DOWNLOADED FROM STUCOR APP

**SHREE SATHYAM COLLEGE OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF ECE**

**ALL THE 5 UNITS NOTES**

**FOR II YEAR / III SEMESTER**

**EC8392-DIGITAL ELECTRONICS**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

## UNIT –I

## DIGITAL FUNDAMENTALS

## PART B

**1. State and prove the postulates, theorems of Boolean algebra. (8+8) (U.Q)**

Huntington listed six theorems of Boolean algebra and four of its postulates.

**Postulates and theorems of Boolean algebra.**

| | | |
|---|---|---|
| Postulate 2 | a) x + 0 = x | b) x . 1 = x |
| Postulate 5 | a) x + x'= 1 | b) x . x' = 0 |
| Theorem 1 | a) x + x = x | b) x.. x = x |
| Theorem 2 | a) x + 1 = 1 | b) x.. 0 = 0 |
| Theorem 3 (Involution) | a) (x') '= x | |
| Postulate 3 (Commutative) | a) x + y = y + x | b)xy = yx |
| Theorem 4 (Associative) | a) x +(y +z) = (x +y )+z | b)x(yz) = (xy)z |
| Postulate 4 (Distributive) | a) x (y +z) = xy +xz | b) x+ yz = (x+y) (x+z) |
| Theorem 5 Demorgan's | a) (x +y)'= x'y' | b) (xy)'= x'+y' |
| Theorem 6 | a)x + xy = x | b) x (x+y) = x |

**Postulate 2 :**

(a) x + 0 = x

L.H.S

Let x = 0

0 + 0 = 0 ⟶ x

Let x = 1

1 + 0 = 1 ⟶ x

In both cases,

L.H.S = R.H.S

Therefore,

x + 0 = x

(b) x .1 = x

L.H.S

Let x = 0

0.1 = 0 ⟶ x

Let x = 1



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

$1 \cdot 1 = 1 \longrightarrow x$

In both cases,

L.H.S = R.H.S, Therefore, $x \cdot 1 = x$

**Postulate 5:**

(a) $x + x' = 1$

L.H.S

Let x = 0

$0 + 1 = 1 \longrightarrow x$

Let x = 1

$1 + 0 = 1 \longrightarrow x$

In both cases,

L.H.S = R.H.S

Therefore,

$x + x' = 1$

(b) $x \cdot x' = 0$

L.H.S

Let x = 0

$0.1 = 0 \longrightarrow x$

Let x = 1

$1 \cdot 0 = 1 \longrightarrow x$

In both cases,

L.H.S = R.H.S

Therefore, $x \cdot x' = 0$

**Theorem 1:**

a) $x + x = x$

L.H.S

$x + x = (x+x).1$ by postulate 2(b)

$= (x+x).(x+x')$ 5(a)

$= xx + xx + xx' + xx' = x\,x + xx' = x + xx'$ 4(b)

$= x + 0$ 5(b)

$= x \longrightarrow$ R.H.S 2(a)

b) $x \cdot x = x$

L.H.S

$x \cdot x = xx + 0$ by postulate 2(a)

$= xx + xx'$ by postulate 5(b)

$= x(x + x')$ 4(a)

$= x \cdot 1$ 5(a)

$= x \longrightarrow$ R.H.S 2(b)

**Theorem 2**

(a) $x + 1 = 1$

L.H.S

$x + 1 = 1.(x+1)$ by postulate 2(b)

$= (x+x')(x+1)$ 5(a)



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

= x + x'.1 4(a)
= x + x' 2(b)
= 1 5(a)

(b) x . 0 = 0
It is proved by duality.

**Theorem 3:**

- From postulate 5, we have x + x' = 1 and x . x' = 0, which defines the complement of x.
- The complement of x' is x and also (x')'. Therefore, since the complement is unique, we have that (x')' = x

**Postulate 3: (Commutative)**

(a). x + y = y + x

- This states that the order in which the variables are OR ed makes no difference in the output.

| x | y | x + y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

L.H.S

| x | y | y + x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

R.H.S

b) x y = y x

- The commutative law of multiplication states that the order in which the variables are ANDed makes no difference in the output.

L.H.S

| x | y | x y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

R.H.S

| x | y | yx |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Theorem 4, associative.**

a) x + (y+z) = (x+y) + z



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

This law states that in the ORing of several variables, the result is the same regardless of the grouping of the variables.

L.H.S

| x | y | z | y+z | x+(y+z) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

R.H.S

| x | y | z | x+y | (x+y)+z |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 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 | 1 |

b) x(yz) = (xy)z

- This associative law of multiplication states that it makes no difference in what order the variables are grouped when ANDing several variables.

L.H.S

| x | y | z | yz | x(yz) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

R.H.S

| x | y | z | xy | z(xy) |
|---|---|---|---|---|
| 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 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Postulate 4: distributive**

a) x (y +z) = xy + xz

- This distributive law states that ORing several variables and ANDing the result with a single variable is equivalent to ANDing the result with a single variable with each of the several variables and then ORing the products.

L.H.S

R.H.S

| x | y | z | xy | xz) | xy+xz |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |

| x | y | z | y+z | x(y+z) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

b) x + yz = (x + y) (x + z)

L.H.S

| x | y | z | yz | x+yz |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

R.H.S

| x | y | z | x+y | x+z | (x+y)(x+z) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

| x | y | x’ | y’ | x’. y’ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |

**Theorem 5: Demorgan’s Theorem:**

a) (x+y)’=x’y’

- The complement of a sum is equal to the product of the complements.

b) (AB)’ = A’+B’

- The complement of a product is equal to the sum of the complements.



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |

**Theorem 6: Absorption**

a) x + xy = x

x + xy = x.1+ xy — by postulate 2(b)
= x(1+y) — 4(a)
= x(y+1) — 3(a)
= x.1 — 2(a)
= x — 2(b)

b) x(x+y) = x

This can be proved by duality.

---

**2. Use Quine McClusky method to obtain the minimal sum for the following function. $F(X_1X_2X_3X_4) = \sum (0, 1, 3, 6, 7, 14, 15)$ (April 2003)**

**Step1:** List all minterms in the binary form

| Minterm | Binary Representation |
|---|---|
| $m_0$ | 0000 |
| $m_1$ | 0001 |
| $m_3$ | 0011 |
| $m_6$ | 0110 |
| $m_7$ | 0111 |
| $m_{14}$ | 1110 |
| $m_{15}$ | 1111 |

| x | y | x +y | (x+y)' |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

**Step2:** Arrange the minterms according to the 1's.

| Minterm | Binary Representation |
|---|---|
| $m_0$ | 0000 |
| $m_1$ | 0001 |
| $m_3$ | 0011 |
| $m_6$ | 0110 |
| $m_7$ | 0111 |
| $m_{14}$ | 1110 |
| $m_{15}$ | 1111 |



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Step3:** Compare each every term in the category and if they position, put s check in the next column with they differed.

| Prime Implicants | Binary Representation |
|---|---|
| $m_0,m_1$ | 000- |
| $m_1, m_3$ | 00-1 |
| $m_3,m_7$ | 0-11 |
| $m_6,m_7,m_{14},m_{15}$ | -11- |

binary number with adjacent next higher differ, only by one bit mark and copy the term '-' in the position that

**Step 4:** Step 3 is repeated until a single pass through cycles yield no further elimination of literals.

| Minterm | Binary Representation | Minterm | Binary Representation |
|---|---|---|---|
| $m_0,m_1$ | 000- | $m_0,m_1$ | 000- |
| $m_1, m_3$ | 00-1 | $m_1, m_3$ | 00-1 |
| $m_3,m_7$ | 0-11 | $m_3,m_7$ | 0-11 |
| $m_6,m_7$ | 011- | $m_6,m_7,m_{14},m_{15}$ | -11- |
| $m_6,m_{14}$ | -110 | $m_6,m_{14},m_7,m_{15}$ | -11- |
| $m_7,m_{15}$ | -111 | | |
| $m_{14},m_{15}$ | 111- | | |

**Step5:** List the prime implicants.

**Step 6:** Select the minimum number of prime implicants which must cover all the minterms.

The selection procedure is as follows,

| Prime Implicants | $m_0$ | $m_1$ | $m_3$ | $m_6$ | $m_7$ | $m_{14}$ | $m_{15}$ |
|---|---|---|---|---|---|---|---|
| $m_0,m_1$ | * | * | | | | | |
| $m_1, m_3$ | | * | * | | | | |
| $m_3,m_7$ | | | * | | * | | |
| $m_6,m_7,m_{14},m_{15}$ | | | | * | * | * | * |

(i) Search the single dot columns and select the prime implicants corresponding to that dot by putting the check mark infront of it.

(ii) Search for multi dot column one by one. If the corresponding minterm is already included in the final expression ignore the minterm and go to next multi-dot column; otherwise include the corresponding prime implicant in the final expression.

The simplified expression is,

$$F(X_1X_2X_3X_4) = (000-) + (-11-) + (00-1) = \bar{x}_1\bar{x}_2\bar{x}_3 + x_2x_3 + \bar{x}_1\bar{x}_2x_4$$



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**3. Simplify the function using Karnaugh map.**

**(i) F (ABCD) = ∑ (0, 1, 2, 4, 5, 7, 11, 15)**

**(ii) F (WXYZ) = ∑ (2, 3, 10, 11, 12, 13, 14, 15) (April 2003)**

(i) **F(A,B,C,D) = ∑ (0,1,2,4,5,7,11,15)**

| | | | |
|---|---|---|---|
| 1 | 1 | | 1 |
| 1 | 1 | 1 | |
| | | 1 | |
| | | 1 | |

$$= \overline{A}\overline{B}\overline{D} + \overline{A}\overline{C} + BCD + ACD$$

(ii) F(W,X,Y,Z) = ∑ (2,3,10,11,12,13,14,15)

| | | | |
|---|---|---|---|
| | | 1 | 1 |
| | | | |
| 1 | 1 | 1 | 1 |
| | | 1 | 1 |

$$= \overline{x}y + wx$$

**4. Prove that** $(x_1 + x_2) \bullet (\overline{x_1} \bullet \overline{x_3} + x_3)\overline{(\overline{x_2} + x_1 \bullet x_3)} = \overline{x_1}x_2$ **(May 2003) (5)**

**Solution:**

L.H.S

$$= (x_1 + x_2) \bullet (\overline{x_1} \bullet \overline{x_3} + x_3)\overline{(\overline{x_2} + x_1 \bullet x_3)}$$

$$= (x_1 + x_2) \bullet (\overline{x_1} \bullet \overline{x_3} + x_3)(x_2 \bullet \overline{(x_1 x_3)})$$

$$= (x_1 + x_2) \bullet (\overline{x_1} \bullet \overline{x_3} + x_3)(x_2 \bullet (\overline{x_1} + \overline{x_3}))$$



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

$= (x_1 + x_2) \bullet (\overline{x_1}\,\overline{x_3} + x_3) \bullet (x_2\overline{x_1} + x_2\overline{x_3})$

$= (x_1 + x_2) \bullet (\overline{x_1} + x_3) \bullet (x_2\overline{x_1} + x_2\overline{x_3})$ $\quad A + \overline{A}B = A + B$

$= (x_1x_3 + \overline{x_1}x_2 + x_2x_3) \bullet (x_2\overline{x_1} + x_2\overline{x_3})$

$= (x_2\overline{x_1} + \overline{x_1}x_2x_3 + \overline{x_1}x_2\overline{x_3})$

$= (x_2\overline{x_1}(1 + x_3 + \overline{x_3})$

$= (x_2\overline{x_1})$ $\quad (1 + x = x)$

= R.H.S.

Thus, it is proved.

---

**5.**

**(i) Express the function f(x,y,z) =** $XY + X\overline{Z}$ **as a product of sum terms form.**

**(ii) Express the following function as the minimal sum of products, using a K-map (12).**

**F (a,b,c,d) =∑m(0,2,4,5,6,8,10,15) + ∑Φ(7,13,14) (April 2005).**

**Solution:**

(i) f(x,y,z) = $XY + X\overline{Z}$

$= X(Y + \overline{Z})$

$= (X + Y \bullet \overline{Y} + Z \bullet \overline{Z})(X \bullet \overline{X} + Y + \overline{Z})$

$= (X + Y + Z)(X + Y + \overline{Z})(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(X + Y + \overline{Z})$ $(\overline{X} + Y + \overline{Z})$

$= (X + Y + Z)(X + Y + \overline{Z})(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(\overline{X} + Y + \overline{Z})$

$= M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_5$

$= \pi M (0,1,2,3,5)$

(ii) F (a,b,c,d) =∑m(0,2,4,5,6,8,10,15) + ∑Φ(7,13,14)

| | | | |
|---|---|---|---|
| 1 | | | 1 |
| 1 | 1 | x | 1 |
| | x | 1 | x |
| 1 | | | 1 |

$= \overline{b}\overline{d} + \overline{a}b + bcd$

---



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**6. Implement the switching function whose octal designation is 274 using NOR gates. (April 2003)**

**Solution:**

The switching function with octal designation of 274 can be given as,

$$F(A,B,C) = \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C}$$

$$F(A,B,C) = \bar{\bar{F}}(A,B,C) = \overline{\bar{A}B\bar{C} + ABC + A\bar{B}\bar{C}}$$

$$= \overline{\bar{A}B\bar{C}} \bullet \overline{ABC} \bullet \overline{A\bar{B}\bar{C}}$$

$$= (\bar{\bar{A}} + \bar{B} + \bar{\bar{C}}) \bullet (\bar{A} + \bar{B} + \bar{C}) \bullet (\bar{A} + \bar{\bar{B}} + \bar{\bar{C}})$$

$$= (A + \bar{B} + C) \bullet (\bar{A} + \bar{B} + \bar{C}) \bullet (\bar{A} + B + C)$$

---

**7. Simplify the Boolean function using K-map and tabular methods. Compare the methods. F (A, B, C, D) =∑m(4,5,6,7,8) d (A, B, C, D) = ∑m(11,12,13,14,15) Implement using only NAND gates. (Nov 2004)**

**Solution:**

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| x | x | x | x |
| 1 | | x | |

$$= B + A\bar{C}\bar{D}$$

**Tabulation method:**

(i) Arranging the minterms according to the number of 1's

| Minterm | Binary Representation | Minterm | Binary Representation |
|---|---|---|---|
| $m_4$ | 0100 | $m_4$ | 0100 |
| $m_5$ | 0101 | $m_8$ | 1000 |
| $m_6$ | 0110 | $m_5$ | 0101 |
| $m_7$ | 0111 | $m_6$ | 0110 |
| $m_8$ | 1000 | $m_{12}$ | 1100 |
| $m_{11}$ | 1011 | $m_7$ | 0111 |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| $m_{12}$ | 1100 | $m_{11}$ | 1011 |
| $m_{13}$ | 1101 | $m_{13}$ | 1101 |
| $m_{14}$ | 1110 | $m_{14}$ | 1110 |
| $m_{15}$ | 1111 | $m_{15}$ | 1111 |

Elimination of literals:

| Minterm | Binary Representation | Minterm | Binary Representation |
|---|---|---|---|
| 4,5 | 010- | 4,5,6,7 | 01-- |
| 4,6 | 01-0 | 4,6,5,7 | 01-- |
| 4,12 | -100 | 4,12,5,13 | -10- |
| 8,12 | 1-00 | 5,7,13,15 | -1-1 |
| 5,7 | 01-1 | 5,13,7,15 | -1-1 |
| 5,13 | -101 | 6,7,14,15 | -11- |
| 6,7 | 011- | 6,14,7,15 | -11- |
| 6,14 | -110 | | |
| 7,15 | -111 | | |
| 11,15 | 1-11 | | |
| 13,15 | 11-1 | | |
| 14,15 | 111- | | |

Further elimination of literals:

| Minterm | Binary Representation |
|---|---|
| 4,5,6,7 | 01-- |
| 5,7,13,15 | -1-1 |
| 4,5,12,13,6,7,14,15 | -1-- |
| 8,12 | 1-00 |
| 11,15 | 1-11 |

Selecting the minimum number of prime implicants as

| Prime Implicants | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ | $m_{11}$ | $m_{12}$ | $m_{13}$ | $m_{14}$ | $m_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 4,5,6,7 | * | * | * | * | | | | | | |
| 5,7,13,15 | | * | | * | | | | * | | * |
| 4,5,12,13,6,7,14,15 | * | * | * | * | | | * | * | * | * |
| 8,12 | | | | | * | | * | | | |
| 11,15 | | | | | | * | | | | * |

$= B + A\bar{C}\bar{D}$

---

**8. (i) State and prove demorgan's theorem and expand the function**

**F = $(\overline{((A+B)C+\bar{C}D)}$ (6) Nov 2003**

**Solution:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

$$(\overline{((A+B)C+\overline{C}\overline{D})} = \overline{(A+B)C} \bullet \overline{\overline{C}\overline{D}}$$
$$= \overline{((A+B)+\overline{C})} \bullet (C+\overline{D})$$
$$= ((\overline{A} \bullet \overline{B})+\overline{C}) \bullet (C+\overline{D})$$
$$= (\overline{A}\overline{C}+\overline{B}\overline{C})(C+\overline{D})$$
$$= \overline{A}\overline{C}C+\overline{B}\overline{C}C+\overline{A}\overline{C}\overline{D}+\overline{B}\overline{C}\overline{D}$$
$$= \overline{A}\overline{C}\overline{D}+\overline{B}\overline{C}\overline{D}$$

**(ii) Simplify the following switching function using karnaugh map,**
**F(A,B,C,D) = ∑ (0,5,7,8,9,10,11,14,15) + Φ (1,4,13) (11) (Nov.2003)**

---

**9. Simplify the Boolean function using K-map and tabular methods. Compare the methods. F (A, B, C, D) =∑m(4,5,6,7,8) d (A, B, C, D) = ∑m(11,12,13,14,15) Implement using only NAND gates. (Nov 2004)**

**Solution:**

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| x | x | x | x |
| 1 | | x | |

$= B + A\overline{C}\overline{D}$

**Tabulation method:**

(i) Arranging the minterms according to the number of 1's

| Minterm | Binary Representation | Minterm | Binary Representation |
|---|---|---|---|
| $m_4$ | 0100 | $m_4$ | 0100 |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| $m_5$ | 0101 | $m_8$ | 1000 |
| $m_6$ | 0110 | $m_5$ | 0101 |
| $m_7$ | 0111 | $m_6$ | 0110 |
| $m_8$ | 1000 | $m_{12}$ | 1100 |
| $m_{11}$ | 1011 | $m_7$ | 0111 |
| $m_{12}$ | 1100 | $m_{11}$ | 1011 |
| $m_{13}$ | 1101 | $m_{13}$ | 1101 |
| $m_{14}$ | 1110 | $m_{14}$ | 1110 |
| $m_{15}$ | 1111 | $m_{15}$ | 1111 |

Elimination of literals:

| **Minterm** | **Binary Representation** | **Minterm** | **Binary Representation** |
|---|---|---|---|
| 4,5 | 010- | 4,5,6,7 | 01-- |
| 4,6 | 01-0 | 4,6,5,7 | 01-- |
| 4,12 | -100 | 4,12,5,13 | -10- |
| 8,12 | 1-00 | 5,7,13,15 | -1-1 |
| 5,7 | 01-1 | 5,13,7,15 | -1-1 |
| 5,13 | -101 | 6,7,14,15 | -11- |
| 6,7 | 011- | 6,14,7,15 | -11- |
| 6,14 | -110 | | |
| 7,15 | -111 | | |
| 11,15 | 1-11 | | |
| 13,15 | 11-1 | | |
| 14,15 | 111- | | |

Further elimination of literals:

| Minterm | Binary Representation |
|---|---|
| 4,5,6,7 | 01-- |
| 5,7,13,15 | -1-1 |
| 4,5,12,13,6,7,14,15 | -1-- |
| 8,12 | 1-00 |
| 11,15 | 1-11 |

Selecting the minimum number of prime implicants as

| Prime Implicants | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ | $m_{11}$ | $m_{12}$ | $m_{13}$ | $m_{14}$ | $m_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 4,5,6,7 | * | * | * | * | | | | | | |
| 5,7,13,15 | | * | | * | | | | * | | * |
| 4,5,12,13,6,7,14,15 | * | * | * | * | | | * | * | * | * |



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8,12 | | | | | * | | * | | | |
| 11,15 | | | | | | * | | | | * |

$= B + A\overline{C}\overline{D}$

---

**10. (i) Specify the radix and the symbols used in (a) binary (b) ternary (c) quinary (d) octal (e) Hexadecimal number system. (4) (April 2003)**

**(a) Binary:**

- Binary system with its two digits is a base (radix) two system. The two binary digits (bits) are 1 and 0. In binary system each binary digit commonly known as bit, has its own value or weight. The number 2 is used at the suffix of binary number. Eg.$1101.101_2$

| BASE | Number System | Digit Symbols |
|---|---|---|
| 2 | Binary | 0,1 |
| 3 | Ternary | 0,1,2 |
| 4 | Quaternary | 0,1,2,3 |
| 5 | Quinary | 0,1,2,3,4 |
| 8 | Octal | 0,1,2,3,4,5,6,7 |
| 10 | Decimal | 0,1,2,3,4,5,6,7,8,9 |
| 12 | Duodecimal | 0,1,2,3,4,5,6,7,8,9,A,B |
| 16 | Hexadecimal | 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E.F |

**(b) Ternary:**

- Ternary system is base (radix) 3 system having three digits 0, 1, 2. The number 3 is used at the suffix of ternary number to indicate that it is a ternary number. Eg:$(102)_3$.

**(c) Quinary:**

- Quinary number system is base (radix) 5 system having five digits 0, 1, 2,3,4,5. The number 5 is used at the suffix of quinary number. Eg: $(420)_5$.

**(d) Octal Number system:**

- Octal number system is base (radix) 8 system having 8 digits 0, 1, 2, 3, 4, 5, 6, 7. The number 8 is used at the suffix of octal number. Eg: $(267)_8$.

**(e) Hexa decimal number system:**

- Hexadecimal number system has a base of 16 having 16 digits:0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F. This system is useful for human



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

communications between with a computer. $(AB.CD)_H$.

**(ii)Convert $(329.678)_{10}$ to an equivalent number in base 6 having a conversion error less than 0.001. (4) (April 2003)**

**(a) Integer Part:**

| | | |
|---|---|---|
| 6 | 329 | |
| 6 | 54 | 5 |
| 6 | 9 | 0 |

$(905)_6$

**(b) Fractional Part:**

| | | |
|---|---|---|
| 0.678 x 6 | 4 | 0.068 |
| 0.068 x 6 | 0 | 0.408 |
| 0.408 x 6 | 2 | 0.448 |
| 0.448 x 6 | 2 | 0.688 |

$(0.4022)_6$

$= (905.4022)_6$

**11. Perform the following arithmetic using 9's arithmetic. Compare them.**

**(i) 835 – 274**

**(ii) 429 - 476 using BCD and excess 3 codes. (16) Nov 2004.**

**9's complement subtraction:**

**Steps:**

(i) Add the minuend (M) to the (r-1)'s complement of subtrahend N.

(ii) Check the result obtained in step 1.

(a) If an end carry occurs add 1 to the LSB (end around carry).

(b) If an end carry doesn't occur, take the (r-1)'s complement of the number obtained in step1 and place a negative sign in front of it.

**(i ) 835 – 274 (using 9's complement)**

**Step 1:**

9's complement of 274 = $r^n - r^m - N$

$= 10^3 - 1 - 274$

$= 999 - 274$

$= 725$

**Step 2:**

M + (r-1)'s complement = 835 +

= 725

= 1 560

End carry occurs, therefore adding 1 we get

= 560

= 1



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

= 561

**(ii) 429 - 476 using BCD and excess 3 codes.**

BCD addition procedure:

(i) Add two BCD numbers using ordinary binary addition.

(ii) If four-bit is equal to or less than 9, no correction is needed. The sum is in proper BCD form.

(iii) If the four bit sum is greater than 9 or if a carry is generated from the four bit sum the sum is invalid.

(iv) To correct the invalid sum, add $0110_2$ to the four bit sum. If a carry results from this addition, add it to the next higher order BCD digit.

**(a) Using BCD subtraction:**

Step 1: (9's complement):

9's complement of 476 = $r^n$-$r^m$-N

= $10^3$-$10^0$- 476

= 999 – 476 = 523

Step 2:

M + (r-1)'s complement = 0100 0010 1001

= 0101 0010 0011

= 1001 0100 1100

0110

= 1001 0101 0010

= 9 5 2

9's complement 952 = $10^3$-$10^0$-952

= 999 – 952

= - 47

**(b) Excess 3 code (9's complement).**

To perform excess 3 subtraction we have to

- Complement the subtrahend.
- Add complemented subtrahend to minuend.
- If carry =1 result is positive. Add 3 and end around carry.
- If carry =0 result is negative. Subtract 3.

Excess 3 code for 476 = 0111 1010 1001

**Step 1:**

9's complement of 476 = 1000 0101 0110

**Step2:**

M + 9's complement of N = 0111 0101 1100

= 1000 0101 0110

= 1111 1011 0010

(-3)0011(-3) 0011 (+3) 0011

= 1100 1000 0101

No carry occurs therefore taking 9'complement we get,



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

= - 0011 0111 1010
= - 3 7 A

(-37A) is the excess 3 code for (-047)

## PART C

## ASSIGNMENT QUESTIONS

**1. Determine the minimum sum of products and minimum product of sums for F=b'c'd'+bcd+acd'+a'b'c+a'bc'd**
**Find the minterm expansion of**
**f(a,b,c,d)= a'(b'+d)+acd'. (May/Jun2005)**

**2.Using tabulation method simplify the Boolean function F(w,x.y.z)=∑m(1,2,3,5,9,12,14,15) which has don't care conditions d(4,8,11).(May/Jun 2007)(Nov/Dec 2005).**

**3. Show that how NAND gates can be used to implement the basic Boolean functions. (April /May2004).**

**4.Reduce the Boolean function using K-Map technique and implement using gates F(w,x,y,z)=∑(0,1,4,8,9,10) which has the don't care conditions d(w,x,y,z)=∑(2,11) (May/Jun 2007).**

**5.Simplify the following Boolean function using tabulation method F(A,B,C,D)=∏M(0,2,3,6,7,8,10,12,13).**

**6. Simplify the following Boolean function using tabulation method**
**F(A,B,C,D)=∑m(2,3,7,9,11,13)+∑d(1,10,15) (Dec 2009)**

**7. Reduce the Boolean function using K-Map technique**
**(a)F=∑m(0,2,4,5,6,7,8,10,11,12,14,15)**
**(b)F=π (0, 6, 7, 8, 12, 13, 14, 15) (Dec 2009)**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| Input | Output |
|---|---|

## UNIT- II

## COMBINATIONAL CIRCUITS

### PART B

1. **Design a half adder and full adder circuits or design a full adder (April 2003)**

**Half adder:**

- A combinational circuit that performs the addition of two bits is called half adder.
- This circuit needs two binary inputs and two binary outputs.
- The input variables designate the augend and addend bits; the output variables produce the sum and carry.
- Symbols x and y are assigned to two inputs, s for (sum) and c (carry) to the outputs.

**Truth Table**



STUCOR APP
DOWNLOADED FROM STUCOR APP

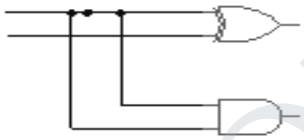
DOWNLOADED FROM STUCOR APP

:

| x | y | Carry (C) | Sum (S) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

- Boolean expression for half adder can be obtained as,
  $s = x'y + x'y = x \oplus y$
  $c = xy$
- The logic diagram of the half adder implemented with an exclusive OR and an AND gate. Half adder can be used to construct a full adder.

**Logic diagram:**



**Full adder:**

- A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is a full adder.
- Two half adders can be used to implement a full adder.
- It consists of three inputs and two outputs.
- Two of the input variable denoted by x and y, represent the two significant bits added.
- The third input, z represents the carry from the previous lower significant position.
- The two outputs are designated by the symbols s for sum and c for carry.

**Truth Table:**

| x | y | z | c | s |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Full Adder:**

Boolean expression for the full adder can be obtained as,

S = x'y'z+x'yz'+xy'z'+xyz

= z (x'y'+xy) +z' (x'y+xy')

= z (x'y+xy')'+z' (x'y+xy')

Let A = x'y+xy'= x ⊕ y

= zA'+z'A = z ⊕ A

= z ⊕ (x ⊕y) (A = x ⊕ y)

C = xyz + xyz' + xy'z + x'yz

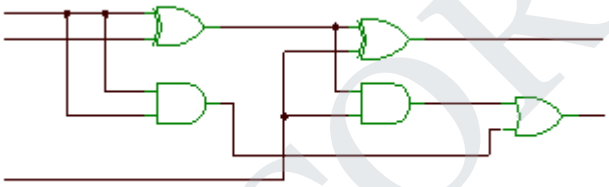
= xy (z+z') + z (xy'+x'y)

= xy + z (xy'+x'y)

C = z (x ⊕ y) + xy

| x | y | B | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |

Implementation of full adder with two Half adders and an OR gate.



---

## 2. Design a half subtractor and full subtractor circuits (Nov/Dec2003).

**Half subtractor:**

- Half subtractor is a combinational circuit that subtracts the two bits and produces their difference.
- It also has an output to specify if a 1 has been borrowed.
- It has two inputs x (minuend) and y (subtrahend)
- It has two outputs, one generates the difference (D) and other designated as b for borrow.
- The truth table for input output relationships of a half subtractor is given as,

**Truth Table:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

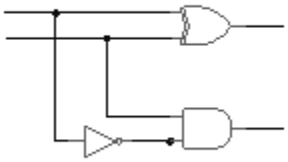
| | | | |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

- The Boolean functions for the two outputs of the subtractor are derived as follows

  $D = x'y + x'y = x \oplus y$

  $B = x'y$

**Implementation of half subtractor.**



**Full subtractor:**

- A full adder is a combinational circuit that performs a subtraction between two bits, taking into account that 1 may have borrowed by a lower significant stage.
- The circuit has three inputs and two outputs.
- The three inputs denote x, y and z denote the minuend, subtrahend and previous borrow respectively.
- The two outputs, D and B, represent the difference and output borrow respectively.

