a

complete

note

for

LOGIC CIRCUIT

prepared

by

PRALHAD CHAPAGAIN

a

complete

note

for

LOGIC CIRCUIT

prepared

by

PRAHLAD CHAPAGAIN

Numerical representation:

Numerical data in a computer are written in basic units of storage made up of a fixed number of consecutive bits. The most commonly used units in the computer and communication industries are the byte (8 consecutive bits) and word (16 consecutive bits), and the double word (32 consecutive bits). A number is represented in each of these units by setting the bits according to the binary representation of the number. By convention the bits in a byte are numbered, from right to left, beginning with zero. Thus, the rightmost bit is bit number 0 and the leftmost bit is number 7. The rightmost bit is called the least significant bit and the leftmost bit is called the most significant bit.

Digital number system:

A digital system can understand positional number system only where there are a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

A value of each digit in a number can be determined using

* The digit
* The position of the digit in the number
* The base of the number system

S.N.	Number system and Description
1.	Binary Number system • Base 2, Digit used : 0, 1
2	Octal number system • Base 8 Digit used : 0 to 7
3.	Decimal number system • Base 10 Digit used : 0 to 9

4.	Hexa Decimal Number System Base 16. Digits used: 0 to 9. Letters used: A-F

Analog system

Analog system carries signals in the forms of voltage, current which are continuous function of continuous time variable. In other words analog system processes analog signal to give output in required analog form.

An analog signal is any time continuous signal where some time varying feature of the signal is a representation of some other time varying quantity eg. Sound, light, temperature, position, radio wave etc.

Advantages:

* It has the potential for an infinite amount of signal resolution
* Processing may be achieved more simply than digital

Disadvantages:

* The primary disadvantage of analog signal is that in any system it is completely influenced by noise i.e. random variation.

Digital system

In a digital system, each signal is represented by a sequence of numbers. In other words digital system are those system which proces digital input to give the output in digital form. Digital signals are digital representation of discrete time signals which are often derived from analog signal.

Eg: Cell phones, calculator, digital camera etc.

Advantages

* More immune to noise and it is reliable
* Processing of digital system is more flexible and accurate
* Storage of digital system is easy.
* Modification of digital system is easy and effective
* Privacy is preserved
* Greater dynamic range
* Any kind of information (data, voice, and video) can be merged
* Errors can be detected and correcting using coding.

Disadvantages

* Bandwidth requirement is very high
* Quantization noise affect the system
* Synchronization is required

Digital computer:

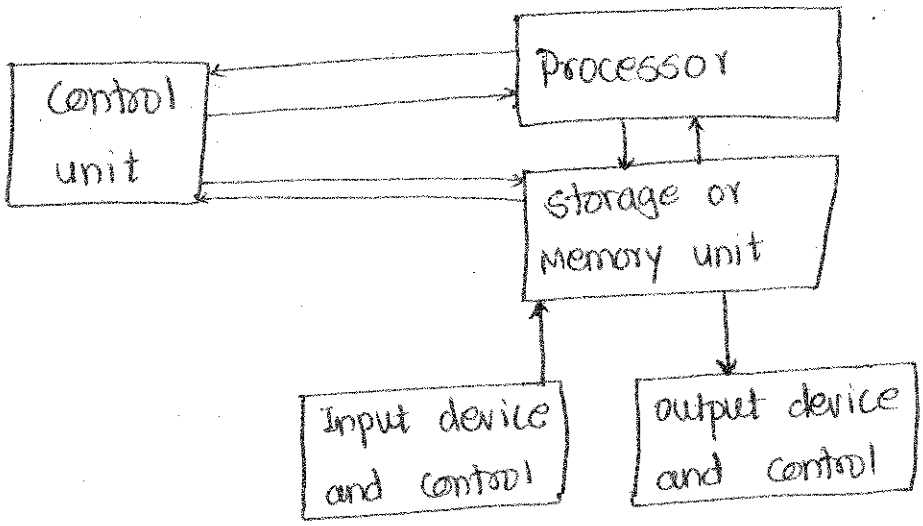


Fig: Block diagram of a digital computer/system

The memory unit:

It stores programs as well as input output and intermediate data. Ex: RAM, ROM

The processor unit:

It performs arithmetic, logical and other data processing task as specified by the program. Eg: Microprocessor

The control unit:

It supervises the flow of information between the various units. It retrieves the instructions one by one from the program which is stored in memory. For each instruction control unit informs the processor to execute the operation specified by the instruction.

Input device:

The program and data prepared by the users are transferred into the memory unit by means of input devices such as: keyboard, mouse etc.

Output device:

It receives the result of the computation and the results are presented to the user. Eg: Printer, monitor etc.

example of digital computer is: electronic calculator

NOTE:

* A CPU enclosed in small integrated-circuit package is called a microprocessor.
* A CPU combined with memory and interface control to form a small-size computer is called a microcomputer.

Number Representation system:

Decimal Base 10	Binary Base 2	Octal Base 8	Hexadecimal Base 16
0	0 0 0 0	0	0
1	0 0 0 1	1	1
2	0 0 1 0	2	2
3	0 0 1 1	3	3
4	0 1 0 0	4	4
5	0 1 0 1	5	5
6	0 1 1 0	6	6
7	0 1 1 1	7	7
8	1 0 0 0	10	8
9	1 0 0 1	11	9
10	1 0 1 0	12	A
11	1 0 1 1	13	B
12	1 1 0 0	14	C
13	1 1 0 1	15	D
14	1 1 1 0	16	E
15	1 1 1 1	17	F

Conversion of Binary to decimal

i) $(1010)_2 = (?)_{10}$

$= 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

$= 8 + 0 + 2 + 0$

$= 10$

$\therefore (1010)_2 = (10)_{10}$

ii) $(1111)_2 = (?)_{10}$

$= 1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$

$= 8 + 4 + 2 + 1$

$= 15$

$\therefore (1111)_2 = (15)_{10}$

iii) $(0.1111)_2 = (?)_{10}$

$= 1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$

$= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = \frac{8+4+2+1}{16} = \frac{15}{16} = 0.9375$

$\therefore (0.1111)_2 = (0.9375)_{10}$

iv) $(100110.01)_2 = (?)_{10}$

$= 1\times2^{5} + 0\times2^{4} + 0\times2^{3} + 1\times2^{2} + 1\times2^{1} + 0\times2^{0} + 0\times2^{-1} + 1\times2^{-2}$

$= 32 + 0 + 0 + 4 + 2 + 0 + 0 + \frac{1}{4} = 38 + 0.25 = 38.25$

$\therefore (100110.01)_2 = (38.25)_{10}$

Conversion of Decimal to Binary

i) $(10)_{10} = (?)_2$

$\therefore (10)_{10} = (1010)_2$

2	10	
2	5	→ 0
2	2	→ 1
	1	→ 0 ↑

ii) $(255)_{10} = (?)_2$

$\therefore (255)_{10} = (11111111)_2$

2	255	
2	127	→ 1
2	63	→ 1
2	31	→ 1
2	15	→ 1
2	7	→ 1
2	3	→ 1
	1	→ 1 ↑

iii) $(0.6875)_{10} = (?)_2$

	integer	fraction	coefficient
0.6875×2	= 1	+ 0.3750	$a_{-1} = 1$
0.3750×2	= 0	+ 0.7500	$a_{-2} = 0$
0.7500×2	= 1	+ 0.5000	$a_{-3} = 1$
0.5000×2	= 1	+ 0.0000	$a_{-4} = 1$

$\therefore (0.6875)_{10} = (0.1011)_2$

iv) $(29.125)_{10} = (?)_2$

2	29	
2	14	→ 1
2	7	→ 0
2	3	→ 1
	1	→ 1 ↑

For fraction

$0.125\times2 = 0.250 \rightarrow 0$

$0.250\times2 = 0.500 \rightarrow 0$

$0.500\times2 = 1.000 \rightarrow 1$

$\therefore (29.125)_{10} = (11101.001)_2$

Conversion of octal to decimal

i) $(765)_8 \rightarrow (?)_{10}$

$= 7\times8^2 + 6\times8^1 + 5\times8^0$

$= 448 + 48 + 5$

$= (501)_{10}$

$\therefore (765)_8 = (501)_{10}$

ii) $(0.765)_8 \rightarrow (?)_{10}$

$= 7\times8^{-1} + 6\times8^{-2} + 5\times8^{-3}$

$= \frac{7}{8} + \frac{6}{64} + \frac{5}{512}$

$= (0.9785)_{10}$

$\therefore (0.765)_8 = (0.9785)_{10}$

Conversion of decimal to octal

i) $(501)_{10} = (?)_8$

$\Rightarrow (501)_{10} = 765$

8	501	
8	62	→ 5
	7	→ 6 ↑

ii) $(0.84)_{10} = (?)_8$

$\Rightarrow 0.84\times8 = 6.72 \rightarrow 6$

$0.72\times8 = 5.76 \rightarrow 5$

$0.76\times8 = 6.08 \rightarrow 6$

$0.08\times8 = 0.64 \rightarrow 0$

$0.64\times8 = 5.12 \rightarrow 5$

⋮

$(0.65605)_8$

$\Rightarrow (0.84)_{10} = (0.65605)_8$

Conversion of octal to Binary

i) $(765)_8 = (?)_2$

$\therefore (765)_8 = (\text{nonnnnnnn})_2$

2	765	
2	382	→ 1
2	191	→ 0
2	95	→ 1
2	47	→ 1
2	23	→ 1
2	11	→ 1
2	5	→ 1
2	2	→ 1
	1	→ 0

