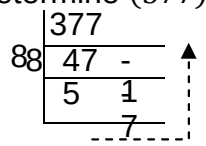
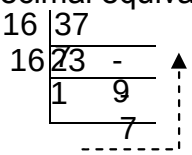
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
EE8351 DIGITAL LOGIC CIRCUITS
UNIT – I
NUMBER SYSTEM AND DIGITAL LOGIC FAMILIES Part – A

1. Determine $(377)_{10}$ in octal and Hexa-decimal equivalent.[N/D'14]





$(377)_{10}=(571)_8$ $(377)_{10}=(179)_{16}$

2. Compare the totem – pole output with open collector output. [N/D'14]

S.No.	Totem pole	Open Collector
1.	Output stage consists of pull-up transistor, diode resistor and pull-down transistor.	Output stage consists of only pull-down transistor.
2.	External pull-up resistor is not required	External pull-up resistor is required for proper operation of gates.
3.	Operating speed is high.	Operating speed is low.
4.	Output of two gates cannot be tied together.	Output of two gates can be tied together using wired AND technique.

3. Convert : [A/M'15]
   a) $(475.25)_8$ to its decimal equivalent.
   $=4x8^2+7x8^1+5x8^0+2x8^{-1}+5x8^{-2}$
   $=256+56+5+0.25+0.078125$
   $=(317.32814)_{10}$

   b) $(549.B4)_{16}$ to its binary equivalent.
   $=5x16^2+4x16^1+9x16^0+11x16^{-1}+4x16^{-2}$
   $=261+64+9+0.6875+0.01562$
   $=(334.703)_{10}$

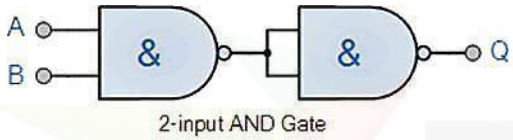
4. Define propagation delay. [A/M'15]
   Propagation delay is the average transition delay time for the signal to propagate from input to output then the signals change in value. It is expressed in ns.

5. What is unit distance code? Give an example. [N/D'15]
   Unit distance code is a non- weighted code in which next increment or decrement causes the bit transition only at one place. Ex: Gray code.

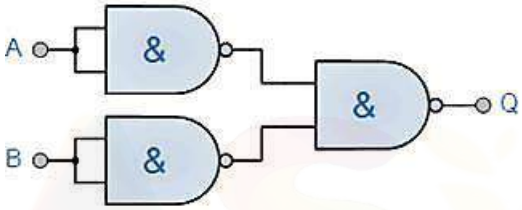
6. Define Fan-out. [N/D'15]

Fan- Out is defined as the maximum number of inputs of several gates that can be driven by the output of logic gate maintaining its output levels within the specified limits.

7. Construct OR gate and AND gate using NAND gate. [N/D'16]



2-input AND Gate



2-input OR Gate

8. Convert the following Excess-3 numbers into decimal numbers [N/D'16]
   a) 1011
   b) 1001 0011 0111

   a) Binary equivalent of Excess-3 1011 is →1000
      Decimal equivalent of 1000 → $(8)_{10}$

   b) Binary equivalent of Excess-3 1001 0011 0111 is →0110 0000 0100
      Decimal equivalent of 1000 → $(604)_{10}$

9. Reduce $a(b + b'c) + ab'$ [A/M'17]

$$a(b + b'c) + ab' = ab + ab'c + ab'$$
$$= a(b + b') + ab'c = a + ab'c$$

10. Convert 143_{10} into binary and binary coded decimal equivalent. [A/M'17]

Binary coded decimal equivalent → 143_{10} is 0001 0100 0011

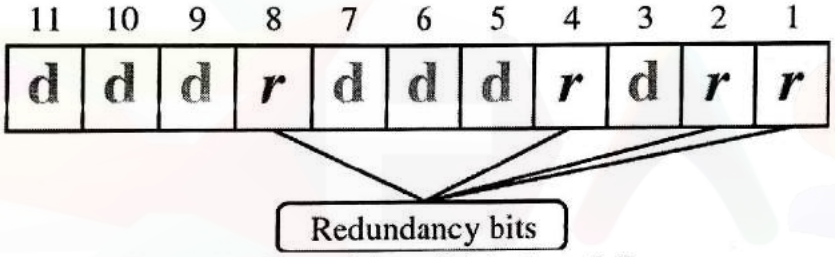
DIGITAL LOGIC CIRCUITS

UNIT – I
NUMBER SYSTEM AND DIGITAL LOGIC FAMILIES

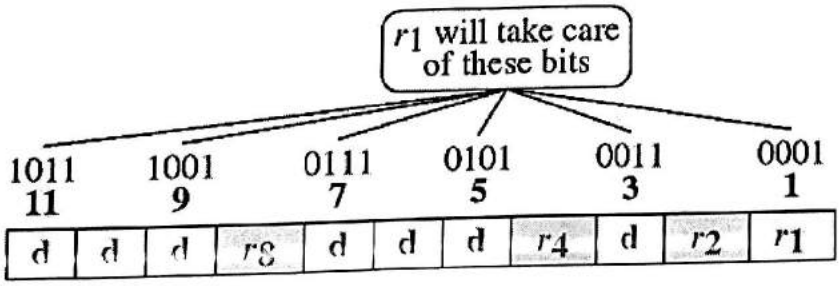
Part – B

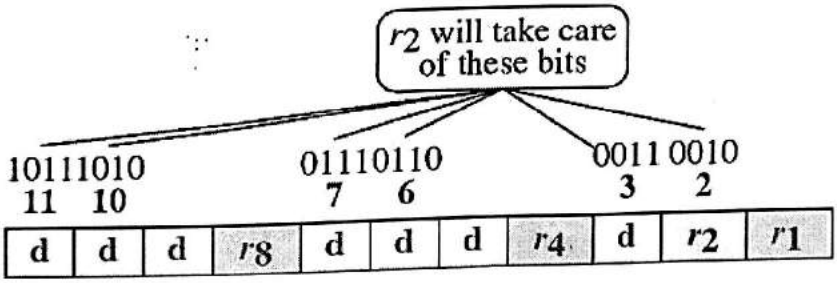
1. Explain Hamming code with an example. State its advantages over parity codes. (8) [N/D'14]

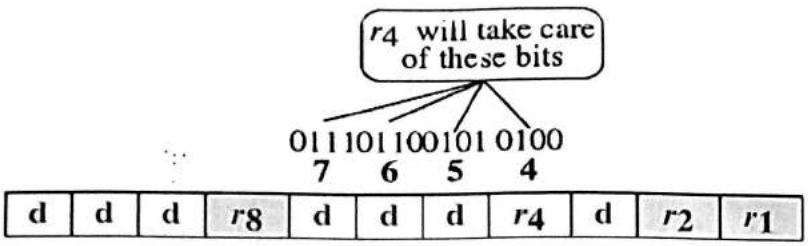
- Hamming code is a set of error-correction code s that can be used to detect and correct bit errors that can occur when computer data is moved or stored.
- Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors.
- Hamming codes are perfect codes for achieving the highest possible rate for codes with their block length and minimum distance called as min Hamming distance.
- A Hamming can be applied to the data unit of any length and uses the relationship between data and redundancy bits.
- For example the seven bit ASCII code requires 4 redundancy bits that can be added to the end of the data or between the original data bits.

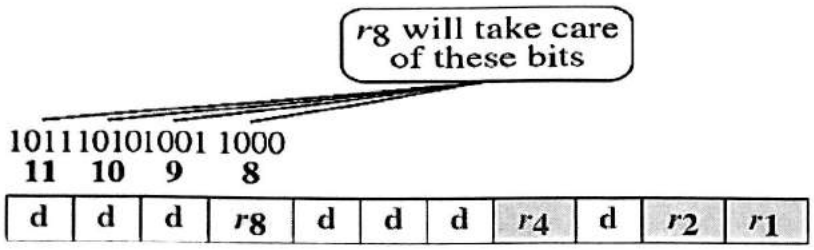


- In Hamming code, the position of r bits is as follows:
  - r_1 – bits 1,3,5,7,9,11
  - r_2 – bits 2,3,6,7,10,11
  - r_3 – bits 4,5,6,7
  - r_4 – bits 8,9,10,11
  - ➢ The position of r_1 is chosen for the values which has 1 in the right most position.
  - ➢ The position of r_1 is chosen for the values which has 1 in the second last position.
  - ➢ The position of r_1 is chosen for the values which has 1 in the third last position.
  - ➢ The position of r_1 is chosen for the values which has 1 in the fourth last position and so on.

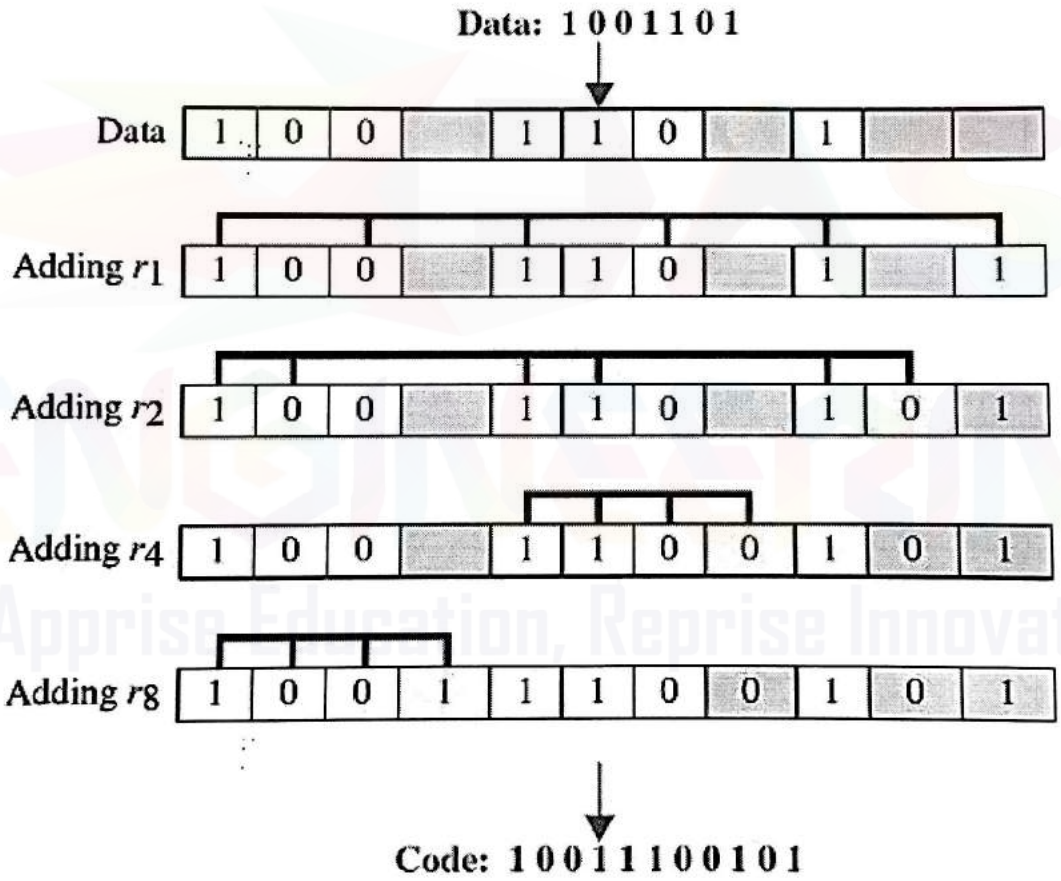








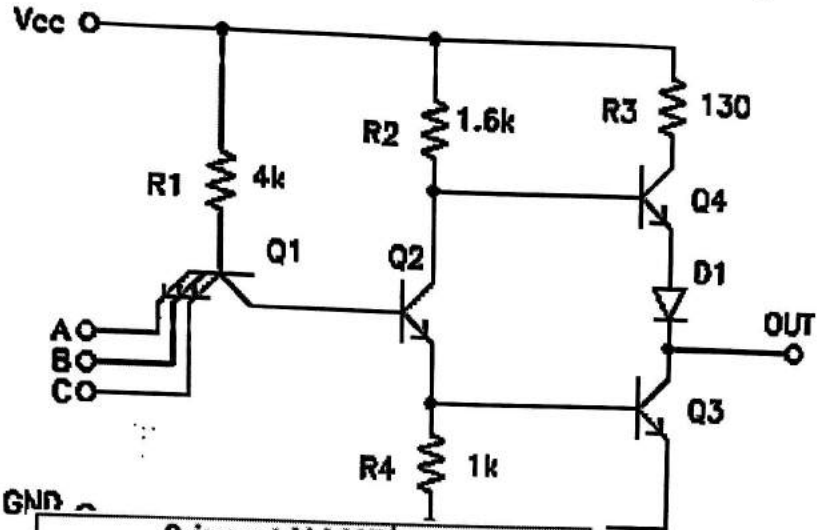
Example of Hamming Code:



Advantages of Hamming code over parity codes

- The advantage of Hamming code over a simple parity system that it can correct a single bit error and can also detect 2-bit errors.
- Whereas a parity system with one parity bit can detect single bit errors but cannot correct them.

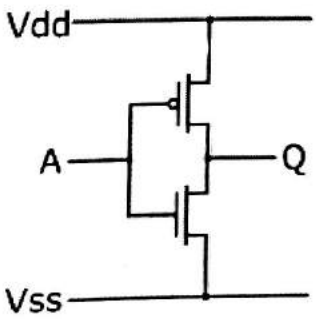
2. Design TTL logic circuit for a 3-input NAND gate. (8) [N/D'14]



3 input NAND gate				
A	B	C	A.B.C	$\overline{A.B.C}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

- If any input is low, the corresponding base-emitter junction becomes forward-biased and the transistor conducts.
- The other characteristics of the circuit and its transfer characteristic are identical to those of the inverter circuit.

3. Draw the MOS logic circuit for NOT gate and explain its operation. (8) [N/D'14]



- When a low voltage (0 V) is applied at the input, the top transistor (P-type) is conducting (switch closed) while the bottom transistor behaves like an open circuit.
- Therefore, the supply voltage (5 V) appears at the output.

- Conversely, when a high voltage (5 V) is applied at the input, the bottom transistor (N-type) is conducting (switch closed) while the top transistor behaves like an open circuit.
- Hence, the output voltage is low (0 V).
- The function of this gate can be summarized by the following table:

Input	Output
High (logic1)	Low (logic 0)
Low (logic 0)	High (logic1)

- The output is the opposite of the input – this gate inverts the input.
- One of the transistors will be an open circuit and no current flows from the supply voltage to ground.

4. With circuit schematic, explain the operation of a two input TTL NAND gate with totem-pole output. (10) [A/M'15]
5. With circuit schematic explain the working of a two – input TTL NAND gate. (7) [A/M'17]
   - The circuit structure is identical to TTL inverter circuit except for the multiple emitter input transistor.
   - This is used to implement a diode switching structure in active transistor form using parallel junction diffusions for several emitters.

Fig. 3.1 Multiple Input Emitter Structure of TTL

- If any input is low, the corresponding base-emitter junction becomes forward-biased and the transistor conducts.
- The other characteristics of the circuit and its transfer characteristic are identical to those of the inverter circuit.

Logical Operation

- The direction of conduction of T1 can be in the forward or reverse mode so this should also be noted in the table.
- It can be seen from the table that the output goes LO only when both inputs are HI which verifies the NAND function.

IN1	IN2	T_1	T_2	T_3	T_4	D	Output
LO	LO	ON	OFF	OFF	ON	ON	HI
LO	HI	ON	OFF	OFF	ON	ON	HI
HI	LO	ON	OFF	OFF	ON	ON	HI
HI	HI	ON	ON	ON	OFF	OFF	LO

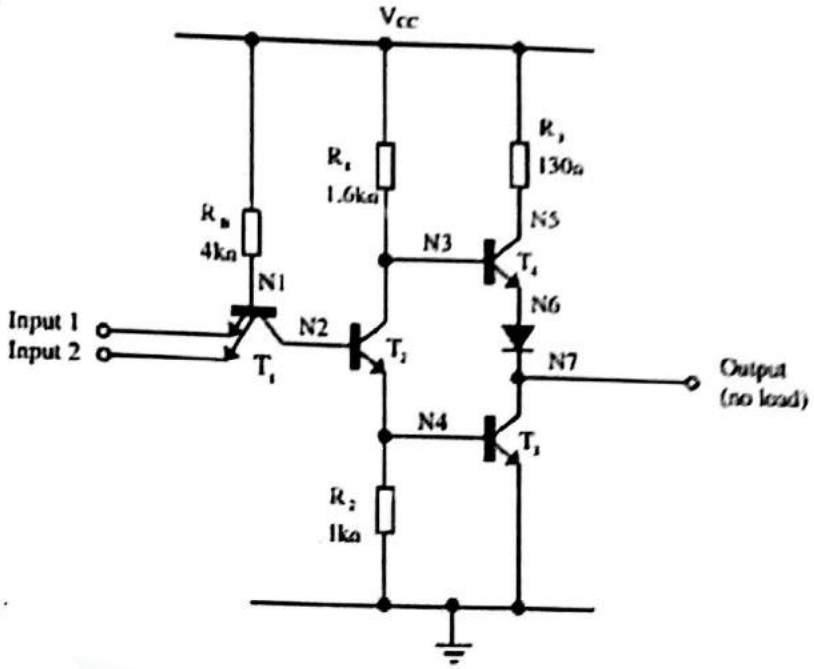
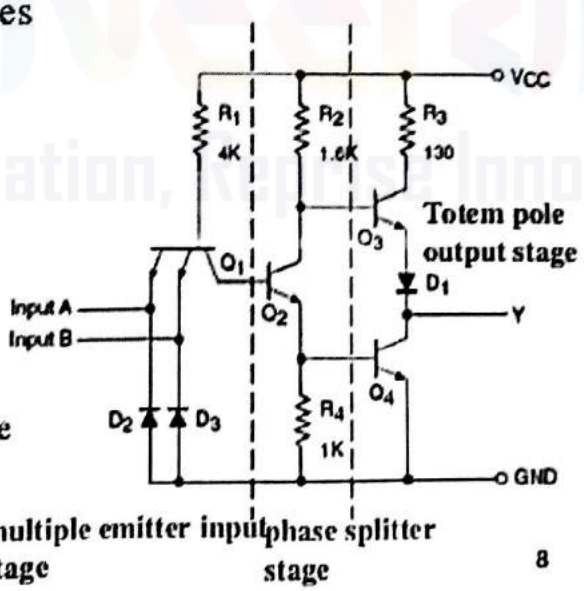


Fig. 3.2 Circuit Diagram of a Standard 2-input TTL NAND Gate

6. Compare totem pole and open collector outputs. (6) [A/M'15]
7. Compare Totem pole and open collector outputs. (6) [A/M'17]

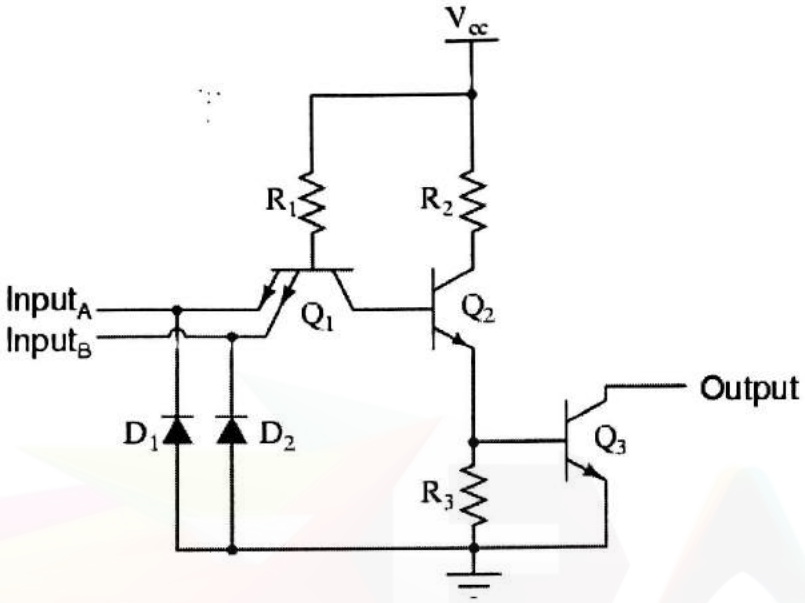
NAND gate with Totem pole output

- Below is the circuit of a totem-pole NAND gate, which has got three stages
  - Input Stage
  - Phase Splitter Stage
  - Output Stage



Standard TTL NAND gate

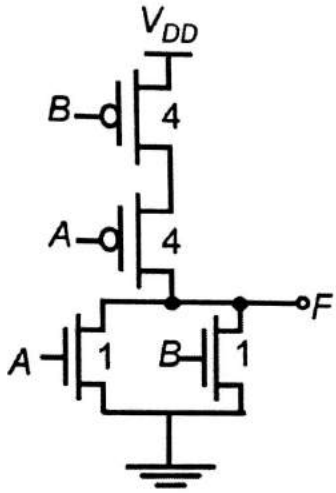
NAND gate with collector output



S.No.	Totem pole	Open Collector
1.	Output stage consists of pull-up transistor, diode resistor and pull-down transistor.	Output stage consists of only pull-down transistor.
2.	External pull-up resistor is not required	External pull-up resistor is required for proper operation of gates.
3.	Operating speed is high.	Operating speed is low.
4.	Output of two gates cannot be tied together.	Output of two gates can be tied together using wired AND technique.