The truth table for the circuit is given as,

**Truth Table:**

| x | y | z | B | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |

- The simplified Boolean expression for the two outputs of the full subtractor are derived as,

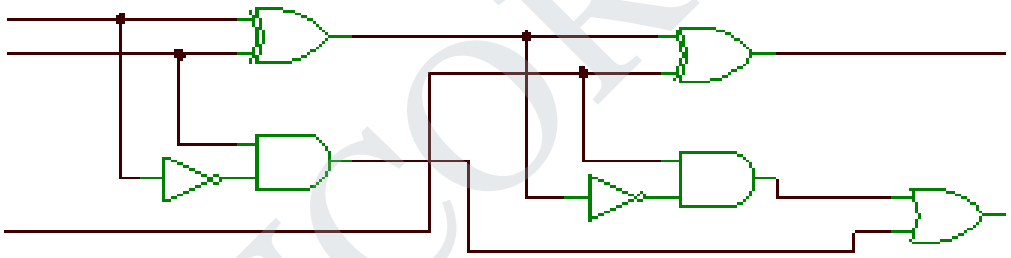
$$S = x'y'z+x'yz'+xy'z'+xyz$$
$$= z(x'y'+xy)+z'(x'y+xy')$$
$$= z(x'y+xy')'+z'(x'y+xy')$$

Let $A = x'y+xy' = x \oplus y$

$$= zA'+z'A = z \oplus A$$
$$= z \oplus (x \oplus y) \quad (A = x \oplus y)$$

$$B = x'yz + x'yz' + x'y'z + xyz$$
$$= x'y\,(z+z') + z\,(x'y'+xy)$$
$$= x'y + z(x'y+xy')'$$
$$B = z\,(x \oplus y)' + x'y$$

Implementation of full subtractor with two Half subtractor and an OR gate.



---

**3. (i) Draw the Logic diagram of a 4 bit parallel adder/subtractor using full adders and explain (Nov 2003).**

**(ii) Design a 4 bit binary subtractor circuit.**

**(iii) Design a 4-bit adder subtractor circuit.**

**(i) 4-bit Binary Adder :( 4 bit binary ripple carry adder)**

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in a chain.



STUCOR APP

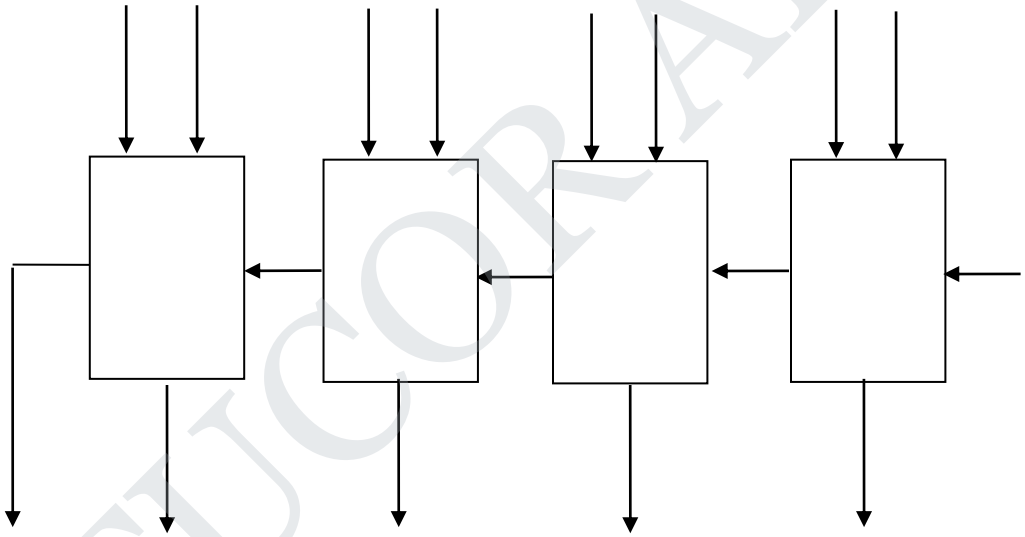
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- The input carry is $C_0$ and it ripples through the full adders to the output carry $C_4$.
- The S outputs generate the required sum bits.
- An n-bit adder requires n full adders in cascade.
- The input carry $C_0$ in the least significant position must be 0.

4-bit adder

| Subscript I | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|
| Input carry | 0 | 1 | 1 | 0 | $C_i$ |
| Augend | 1 | 0 | 1 | 1 | $A_i$ |
| Addend | 0 | 0 | 1 | 1 | $B_i$ |



**(ii) 4-bit Binary Subtractor:**

- The subtraction of unsigned binary numbers can be done by means of complements.
- The subtraction of A-B can be done by taking the 2's complement of B and adding it to A.
- The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.
- The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.
- The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder.

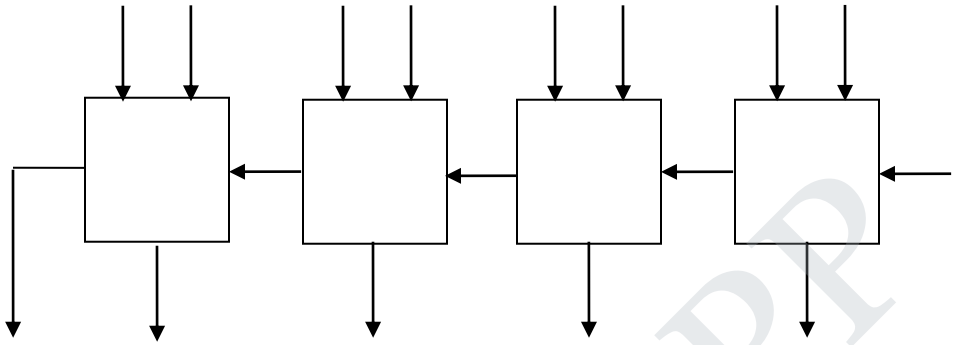


STUCOR APP

DOWNLOADED FROM STUCOR APP

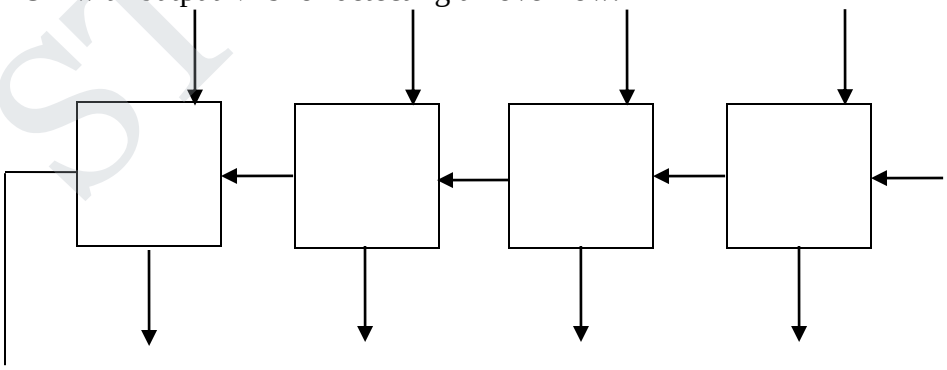
DOWNLOADED FROM STUCOR APP

- The input carry $C_0$ must be equal to1 when performing subtraction.
- The operation thus performed becomes A, plus the 1's complement of B, plus 1.
- This is equal to A plus the 2's complement of B.



**(iii) 4-Bit Adder Subtractor:**

- The addition and subtraction operations can be combined into one circuit with one common binary adder. This is achieved by including an exclusive OR gate with each full adder.
- The mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor.
- Each exclusive-OR gate receives input M and one of the inputs of B.
- When M=0, we have $B \oplus 0=B$. The full adders receive the value of B, the input carry is zero, and the circuit performs A plus B.
- When M=1, we have $B \oplus 1=B'$ and $C_0 = 1$. The B inputs are all complemented and a1 is added through the input carry.
- The circuit performs the operation A plus the 2's complement of B.The exclusive-OR with output V is for detecting an overflow.





STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**4. Design a parity generator to generate an odd parity bit for a 4-bit word. Use EX-OR and EX-NOR gates (April 2003).**

Solution:

- A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even.

**4-bit odd parity generator:**

**Truth table:**

| Four-bit Message | | | | Parity Bit |
|---|---|---|---|---|
| w | x | y | z | P |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**K-map simplification:**

| | | | |
|---|---|---|---|
| 1 | | 1 | |
| | 1 | | 1 |
| 1 | | 1 | |



STUCOR APP

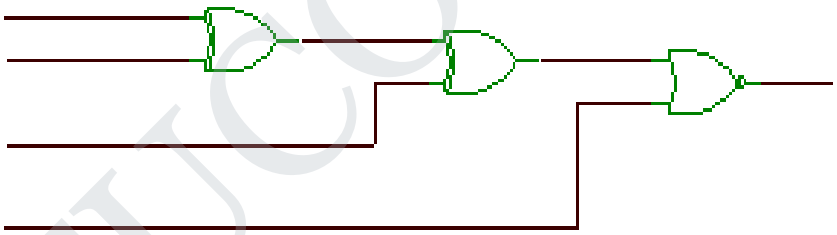
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| | 1 | | 1 |

$$= \overline{w}\overline{x}\overline{y}\overline{z} + \overline{w}\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}xy\overline{z} + w\overline{x}\overline{y}z + wxyz + w\overline{x}y\overline{z} + \overline{w}\overline{x}y\overline{z}$$
$$= \overline{w}\overline{x}(\overline{y}\overline{z} + yz) + \overline{w}x(\overline{y}z + y\overline{z}) + wx(\overline{y}\overline{z} + yz) + w\overline{x}(\overline{y}z + y\overline{z})$$
$$= (\overline{w}\overline{x} + \overline{w}\overline{x})(\overline{y}\overline{z} + y\overline{z}) + (\overline{y}\overline{z} + yz)(\overline{w}\overline{x} + wx)$$
$$= (w \oplus x)(y \oplus z) + \overline{(y \oplus z)}\,\overline{(w \oplus x)}$$
$$= CD + \overline{C}\,\overline{D} \text{ where C} = = (w \oplus x) and D = (y \oplus z)$$
$$= \overline{C \oplus D} = \overline{(w \oplus x \oplus y \oplus z)}$$
$$= \overline{(w \oplus x \oplus y \oplus z)}$$

**Logic diagram:**



---

5. **Design a combinational logic circuit to compare two 2-bit binary numbers A and B and to generate the outputs A< B, A=B and A>B. Is there a way to derive the third output from the first two outputs?(May 2006)**

**2 bit magnitude comparator:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- For comparing two 2-bit binary numbers, it requires four inputs and three outputs.
- The four inputs are designated as $A_1A_0B_1B_0$ and the three outputs are designated as A<B, A=B, A<B.
- The truth table that relates the input and output variables are tabulated as,

| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| $A_1$ | $A_0$ | $B_1$ | $B_0$ | A>B | A=B | A<B |
| 0 | 0 | 0 | 0 | 0 | 1 | 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 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

K-map simplification:

| | | | |
|---|---|---|---|
| | | | |
| 1 | | | |
| 1 | 1 | | 1 |
| 1 | 1 | | |

| | | | |
|---|---|---|---|
| 1 | | | |
| | 1 | | |
| | | 1 | |
| | | | 1 |

| | | | |
|---|---|---|---|
| | 1 | 1 | 1 |
| | | 1 | 1 |
| | | | |
| | | 1 | |

$$A = B = \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1}A_0\,\overline{B_1}B_0 + A_1A_0B_1B_0 + A_1\overline{A_0}B_1\overline{B_0}$$
$$= \overline{A_1}\,\overline{B_1}(\overline{A_0}\,\overline{B_0} + A_0B_0) + A_1B_1(A_0B_0 + \overline{A_0}\,\overline{B_0})$$
$$= (A_0 \oplus B_0)(A_1 \oplus B_1)$$



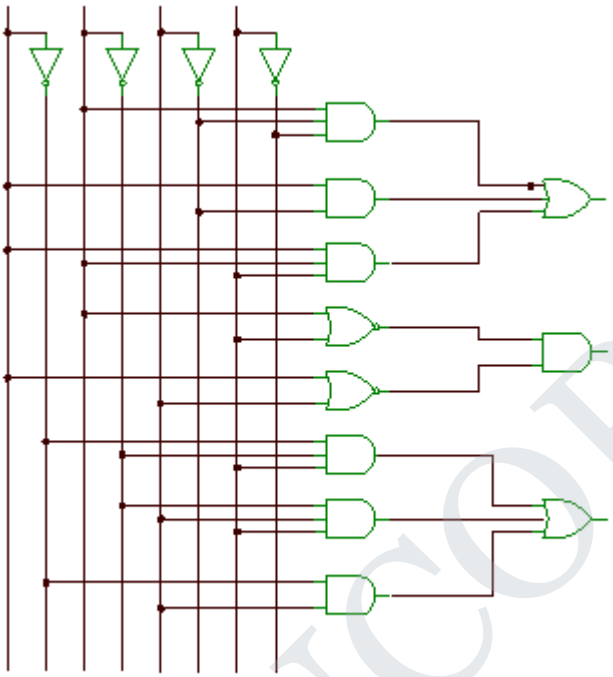
STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

$$A < B = \overline{A}_1 \overline{A}_0 B_0 + \overline{A}_0 B_1 B_0 + \overline{A}_1 B_1$$

**Logic Diagram:**



- The comparison of two nos. is an operation that determines if one number is greater than, less than, or equal to the other number.
- A magnitude comparator is a combinational circuit that (determines) compares two numbers, A & B and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables that indicate whether A>B, A=B or A<B.

---

## 6. Design a Carry look ahead generator (Apr/May 2004).

**Carry propagation:**

- The total propagation time is equal to the propagation delay of a typical gate times the number of gate level in the circuit.
- The longest propagation delay in an adder is the time it takes the carry to propagate through the full adders.

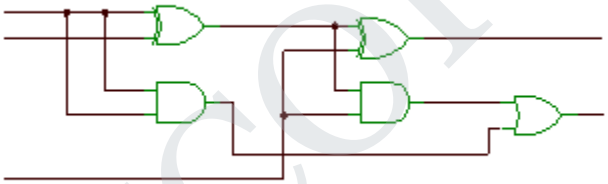


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- Each bit of the sum output depends on the value of the input carry. Thus only after the carry propagation and ripples through all stages will the last output $S_3$ and carry $C_4$ settle to their final correct value.
- The number of gate levels for the carry propagation can be found from the circuit of the full adder. The signals at $P_i$ and $G_i$ settle to their steady state values after they propagate through their respective gates and depend only on the input augend and addend bits.
- The signals from the input carry $C_i$ to the output carry $C_{i+1}$, propagates through an AND gate and an OR gate, which constitute two gate levels. If there are four full adders in the adder, the output carry $C_4$ would have 2 x 4=8 gate levels from $C_0$ to $C_4$. For an n-bit adder, there 2n gate levels for the carry to propagate from input to output.

- The carry propagation time should be limited because the outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs.
- The time consumed during the addition process is very critical. Therefore for reducing the carry propagation delay time there are two solutions.
  (i) To employ faster gates with reduced delays. However, physical circuits have a limit to their capability.
  (j) Another solution is to increase the equipment complexity in such a way that the carry delay time is reduced.. The most widely used technique employs the principle of carry look ahead.



➢ Consider the circuit of the full adder shown in the figure if we define two new binary variables

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

➢ The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

$G_i$ is called a carry generate and it produces a carry 1 when both $A_i$ and $B_i$ are 1, regardless of the input carry $C_i$. $P_i$ is called a carry propagate because it is the term associated with the propagation of the carry from $C_i$ to $C_{i+1}$.

➢ The Boolean functions for the carry outputs of each stage and substitute for each $C_i$ its value from the previous equations,

$C_0$ = input input carry
$C_1 = G_0+P_0C_0$
$C_2 = G_1+P_1C_1$
$= G_1+P_1 (G_0+P_0C_0)$
$= G_1+P_1G_0+P_1P_0C_0$
$C_3 = G_2+P_2C_2$



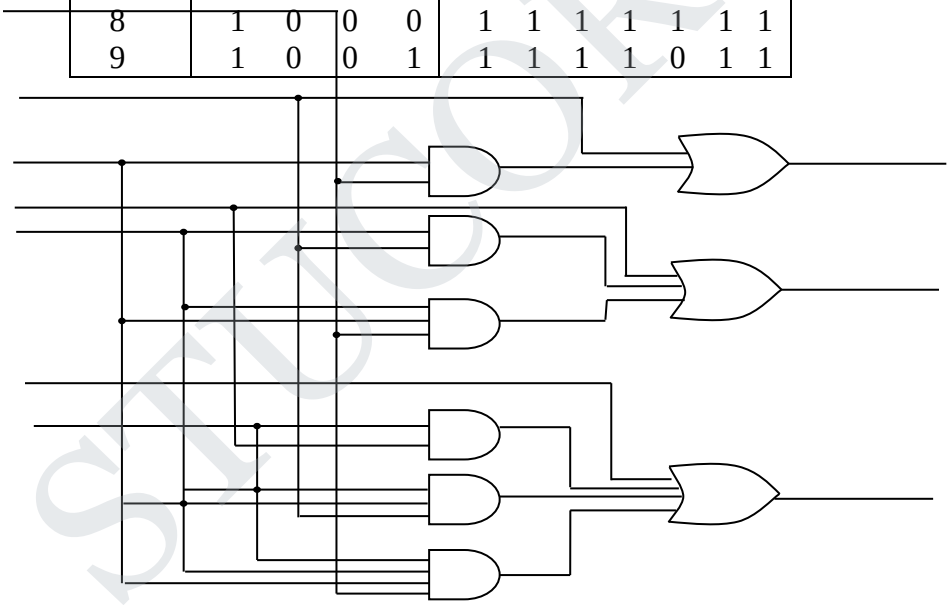
STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

$$= G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$$

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



- Since the boolean function for each output carry is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate (or by two-level NAND).
- The three Boolean functions for $C_1$,$C_2$ and $C_3$ are implemented in the carry look ahead generator shown in the fig. From the circuit it is noted that $C_3$ does not have to wait for $C_2$ and $C_1$ to propagate; since $C_3$ is propagated at the same time as $C_1$ and $C_2$.



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- The construction of a 4-bit adder with a carry look ahead scheme is drawn.Each sum of output requires two exclusive-OR gates. The output of the first exclusive-OR gate generates the $P_i$ variable, and the AND gate generates the $G_i$ variable.
- The carries are propagated through the carry look ahead generator and applied as input inputs to the second exclusive-OR gate. All output carries are generated after a delay through two levels of gates. Thus, outputs $S_1$ through $S_3$ have equal propagation delay times.

## 7. Design a BCD to 7 segment code converter (May/June 2006).

- In most applications, seven segment displays are used to give a visual communication of the output states of digital ICs such as decade counters, latches etc.
- These outputs are usually in four bit BCD form, and are thus not suitable for directly driving seven segment displays.
- Therefore special BCD to seven segment decoder/driver ICs are used to convert the BCD signal into a form suitable for driving these displays.
- If 7-segment display is common anode, the segment driver output must be active low to glow the segment.
- In case of common cathode type 7-segment display, the segment driver output must be active high to glow the segment.

Truth table for common cathode 7-segment display/driver

K – map simplification

| | | | |
|---|---|---|---|
| 1 | | 1 | 1 |
| | 1 | 1 | 1 |
| x | x | x | x |
| 1 | 1 | x | x |

$a = A + C + BD + \overline{B}\overline{D}$

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | | 1 | |
| x | x | x | x |
| 1 | 1 | x | x |

$b = \overline{B} + \overline{C}\overline{D} + CD$

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | |
| 1 | 1 | 1 | 1 |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| x | x | x | x |
| 1 | 1 | x | x |

| | | | |
|---|---|---|---|
| 1 | | 1 | 1 |
| | 1 | | 1 |
| x | x | x | x |
| 1 | 1 | x | x |

$c = B + \overline{C} + D$

$d = \overline{B}\overline{D} + C\overline{D} + B\overline{C}D + \overline{B}C + A$

| | | | |
|---|---|---|---|
| 1 | | | 1 |
| | | | 1 |
| X | x | x | x |
| 1 | | x | x |

| | | | |
|---|---|---|---|
| 1 | | | |
| 1 | 1 | | 1 |
| x | X | x | x |
| 1 | 1 | x | x |

| | | | |
|---|---|---|---|
| | | 1 | 1 |
| 1 | 1 | | 1 |
| x | x | x | x |
| 1 | 1 | x | x |

$e = \overline{B}\overline{D} + C\overline{D}$

$f = A + \overline{C}\overline{D} + B\overline{C} + B\overline{D}$

$g = A + B\overline{C} + \overline{B}C + C\overline{D}$

**Logic Diagram**

STUCOR APP



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**8. Design a BCD to Gray code converter. Use don't cares(Nov/Dec 2004)**

Step 1:

- BCD to gray code converter requires four inputs and four outputs. The four input variables are A, B, C, D and four output variables are $G_3$, $G_2$, $G_1$, $G_0$ **.**

Step 2:

- The relationship between input and output variables are tabulated as,

| BCD Code | | | | Gray Code | | | |
|---|---|---|---|---|---|---|---|
| A | B | C | D | $G_3$ | $G_2$ | $G_1$ | $G_0$ |
| 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 |
| 1 | 0 | 1 | 0 | x | x | x | x |
| 1 | 0 | 1 | 1 | x | x | x | x |
| 1 | 1 | 0 | 0 | x | x | x | x |
| 1 | 1 | 0 | 1 | x | x | x | x |
| 1 | 1 | 1 | 0 | x | x | x | x |
| 1 | 1 | 1 | 1 | x | x | x | x |

Step 3:

Boolean functions interms of input variables are simplified as,

| | | | |
|---|---|---|---|
| | 1 | | 1 |
| | 1 | | 1 |
| X | X | X | X |
| | 1 | X | X |

| | | | |
|---|---|---|---|
| | | 1 | 1 |
| 1 | 1 | | |
| X | X | X | X |
| | | X | X |

$$G_0 = \overline{C}D + C\overline{D} = C \oplus D$$

$$G_1 = B\overline{C} + \overline{B}C = B \oplus C$$

| | | | |
|---|---|---|---|
| | | | |



STUCOR APP

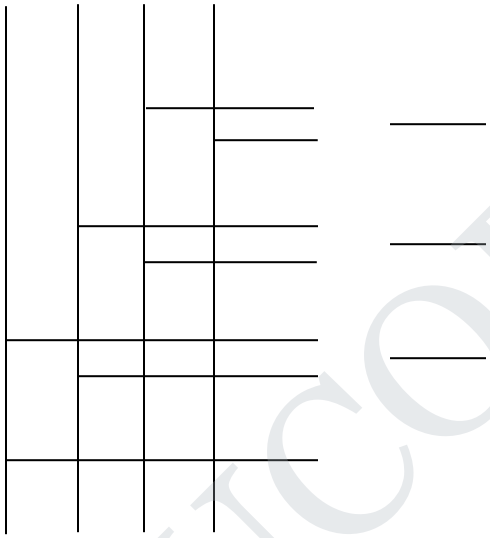
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| X | X | X | X |
| 1 | 1 | X | X |

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| X | X | X | X |
| 1 | 1 | X | X |

Step 4:
**Logic diagram:**



STUCOR APP

---

## 9. Design a 4-bit binary to BCD code converter.(Nov 2005)

- The design of combinational circuits starts from the specification of the problem and ends in a set of Boolean functions from which the logic diagram can be obtained. The procedure involves the following steps:
  1. From the specifications of the circuit, determine the number of inputs and outputs and assign a symbol to each.
  2. Derive the truth table that defines the required relationship between inputs and outputs.
  3. Obtain the simplified Boolean functions for each output as a function of the input variables.
  4. Draw the logic diagram and verify the correctness of the design.

  Step 1:



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- 4-bit binary to BCD code converter requires four inputs and five outputs. The four input variables are A, B, C, D and output variables are designated as $B_4$, $B_3$, $B_2$, $B_1$, $B_0$.

Step 2:

- The relationship between input and output variables are tabulated as,

Truth table for binary to BCD code conversion.

| Binary Code | | | | BCD code | | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | D | $B_4$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Step 3:

- Boolean functions for each output as a function of the input variables are simplified as,

| | | | |
|---|---|---|---|
| | 1 | 1 | |
| | 1 | 1 | |
| | 1 | 1 | |
| | 1 | 1 | |

$B_0 = D$

| | | | |
|---|---|---|---|
| | | 1 | 1 |
| | | 1 | 1 |
| 1 | 1 | | |
| | | | |

$B_1 = AB\overline{C} + \overline{A}C$



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| | | | |
| 1 | 1 | 1 | 1 |
| | | 1 | 1 |
| | | | |

$B_2 = \overline{A}B + BC$

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| 1 | 1 | | |

$B_3 = A\overline{B}\overline{C}$

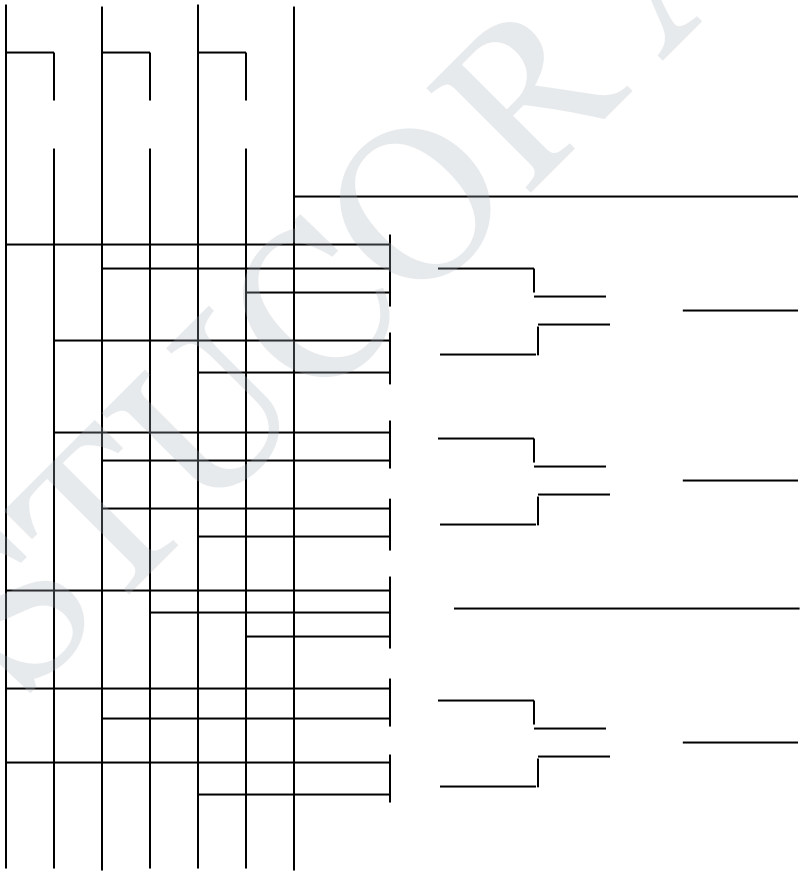
| | | | |
|---|---|---|---|
| | | | |
| | | | |
| 1 | 1 | 1 | 1 |
| | | 1 | 1 |

$B_4 = AB + AC$

Step 4:

- From the Boolean expressions the corresponding logic diagram is drawn as,

**Logi**





STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

## 10. Design a 4-bit binary to gray code converter. (Nov 2005)

Step 1:

- 4-bit binary to gray code converter requires four inputs and four outputs. The input variables are A, B, C, D and output variables are $G_3$, $G_2$, $G_1$, $G_0$.

Step 2:

- The relationship between input and output variables are tabulated as

| Binary Code | | | | Gray Code | | | |
|---|---|---|---|---|---|---|---|
| A | B | C | D | $G_3$ | $G_2$ | $G_1$ | $G_0$ |
| 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 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Step 3:

- Boolean functions are simplified as,

For $G_0$

| | | | |
|---|---|---|---|
| | 1 | | 1 |
| | 1 | | 1 |
| | 1 | | 1 |
| | 1 | | 1 |

$$G_0 = \overline{C}D + C\overline{D} = C \oplus D$$

For $G_1$

| | | | |
|---|---|---|---|
| | | 1 | 1 |
| 1 | 1 | | |
| 1 | 1 | | |
| | | 1 | 1 |

$$G_1 = B\overline{C} + \overline{B}C = B \oplus C$$



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | | | |
|---|---|---|---|
| | | | |
| 1 | 1 | 1 | 1 |
| | | | |
| 1 | 1 | 1 | 1 |

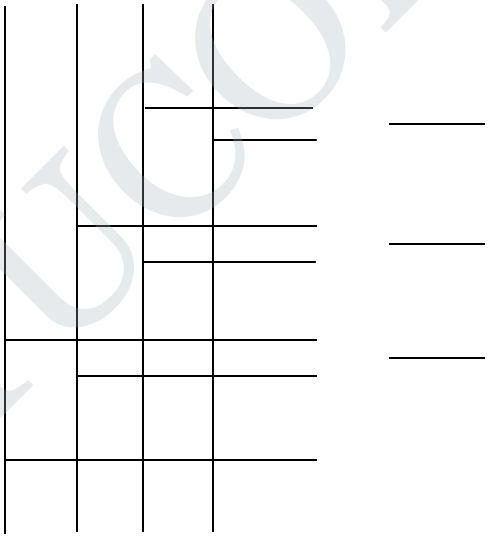
$G_2 = \overline{A}B + A\overline{B} = A \oplus B$

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

$G_3 = D$

Step 4:

- From the Boolean functions the logic diagram is drawn as,





STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

## PART C

## ASSIGNMENT QUESTIONS

**1.Design and explain the working of full adder.(Nov 2009)**

**2.Explain octal to binary encoder.(Nov 2009)**

**3.Design BCD to EX-3 code converter**

**4.Implement the following Boolean function using 4:1 MUX**
**F(A,B,C,D)=Σm(0,1,2,4,6,9,12,14)**

**5.Implement the following Boolean function using 8:1 multiplexer**
**F(A,B,C,D)=A'BD'+ACD+B'CD+A'C'D**

STUCOR APP



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

## UNIT –III

## SYNCHRONOUS SEQUENTIAL CIRCUITS

### PART B

**1. Using D flip-flops, design a synchronous counter which counts in the sequence 000, 001, 010, 011, 100, 101, 110, 111, 000.**

**(Apr / May 2003)**

**State Table:**

- From the given description the state table is derived as,

| Present State | | | Next state | | |
|---|---|---|---|---|---|
| A | B | C | A | B | C |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |

- The Boole g K-map as,

$D_A$ $D_B$ $D_C$

| | | | |
|---|---|---|---|
| | | 1 | |
| 1 | 1 | | 1 |

| | | | |
|---|---|---|---|
| | 1 | | 1 |
| | 1 | | 1 |

| | | | |
|---|---|---|---|
| 1 | | | 1 |
| 1 | | | 1 |

$D_A = \overline{A}BC + A\overline{B} + A\overline{C}$ $D_B = \overline{B}C + B\overline{C}$ $D_C = \overline{C}$

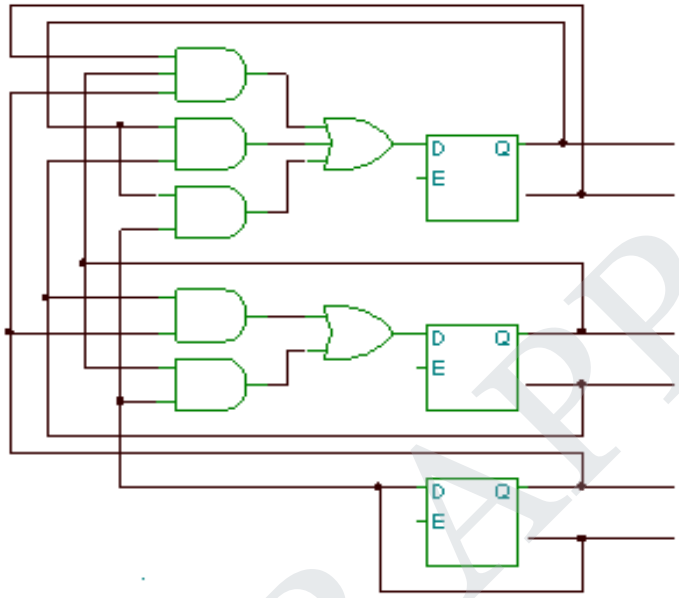


STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Logic Diagram:**




## 2. Using JK flip-flops, design a parallel counter which counts in The sequence 000, 111, 101, 110, 001, 010, 000……

## (Nov 2003)

**State Table:**

- From the given description the state table is derived as,

| Present State | | | Next state | | | Flip-flop Inputs | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | A | B | C | $J_A$ | $K_A$ | $J_B$ | $K_B$ | $J_C$ | $K_C$ |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | x | 1 | x | 1 | x |
| 1 | 1 | 1 | 1 | 0 | 1 | x | 0 | x | 1 | x | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | x | 0 | 1 | x | x | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | x | 1 | x | 1 | 1 | x |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | x | 1 | x | x | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | x | 1 | 0 | x |



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- The minterms 011 & 100 are not included in count sequence and they are treated as don't care conditions.