$= 7\times8^2+6\times8^1+5\times8^0 \quad = (501)_{10}$

$\therefore (765)_8 = [111110101]_2$

2	501
2	250 → 1
2	125 → 0
2	62 → 1
2	31 → 0
2	15 → 1
2	7 → 1
2	3 → 1
	1 → 1

Directly convert

7 → 111
6 → 110
5 → 101
$\Rightarrow (765)_8 =$ $(111110101)_2$

ii) $(437.126)_8 = (?)_2$

$= 4\times8^2+3\times8^1+7\times8^0+1\times8^{-1}+2\times8^{-2}+6\times8^{-3}$

$= (287.1679688)_{10}$

$= (100011111.001010110)_2$

2	287
2	143 → 1
2	71 → 1
2	35 → 1
2	17 → 1
2	8 → 1
2	4 → 0
2	2 → 0
	1 → 0

For fraction

$0.1679688\times2 = 0.3359375 \to 0$
$0.3359375\times2 = 0.671875 \to 0$
$0.671875\times2 = 1.34375 \to 1$
$0.34375\times2 = 0.6875 \to 0$
$0.6875\times2 = 1.375 \to 1$
$0.375\times2 = 0.75 \to 0$
$0.75\times2 = 1.5 \to 1$
$0.5\times2 = 1.0 \to 1$
$0.0\times2 = 0 \to 0$

Direct conversion

4 → 100
3 → 011
7 → 111
1 → 001
2 → 010
6 → 110

$\therefore (437.126)_8 = (100011111.001010110)_2$

Conversion of Binary to octal

i) $(100011111.00101011)_2 \to (?)_8$

$= 1\times2^8+0\times2^7+0\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+0\times2^{-2}+1\times2^{-3}$
$+0\times2^{-4}+1\times2^{-5}+0\times2^{-6}+1\times2^{-7}+1\times2^{-8}$

$= 256+16+8+4+2+1+\frac{1}{8}+\frac{1}{32}+\frac{1}{128}+\frac{1}{256}$

$= (287.1679688)_{10}$

$\therefore (437.126)_8$

8	287
8	35 → 7
	4 → 3

For fraction

$0.1679688\times8 = 1.34375 \to 1$
$0.34375\times8 = 2.75 \to 2$
$0.75\times8 = 6 \to 6$

Conversion of hexadecimal to Decimal

i) $(3CD.01A)_{16} \to (?)_{10}$

$= 3\times16^2+12\times16^1+13\times16^0+0\times16^{-1}+1\times16^{-2}+10\times16^{-3}$

$= (973.0063477)_{10}$

ii) $(FA0)_{16} \to (?)_{10}$

$= 15\times16^2+10\times16^1+0\times16^0 \quad = (4000)_{10}$

Conversion of Decimal to hexadecimal :

i) $(1024)_{10} = (?)_{16}$

$\therefore (1024)_{10} = (400)_{16}$

16 | 1024
16 | 64 → 0
4 → 0 ↑

ii) $(0.750)_{10} = (?)_{16}$

$0.750 \times 16 = 12.0 \rightarrow C$

$\therefore (0.750)_{10} = (0.C)_{16}$

Conversion of hexadecimal to binary

i) $(FA0)_{16} = (?)_2$

$\Rightarrow (FA0)_{16} = (1111\ 1010\ 0000)_2$

Conversion of hexadecimal to octal:

i) $(400)_{16} \rightarrow (?)_8$

$= (400)_{16} \rightarrow (\underbrace{00}_{2}\ \underbrace{000}_{0}\ \underbrace{000}_{0}\ \underbrace{000}_{0})_2$

$\therefore (400)_{16} = (2000)_8$

eg: $(4021.2)_5 = (?)_{10}$

$= 4\times5^3 + 0\times5^2 + 2\times5^1 + 1\times5^0 + 2\times5^{-1}$

$= (511.4)_{10}$

eg: $(4\ 7\ 9)_{2421} = (?)_2$

$\therefore 4 \rightarrow 0100$
$7 \rightarrow 1101$
$9 \rightarrow 1111$

$\therefore (479)_{2421} \rightarrow (0100\ 1101\ 1111)_2$

ADD and multiply

i) $(1230)_4$ and $(23)_4$

Add: $(1^{1}230)_4$
$+(23)_4$
$(1313)_4$

aiken code

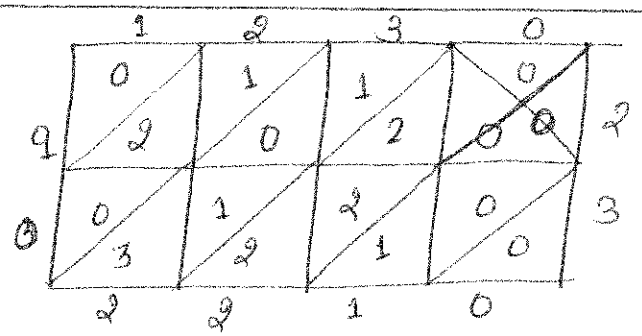
Decimal	2	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

multiply: $(1230)_4$ × $(23)_4$

$(11010)_4$

$(3120)_4$

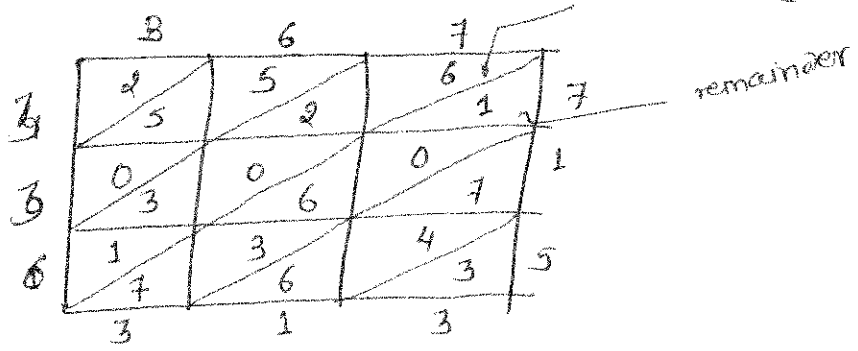
$(10221 0)_4$



$(10221 0)_4$.

ii) $(367)_8$ and $(715)_8$

multiply:



$(336313)_8$.

Complement's

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulations. There are two types of complements for each base-r system.

1) the r's complement
2) the (r-1)'s complement

1) The r's complement

Given a positive number N in base r with an integer part of n digits, the r's complement of N is defined as,

$$\begin{cases} r^n - N & \text{for } N \neq 0 \\ 0 & \text{for } N = 0 \end{cases}$$

Eg:

① Find 10's complement of $(52520)_{10}$

Here, $r = 10$, $n = 5$, $N = 52520$

∴ 10's complement $= r^n - N = 10^5 - 52520 = 100000 - 52520$

$= 47480$

② Find 10's complement of $(0.3267)_{10}$

Here, $r = 10$, $n = 0$, $N = 0.3267$

∴ 10's complement $= r^n - N = 10^0 - 0.3267 = 1 - 0.3267$

$= 0.6733$

3) Find 10's complement of $(25.639)_{10}$

Here, $r = 10$, $n = 2$, $N = 25.639$

$\therefore$ 10's complement $= r^n - N = 10^2 - 25.639 = 100 - 25.639$

$= 74.361$

4) Find the 2's complement of $(101100)_2$

Here, $r = 2$, $n = 6$, $N = 101100$

$\therefore$ 2's complement $= r^n - N = (2^6)_{10} - (101100)_2 = (64)_{10} - (101100)_2$

$= (1000000)_2 - (101100)_2 = (010100)_2$

5) Find the 2's complement of $(0.0110)_2$

Here, $r = 2$, $n = 0$, $N = (0.0110)_2$

$\therefore$ 2's complement $= r^n - N = (2^0)_{10} - (0.0110)_2 = (1)_2 - (0.0110)_2$

$= (0.1010)_2$

2) The (r-1)'s complement

Given a positive number N in base r with an integer part of n digits and a fraction part of m digits, the (r-1)'s complement of N is defined as $r^n - r^{-m} - N$ for $N \neq 0$.

Eg: Find (r-1)'s complement of:

a) $(52520)_{10}$

Here, $r = 10$, $n = 5$, $m = 0$, $N = (52520)_{10}$

$\therefore$ 9's complement $= 10^5 - 10^0 - 52520$

$= 100000 - 1 - 52520$

$= 47479$

b) $(0.3267)_{10}$

Here, $r = 10$, $n = 0$, $m = 4$, $N = (0.3267)_{10}$

9's complement $= 10^0 - 10^{-4} - (0.3267)_{10}$

$= 1 - \frac{1}{10000} - 0.3267$

$= (0.6732)_{10}$

c) $(25.639)_{10}$

Here, $r = 10$, $n = 2$, $m = 3$, $N = (25.639)_{10}$

9's complement $= 10^2 - 10^{-3} - (25.639)_{10}$

$= 99.999 - 25.639 = (74.360)_{10}$

d) $(101100)_2$

Here, $r = 2$, $n = 6$, $m = 0$, $N = (101100)_2$

1's Complement $= (2^6 - 2^0)_{10} - (101100)_2$

$= (63)_{10} - (101100)_2 = (111111)_2 - (101100)_2$

$= (010011)_2$

e) The 1's complement of $(0.0110)_2$

Here, $r = 2$, $n = 0$, $m = 4$, $N = (0.0110)_2$

1's complement $= (2^0 - 2^{-4})_{10} - (0.0110)_2$

$= \left(1 - \frac{1}{16}\right)_{10} - (0.0110)_2$

$= (0.1111)_2 - (0.0110)_2 = (0.1001)_2$

NOTE:

⇒ 10's complement of a decimal number can be formed by leaving all least significant zeros unchanged, subtracting the first nonzero least significant digit from 10, and then subtracting all other higher significant digits from 9.