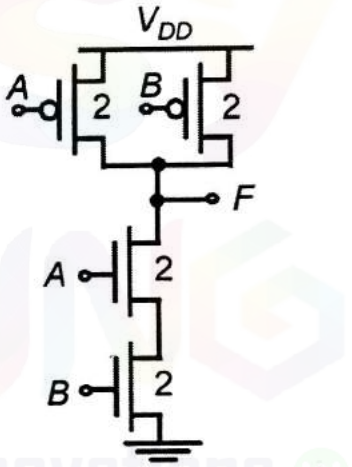
8. Draw the CMOS logic circuit for NOR gate and explain its operation. (8) [N/D'15]
   - The circuit has two inputs and one output.
   - Whenever at least one of the inputs is high, the corresponding N-type transistor will be closed while the P-type transistor will be open.
   - Consequently, the output voltage will be low.
   - Conversely, if both inputs are low, then both P-type transistors at the top will be closed circuits and the N-type transistors will be open.
   - Hence, the output voltage is high.
   - The function of this gate can be summarized by the following table:

Input (A)	Input (B)	Output
Low	Low	High
Low	High	Low
High	Low	Low
High	High	Low



- If logical 1's are associated with high voltages then the function of this gate is called NOR for negated OR.
- There is never a conducting path from the supply voltage to ground.

9. Explain with an aid of circuit diagram the operation of 2 input CMOS NAND gate and list out its advantages over other logic families. (10) [N/D'16]
   - The circuit has two inputs and one output.
   - Whenever at least one of the inputs is low, the corresponding P-type transistor will be conducting while the N-type transistor will be closed.
   - Consequently, the output voltage will be high.
   - Conversely, if both inputs are high, then both P-type transistors at the top will be open circuits and both N-type transistors will be conducting.
   - Hence, the output voltage is low.
   - The function of this gate can be summarized by the following table:



Input (A)	Input (B)	Output
Low	Low	High
Low	High	High
High	Low	High
High	High	Low

- If logical 1's are associated with high voltages then the function of this gate is called NAND for negated AND.
- There is never a conducting path from the supply voltage to ground.

Advantages of CMOS over other logic families

a) Low power consumption
b) High fan out
c) Temperature stability
d) Noise immunity

10. Briefly discuss weighted Binary code. (4) [N/D'15]

Weighted Codes:

- The main characteristic of a weighted code is, each binary bit is assigned by a "weight" and values depend on the position of the binary bit.
- The sum of the weights of these binary bits, whose value is 1 is equal to the decimal digit which they represent.
- In other words, if w1, w2, w3 and w4 are the weights of the binary digits, and x1, x2, x3 and x4 are the corresponding bit values, then the decimal digit N=w4x4 + w3x3+w2x2+w1x1 is represented by the binary sequence x4x3x2x1.
- A sequence of binary bits which represents a decimal digit is called a "code word".
- Thus x4x3x2x1 is a code word of N.
- Example of these codes is: BCD, 8421, 6421, 4221, 5211, 3321 etc.
- Weighted codes are used in:
  a) Data manipulation during arithmetic operation.
  b) For input/output operations in digital circuits.
  c) To represent the decimal digits in calculators, volt meters etc.

Encode the binary word 1011 into seven bit even parity Hamming code.

(8) [A/M'15]

Solution:

Given the binary word

b_3	b_2	b_1	b_0
1	0	1	1

The 7-bit Hamming (7, 4) code word $h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7$ associated with a 4-bit binary number $b_3\ b_2\ b_1\ b_0$ and is given as

$h_1 = b_3 \oplus b_2 \oplus b_0 = 1 \oplus 0 \oplus 1 = 0$

$h_2 = b_3 \oplus b_1 \oplus b_0 = 1 \oplus 1 \oplus 1 = 1$

$h_3 = b_3 = 0$

$h_4 = b_2 \oplus b_1 \oplus b_0 = 0 \oplus 1 \oplus 1 = 0$

$h_5 = b_2 = 0$

$h_6 = b_1 = 1$

$h_7 = b_0 = 1$

∴ The 7-bit Even parity Hamming code is given as

h_1	h_2	h_3	h_4	h_5	h_6	h_7
0	1	0	0	0	1	1

Perform the following addition using BCD and Excess-3 addition

(205 + 569) (8) [1/M's]

Solution :

BCD Addition

2 0 5	0010	0000	0101	
5 6 9	0101	0110	1001	
7 7 4	0111	0110	1110	1110 > 9
		↓	0110	
		0110 [1]	0100	
	↓	1↙	↓	
	0111	0111	0100	
	7	7	4	

Excess-3 Addition

Excess-3 code of 205

↓

0101 0011 1000

Excess-3 code of 569

↓

1000 1001 1100

205 → 0101 0011 1000

569 → 1000 1001 1100

1101 1100 0100

1 ↓

1101 0100

0011

1101 1101 0111

0011 0011

1010 1010 0111

10 10 7 → Excess-3 Code

Add 3 to correct 0000

Binary equivalent of Excess-3 code

1010 1010 0111

↓ ↓ ↓

0111 0111 0100

↓ ↓ ↓

7 7 4

205

569

774

A 12-bit Hamming code word containing 8-bits of data and 4 parity bits is read from memory. what was the original 8-bit data word that has been written into memory if the 12-bit word read out is as

(1) 101110010100

(2) 111111110100

(12) [N/D'15]

Solution:

(1) 101110010100

12-bit Hamming code is given as

D_{12}	D_{11}	D_{10}	D_9	P_8	D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	1	1	0	0	1	0	1	0	0

P_1, P_2, P_3 & P_4 are parity bits while the remaining bits are data-bits.

∴ 8-bit data is 10110011

(2) 111111110100

12-bit Hamming code

D_{12}	D_{11}	D_{10}	D_9	P_8	D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	1	1	1	1	1	1	1	0	1	0	0

8-bit data is 11111111

UNIT – II
COMBINATIONAL CIRCUITS
Part – B

1. Design a 4-Bit binary to gray code converter and implement it using logic gates. (8)[N/D'14]

- The logical circuit which converts binary code to equivalent gray code is known as **binary to gray code converter**.
- The gray code is a non weighted code.
- The successive gray code differs in one bit position only that means it is a unit distance code. It is also referred as cyclic code. It is not suitable for arithmetic operations. It is the most popular of the unit distance codes. It is also a reflective code.

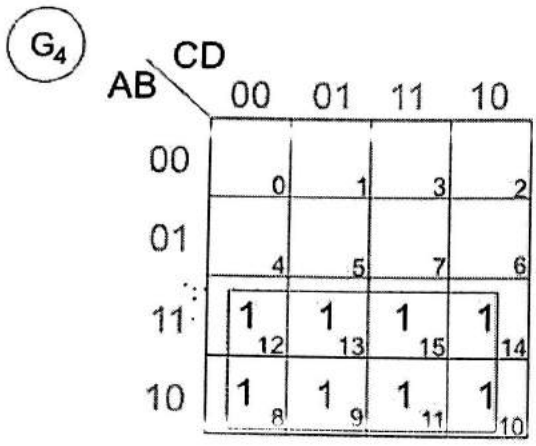
Decimal Number	4 bit Binary Number				4 Bit Gray code			
	A	B	C	D	G_1	G_2	G_3	G_4
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	0
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

That means, in 4 bit gray code, (4-1) or 3 bit code is reflected against the axis drawn after $(2^{4-1})^{th}$ or 8^{th} row.

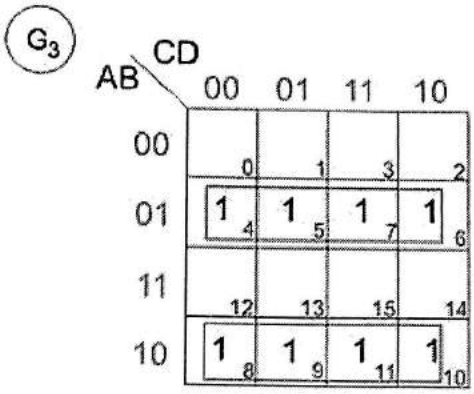
The bits of 4 bit gray code are considered as $G_4G_3G_2G_1$. Now from conversion table,

$$G_4 = \sum m(8,9,10,11,12,13,14,15),\ G_3 = \sum m(4,5,6,7,8,9,10,11)$$
$$G_2 - \sum m(2,3,4,5,10,11,12,13),\ G_1 = \sum m(1,2,5,6,9,10,13,14)$$

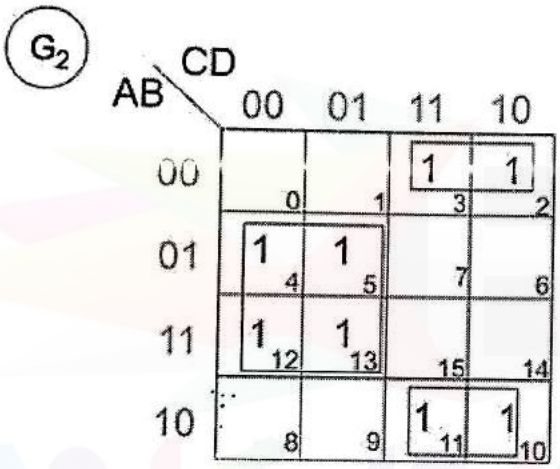
K-map Simplification



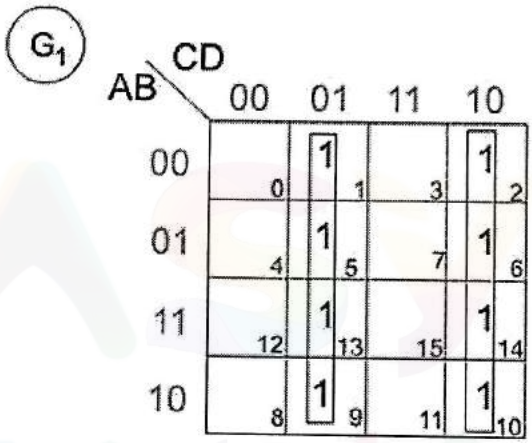
$G_4 = A$



$G_3 = \bar{A}B + A\bar{B} = A \oplus B$

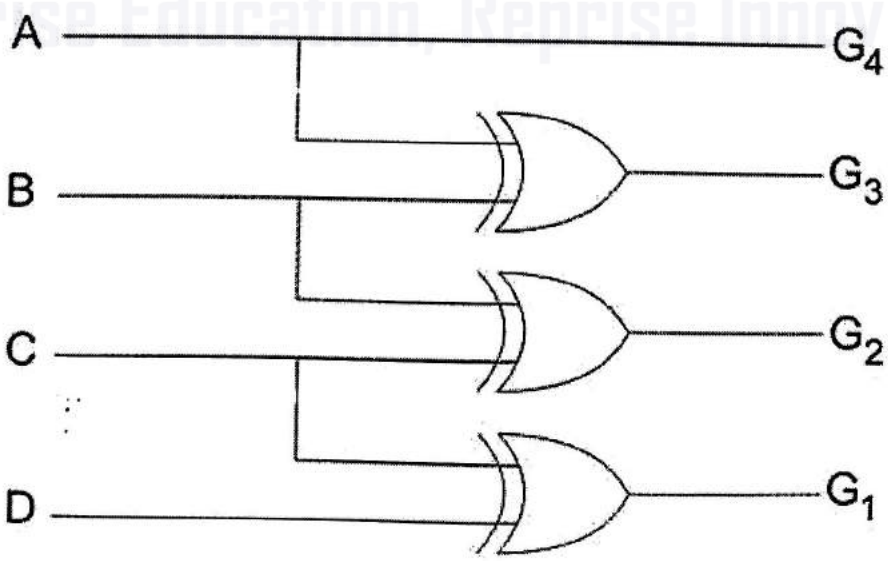


$G_2 = B\bar{C} + \bar{B}C = B \oplus C$



$G_1 = \bar{C}D + C\bar{D} = C \oplus D$

Logic Diagram:



Logic Circuit for Binary to Gray Code Converter

2. Design a full subtractor and implement it using logic gates. (8)[N/D'14]

Full Subtractor

- Full subtractor performs subtraction of two bits, one is minuend and other is subtrahend.
- The input of a full subtractor consists of three bits (A, B and Bin) and two outputs are DIFFERENCE output (D) and BORROW (Bout)
- Full subtractor performs subtraction of two bits, one is minuend and other is subtrahend. In full subtractor '1' is borrowed by the previous adjacent lower minuend bit.

Truth Table

Input			Output	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map Simplification

For D

A \ BB_{in}	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

For B_{out}

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

$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$

$B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$

Fig. 3.21 Maps for full-subtractor

Logic Diagram using logic gates

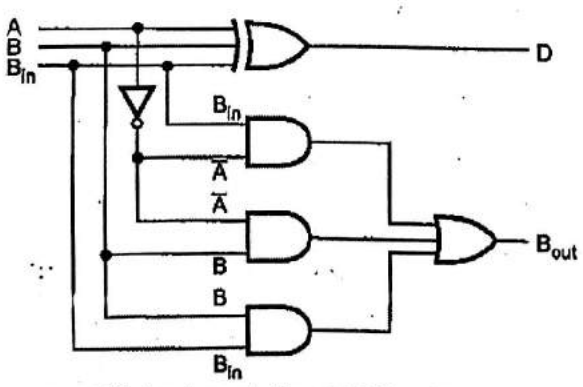


Fig. 3.23 Implementation of full-subtractor

Logic Diagram using half subtractors

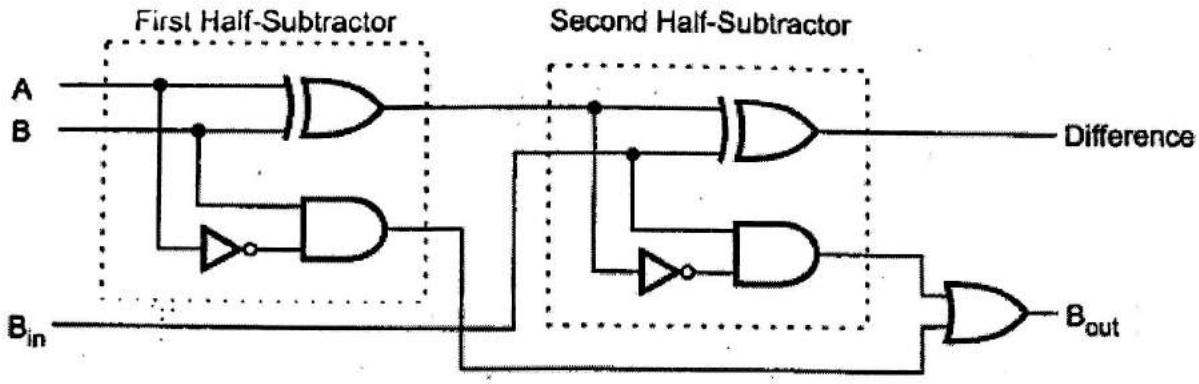


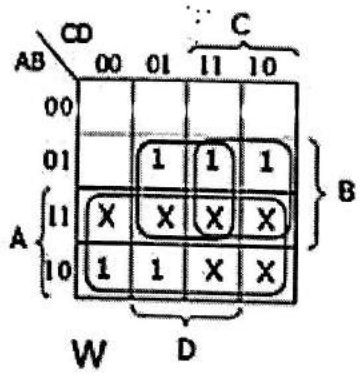
Fig. 3.24 Implementation of a full-subtractor with two half-subtractors and an OR gate

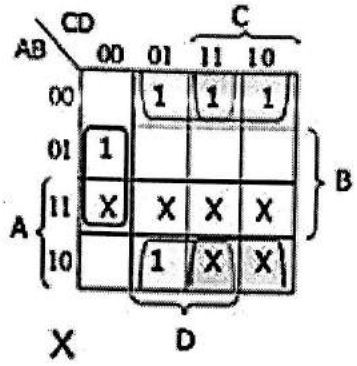
3. Design a BCD to Exces-3 code converter. (8) [A/M'15]

Truth table :

	BCD				Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X

K-map Simplification



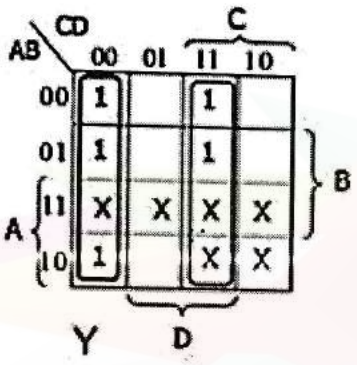


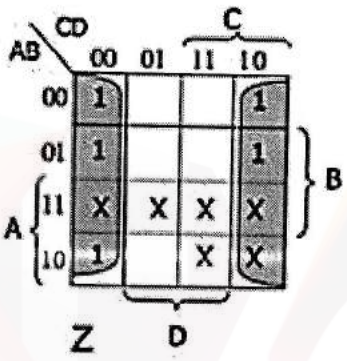
$W = A + B.C + B.D$

$X = B'.C + B'.D + B.C'.D'$

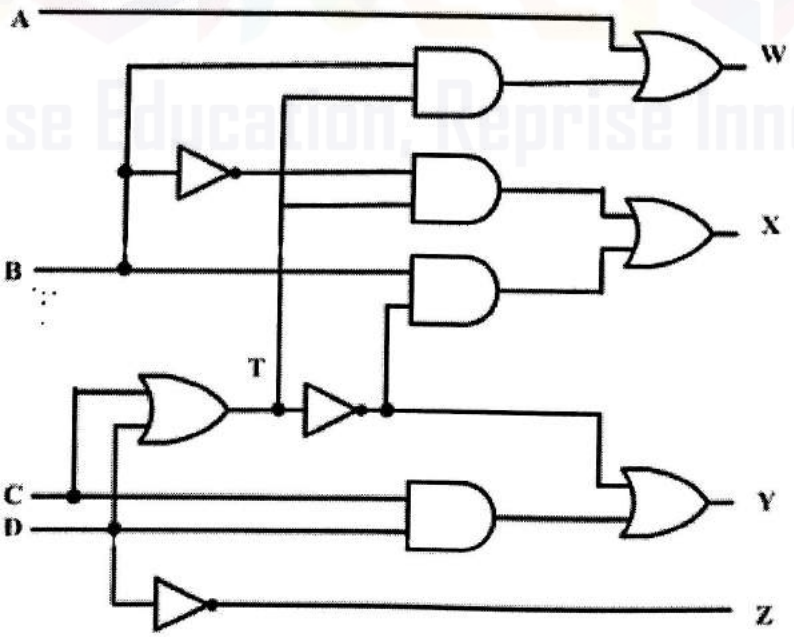
$Y = C.D + C'.D'$

$Z = D'$





Logic Diagram of BCD to Excess-3 code converter

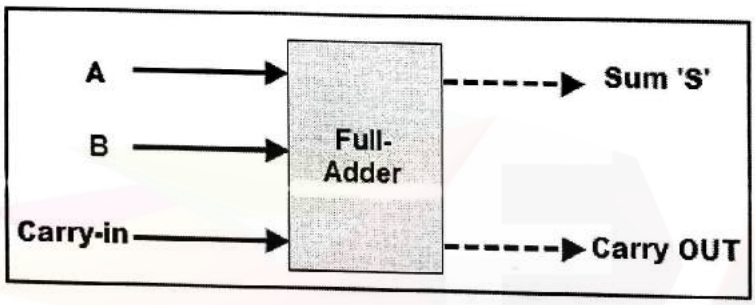


4. Design a full adder using two half adders and an OR gate. (8)[A/M'15]
5. Design a full adder using only NOR gates. (6) [A/M'17]

- An adder is a digital circuit that performs addition of numbers.
- The half adder adds two binary digits called as augend and addend and produces two outputs as sum and carry; XOR is applied to both inputs to produce sum and AND gate is applied to both inputs to produce carry.

Full Adder

- The full adder adds 3 one bit numbers, where two can be referred to as operands and one can be referred to as bit carried in. And produces 2-bit output, and these can be referred to as output carry and sum.
- The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs.
- The first two inputs are A and B and the third input is an input carry as C-IN.