- The

| | | | |
|---|---|---|---|
| 1 | | x | |
| x | x | x | x |

| | | | |
|---|---|---|---|
| x | x | x | x |
| x | | | 1 |

| | | | |
|---|---|---|---|
| 1 | 1 | x | x |
| X | 1 | x | x |

Boolean expressions are simplified by using K-map as,

**$J_A$** **$K_A$** **$J_B$**

= 1

| | | | |
|---|---|---|---|
| x | x | x | 1 |
| x | x | 1 | 1 |

$K_A = \overline{C}$

$J_A$

| | | | |
|---|---|---|---|
| 1 | x | x | |
| x | x | x | 1 |

| | | | |
|---|---|---|---|
| | | 1 | |
| 1 | 1 | | 1 |

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

$J_B$

**$K_B$** **$J_C$** **$K_C$**

$K_B = 1$ $J_C = \overline{B} + A$ $K_C = \overline{B}$

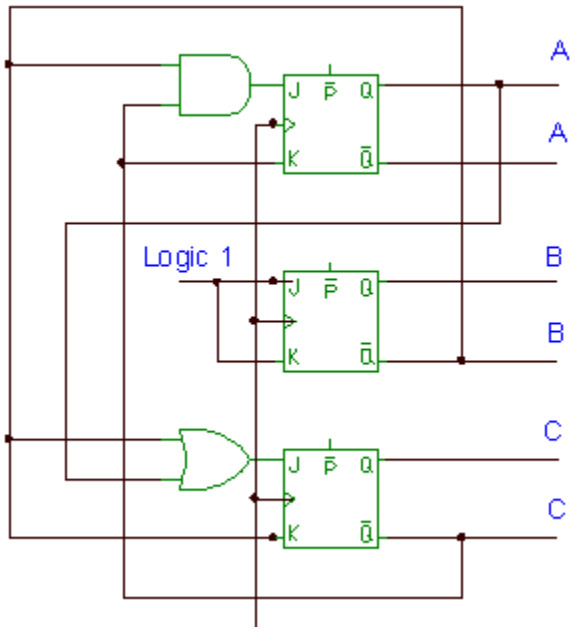
**Logic diagram:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP



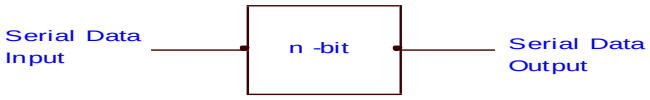

### 3. Draw a 4 bit serial in serial out shift register . (May 2006)

- Shift registers are classified depending upon the way in which data is enabled and retrieved. They are
  - i. Serial In Serial Out (SISO)
  - ii. Serial In Parallel Out (SIPO)
  - iii. Parallel In Serial Out (PISO)
  - iv. Parallel In Parallel Out (PIPO)

**Serial In Serial Out (SISO)**

- Input data is applied one bit at a time to the first flip flop in a chain and read out from the last flip flop in a chain one bit at a time.

**Block Diagram (SISO):**



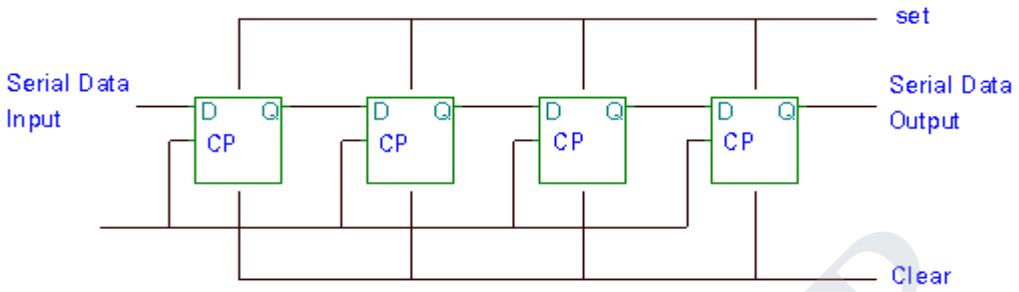

- Registers can be designed using discrete flip flops( S R, JK and D FF).



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- Data may be shifted left (from low to higher order bits) using shift left register or shift right (from high to low order bits) using shift right register.




- D FF are connected in serial in serial out passion.
- The output of Q of one FF is connected to the D input of the next FF.
- CLK,PRE', and CLR' signal are connected in parallel to all four FFs, so that they are all clocked, all set or reset at the same time.
- When the clock pulse is given the data are shifted to the right.

## 4. Draw a 4 bit serial in parallel out (SIPO) and explain its operation.

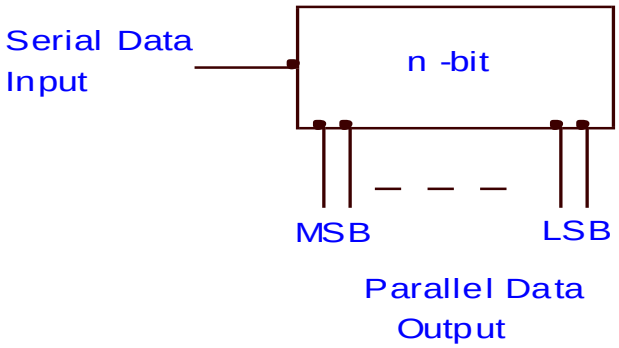
- The second type of register is one in which data is shifted in serially, but shifted out in parallel.
- In order to shift the data out in parallel it is simply necessary to have all the data available as outputs at the same time.
- This is done by connecting the output of each flip flop to an output pin.

**Block Diagram:**

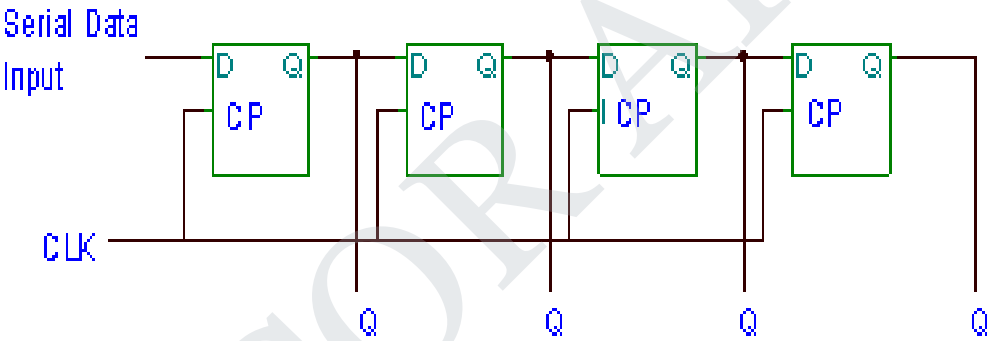


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




**A Serial In Parallel Out shift register:**




- Input data is applied one bit at a time to the D input of the first flip flop in a chain and read out from the Q outputs in parallel after a data word is all shifted in.

**5. Draw a 4 bit parallel in serial out shift register and explain briefly. Or Explain the parallel load serial out data transfer operation in a 4-bit shift register. (May 2006)**

**Parallel In Serial Out:**

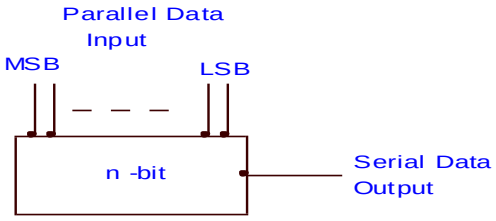
- In a parallel in serial out shift registers, the bits are entered simultaneously into their respective flip flops.

**Block Diagram:**



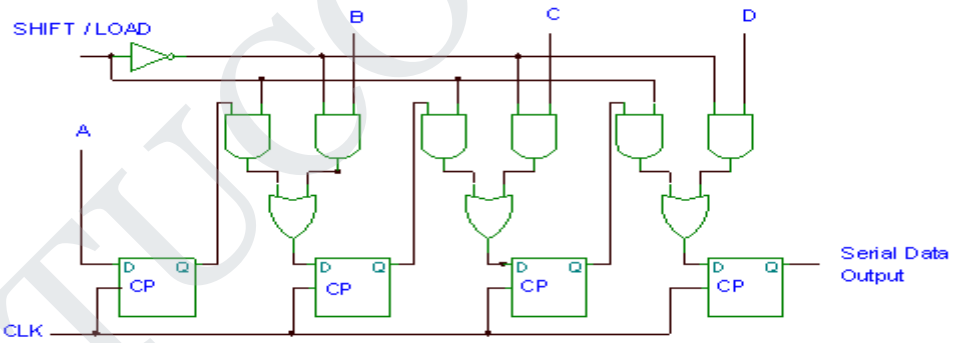
STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




**Circuit diagram:**

- It has four input data lines A, B, C and D and a shift / LOAD' input that allows four bits data to be entered into the shift register in a parallel fashion.
- When shift/LOAD' is low, gates $G_1$, $G_2$ and $G_3$ are enabled, allowing each data bit applied to the D input of the respective flip flops.
- When a clock pulse is obtained, the flip flops store all bits simultaneously.
- When shift/LOAD' is high, gates $G_4$, $G_5$ and $G_6$ are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the shift/LAOD' control signal.




## 6.Explain in detail PIPO (Parallel In Parallel Out) shift register.

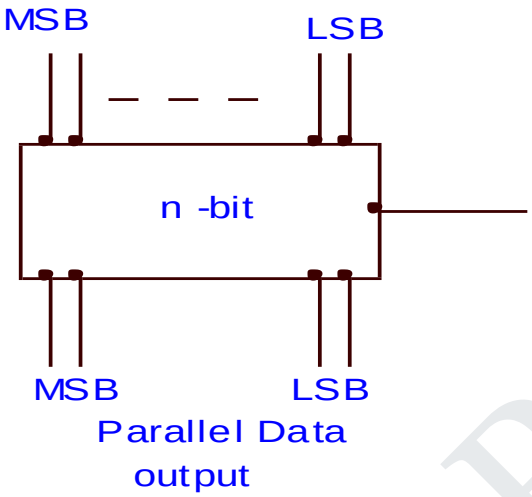
- In PIPO bits are enabled simultaneously into their respective flip flops and the Q output are read out in parallel.

**Block Diagram:**



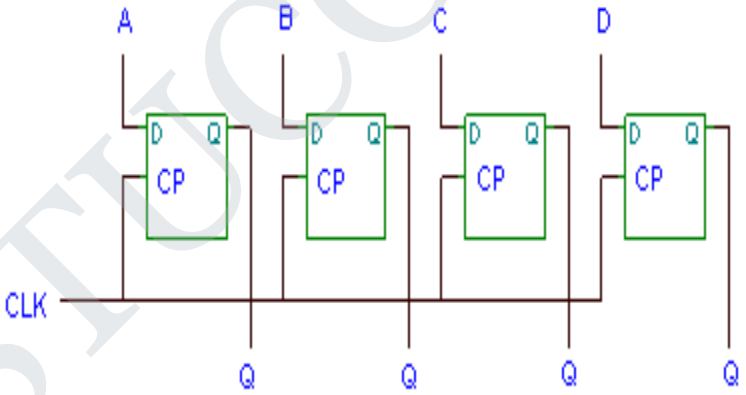
STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




- This type of register is simply used to store data, and is sometimes called a data register or data latch.

**Logic Diagram:**




## 7.Explain the operation of SR latch using NOR gates.

**NOR GATE:**

- SR latch is a circuit with two cross coupled NOR gates or two cross coupled
- NAND gates.
- It has two inputs labeled s for Set and R for Reset.
- The latch has two states,

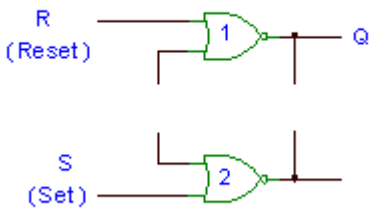


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- When o/p Q =1 and Q'= 0, it is said to be in set state.
- When o/p Q =0 and Q'= 1, it is said to be in reset state.
- When both inputs are equal to 1 at the same time, an undefined state with both outputs equal to 0 occurs.
- When both inputs are equal to 0 at the same time, there is no change in the output.

**Logic Diagram:**




**Function Table:**

| S | R | Next State | |
|---|---|---|---|
| | | Q(t) | Q'(t) |
| 0 | 0 | Q(t) | Q'(t) |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**Case (i): S = R = 0**

- When S=R=0, the next state doesn't change, it remains in the previous state.
- When the input, S = 0 and R = 0 is applied, assume Q = 0 and Q'=1, the output of NOR 1 becomes zero since its input, R=0 and Q'=1. The output of NOR gate 2 becomes 1, since its input S=0 and Q=0. Thus the circuit remains in the reset state.
- Similarly when the input S=0 and R=0 is applied with Q=1 and Q'=0(set state), output of NOR 2 becomes zero since its input, S =0 and Q =1. The output of NOR gate 1 becomes 1, since its input R=0 and Q'=0. Thus the circuit remains in the set state.

**Case (ii) S = 0 and R = 1**

- When the input S=0 and R=1 is given, the output of NOR gate1, i.e. is Q becomes zero, which is fed back to the input of NOR gate 2 (with S=0) the output of Q' becomes 1. The circuit is said to be in the reset state (Q=0 and Q' = 1).

**Case (iii) S =1 and R = 0**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- When the input S=1 and R=0 is given, the output of NOR gate2, i.e. is Q' becomes zero, which is fed back to the input of NOR gate 1(with R=0) the output of Q becomes 1. The circuit is said to be in the set state (Q=1 and Q' = 0).

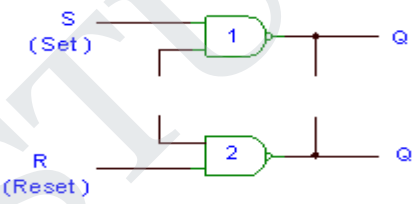
**Case (iv) S=1 and R=1**

- The logical operation of NOR gate is such that when any one of the input is high, the output will become zero. Therefore when the input S=1 R=1 is given, the output of both gate becomes zero which is undefined state.

## 8. Realize a SR flip-flop using NAND gates and explain its operation. (Apr 2003)

- The inputs signal for NAND circuit requires the complement of values used for NOR latch. Therefore it is referred to as S'-R' latch.
- When the input of NAND gate is S=1 and R=1 the output doesn't change. It remains in the previous state (No Change).
- When the input of NAND gate is S=0 and R=0 the output of NAND gates becomes HIGH which is defined.

**Logic Diagram:**




**Function Table:**

| S | R | Next State | |
|---|---|---|---|
| | | Q(t) | Q'(t) |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | Q(t) | Q'(t) |

**Case (i): S = R = 0**

- The logical operation of NAND gate is such that when any one of the input is low(0), the output will becomes high. Therefore when the input S=0 R=0 is given, the output of both gate becomes high which is undefined state.

**Case (ii) S=0 and R=1**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- When the input S=0 and R=1 is given, the output of NAND gate1,i.e. is Q becomes high, which is fed back to the input of NAND gate 2 (with R=1) the output of Q' becomes 0. The circuit is said to be in the set state (Q = 1 and Q' = 0).

**Case (iii) S=1 and R= 0**

- When the input S=1 and R=0 is given, the output of NAND gate2, i.e. is Q' becomes high, which is fed back to the input of NAND gate 1(with R=0) the output NAND gate becomes 0. The circuit is said to be in the reset state (Q=0 and Q' = 1).

**Case (iv) S=1 and R=1 (No change)**

- When S=R=1, the next state doesn't change, it remains in the previous state.
- When the input, S = 1 and R = 1 is applied, assume Q = 0 and Q'=1, the output of NAND 2 becomes high since its input, R=1 and Q=0. The output of NAND gate 1 becomes 1, since its input S=1 and Q=0. Thus the circuit remains in the reset state.
- Similarly when the input S=1 and R=1 is applied with Q=1 and Q'=0(set state), output of NAND 1 becomes high, since its input are S =1 and Q' =0. The output of NAND gate 2 becomes 0, since its input R=1 and Q=1. Thus the circuit remains in the set state.

**SR latch with Control Input:**

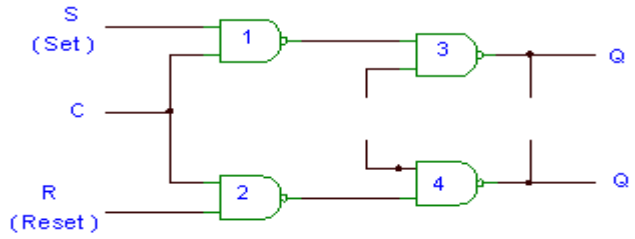
- SR latch with control input determines when the state of the latch can be changed.
- It consists of the basic SR latch and two additional NAND gates.
- The control input C acts as an enable signal for the other two inputs.
- The output of the NAND gates stays at the logic 1 level as long as the control input remains at 0. This is the quiescent condition for the SR latch.

**Circuit diagram:**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




- When the control input goes to 1, information from the S or R input is allowed to effect the SR latch.

| C | S | R | Next State of Q |
|---|---|---|---|
| 0 | X | x | No change –Quiescent condition |
| 1 | 0 | 0 | No change |
| 1 | 0 | 1 | Q = 0, Reset State |
| 1 | 1 | 0 | Q = 1,Set state |
| 1 | 1 | 1 | Indeterminate State |

- To change to the reset state, the inputs must be S=0, R=1 and C=1. In either case, when C returns to 0, the circuit remains in it's current state. Control input disables the circuit by applying 0 to C, so that the state of the output doesn't change regardless of the values of S and R.
- When C=1 and both the S and R inputs are equal to 0, the state of the circuit doesn't change.
- An indeterminate state occurs when all the three inputs are equal to 1. This condition places 0's on both inputs of the basic SR latch, which places it in the undefined state.

**Advantages:**

- It is an important circuit because other latches and flip-flops are constructed from it.

**Disadvantages:**

- In the indeterminate state, when the control input goes back to 0, one cannot conclusively determine the next state as it depends on whether the S or R input goes to zero first. This makes it difficult to manage and it is seldom used in practical.

## 9. Explain the operation of D latch with diagram.

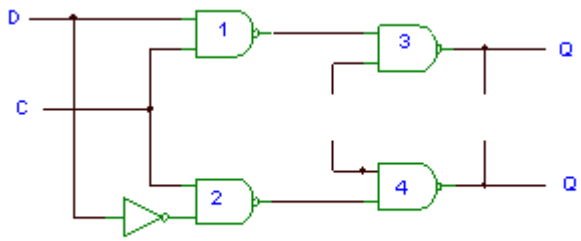


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- The undesirable condition of the indeterminate state in the SR latch is eliminated by making S and R are never equal to 1 at the same time. This is done in the D latch.
- D latch has only two inputs D (data) and C (control).

**Logic diagram:**




**Function table:**

| C | D | Next state of Q |
|---|---|---|
| 0 | x | No change |
| 1 | 0 | Q=0, Reset state |
| 1 | 1 | Q=1, Set State |

- The D input goes directly to the S input and its complement is applied to the R input.
- As long as the control input is at zero, the cross coupled SR latch has both inputs at the 1 level and the circuit cannot change state regardless of the value of D.
- The D input is sampled when C=1. if D=1 the Q output goes to 1, placing the circuit in the SET state .
- If D =0 the output Q goes to zero placing the circuit in the RESET state.
- The D latch hold data in its internal storage it is suited for use as a temporary storage for binary information between a unit and its environment.
- The binary information present at the data input of the D latch is transferred to the Q output when the control input is enabled. This situation provides the path from input D to the output and for this reason the circuit is often called a transparent latch.

## 10. Explain the working of Master-slave flip-flop.

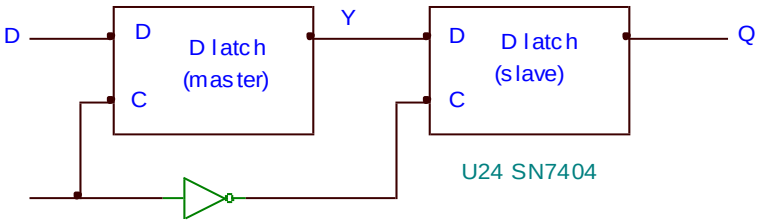


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Master Slave D Flip-Flop:**

- It consists of two D latches and an inverter.
- The first latch is called the master and the second the slave.




- The circuit samples the D input and changes its output Q only at the negative edge of the controlling clock (designated as CLK).
- When the clock is 0, the output of the inverter is 1.
- The slave latch is enabled and its output Q is equal to the master output Y.
- The master latch is disabled because CLK =0 .When the input changes to the logic 1 level, the data from the external d input is transferred to the master.
- The Slave input is disabled as long as the clock remains in the 1 level because its C input is equal to 0.
- Any change in the input changes the master output at Y but cannot effect the slave output.
- When the pulse returns to 0, the master is disabled and is isolated from the D input.
- At the same time, the slave is enabled and the value of Y is transferred to the output of the flip flop at Q. Thus the output of the flip-flop can change only during the transition of the clock from 1 to 0.
- Master Slave flip flop dictates that the output may change only during the negative edge of the clock.
- It is also possible to design the circuit so that the flip flop output changes on the positive edge of the clock.
- This happens in a flip flop that has an additional inverter between the CLK terminal and the junction between the other inverter and input C of the master latch.



STUCOR APP
DOWNLOADED FROM STUCOR APP

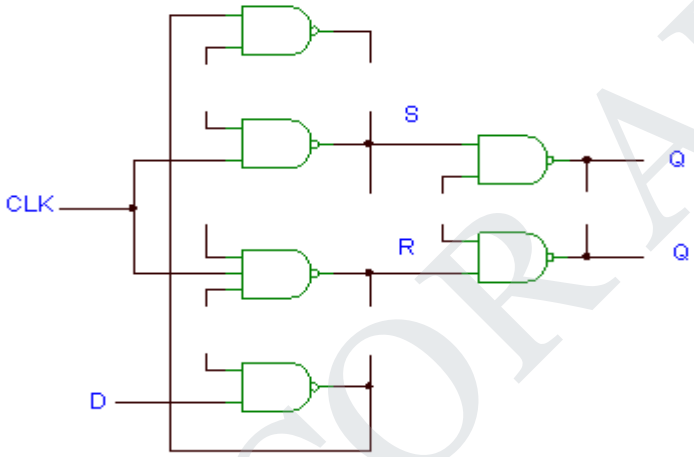
DOWNLOADED FROM STUCOR APP

## 11. Explain the functional operation of D-type edge triggered flip-flop by giving its logical diagram and truth table. (May 2006)

**Edge-triggered D flip flop:**

- Edge triggered D flip-flop uses three S-R latches.
- Two latches respond to the external D (data) and CLK(clocks) inputs.
- The third latch provides the outputs for the flip flop.
- The S and R inputs of the output latch are maintained at logic 1 level when CLK = 0.

**Logic diagram:**




- This causes the output to remain in its present state.
- Input D may equal to 0 or 1.
- If D = 0 when CLK becomes 1, R changes to 0. This causes the flip flop to go to the reset state, making Q = 0.
- If there is a change in the D input while CLK = 1, terminal remains at 0. Thus the flip flop is locked out and is unresponsive to further changes in the input.
- When the clock returns to 0, R goes to 1, placing the output latch in the quiescent condition without changing the output. Similarly, if D = 1 when CLK goes from 0 to 1, S changes to 0. This causes the circuit to go to the set state making Q = 1. Any change in D while CLK =1 does not affect the output.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

- When the input clock in the positive-edge triggered flip flop makes a positive transition, the value of D is transferred to Q. A negative transition from 1 to 0 does not affect the output.

## 12. Write short notes on the following flip flops.

(i) JK Flip flop

(ii) D Flip flop

(iii)T Flip flop.

**JK Flip Flop:**

- D flip flop can be used to construct other flip flops.
- There are three operations that can be performed with a flip flop
  - Set it to 1
  - Reset it to 0
  - Complement its output.
- The JK flip flop performs all the three operations.
  - **Operation:**
- The J input sets the flip flop to 1, the K input resets it to 0, and when both inputs are
  - enabled the output is complemented.
- It is made by investigating the circuit applied to the D input;
  - D = JQ'+K'Q
- When J=1 and K= 0, D = Q' + Q =1,So the next clock edge sets the output
  - to 1.
- When J=0 and K=1, D=0, so the next clock edge resets the output to 0.
- When both J=K=1, D=Q', the next clock edge compliments the output.
- When both J=K=0, D=Q the clock edge leaves the output unchanged.
- **Characteristics Table:**
- A characteristic table defines the logical properties of flip flop by describing its operation in a tabular form

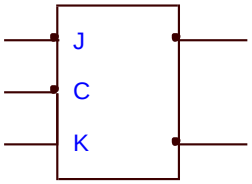


STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Graphic symbol**




**JK FLIP FLOP**

| J | K | Next State | |
|---|---|---|---|
| | | Q(t) | Q'(t) |
| 0 | 0 | Q(t) | Q'(t) |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | Q'(t) | Q(t) |

- From the table, the next state is equal to the present state when inputs J&K=0
- i.e. .Q (t+1) = Q (t).

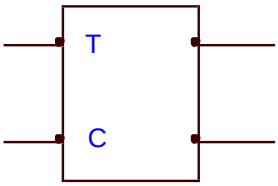
**Characteristic Equation:**

- The logical properties of a flip flop as described in the characteristic table can be expressed algebraically with a characteristic equation is,
- Q (t+1) = JQ'+K'Q

**T FLIP FLOP (toggle)**

- T flip flop is a complementing flip flop and can be obtained from a JK flip flop when inputs J and K are tied together.
- When T =0 (J=K=0) a clock edge does not change the output.

**Graphic Symbol:**



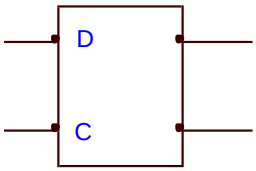

- When T=1 (J=K=1) a clock edge complements the output.
- The complementing flip flop is useful for designing counter.
- The T flip flop can be constructed with a D flip flop and an exclusive OR gate. The expression for D input is D = T + Q = TQ'+T'Q



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Graphic Symbol:**




**Characteristic Table:**

| T | Q(t+1) |
|---|---|
| 0 | Q(t) |
| 1 | Q'(t) |

**D FLIP FLOP:**

- In a D flip flop input is transferred to the output.
- The next state of a D flip flop is depends only on the D input and independent of the present state.
- Next value is equal to the value of D. It doesn't have a no change condition.

**Characteristic table:**

| D | Q(t+1) |
|---|---|
| 0 | 0 |
| 1 | 1 |

- The characteristic equation of D flip flop can be written as,

$$Q(t+1) = D$$

## 13. Explain the operation of asynchronous reset with diagram?

**Direct Inputs:**

- Some flip flops have asynchronous inputs that are used to force the flip flop to a particular state independent of clock
- **The input that sets the flip flop to 1 is called preset or direct set.**
- **The input that clears the flip flop to 0 is called clear or direct reset.**
- When power is turned on in a digital system the state of the flip flop is unknown.
- The direct inputs are useful for bringing all flip flops in the system to a known state prior to the clocked operation.



STUCOR APP
DOWNLOADED FROM STUCOR APP

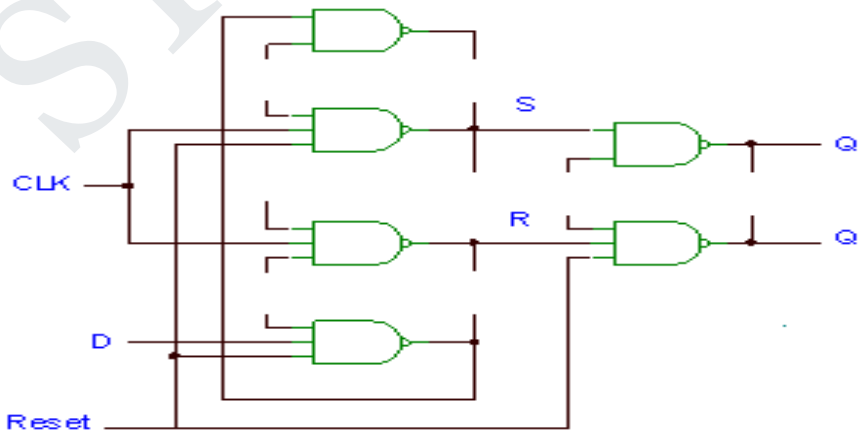
DOWNLOADED FROM STUCOR APP

- A positive edge triggered D flip flop with asynchronous reset is similar to the edge triggered D FF except that it has additional reset input connections to three NAND gates.
- When the reset input is 0, it forces output Q' to stay at 1, which in turn, clears output Q to 0, thus resetting the flip flop.
- Two other connections from the reset input ensure that the S input of the third SR latch stays at logic 1, while the reset input is at 0 regardless of the values of D and CLK.
- The graphic symbol for the D flip flop with a direct reset has an additional input marked with R. The bubble along the input indicates that the reset is active at the logic 0 level.
- Flip flops with a direct set use the symbol S for the asynchronous set input.