⇒ 2's complement can be formed by leaving all least significant zeros and the first non zero digit unchanged, and then replacing 1's by 0's and 0's by 1's in all other higher significant digits.

⇒ 9's complement of a decimal number is formed simply by subtracting every digit from 9.

⇒ 1's complement of a binary number is formed simply by changing 1's to 0's and 0's to 1's.

⇒ r's complement can be obtained from the (r-1)'s complement after the addition of r^{-m} to the least significant digit.

[Q. How can you find r's complement using (r-1)'s complement? Explain with example]

Subtraction with r's complement:

The subtraction of two positive numbers (M - N), both of base r, may be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N
2. Inspect the result obtained in step 1 for an end carry:
   (a) If an end carry occurs, discard it. The remaining is the answer.
   (b) If an end carry does not occur, take the r's complement of the number obtained in step 1 and place a negative sign in front, that is a required answer.

Examples:

(a) using r's complement, subtract

(i) 72532 - 3250

Here, M = 72532 and N = 3250 = 03250

Now, r's complement of N, Nr's = 10^5 - 03250.

= 96750

Now, M + Nr's = 72532 + 96750

= 169282

Here, 1 is end carry value so it is discarded. Then, the required answer = 69282.

(ii) 3250 - 72532

Here, M = 3250 and N = 72532

= 03250

Now, r's complement of N, Nr's = 10^5 - 72532 = 27468

Now, M + Nr's = 03250 + 27468

= 30718

Here, no end carry present. So, take r's complement. i.e. r's complement of (M + Nr's) is. 10^5 - 30718

= 69282

Thus, required solution = -69282

(iii) $(1010100)_2 - (1000100)_2$

Here, M = 1010100, N = 1000100

Now, r's complement of N, Nr's = $(2^7)_{10} - (1000100)_2$

$(10000000)_2 - (1010100)_2 = (111100)_2$

Now, M + Nr's = 1010100 + 111100

= 1 0010000

∴ Here, 1 is end carry so it is discarded. Then required answer is, $(10000)_2$

(iv) $(1000100)_2 - (1010100)_2$

Here, M = 1000100 and N = 1010100

Now, r's complement of N. Nr's = $(2^7)_{10} - (1010100)_2$

$= (10000000)_2 - (1010100)_2$

$= (0101100)_2$ ~~(0101000)~~

Now, M + Nr's = 1000100 + ~~0101000~~ (101100)$_2$

$= (1101100)_2$ ~~(1101100)~~ $(1110000)_2$

∴ Here, no end carry present so, takes r's complement of

M + Nr's = $(2^7)_{10} - (1110000)_2$

$= (10000000)_2 - (1101100)_2$

$= (10000)_2$

∴ Required solution is, $(-10000)_2$

Subtraction using (r-1)'s complement:

The subtraction of two positive numbers M-N, both of base r may be done as follows:

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

② Inspect the result obtained in step 1

ⓐ If an end carry occurs, add 1 to the least significant digit (end-around carry). The result is answer

ⓑ If an end carry does not occur, take (r-1)'s complement of number obtained in step 1 and place negative (-ve) sign.

Examples:

Subtract using (r-1)'s complement

① 72532 - 3250

Here, M = 72532 , N = 3250 = 03250

Now, (r-1)'s complement of N, N(r-1)'s $= 10^5 - 10^0 - 03250$

$= 96749$

and, M + N(r-1)'s = 72532 + 96749

= (1)69281

Here, end carry is present. So,

Required answer = $\begin{array}{r} 69281 \\ +\ 1 \\ \hline 69282 \end{array}$

② $(1010100)_2 - (1000100)_2$

Here, M = 1010100 and, N = 1000100

Now, (r-1)'s complement of N, N(r-1)'s $= (2^7 - 2^0)_{10} - (1000100)_2$

$= (1111111 - 1000100)_2$

$= (0111011)_2$

and, M + N(r-1)'s $= (1010100)_2 + (0111011)_2$

$= (1\,1000\,1111)_2$

Here, end carry is present. So, Required answer is, $\begin{array}{r} 0001111 \\ +\ 1 \\ \hline (10000)_2 \end{array}$

③ 1000100 - 1010100

Here, M = 1000100 and N = 1010100

Now, (r-1)'s complement of N, N(r-1)'s $= (2^7 - 2^0)_{10} - (1010100)_2$

$= (1111111)_2 - (1010100)_2$

$= (0101011)_2$

Now, M + N(r-1)'s $= (1000100 + 0101011)_2$

$= (1101111)_2$

Here, no end carry present so, (r-1)'s complement of M + N(r-1)'s is, $(2^7 - 2^0)_{10} - (1101111)_2 = (1111111)_2 - (1101111)_2$

$= (10000)_2$

∴ Required solution is, $(-10000)_2$

Binary code:

Binary code is the code used in digital computers based on a binary number system in which there are only two possible states '0' and '1' usually called 'off' and 'ON'. It is possible to arrange 'n' bits to 2^n possible states.

Binary coded decimal code:

Binary codes for decimal digits required a minimum a 4 bits. Numerous different codes can be obtained by arranging 4 or more bits in 10 (ten) distinct possible combinations. BCD is a straight assignment of binary equivalent. The weights in the BCD codes are 8 4 2 1.

The Binary conversion of 13 is 1101 But the coding of decimal with BCD is 00010011

Decimal digit	BCD 8 4 2 1	Excess - 3	8 4 -2 -1	aiken 2 4 2 1	Biquinary 5 0 4 3 2 1 0
0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	0 1 0 0 0 0 1
1	0 0 0 1	0 1 0 0	0 1 1 1	0 0 0 1	0 1 0 0 0 1 0
2	0 0 1 0	0 1 0 1	0 1 1 0	0 0 1 0	0 1 0 0 1 0 0
3	0 0 1 1	0 1 1 0	0 1 0 1	0 0 1 1	0 1 0 1 0 0 0
4	0 1 0 0	0 1 1 1	0 1 0 0	0 1 0 0	0 1 1 0 0 0 0
5	0 1 0 1	1 0 0 0	1 0 1 1	1 0 1 1	1 1 0 0 0 0 0
6	0 1 1 0	1 0 0 1	1 0 1 0	1 1 0 0	1 0 0 0 0 1 0
7	0 1 1 1	1 0 1 0	1 0 0 1	1 1 0 1	1 0 0 0 1 0 0
8	1 0 0 0	1 0 1 1	1 0 0 0	1 1 1 0	1 0 0 1 0 0 0
9	1 0 0 1	1 1 0 0	1 1 1 1	1 1 1 1	1 0 1 0 0 0 0

Gray code (The Reflected code):

Its an unweighted code. The advantage of the gray code over pure binary code is that a number in the gray code changes by only one bit as it proceeds from one number to the next. A typical application of the gray code occurs when the analog data are represented by a continuous change of a position.

Gray code	Decimal equivalent
0 0 0 0	0
0 0 0 1	1
0 0 1 1	2
0 0 1 0	3
0 1 1 0	4
0 1 1 1	5
0 1 0 1	6
0 1 0 0	7
1 1 0 0	8
1 1 0 1	9
1 1 1 1	10
1 1 1 0	11
1 0 1 0	12
1 0 1 1	13
1 0 0 1	14
1 0 0 0	15

Conversion

① Binary to Gray:

(a) $(10110)_2 = (?)_{gray}$

Here, $1 \rightarrow 1$

$1+0 \rightarrow 1$

$0+1 \rightarrow 1$

$1+1 \rightarrow 0$

$1+0 \rightarrow 1$

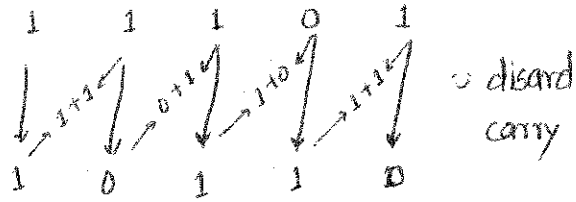
∵ discard carry

$\therefore (10110)_2 = (11101)_{gray}$

② Gray to Binary:

(a) $(11101)_{gray} = (?)_2$

Here,



$\therefore (11101)_{gray} = (10110)_2$

Excess-3 code:

A decimal code that has been used in some old computer is the excess-3 code. This is an unweighted code; its code assignment is obtained from the corresponding value of BCD after the addition of 3.

The Excess-3 code is self-complementary code, i.e. the 9's complement of the decimal number is easily obtained by the changing 1's to 0's and 0's to 1's.

Proof: Excess-3 code of $(2)_{10}$ is $(0101)_{Ex-3}$ ..... ①

Now, 9's complement of 2 is, $\Rightarrow r^n - r^{-m} - N$

$= 10^1 - 10^0 - 2 \quad = 10 - 3$

$= 7$

Now, $(7)_{10} \xrightarrow{\text{Excess-3}} (1010)_{Ex-3}$ ..... ⑪

If we obtained 1's complement of ①, we get

$(0101)_{Ex-3} \xrightarrow{\text{1's complement}} (1010)_{Ex-3}$ ..... ⑪

As, Equation ⑪ and ⑪ are equal. It means that 1's complement of a number expressed in excess-3 code is equivalent to 9's complement of decimal number itself. So, Excess-3 code is considered to be self complementary code.

NOTE: The 2, 4, 2, 1 and the 8, 4, -2, -1 are also self-complementary codes.