Full Adder Truth Table:

INPUTS			OUTPUT	
A	B	C-IN	C-OUT	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

K-map Simplification

For Carry (C_{out})

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

For Sum

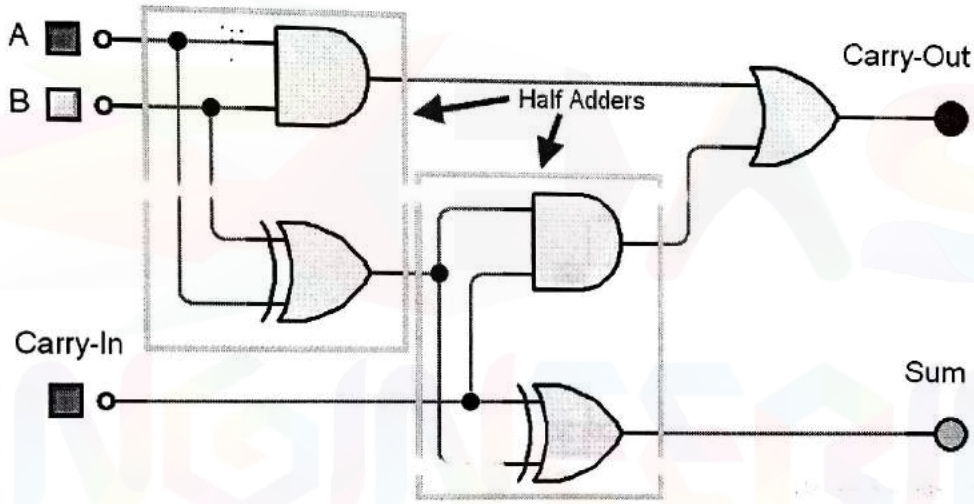
A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$C_{out} = AB + A C_{in} + B C_{in}$

$Sum = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$

Fig. 3.15 Maps for full-adder

Logic Diagram using half adders



6. Design a 3 x 8 decoder and explain its operation as a minterm generator. (7) [A/M'17]

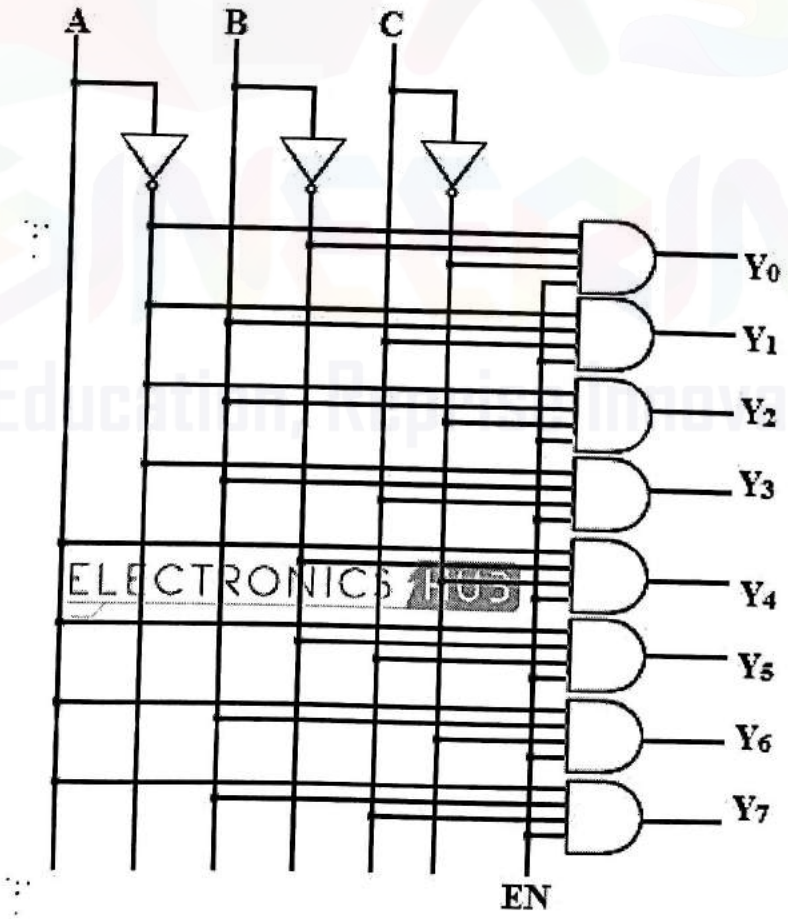
Truth table:

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

Logic function:

$$Y_0 = A'B'C'$$
$$Y_1 = A'B'C$$
$$Y_2 = A'BC'$$
$$Y_3 = A'BC$$
$$Y_4 = AB'C'$$
$$Y_5 = AB'C$$
$$Y_6 = ABC'$$
$$Y_7 = ABC$$

Logic Diagram:



7. Design a full subtractor and realize using logic gates. (8) [N/D'15]
8. Design a full subtractor and realize using logic gates. Also implement the same using half subtractors. (13) [N/D'16]

Truth Table:

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 3.9 Truth table for full-subtractor

K-map simplification:

For D

A \ BB_{in}	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

$D = \bar{A}B\bar{B}_{in}+\bar{A}\bar{B}B_{in}+A\bar{B}\bar{B}_{in}+ABB_{in}$

For B_{out}

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

$B_{out} = \bar{A}B_{in}+\bar{A}B+BB_{in}$

Fig. 3.21 Maps for full-subtractor

Full subtractor using two half subtractors

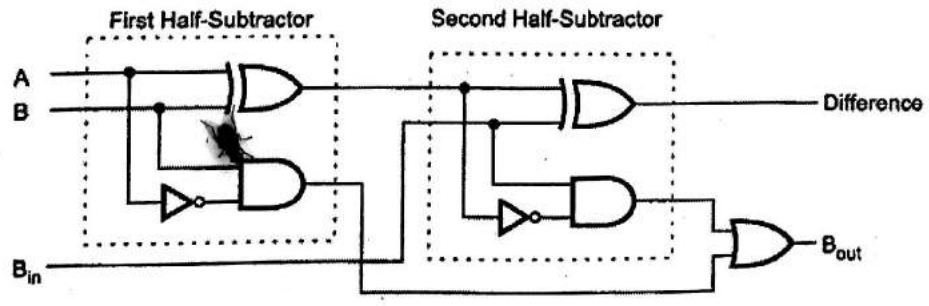


Fig. 3.24 Implementation of a full-subtractor with two half-subtractors and an OR gate

Full subtractor using logic gates

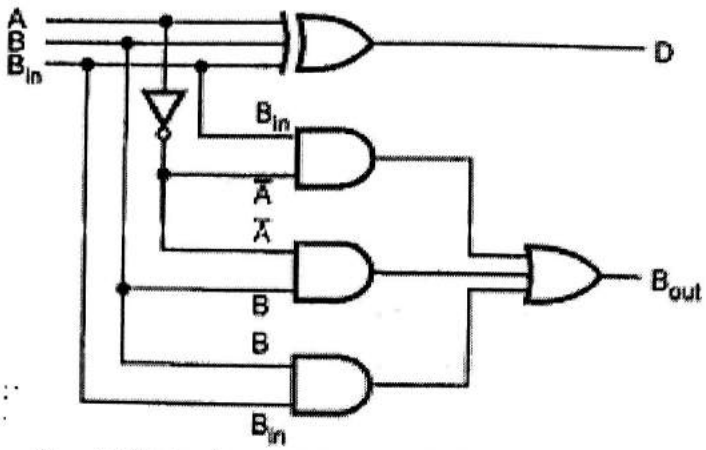
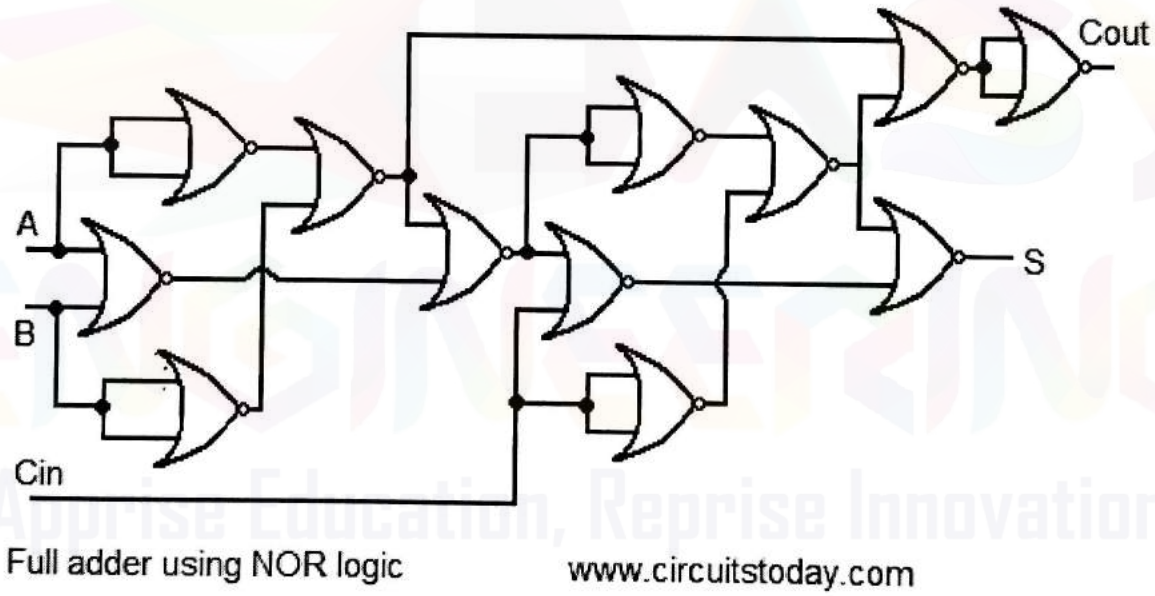


Fig. 3.23 Implementation of full-subtractor

9. Design a full adder using only NOR gates. (6) [A/M'17]



Full adder using NOR logic www.circuitstoday.com

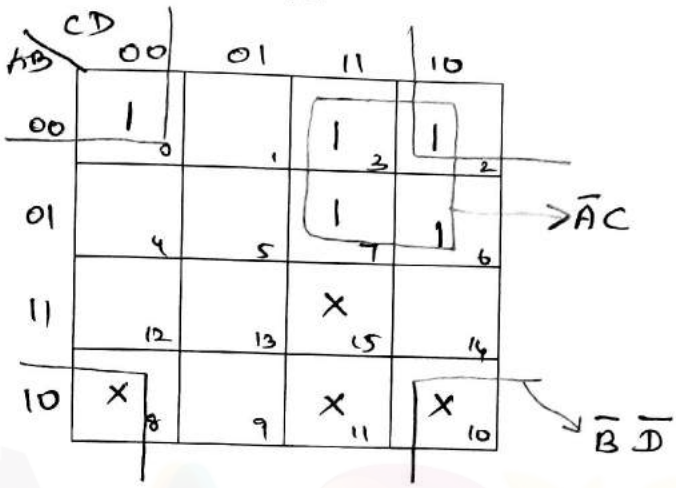
Simplify the logical expression using k-map in SOP and POS form. (13) [N/D'16]

$F(A,B,C,D) = \Sigma m(0,2,3,6,7) + d(8,10,11,15)$

Truth Table:

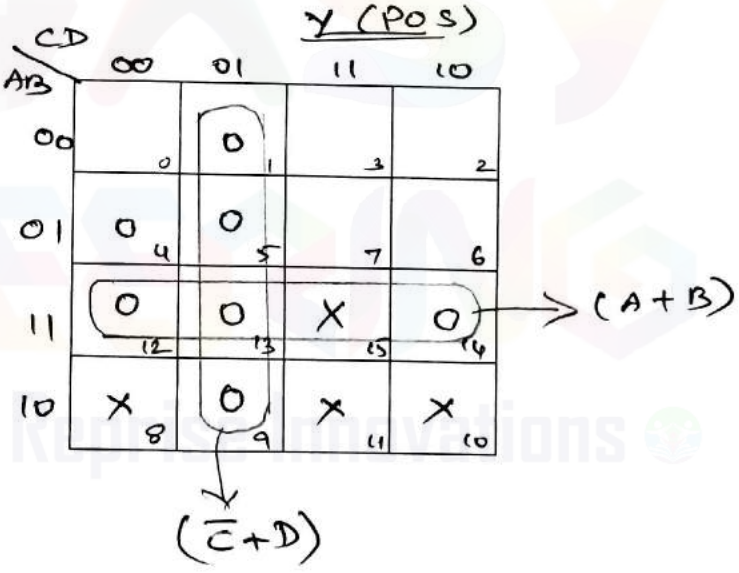
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	X
1	0	0	1	0
1	0	1	0	X
1	0	1	1	X
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	X

Y (SOP)



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

Y (POS)



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

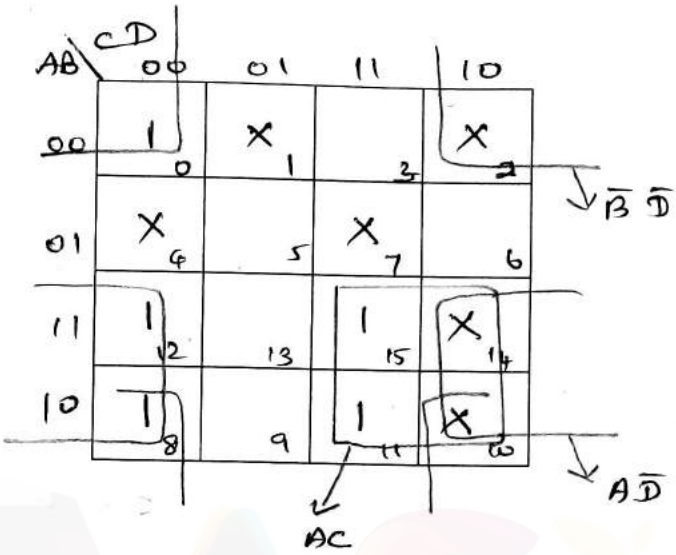
Y in SOP form	$y = \bar{B}\bar{D} + \bar{A}C$
y in POS form	$y = (A+B)(\bar{C}+D)$

Simplify the Boolean function using k-map and implement using only NAND gates

$F(A, B, C, D) = \Sigma m(0, 8, 11, 12, 15) + \Sigma d(1, 2, 4, 7, 10, 14)$

Make the essential and non-essential prime implicants. (8) [N/D'15]

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	X
0	0	1	0	X
0	0	1	1	0
0	1	0	0	X
0	1	0	1	0
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	X
1	1	1	1	1



$f = AC + A\bar{D} + \bar{B}\bar{D}$

Essential Prime Implicants:

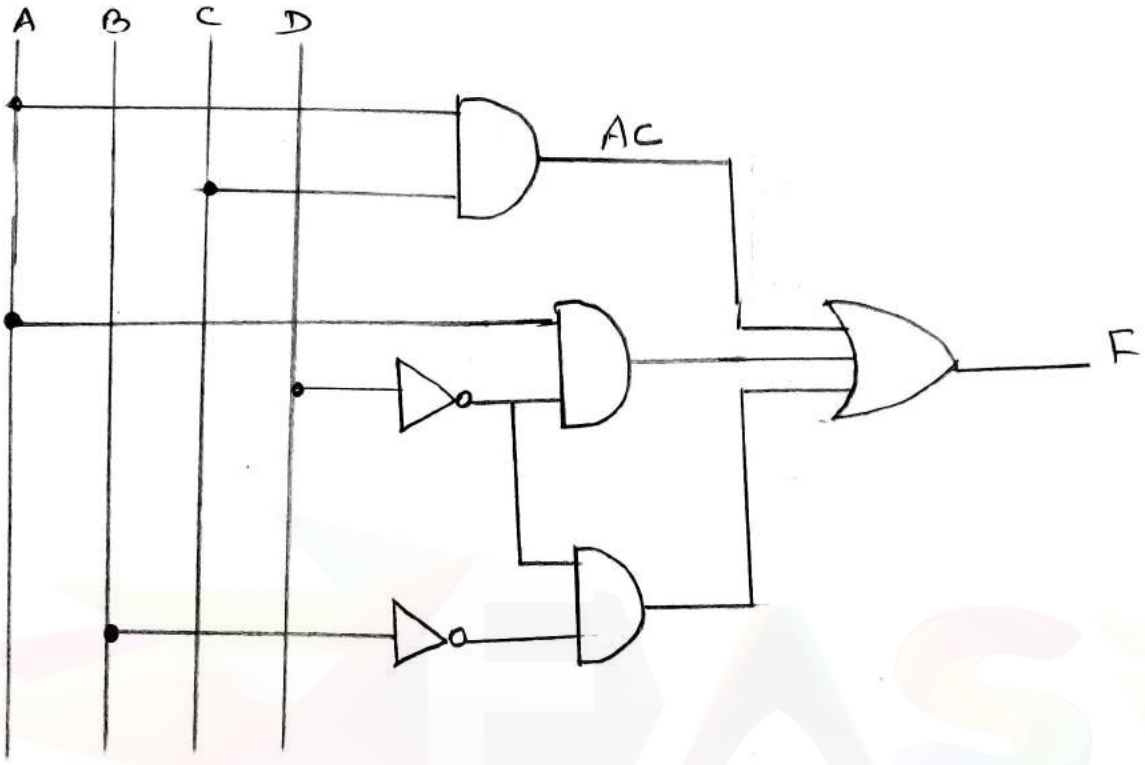
AC, $A\bar{D}$

Non-essential Prime Implicants

$\bar{B}\bar{D}$

Logic Diagram

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

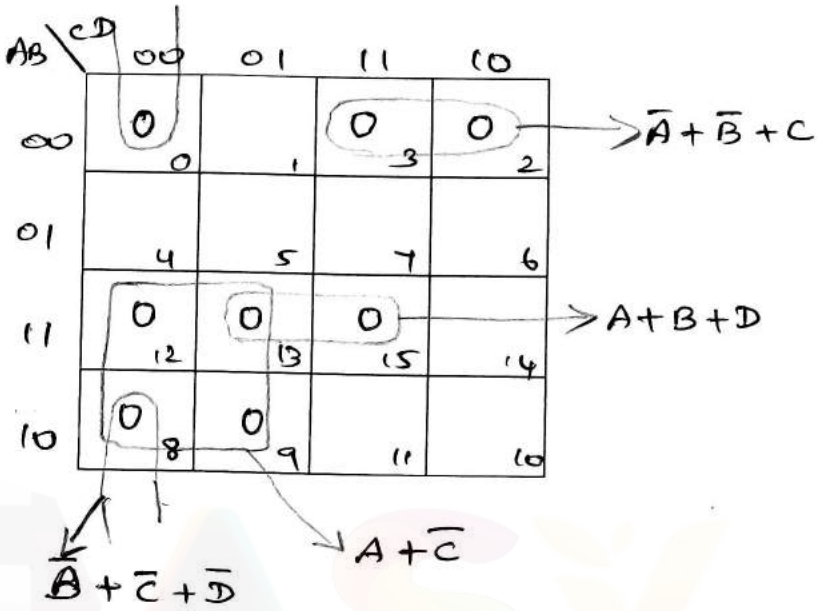


Reduce the following function using k-map

$F(A, B, C, D) = \prod M(0, 2, 3, 8, 9, 12, 13, 15)$

(8) [A/M'15]

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

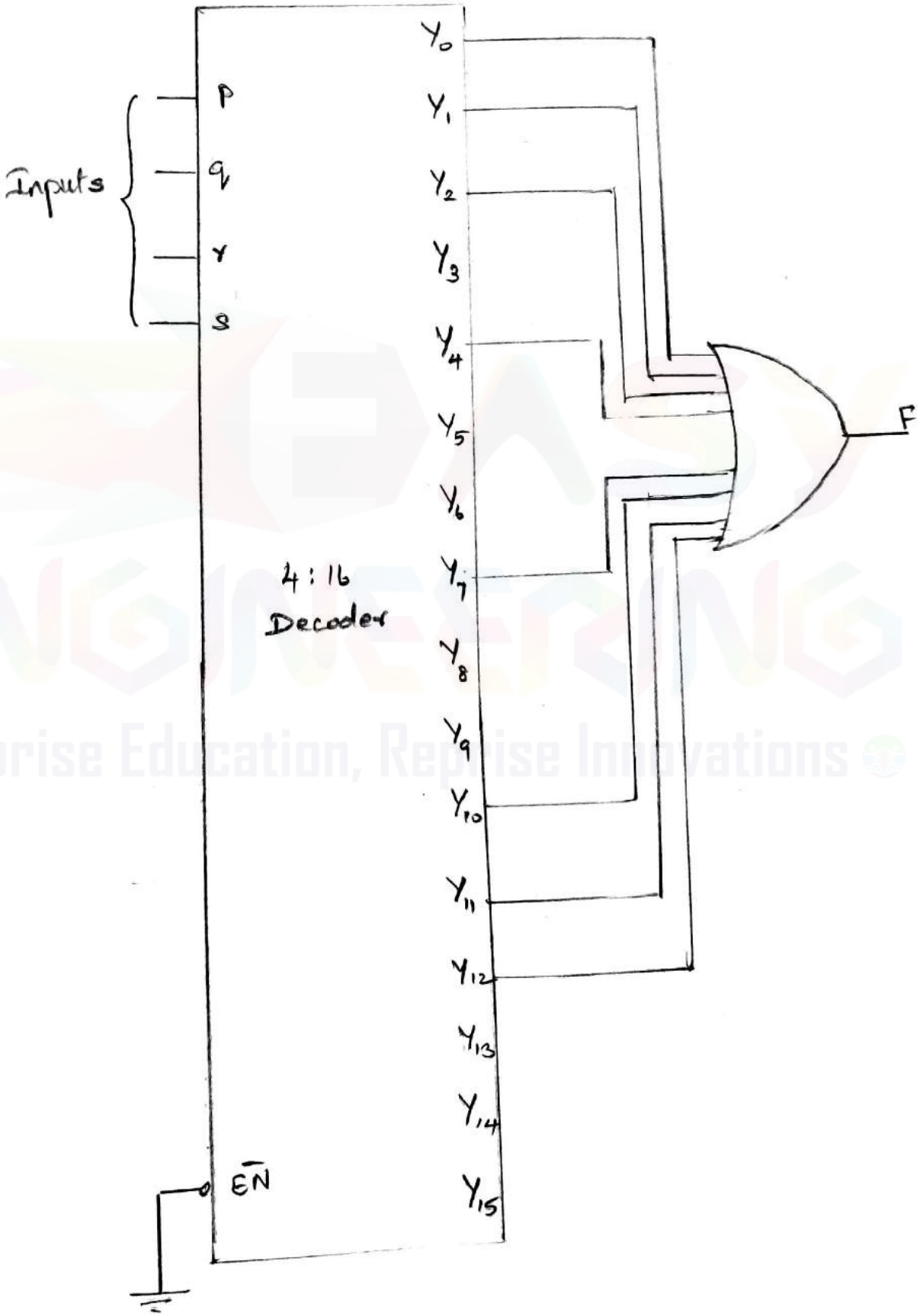


$$F = (\bar{A}+\bar{B}+C)(A+\bar{C})(\bar{B}+\bar{C}+\bar{D})(A+B+D)$$

Implement the function :

