or semi-permanently stored data, which can be read from the memory but either cannot be changed at all or cannot be changed without specialization equipment. A ROM stores data that are used repeatedly in system applications, such as tables, conversions, or programmed instructions for system initialization and operation. ROMs retain stored data when the power is OFF and are therefore nonvolatile memories.

b) EPROM

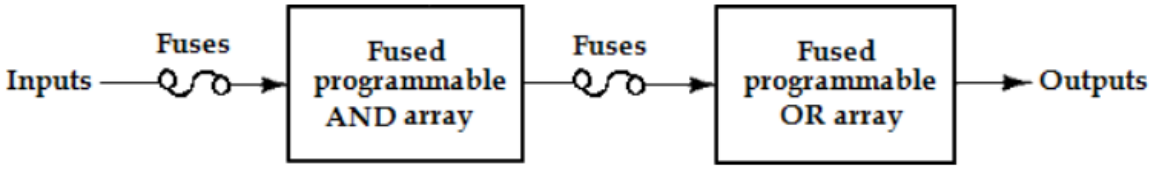
An EPROM is an erasable PROM. Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased first. An EPROM uses an NMOSFET array with an isolated-gate structure. The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time. The data bits in this type of array are represented by the presence or absence of a stored gate charge. Erasure of a data bit is a process that removes the gate charge.

Two basic types of erasable PROMs are the ultraviolet erasable PROM (UV EPROM) and the electrically erasable PROM (EEPROM).

UV EPROM: You can recognize the UV EPROM device by the transparent quartz lid on the package, as shown in Figure below. The isolated gate in the FET of an ultraviolet EPROM is "floating" within an oxide insulating material. The programming process causes electrons to be removed from the floating gate. Erasure is done by exposure of the memory array chip to high-intensity ultraviolet radiation through the quartz window on top of the package. The positive charge stored on the gate is neutralized after several minutes to an hour of exposure time. In EPROM‘s, it is not possible to erase selective information, when erased the entire information is lost. The chip can be reprogrammed. It is ideally suited for product development, college laboratories, etc.

c) PLA

A PLA consists of a programmable AND array and a programmable OR array. The product terms in the AND array may be shared by any OR gate to provide the required sum of product implementation. The PLA is developed to overcome some of the limitations of the PROM. The PLA is also called an FPLA (Field Programmable Logic Array) because the user in the field, not the manufacturer, programs it.



Programmable Logic Array (PLA)

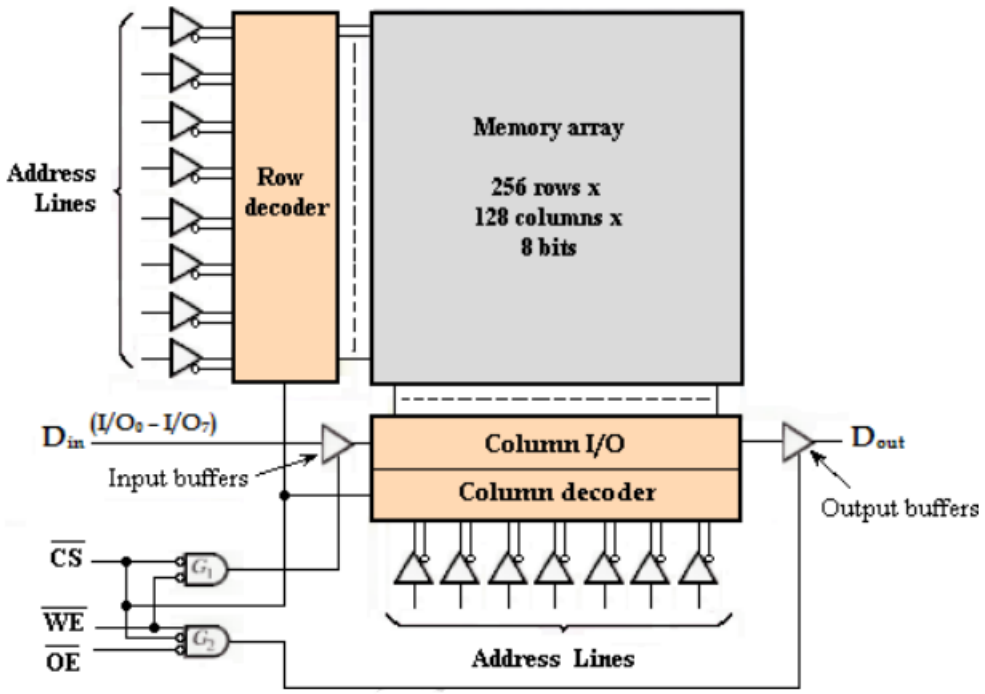
2. Explain about RAM with neat diagram [MAY 2016, MAY 2018, SEP 2020]

RAMs are read/write memories in which data can be written into or read from any selected address in any sequence. When a data unit is written into a given address in the RAM, the data unit previously stored at that address is replaced by the new data unit. When a data unit is read from a given address in the RAM, the data unit remains stored and is not erased by the read operation. This nondestructive read operation can be viewed as copying the content of an address while leaving the content intact. A RAM is typically used for short-term data storage because it cannot retain stored data when power is turned off. The two categories of RAM are the static RAM (SRAM) and the dynamic RAM (DRAM).