(Proof yourself).

Error detection code:

Any external noise introduced into a physical communication medium changes bit values from 0 to 1 or vice versa. An error-detection is the ability to detect the presence of errors caused by noise or other impairments during transmission from transmitter to receiver. The detected error can not be corrected, but its presence is indicated.

Methods:

- Parity Bit
- checksum
- Cyclic Redundancy check

Parity Bit

Parity Bit is a bit (1 or 0) that is added to ensure that the number of bits with the value one (1) in a set of bits is even or odd. It is used as simplest form of error ~~dect~~ detecting code. There are two types of parity bit.

1) even parity bit
2) odd parity bit

1) even parity bit

When using even parity, the parity bit is set to 1 if the number of 1's in a given set of bits is odd, making the entire sets of bit (1's) even.

2) odd parity bit

When using odd parity, the parity bit is set to 1 if the number of 1's in a given set of bits is even, making the entire sets of bit (1's) odd.

eg: Using even parity

A ⟶ 01101 ⟶ B

As, no of 1's is odd, we should add 1 to make even parity.

So, Error detecting code [EDC] = 01101 1
original code parity bit

B receives, 011011 (correct signal)

If noise occur along transmission. Let, the EDC becomes 011111

That means B receives, 011111 which has odd parity. That means incorrect signal is received.

Limitation:

The parity method detects the presence of any odd combination of errors. An even combination of error is undetectable.

NOTE:

No parity: Lack of any error checking or the bit used for parity checking

pace parity: form of information data integrity check where the parity bit is always 0; otherwise an error has occured.

ark parity: form of information data integrity check where the parity bit is always 1; otherwise an error has occured.

Instruction code:

An instruction code is a group of bits that instructs the computer to perform the specific operation. It is divided into two parts: op-code and operand.

The most basic part of an instruction code is its operation parts called as 'op-code'. The operation code of an instruction is a group of bits that defines particular operation add, subtract, multiply, divide etc. The number of bits required for the operation code of an instruction depend on the total number of operation available in the computer. The operation code must consist of atleast n bits, for given 2^n distinct operators.

Consider a computer with 64 distinct operation. One of them is add operation. The operation code consist of six (6) bit with the bit configuration '110010' assign to the add operator. When this operation is decode in the control unit, the computer issue control signal to read and operand from memory and add operand to the processor resistance.

Alpha-numeric code:

An alpha-numeric code is a binary code of a group of elements consisting of ten decimal digits, The 26 letters of alphabet and certain number of special symbol, such as #, $ etc. The total number of elements in an alpha-numeric code is greater than 36. It must be coded with the minimum of 6 bits. Two mostly used alpha-numeric are ASCII code and EBCDIC.

1) ASCII code:

The American standard code for Information Interchange (ASCII) is a widely used alpha-~~number~~ numeric code. It is basically a 7-bit code. Since, it can create $2^7 = 128$ bit patterns. The ASCII can be used to encode both the lower case and upper case characters of alphabet and some symbols '#', '$' etc.

A → 65 0 → 48
a → 97 blank → 32

2) EBCDIC:

The Extended Binary Coded Decimal Interchange Code (EBCDIC) is an 8-bit alpha numeric code. It can create $2^8 = 256$ different bit patterns. EBCDIC code can encode all the symbol and character found in ASCII code. It also encodes many other symbols which are not encoded by ASCII code. Infact, many of the bit patterns in EBCDIC code are unassign.

A → 193 a → 129
0 → 240
blank → ~~64~~ 64

Determine the value of Base x if:

i) $(211)_x = (152)_8$

so i'n: $2x^2 + 1x^1 + 1x^0 = 1\times8^2 + 5\times8^1 + 2\times8^0$

or, $2x^2 + x + 1 = 64 + 40 + 2$

or, $2x^2 + x - 105 = 0$

solving eq^n. we get

$x = 7$ #

ii) $(193)_x = (623)_8$

so i'n: $1x^2 + 9x^1 + 3x^0 = 6\times8^2 + 2\times8^1 + 3\times8^0$

or, $x^2 + 9x + 3 = 384 \rightarrow 16 + 3$

or, $x^2 + 9x - 400 = 0$

solving, eq^n we get

$x = 16$ #

It is defined with the set of elements, set of operator and number of postulates. It is developed by George Boole in 1854 and was considered as Boolean algebra. In Boolean algebra $1+1=1$, where as in ordinary binary algebra $1+1=10$.

Binary operator:

A binary operator defined on a set 'S' of elements is a rule that assigns to each pair of elements from 'S' a unique element from 'S'.

Eg: If $(a,b) \in S$ and

$a * b = c \quad \because c \in S$

Then, * is a binary operator.

Boolean laws:

There are three types of laws:

1) Commutative law:

Any binary operation which satisfies the following expression is called as commutative operation:

Law 1 : $A+B = B+A$

Law 2 : $A \cdot B = B \cdot A$

Proof:

		Law 1		Law 2	
A	B	A+B	B+A	A·B	B·A
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	1	1	1

Actually, the commutative law states that changing the sequence of the variables (inputs) does not have any effect on the output of a logic circuit.

2) Associative law:

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same. This means that we have:

Law 1 : $A+(B+C) = (A+B)+C$

Law 2 : $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

Proof:

A	B	C	B+C	A+(B+C)	A+B	(A+B)+C	B·C	A·(B·C)	A·B	(A·B)·C
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	0	0	0	0
0	1	0	1	1	1	1	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0
1	0	0	0	1	1	1	0	0	0	0
1	0	1	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1

Law 1 — Law 2

$A+(B+C) = (A+B)+C$ $\quad$ $A\cdot(B\cdot C) = (A\cdot B)\cdot C$

3. Distributive law:

The Distributive law states that.

Law 1: $A\cdot(B+C) = A\cdot B + A\cdot C$

Law 2: $(A+B)\cdot(A+C) = A+B\cdot C$

A	B	C	A·B	A·C	A·B+A·C	B+C	A·(B+C)	B·C	A+B·C	A+B	A+C	(A+B)·(A+C)
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	0	0	0	1	0	0
0	1	1	0	0	0	1	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1	0	1	1	1	1
1	1	0	1	0	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1

Law 1 — Law 2

$A\cdot(B+C) = A\cdot B + A\cdot C$ $\quad$ $(A+B)\cdot(A+C) = A+B\cdot C$

Rules in Boolean Algebra:

In Boolean algebra certain rules are followon. They are:

1. We will use capital letters for representating variable and function of variable.

2. The complement of a variable is represented by a bar over a letter eg $\bar{A}$, $\bar{B}$
3. The logical 'AND' function of two variable is represented by 'dot' (.)
4. The logical 'OR' operation of two variable is represented by 'plus' (+)
5. The addition of variables in boolean algebra involves the variable either a 'zero' or 'one' eg: $0+0=0$, $0+1=1$, $1+0=1$, $1+1=1$
6. Multiplication in Boolean algebra follows the same basic rules that are in binary operations.
   eg $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, $1 \cdot 0 = 0$, $1 \cdot 1 = 1$.

Some Basic rules:

1) $A + 0 = A$
2) $A + 1 = 1$
3) $A \cdot 0 = 0$
4) $A \cdot 1 = A$
5) $A + A = A$
6) $A + \bar{A} = 1$
7) $A \cdot \bar{A} = 0$
8) $A \cdot A = A$
9) $\bar{\bar{A}} = A$
10) $A + AB = A(1+B) = A \cdot 1 = A$
11) $A + \bar{A}B = A + B$

Duality Principle:

- An important property of Boolean algebra
- It states that in a two valued Boolean algebra, the dual of an algebric expression can be obtained simply by interchanging OR and AND operators and by replacing 1s by 0s and 0s by 1s.
- It has several applications in digital systems

Duality Theorem

According to the duality theorem, the following conversions are possible in a given Boolean expression:

i) We can change each AND operation to an OR operation
ii) We can change each OR operation to an AND operation
iii) We can complement any 1 or 0 appearing in the expression.

→ duality theorem is sometimes useful in creating new expressions from the given Boolean expressions.

Example: Find the dual of following Boolean expressions:

1. $A + AB = A$

2. $A + \bar{A}B = A + B$

3. $A + \bar{A} = 1$

4. $(A+B)(A+C) = A + BC$

Sol^n:

S.N.	Given expression	Dual of given expression
1	$A + AB = A$	~~$A(\bar{A}+B) = AB$~~ $A \cdot (A+B) = A$
2	$A + \bar{A}B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$
3	$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$
4	$(A+B)(A+C) = A + BC$	$A \cdot B + A \cdot C = A \cdot (B + C)$

Reduce the following Boolean expression:

a) $A \cdot \bar{A} \cdot C = ?$

$= 0$

b) $ABCD + ABD = ?$

$= ABD(C+1)$

$= ABD$

c) $XY + XYZ + XY\bar{Z} + \bar{X}YZ$

$= XY + XY\bar{Z} + XYZ + \bar{X}YZ$

$= XY(1+\bar{Z}) + YZ(X+\bar{X})$

$= XY + YZ$

$= Y(X+Z)$

d) $AB + \bar{A}C + A\bar{B}C(AB+C)$

$= AB + \bar{A}C + A\bar{B}C AB + A\bar{B}CC$

$= AB + \bar{A}C + 0 + A\bar{B}C$

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

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

$= A(B+C) + \bar{A}C \quad [\because B + \bar{B}C = B + C]$

$= AB + AC + \bar{A}C$

$= AB + 1 \quad [\because AC + \bar{A}C = 1]$

$= 1$

Prove the following Boolean expression:

i) $(A+B)(A+C) = A + BC$

L.H.S $(A+B)(A+C)$

$= AA + AC + BA + BC$

$= A + AC + BA + BC$

$= A(1+C) + BA + BC$

$= A(1+B) + BC$

$= A + BC$

$= R.H.S$

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

L.H.S $(A+B)(A+\bar{B})(\bar{A}+C)$

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

$= [A(1+\bar{B}+B+0)](\bar{A}+C)$

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

$= A\bar{A} + AC$

$= 0 + AC$

$= AC$

$= R.H.S$

iii) $ABC + A\bar{B}C + AB\bar{C} = A(B+C)$

L.H.S. $ABC + A\bar{B}C + AB\bar{C}$

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

$= AC + AB\bar{C}$

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

$= A(B+C)$

$= R.H.S$

iv) $A + \bar{A}B = A + B$

L.H.S. $A + \bar{A}B$

$= A \cdot 1 + \bar{A}B$

$= A(1+B) + \bar{A}B$

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

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

$= A + B$

$= R.H.S$

Logic Gates:

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one inputs and only one output. The relationship between the input and the output is based on a certain logic.