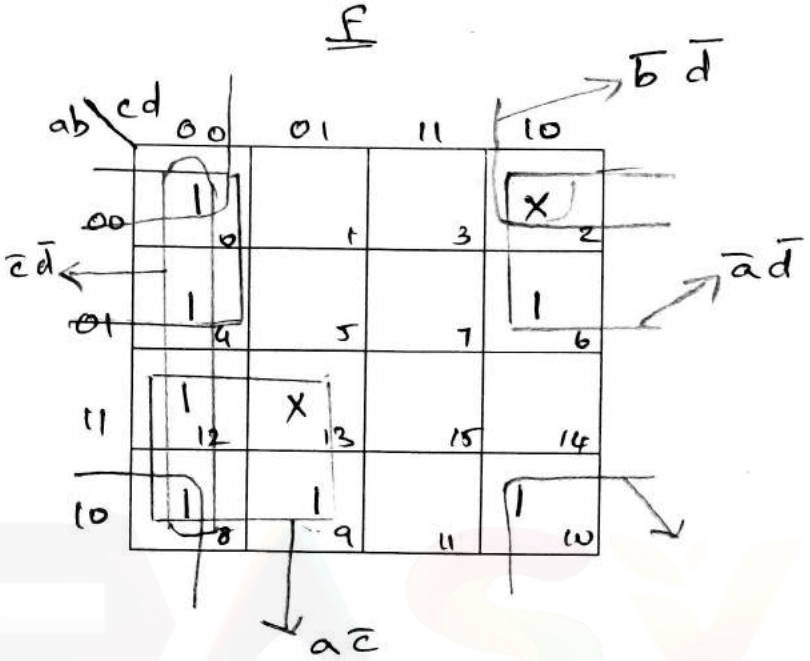
$F(P, q, r, s) = \Sigma(0, 1, 2, 4, 7, 10, 11, 12)$ using decoder.

(8) [M/D'14]



Minimize the function $F(a,b,c,d) = \Sigma(0,4,6,8,9,10,12)$ with $d = \Sigma(2,13)$. Implement the function using only NOR gates (8) [N/D'14]

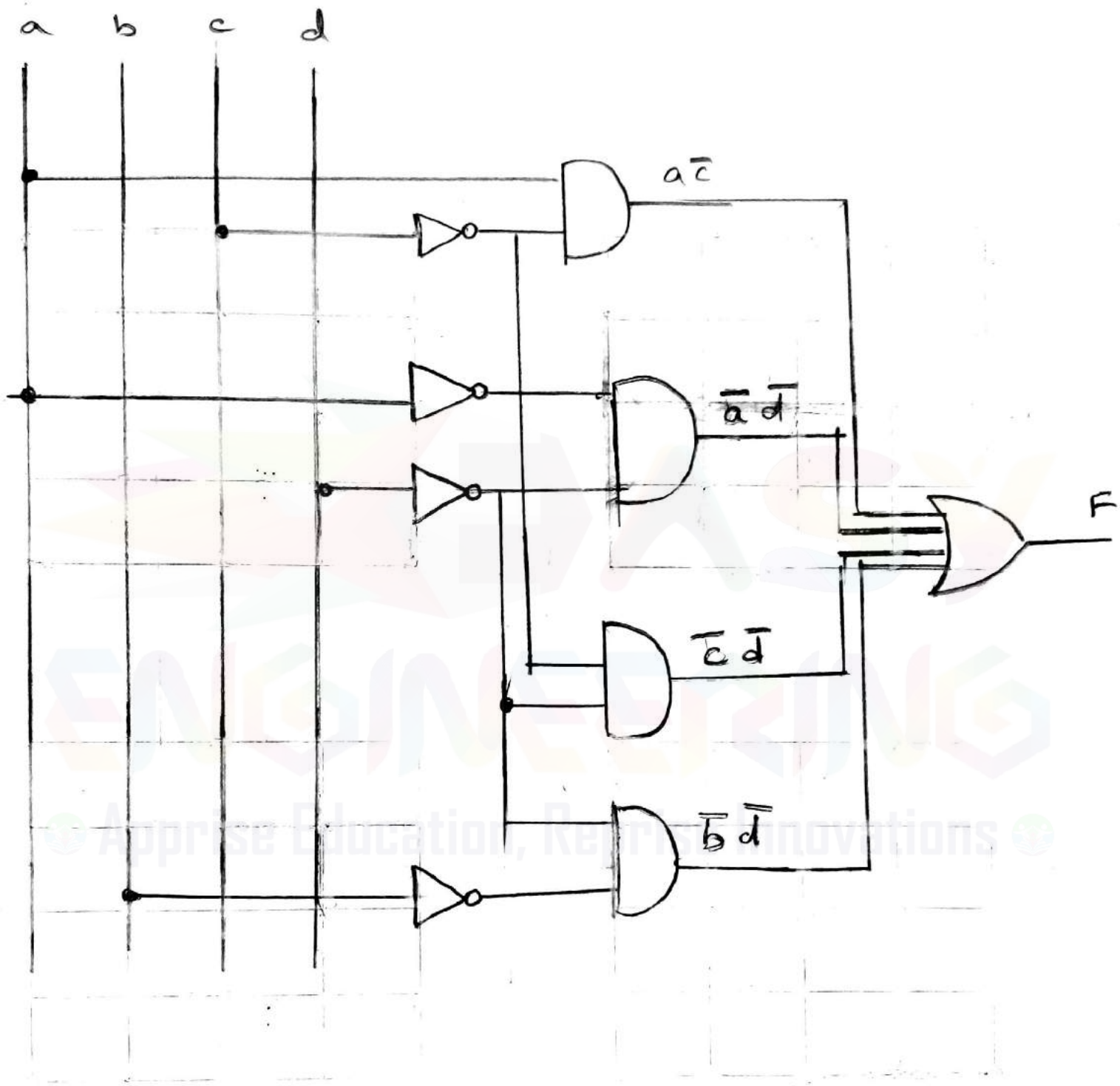
a	b	c	d	F
0	0	0	0	1
0	0	0	1	X
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	X
1	1	1	0	0
1	1	1	1	0



$$F(a,b,c,d) = \bar{c}\,\bar{d} + \bar{b}\,\bar{d} + a\bar{c} + \bar{a}\,\bar{d}$$

Logic Diagram:

$$f = \bar{c}\bar{d} + \bar{b}\bar{d} + a\bar{c} + \bar{a}\bar{d}$$



EE6301 – DIGITAL LOGIC CIRCUITS
UNIT – III
SYNCHRONOUS SEQUENTIAL CIRCUITS
Part – A

1. Convert T flip into D Flip flop [A/M-2015]

1. Truth Table for T Flip Flop

Input	Outputs	
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

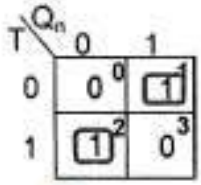
2. Excitation Table for D Flip Flop

Outputs		Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

3. Conversion Table

T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

4. K-map Simplification



$D = T\bar{Q}_n + \bar{T}Q_n$

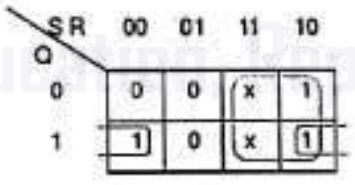
$= T \oplus Q_n$

2. State the rules for state assignment [A/M-2015]
- States which have the same next state, for a given input, should be given adjacent assignments
- States which are the next states of the same state should be given adjacent assignments
- To simplify the output function, states which have the same output for a given input should be given adjacent assignments.

3. Draw the truth table and state diagram of SR flip-flop [N/D-2015]

The SR flip-flop state table

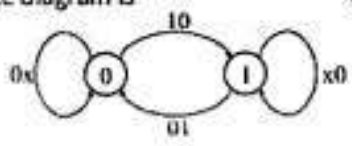
S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	x



$Q(t+1) = S + \bar{R}Q(t)$

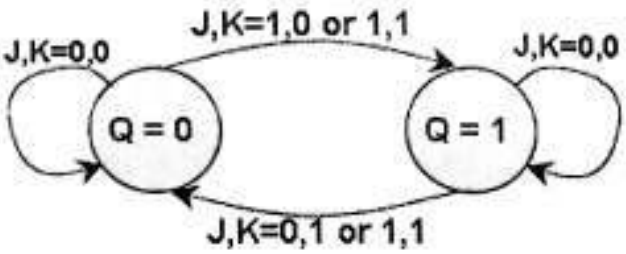
characteristic equation

The state diagram is



S	R	Q	State
0	0	Previous State	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Forbidden

4. Draw the state diagram of JK flip flop. [N/D-16]



5. What is edge triggered flip flops? [N/D-2015]

An edge-triggered flip-flop changes states either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse on the control input.

Inputs		Outputs		
D	C	Q	Q'	Comments
0	↑	0	1	RESET
1	↑	1	0	SET

If there is a HIGH on the D input when a clock pulse is applied, the flip-flop SETs and stores a 1.
If there is a LOW on the D input when a clock pulse is applied, the flip-flop RESETs and stores a 0.

6. State any two differences between Moore and Mealy state machines.[M/J-2016] [N/D-16]

	Mealy	Moore
(1)	O/Ps depend on the present state and present I/Ps	O/Ps depend only on the present state
(2)	The O/P changes asyn -chronously with the enabling clock edge	Since the O/Ps change when the state changes, and the state change is synchronous with the enabling clock edge, O/Ps change synchronously with this clock edge
(3)	A counter is not a Mealy machine	A counter is a Moore machine (o/ps = state bits)

7. Give the Characteristic equation and characteristic table of SR flip-flop.[M/J-2016]

Characteristic table of SR flip-flop:

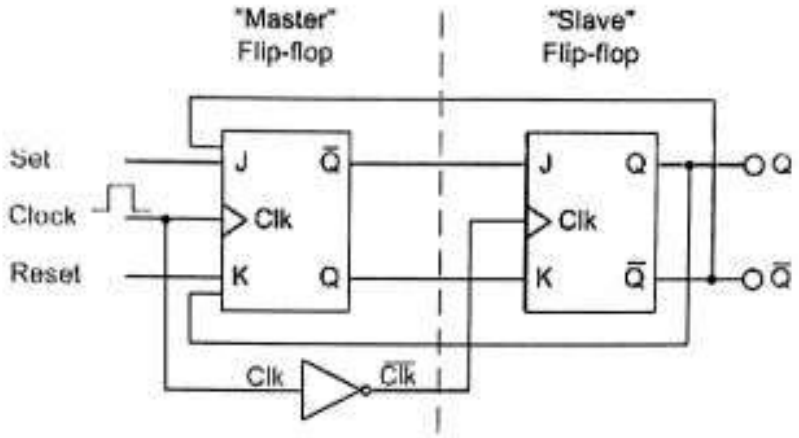
S	R	Q_n	Q_{n+1}	State
0	0	0	0	No Change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Invalid
1	1	1	X	

Characteristic equation of SR flip-flop

$$Q_{n+1} = S + R'Q_n$$

UNIT – III
Part – B

1. Explain the operation of a master slave JK flip flop (8) [A/M-2015], [M/J-2016]
   - Master-slave flip flop is designed using two separate flip flops.
   - Out of these, one acts as the master and the other as a slave. The figure of a master-slave J-K flip flop is shown below.



   - Both the J-K flip flops are connected in a series connection.
   - The output of the master J-K flip flop is fed to the input of the slave J-K flip flop.
   - The output of the slave J-K flip flop is given as a feedback to the input of the master J-K flip flop.
   - The clock pulse [Clk] is given to the master J-K flip flop and it is sent through a NOT Gate and thus inverted before passing it to the slave J-K flip flop.

Operation

- When Clk=1, the master J-K flip flop gets disabled. The Clk input of the master input will be the opposite of the slave input. So the master flip flop output will be recognized by the slave flip flop only when the Clk value becomes 0.
- When the clock pulse males a transition from 1 to 0, the locked outputs of the master flip flop are fed through to the inputs of the slave flip-flop making this flip flop edge or pulse-triggered.

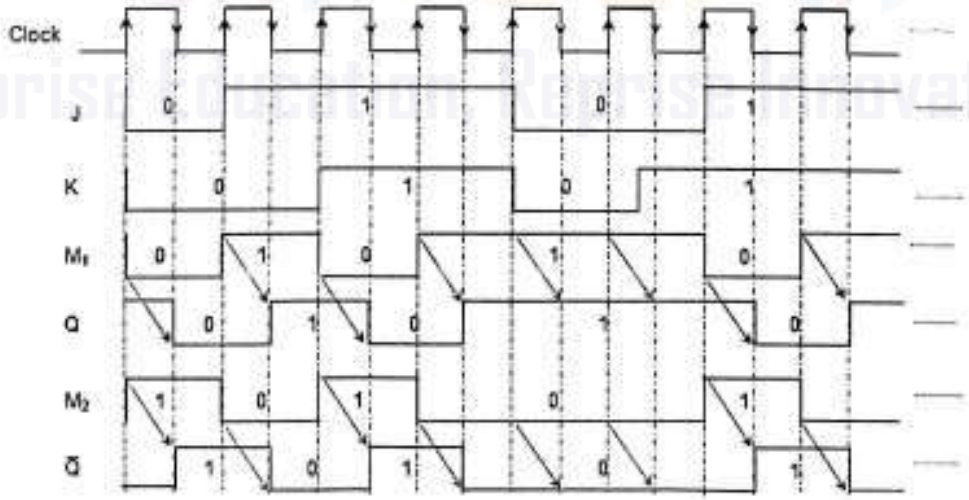
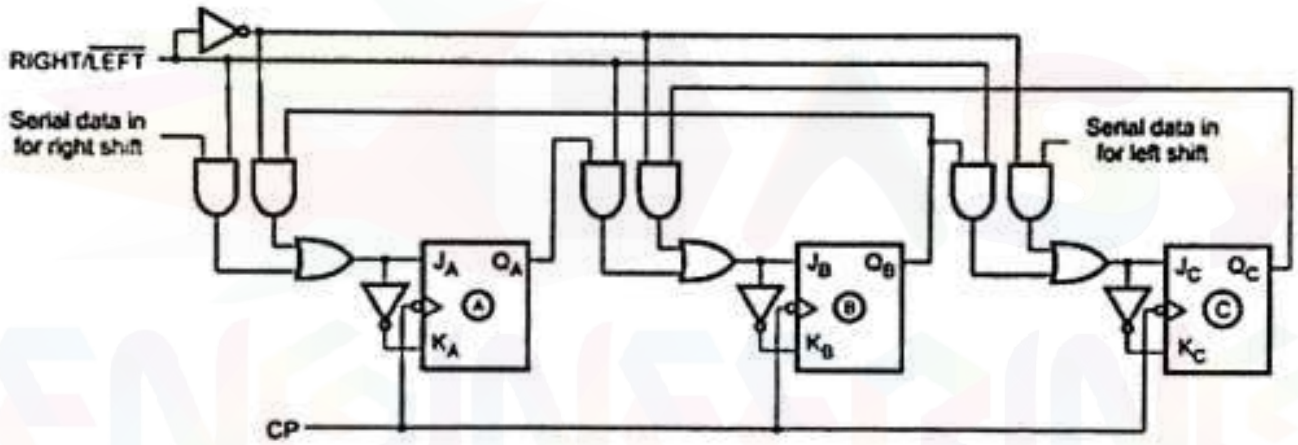


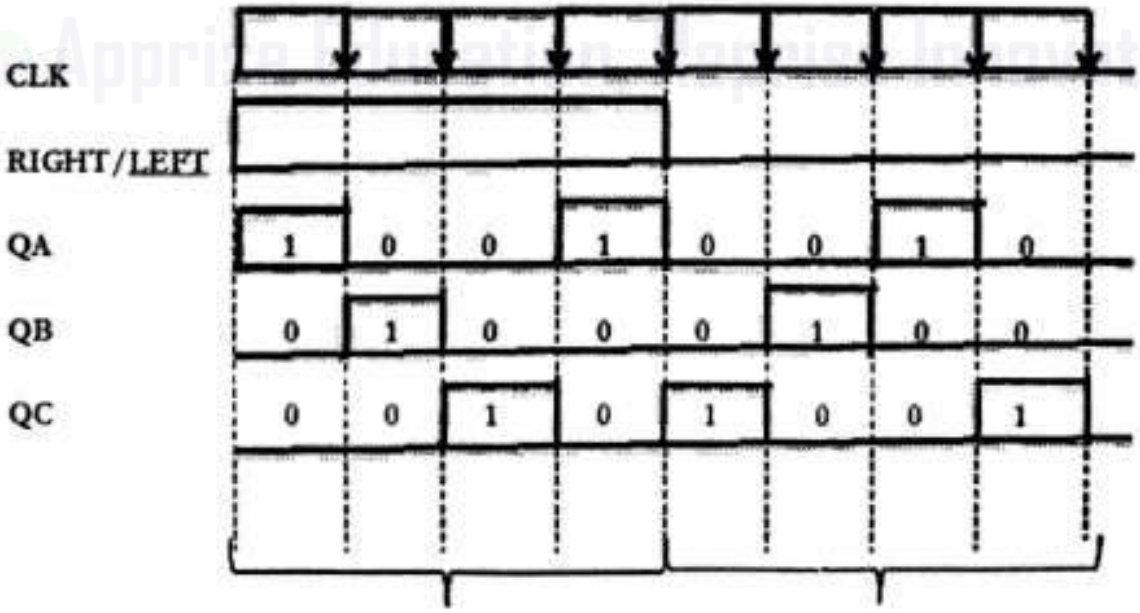
Figure 3 Timing diagram for master-slave JK flip-flop

- Thus, the circuit accepts the value in the input when the clock is HIGH, and passes the data to the output on the falling-edge of the clock signal.
- This makes the Master-Slave J-K flip flop a Synchronous device as it only passes data with the timing of the clock signal.