Static RAMs generally use flip-flops as storage elements and can therefore store data indefinitely as long as dc power is applied. Dynamic RAMs use capacitors as storage elements and cannot retain data very long without the capacitors being recharged by a process called refreshing. Both SRAMs and DRAMs will lose stored data when dc power is removed and, therefore, are classified as volatile memories.

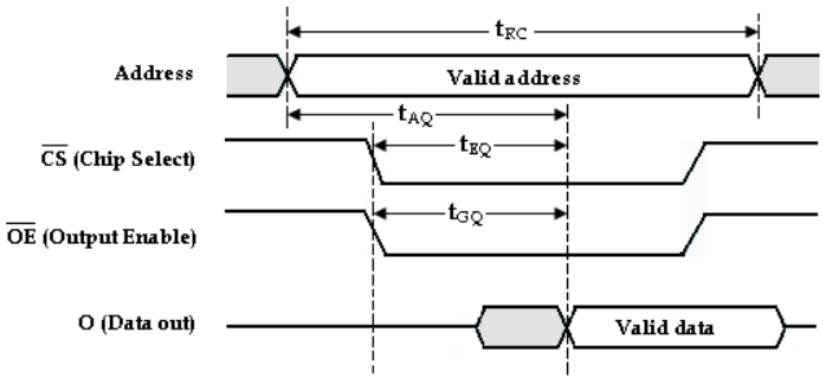
Data can be read much faster from SRAMs than from DRAMs. However, DRAMs can store much more data than SRAMs for a given physical size and cost because the

DRAM cell is much simpler, and more cells can be crammed into a given chip area than in the SRAM.



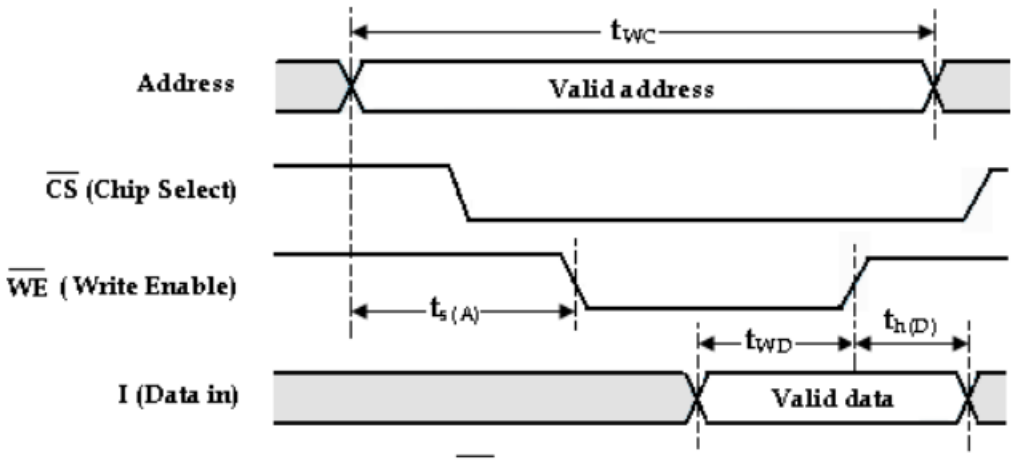
Memory block diagram

Read and Write Cycles: For the read cycle shown in part (a), a valid address code is applied to the address lines for a specified time interval called the read cycle time, tWC. Next, the chip select (CS) and the output enable (DE) inputs go LOW. One time interval after the DE input goes LOW; a valid data byte from the selected address appears on the data lines. This time interval is called the output enable access time, tGQ. Two other access times for the read cycle are the address access time, tAQ, measured from the beginning of a valid address to the appearance of valid data on the data lines and the chip enable access time, tEQ, measured from the HIGH-to-LOW transition of CS to the appearance of valid data on the data lines. During each read cycle, one unit of data, a byte in this case is read from the memory.



(a) Read Cycle ($\overline{WE}$ HIGH)

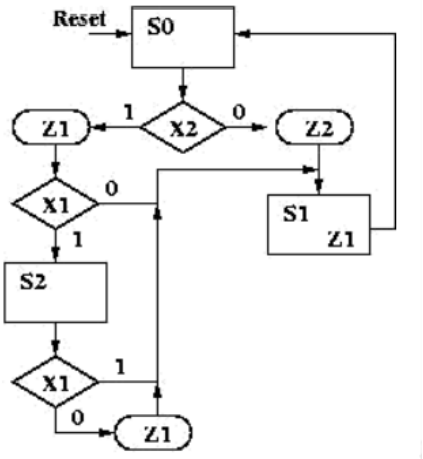
For the write cycle shown in Figure (b), a valid address code is applied to the address lines for a specified time interval called the write cycle time, tWE . Next, the chip select (CS) and the write enable (WE) in puts go LOW. The required time interval from the beginning of a valid address until the WE input goes LOW is called the address setup time, t s(A). The time that the WE input must be LOW is the write pulse width. The time that the input WE must remain LOW after valid data are applied to the data inputs is designated t WD; the time that the valid input data must remain on the data lines after the WE input goes HIGH is the data hold time, t h(D). During each write cycle, one unit of data is written into the memory.