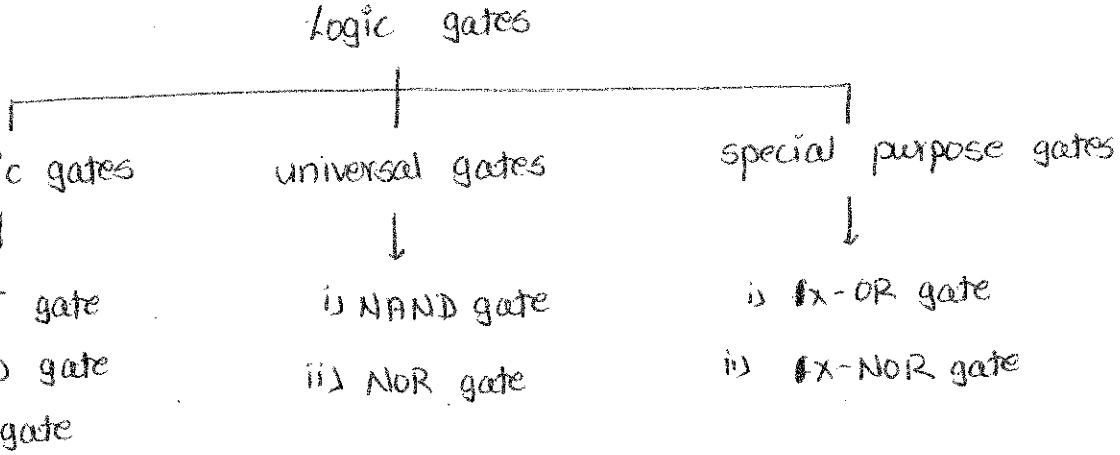


fig: classification of gates

Truth table

The operation of a logic gate can be best understood with the help of a table called Truth Table. The truth table consists of all the possible combinations of the inputs and the corresponding state of output of a logic gate.

Boolean expression

The relationship between the inputs and the outputs of a gate can be expressed mathematically by means of the Boolean expression.

1) NOT gate: (IC 7404)

NOT gate performs logic operation complement. It has one i/p one o/p.

logic symbol:

Truth table

x	$\bar{x}$
0	1
1	0

2) AND gate: (IC 7408)

The AND gate performs Logic operation multiplication. It may have two or more input and one output

logic symbol:

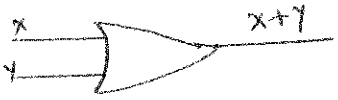
Truth table

x	y	x·y
0	0	0
0	1	0
1	0	0
1	1	1

3) OR gate: (IC 7432)

The OR gate performs logic operation addition. It may have two or more input and one output.

logic symbol:



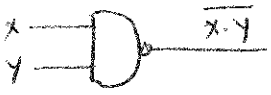
Truth table

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

4) NAND gate: (IC 7400)

The NAND gate performs combination of 'NOT' and 'AND' operation. It has two or more inputs and one output.

Logic symbol:



Truth table

x	y	x·y	$\overline{x \cdot y}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

5) NOR gate: (IC 7402)

NOR gate is a combination of 'NOT' and 'OR' operation. It have two (or more) input and one output

Logic symbol:

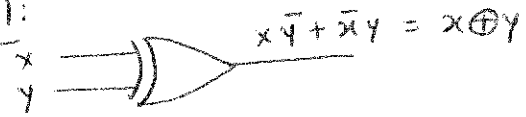
Truth table

x	y	x+y	$\overline{x+y}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

6) X-OR gate: (IC 7486)

It is extensively known as exclusive 'OR' gate. It has two or more inputs and single output

logic symbol:



Truth table

x	y	$\bar{x}$	$\bar{y}$	$x\bar{y}$	$\bar{x}y$	$x \oplus y = x\bar{y} + \bar{x}y$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

) X- 'NOR' gate (IC 74266)

It is also called exclusive NOR gate. It has two or more input and single output.

Logic gate:

$= X \odot Y = \overline{x\bar{y} + \bar{x}y} = xy + \bar{x}\bar{y}$

Truth table:

X	Y	$X \odot Y$
0	0	1
0	1	0
1	0	0
1	1	1

* Simplify the expression mentioned below and draw logic circuit for simplified expression.

i) $\bar{A}BC + ABC + A\bar{B}C + AB\bar{C} + AB\bar{C} + ABC$

$= BC(\bar{A}+A) + AC(\bar{B}+B) + AB(\bar{C}+C)$

$= BC + AC + AB$

Logic circuit:

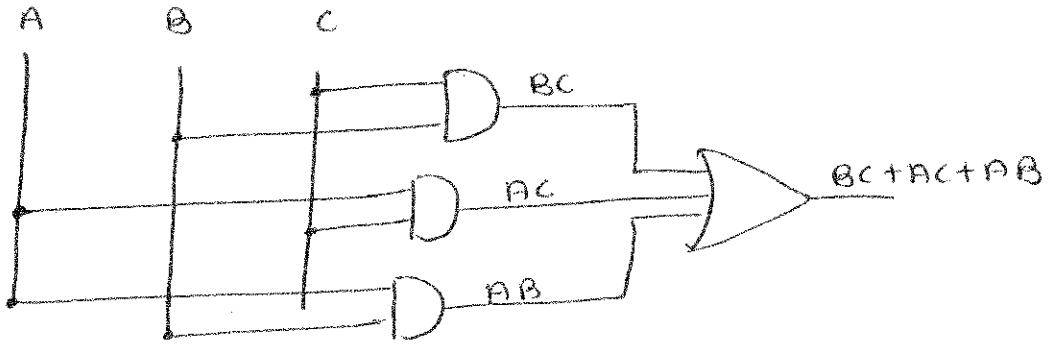


fig: Logic circuit implementation of simplified expression

ii) $ABC + \bar{A}\bar{B}\bar{C} + AB\bar{C} + A\bar{B}$

$= ABC + AB\bar{C} + \bar{A}\bar{B}\bar{C} + A\bar{B}$

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

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

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

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

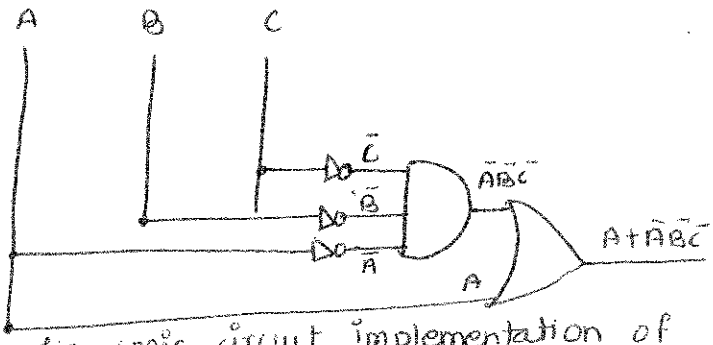


fig: Logic circuit implementation of simplified expression.

De- Morgan's Theorem:

De - Morgan Suggested two theorems that forms important part of Boolean algebra.

De - Morgan's 1st law:

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

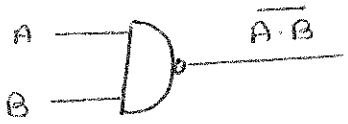
"The complement of a product is equal to the sum of individual complement" i.e. $\overline{A \cdot B} = \bar{A} + \bar{B}$

Truth table

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

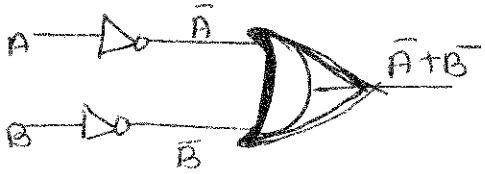
Proof: L.H.S.

$\overline{A \cdot B}$



R.H.S.

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



Truth table

A	B	$\bar{A}$	$\bar{B}$	$\bar{A}+\bar{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

$\therefore$ L.H.S = R.H.S

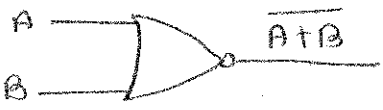
proved

De - Morgan's 2nd law:

"The complement of sum is equal to the product of the individual complement" i.e. $\overline{A+B} = \bar{A} \cdot \bar{B}$

Proof: L.H.S

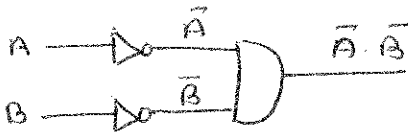
$\overline{A+B}$



Truth table

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

R·H·S $= \bar{A} \cdot \bar{B}$



Truth table:

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

$\therefore$ L·H·S = R·H·S

proved

Introduction to universal gates:

The NAND and NOR gates are called as universal Gates because it is possible to implement any boolean expression with the help of only NAND or only NOR gates. Hence, a user can build any combinational circuit with the help of only NAND gates or only NOR gates. This is a great advantage because a user will have to make a stock of only NAND or NOR gate ICs.