2. Design a 3-bit bidirectional shift register. (8) [A/M-2015]

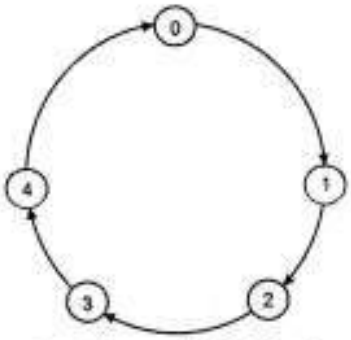
- The above diagram shows 3-bit Bi-directional Shift register, this type of register allows shifting of data either to the left side or to the right side.
- It is implemented by using logic gate circuitry that enables the transfer of data from one stage to the next stage to left or to the right, depending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





3. Design a MOD-5 synchronous counter using JK flip-flops (8) [A/M-2015]

State Diagram:

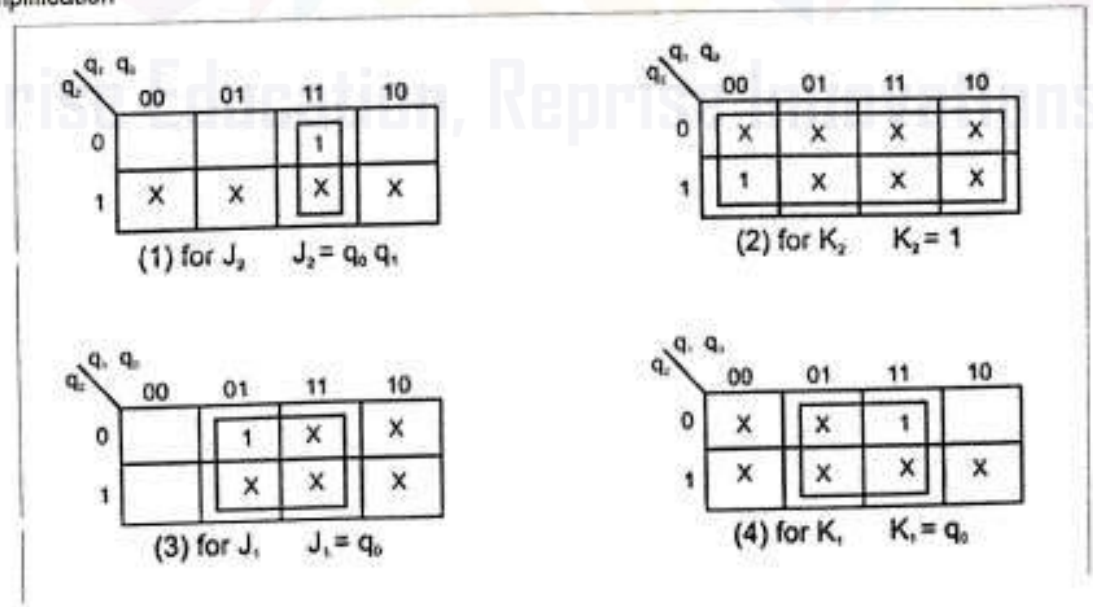


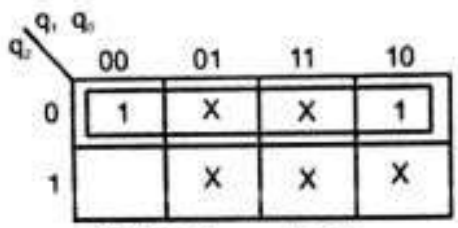
State Diagram of Mod - 5 Counter

State Table:

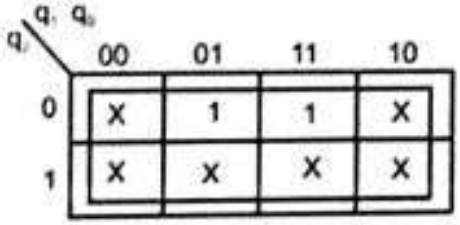
PS			NS			Excitation Inputs					
q_2	q_1	q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	x	1
1	0	0	0	0	0	x	1	0	x	0	x
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

K-map simplification



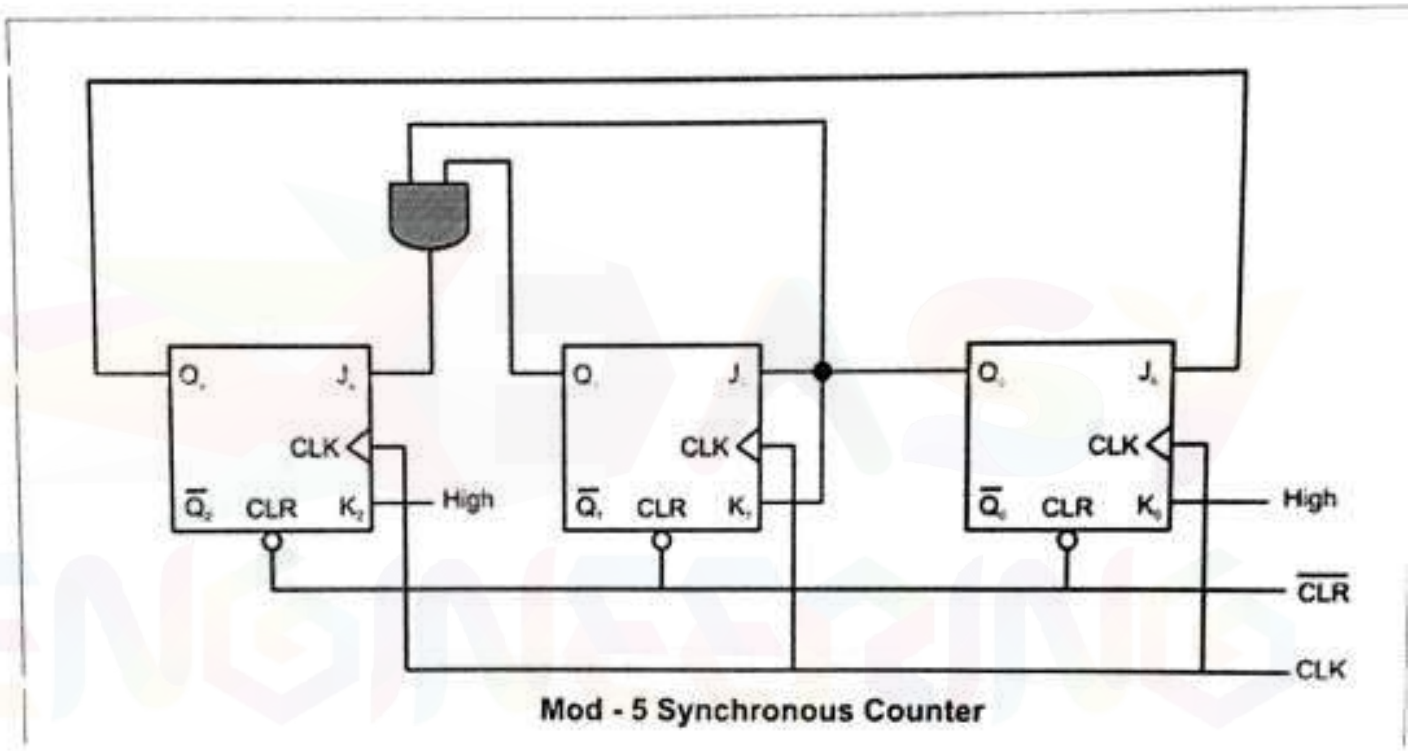


(5) for J_0 $J_0 = \overline{q}_0$



(6) for K_0 $K_0 = 1$

Logic Diagram:



Mod - 5 Synchronous Counter

4. Design a sequence detector to detect the sequence 101 using JK flip flop (8) [A/M-2015]

State Diagram

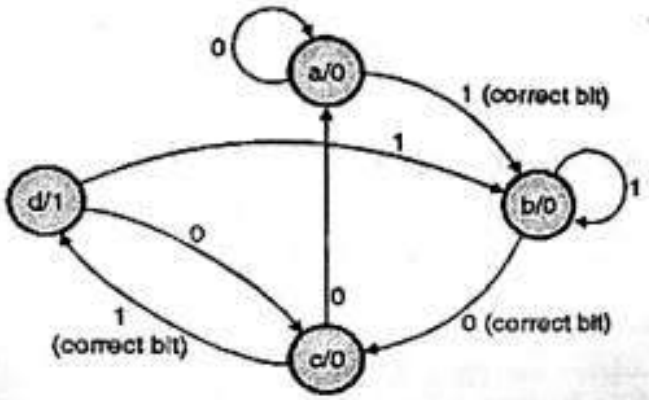


Fig3. State diagram for 101 detector

State table:

Present state	Next state		Output	
	x = 0	x = 1	x = 0	x = 1
a	a	b	0	0
b	c	b	0	0
c	a	d	0	1
d	c	b	0	0

Fig4. State table for above state diagram

Excitation table:

Input	Present state		Next state		F/F inputs		Output
x	B	A	B+1	A+1	D_B	D_A	Y
0	0	0 (a)	0	0 (a)	0	0	0
0	0	1 (b)	1	0 (c)	1	0	0
0	1	0 (c)	0	0 (a)	0	0	0
0	1	1 (d)	1	0 (c)	1	0	0
1	0	0 (a)	0	1 (b)	0	1	0
1	0	1 (b)	0	1 (b)	0	1	0
1	1	0 (c)	1	1 (d)	1	1	0
1	1	1 (d)	0	1 (b)	0	1	1

Fig5. Excitation table

K-map simplification:

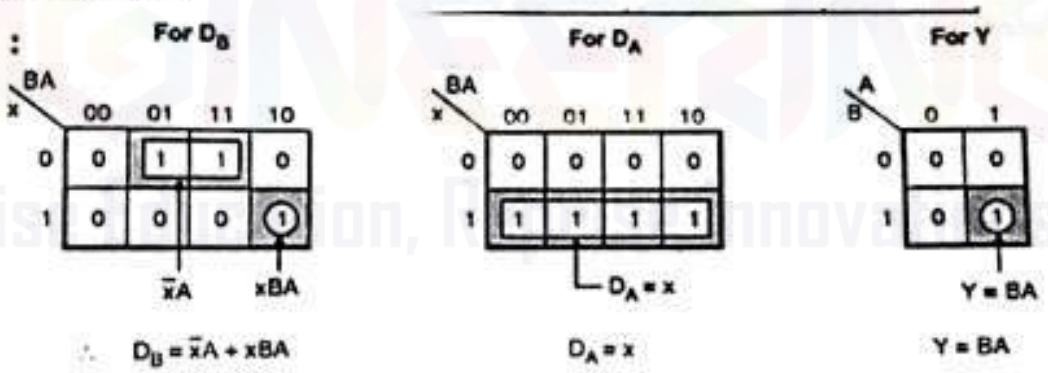


Fig6. K-maps

Logic Diagram:

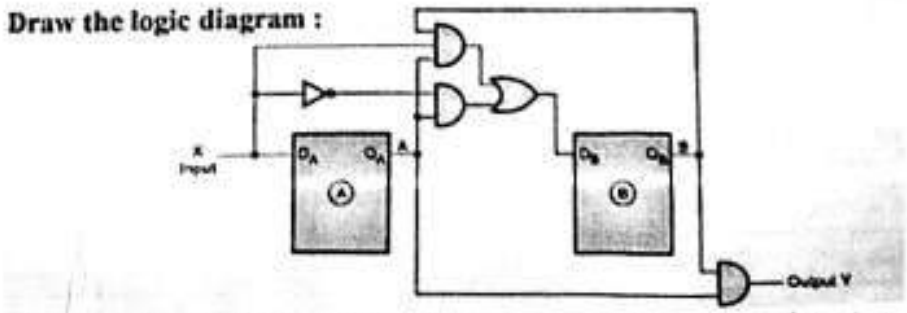


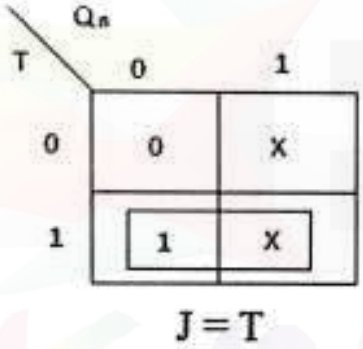
Fig7. Circuit diagram for implementation of 101 sequence detector

5. Realize T flip flop using JK flip flop. (4) [N/D – 2015] .

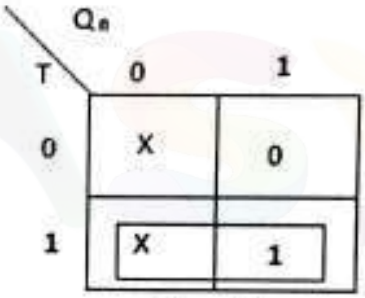
State Table

T	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-map simplification

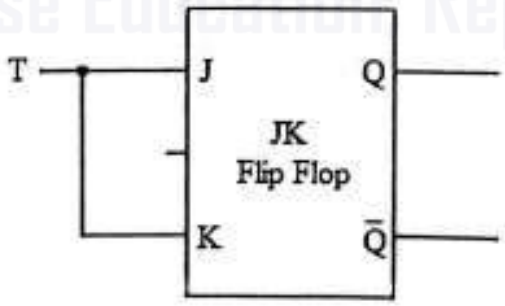


$J = T$



$K = T$

Logic Diagram



6. Write short notes on SIPO and draw the output waveform.(6) [N/D-16]

- In Serial In Parallel Out (SIPO) shift registers, the data is stored into the register serially while it is retrieved from it in parallel-fashion.
- Figure 1 shows an n-bit synchronous SIPO shift register sensitive to positive edge of the clock pulse.
- The data word which is to be stored (Data in) is fed serially at the input of the first flip-flop (D_1 of FF_1).
- It is also seen that the inputs of all other flip-flops (except the first flip-flop FF_1) are driven by the outputs of the preceding ones say for example, the input of FF_2 is driven by the output of FF_1.
- In this kind of shift register, the data stored within the register is obtained as a parallel-output data word (Data out) at the individual output pins of the flip-flops (Q_1 to Q_n).

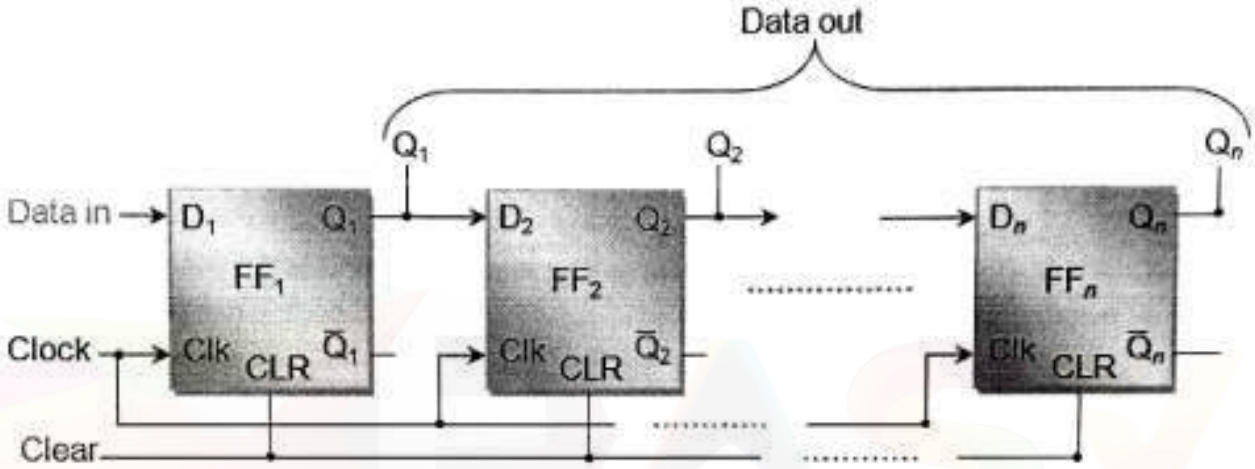


Figure 1 *n*-bit Serial-In Parallel-Out Right-Shift Shift Register

- In general, the register contents are cleared by applying high on the clear pins of all the flip-flops at the initial stage. After this, the first bit, B_1 of the input data word is fed at the D_1 pin of FF_1.
- This bit (B_1) will enter into FF_1, get stored and thereby appears at its output Q_1 on the appearance of first leading edge of the clock. Further at the second clock tick, the bit B_1right-shifts and gets stored into FF_2 while appearing at its output pin Q_2 while a new bit, B_2enters into FF_1.
- Similarly at each clock tick the data within the register moves towards right by a single bit while a new bit of the input word enters into the register. Meanwhile one can extract the bits stored within the register in parallel-fashion at the individual flip-flop outputs.
- Analyzing on the same grounds, one can note that the n-bit input data word is obtained as an n-bit output data word from the shift register at the rising edge of the n^{th} clock pulse. This working of the shift-register an be summarized as in Table I and the corresponding wave forms are given by Figure 2.

Table I Data Movement in Right-Shift SIPO Shift Register

Clock Cycle	Data in	Q_1	Q_2	...	Q_n
1	B_1	B_1	0	...	0
2	B_2	B_2	B_1	...	0
3	B_3	B_3	B_2	...	0
4	B_4	B_4	B_3	...	0
5	B_5	B_5	B_4	...	0
6	B_6	B_6	B_5		0
.	.	.	.	...	.
.	.	.	.		.
.	.	.	.		.
n	B_n	B_n	B_{n-1}	...	B_1

Output of SIPO (right-shift) Shift Register

Output Waveform

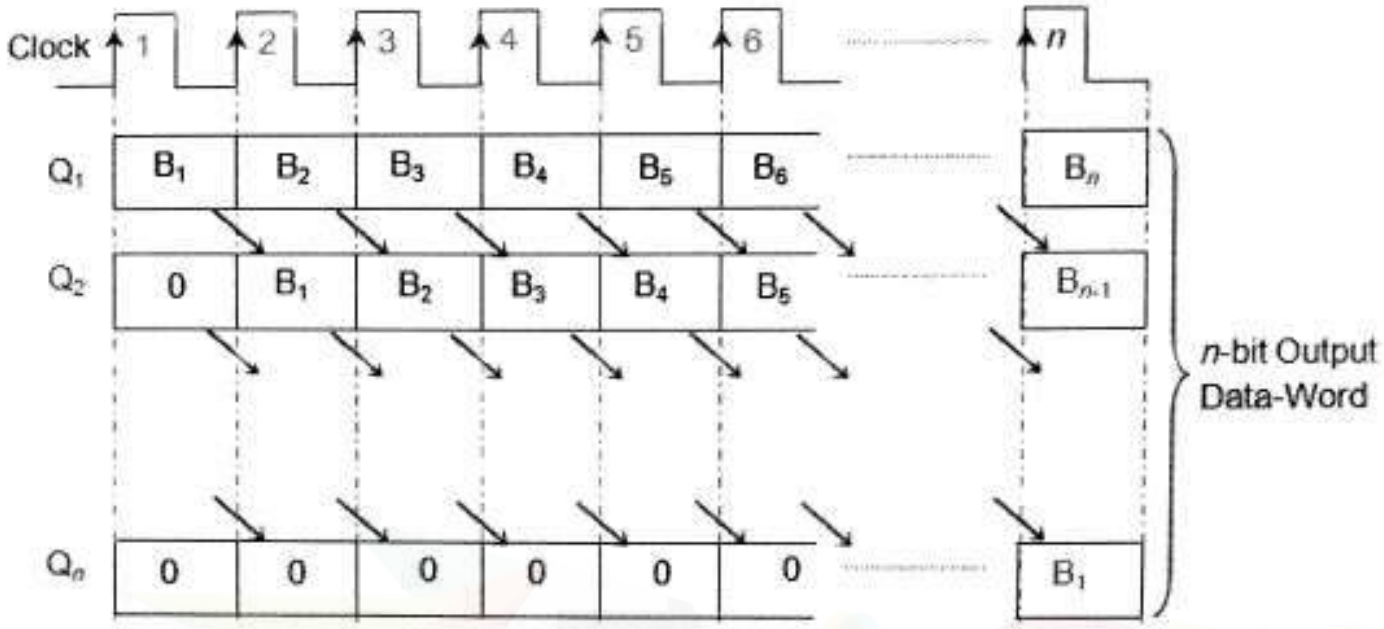
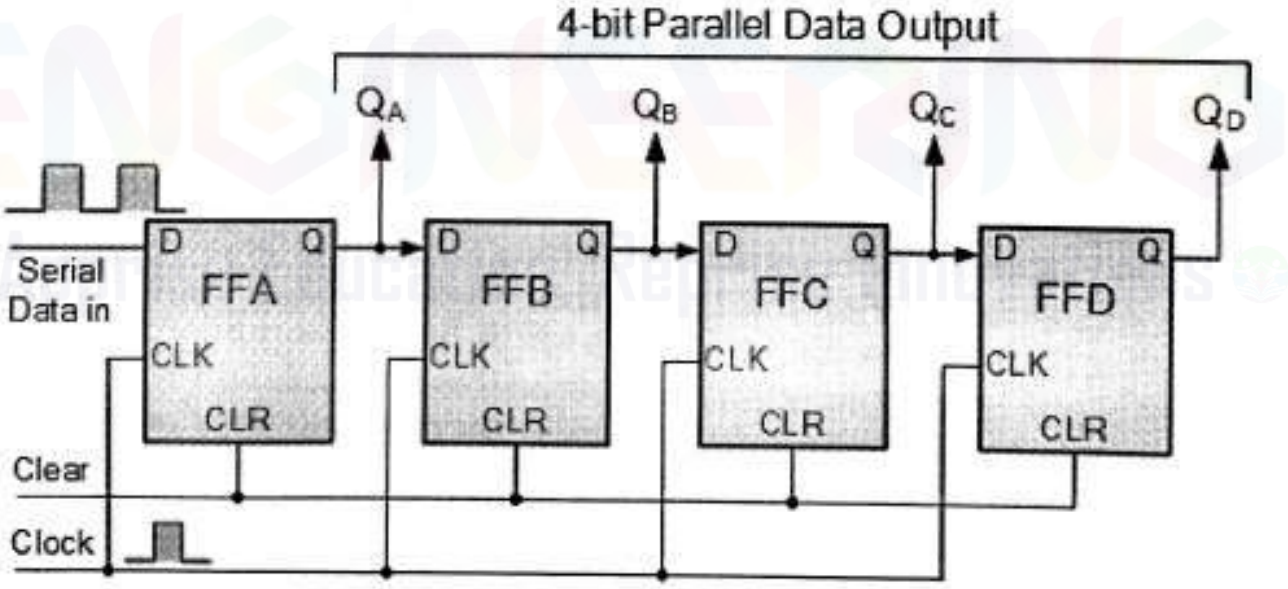


Figure 2 Output Waveform of *n*-bit Right-Shift SIPO Shift Register

Logic Diagram



7. Explain the realization of JK flip flop from T flip flop. (7) [N/D-16]

1. Truth Table for JK Flip Flop

Inputs		Outputs	
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

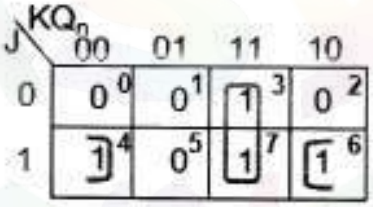
2. Excitation Table for T Flip Flop

Outputs		Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

3. Conversion Table

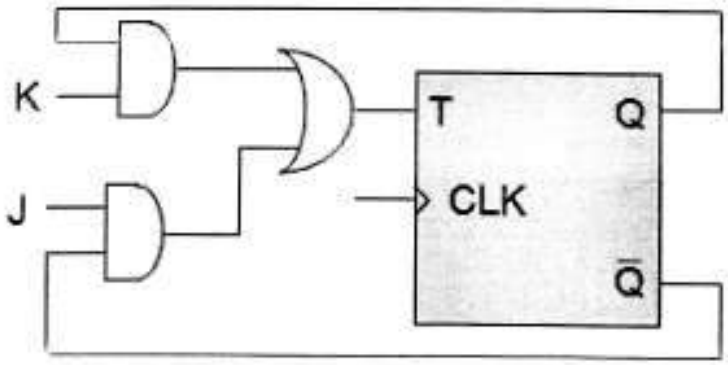
J	K	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

4. K-map Simplification



$$T = J\bar{Q}_n + KQ_n$$

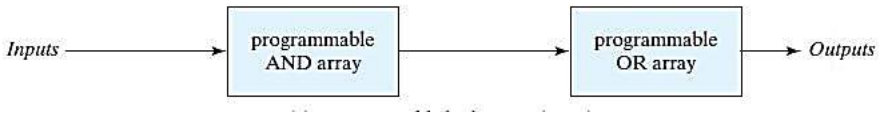
5. Circuit Design



EE8351 – DIGITAL LOGIC CIRCUITS
UNIT – IV
ASYNCHRONOUS SEQUENTIAL CIRCUITSAND PROGRAMMABLE LOGIC DEVICES
PART – A

1. What is deadlock condition? [N/D'14]
   A condition resulting when one task is waiting to access a resource that another is holding, and vice versa.