**Operation:**

- Function table specifies the operation of the circuit.
- When R =0, the output is reset to 0. This state is independent of the values of D or C.
- Normal clock operation can proceed only after the reset input goes to logic 1.
- The clock at C is shown with upward arrow to indicate that the flip flop triggers on the positive edge of the clock
- The value in D is transferred to Q with every positive-edge clock signal, provided that R=1.

**Circuit Diagram:**






STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**14. Design a binary counter using T flip-flops to count pulses in the following sequence**

**(i) 000, 001, 010, 011, 100, 101, 110, 111, 000**

**(ii) 000, 100, 111, 010, 011, 000** **Apr/May 2004**

**State Table:**

State table for the given description can be derived as,

| Present State | | | Next state | | | Flip-Flop Inputs | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | A | B | C | $T_A$ | $T_B$ | $T_C$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

The Boolean expressions are simplified by using K-map as,

$T_A$

| | | | |
|---|---|---|---|
| | | 1 | |
| | | 1 | |

$T_A = BC$

$T_B$

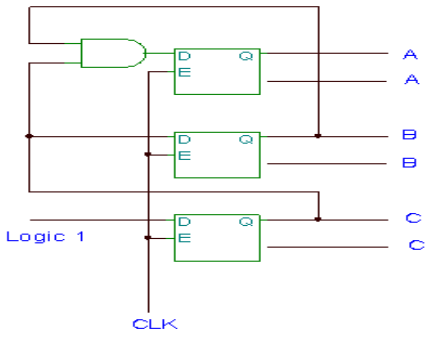
| | | | |
|---|---|---|---|
| | 1 | 1 | |
| | 1 | 1 | |

$T_B = C$

$T_C$

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

$T_C = 1$






STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**ii) 000, 100, 111, 010, 011, 000**

**State Table:**

State table for the given description can be derived as,

| Present State | | | Next state | | | Flip-Flop Inputs | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | A | B | C | $T_A$ | $T_B$ | $T_C$ |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

The Boolean expressions are simplified by using K-map as,

$T_A$

| | | | |
|---|---|---|---|
| | x | 1 | |
| 1 | x | | x |

$T_A = \overline{A}\overline{B} + AB$

$T_B$

| | | | |
|---|---|---|---|
| 1 | x | | |
| | x | 1 | x |

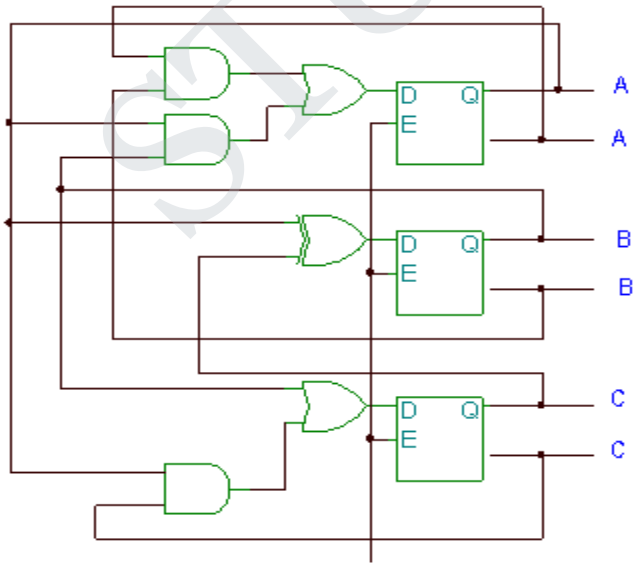
$T_B = \overline{A}C + A\overline{C}$

$T_C$

| | | | |
|---|---|---|---|
| | x | 1 | 1 |
| 1 | x | 1 | x |

$T_C = B + A\overline{C}$

**Logic Diagram:**






STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**PART C**

## ASSIGNMENT QUESTIONS

**1. Draw the schematic diagram of master slave JKFF and input and output waveforms. Discuss how it prevents race around condition.**

**2. Draw an asynchronous 4 bit down counter and explain its working.**

**3. Draw a synchronous 4 bit up-down counter and explain its working.**

**4. Mention the two models in sequential circuits and distinguish between them.**

**5. Explain the operation of 4 bit shift register.**

STUCOR APP



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

## UNIT –IV

## ASYNCHRONOUS SEQUENTIAL CIRCUITS

### PART B

**1. Write a note on Analysis of Sequential Circuits**

**The behavior of a sequential circuit is determined from the inputs, the outputs and the states of its flip-flops. Both the output and the next state are a function of the inputs and the present state.**

**The suggested analysis procedure of a sequential circuit is**

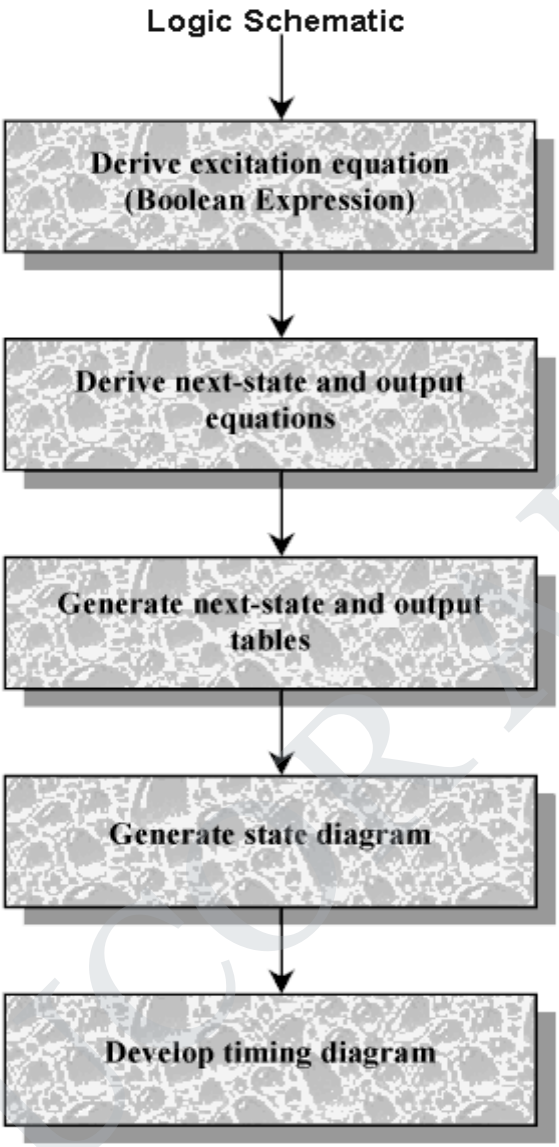
STUCOR APP



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




**We start with the logic schematic from which we can derive excitation equations for each flip-flop input. Then, to obtain next-state equations, we insert the excitation equations into the characteristic equations. The output equations can be derived from the schematic, and once we have our output and next-state equations, we can generate the next-state and output tables as well as state diagrams. When we reach this stage, we use either the table or the state diagram to develop a timing diagram which can be verified through simulation.**

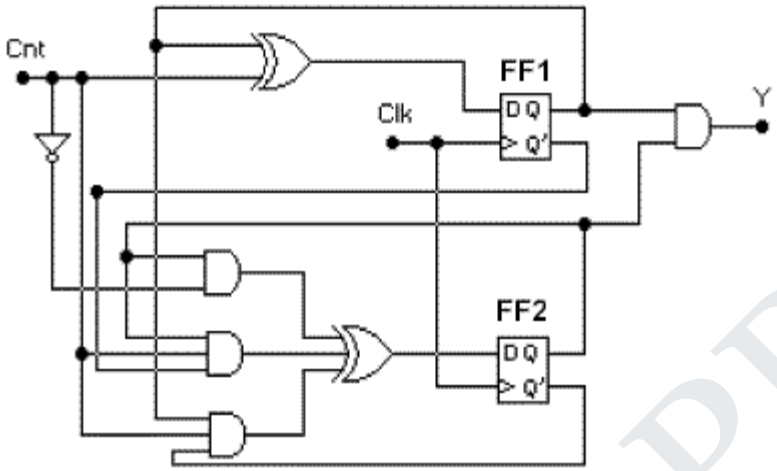
**Example**

**Derive the next state, the output table and the state diagram for the sequential circuit shown**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




**SOLUTION:**

**Excitation equations:**

**D0 = Cnt⊕Q0 = Cnt'*Q0 + Cnt*Q0'**

**D0 = Cnt'*Q1 + Cnt*Q1'*Q0 + Cnt*Q1*Q0'**

**Next-state equations:**

**Q0(next) = D0 = Cnt'*Q0 + Cnt*Q0'**

**Q1(next) = D0 = Cnt'*Q1 + Cnt*Q1'*Q0 + Cnt*Q1*Q0'**

**In addition, however, we have computed the output equation.**

**Output equation: Y = Q1Q0**

**As this equation shows, the output Y will equal to 1 when the counter is in state Q1Q0 = 11, and it will stay 1 as long as the counter stays in that state.**



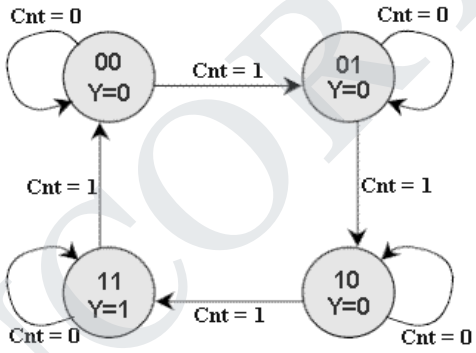
STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

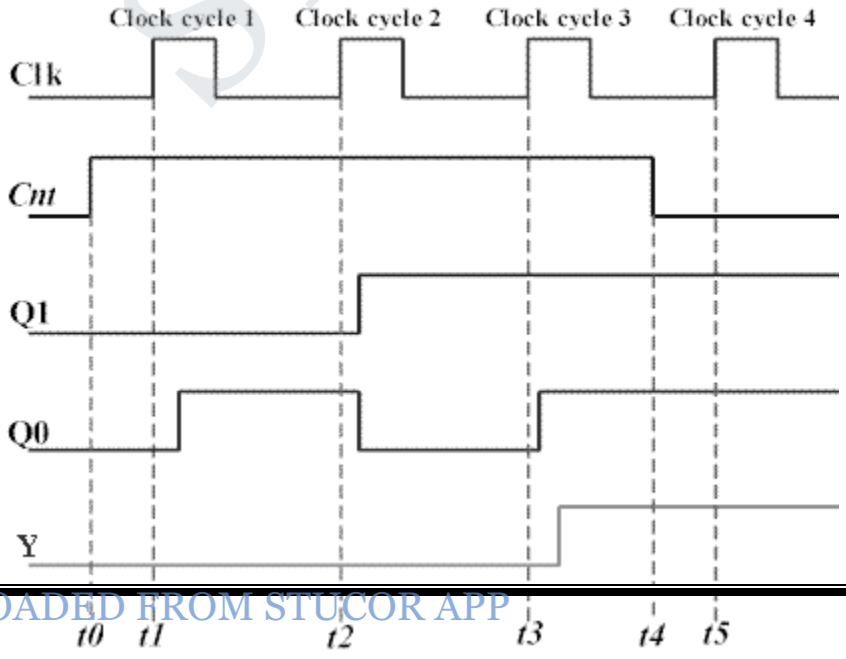
**Next-state and output table:**

| Present State Q1 Q0 | Next State Cnt=0 | Next State Cnt=1 | Output Z |
|---|---|---|---|
| 0 0 | 0 0 | 0 1 | 0 |
| 0 1 | 0 1 | 1 0 | 0 |
| 1 0 | 1 0 | 1 1 | 0 |
| 1 1 | 1 1 | 0 0 | 1 |

**State diagram:**




**Timing diagram:**




STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

Note that the counter will reach the state Q1Q0 = 11 only in the third clock cycle, so the output Y will equal 1 after Q0 changes to 1. Since counting is disabled in the third clock cycle, the counter will stay in the state Q1Q0 = 11 and Y will stay asserted in all succeeding clock cycles until counting is enabled again.

**2. Write in detail about Design of Sequential Circuits**

The design of a synchronous sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which a logic diagram can be obtained. In contrast to a combinational logic, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalence representation, such as a state diagram.

A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding the combinational structure which, together with the flip-flops, produces a circuit that fulfils the required specifications. The number of flip-flops is determined from the number of states needed in the circuit.

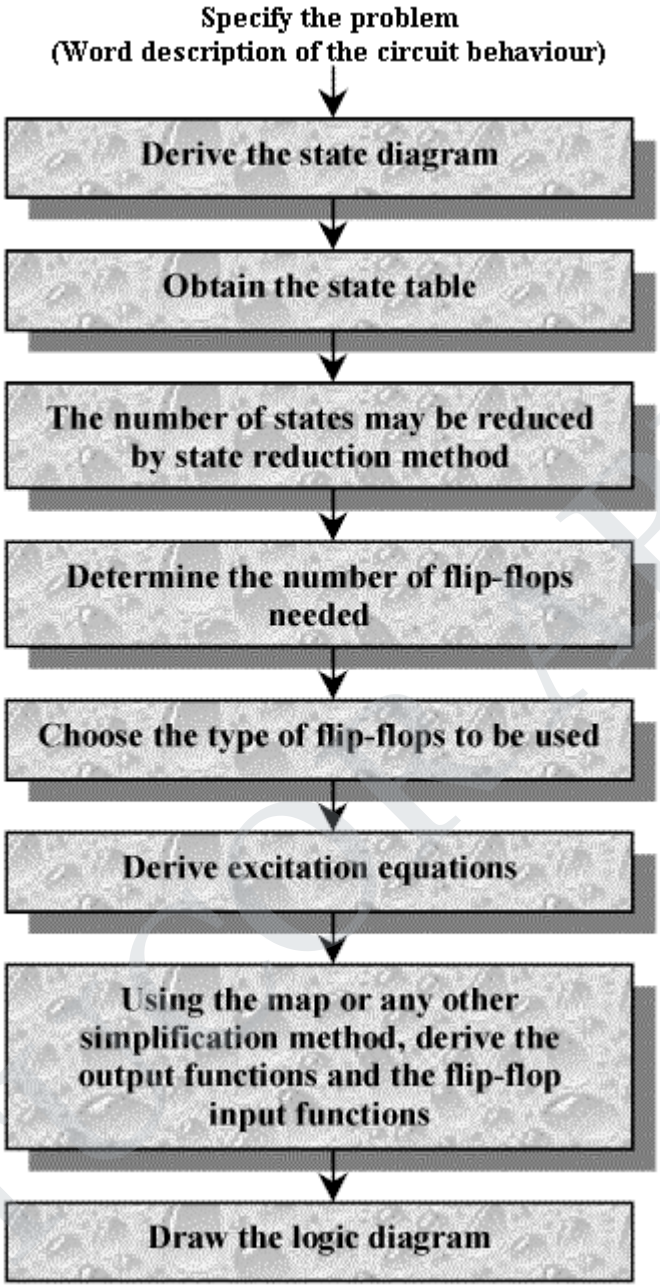
**The recommended steps for the design of sequential circuits are set out below.**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP



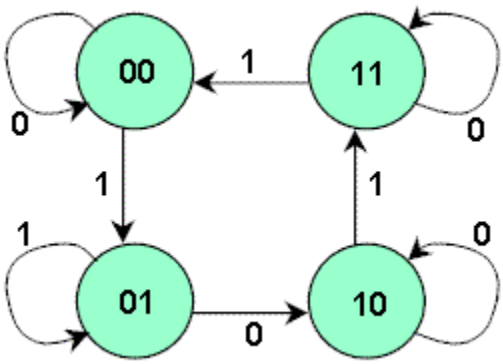

**Example:**

**Design a synchronous sequential circuit whose state diagram is shown. The type of flip-flop to be use is J-K.**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




**From the state diagram, we can generate the state table. Note that there is no output section for this circuit. Two flip-flops are needed to represent the four states and are designated Q0Q1. The input variable is labelled x.**

**State table.**

| Present State | Next State | |
|---|---|---|
| Q0 Q1 | x = 0 | x = 1 |
| 0 0 | 0 0 | 0 1 |
| 0 1 | 1 0 | 0 1 |
| 1 0 | 1 0 | 1 1 |
| 1 1 | 1 1 | 0 0 |

**We shall now derive the excitation table and the combinational structure. The table is now arranged in a different form, where the present state and input variables are arranged in the form of a truth table. Remember, the excitable for the JK flip-flop was derive in**

**Excitation table for JK flip-flop**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| Output Transitions Q □Q(next) | Flip-flop inputs J K |
|---|---|
| 0 □ 0 | 0 X |
| 0 □ 1 | 1 X |
| 1 □ 0 | X 1 |
| 1 □ 1 | X 0 |

**Excitation table of the circuit**

| Present State Q0 Q1 | Next State Q0 Q1 | Input x | Flip-flop Inputs J0K0 | J1K1 |
|---|---|---|---|---|
| 0 0 | 0 0 | 0 | 0 X | 0 X |
| 0 0 | 0 1 | 1 | 0 X | 1 X |
| 0 1 | 1 0 | 0 | 1 X | X 1 |
| 0 1 | 0 1 | 1 | 0 X | X 0 |
| 1 0 | 1 0 | 0 | X 0 | 0 X |
| 1 0 | 1 1 | 1 | X 0 | 1 X |
| 1 1 | 1 1 | 0 | X 0 | X 0 |
| 1 1 | 0 0 | 1 | X 1 | X 1 |

**In the first row, we have a transition for flip-flop Q0 from 0 in the present state to 0 in the next state. In Table 10 we find that a transition of states from 0 to 0 requires that input J = 0 and input K = X. So 0 and X are copied in the first row under J0 and K0 respectively. Since the first row also shows a transition for the flip-flop Q1 from 0 in the present state to 0 in the next state, 0 and X are copied in the first row under J1 and K1. This process is continued for each row of the table and for each flip-flop, with the input conditions as specified in Table 10.**

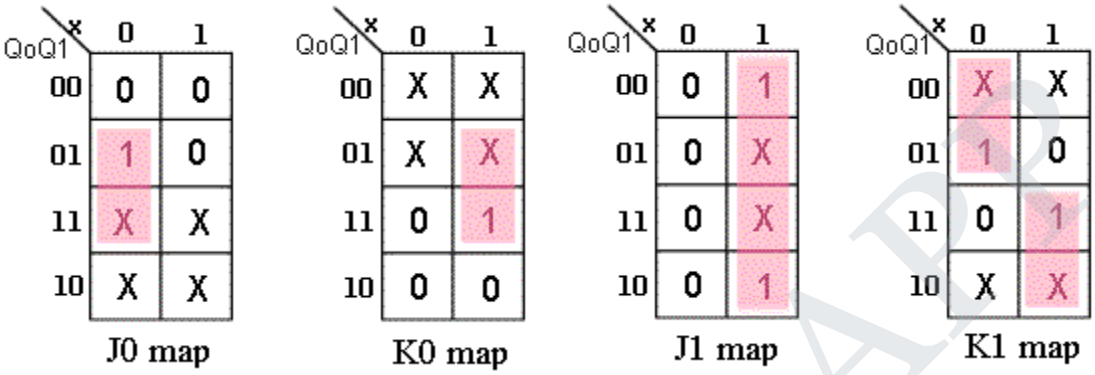


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**The simplified Boolean functions for the combinational circuit can now be derived. The input variables are Q0, Q1, and x; the output are the variables J0, K0, J1 and K1. The information from the truth table is plotted on the Karnaugh maps**

**Karnaugh Maps**

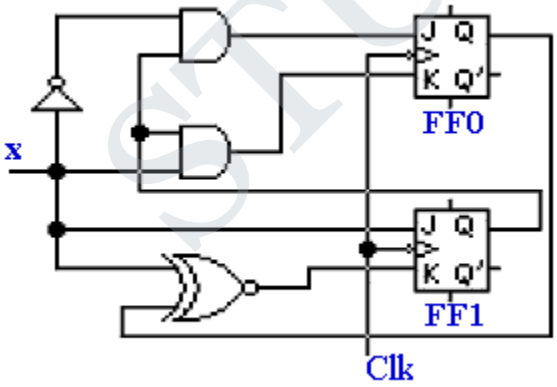



**The flip-flop input functions are derived:**

**J0 = Q1*x'** **K0 = Q1*x**

**J1 = x** **K1 = Q0'*x' + Q0*x = Q0 x**

**The logic diagram is drawn**




**3. Design a sequential circuit whose state tables are specified, using D flip-flops.**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**State table of a sequential circuit.**

| Present State | Next State | | Output | |
|---|---|---|---|---|
| Q0 Q1 | x = 0 | x = 1 | x = 0 | x = 1 |
| 0 0 | 0 0 | 0 1 | 0 | 0 |
| 0 1 | 0 0 | 1 0 | 0 | 0 |
| 1 0 | 1 1 | 1 0 | 0 | 0 |
| 1 1 | 0 0 | 0 1 | 0 | 1 |

**Excitation table for a D flip-flop.**

| Output Transitions | Flip-flop inputs |
|---|---|
| Q□Q(next) | D |
| 0 □ 0 | 0 |
| 0 □ 1 | 1 |
| 1 □ 0 | 0 |
| 1 □ 1 | 1 |

**Next step is to derive the excitation table for the design circuit, which is. The output of the circuit is labelled Z.**

| Present State | Next State | Input | Flip-flop Inputs | Output |
|---|---|---|---|---|



STUCOR APP
DOWNLOADED FROM STUCOR APP

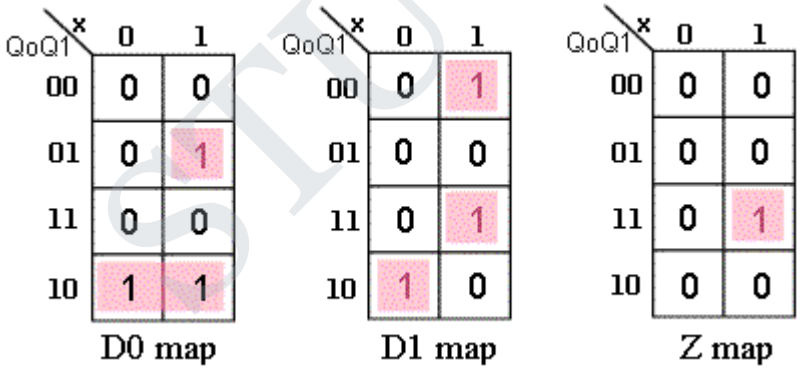
DOWNLOADED FROM STUCOR APP

| Q0 Q1 | Q0 Q1 | x | D0 | D1 | Z |
|---|---|---|---|---|---|
| 0 0 | 0 0 | 0 | 0 | 0 | 0 |
| 0 0 | 0 1 | 1 | 0 | 1 | 0 |
| 0 1 | 0 0 | 0 | 0 | 0 | 0 |
| 0 1 | 1 0 | 1 | 1 | 0 | 0 |
| 1 0 | 1 1 | 0 | 1 | 1 | 0 |
| 1 0 | 1 0 | 1 | 1 | 0 | 0 |
| 1 1 | 0 0 | 0 | 0 | 0 | 0 |
| 1 1 | 0 1 | 1 | 0 | 1 | 1 |

**Excitation table**

**Now plot the flip-flop inputs and output functions on the Karnaugh map to derive the Boolean expressions, which is shown**

**Karnaugh maps**




**The simplified Boolean expressions are:**



STUCOR APP
DOWNLOADED FROM STUCOR APP

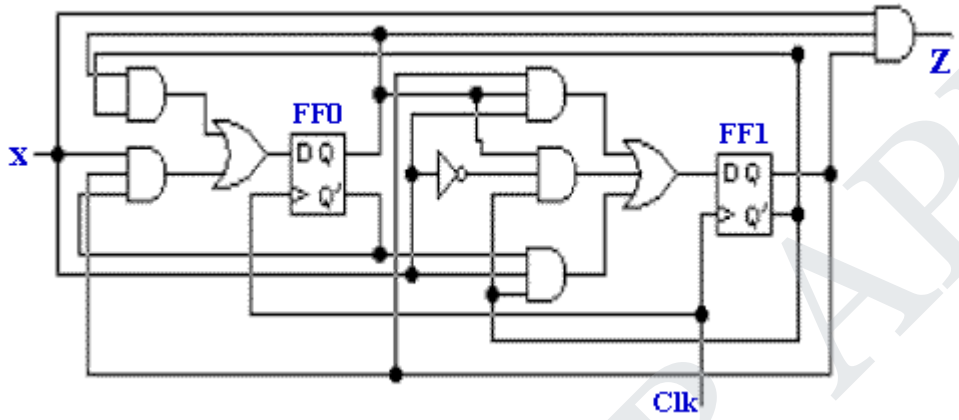
DOWNLOADED FROM STUCOR APP

**D0 = Q0*Q1' + Q0'*Q1*x**

**D1 = Q0'*Q1'*x + Q0*Q1*x + Q0*Q1'*x'**

**Z = Q0*Q1*x**

**Finally, draw the logic diagram.**




**4. Give a brief note on Synchronous and Asynchronous circuits**

**Sequential circuits are divided into two main types: synchronous and asynchronous. Their classification depends on the timing of their signals.**

***Synchronous* sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a free-running clock signal. The clock signal is generally some form of square wave as shown in Figure below.**

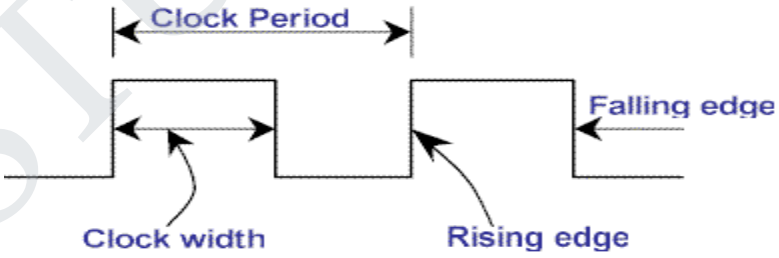



**Figure. Clock Signal**

**From the diagram you can see that the clock period is the time between successive transitions in the same direction, that is, between two rising or two falling edges. State transitions in synchronous sequential circuits are made to take place at times when the clock is making a transition from 0 to 1 (rising edge) or from 1 to 0 (falling edge). Between successive clock pulses there is no change in the information stored in memory.**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**The reciprocal of the clock period is referred to as the clock frequency. The clock width is defined as the time during which the value of the clock signal is equal to 1. The ratio of the clock width and clock period is referred to as the duty cycle. A clock signal is said to be** active high **if the state changes occur at the clock's rising edge or during the clock width. Otherwise, the clock is said to be active low. Synchronous sequential circuits are also known as clocked sequential circuits**.

**The memory elements used in synchronous sequential circuits are usually flip-flops. These circuits are binary cells capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the bit stored in it. Binary information can enter a flip-flop in a variety of ways, a fact which give rise to the different types of flip-flops. For information on the different types of basic flip-flop circuits and their logical properties, see the previous tutorial on flip-flops.**

**In *asynchronous* sequential circuits, the transition from one state to another is initiated by the change in the primary inputs; there is no external synchronisation. The memory commonly used in asynchronous sequential circuits are time-delayed devices, usually implemented by feedback among logic gates. Thus, asynchronous sequential circuits may be regarded as combinational circuits with feedback. Because of the feedback among logic gates, asynchronous sequential circuits may, at times, become unstable due to transient conditions. The instability problem imposes many difficulties on the designer. Hence, they are not as commonly used as synchronous systems.**

**5. Write a brief note on Hazards**

**In digital logic, a hazard in a system is an undesirable effect caused by either a deficiency in the system or external influences. Logic hazards are manifestations of a problem in which changes in the input variables do not change the output correctly due to some form of delay caused by logic elements (NOT, AND, OR gates, etc.) This results in the logic not performing its function properly. The three different most common kinds of hazards are usually referred to as static, dynamic and function hazards.**

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



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**Static Hazards**

**A static hazard is the situation where, when one input variable changes, the output changes momentarily when it shouldn't. This particular type of hazard is usually due to a NOT gate within the logic. The hazard can be dealt with in two ways:**

- **Insert another (additional) delay to the circuit. This then eliminates the static hazard.**
- **Eliminate the hazard by inserting more logic to counteract the effects (Note this makes assumptions that the logic will fail)**

**The first case is the most used of the two options. This is because it does not make assumptions about the logic, instead the method adds redundancy to overcome the hazard.**

**To solve the hazard we shall use our previous example and apply a theory that David A. Huffman discovered. The insertion of a redundant loop can eliminate a static hazard.**

**In the next example, it will also be evident that there will not be a situation where a static '0' occurs. A static '0' hazard is one which briefly goes to '1' when it should remain at '0'. A static '1' hazard is the reverse of this situation, i.e. the output should remain at '1' yet under some condition it briefly changes state to '0' (something we shall see in the following example)..**

**Example of a static hazard**

**Let us consider an imperfect circuit that suffers from a delay in the physical logic elements i.e. AND gates etc. The simple circuit performs the function:**

**$f = X_1 * X_2 + X_1' * X_3$**

**and the logic diagram can be shown as follows:**

**If we first look at the starting diagram, it is clear that if no delays were to occur, then the circuit would function normally. However since this isn't a perfect circuit, and an error occurs when the input changes from 111 to 011. i.e. when X1 changes state. Now we know**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**roughly how the hazard is occurring, for a clearer picture and the solution on how to solve this problem, we look to the Karnaugh Map.**

**This Karnaugh Map shows the circuit. The two gates are shown by solid rings, and the hazard can be seen under the dashed ring. The theory proved by Huffman tells us that by adding a redundant loop 'X2X3' this will eliminate the hazard. So the resulting logic is of the form shown in the next figure.**

**So our original function is now: $f = X_1 * X_2 + X_1' * X_3 + X_2 * X_3$**

**Now we can see that even with imperfect logic elements, our example will not show signs of hazards when X1 changes state. This theory can be applied to any logic system. Computer programs deal with most of this work now, but for simple examples it is quicker to do the debugging by hand. When there are many input variables (say 6 or more) it will become quite difficult to 'see' the errors on a Karnaugh map.**

**6. Explain in detail about hazards**

When designing asynchronous sequential circuits, care must be taken to conform with certain restrictions and precautions to ensure proper operation. The circuit must be operated in fundamental mode with only one input changing at any time and must be free of critical races. In addition, there is one more phenomenon, called *hazard,* that may cause the circuit to malfunction. 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. It is therefore necessary to check for possible hazards and determine whether they cause improper operations. Steps must then be taken to eliminate their effect.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

A hazard is a condition where a single variable change produces a momentary output change when no ouput change should occur. The circuit of Fig. 9-33(a) demonstrates the occurrence of a hazard. Assume that all three inputs are initially equal to 1. This causes the output of gate 1 to be 1, that of gate 2 to be 0, and the output of the circuit to be equal to 1. Now consider a change of $x_2$ from 1 to 0. The output of gate 1 changes to 0 and that of gate 2 changes to 1, leaving the output at 1. However, the output may momentarily go to 0 if the propagation delay through the inverter is taken into consideration. The delay in the inverter may cause the output of gate 1 to change to 0 before the output of gate 2 changes to 1. In that case, both inputs of gate 3 are momentarily equal to 0, causing the output to go to 0 for the short interval of time that the input signal from $x_2$ is delayed while it is propagating through the inverter circuit.