1) NAND Gate as a universal gate:

A NAND gate is expressed mathematically as:

$$Y = \overline{A \cdot B}$$

Hence, we have to bring the given Boolean expression into this form.

a) NOT gate using NAND gate

When, Input, A = B

$$Y = \overline{A \cdot B} = \overline{A \cdot A}$$

But, $A \cdot A = A$

$\therefore Y = \bar{A}$

Fig: NOT gate using NAND gate

b) AND gate using NAND gate

The Boolean expression for an AND gate is, $Y = A \cdot B$

Taking double inversion of RHS, $y = \overline{\overline{A \cdot B}} = A \cdot B$

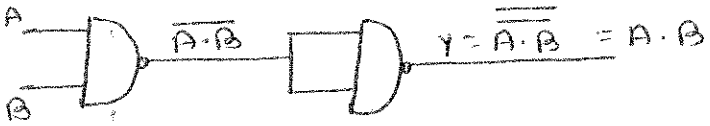


Fig: AND gate using NAND gate

c) OR gate using NAND gate

The Boolean expression for OR gate is,

$$Y = A + B$$

Taking double inversion on RHS. we get,

$$Y = \overline{\overline{A+B}}$$

Using De-morgan's theorem,

$$\overline{A+B} = \bar{A} \cdot \bar{B}$$

$$\therefore Y = \overline{\bar{A} \cdot \bar{B}}$$

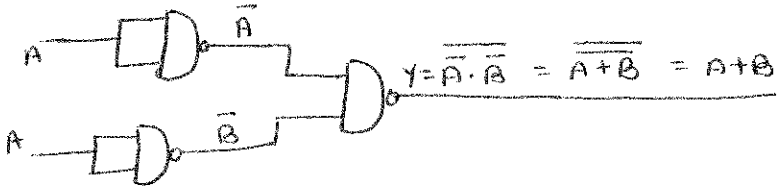


Fig: OR gate using NAND gate

d) NOR gate using NAND gate:

The Boolean expression for NOR gate is.

$$Y = \overline{A+B}$$

According to De Morgan's theorem,

$$Y = \overline{A+B} = \bar{A} \cdot \bar{B}$$

Taking double inversion of RHS, we get

$$Y = \overline{\overline{\bar{A} \cdot \bar{B}}}$$

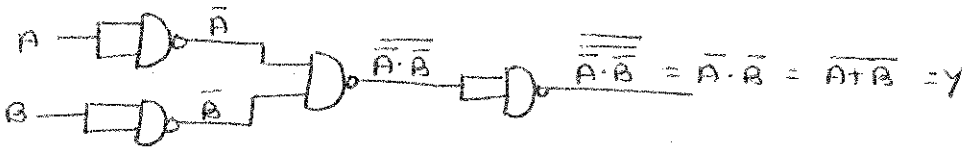


Fig: NOR gate implementation of NAND gate

e) x-OR gate using NAND gate only:

The Boolean expression for x-OR gate is,

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

Taking double inversion of RHS, we get $Y = \overline{\overline{\bar{A}B + A\bar{B}}}$

using De-morgan's theorem:

$$\overline{\bar{A}B \cdot A\bar{B}}$$

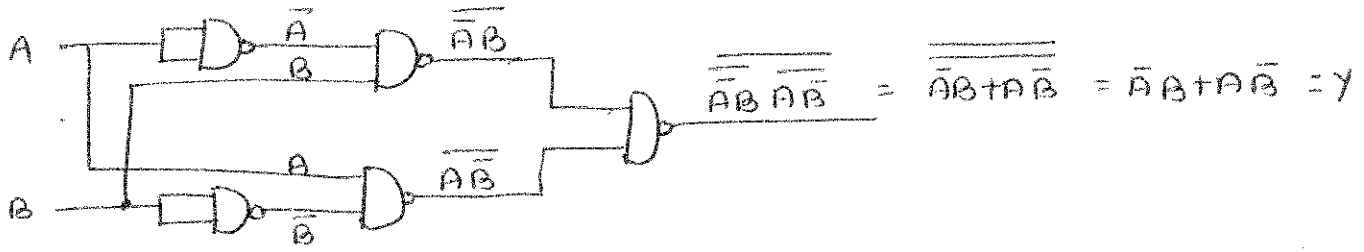


Fig: X-OR gate using NAND gate

f) X-NOR using NAND gate only

The Boolean expression for X-NOR is given by.

$$Y = A \odot B = \bar{A}\bar{B} + AB$$

Taking double inversion of RHS.

$$Y = \overline{\bar{A}\bar{B} + AB}$$

$$= \overline{\overline{\bar{A}\bar{B}} \cdot \overline{AB}}$$

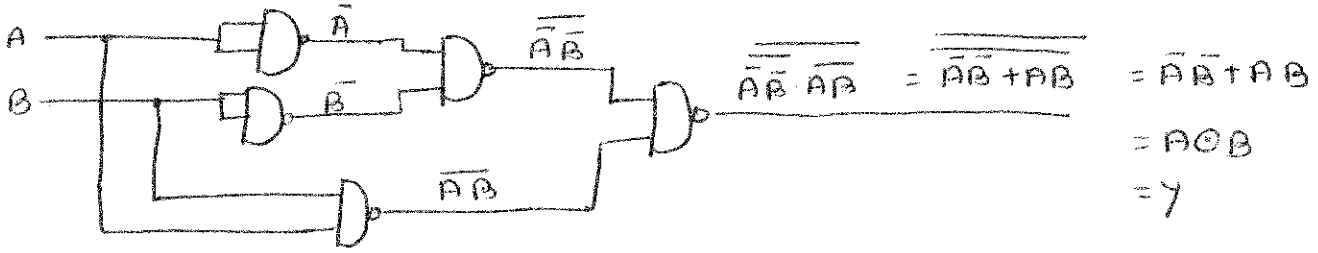


Fig: X-NOR using NAND gate

g) NOR gate as universal gate:

The Boolean expression of the given logic circuit must be first converted into the NOR format. $Y = \overline{A+B}$

g) NOT gate using NOR gate

When, input $A = B$

$$Y = \overline{A+B} = \overline{A+A}$$

$\because A + A = A$

$\therefore Y = \bar{A}$

Fig: NOT gate using NOR gate

b) OR gate using NOR gate:

Boolean expression for an OR gate is, $Y = A+B$

Taking double inversion of RHS, $Y = \overline{\overline{A+B}}$

$\therefore Y = A+B = \overline{\overline{A+B}}$

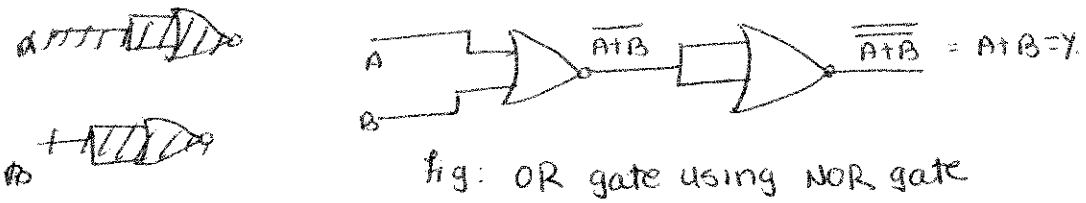


fig: OR gate using NOR gate

c) AND gate using NOR gate:

Boolean expression for an AND gate is, $Y = A \cdot B$

Taking double inversion of RHS. $Y = \overline{\overline{A \cdot B}} = \overline{\overline{A}+\overline{B}}$ [: using de-morgan's theorem]

Therefore,

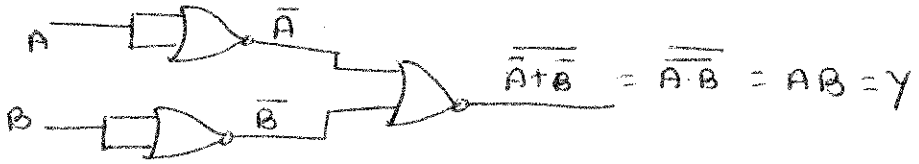


fig: AND gate using NOR gate

D) NAND gate using NOR gate:

Boolean expression for an NAND gate is,

$Y = \overline{A \cdot B}$

using, De-morgan's theorem $Y = \overline{A}+\overline{B}$

Taking double inversion on RHS, $Y = \overline{\overline{\overline{A}+\overline{B}}}$