2. Draw the block diagram of PLA. [N/D'14]



3. State the difference between static – 0 and static – 1 hazard.[A/M'15]
   Static-1 Hazard: the output is currently 1 and after the inputs change, the output momentarily changes
   to 0 before settling on 1
   Static-0 Hazard: the output is currently 0 and after the inputs change, the output momentarily changes to 1 before settling on 0

4. What is a PROM? [A/M'15] , [N/D'15]
   PROM- Programmable Read Only Memory is a device that contains Fixed AND and Programmable
   OR functions. IT contains fuses inact giving all 1`s in the stored bits and blown fuses by applying high voltage defining 0 states.

5. Compare pulsed mode and fundamental mode asynchronous circuit.[N/D'15]

S.No.	Pulsed mode asynchronous circuit	Fundamental mode asynchronous circuit
1.	Inputs are levels.	Inputs are pulses.
2.	Memory elements are either clocked flip – flops or time delay elements.	Memory elements are clocked flip – flops
3.	More difficult to dsign.	Easy to design.

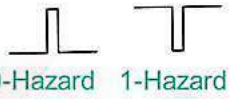
6. What are the two types of asynchronous sequential circuits? [M/J'16]
   The two types of asynchronous sequential circuits are:
   a) Fundamental mode circuits
   b) Pulse mode circuits

7. State the difference between PROM, PLA and PAL. [M/J'16], [A/M'17]

Types of PLD s	AND array	OR array
PROM	Fixed	Programmable
PLA	Programmable	Programmable
PAL	Programmable	Fixed

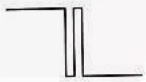
8. What is static hazard and dynamic hazard?[N/D'16]

- Static Hazard – Output value the same after input change



0-Hazard 1-Hazard

- Dynamic Hazard – Output value different after input change



9. Define races in asynchronous sequential circuits.[N/D'16]
When 2 or more binary state variables change their value in response to a change in an input variable, race condition occurs in an asynchronous sequential circuit. In case of unequal delays, a race condition may cause the state variables to change in an unpredictable manner.

10. What is a flow table? Give example. [A/M'17]
A state transition table with its internal state being symbolised with letters.
Examples:

y \ x	0	1
a	(a)	b
b	c	(b)
c	(c)	d
d	a	(d)

(a) Four states with one input

x_1x_2	00	01	11	10
a	(a), 0	(a), 0	(a), 0	b , 0
b	a , 0	a , 0	(b), 1	(b), 0

(b) Two states with two inputs and one output

PART – B

1. Explain the various types of hazards in sequential circuit design and the methods to eliminate them. Give suitable examples. (16) [N/D'14]

Hazards in any system are obviously an un-desirable effect caused by either a deficiency in the system or external. Influences. In digital logic hazards are usually referred to in one of three ways:

a) Static Hazards
b) Dynamic Hazards
c) Function Hazards

These logic hazards are all subsets of the same problem: - When changes in the input variables do not change the output 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.

This is however a temporary problem, and the logic will finally come to the desired function. Despite the logic arriving at the correct output, it is imeritive that hazards be eliminated as they can have an effect on other systems. Imagine hazards like this in a piece of hospital equipment.

Static Hazards

Definition:- *"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. We can see the effects of the delay in the circuit from the following flash animation.

The hazard can be dealt with in two ways:

1. Insert another (additional) delay to the circuit. This then eliminates the static hazard.
2. 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 '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 Static Hazards

The Static '1' Hazard.

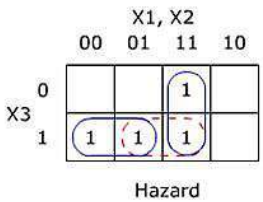
Let us consider an imperfect circuit that suffers from a delay in the physical logic elements i.e. AND gates etc.

The simple circuit performs the function:

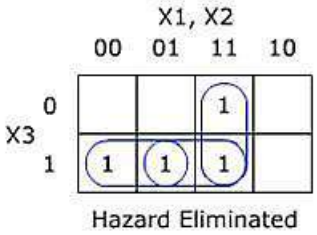
f = X1.X2 + X1'.X3

and the logic diagram can be shown as follows:

At the starting diagram, it is clear that if no delays were to occur, then the circuit would function normally. However since this is not a perfect circuit, and an error occurs when the input changes from 111 to 011. i.e. When X1 changes state.



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 elimate the hazard. So the resulting logic is of the form shown in the next figure.

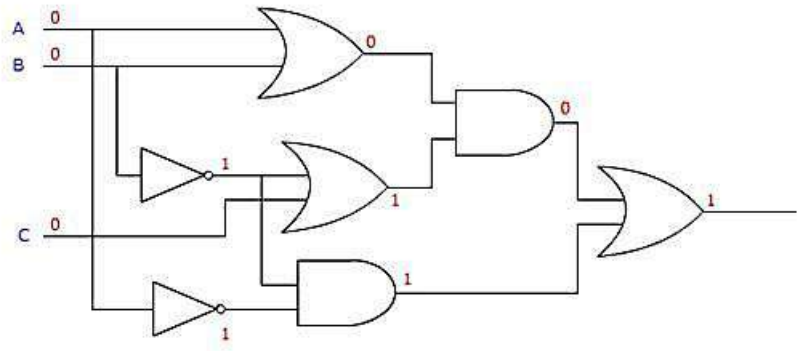


So our original function is now: f =X1.X2 + X1'.X3 + X2.X3
It is observed that even with imperfect logic elements, this example will not show signs of hazards when X1 changes state. This theory can be applied to any logic system.

Dynamic Hazards

- Definition:- *"A dynamic hazard is the possibility of an output changing more than once as a result of a single input change"*
- Dynamic hazards often occur in larger logic circuits where there are different routes to the output (from the input). If each route has a different delay, then it quickly becomes clear that there is the potential for changing output values that differ from the required / expected output. e.g. A logic circuit is meant to change output state from '1' to '0', but instead changes from '1' to '0' then '1' and finally rests at the correct value '0'. This is a dynamic hazard.
- Dynamic hazards take a more complex method to resolve. The below example shows how a dynamic hazard can occur but now how to solve it.

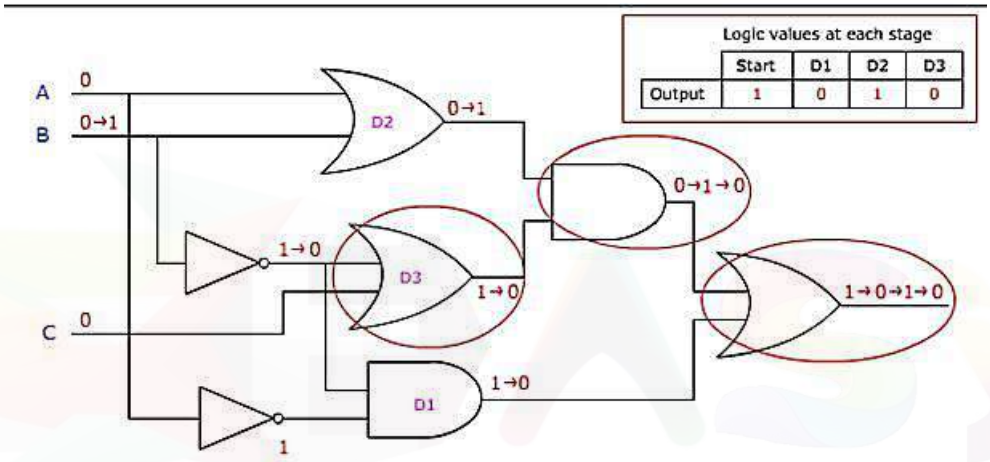
Let us take the circuit above, and see the proper logic output with the logic values above



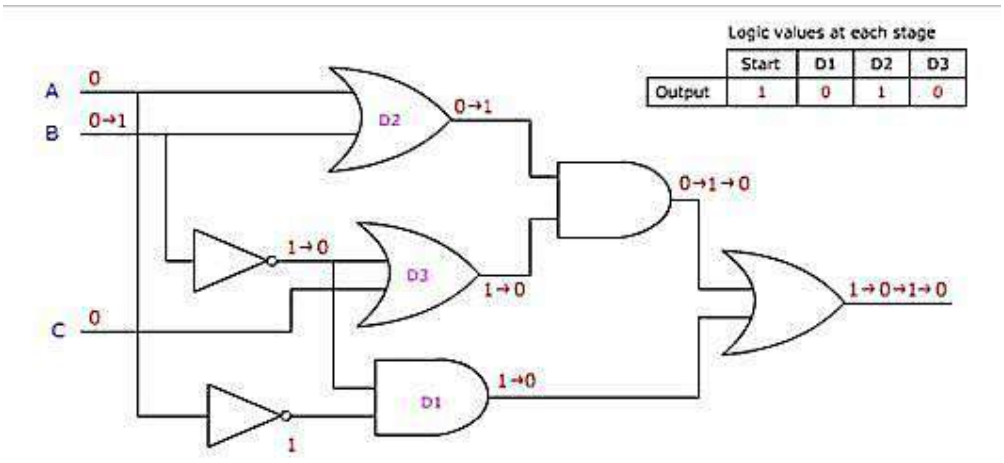
Let artificial delays be introduced in some of the elements. Marked D1, D2 and D3.

- Consider D1 < D2 < D3.
  i.e. The delay in D1 is less than the delay in D2, and the delay in D3 is greater than the delay in D2.

- Say that input B changes from 0 to 1. Working with one delay at a time the output can be determined.
- There could be a different output value for every different delay.
- The NOT gate shown changes from 1 to 0. Remembering that D1 is the shortest of the three delays, the AND gate shown will also change from 1 to 0.
- Because the other delays (D2, D3) are longer than D1, and because the other gates have no delay (or neglible delay), our output changes from 1 to 0. (The first effect of the hazard).
- The next delay to occur is D2. So the OR gate shown implements the change (0 OR 1 = 1). Now the AND gate has only seen the change in one of its inputs due to the delay D3 being longer than the other delays. So momentarily, both inputs are logic 1 which means the AND gates output goes to logic 1 hence changing the output of the entire circuit (1 OR 0 = 1).



- The next delay to occur is D3. The OR gate reacts slower than D1 to the change in input, but now the OR gate implements the change (0 OR 0 = 0). There is a knock on effect to the AND gate (as 0 AND 1 = 0) and then again to the output of the entire circuit. The output rests at 0.
- This is the final stage as there are no more delays in the circuit and therefore no more changes of state due to input B changing.
  D3(1 OR 0 = 1).
- The circuit finally rests at the correct logical value for an ideal circuit. However the output has changed values twice before coming to the correct result on the third change. If this was an ideal circuit then the output should have changed only once.



2. Describe with reasons the effect of races in asynchronous sequential circuit design. Explain its type with illustrations. Show the method of race – free state assignments with examples. (16) [N/D'14]
   - A race condition or race hazard is the behaviour of an electronic, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events.
   - A race condition may occur in a system of logic gates where inputs vary. If a given output depends on the state of the inputs it may only be defined for steady-state signals. As the inputs change state a small delay will occur before the output changes due to the physical nature of the electronic system. The output may, for a brief period, change to an unwanted state before settling back to the designed state. Certain systems can tolerate such glitches but if this output functions as a clock signal for further systems that contain memory, for example, the system can rapidly depart from its designed behaviour..

Types:

❖ **Critical and non-critical forms**

a) A *critical race condition* occurs when the order in which internal variables are changed determines the eventual state that the state machine will end up in.

b) A *non-critical race condition* occurs when the order in which internal variables are changed does not determine the eventual state that the state machine will end up in.

❖ **Static, dynamic, and essential forms**

a) A *static race condition* occurs when a signal and its complement are combined together.

b) A *dynamic race condition* occurs when it results in multiple transitions when only one is intended. They are due to interaction between gates. It can be eliminated by using no more than two levels of gating.

c) An *essential race condition* occurs when an input has two transitions in less than the total feedback propagation time. Sometimes they are cured using inductive delay line elements to effectively increase the time duration of an input signal.

Design techniques such as Karnaugh maps encourage designers to recognize and eliminate race conditions before they cause problems. Often logic redundancy can be added to eliminate some kinds of races.

RACE -FREE STATE ASSIGNMENT

- Once a reduced flow table has been derived for an asynchronous sequential circuit, the next step in the design is to assign binary variables to each stable state. This assignment results in the transformation of the flow table into its equivalent transition table.
- The primary objective in choosing a proper binary state assignment is the prevention of critical races. Critical races can be avoided by making a binary state assignment in such a way that only one variable changes at any given time when a state transition occurs in the flow table.

Three-Row Flow-Table Example

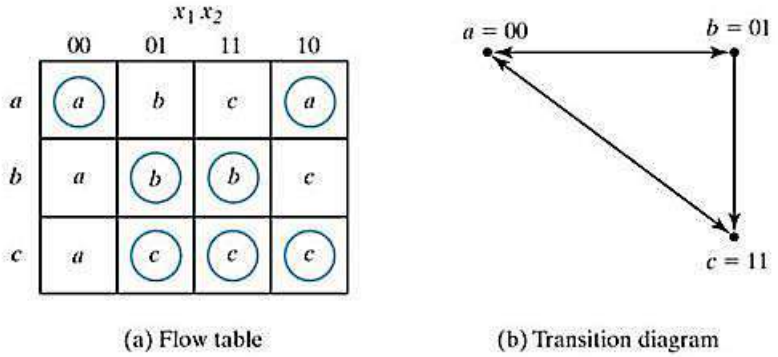


Fig: Three row flow table example

- To avoid critical races, we must find a binary state assignment such that only one binary variable changes during each state transition.
- An attempt to find such an assignment is shown in the transition diagram.
- State a is assigned binary 00, and state c is assigned binary 11.
- This assignment will ca use a critical race during the transition from a to c because there are two changes in the binary state variables and the transition from a to c may occur directly or pass through b.
- Note that the transition from c to a also ca uses a race condition, but it is noncritical because the transition does not pass through other states.
- A race-free assignment can be obtained if we add an extra row to the flow table. The use of a fourth row does not increase the number of binary state variables, but it allows the formation of cycles between two stable states.
- The transition table corresponding to the flow table with the indicated binary state assignment is shown in Fig. The two dashes in row d represent unspecified states that can be considered don't-care conditions. However, care must be taken not to assign 10 to these squares, in order to avoid the possibility of an unwanted stable state being established in the fourth row.

Four-Row Flow-Table Example

A flow table with four rows requires a minimum of two state variables. Although a race-free assignment is sometimes possible with only two binary state variables, in many cases the requirement of extra rows to avoid critical races will dictate the use of three binary state variables

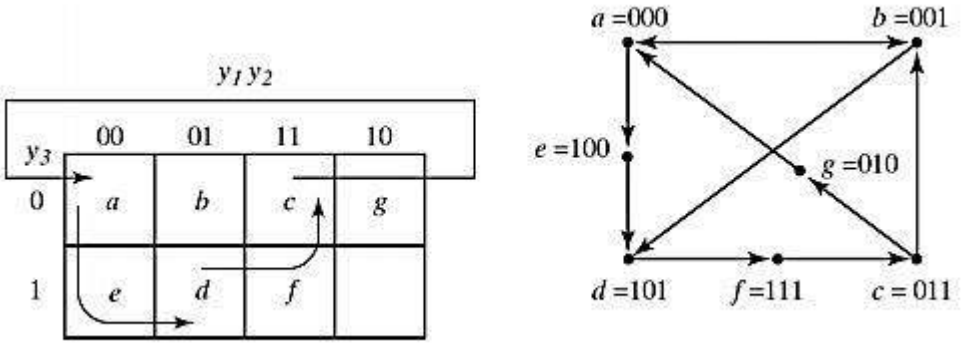


Fig: Four-row flow-table example

- The following figure shows a state assignment map that is suitable for any four-row flow table. States a, b, c and d are the original states and e, f and g are extra states.
- The transition from a to d must be directed through the extra state e to produce a cycle so that only one binary variable changes at a time.
- Similarly, the transition from c to a is directed through g and the transition from d to c goes through f. By using the assignment given by the map, the four-row table can be expanded to a seven-row table that is free of critical races.

	00	01	11	10
000 = *a*	*b*	(*a*)	*e*	(*a*)
001 = *b*	(*b*)	*d*	(*b*)	*a*
011 = *c*	(*c*)	*g*	*b*	(*c*)
010 = *g*	–	*a*	–	–
110 –	–	–	–	–
111 = *f*	*c*	–	–	*c*
101 = *d*	*f*	(*d*)	(*d*)	*f*
100 = *e*	–	–	*d*	–

Fig: State assignment to modified flow table

Note that although the flow table has seven rows there are only four stable states. The uncircled states in the three extra rows are there merely to provide a race-free transition between the stable states.

Multiple-Row Method

- The method for making race-free stale assignments by adding extra rows in the flow table is referred to as the shared-row method.
- A second method called the multiple-row method is not as efficient, but is easier to apply.
- In multiple- row assignment each state in the original row table is replaced by two or more combinations of state variables.