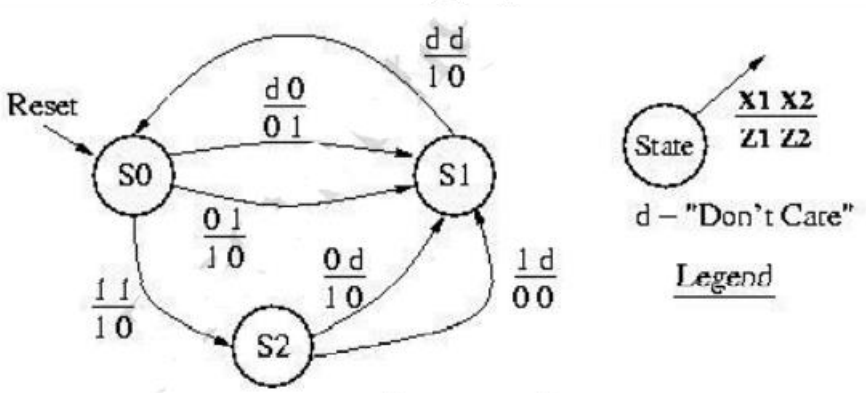
(b) Write cycle ($\overline{WE}$ LOW)

3.Design an asynchronous sequential circuit with inputs X1 and X2, and outputs Z1 and Z2. In state S0 the outputs are immediately dependent on the input. In state S1, output Z1 is always asserted. In state S2, output Z1 is dependent on input X1 but Z2 is not asserted. [NOV 2016]

CIRCUITE DIAGRAM: -



STATE DIAGRAM: -



Corresponding state diagram

4.What are hazards? Classify hazards explain them with suitable examples.

[NOV 2016]

Hazards are the unwanted switching transient that may appear at the output of the digital circuit. Such hazards may result in a malfunction in the output of the circuit.

The propagation delay associated with the logic gates in the circuit is the main cause of hazard. Hazards occur in combinational circuits, which will cause faulty operation. Hazards will also occur in asynchronous sequential circuits where logic gates are used, affecting the stable state.

There are two main types of hazard: Static Hazard and Dynamic Hazard.

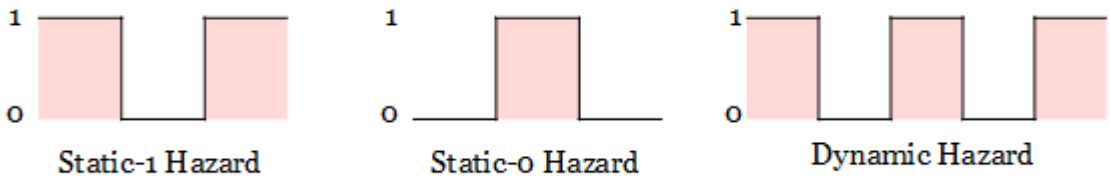
- **Static Hazard**

When the input to the logic circuit changes, the output has to remain at a particular logic, but instead it may change its value momentarily. If this happens in the logic circuit, then there exists a **static hazard**.

The static hazard can be classified as a static-0 hazard and static-1 hazard. Instead of remaining at logic 1, the output goes to logic 0 momentarily, then it is a **static-1 hazard**. Similarly, if the output monetarily goes to logic 1, instead of remaining at 0, it is a **static-0 hazard**.

- **Dynamic Hazard**

A hazard is said to be **dynamic** if the output changes two or more times when it should change from 1 to 0 or from 0 to 1.



Hazards are the temporary problem, as the logic circuit will gradually settle to the desired function. But, despite logic arriving at correct output, it is imperative that hazards be removed as they can have an effect on the other connected systems.

5. Write a short note on

(a) Static RAM.

(b) MOSFET RAM. [MAY 2017, NOV 2017, NOV 2019, SEP 2020]

a) Static RAM

Static RAM (SRAM) consists of flip-flops, a bistable circuit composed of four to six transistors. Once a flip-flop stores a bit, it keeps that value until the opposite value is stored in it. SRAM gives fast access to data, but it is physically relatively large. It is

used primarily for small amounts of memory called registers in a computer's central processing unit (CPU) and for fast "cache" memory.

SRAM may be integrated as RAM or cache memory in micro-controllers (usually from around 32 bytes up to 128 kilobytes), as the primary caches in powerful microprocessors, such as the x86 family, and many others (from 8 KB, up to many megabytes), to store the registers and parts of the state-machines used in some microprocessors (see register file), on application-specific ICs, or ASICs (usually in the order of kilobytes) and in Field Programmable Gate Array and Complex Programmable Logic Device.

b) MOSFET RAM

The MOSFET stands for METAL OXIDE SEMICONDUCTOR FIELD EFFECT TRANSISTOR. In MOSFET, the MOS part is related to the structure of the transistor, while the FET part is related to how it works. It is also known as IGFET (Insulated Gate Field Effect Transistor). The following image we have shown is a practical MOSFET. But in the digital world, the size of MOSFET is too small (in nm) that billions of them can be fabricated on a single chip.

MOSFETs combine together to form the basic memory element of a static RAM. At the lowest level MOSFETs are interconnected to form logic gates. At the next level, the gates are combined to form processing units that perform thousands of logical and arithmetical operations.

6. a) Implement a combinational circuit in PLA for the following Boolean function.

F1 (A, B, C) = ∑m (0, 1, 2, 4) [NOV 2017,MAY 2019]

F2 (A, B, C) = ∑m (0, 5, 6, 7)

b) Implement the following Boolean expressions in PROM.