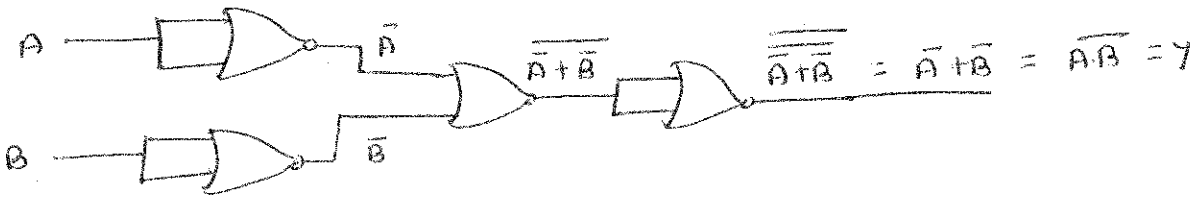


fig: NAND gate using NOR gate

e) X-OR gate using NOR gate

Boolean expression for x-OR gate is,

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

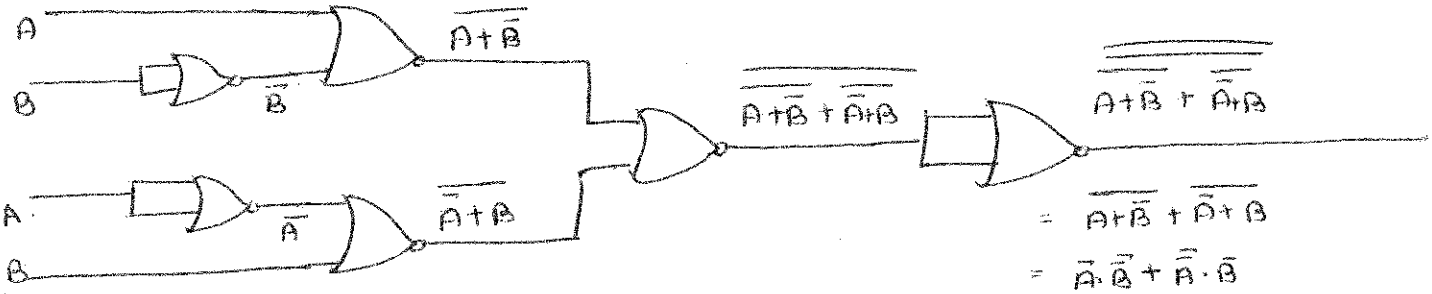
Taking double inversion on RHS.

$$Y = \overline{\overline{\bar{A}B + A\bar{B}}} = \overline{\bar{A}B} \cdot \overline{A\bar{B}} = \overline{\bar{A}B} + \overline{A\bar{B}}$$

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

Taking double inversion on RHS.

$$Y = \overline{\overline{\overline{A + \bar{B}} + \overline{\bar{A} + B}}}$$



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

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

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

$= Y$

Fig: x-OR gate using NOR gate

f) X-NOR gate using NOR gate

Boolean expression for x-NOR gate is,

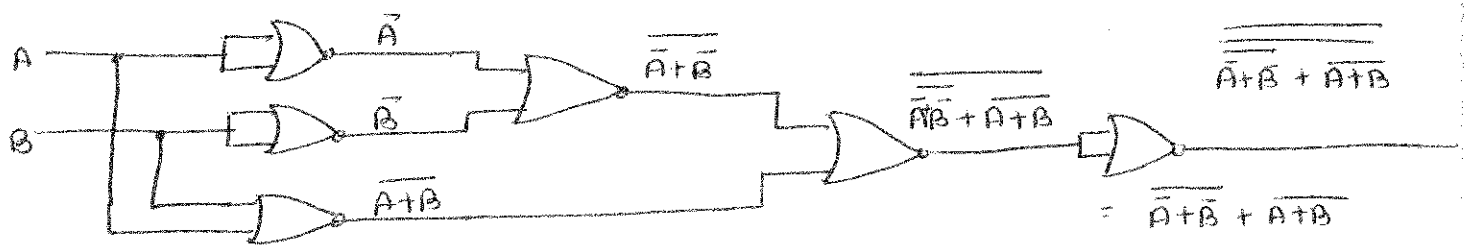
$Y = A \odot B = \bar{A}\bar{B} + AB$

Taking double inversion on RHS

$$Y = \overline{\overline{\bar{A}\bar{B} + AB}} = \overline{\bar{A}\bar{B}} \cdot \overline{AB}$$

$$= \overline{\bar{\bar{A}} + \bar{\bar{B}}} \cdot \overline{\bar{A} + \bar{B}} = \overline{A + B} + \overline{\bar{A} + \bar{B}}$$

Taking double inversion on RHS, $Y = \overline{\overline{\overline{A+B} + \overline{\bar{A}+\bar{B}}}}$



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

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

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

$= Y$

Fig: x-NOR gate using NOR gate

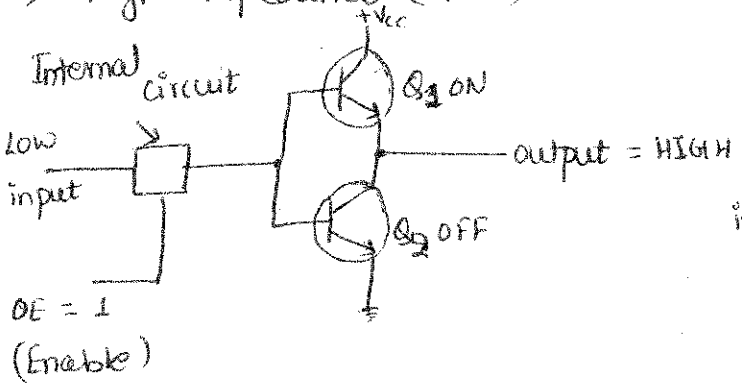
Tristate (Three state) Logic:

Tristate logic allows three possible output states namely,

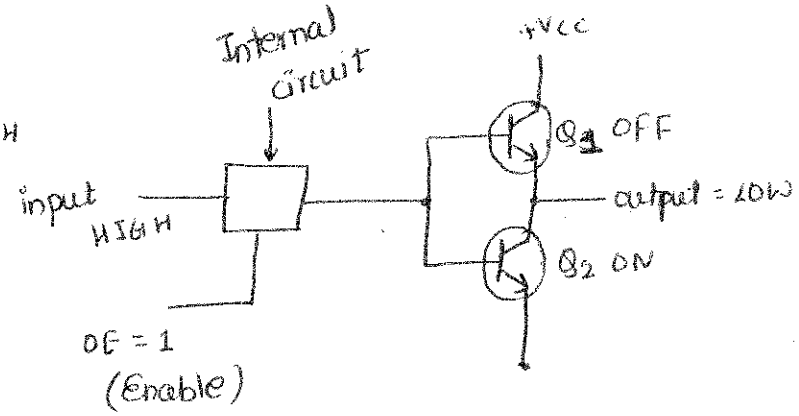
i) High

ii) Low and

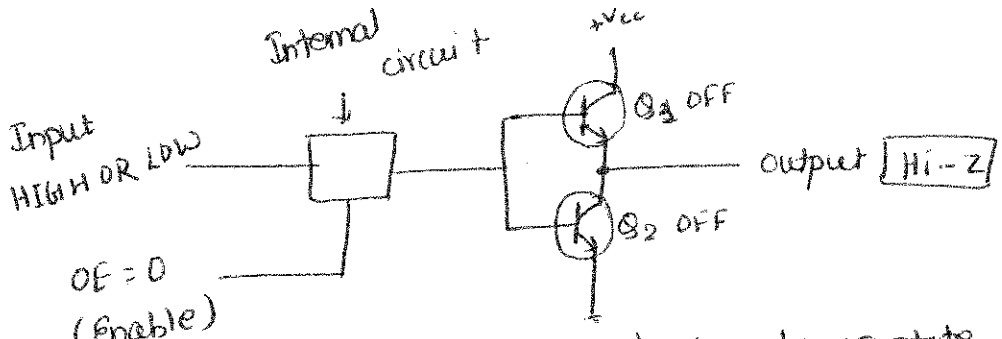
iii) High impedance (Hi-Z) state.



(a) State 1 : High



(b) State 2 : LOW



(c) State 3 : High impedance state

An additional input called output enable (OE) is introduced. Q_1 and Q_2 are pull up and pull down transistors. The output stage block diagram of a tri-state inverter is shown in figure above.

When, OE = 1, then the inverter operates as a normal inverter. If input is 0 then output will be 1 (HIGH). The pull up transistor Q_1 will be ON

Similarly with OE = 1 and input HIGH (1), the output will be low because the pull down transistor Q_2 will be ON.

High Impedance State:

If the enable output OE = 0 (zero), then irrespective of the status of input, both the transistors will remain off as shown in figure (c). This state of operation is called as high impedance (Hi-z) state. In this state the output terminal is essentially open circuit i.e., not connected anywhere.

A Boolean function is an expression formed with binary variables, two binary operations 'OR' & 'AND', the unary operator 'NOT', "equal" sign. For a given Boolean function the output can be either 0 or 1.

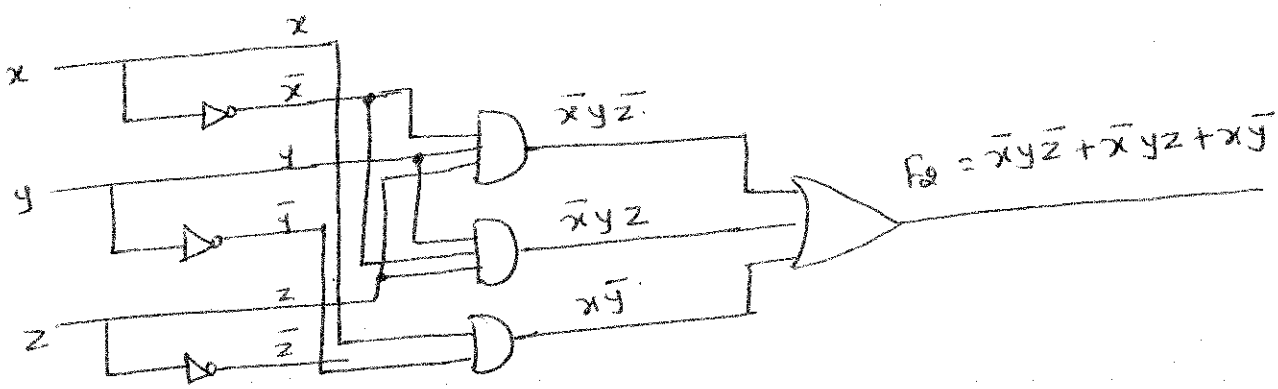
Example: $f_1 = xy\bar{z}$

The function f_1 is 1 when the input $x = 1, y = 1$ and $z = 0$.