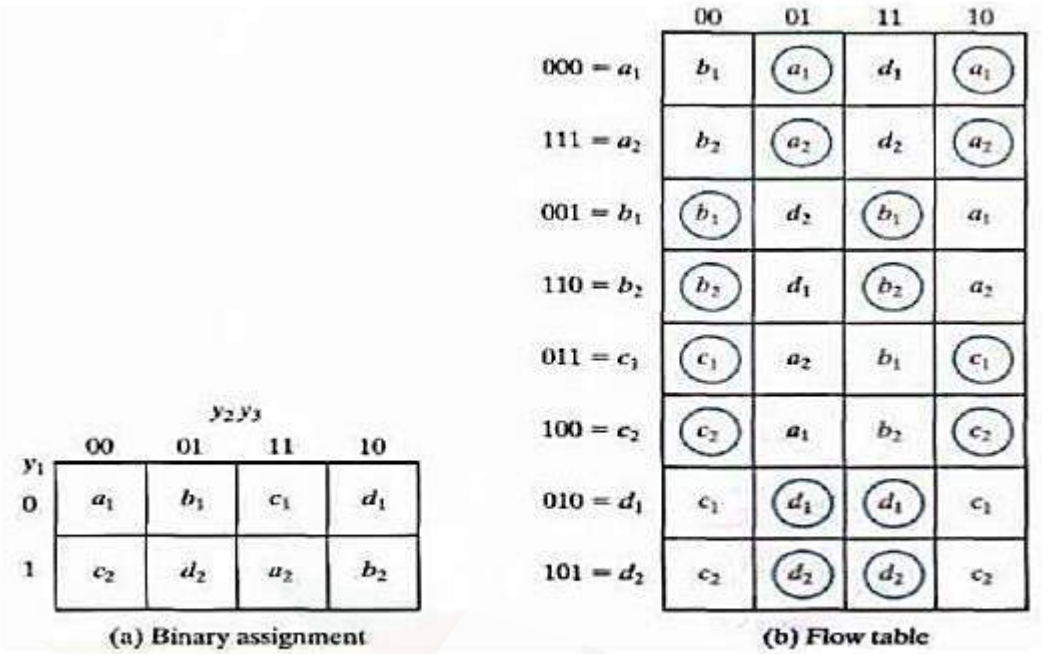


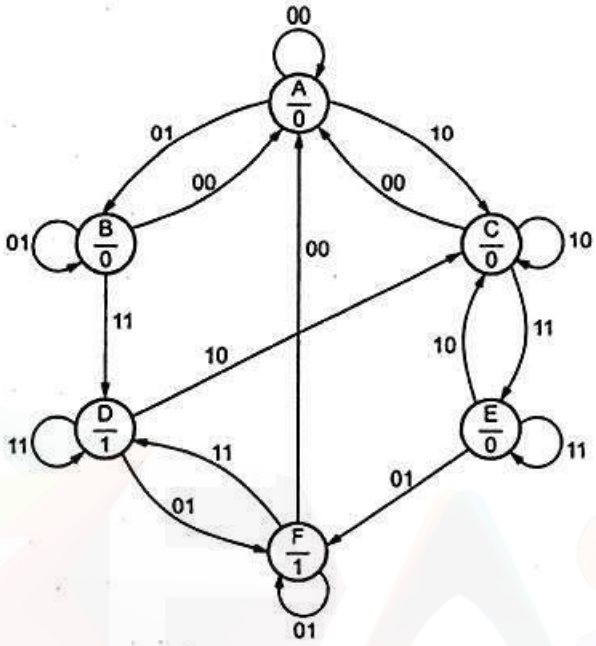
Fig: Multiple row assignment

Fig: Multiple row assignment

- There are two binary state variables for each stable state, each variable being the logical complement of the other. For example, the original state a is replaced with two equivalent states a1 =000 and a2 = 111.
- The output values, not shown here must be the same in a1 and a2. Note that a1 is adjacent to b1, c2 and d1, and a2 is adjacent to c1, b2 and d2, and similarly each state is adjacent to three states with different letter designations.
- The expanded table is formed by replacing each row of the original table with two rows. In the multiple-row assignment, the change from one stable state 10 another will always cause a change of only one binary state variable.
- Each stable stale has two binary assignments with exactly the same output.

3. 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.(16) [A/M'15]

State diagram



Primitive flow table constructed from state diagram

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, –	–, –

Flow table with state assignment

Present state F_2 F_1	Next state, Output Z for X_2X_1 Inputs			
	00	01	11	10
S_0 → 0 0	(S_0), 0	(S_0), 0	S_2, –	S_1, –
S_1 → 0 1	S_0, –	S_3, –	(S_1), 0	(S_1), 0
S_2 → 1 0	S_0, –	(S_2), 1	(S_2), 1	S_3, –
S_3 → 1 1	–, –	S_2, –	–, –	S_1, –

Flow table converted into a transition table

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

K-map simplification

For F_2^+

F_2F_1 \ X_2X_1	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	X	1	X	0
10	0	1	1	1

$F_2^+ = \bar{F}_2X_2X_1 + F_2\bar{X}_2X_1 + F_2X_1 + F_2\bar{F}_1X_2$

For F_1^+

F_2F_1 \ X_2X_1	00	01	11	10
00	0	0	0	1
01	0	1	1	1
11	X	0	X	1
10	0	0	0	1

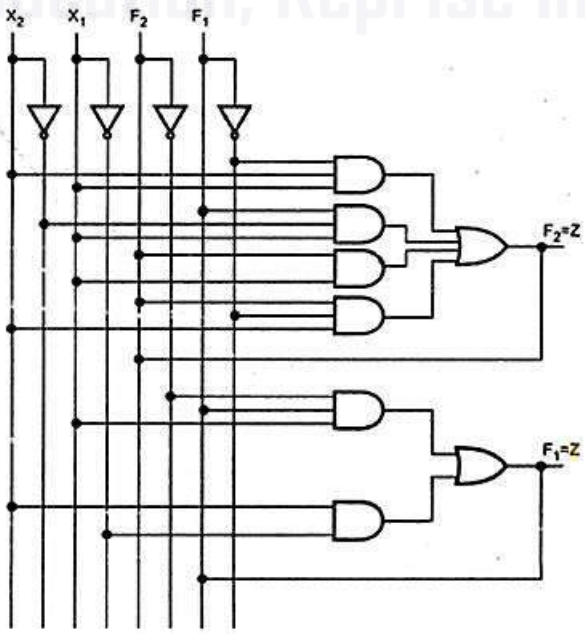
$F_1^+ = \bar{F}_2F_1X_1 + X_2\bar{X}_1$

For Z

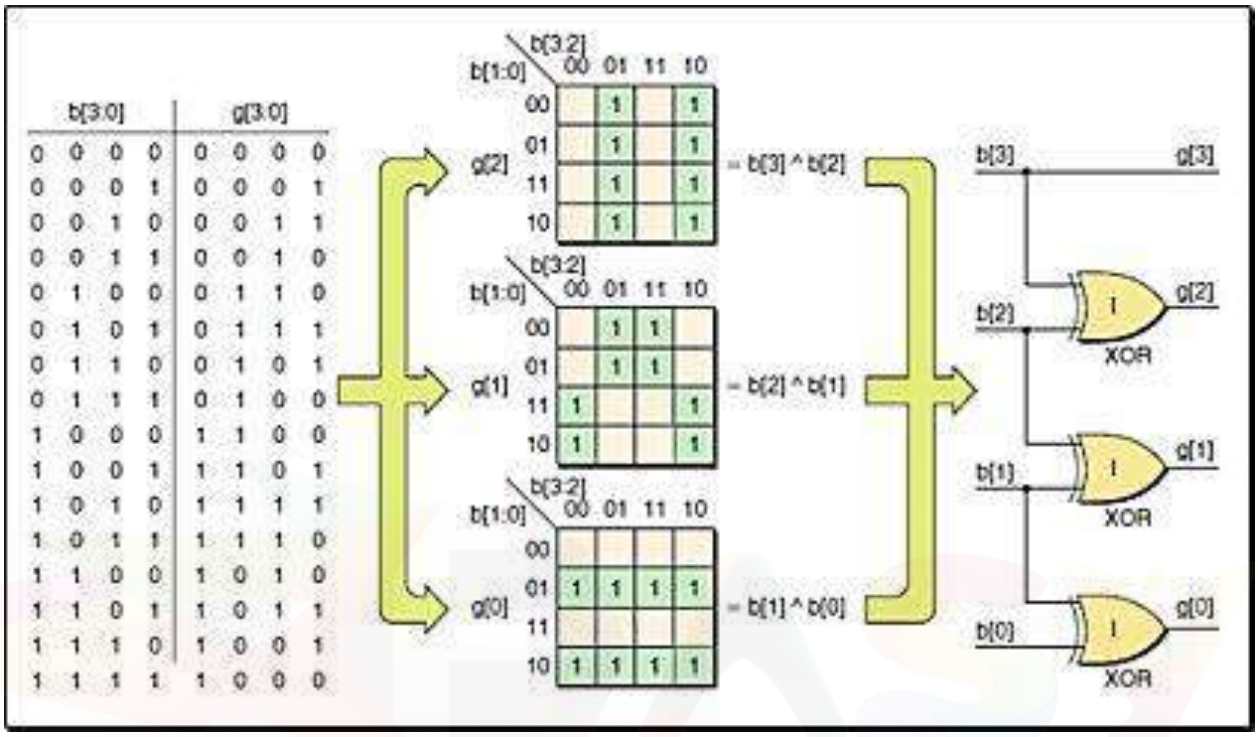
F_2F_1 \ X_2X_1	00	01	11	10
00	0	0	X	X
01	X	X	0	0
11	X	X	X	X
10	X	1	1	X

$Z = F_2$

Logic diagram



4. Show how to program the fusible links to get a 4 bit Gray code from the binary inputs using PLA and PAL and compare the design requirements with PROM. (16) [N/D'15]
   Gray code generator using PROM



PROM based binary to gray code converter

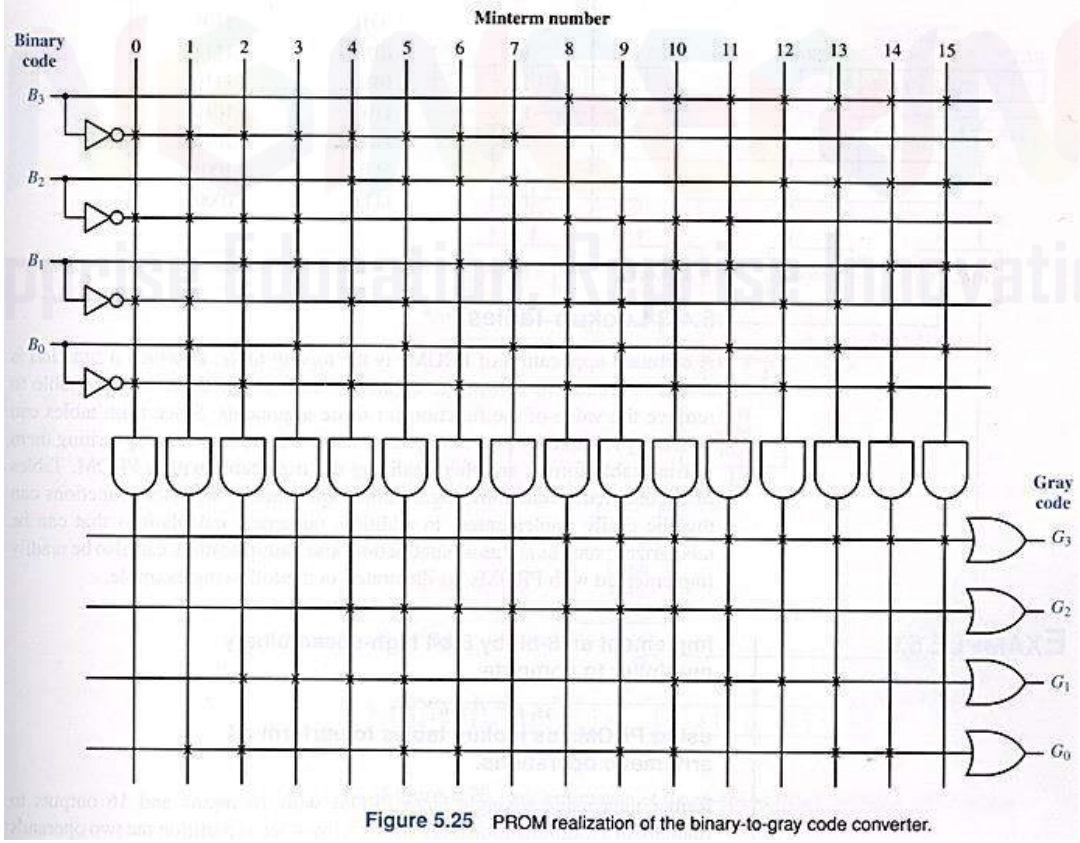
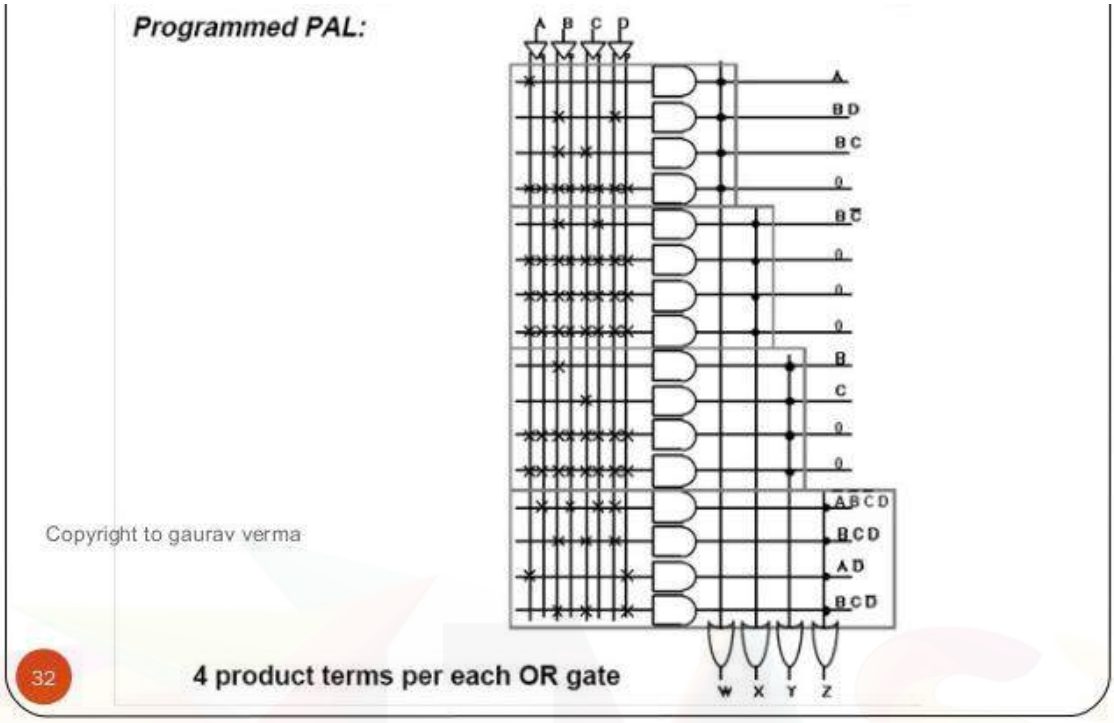


Figure 5.25 PROM realization of the binary-to-gray code converter.

PAL based binary to gray code converter



Binary to gray code converter Boolean expression are concluded as

$$Y_3 = A$$

$Y_2 = A\bar{B} + \bar{A}B$

$Y_1 = B\bar{C} + \bar{B}C$

$Y_0 = C\bar{D} + \bar{C}D$

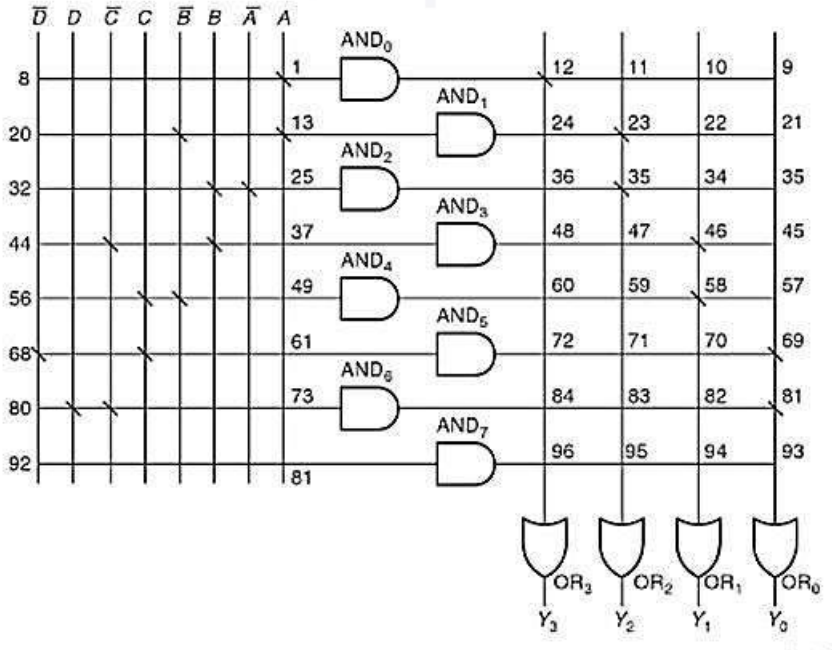
It is noted that 4 x 7 x 4 PLA is needed to implement gray code converter and same for PAL also.

Total number of AND OR gates links are

96 – PLA

56 – PAL

64 – PROM



5. What are static – 0 and static – 1 hazard? Explain the removal of hazards using hazard covers in k-map. (8) [M/J'16]

Static Hazards

A static hazard occurs when a single input variable change should cause no change in the output of a combinational logic circuit, but a short glitch of the incorrect logic level occurs.

The problem occurs because real physical implementations of logic functions have finite propagation times which are variable, and if two inputs to a gate should theoretically change simultaneously, one will actually change before the other.

If more than one input variable changes "simultaneously" there is no way to guarantee that such glitches will not occur.

Types of Static Hazards

Static – 1 hazard :- A static 1 hazard may occur in a two level sum of products (SOP) implementation.

Static – 0 hazard :- A static 0 hazard may occur in a two level product of sums (POS)implementation.

Example of Static Hazards

The Static '1' Hazard.

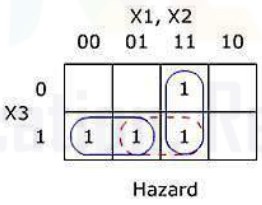
Let us consider an imperfect circuit that suffers from a delay in the physical logic elements i.e. AND gates etc.

The simple circuit performs the function:

f = X1.X2 + X1'.X3

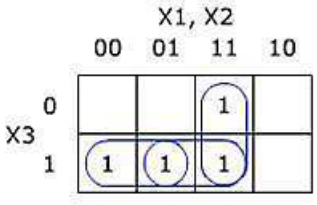
and the logic diagram can be shown as follows:

At the starting diagram, it is clear that if no delays were to occur, then the circuit would function normally. However since this is not a perfect circuit, and an error occurs when the input changes from 111 to 011. i.e. When X1 changes state.



Hazard

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 elimate the hazard. So the resulting logic is of the form shown in the next figure.



Hazard Eliminated

So our original function is now: f =X1.X2 + X1'.X3 + X2.X3

It is observed that even with imperfect logic elements, this example will not show signs of hazards when X1 changes state. This theory can be applied to any logic system.

6. Explain cycles and races in asynchronous sequential circuits. (8) [M/J'16]

Races in asynchronous sequential circuits

- Race condition: two or more binary state variables will change value when one input variable changes Cannot predict state sequence if unequal delay is encountered
- Non-critical race: The final stable state does not depend on the change order of state variables Critical race: The change order of state variables will result in different stable states.

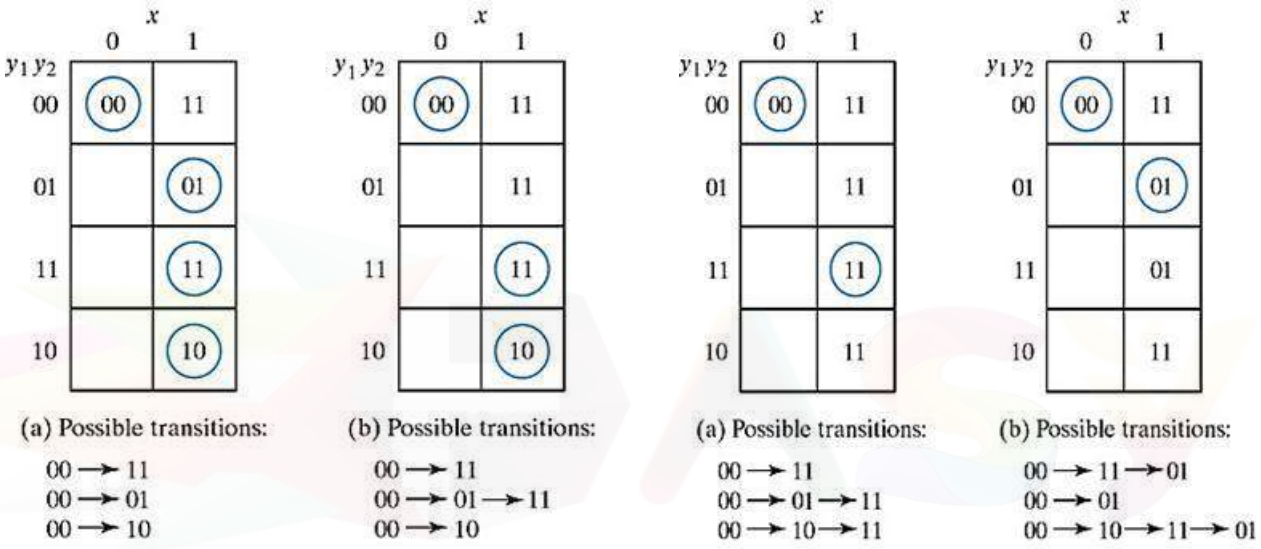


Fig. 9-7 Examples of Critical Races

Fig. 9-6 Examples of Noncritical Races

Cycles in asynchronous sequential circuits

- A cycle occurs when an asynchronous circuit makes a transition through a series of unstable states.
- When a state assignment is made so that it introduces cycles.
- Care must be taken so that each cycle terminates on a stable state.
- If a cycle does not contain a stable state, the circuit will go from one unstable state to another, until the inputs are changed.