F1 (A, B, C) = ∑m (0, 1, 3, 5, 7) [NOV 2017]

F2 (A, B, C) = ∑m (1, 2, 5, 6)

Sol :-

a)

Step 1: Truth table for the given functions

A	B	C	F_1	F_2
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Step 2: K-map Simplification

A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

$F_1 = A'B' + A'C' + B'C'$

A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$F_2 = AC + AB + A'B'C'$

With this simplification, total number of product term is 6. But we require only 4 product terms. Therefore find out F_1' and F_2'.

A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

$F_1' = AC + BC + AB$

A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$F_2' = A'C + A'B + A'B'C'$

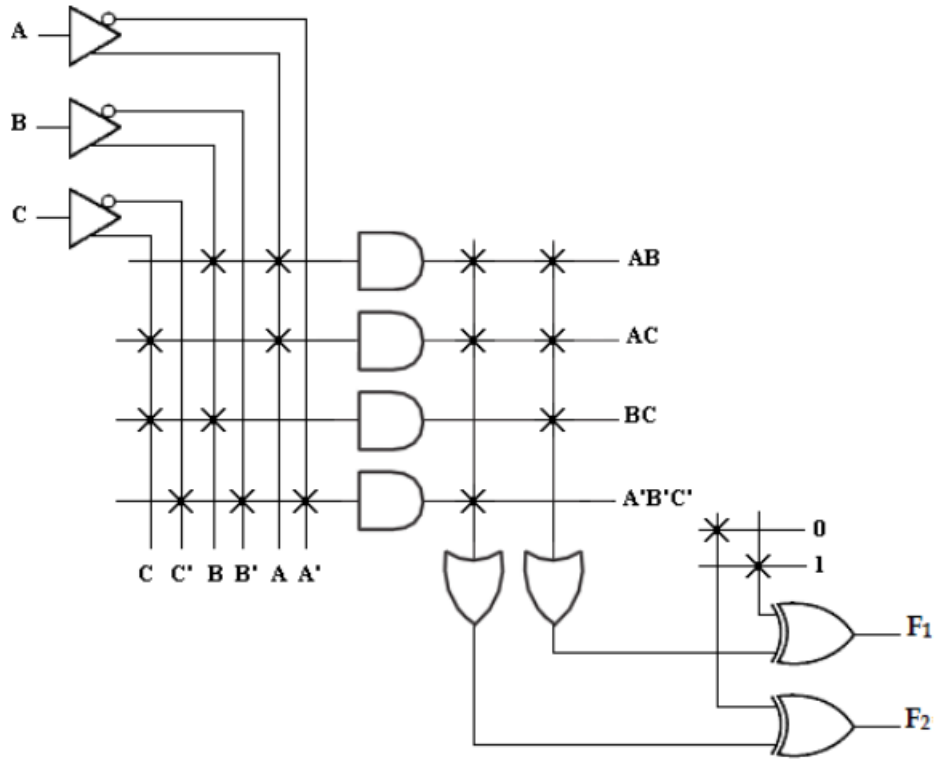
Now select, F_1' and F_2, the product terms are AC, AB, BC and A'B'C'

Step 3: PLA Program table:

	Product term	Inputs			Outputs	
		A	B	C	F_1 (C)	F_2 (T)
AB	1	1	1	-	1	1
AC	2	1	-	1	1	1
BC	3	-	1	1	1	-
A'B'C'	4	0	0	0	-	1

In the PLA program table, first column lists the product terms numerically as 1, 2, 3, and 5. The second column (Inputs) specifies the required paths between the AND gates and the inputs. For each product term, the inputs are marked with 1, 0, or - (dash). If a variable in the product form appears in its normal form, the corresponding input variable is marked with a 1. If it appears complemented, the corresponding input variable is marked with a 0. If the variable is absent in the product term, it is marked with a dash (-). The third column (output) specifies the path between the AND gates and the OR gates. The output variables are marked with 1's for all those product terms that formulate the required function.

Step 4: PLA Diagram



The PLA diagram uses the array logic symbols for complex symbols. Each input and its complement is connected to the inputs of each AND gate as indicated by the intersections between the vertical and horizontal lines. The output of the AND gate are connected to the inputs of each OR gate. The output of the OR gate goes to an EX-OR gate where the other input can be programmed to receive a signal equal to either logic 1 or 0. Programmable Logic Devices, Memory 5.32 The output is inverted when the EX-OR input is connected to 1 ie., (x ⊕1= x'). The output does not change when the EX-OR input is connected to 0 ie., (x ⊕0= x).

b) F1 (A, B, C) = ∑m (0, 1, 2, 4)