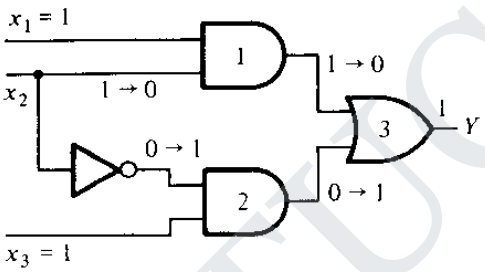
The circuit of Fig. 9-33(b) is a NAND implementation of the same Boolean function. It has a hazard for the same reason. Because gates 1 and 2 are NAND gates, their outputs are the complement of the outputs of the corresponding AND gates. When $x_2$ changes from 1 to 0, both inputs of gate 3 may be equal to 1, causing the output to produce a momentary change to 0 when it should have stayed at 1.

The two circuits shown in Fig. 9-33 implement the Boolean function in sum of products.

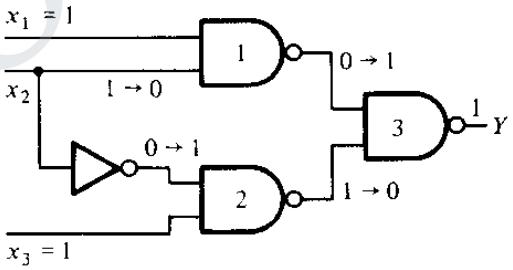
$$Y = x_1x_2 + x_2'x_3$$

This type of implementation may cause the output to go to 0 when it should remain a 1. If the circuit is implemented in product of sums (see Section 3-5),

$$Y = (x_1 + x_2')(x_2 + x_3)$$




(a) AND-OR circuit




(b) NAND circuit



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

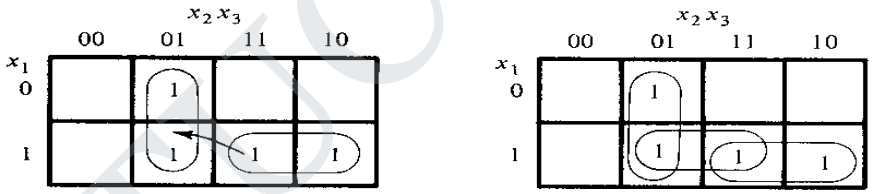
(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazard

**FIGURE 9-34**
Types of hazards

then the output may momentarily go to 1 when it should remain 0. The first case is referred to as *static 1-hazard* and the second case as *static 0-hazard*. A third type of hazard, known as *dynamic hazard*, causes the output to change three or more times when it should change from 1 to 0 or from 0 to 1. Figure 9-34 demonstrates the three types of hazards. When a circuit is implemented in sum of products with AND–OR gates or with NAND gates, the removal of static 1-hazard guarantees that no static 0-hazards or dynamic hazards will occur.

The occurrence of the hazard can be detected by inspecting the map of the particular circuit. To illustrate, consider the map in Fig. 9-35(a), which is a plot of the function implemented in Fig. 9-33. The change in $x_2$ from 1 to 0 moves the circuit from minterm 111 to minterm 101. The hazard exists because the change of input results in a different product term covering the two minterms. 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 of Fig. 9-33. 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. This is shown in the map of Fig. 9-35(b), where the two minterms that cause the hazard are combined into one product term. The hazard-free circuit obtained by this configuration is shown in Fig. 9-36. The extra gate in the circuit generates the product term $x_1x_3$. In general, hazards in combinational circuits can be removed by covering any two minterms that may produce a hazard with a product term common to both. The removal of hazards requires the addition of redundant gates to the circuit.




(a) $Y = x_1x_2 + x_2'x_3$ (b) $Y = x_1x_2 + x_2'x_3 + x_1x_3$

Thus far we have considered what are known as static and dynamic hazards. There is another type of hazard that may occur in asynchronous sequential circuits, called *essential hazard*. An essential hazard is caused by unequal delays along two or more paths that originate from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause such a hazard. Essential hazards cannot be corrected by adding redundant gates as in static hazards. The problem that they impose can be corrected by adjusting the amount of delay in the affected path. To avoid essential hazards, each feedback loop must be handled with individual care to ensure that the delay in the feedback path is long enough compared to delays of other signals that originate from the input terminals. This problem tends to be specialized, as it depends on the particular circuit used and the amount of delays that are encountered in its various paths.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**7. Design a 1-bit adder with carry using VERILOG**

```
module addbit (
a       , // first input
b       , // Second input
ci      , // Carry input
sum     , // sum output
co        // carry output
);
//Input declaration
input a;
input b;
input ci;
//Ouput declaration
output sum;
output co;
//Port Data types
wire  a;
wire  b;
wire  ci;
wire  sum;
wire  co;
//Code starts here
assign {co,sum} = a + b + ci;
endmodule // End of Module addbit
```

**8. Write a brief note on VERILOG coding style**

- **Use meaningful names for signals and variables**
- **Don't mix level and edge sensitive elements in the same always block**
- **Avoid mixing positive and negative edge-triggered flip-flops**
- **Use parentheses to optimize logic structure**
- **Use continuous assign statements for simple combo logic**
- **Use nonblocking for sequential and blocking for combo logic**
- **Don't mix blocking and nonblocking assignments in the same always block (even if Design compiler supports them!!).**
- **Be careful with multiple assignments to the same variable**

  **Define if-else or case statements explicitly**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**9. Design a 3:8 decoder using VERILOG**

```
module decoder_always (in,out);
input [2:0] in;
output [7:0] out;
reg [7:0] out;

always @ (in)
begin
out = 0;
case (in)
3'b001 : out = 8'b0000_0001;
3'b010 : out = 8'b0000_0010;
3'b011 : out = 8'b0000_0100;
3'b100 : out = 8'b0000_1000;
3'b101 : out = 8'b0001_0000;
3'b110 : out = 8'b0100_0000;
3'b111 : out = 8'b1000_0000;
endcase
end

endmodule
```

**10. Explain in detail about ASM**

A flow chart is a convenient way to specify the sequence of procedural steps and decision paths for an algorithm. A flow chart for a hardware algorithm translates the word statement to an information diagram that enumerates the sequence of operations together with the conditions necessary for their execution. A special flow chart that has been developed specifically to define digital hardware algorithms is called an *algorithmic state machine* (ASM) chart. A *state machine* is another term for a sequential circuit, which is the basic structure of a digital system.

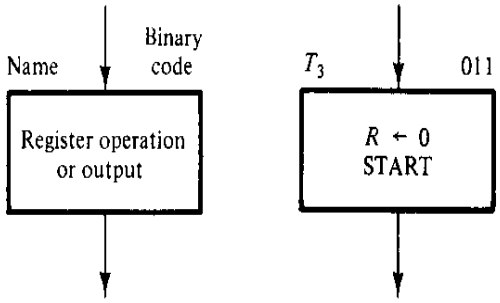
The ASM chart resembles a conventional flow chart, but is interpreted somewhat differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. The ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next. It is specifically adapted to specify accurately the control sequence and data-processing operations in a digital system, taking into consideration the constraints of digital hardware.

This chapter presents a method of digital logic design using the ASM chart. The various blocks that make up the chart are first defined. The timing relationship between the blocks is then explained by example. Various ways of implementing the control logic are discussed together with examples of ASM charts and the corresponding digital systems that they represent.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




(a) General description (b) Specific example

**FIGURE 8-2**
State box

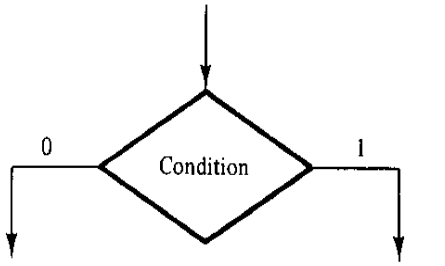



**FIGURE 8-3**
Decision box

The ASM chart is a special type of flow chart suitable for describing the sequential operations in a digital system. The chart is composed of three basic elements: the state box, the decision box, and the conditional box. A state in the control sequence is indicated by a state box, as shown in Fig. 8-2. The shape of the state box is a rectangle within which are written register operations or output signal names that the control generates while being in this state. The state is given a symbolic name, which is placed at the upper left corner of the box. The binary code assigned to the state is placed at the upper right corner. Figure 8-2(b) shows a specific example of a state box. The state has the symbolic name $T_3$, and the binary code assigned to it is 011. Inside the box is written the register operation $R \leftarrow 0$, which indicates that register $R$ is to be cleared to 0 when the system is in state $T_3$. The START name inside the box may indicate, for example, an output signal that starts a certain operation.

The decision box describes the effect of an input on the control subsystem. It has a diamond-shaped box with two or more exit paths, as shown in Fig. 8-3. The input condition to be tested is written inside the box. One exit path is taken if the condition is true and another when the condition is false. When an input condition is assigned a binary value, the two paths are indicated by 1 and 0.

The state and decision boxes are familiar from use in conventional flow charts. The third element, the conditional box, is unique to the ASM chart. The oval shape of the conditional box is shown in Fig. 8-4. The rounded corners differentiate it from the state box. The input path to the conditional box must come from one of the exit paths of a decision box. The register operations or outputs listed inside the conditional box are generated during a given state provided that the input condition is satisfied. Figure 8-5 shows an example with a conditional box. The control generates a START output signal when in state $T_1$. While in state $T_1$, the control checks the status of input $E$. If $E = 1$, then $R$ is cleared to 0; otherwise, $R$ remains unchanged. In either case, the next state is $T_2$.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

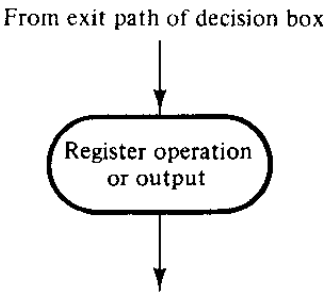



**FIGURE 8-4**
Conditional box

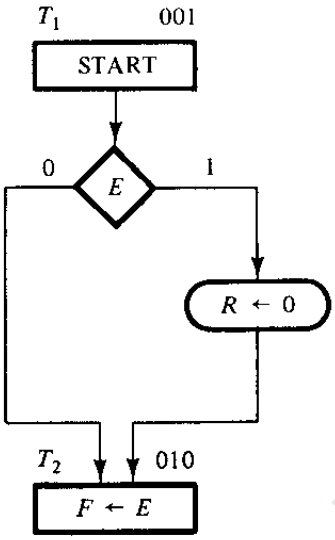



**FIGURE 8-5**
Example with conditional box

An ASM block is a structure consisting of one state box and all the decision and conditional boxes connected to its exit path. An ASM block has one entrance and any number of exit paths represented by the structure of the decision boxes. An ASM chart consists of one or more interconnected blocks. An example of an ASM block is shown in Fig. 8-6. Associated with state $T_1$ are two decision boxes and one conditional box. The diagram distinguishes the block with dashed lines around the entire structure, but

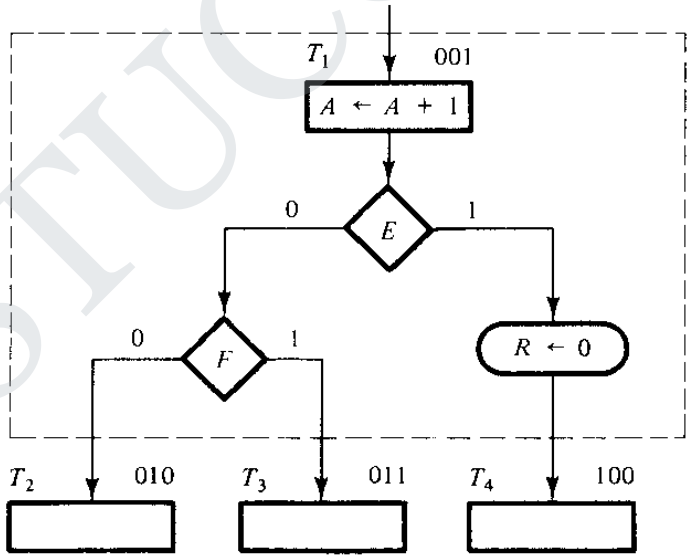



**FIGURE 8-6**
ASM block



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

this is not usually done, since the ASM chart uniquely defines each block from its structure. A state box without any decision or conditional boxes constitutes a simple block.

Each block in the ASM chart describes the state of the system during one clock-pulse interval. The operations within the state and conditional boxes in Fig. 8-6 are executed with a common clock pulse while the system is in state $T_1$. The same clock pulse also transfers the system controller to one of the next states, $T_2$, $T_3$, or $T_4$, as dictated by the binary values of $E$ and $F$.

The ASM chart is very similar to a state diagram. Each state block is equivalent to a state in a sequential circuit. The decision box is equivalent to the binary information written along the directed lines that connect two states in a state diagram. As a consequence, it is sometimes convenient to convert the chart into a state diagram and then use sequential-circuit procedures to design the control logic. As an illustration, the ASM chart of Fig. 8-6 is drawn as a state diagram in Fig. 8-7. The three states are symbolized by circles, with their binary values written inside the circles. The directed lines indicate the conditions that determine the next state. The unconditional and conditional operations that must be performed are not indicated in the state diagram.

A digital system is quite often defined by the registers it contains and the operations that are performed on the data stored in them. A *register* in its broader sense includes storage registers, shift registers, counters, and single flip-flops. Examples of register operations are shift, increment, add, clear, and data transfer. It is sometimes convenient to adopt a suitable notation to describe the operations performed among the registers.

Table 8-1 gives examples of symbolic notation for some register operations. A register is designated by one or more capital letters such as $A$, $B$, or $RA$. The individual cells or flip-flops within an $n$-bit register are numbered in sequence from 1 to $n$ or from 0 to $n - 1$. A single flip-flop is considered a 1-bit register. The transfer of data from one register to another is symbolized by a directed arrow that denotes a transfer of contents

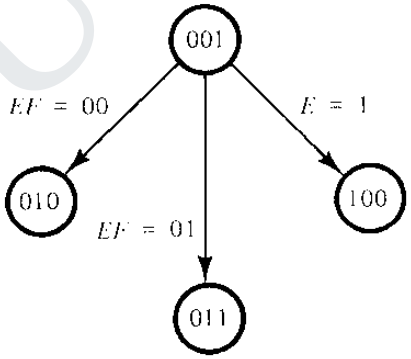



**FIGURE 8-7**
State-diagram equivalent to the ASM chart of Fig. 8-6



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**TABLE 8-1**
**Symbolic Notation for Register Operations**

| Symbolic Notation | Description |
|---|---|
| $A \leftarrow B$ | Transfer contents of register $B$ into register $A$ |
| $R \leftarrow 0$ | Clear register $R$ |
| $F \leftarrow 1$ | Set flip-flop $F$ to 1 |
| $A \leftarrow A + 1$ | Increment register $A$ by 1 (count-up) |
| $A \leftarrow A - 1$ | Decrement register $A$ by 1 (count-down) |
| $A \leftarrow A + B$ | Add contents of register $B$ to register $A$ |

from the source register to the destination register. The register-clear operation is symbolized by a transfer of 0 into the register. A single flip-flop can be set to 1 or cleared to 0. To increment a register by 1, it is necessary that the register be able to count up as in a binary counter. The decrement operation requires a count-down counter. The contents of two registers can be added by means of an adder circuit. Some operations, such as the shift operation, do not have a known symbol. In such a case, we will use the words "shift right $R$" to denote a shift right of register $R$.

**11 Give the hazard free realization for the following functions.**

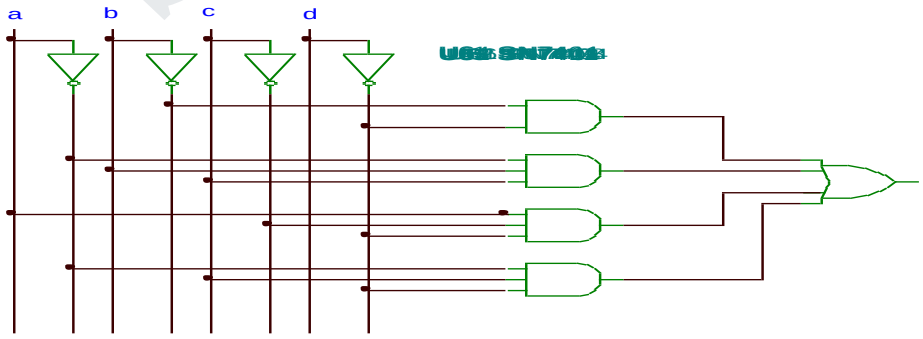
**f(a, b, c, d) = Σ m (0, 2, 6, 7, 8, 10, 12)** **Nov 2005**

K-MAP

| | | | |
|---|---|---|---|
| 1 | | | 1 |
| | | 1 | 1 |
| 1 | | | |
| 1 | | | 1 |

= b'd'+ a'bc + ac'd'+ a'cd'

**Circuit Diagram:**






STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**12. Design an asynchronous sequential circuit that has two inputs $X_2$ and $X_1$ and one output z. When $x_1 = 0$, the output Z is 0. The first change in $X_2$ that occurs while $X_1$ is 1 will cause output Z to be 1. The output Z will remain 1 until $X_1$ returns to zero**

Solution: The state diagram for above problem statement can be given as

A primitive flow table is constructed from the state diagram

| Present state | $X_2X_1$ inputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Next state | | | | Z output | | | |
| | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| A | A | B | - | C | 0 | - | - | - |
| B | A | B | D | - | - | 0 | - | - |
| C | A | - | E | C | - | - | - | 0 |
| D | - | F | D | C | - | - | 1 | - |
| E | - | F | E | C | - | - | 0 | - |
| F | A | F | D | - | - | 1 | - | - |

The merger graph for above primitive flow table can be given as



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

The merger graph gives the two compatible pairs as a set of maximal compatibles.

(A, B) – $S_0$

(C, E) – $S_1$

(D, F) – $S_2$

This set of maximal compatible covers all of the original states resulting in the reduced flow table as

| Present state | $X_2X_1$ inputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Next state | | | | Z output | | | |
| | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| $S_0$ | $S_0$ | $S_0$ | $S_2$ | $S_1$ | 0 | 0 | - | - |
| $S_1$ | $S_0$ | $S_2$ | $S_1$ | $S_1$ | - | - | 0 | 0 |
| $S_2$ | $S_0$ | $S_2$ | $S_2$ | $S_1$ | - | 1 | 1 | - |

Now if we assign $S_0$ = 00, $S_1$ = 01 and $S_2$ = 10 then we need one more state $S_3$ = 11 to prevent critical race during transition of $S_1$ to $S_2$ or $S_2$ to $S_1$.By introducing $S_3$ the transitions $S_1 – S_2$ and $S_2 – S_1$ are routed through $S_3$. Thus, after state assignment the flow table can be given as shown

| Present state | $X_2X_1$ inputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Next state | | | | Z output | | | |
| | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| $S_0$ 0 0 | $S_0$ | $S_0$ | $S_2$ | $S_1$ | 0 | 0 | - | - |
| $S_1$ 0 1 | $S_0$ | $S_3$ | $S_1$ | $S_1$ | - | - | 0 | 0 |
| $S_2$ 1 0 | $S_0$ | $S_2$ | $S_2$ | $S_3$ | - | 1 | 1 | - |
| $S_3$ 1 1 | - | $S_2$ | - | $S_1$ | - | - | - | - |

The flow table can be converted to a transition table

| Present state | | $X_2X_1$ inputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Next state | | | | Z output | | | |
| $F_2$ | $F_1$ | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 0 | 0 | 00 | 00 | 10 | 01 | 0 | 0 | - | - |
| 0 | 1 | 00 | 11 | 01 | 01 | - | - | 0 | 0 |
| 1 | 0 | 00 | 10 | 10 | 11 | - | 1 | 1 | - |
| 1 | 1 | - | 10 | - | 01 | - | - | - | - |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

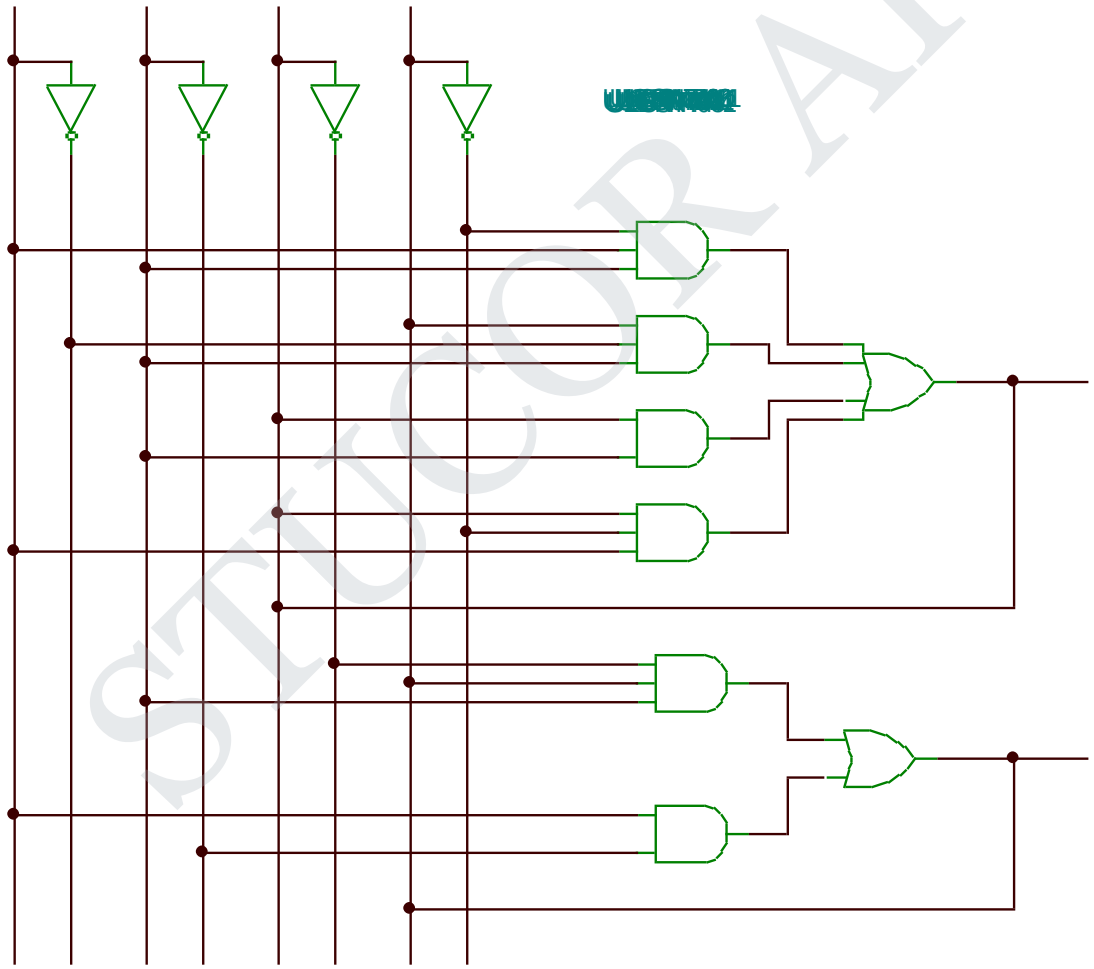
**K-map simplification**

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| X | 1 | X | 0 |
| 0 | 1 | 1 | 1 |

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| X | 0 | X | 1 |
| 0 | 0 | 0 | 1 |

| | | | |
|---|---|---|---|
| 0 | 0 | X | X |
| X | X | 0 | 0 |
| X | X | X | X |
| X | 1 | 1 | X |

**Logic diagram**





STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**13. Summaries the analyzing procedure for asynchronous sequential circuit.**

## Analysis of Fundamental Mode Sequential Circuit

Consider the fundamental mode sequential circuit shown in the Fig. 9.2. The circuit has two input variables I1 and I0 and one output variable Z. The circuit has two feedback paths which provide inputs to the gates, creating latching operation necessary to produce a sequential circuit. The feedback path also generates the state variables X1 and X0 the next state for the circuit is determined by both, the state of input variables and the state variables. From the given sequential circuit we can have next-secondary state and output equations as follows

$$X_1^+ = X_0 \bar{I}_1 + X_0 X_1 I_0$$

$$X_0^+ = X_0 I_0 I_1 + X_1 \bar{I}_0$$

$$Z = X_0 X_1 I_0$$

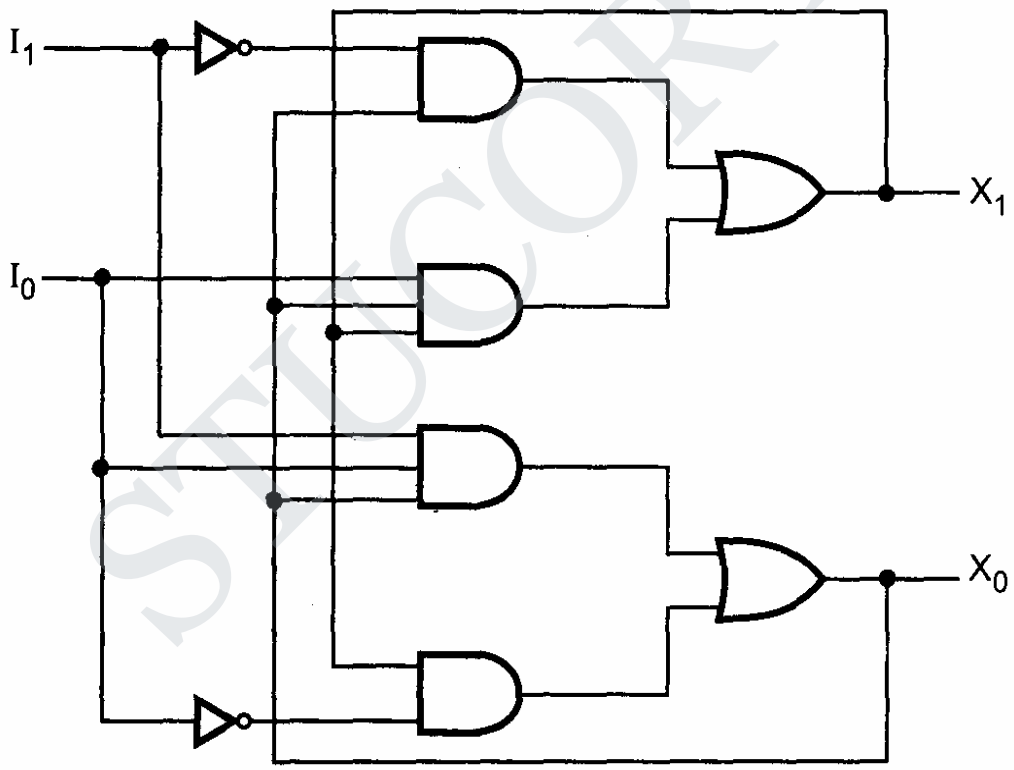



**Fig. 9.2 Fundamental mode asynchronous sequential circuit**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

From these next-secondary state and output equations we can construct the state table indicating present-state, next-secondary state, status of the next-secondary state and the output. The next-secondary state values are found by assigning present state values to the Boolean variables in the next-secondary state equations to determine and X For the given input and secondary state if next-secondary state does then the state is said to be stable not change.

| Present State | | | | Next-Secondary state | | Stable state | Z |
|---|---|---|---|---|---|---|---|
| $X_1$ | $X_0$ | $I_1$ | $I_0$ | $X_1$ | $X_0$ | Yes/No | |
| 0 | 0 | 0 | 0 | 0 | 0 | Yes | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | Yes | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | Yes | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | Yes | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | No | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | No | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | Yes | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | No | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | Yes | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | No | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | Yes | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | No | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | Yes | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | No | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | No | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | No | 0 |

**Table 9.1 State table**

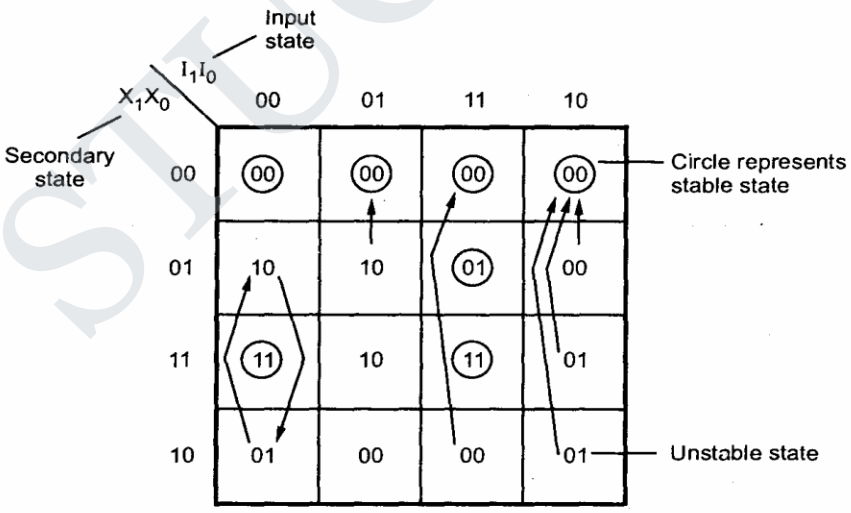



**Fig. 9.3 Transition table**

The Fig. 9.3 shows the transition table. The rows and columns of transition table represent secondary states and input states, respectively. The numbers written in the table represent next-secondary state values. The circle around next-secondary state value indicates that the state is



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

stable. The arrows indicate transitions from unstable states to stable states. For example, if the state (X1, X0, I1, I0) is 1010, the state value is 01 and it is unstable state. The next-secondary stable state will be 0010 as indicated by arrow. There is no stable state for input I1 I0 = 00 with secondary states 10 and 01.Analysis of Pulse Mode Sequential Circuit As mentioned earlier, the pulse mode sequential circuits operate on input pulses rather than levels. Let us consider the asynchronous sequential circuit which is driven by the pulses, as shown in the Fig. 9.4. The two NAND gate latches generate the state variables, A and B. The circuit has four input variables W, X, Y and Z and one output variable C.