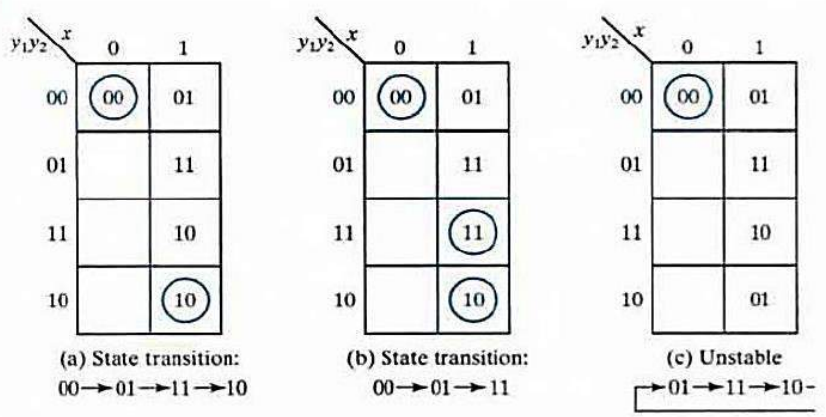


Fig: Examples of cycles

EE8351 – DIGITAL LOGIC CIRCUITS
UNIT – V
VHDL
PART – A

1. Write a VHDL code for 2 x 1 MUX. [N/D'14], [M/J'16], [A/M'17]

```
library ieee;
use ieee.std_logic_1164.all;
entity mux2_1 is
port (a,b,sel:instd_logic; c: out std_logic);
end mux2_1;
architecture muxarch of mux2_1 is
begin
process (a,b,sel)
begin
if s='0' then c<=a;else s='1' then c<=b; end if;
end process;
end muxarch;
```

2. State the advantage of package declaration over component declaration. [N/D'14]
   Package declaration is used to declare components, types, constants, functions and so on. Declared Packages will be shared by many design units.
   Component declaration declares the name of the entity and interface of a component which is used by the design unit. Declared Component will be used by the corresponding design unit.

3. What is a package in VHDL?[A/M'15]
   A VHDL package contains subprograms, constant definitions, and/or type definitions to be used throughout one or more design units. Each package comprises a "declaration section", in which the available (i.e. exportable) subprograms, constants, and types are declared, and a "package body", in which the subprogram implementations are defined, along with any internally-used constants and types.

4. Write the behavioural modelling code for a D flip flop.[A/M'15], [N/D'15], [N/D'16]

```
Library ieee;
use ieee.std_logic_1164.all;
entity dff is
port (D,clk,rst:instd_logic; Q: out std_logic);
end dff;
architecture behave of dff is
begin
process (rst,clk) Begin
if rst='0' then Q<='0'; elseclock'event and clk='1' then Q<=D;
end If;
end process;
end behave;
```

5. List out the operators present in VHDL.[N/D'15]
   Logical operators, Arithmetic operators, Relational operators and shift operators.

6. What is data flow modelling in VHDL? Give its basic mechanism.[M/J'16]
   A dataflow style architecture models the hardware in terms of the movement of data over continuous time between combinational logic components such as adders, decoders and primitive logic gates.
   Basic Mechanism:

```
entity entity_name is
port();
architecture dataflow of entity_name is
…..
begin
…
…
end dataflow;
```

7. Write the VHDL code for a logic gate which gives high output only when both the inputs are high. [N/D'16]

```
entity andgate is
    port(A:in std_logic;
         B:in std_logic;
         Y: in std_logic);
end andgate
```

8. Give the syntax for package declaration and package body in VHDL.[A/M'17]

```
package package_name is
        {package_declarative_item}
end [package_name];

package body package_name is
        {package_declarative_item}
end [package_name];
```

9. What is the purpose of VHDL programming? Or what is the need for VHDL? [M/ J- 13]
   Very high speed integrated circuit hardware description language. It is a language for describing a hardware, which has to be readable for machines and humans at the same time & it structured and comprehensible code, so that the source code can serve as a kind of specification document. Thus it is used for studying digital logic circuits and testing its functions.

PART – B

1. Write VHDL coding for 4 x 1 multiplexer. (7)[N/D'16]

4: 1 MUX

Truth Table:

Input	S1	S0	Ouptut
I0	0	0	10
I1	0	1	I1
I2	1	0	I2
I3	1	1	I3

Logic Diagram:

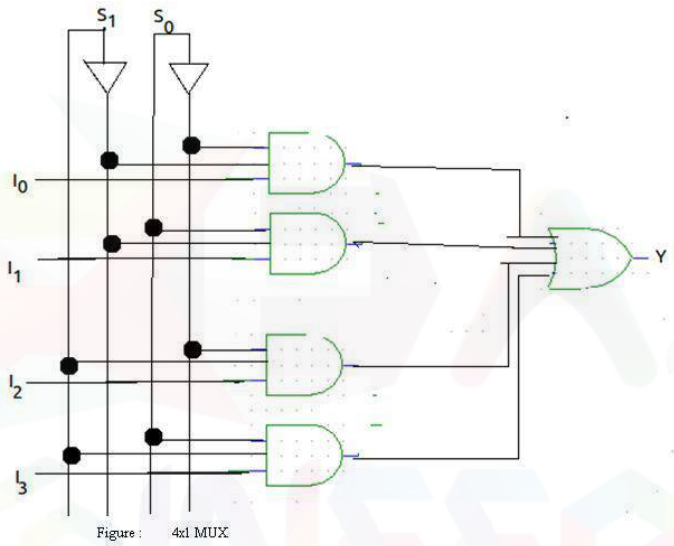


Figure : 4x1 MUX

Program:

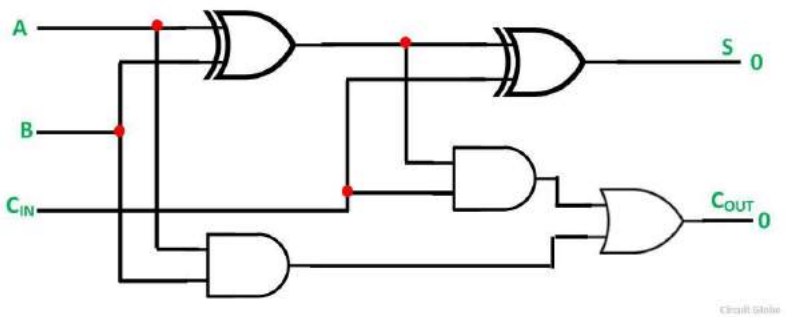
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux4x1 is
    Port ( din : in  STD_LOGIC_VECTOR (3 downto 0);
           en : in  STD_LOGIC;
             sel: in std_logic_vector(1 downto 0);
           dout : out  STD_LOGIC);
end mux4x1;
architecture mux4x1_arch of mux4x1 is
signal iy:std_logic;
begin
    with sel select
        iy<=din(0) when "00",
            din(1) when "01",
            din(2) when "10",
            din(3) when "11",
            '0' when others;
    dout<=iy when en='1' else '0';
end mux4x1_arch;
```

2. Write the VHDL code to realise a full adder using (16)[A/M'15]
3. Write a VHDL program for full adder using structural modelling. (8) [N/D'15]
4. Explain in detail the concept of structural modelling in VHDL with an example of full adder. (13) [N/D'16]
5. Explain the concept of behavioural modelling and structural modelling in VHDL. Take the example of full adder design for both and write the coding.(8) [N/D'14]

Truth Table:

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Logic Diagram



Program

a) Behavioural modelling

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

entity fab is
   Port ( a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          s : out std_logic;
          cr : out std_logic);
end fab;

architecture Behavioral of fab is
begin
        process(a,b,c)
        begin
        if(a='0' and b='0' and c='0')then
        s<='0';
        cr<='0';
        elsif( a='0' and b='0' and c='1')then
        s<='1';
        cr<='0';
        elsif( a='0' and b='1' and c='0')then
        s<='1';
        cr<='0';
        elsif( a='0' and b='1' and c='1')then
        s<='0';
        cr<='1';
        elsif( a='1' and b='0' and c='0')then
        s<='1';
        cr<='0';
        elsif( a='1' and b='0' and c='1')then
        s<='0';
        cr<='1';
```

```
elsif( a='1' and b='1' and c='0')then
s<='0';
cr<='1';
else
s<='1';
cr<='1';
end if;
end process;
end Behavioral;
```

b) Structural modelling

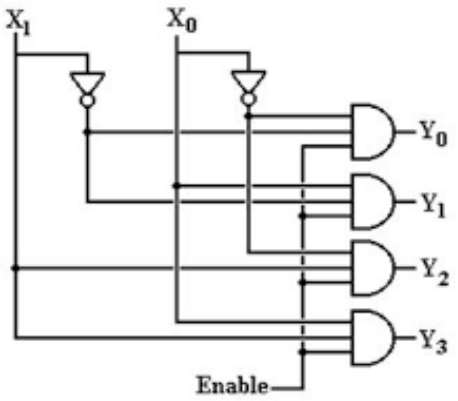
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is
Port ( a,b,cin : in  STD_LOGIC;
sum,cout : out  STD_LOGIC);
end full_adder;

architecture fa_str of full_adder is
component xor2
port(d1,d2:in std_logic;
dz:out std_logic);
end component;
component or2
port(n1,n2:std_logic;
z:out std_logic);
end component;
component and2
port(a1,a2:in std_logic;
u:out std_logic);
end component;
signal s1,s2,s3,s4,s5:std_logic;
begin
        x1:xor2 port map(a,b,s1);
        x2:xor2 port map(s1,cin,sum);
        r1:and2 port map(a,b,s2);
        r2:and2 port map(b,cin,s3);
        r3:and2 port map(a,cin,s4);
        o1:or2 port map(s2,s3,s5);
        o2:or2 port map(s4,s5,cout);
        end fa_str;
```

6. Write a VHDL program for 1 to 4 Demux using dataflow modelling. (8) [N/D'15]

Truth Table:

Input	Select Lines	Output Lines
I	S_1 S_0	D_0 D_1 D_2 D_3
I	0 0	1 0 0 0
I	0 1	0 1 0 0
I	1 0	0 0 1 0
I	1 1	0 0 0 1

Logic Diagram:



Program:

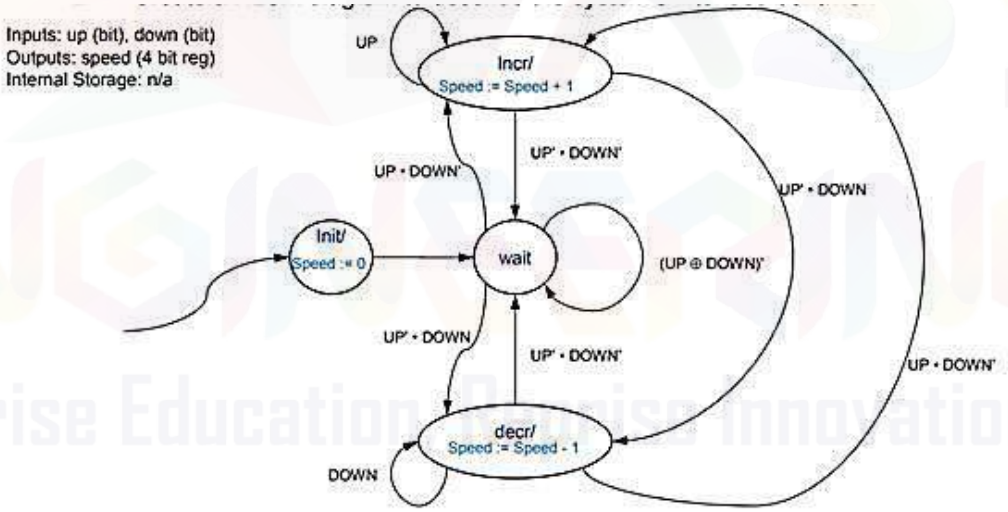
```
library IEEE;
use IEEE.std_logic_1164.all;

entity bejoy_1x4 is
port(s1,s2,data_in : in std_logic;
d1,d2,d3,d4 : out std_logic);
end bejoy_1x4;

architecture arc of bejoy_1x4 is

component dmux
port(sx1,sx2,d : in std_logic;
z1,z2 : out std_logic);
end component;

begin
dmux1 : dmux port map(s1,s2,data_in,d1,d2);
dmux2 : dmux port map(not s1,s2,data_in,d3,d4);
end arc;

library ieee;
use ieee.std_logic_1164.all;

entity dmux is
port(sx1,sx2,d :in std_logic;
z1,z2: out std_logic);
end dmux;

architecture arc of dmux is
begin
z1 <= d and (not sx1) and (not sx2);
z2 <= d and (not sx1) and sx2;
end arc;
```

7. Explain in detail the RTL design procedure. (16) [N/D'15]
   1. Capture a HLSM
      - Create a HLSM diagram to describe the system's intended behavior.
   2. Convert to a Circuit
      a) Create a datapath
         - Create a datapath to carry out the data operations of the HLSM.
         - Use components from a library
         - Include registered outputs.
      b) Connect the datapath to a controller
         - Connect all control signals to the circuit
      c) Derive the controller's FSM.
         - Convert the HLSM to a FSM for the controller
            - Replace data operations with setting and reading of control signals to and from the datapath.
         - Create a circuit for the controller from the FSM

Example:

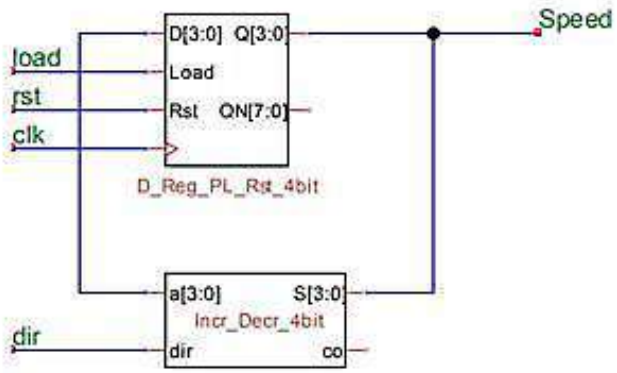
Design a system to control the speed of the conveyor belt on a treadmill.

- Speed is a 4 bit value that is controlled by two buttons
  - Up button increases speed by one
  - Down button decreases speed by one
  - If both are pushed, no change in speed occurs. Speed must initialize to zero upon startup.

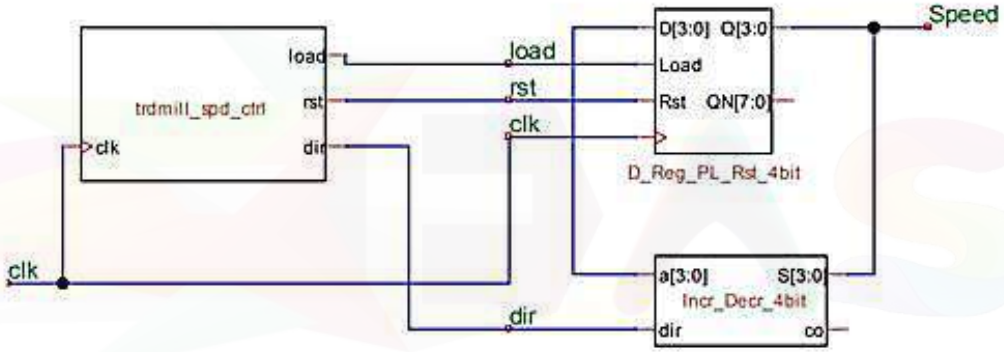


Convert to a Circuit

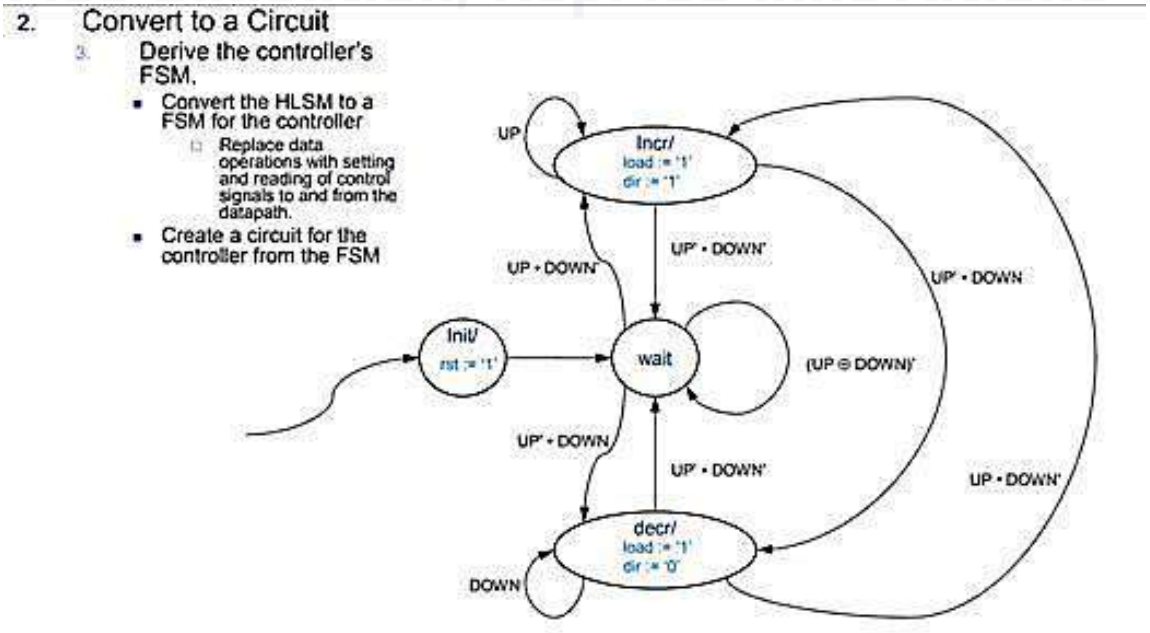
1. Create a data path
   - Create a data path to carry out the data operations of the HLSM.
   - Use components from a library
   - Include registered outputs.



2. Convert to a Circuit
   - Connect the datapath to a controller
   - Connect all control signals to the circuit



- Convert to a Circuit
  - Derive the controller's FSM.
  - Convert the HLSM to a FSM for the controller
  - Replace data operations with setting and reading of control signals to and from the datapath.
  - Create a circuit for the controller from the FSM



8. Explain the various operators supported by VHDL. (8) [M/J'16]
9. Write short notes on built – in operators used in VHDL programming. (6) [N/D'16]

Logical Operators

VHDL Operator	Operation	Operand Type	Result Type
not	logical not	boolean, bit[_vector], std_logic[_vector]	same type
and	logical and	boolean, bit[_vector], std_logic[_vector]	same type
or	logical or	boolean, bit[_vector], std_logic[_vector]	same type
nand	logical nand	boolean, bit[_vector], std_logic[_vector]	same type
nor	logical nor	boolean, bit[_vector], std_logic[_vector]	same type
xor	logical xor	boolean, bit[_vector], std_logic[_vector]	same type
xnor	logical xnor	boolean, bit[_vector], std_logic[_vector]	same type

Examples:

```
carry <= a and b or a and c or b and c;
zout <= (not a) and b;
```

Relational Operators

VHDL Operator	Operation	Operand Type		Result Type
		Left	Right	
=	equality	any type	any type	boolean
/=	inequality	any type	any type	boolean
<	less than	any scalar type	any scalar type	boolean
<=	less than or equal to	any type escalar	any scalar type	boolean
>	greater than	any type escalar	any scalar type	boolean
=>	greater than or equal to	any type escalar	any scalar type	boolean

Example:

```
data <= (a=0) and (b=1);
```

Arithmetic Operators

- The circuit used for an arithmetic operator will be entirely combinational logic
- Arithmetic operators are implemented in two's complement
- The negation operator is implemented as a two's complement negation. Two's complement negation is performed by subtracting the input from zero
- The add operator is usually implemented as a ripple-carry adder. The same circuit is used for either signed or unsigned arithmetic.
- The subtractor operator is implemented as a ripple-borrow subtractor

VHDL Mathematical Operators

VHDL Operator	Operation	Operand Type Left	Operand Type Right	Result Type
Miscellaneous				
**	exponential	any integer	integer	same as left
abs	absolute value	any numeric tpye	any numeric type	same numeric type
Arithmetic				
*	multiplication	any numeric	any numeric	Same numeric
/	division	any numeric	any numeric	Same numeric
mod	modulus	any numeric	any numeric	Same numeric
rem	remainder	any numeric	any numeric	Same numeric

VHDL Operator	Operation	Operand Type Left	Operand Type Right	Result Type
Unary				
+	identity	any numeric	any numeric	same type
-	negation	any numeric	any numeric	same type
Adding				
+	addition	any numeric	same type	same type
-	substraction	any numeric	same type	same type
&	concatenation	any array	same array	same array
		any array	the element	same array
		the element	any array	same array
		the element	the element	any array

VHDL Shift Operators

VHDL Operator	Operation	Operand Type Left	Operand Type Right	Result Type
Shift				
sll	logical shift left	one dimensional array of bit or std_logic	integer	same as left
srl	logical shift right	one dimensional array of bit or std_logic	integer	same as left
sla	arithmetic shift left	one dimensional array of bit or std_logic	integer	same as left
sra	arithmetic shift right	one dimensional array of bit or std_logic	integer	same as left
rol	logical rotate left	one dimensional array of bit or std_logic	integer	same as left
ror	logical rotate right	one dimensional array of bit or std_logic	integer	same as left