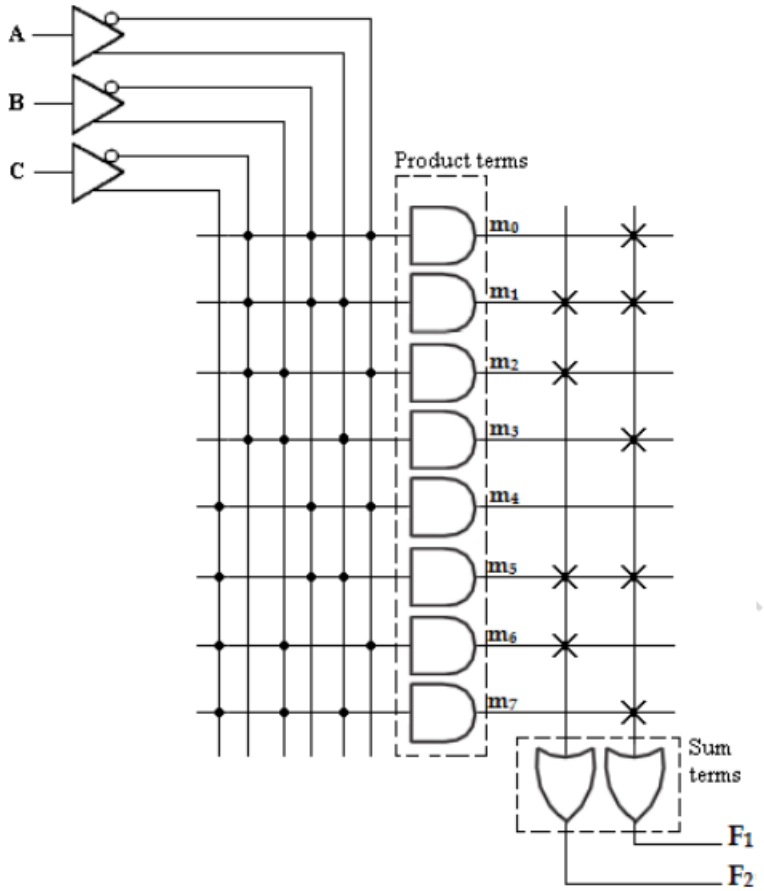
F2 (A, B, C) = ∑m (0, 5, 6, 7)

Sol: -

Step1: Truth table for the given function

A	B	C	F_1	F_2
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	1	0

Step 2: PROM diagram



7.What are transition and flow table. Give example. [NOV 2018].

Transition table :-

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a state table (sometimes called a transition table). The state table for the circuit. The table consists of four sections labeled present state, input, next state, and output . The present-state section shows the states of flip-flops A and B at any given time t . The input section gives a value of x for each possible present state. The next-state section shows the states of the flip-flops one clock cycle later, at time t + 1. The output section gives the value of y at time t for each present state and input condition. The derivation of a state table requires listing all possible binary combinations of present states and inputs. In this case, we have eight binary combinations from 000 to 111. The next-state values are then determined from the logic diagram or from the state equations. The next state of flip-flop A must satisfy the state equation A(t + 1) = Ax + Bx

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Eg :-Transition Table

The next-state section in the state table under column A has three 1's where the present state of A and input x are both equal to 1 or the present state of B and input x are both equal to 1. Similarly, the next state of flip-flop B is derived from the state equation B (t + 1) = A'x and is equal to 1 when the present state of A is 0 and input x is equal to 1. The output column is derived from the output equation y = Ax' + Bx'

Flow table :-

A primitive flow table is a flow table with only one stable total state in each row. The total state consists of the internal state combined with the input.

To derive the primitive flow table, first a table with all possible total states in the system is needed:

	Inputs		Output	
State	**D**	**G**	**Q**	**Comments**
a	0	1	0	$D = Q$ because $G = 1$
b	1	1	1	$D = Q$ because $G = 1$
c	0	0	0	After state *a* or *d*
d	1	0	0	After state *c*
e	1	0	1	After state *b* or *f*
f	0	0	1	After state *e*

Each row in the above table specifies a total state; the resulting primitive table for the gated latch is shown below:

States	Inputs *DG* 00	01	11	10
a	*c*, –	(*a*), 0	*b*, –	–, –
b	–, –	*a*, –	(*b*), 1	*e*, –
c	(*c*), 0	*a*, –	–, –	*d*, –
d	*c*, –	–, –	*b*, –	(*d*), 0
e	*f*, –	–, –	*b*, –	(*e*), 1
f	(*f*), 1	*a*, –	–, –	*e*, –

First, we fill in one square in each row belonging to the stable state in that row. Next recalling that both inputs are not allowed to change at the same time, we enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state. Next we find values for two

more squares in each row. The comments listed in the previous table may help in deriving the necessary information. **A dash indicates don't care conditions.**

- Ø Two or more rows in the primitive flow table can be merged into one if there are non-conflicting states and outputs in each of the columns.
- **Ø Whenever, one state symbol and don't care entries are encountered in the**the**same**state **column,** is listed in the merged row.
- Ø If the state is circled in one of the rows, it is also circled in the merged row.
- Ø The output state is included with each stable state in the merged row.

Now apply these rules to the primitive flow table shown previously. To see how this is done the primitive flow table is separated into two parts of three rows each:

States \ DG	00	01	11	10
a	c, –	a, 0	b, –	–, –
c	c, 0	a, –	–, –	d, –
d	c, –	–, –	b, –	d, 0

States \ DG	00	01	11	10
b	–, –	a, –	b, 1	e, –
e	f, –	–, –	b, –	e, 1
f	f, 1	a, –	–, –	e, –

(a) States that are candidates for merging

States \ DG	00	01	11	10
a, c, d	c, 0	a, 0	b, –	d, 0
b, e, f	f, 1	a, –	b, 1	e, 1

States \ DG	00	01	11	10
a	a, 0	a, 0	b, –	a, 0
b	b, 1	a, –	b, 1	b, 1

(b) Reduced table (two alternatives)

The assignment of distinct binary value to each state converts the flow table into a transition table. In the general case, a binary state assignment must be made to ensure that the circuit will be free of critical races. There can be no critical races in a two-row flow table; therefore,

we can finish the design of the gated latch. Assigning 0 to state **a** and 1 to state **b** in the reduced flow table, we obtain the transition table. The transition table is, in effect, a map for the excitation variable Y. The simplified Boolean function for Y is then obtained from the map.

$$Y = DG + \overline{G}y$$

There are two don't-care outputs in the final reduced flow table. Alternatively, if we replace the don'tcare by 1 when y = 1 and DG= 01, the map reduces to Q = Y. If we assign the other possible values to the don't -care outputs. We can make output Q equal to y.