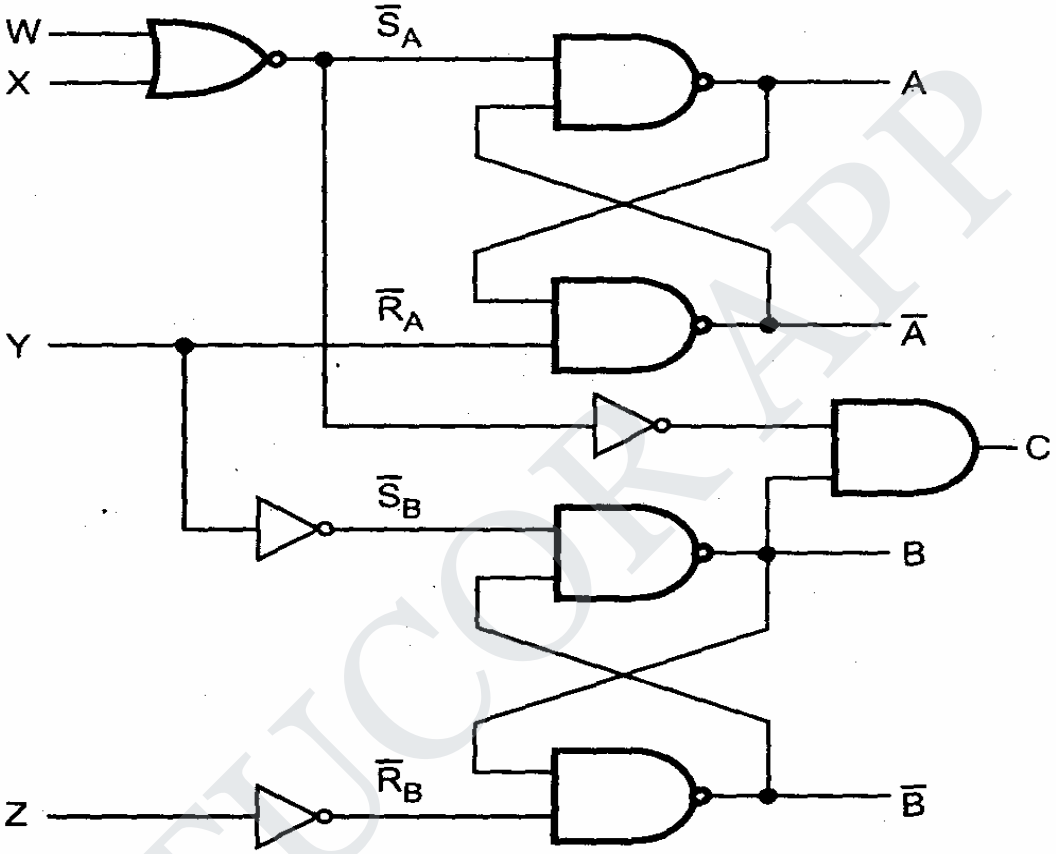



**Fig. 9.4 Pulse mode asynchronous sequential circuit**

From the given sequential circuit we can have excitation and output equations as follows

$$\overline{S}_A = \overline{W + X}\ ,\ S_A = W + X$$
$$\overline{R}_A = Y$$
$$\overline{S}_B = \overline{Y}\ ,\ S_B = Y$$
$$\overline{R}_B = \overline{Z}$$
$$C = (W + X) \cdot B$$



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

The characteristic equation for $\bar{S}\,\bar{R}$ flip-flop is

$$Q_n^+ = S + \bar{R}\,Q_n$$

Using the characteristic equation and excitation equations we have the state variable next-state equations as follows

$$Q_A^+ = W + X + YA$$

$$Q_B^+ = Y + \bar{Z}\,B$$

From these next-state and output equations we can construct the state variable transition table indicating state variables, input variables, next-state values and the output-state.

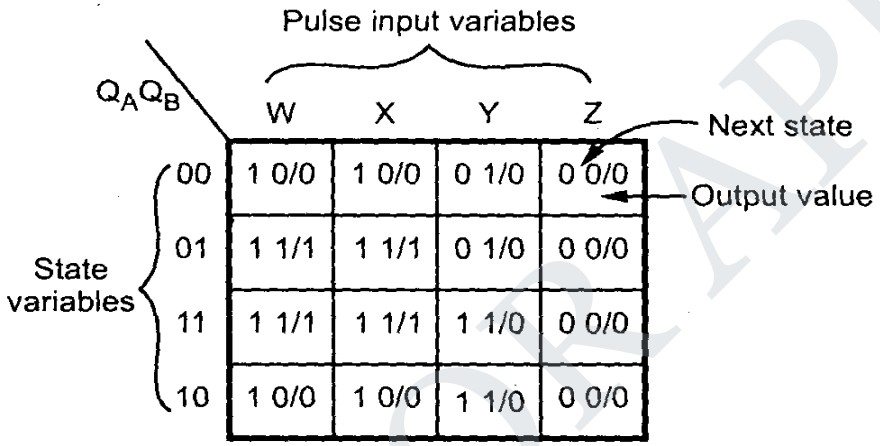



**Fig. 9.5 State variable transition table**

From the state variable transition table we can derive the flow table and from flow table we can derive the state diagram as shown in the Fig. 9.6. The flow diagram can be constructed by replacing next-state and state variable values by actual states.

($S_0$ = 00, $S_1$ = 01, $S_2$ = 11 and $S_3$ = 10).

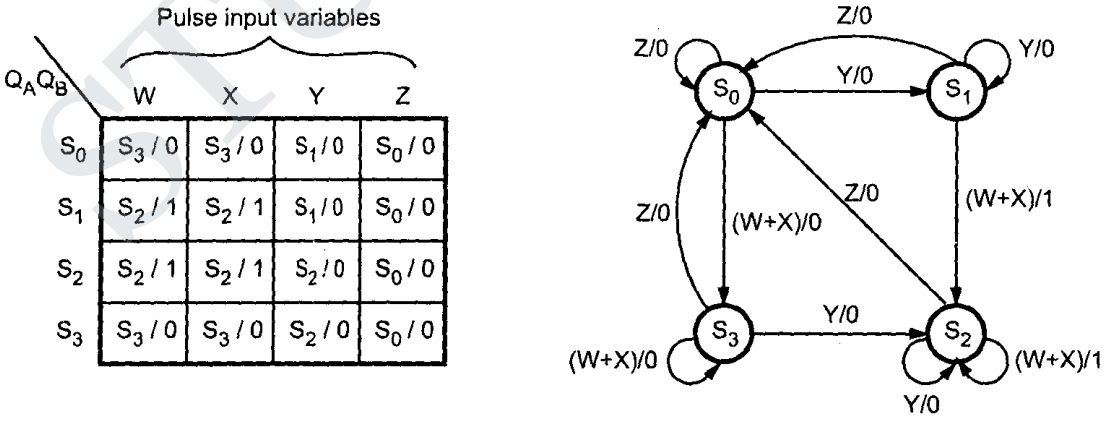



Fig. 9.6 shows the timing diagram for the pulse mode circuit shown in the Fig. 9.6. As shown in the timing diagram, the inputs are the pulses and they occur one at a time.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**14. Summaries the design procedure for asynchronous sequential circuit.**

**Design of Asynchronous Sequential Circuits**

In the previous section we have seen the analysis of asynchronous sequential circuits. It gives us the information of how an existing sequential circuit works. The design process is an exactly reverse process. Here, we know the behavior of the circuit and we have to develop the sequential circuit from scratch. Let us see the steps involved in designing of asynchronous sequential circuit.

1. Construction of a primitive flow table from the problem statement. An intermediate step may include the development of a state diagram.
2. Primitive flow table is reduced by eliminating redundant states by using state reduction techniques.
3. State assignment is made.
4. The primitive flow table is realized using appropriate logic elements.

**Derivation of Primitive Flow Table**

The flow table in the asynchronous sequential circuit is same as that of state table in the synchronous sequential circuit. In asynchronous sequential circuits state table is known as flow table because of the behavior of the asynchronous sequential circuit. The state changes occur independent of a clock, based on the logic propagation delay, and causethe states to "flow" from one to another.

A primitive flow table is a special case of flow table. It is defined as a flow table which has exactly one stable state for each row in the table. The design process begins with the construction of primitive flow table. Let us see the following example to understand the process of construction of the primitive flow table from the problem statement.

❖ Develop the state diagram and primitive flow table for a logic system that has two Inputs X and Y, and a single output Z, which is to behave in the following manner. Initially, both inputs and output are equal to 0. Whenever X = 1 and Y = 0, the Z becomes I and whenever X = 0 and Y = 1, the Z becomes 0. When inputs are zero, i.e. X = Y = 0 or inputs are one, i.e. X = Y = 1, the output Z does not change; it remains in the previous state. The logic system has edge triggered inputs without having a clock. The logic system changes state on the rising edges of the two inputs. Static input values are not to have any effect in changing the Z output.

❖ The construction of a state diagram is accomplished in the same fashion as we discussed in synchronous circuits. The initial A is stable when no input changes have been detected. Now, inputs can change from 00 to 01 or 00 to 10. The input change from 00 to 11 is not allowed according to definition of fundamental mode model (only one input variable change at a time). The input change from 00 to 01 causes a state transition to state B. The state B is stable with a 01 static inputs and Z is a 0. The inputs change from 00 to 10 causes a state transition to state C. The state C is stable with a 10 static inputs and Z is a 1. A state transition B to D occurs when the XY inputs changes from 01 to 11; the Z output remains a 0. Because inputs 11 does not change the state of the output. The state D is stable with a 11 static inputs & Z is a 0. D to C state transition occurs when XY changes from 11 to 10. Once state C is reaches the Z output changes from a 0 to a 1. C to E state transition occurs when XY changes: from 10 to 00; the Z output remains a because inputs 00 does not change the state of the output. The state E is stable with a static 00 inputs and Z is a 1. State C to F transition occurs when inputs change from 10 to 11. The output remains a 1 in state F. The state F is stable with a static 11 inputs and Z is a 1.



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

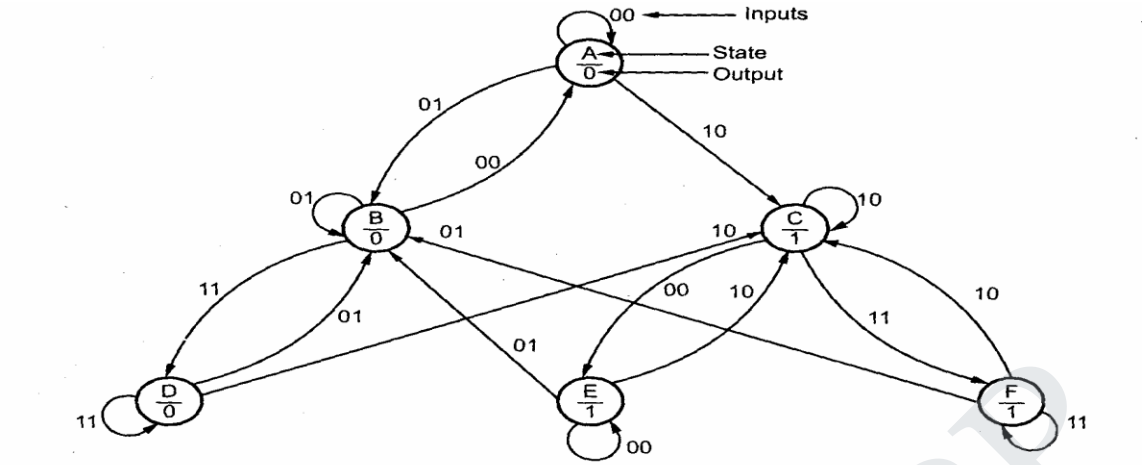



**Fig. 9.7 State diagram for the example problem**

As shown in the Fig. 9.8, the primitive flow table has one row for each state and one column for each input combination. Only one stable state exists for each row in the table. The stable state where no further state transition occur if the inputs are constant. These stable states can be easily identified from the state diagram. For example, state A is stable with output 0 when inputs are 00; state F is stable with output 1 when inputs are 11 and so on. We know that both inputs are not allowed to change inputs are not allowed to change row that differs in two or more simultaneously, so we can enter dash marks in each variables from the input variables associated with the stable state. For example, the first row in the flow table shows a stable state with an input of 00. Since only one input can change at any given time, it can change to 01 or 10, but not to 11 as shown in the state diagram. Therefore, we can enter two dashes in the 11 column of row A. This will eventually result in a don' t care condition for the next state and the output. There maining places in the primitive flow table can be filled by observing state diagram.

For example,
state B is the next state for present state A when input combination is 01 and so on. All outputs associated with unstable states are marked with a dash to indicate don' t-care conditions.

| Present state | Next state, Output Z for X Y Inputs | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| A | (A), 0 | B, – | –, – | C, – |
| B | A, – | (B), 0 | D, – | –, – |
| C | E, – | –, – | F, – | (C), 1 |
| D | –, – | B, – | (D), 0 | C, – |
| E | (E), 1 | B, – | –, – | C, – |
| F | –, – | B, – | (F), 1 | C, – |

**Fig. 9.8 Primitive flow table for example problem**

**Reduction of Primitive Flow Table**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

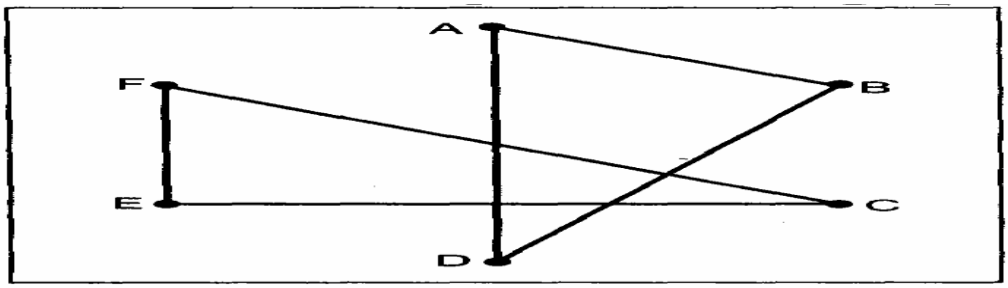



**Fig. 9.9 Merger graph for example problem**

The next step in the design process is to reduce the primitive flow table using state reduction technique. Here, we use merger graph to reduce primitive flow table. According to merger graph we have the same number of vertices as the states in the flow table. The complete line between states vertices is drawn for compatible states. Recall that the states are compatible only when for every input sequence the same output occurs. If two states are incompatible no line is drawn. This is illustrated in Fig. 9.9.In other words, we can say that the two rows of primitive flow table can be merged into a single row if there are no output conflict and no state conflicts in any column. Note that a stable state and the corresponding unstable state are merged to form the corresponding unstable

state are merged to form a stable state. The specified output is filled when specified output is merged with an unspecified output. The merger graph shown in Fig. 9.9 gives the two compatible pairs as a set maximal compatibles.

$$(A, B, D) \rightarrow S_0$$
$$(C, E, F) \rightarrow S_1$$

This set of maximal compatibles covers all of the original states resulting in the reduced flow table as shown in Fig. 9.10.

| Present state | Next state, Output Z for XY inputs | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| $S_0$ | $(S_0)$, 0 | $(S_0)$, 0 | $(S_0)$, 0 | $S_1$, – |
| $S_1$ | $(S_1)$, 1 | $S_0$, – | $(S_1)$, 1 | $(S_1)$, 1 |

**Fig. 9.10 Reduced flow table for example problem**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**15.**

*Design an asynchronous sequential circuit that has two inputs $X_2$ and $X_1$ and one output Z. When $X_1 = 0$, the output Z is 0. The first change in $X_2$ that occurs while $X_1$ is 1 will cause output Z to be 1. The output Z will remain 1 until $X_1$ returns to 0.*

**Sol. :** The state diagram for above problem statement can be given as shown in Fig. 9.26.

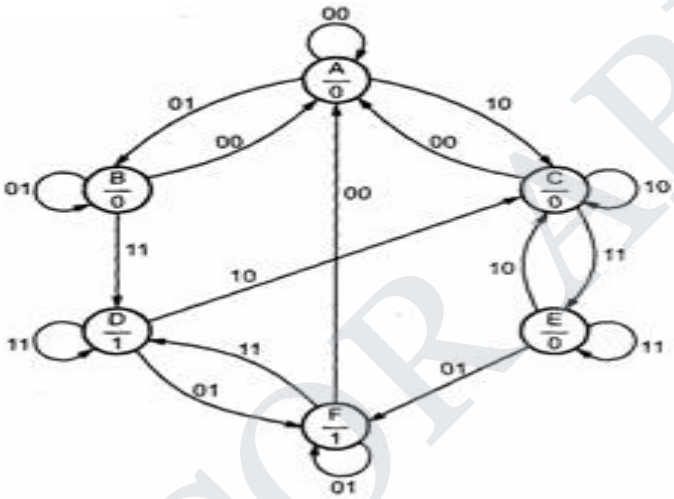



**Fig. 9.26 State diagram for given problem**

A primitive flow table is constructed from the state diagram shown in Fig. 9.27.

| Present state | Next state ,Output Z for $X_2X_1$ Inputs | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| A | (A), 0 | B , – | – , – | C , – |
| B | A , – | (B), 0 | D , – | – , – |
| C | A , – | – , – | E , – | (C), 0 |
| D | – , – | F , – | (D), 1 | C , – |
| E | – , – | F , – | (E), 0 | C , – |
| F | A , – | (F), 1 | D , – | – , – |

**Fig. 9.27 Primitive flow table for given problem**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

The merger graph for above primitive flow table can be given as in Fig. 9.28.

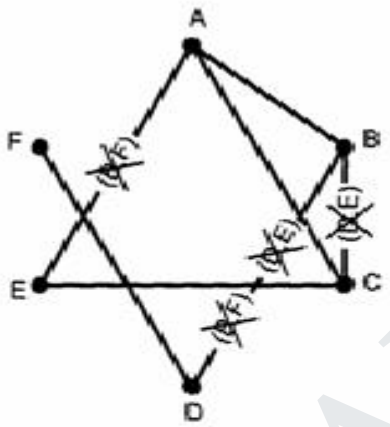



Fig. 9.28 Merger graph

The merger graph gives the two compatible pairs as a set of maximal compatibles.

$(A, B) \rightarrow S_0$

$(C, E) \rightarrow S_1$

$(D, F) \rightarrow S_2$

This set of maximal compatible covers all of the original states resulting in the reduced flow table as shown in Fig. 9.29.

| Present state | Next state, Output Z for $X_2X_1$ Inputs | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| $S_0$ | $(S_0)$, 0 | $(S_0)$, 0 | $S_2$, – | $S_1$, – |
| $S_1$ | $S_0$, – | $S_2$, – | $(S_1)$, 0 | $(S_1)$, 0 |
| $S_2$ | $S_0$, – | $(S_2)$, 1 | $(S_2)$, 1 | $S_1$, – |

Fig. 9.29 Reduced flow table

Now if we assign $S_0 \rightarrow 00$, $S_1 \rightarrow 01$ and $S_2 \rightarrow 10$ then we need one more state $S_3 \rightarrow 11$ to prevent critical race during transition of $S_1 \rightarrow S_2$ or $S_2 \rightarrow S_1$. By introducing $S_3$ the transitions $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$ are routed through $S_3$. Thus, after state assignment the flow table can be given as shown in Fig. 9.30.



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| Present state $F_2$ $F_1$ | Next state, Output Z for $X_2X_1$ Inputs 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| $S_0$ → 0 0 | (S0), 0 | (S0), 0 | S2, – | S1, – |
| $S_1$ → 0 1 | S0, – | S3, – | (S1), 0 | (S1), 0 |
| $S_2$ → 1 0 | S0, – | (S2), 1 | (S2), 1 | S3, – |
| $S_3$ → 1 1 | –, – | S2, – | –, – | S1, – |

**Fig. 9.30 Flow table with state assignment**

The flow table in Fig. 9.31 can be converted to a transition table as shown in Fig. 9.31

| Present state $F_2$ | $F_1$ | Next state, Output Z for $X_2X_1$ Inputs 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, – |

**Fig. 9.31 Transition table**

## K-map simplification

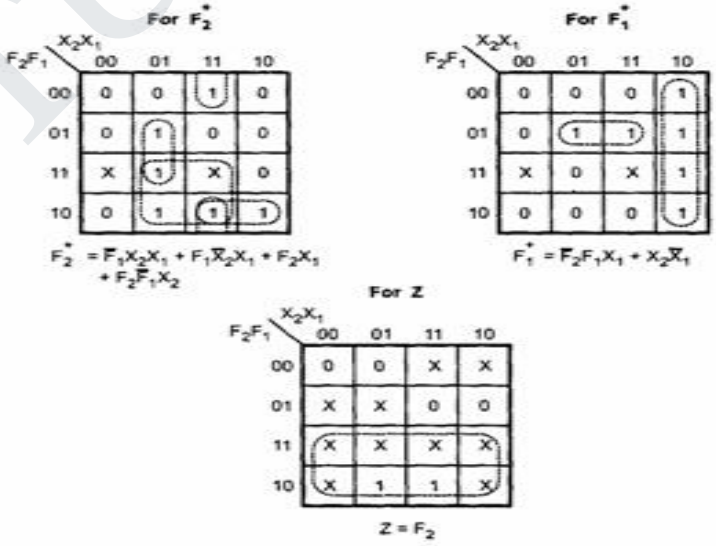


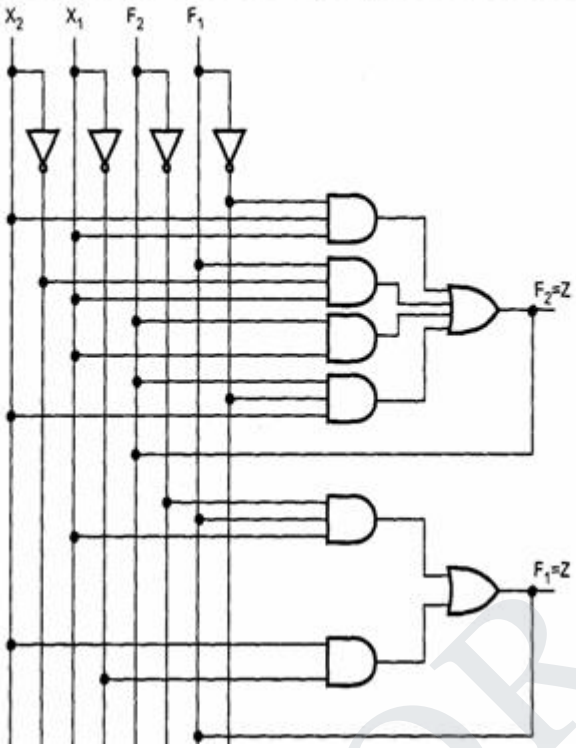

**Fig. 9.32**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

Logic diagram



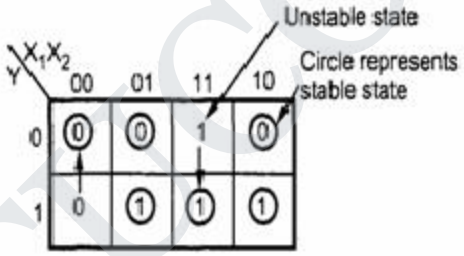




Fig. 9.36 Transition table

| Y \ $X_1X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | – | 0 |
| 1 | – | 1 | 1 | 1 |

Output is mapped for all stable states. For unstable states output is mapped unspecified

Fig. 9.37 Output map

iii) The circuit gives carry output of the full adder circuit.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**16.**

*An asynchronous sequential circuit has two internal states and one output. Th excitation and output function describing the circuit are as follows.*

$Y_1 = x_1x_2 + x_1y_2 + x_2y_1$

$Y_2 = x_2 + x_1y_1y_2 + x_1y_1$

$Z = x_2 + y_1$

**Sol. :** The logic diagram is as shown in the Fig. 9.38.

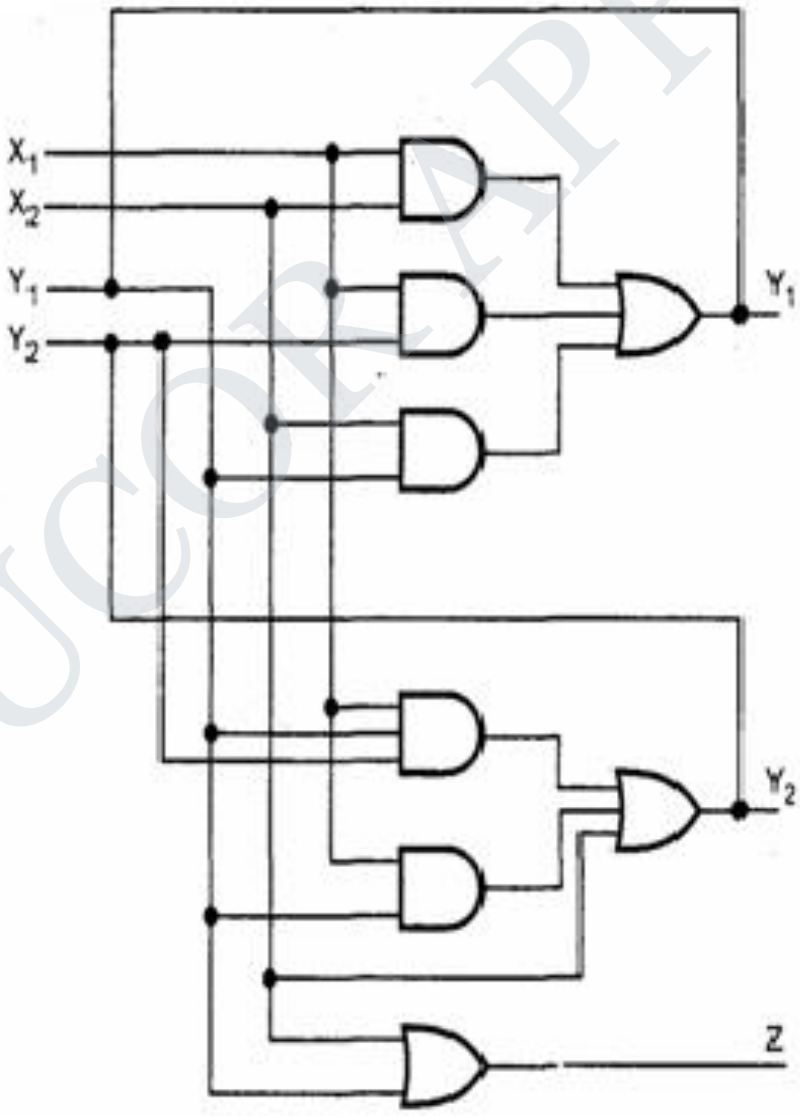



**Fig. 9.38 Logic diagram for given problem**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| Present total state | | | | Next total state | | | | Stable total state | Z |
|---|---|---|---|---|---|---|---|---|---|
| $X_1$ | $X_2$ | $Y_1$ | $Y_2$ | $X_1$ | $X_2$ | $Y_1$ | $Y_2$ | Yes/No | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Yes | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | No | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | No | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | No | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | No | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | Yes | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | No | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | Yes | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | Yes | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | No | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | No | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | Yes | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | No | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | No | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | No | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Yes | 1 |

**Fig. 9.39 Present state, next state table and output**

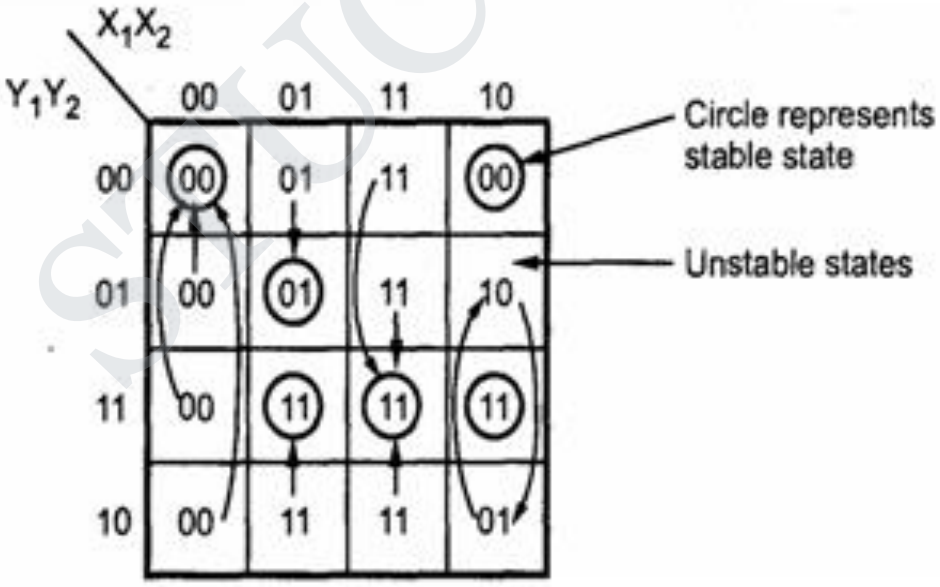



**Fig. 9.40 Transition table**



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**PART C**

## ASSIGNMENT QUESTIONS

**1. Design a asynchronous sequential circuit with 2 inputs T and C. The output attains a value**

**of 1 when T = 1 & c moves from 1 to 0. Otherwise the output is 0.**

**Obtain the state diagram**

**2. State with a neat example the method for the minimization of primitive flow table.**

**3. Explain with neat diagram the different hazards and the way to eliminate them.**

**4. Explain the different methods of state assignment**

**5. Explain in detail about Races.**

STUCOR APP



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

## UNIT –V

## MEMORY DEVICES AND DIGITAL INTEGRATED CIRCUITS

### PART-B

**1. A combinational circuit is defined by the functions $F_1 = \sum m(3,5,7)$ $F_2 = \sum m(4,5,7)$ Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs (Nov 2005).**

| | | 1 | |
|---|---|---|---|
| 1 | 1 | | |

**Solution:**

The

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | |

two functions are simplified in sum of products as,

$F_1 = AC + BC$ $F_2 = AB' + AC$

**PLA programming table:**

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

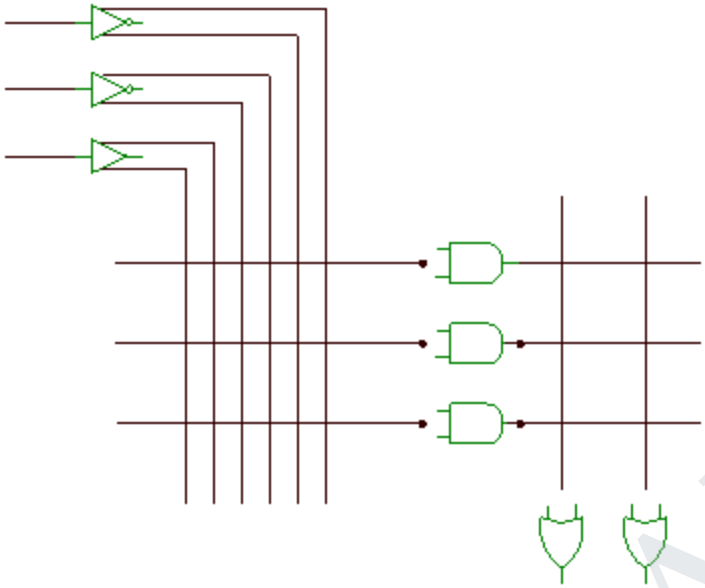
**Logic circuit:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP