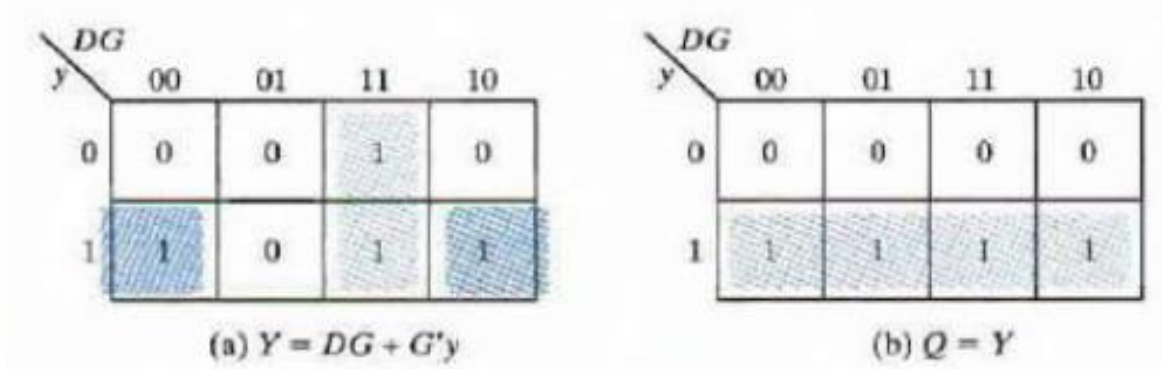
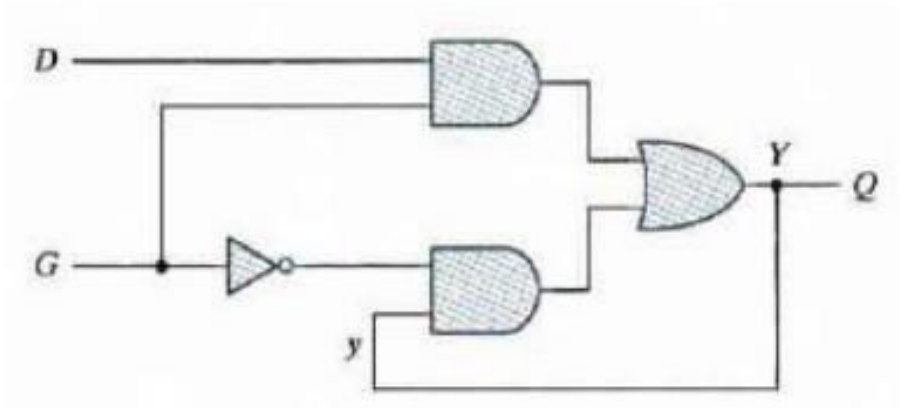


Fig: Gated- latch logic diagram



The stable states in a flow table have specific output values associated with them. The unstable states have unspecified output entriesdesignated by a dash. The output values for the unstable states must be chosen so that no momentary false outputs occur when the circuit switches between stable states. This means that if an output variable is not supposed to change as the result of a transition. then an unstable state that is a transient state between two stable states must have the same output value as the stable states.