**2. A combinational logic is defined by the functions F =∑m (3, 4, 5, 7, 10, 14, 15) and G = ∑m (1, 5, 7, 11, 15) .Implement the circuit using a programmable logic array with 4 inputs, 6 product terms and 2 outputs. (May 2006).**

**Solution**

The two functions are simplified in sum of products by using karnaugh map as,

| | | | |
|---|---|---|---|
| | | 1 | |
| 1 | 1 | 1 | |
| | | 1 | 1 |
| | | | 1 |

| | | | |
|---|---|---|---|
| | 1 | | |
| | 1 | 1 | |
| | | 1 | |
| | | 1 | |

F = $\bar{A}B\bar{C} + \bar{A}CD + BCD + AC\bar{D}$

G = $\bar{A}\bar{C}D + \bar{A}BD + BCD$



STUCOR APP

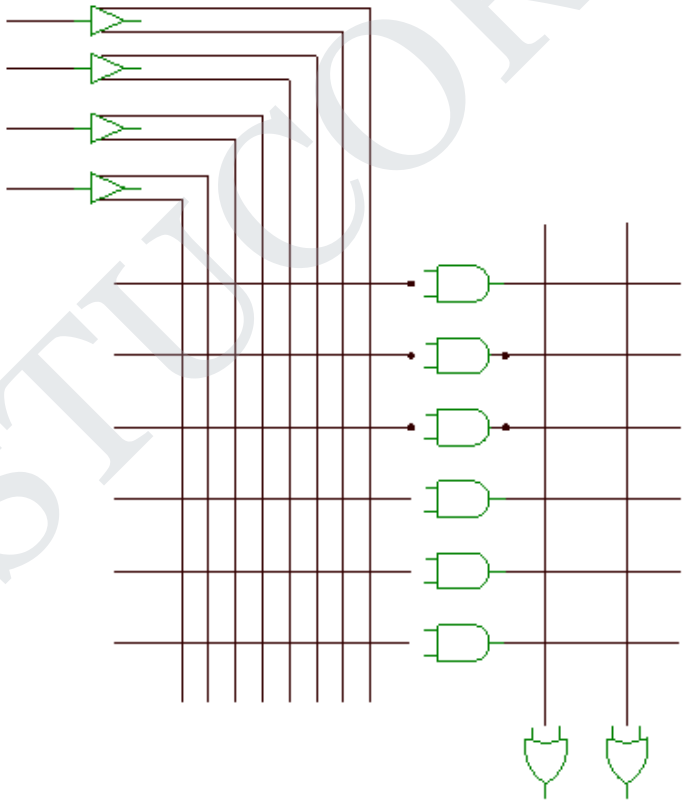
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**PLA Programming Table:**

| | Product Term | Input A | B | C | Output F | G |
|---|---|---|---|---|---|---|
| $\bar{A}B\bar{C}$ | 1 | 0 | 1 | 0 | 1 | - |
| $\bar{A}CD$ | 2 | 0 | 1 | 1 | 1 | - |
| $BCD$ | 3 | 1 | 1 | 1 | 1 | 1 |
| $AC\bar{D}$ | 4 | 1 | 1 | 0 | 1 | - |
| $\bar{A}\bar{C}D$ | 5 | 0 | 0 | 1 | - | 1 |
| $\bar{A}BD$ | 6 | 0 | 1 | 1 | 1 | 1 |

**Logic Diagram:**





STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**3. Design a combinational circuit using a ROM that accepts a 3-bit number and generates an output binary number equal to the square of the given input number. (May 2004)**

- ✓ This generation requires 3 inputs and 6 outputs.
- ✓ The output $B_0$ is always equal to input $A_0$ so it is not generated with ROM.

**(i) Truth Table:**

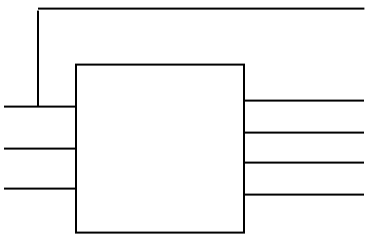
| Inputs | | | Outputs | | | | | | Decimal |
|---|---|---|---|---|---|---|---|---|---|
| $A_3$ | $A_2$ | $A_1$ | $B_5$ | $B_4$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 3 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 4 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 5 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 6 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 7 |

- ✓ O[illegible]
- ✓ Therefore ROM is generated with three inputs and four outputs since the other two
- ✓ outputs are generated without using ROM.

**(ii) ROM Truth Table:**

| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| $A_3$ | $A_2$ | $A_1$ | $B_5$ | $B_4$ | $B_3$ | $B_2$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

**(iii) ROM Implementation:**





STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**4. The following is a truth table of a 3-input, 4-output combinational circuit. Tabulate the PAL programming table for the circuit and mark the fuse map in a PAL diagram.**

| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| x | y | z | A | B | C | D |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | |

From the truth table logical expression can be written with shorthand notations as,

$A(x, y, z) = \sum (1, 2, 4, 6)$ $\quad C(x, y, z) = \sum (1, 2, 5, 6, 7)$

$B(x, y, z) = \sum (0, 1, 3, 6, 7)$ $\quad D(x, y, z) = \sum (1, 2, 3, 5, 7)$

The Boolean expressions are simplified using K-map as

**For A** $\qquad$ **For B**

$A = \bar{x}z + y\bar{z} + \bar{x}\bar{y}z$ $\qquad B = \bar{x}\bar{y} + yz + xy$

**For C** $\qquad$ **For D**

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | |
| | | 1 | 1 |

$C = \bar{x}\bar{y}z + x\bar{z} + y\bar{z} + xy$ $\qquad D = z + \bar{x}y$

Simplifying the four functions to a minimum number of terms as,

$A = x\bar{z} + y\bar{z} + \bar{x}\bar{y}z$ $\qquad C = \bar{x}\bar{y}z + x\bar{z} + y\bar{z} + xy$

$B = \bar{x}\bar{y} + yz + xy$ $\qquad = A + xy$

$D = z + \bar{x}y$



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**PLA Programming Table:**

| Product Term | AND Inputs x | y | z | A | Outputs |
|---|---|---|---|---|---|
| 1. $= x\bar{z} + y\bar{z} + \bar{x}\bar{y}z$ | 1 | - | 0 | - | A |
| 2. | - | 1 | 0 | - | |
| 3. | 0 | 0 | 1 | - | |
| 4. $B = \bar{x}\bar{y} + yz + xy$ | 0 | 0 | - | - | |
| 5. | - | 1 | 1 | - | |
| 6. | 1 | 1 | - | - | |
| 7. $C = A + xy$ | - | - | - | 1 | |

**Fuse Map fo**

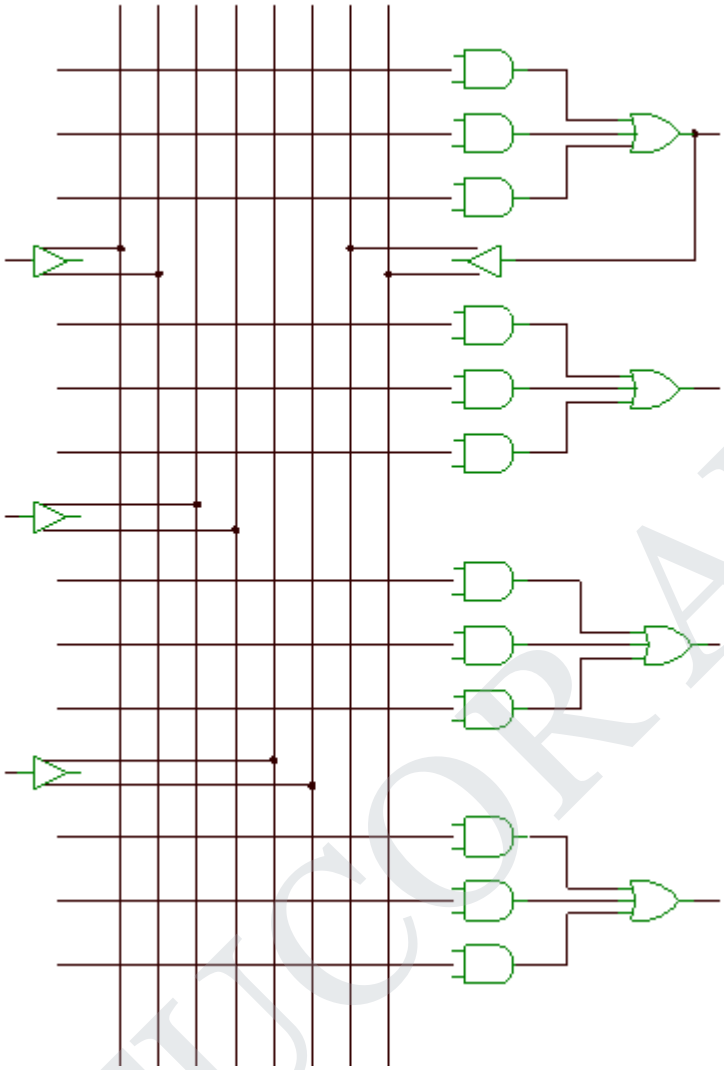
STUCOR APP



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP



**5. Tabulate the PAL programming table for the four Boolean functions listed below**

**A(x, y, z) = ∑ (1, 2, 4, 6)**

**B(x, y, z) = ∑(0, 1, 6, 7)**

**C(x, y, z) = ∑(2, 6)**

**D(x,y,z) = ∑(1, 2, 3, 5, 7)**

**Simplification using K-map:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

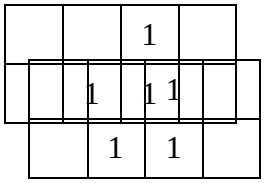
For A

$A = x\bar{z} + \bar{x}yz + y\bar{z}$

For B

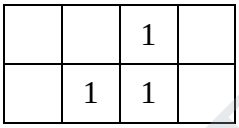
$B = \bar{x}\bar{y} + xy$

For C



$C = y\bar{z}$

For D



$D = z + \bar{x}y$

**PLA Programming Table:**

| Product Term | AND Inputs x | y | z | C | Outputs |
|---|---|---|---|---|---|
| 1. | - | - | 1 | - | $A = C + x\bar{z} + \bar{x}\bar{y}z$ |
| 2. | 1 | - | 0 | - | |
| 3. | 0 | 0 | 1 | - | |
| 4. | 0 | 0 | - | - | $B = \bar{x}\bar{y}xy$ |
| 5. | 1 | 1 | - | - | |
| 6. | - | - | - | - | |
| 7. | - | 1 | 0 | 1 | $C = y\bar{z}$ |
| 8. | - | - | - | - | |
| 9. | - | - | - | - | |
| 10. | - | - | 1 | - | $D = z + \bar{x}y$ |
| 11. | 0 | 1 | - | - | |
| 12. | - | - | - | - | |

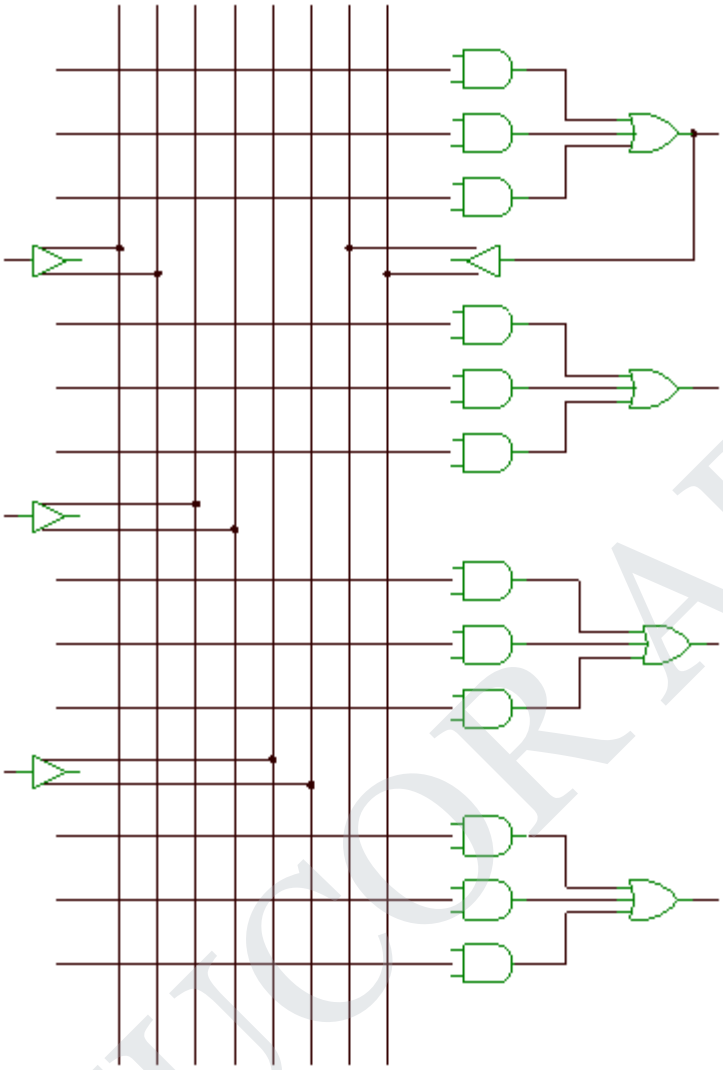
**PAL Fuse Map:**



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP



STUCOR APP



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

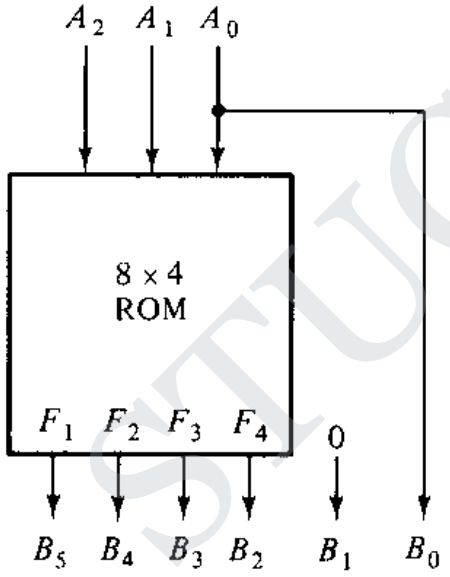
Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

The first step is to derive the truth table for the combinational circuit. In most cases, this is all that is needed. In some cases, we can fit a smaller truth table for the ROM by using certain properties in the truth table of the combinational circuit. Table 5-5 is the

**TABLE 5-5**
**Truth Table for Circuit of Example 5-3**

| Inputs | | | Outputs | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A_2$ | $A_1$ | $A_0$ | $B_5$ | $B_4$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ | Decimal |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 16 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 25 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 36 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 49 |

**6.**




(a) Block diagram

| $A_2$ | $A_1$ | $A_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

(b) ROM truth table



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

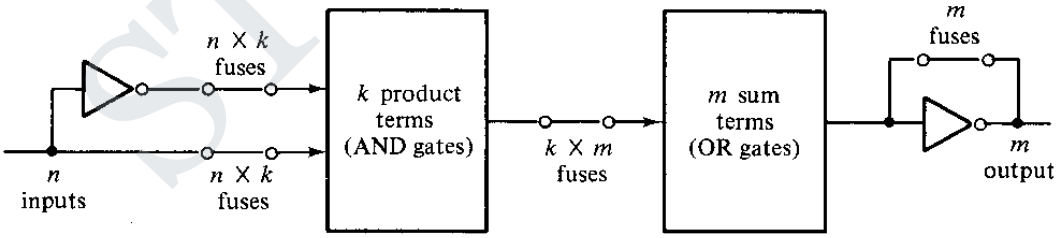
### 7. Write a note on PLA

A combinational circuit may occasionally have don't-care conditions. When implemented with a ROM, a don't-care condition becomes an address input that will never occur. The words at the don't-care addresses need not be programmed and may be left in their original state (all 0's or all 1's). The result is that not all the bit patterns available in the ROM are used, which may be considered a waste of available equipment.

Consider, for example, a combinational circuit that converts a 12-bit card code to a 6-bit internal alphanumeric code (see end of Section 1-7). The input card code consists of 12 lines designated by 0, 1, 2, . . . , 9, 11, 12. The size of the ROM for implementing the code converter must be 4096 × 6, since there are 12 inputs and 6 outputs. There are only 47 valid entries for the card code; all other input combinations are don't-care conditions. Thus, only 47 words of the 4096 available are used. The remaining 4049 words of ROM are not used and are thus wasted.

For cases where the number of don't-care conditions is excessive, it is more economical to use a second type of LSI component called a *programmable logic array*, or PLA. A PLA is similar to a ROM in concept; however, the PLA does not provide full decoding of the variables and does not generate all the minterms as in the ROM. In the PLA, the decoder is replaced by a group of AND gates, each of which can be programmed to generate a product term of the input variables. The AND and OR gates inside the PLA are initially fabricated with fuses among them. The specific Boolean functions are implemented in sum of products form by blowing appropriate fuses and leaving the desired connections.

A block diagram of the PLA is shown in Fig. 5-25. It consists of $n$ inputs, $m$ outputs, $k$ product terms, and $m$ sum terms. The product terms constitute a group of $k$ AND gates and the sum terms constitute a group of $m$ OR gates. Fuses are inserted between all $n$ inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates. Another set of fuses in the output inverters allows the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form. With the inverter fuse in place, the inverter is bypassed, giving an AND-OR implementation. With the fuse blown, the inverter becomes part of the circuit and the function is implemented in the AND-OR-INVERT form.






STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

The size of the PLA is specified by the number of inputs, the number of product terms, and the number of outputs (the number of sum terms is equal to the number of outputs). A typical PLA has 16 inputs, 48 product terms, and 8 outputs. The number of programmed fuses is $2n \times k + k \times m + m$, whereas that of a ROM is $2^n \times m$.

Figure 5-26 shows the internal construction of a specific PLA. It has three inputs, three product terms, and two outputs. Such a PLA is too small to be available commercially; it is presented here merely for demonstration purposes. Each input and its complement are connected through fuses to the inputs of all AND gates. The outputs of the AND gates are connected through fuses to each input of the OR gates. Two more fuses are provided with the output inverters. By blowing selected fuses and leaving others intact, it is possible to implement Boolean functions in their sum of products form.

As with a ROM, the PLA may be mask-programmable or field-programmable. With a mask-programmable PLA, the customer must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a custom-made PLA that has the required internal paths between inputs and outputs. A second type of PLA available is called a *field-programmable logic array*, or FPLA. The FPLA can be programmed by the user by means of certain recommended procedures. Commercial hardware programmer units are available for use in conjunction with certain FPLAs.

**8.**

A combinational circuit is defined by the functions

$$F_1(A, B, C) = \Sigma(3, 5, 6, 7)$$

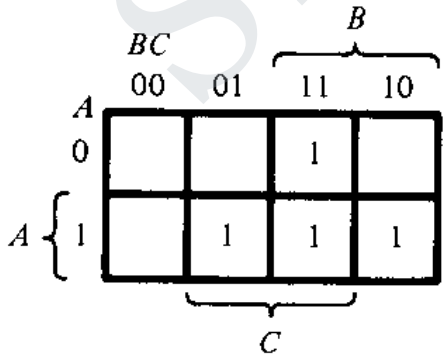
$$F_2(A, B, C) = \Sigma(0, 2, 4, 7)$$

Implement the circuit with a PLA having three inputs, four product terms, and two outputs.

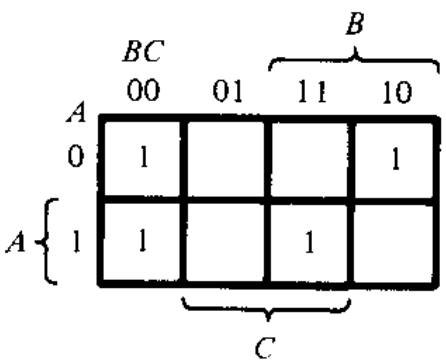
The two functions are simplified in the maps of Fig. 5-28. Both the true values and the complements of the functions are simplified. The combinations that give a minimum number of product terms are

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

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




$F_1 = AC + AB + BC$

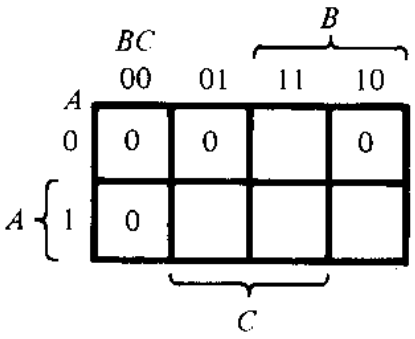



$F_2 = B'C' + A'C' + ABC$

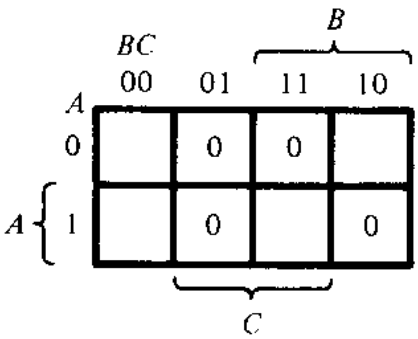


STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP




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




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

PLA program table

| | Product term | Inputs A | Inputs B | Inputs C | Outputs $F_1$ | Outputs $F_2$ |
|---|---|---|---|---|---|---|
| $B'C'$ | 1 | – | 0 | 0 | 1 | 1 |
| $A'C'$ | 2 | 0 | – | 0 | 1 | 1 |
| $A'B'$ | 3 | 0 | 0 | – | 1 | – |
| $ABC$ | 4 | 1 | 1 | 1 | – | 1 |
| | | | | | $C$ | $T$ | $T/C$ |

This gives only four distinct product terms: $B'C'$, $A'C'$, $A'B'$, and $ABC$. The PLA program table for this combination is shown in Fig. 5-28. Note that output $F_1$ is the normal (or true) output even though a $C$ is marked under it. This is because $F_1'$ is generated *prior* to the output inverter. The inverter complements the function to produce $F_1$ in the output. ■

The combinational circuit for this example is too small for practical implementation with a PLA. It was presented here merely for demonstration purposes. A typical commercial PLA would have over 10 inputs and about 50 product terms. The simplification of Boolean functions with so many variables should be carried out by means of a tabulation method or other computer-assisted simplification method. This is where a computer program may aid in the design of complex digital systems. The computer program should simplify each function of the combinational circuit and its complement to a minimum number of terms. The program then selects a minimum number of distinct terms that cover all functions in their true or complement form.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**9. Write a note on PAL**

Programmable logic devices have hundreds of gates interconnected through hundreds of electronic fuses. It is sometimes convenient to draw the internal logic of such devices in a compact form referred to as *array logic*. Figure 5-29 shows the conventional and array logic symbols for a multiple-input AND gate. The conventional symbol is drawn with multiple lines showing the fuses connected to the inputs of the gate. The corresponding array logic symbol uses a single horizontal line connected to the gate input and multiple vertical lines to indicate the individual inputs. Each intersection between a vertical line and the common horizontal line has a fused connection. Thus, in Fig. 5-29(b), the AND gate has four inputs connected through fuses. In a similar fashion, we can draw the array logic for the OR gate or any other type of multiple-input gate.

The programmable array logic (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, but is not as flexible as the PLA. Figure 5-30 shows the array logic configuration of a typical PAL. It has four inputs and four outputs. Each input has a buffer and an inverter gate. Note that the two gates are shown with one composite graphic symbol with normal and complement outputs. There are four sections in the unit, each being composed of a threewide AND-OR array. This is 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 fused programmable inputs. This is 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 the inputs of the AND gates through fuses.

(a) Conventional symbol (b) Array logic symbol

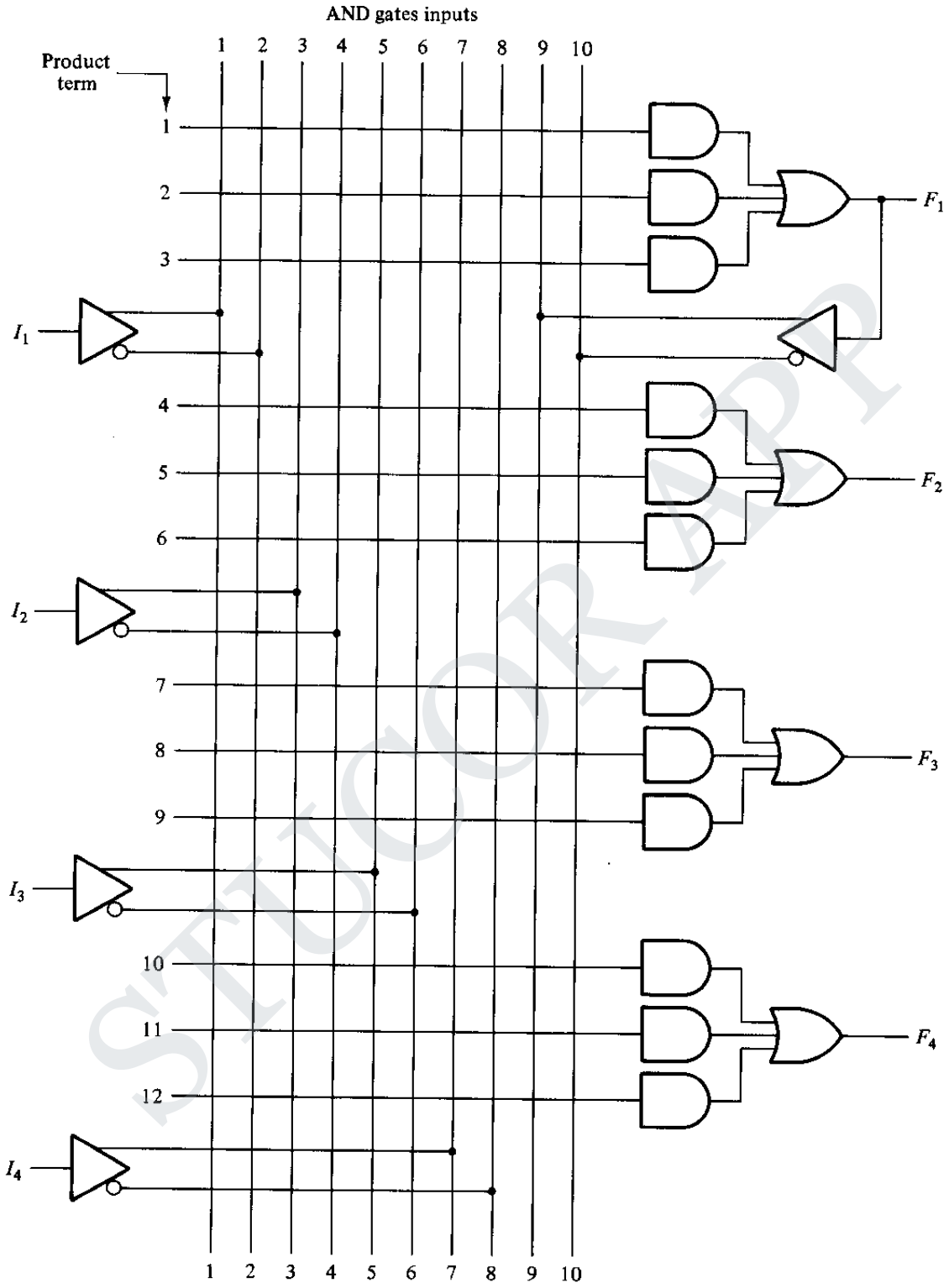
Commercial PAL devices contain more gates than the one shown in Fig. 5-30. A typical PAL integrated circuit may have eight inputs, eight outputs, and eight sections, each consisting of an eightwide AND-OR array. The output terminals are sometimes bidirectional, which means that they can be programmed as inputs instead of outputs if desired.

When designing with a PAL, the Boolean functions must be simplified to fit into each section. Unlike the PLA, a product term cannot be shared among two or more OR gates. Therefore, each function can be simplified by itself without regard to 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.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP






STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**10. Design a combinational circuit using PAL.**

$$w(A, B, C, D) = \Sigma(2, 12, 13)$$

$$x(A, B, C, D) = \Sigma(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$y(A, B, C, D) = \Sigma(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

$$z(A, B, C, D) = \Sigma(1, 2, 8, 12, 13)$$

Simplifying the four functions to a minimum number of terms results in the following Boolean functions:

$$w = ABC' + A'B'CD'$$

$$x = A + BCD$$

$$y = A'B + CD + B'D'$$

$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$

$$= w + AC'D' + A'B'C'D$$

Note that the function for $z$ has four product terms. The logical sum of two of these terms is equal to $w$. By using $w$, it is possible to reduce the number of terms for $z$ from four to three.

The PAL programming table is similar to the one used for the PLA except that only the inputs of the AND gates need to be programmed. Table 5-6 lists the PAL programming table for the four Boolean functions. The table is divided into four sections with three product terms in each to conform with the PAL of Fig. 5-30. The first two sections need only two product terms to implement the Boolean function. The last section for output $z$ needs four product terms. Using the output from $w$, we can reduce the function to three terms.

The fuse map for the PAL as specified in the programming table is shown in Fig. 5-31. For each 1 or 0 in the table, we mark the corresponding intersection in the diagram with the symbol for an intact fuse. For each dash, we mark the diagram with blown fuses in both the true and complement inputs. If the AND gate is not used, we leave all its input fuses intact. Since the corresponding input receives both the true and complement of each input variable, we have $AA' = 0$ and the output of the AND gate is always 0.

As with all PLDs, the design with PALs is facilitated by using computer-aided design techniques. The blowing of internal fuses is a hardware procedure done with the help of special electronic instruments.