a	a, 0	b, –
b	c, –	b, 0
c	c, 1	d, –
d	a, –	d, 1

(a) Flow table

0	0
X	0
1	1
X	1

(b) Output assignment

Fig: Assigning output values to unstable states

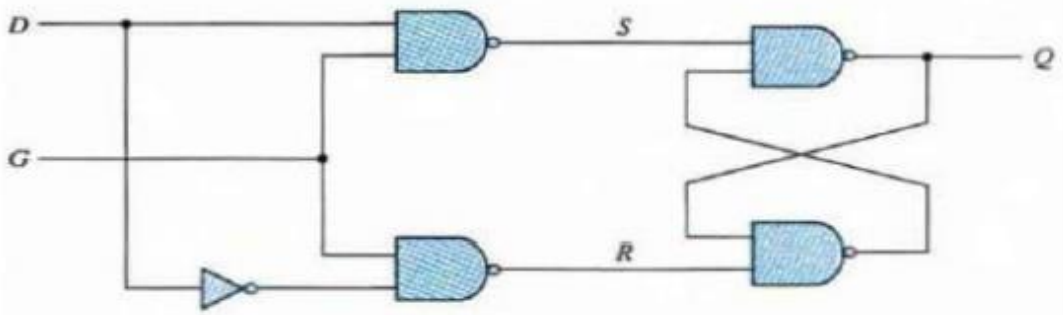
y \ DG	00	01	11	10
0	0	0	1	0
1	X	0	X	X

(a) $S = DG$

y \ DG	00	01	11	10
0	X	X	0	X
1	0	1	0	0

$R = D'G$

(a) Maps for S and R



(b) Logic diagram

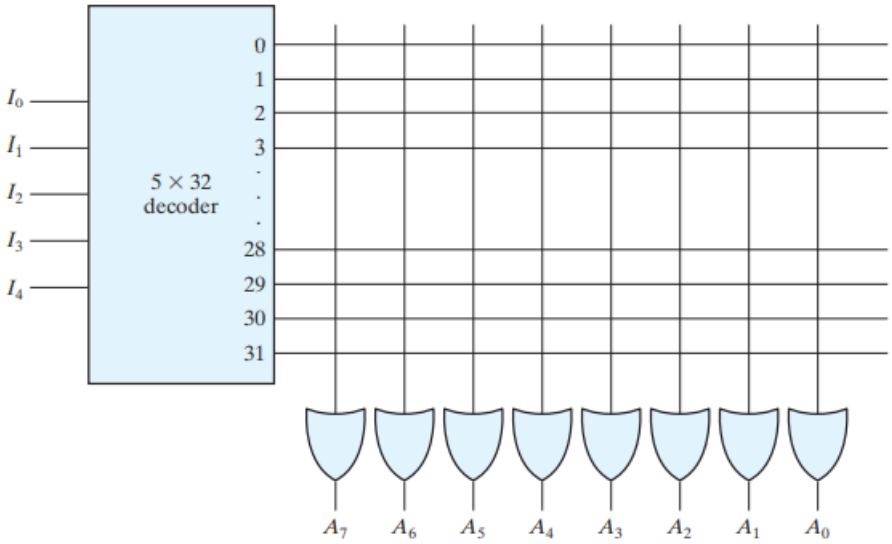
Fig: circuit with SR latch

If an output variable is 10 change value as a result of a change in state, then this variable is assigned a **don' -t**care condition. If a 0 is entered as the output value for the unstable state c, then the change in the output variable will not take place until the end of the transition. If a 1 is entered, the change will take place at the start of the transition, since it makes no difference, Reduction of State and Flow tables when the change in output occurs, we place a don't-care entry for the output associated with unstable state.

8. Explain How to make 32x8 ROM using ROM? Draw the circuit diagram.

[MAY 2019, NOV 2019]

The unit consists of 32 words of 8 bits each. There are five input lines that form the binary numbers from 0 through 31 for the address. Figure 7.10 shows the internal logic construction of this ROM. The five inputs are decoded into 32 distinct outputs by means of a 5 * 32 decoder. Each output of the decoder represents a memory address. The 32 outputs of the decoder are connected to each of the eight OR gates. The diagram shows the array logic convention used in complex circuits. Each OR gate must be considered as having 32 inputs. Each output of the decoder is connected to one of the inputs of each OR gate. Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains 32 * 8 = 256 internal connections. In general, a 2k * n ROM will have an internal k * 2k decoder and n OR gates. Each OR gate has 2k inputs, which are connected to each of the outputs of the decoder.



The internal binary storage of a ROM is specified by a truth table that shows the word content in each address. For example, the content of a 32 * 8 ROM may be specified with a truth table. The truth table shows the five inputs under which are listed all 32 addresses. Each address stores a word of 8 bits, which is listed in the outputs columns. The table shows only the first four and the last four words in the ROM. The complete table must include the list of all 32 words.

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮							⋮			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Truth table for ROM

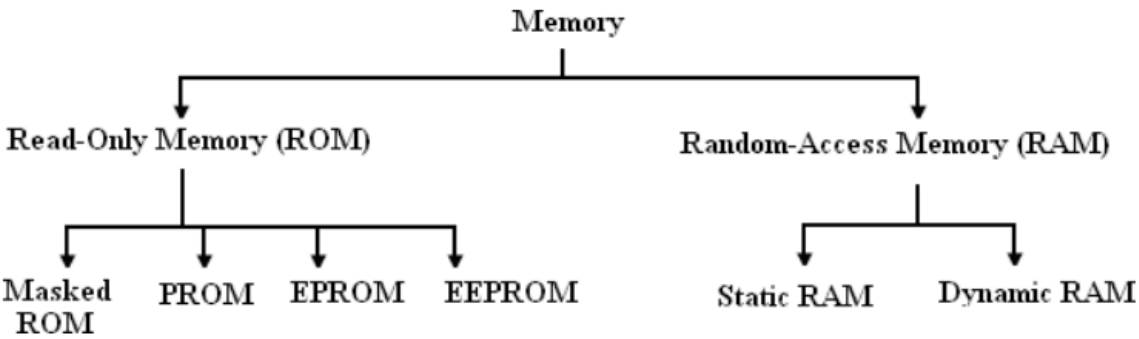
9. Describe various memories used in digital logic. Explain any two types.

[NOV 2018, MAY 2019]

The mode of access of a memory system is determined by the type of components used. In a random-access memory, the word locations may be thought of as being separated in space, each word occupying one particular location. In a sequential-access

memory, the information stored in some medium is not immediately accessible, but is available only at certain intervals of time. A magnetic disk or tape unit is of this type. Each memory location passes the read and write heads in turn, but information is read out only when the requested word has been reached. In a random-access memory, the access time is always the same regardless of the particular location of the word. In a sequential- access memory, the time it takes to access a word depends on the position of the word with respect to the position of the read head; therefore, the access time is variable.