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**PAL Programming Table**

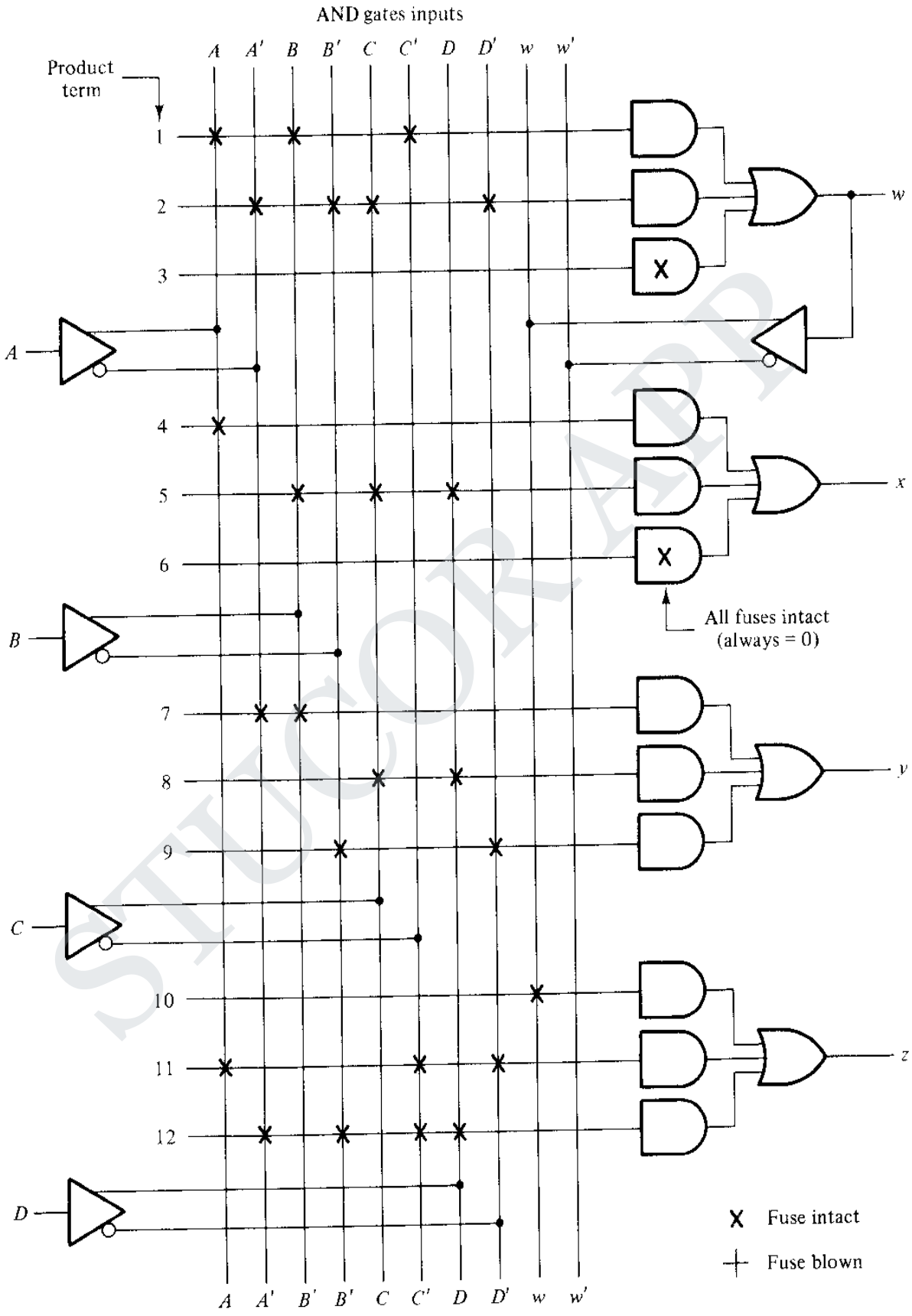
| Product Term | AND Inputs A | B | C | D | W | Outputs |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | – | – | $w = ABC'$ |
| 2 | 0 | 0 | 1 | 0 | – | $+ A'B'CD'$ |
| 3 | – | – | – | – | – | |
| 4 | 1 | – | – | – | – | $x = A$ |
| 5 | – | 1 | 1 | 1 | – | $+ BCD$ |
| 6 | – | – | – | – | – | |
| 7 | 0 | 1 | – | – | – | $y = A'B$ |
| 8 | – | – | 1 | 1 | – | $+ CD$ |
| 9 | – | 0 | – | 0 | – | $+ B'D'$ |
| 10 | – | – | – | – | 1 | $z = w$ |
| 11 | 1 | – | 0 | 0 | – | $+ AC'D'$ |
| 12 | 0 | 0 | 0 | 1 | – | $+ A'B'C'D$ |



STUCOR APP

DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP






STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

**11.A combinational circuit is defined by the functions F0 = _m(0,1,4,6) = A'B'+AC', F1 = _m(2,3,4,6,7) = B+AC', F2 = _m(0,1,2,6) =A'B'+BC', F3 = _m(2,3,5,6,7) =AC+B Implement the circuit with a PLA having 3 inputs , 5 product terms and 4 outputs.**

A programmable logic array (PLA) performs the same basic function as a ROM. It is the most flexible device in the family of PLDs. The internal organization of a PLA is different from that of the ROM. The decoder of the ROM is replaced with an AND array that realizes elected product terms of the input variables. The AND array is followed by an OR array which Ors together the product terms needed to form the output functions. Both the AND and the OR arrays are programmable giving a lot of flexibility for implementing logic design.

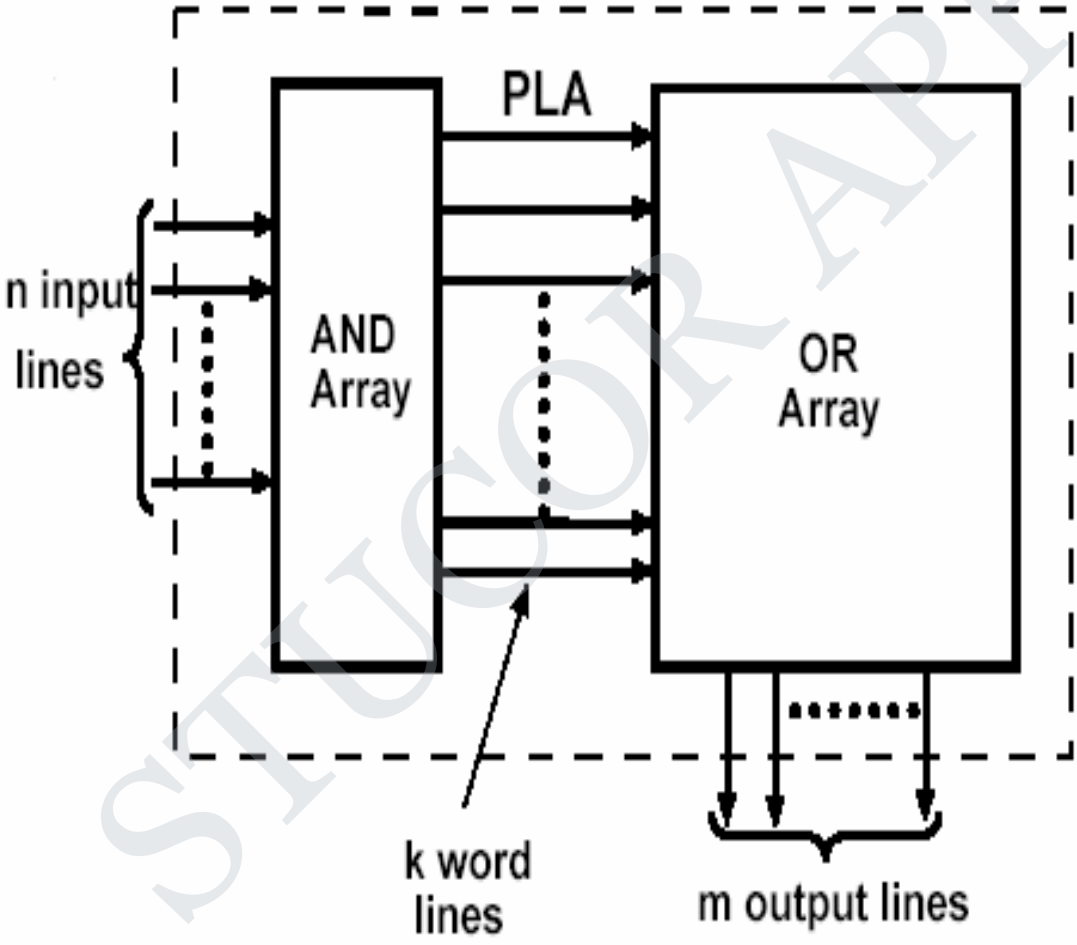



*Fig 8 – Internal Logic Diagram of a PLA*

Internally the PLA uses NOR-NOR logic but the added input and output inverting buffers make it equivalent to AND-OR logic. Logic gates are formed in the array by connecting NMOS switching transistors between the column line and row line. The transistors act as switches, so if the gate input is a logic zero, the transistor is turned off whereas if the gate input is a logic one, the transistor provides a conducting path to ground.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

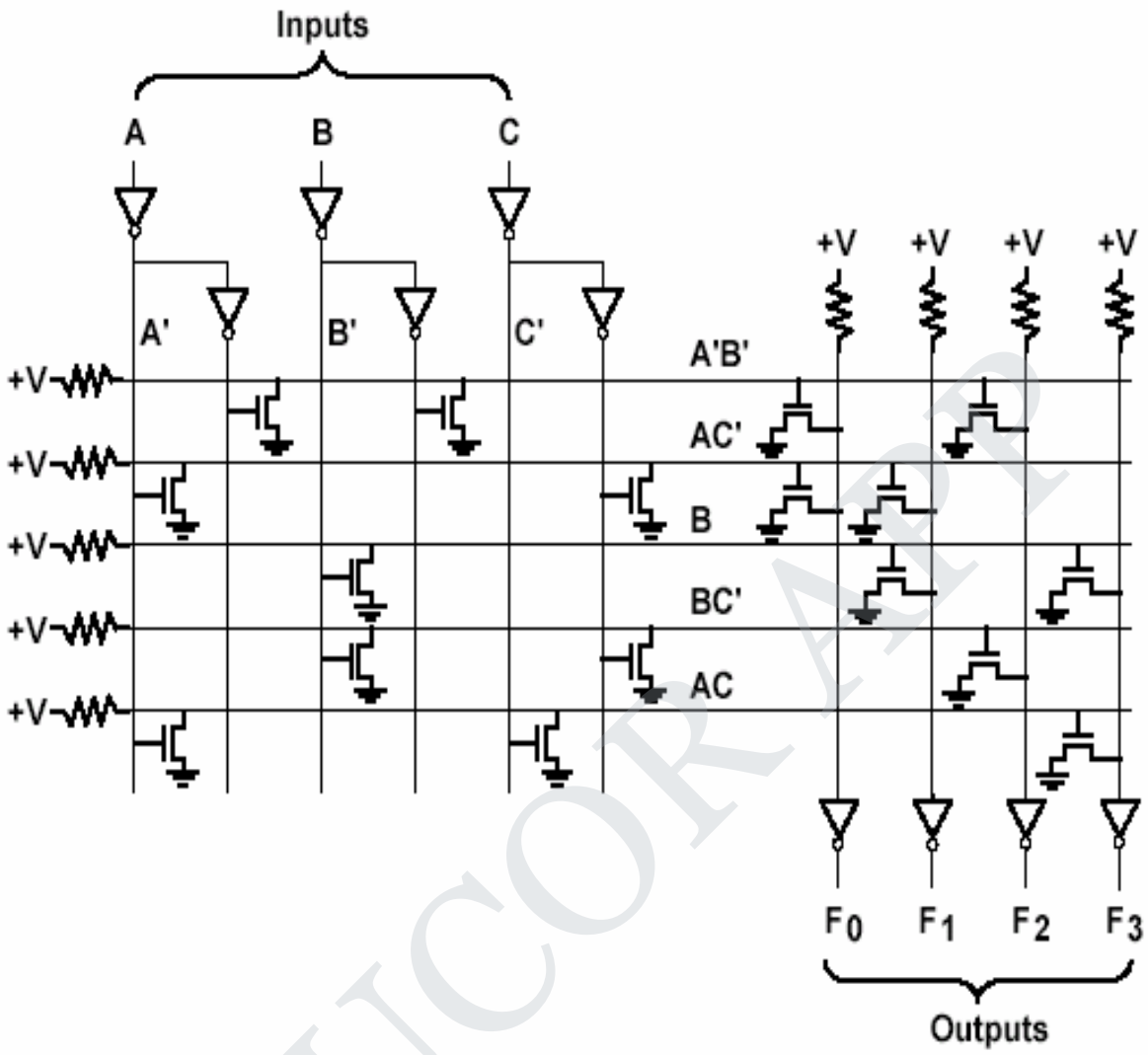



*Fig 9 –PLA with 3 inputs , 5 product terms and 4 outputs*

The above set of formulas are implemented using NOR-NOR logic of PLA as shown in Fig. 9 by placing the NMOS switching transistors wherever the connection has to be established. The same set of equations can be implemented using an AND-OR array equivalent as shown in Fig. 10 on the next page.

F0 = _m(0,1,4,6) = A'B'+AC'

F1 = _m(2,3,4,6,7) = B+AC'

F2 = _m(0,1,2,6) =A'B'+BC'



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

F3 = _m(2,3,5,6,7) =AC+B

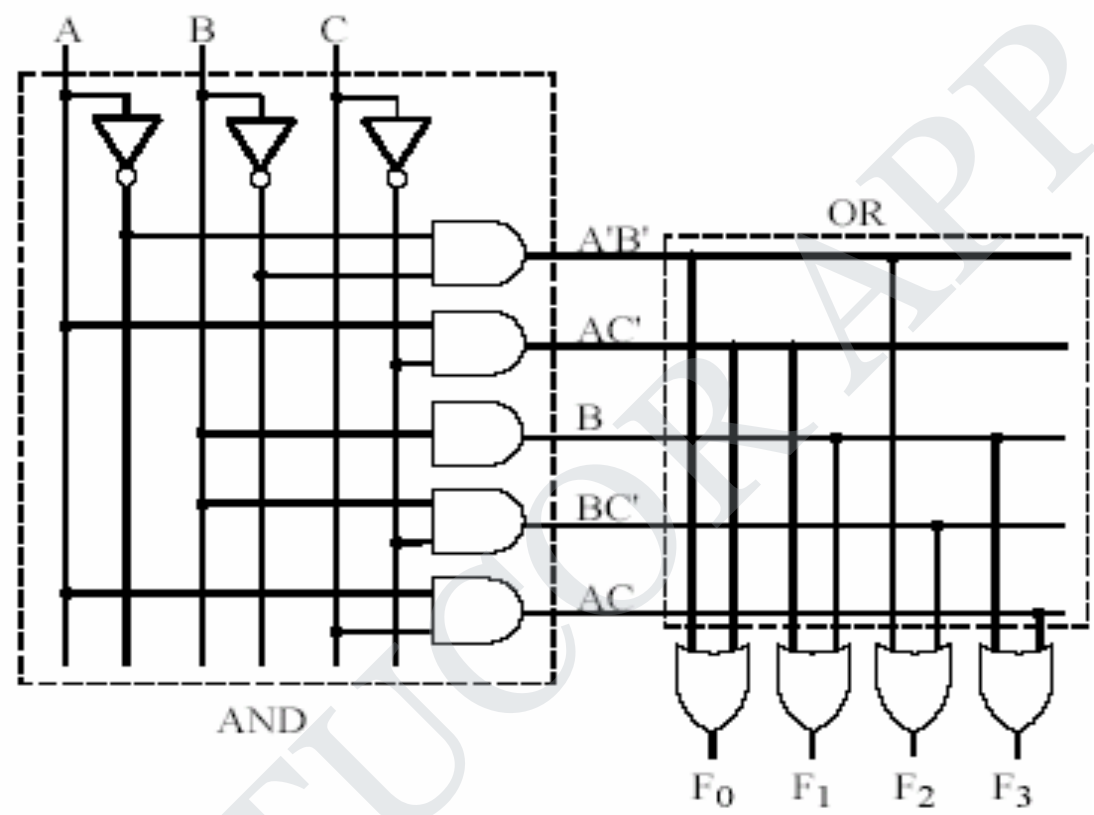



*Fig 10 –AND-OR array equivalent of Fig. 9*

The contents of a PLA can be specified by a modified truth table as shown in Fig.11. The input side of the table specifies the product terms . The symbols 0,1 and – indicate whether a variable is complemented, not complemented or not present in the corresponding product term. The output side of the table specifies which product terms appear in which output function. A 1 or 0 in the output terms indicate whether a given product term is present or not present in the corresponding output function.



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

| | Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|---|
| Product | A | B | C | F0 | F1 | F2 | F3 |
| A'B' | 0 | 0 | – | 1 | 0 | 1 | 0 |
| AC' | 1 | – | 0 | 1 | 1 | 0 | 0 |
| B | – | 1 | – | 0 | 1 | 0 | 1 |
| BC' | – | 1 | 0 | 0 | 0 | 1 | 0 |
| AC | 1 | – | 1 | 0 | 0 | 0 | 1 |

*fig 11 – PLA Table for Fig. 10*

The first row of the table indicates that the term A'B' is present in output functions F0 and F2. The second row indicates that AC' is present in F0 and F1. This PLA table can be written directly using the given set of equations to be realized using PLA logic.

**12. Realization of a given function using min number of rows in the PLA**

**F1 = ∑m(2,3,5,7,8,9,10,11,13,15) ….(1)**
**F2 = ∑m(2,3,5,6,7,10,11,14,15) …. (2)**
**F3 = ∑m(6,7,8,9,13,14,15) …. (3)**

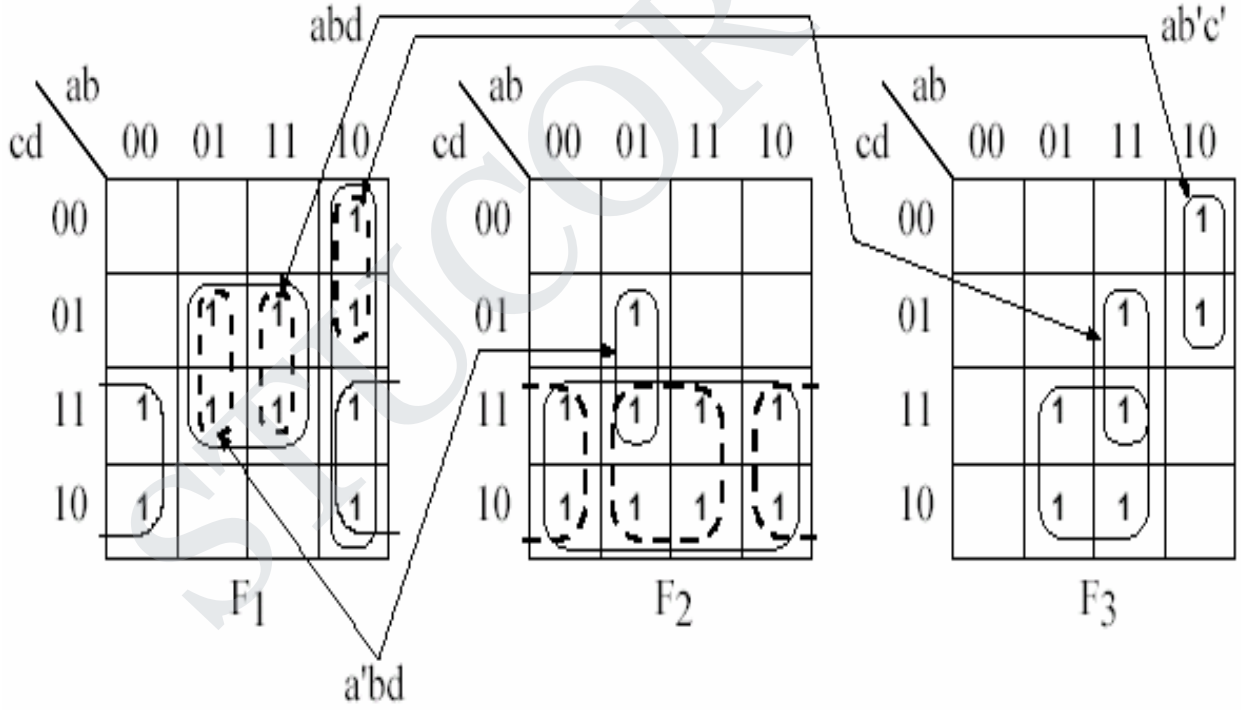



*Fig 12 – Multiple Output Karnaugh Map*

F1 = BD+B'C+AB' ….(4)
F2 = C+A'BD ….(5)



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

F3 = BC+AB'C'+ABD ….(6)

Equations 1,2 and 3 can be reduced to equations 4, 5 and 6 respectively using Karnaugh map as shown in Fig.12.

If we implement these reduced equations 4,5 and 6 in a PLA then a total of 8 different product terms (including C) are required. So instead of minimizing each function separately, we have to minimize the total number of rows in the PLA table.When we are trying to design a logic using PLA, the number of terms in each equation is not important since the size of the PLA does not depend on the number of terms. The term AB'C' is already present in function F3. So we can use it in F1 instead of AB' by writing AB' as AB'(C+C') . F1 can be now written as

F1 = BD + B'C + AB'(C+C')
= BD + B'C + AB'C + A'B'C'
= BD + B'C (1+ A) + A'B'C'
= BD + B'C + A'B'C'

This simplification of F1 eliminates the need to use a separate row for the original term AB' present in F1 of equation (4).

Since the terms A'BD and ABD are needed in F2 and F3 respectively, we can replace the term BD in F1 with A'BD + ABD. This eliminates the need for a row to implement the term BD in PLA. Similarly, since B'C and BC are used in F1 and F3 respectively, w can replace C in F3 with B'C + BC. Now the equations for F1, F2 and F3 with the above said changes can be written as :

F1 = BD(A+A') + B'C + AB'(C+C')
= ABD + A'BD + B'C +AB'C' ….(7)
F2 = C(B+B') +A'BD
= BC + B'C + A'BD ….(8)
F3 = BC+AB'C'+ABD ….(9)

The current equations for F1, F2 and F3 as shown in equations 7,8 and 9 respectively have only 5 different product terms. So the PLA table can now be written with only 5 rows. This is a significant improvement over the equations 4, 5 and 6, which resulted in 8 product terms. The reduced PLA table corresponding to equations 7, 8 and 9 is as shown in the figure 13.

| a | b | c | d | F1 | F2 | F3 |
|---|---|---|---|---|---|---|
| 0 | 1 | - | 1 | 1 | 1 | 0 |
| 1 | 1 | - | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | - | 1 | 0 | 1 |
| - | 0 | 1 | - | 1 | 1 | 0 |
| - | 1 | 1 | - | 0 | 1 | 1 |

*Fig 13 – Reduced PLA table*



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

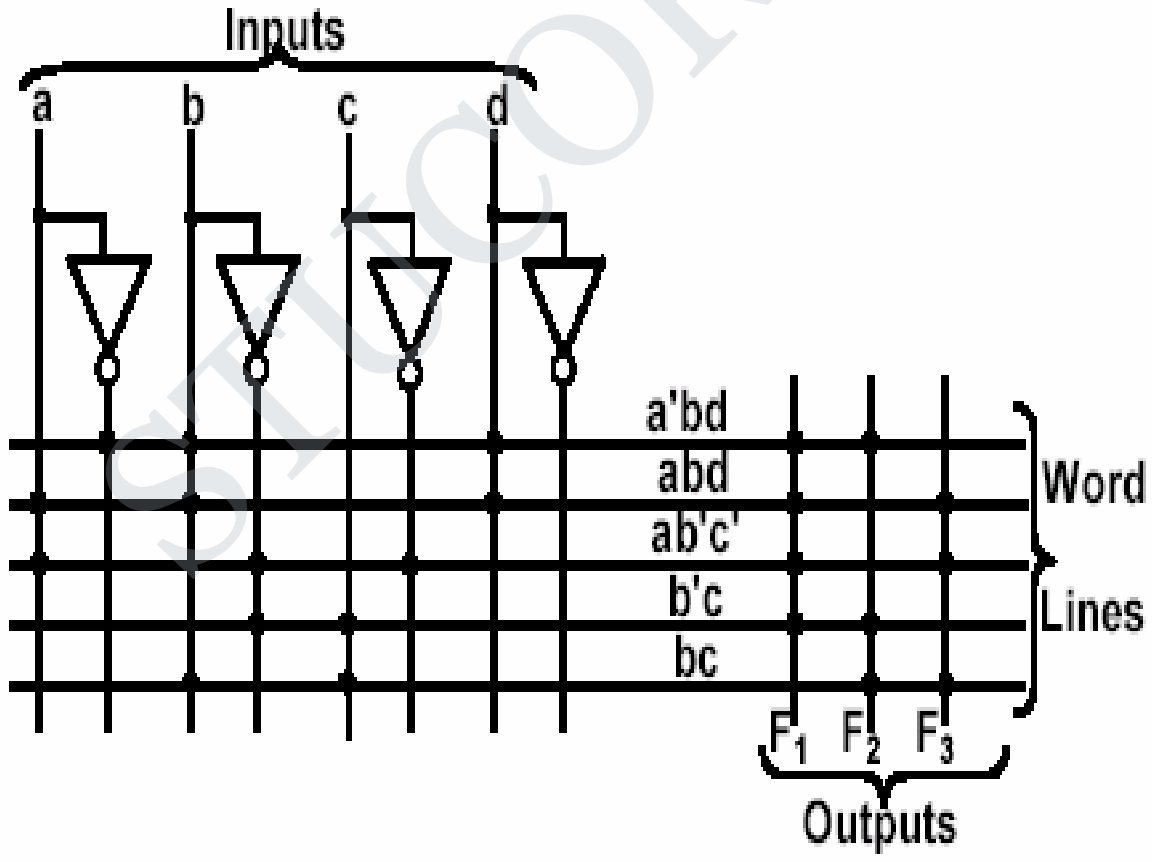
PLA table is significantly different from that of ROM truth table. In a truth table, each row represents a minterm ; therefore, one row will be exactly selected by each combination of input values. The 0s and 1s of the output portion of the selected row determine the corresponding output values.

On the other hand, each row in a PLA table represents a general product term. Therefore, 0, 1 or more rows may be selected by each combination of input values. To determine the value of F for a given input combination, the values of F in the selected rows of the PLA table must be ORed together.

For example, if ***abcd=0001*** is given as input , no rows are selected as this combination does not exist in the PLA table; and all the F outputs are 0. If abcd = 1001, only the third row is selected resulting in F1F2F3 = 101. If ***abcd = 0111,*** the first and the fifth rows are selected.

Therefore, to get the values for F1, F2 and F3 is got by ORing the respective values of F1, F2 and F3 in the corresponding rows resulting in F1 = 1 + 0 = 1, F2 = 1+1 = 1 and F3 = 0 + 1 = 1.

## PLA Realization of Equations






STUCOR APP
DOWNLOADED FROM STUCOR APP

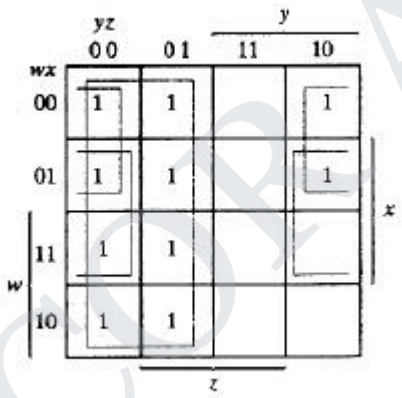
DOWNLOADED FROM STUCOR APP

*Fig 14–PLA Realization of Equations 7, 8 and 9.*

Fig 14 shows the PLA structure, which has four inputs, five product terms and three outputs as shown in equation 7,8 and 9. A dot at the intersection of word line and an input or output line indicates the presence of a switching element in the array.

**13. Simplify the Boolean function *F(w,x,y,z)* = S(0,1,2,4,5,6,8,9, 12, 13, 14)**

Since the function has four variables, a four-variable map must be used. The minterms listed in the sum are marked by 1's in the map of Fig. 8.9. Eight adjacent squares marked with 1's can be combined to form the one literal term *y'*. The remaining three 1 's on the right cannot be combined to give a simplified term. They must be combined as two or four adjacent squares. The larger the number of squares combined, the smaller the number of literals in the term.




**Map for F(w,x,y,z) = S(0,1,2,4,5,6,8,9,12,13,14) = y' + w'z' +xz'**

In this example, the top two l's on the right are combined with the top two l's on the left to give the term *w'z'*. Note that it is permissible to use the same square more than once. We are now left with a square marked by 1 in the third row and fourth column (square 1110). Instead of talking this square alone (which will give a term of four literals), we combine it with squares alone, ready used to form an area of four adjacent squares. These squares make up the two middle rows and the two end columns, giving the term xz'. The simplified function is

*F = y' + w'z' + xz'*

**Prime Implicants**

When choosing adjacent squares in a map, we must ensure that all the minterms of the function are covered when combining the squares. At the same time, it is necessary to minimize the number of terms in the expression and avoid any redundant terms whose minterms are already covered by other terms.

Sometimes there may be two or more expressions that satisfythe simplification criteria. The procedure for combining squares in the map may be made more systematic if we understand the meaning of the terms referred to as prime implicant and essential prime implicant. A *prime*



STUCOR APP
DOWNLOADED FROM STUCOR APP

DOWNLOADED FROM STUCOR APP

*implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map. If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be *essential.*

The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares. This means that a single 1 on a map represents a prime implicant if it is not adjacent to any other l's. Two adjacent 1 's form a prime implicant, provided that they are not within a group of four adjacent squares. Four adjacent l's form a prime implicant if they are not within a group of eight adjacent squares, and so on.

The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it. The prime implicant is essential if it is the only prime implicant that covers the minterm.

**PART C**

**ASSIGNMENT QUESTIONS**

**1. A combinational circuit is defined by the functions F0 = _m(0,1,4,6) = A'B'+AC',**

**F1 = _m(2,3,4,6,7) = B+AC', F2 = _m(0,1,2,6) =A'B'+BC', F3 = _m(2,3,5,6,7) =AC+B**

**Implement the circuit with a PLA having 3 inputs , 5 product terms and 4 outputs.**

1. **Explain the architecture of FPGA**
2. **Draw a RAM cell and explain its working**
3. **Describe the ROM Organization.**
4. **Explain EPROM and PLA.**



STUCOR APP

DOWNLOADED FROM STUCOR APP