Integrated circuit RAM units are available in two operating modes: static and dynamic. Static RAM (SRAM) consists essentially of internal latches that store the binary information. The stored information remains valid as long as power is applied to the unit. Dynamic RAM (DRAM) stores the binary information in the form of electric charges on capacitors provided inside the chip by MOS transistors. The stored charge on the capacitors tends to discharge with time, and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. DRAM offers reduced power consumption and larger storage capacity in a single memory chip. SRAM is easier to use and has shorter read and write cycles.



Classification of memories

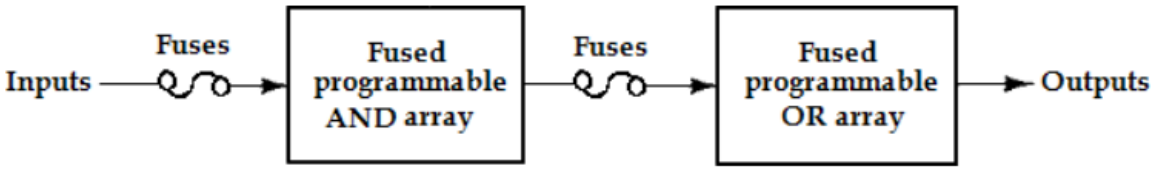
Memory units that lose stored information when power is turned off are said to be volatile. CMOS integrated circuit RAMs, both static and dynamic, are of this category, since the binary cells need external power to maintain the stored information. In contrast, a nonvolatile memory, such as magnetic disk, retains its stored information after the removalof power. This type of memory is able to retain information because the data stored on magnetic components are represented by the direction of magnetization, which

is retained after power is turned off. ROM is another nonvolatile memory. A nonvolatile memory enables digital computers to store programs that will be needed again after the computer is turned on. Programs and data that cannot be altered are stored in ROM, while other large programs are maintained on magnetic disks. The latter programs are transferred into the computer RAM as needed. Before the power is turned off, the binary information from the computer RAM is transferred to the disk so that the information will be retained.

10. Explain the structure of PLA & PAL. [MAY 2016, NOV 2019, SEP 2020]

PLA : -

A PLA consists of a programmable AND array and a programmable OR array. The product terms in the AND array may be shared by any OR gate to provide the required sum of product implementation. The PLA is developed to overcome some of the limitations of the PROM. The PLA is also called an FPLA (Field Programmable Logic Array) because the user in the field, not the manufacturer, programs it.



Programmable Logic Array (PLA)

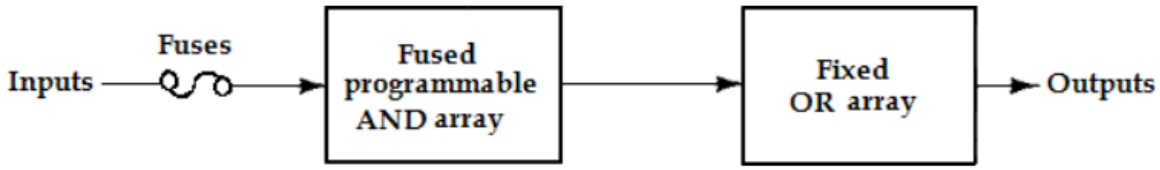
The PLA is similar in concept to the PROM, except that the PLA does not provide full decoding of the variables and does not generate all the minterms. The decoder is replaced by an array of AND gates that can be programmed to generate any productterm of the input variables. The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.

The internal logic of a PLA with three inputs and two outputs. Such a circuit is too small to be useful commercially, but is presented here to demonstrate the typical logic configuration of a PLA. The diagram uses the array logic graphic symbols for complex circuits. Each input goes through a buffer–inverter combination, shown in the diagram with a

composite graphic symbol, that has both the true and complement outputs. Each input and its complement are connected to the inputs of each AND gate, as indicated by the intersections between the vertical and horizontal lines. The outputs of the AND gates are connected to the inputs of each OR gate.

PAL : -

The basic PAL consists of a programmable AND array and a fixed OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate. It is developed to overcome certain disadvantages of the PLA, such as longer delays due to the additional fusible links that result from using two programmable arrays and more circuit complexity.



Programmable Array Logic (PAL)

The PAL is a programmable logic device with a fixed OR array and a programmable AND array. Because only the AND gates are programmable, the PAL is easier to program than, but is not as flexible as, the PLA. The logic configuration of a typical PAL with four inputs and four outputs. Each input has a buffer–inverter gate, and each output is generated by a fixed OR gate. There are four sections in the unit, each composed of an AND–OR array that is three wides, the term used to indicate that there are three programmable AND gates in each section and one fixed OR gate. Each AND gate has 10 programmable input connections, shown in the diagram by 10 vertical lines intersecting each horizontal line. The horizontal line symbolizes the multiple-input configuration of the AND gate. One of the outputs is connected to a buffer–inverter gate and then fed back into two inputs of the AND gates.

A typical PAL integrated circuit may have eight inputs, eight outputs, and eight sections, each consisting of an eight-wide AND–OR array. The output terminals are sometimes driven by three-state buffers or inverters. In designing with a PAL, the Boolean functions must be simplified to fit into each section. Unlike the situation with a PLA, a product term cannot be shared among two or more OR gates. Therefore, each function can be simplified by itself,

without regardto common product terms. The number of product terms in each section is fixed, and if the number of terms in the function is too large, it may be necessary to use two sections to implement one Boolean function.