Implementation of Boolean function usig GATE:

A Boolean function may be transformed from an algebric expression into a logic diagram, composed of 'AND', 'OR' and 'NOT' gate. The logic diagram includes an inverter circuit, for every variable present in its complement form.

Example: $f_2 = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}$



Algebric Manipulation of Boolean function:

A literal is a primed or unprimed variable. When the Boolean function is implemented with logic gates each literal designates to a 'AND' gate, and each terms is implemented with the gate. The minimization of a number of literals and the number of terms results in a circuit with less equipments. The number of literals in a Boolean function can be minimized by algebric manipulation.

Eg: $x + x'y = (x + x')(x + y)$ $\Rightarrow x(x' + y) = x \cdot x' + xy$

$= x + y$ $= xy$

Complement of function:

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebrically through De Morgan's theorem.

The generalized form of De Morgan's theorem states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

A simpler procedure for deriving the ~~component~~ complement of a function is to take the dual of the function and complement each literal. This method follows from the generalized De Morgan's theorem.

Example:

1. $F_1 = x'yz' + x'y'z$

The dual of F_1 is $(x'+y+z')(x'+y'+z)$

Complement each literal : $(x+y'+z)(x+y+z') = F_1'$

Venn Diagram:

It is used to show relationship among the variables of the boolean expression. The diagram consist of a rectangle inside of which overlapping circles are drawn 1 for each variables.

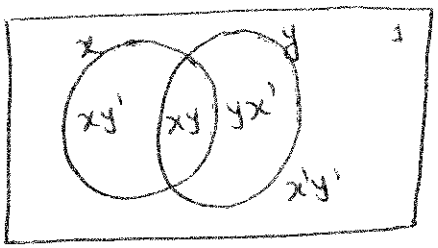


fig: Venn diagram

Here, x and y are two overlapping circles in which x.y areas are common to both, xy' areas belongs to x-only and x'y belongs to y-only. x'.y' area is area not belonging to any circle.

Venn diagram is also used to illustrate the postulates of boolean

algebra and to show the validity of theorems.

Example:

$$x + xy = x$$

$$x + x' = 1$$

Test vectors:

Test vector is a set of inputs provided to a system in order to test that system.

Standard form and canonical form

Canonical form:

Canonical form are of two types.

① Max term

② Min term

① Max term

Each max term is obtained from an 'OR' logic of n variables, with each variable being unprime if the corresponding bit is zero(0) and primed if one(1).

X	Y	Z	Max term $x+y+z$	Designation
0	0	0	$x + y + z$	M_0
0	0	1	$x + y + z'$	M_1
0	1	0	$x + y' + z$	M_2
0	1	1	$x + y' + z'$	M_3
1	0	0	$x' + y + z$	M_4
1	0	1	$x' + y' + z'$	M_5
1	1	0	$x' + y' + z$	M_6
1	1	1	$x' + y' + z'$	M_7

Min terms:

Each min term is obtained from ~~and~~ 'AND' logic of the 'n' variables with each variable being prime if the corresponding bit of the binary number is a zero and unprime if 1.

x	y	z	Min terms x y z	Designation
0	0	0	$x'y'z'$	M_0
0	0	1	$x'y'z$	M_1
0	1	0	$x'yz'$	M_2
0	1	1	$x'yz$	M_3
1	0	0	$xy'z'$	M_4
1	0	1	$xy'z$	M_5
1	1	0	xyz'	M_6
1	1	1	xyz	M_7

Standard form:

In standard form the terms that form the function may contain one, two or any number of literal/variable. There are two types of standard form:

a) Sum of product [SOP]

b) Product of Sum [POS]

a) Sum of Product [SOP]

SOP is a boolean expression containing terms with 'AND' logic of 1 or more literals. Eg:

$$f = xyz + x'yz + x'y'z$$

b) Product of sum [POS]

The POS is a Boolean expression containing terms

with 'OR' logic of one or more literals. Eg: $(x+y+z)\cdot(x'+y+z)\cdot(x+y'+z')$

Boolean function expressed interms of sop and pos:

ⓐ sop [Min terms]

ⓘ $F = x'y'z' + x'yz + xy'z + xyz'$

$= m_0 + m_3 + m_4 + m_5$

$= \Sigma(0, 3, 4, 5)$

ⓘⓘ $F = A + B'C$

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

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

$= \overset{ABC'+}{ABC} + \overset{AB'C}{A}B'C + AB'C + A'B'C$

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

$= M_7 + M_6 + M_5 + M_4 + M_5 + M_1$

$= M_7 + M_6 + M_5 + M_4 + M_1$

$= \Sigma(7, 6, 5, 4, 1)$

ⓑ pos [Max term]

procedure:

i) Use distributive law

ii) Then, any missing variable in each term is 'OR'ed with $x \cdot x'$ if x is missing.

ⓘ $F = x + yz$

$= (x+y)(x+z)$

$= (x+y+zz')(x+y\cdot y'+z)$

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

$= M_0 \cdot M_1 \cdot M_0 \cdot M_2$

$= M_0 \cdot M_1 \cdot M_2$

$F = \Pi(0, 1, 2)$

(1) $f = xz + yz'$

$= (xz + y)(xz + z')$

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

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

$= (x+y+zz')(xx'+y+z)(z'+yy'+x)$

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

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

$= M_0 * M_1 * M_2 * M_4 * M_1 * M_3$

$= M_0 M_1 M_2 M_3 M_4$

$= \Pi(0, 1, 2, 3, 4)$

Conversion between canonical form:

step 1 : To convert from one canonical form to another interchange the symbol.

i.e. $\Sigma \longrightarrow \Pi$

$\Pi \longrightarrow \Sigma$

step 2 : List those number missing from the original form.

$f = \Sigma(0, 1, 7)$

$f = \Pi(2, 3, 4, 5, 6)$

K-Map [karnaugh map]:

K-map is regarded as a diagrametic or pictorial form of a fourth table. The map is a diagram made up of squares. Each square represents one min term. The map represents a visual diagram of all possible ways of function, may be expressed in a standard form.

Basic k-map:

y \ x	x'	x
	0	1
y' 0	x'y'	xy'
y 1	x'y	xy

Three (3) variable k-map:

There are 8 min terms for 3 binary variables. A map consists of 8 squares. The min terms are arranged not in a binary sequence but in sequence similar to gray code. The characteristics of the sequence is that only one bit is changes from one sequence to another.

x \ yz	y'z'	y'z	yz	yz'
	00	01	11	10
0	x'y'z' (M_0)	x'y'z (M_1)	x'yz (M_3)	x'yz' (M_2)
1	xy'z' (M_4)	xy'z (M_5)	xyz (M_7)	xyz' (M_6)

Simplification

eg: $F_1 = \sum(M_5, M_7) = \sum(5,7)$

$= m_5 + m_7 = xy'z + xyz = xz(y'+y) = xz$

$F_2 = x'yz + x'yz' + xy'z' + xy'z$

x \ yz	00	01	11	10
0	0	0	1	1
1	1	1	0	0

$f_2 = x'y + xy'$

Four variable k-map:

There are 16 min terms for 4 binary variables. A map consists of 16 squares.

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Simplification:

1. One square box represents one min terms, giving a term of four literals
2. Two adjacent square box represents a term of three literals.
3. Four adjacent square Box represents a term of two literals.
4. Eight adjacent square box repreesents a term of one literals.
5. Sixten adjacent square box represents a function equal to one.

Simplify:

$$f = \sum (0, 1, 2, 12, 13, 14, 15)$$

$$= m_0 + m_1 + m_2 + m_{12} + m_{13} + m_{14} + m_{15}$$

WX \ YZ	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	1	1	1	1
10	0	0	0	0

$$f = w'x'y' + w'x'z' + wx$$

Five-variable K-map

AB \ CDE	000	001	011	010	110	111	101	100
00	M_0	M_1	M_3	M_2	M_6	M_7	M_5	M_4
01	M_8	M_9	M_{11}	M_{10}	M_{14}	M_{15}	M_{13}	M_{12}
11	M_{24}	M_{25}	M_{27}	M_{26}	M_{30}	M_{31}	M_{29}	M_{28}
10	M_{16}	M_{17}	M_{19}	M_{18}	M_{22}	M_{23}	M_{21}	M_{20}

Don't care condition:

There are some combination of inputs for which output is not specified and such output does not affect the whole system, which are known as Dan't care condition. The Don't care min terms are denoted by 'x' sign.

Simplify:

$F(w,x,y,z) = \Sigma(1,3,7,11,13)$ and the don't care condition

$d(w,x,y,z) = \Sigma(0,2,5)$

soln: k-map

wx \ yz	00	01	11	10
00	X	1	1	X
01		X	1	
11		1		
10			1	

$F = w'x' + w'z + xy'z + x'yz$

8. Simplify the following expression using k-map and implement using NAND gate only.

$F(w,x,y,z) = \Sigma(1,3,7,11,15)$

$d(w,x,y,z) = \Sigma(0,2,5)$

soln: Here, $f(w,x,y,z) = m_1 + m_3 + m_7 + m_{11} + m_{15}$

$d(w,x,y,z) = m_0 + m_2 + m_5$

k-map

wx \ yz	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$F =$ [redacted]

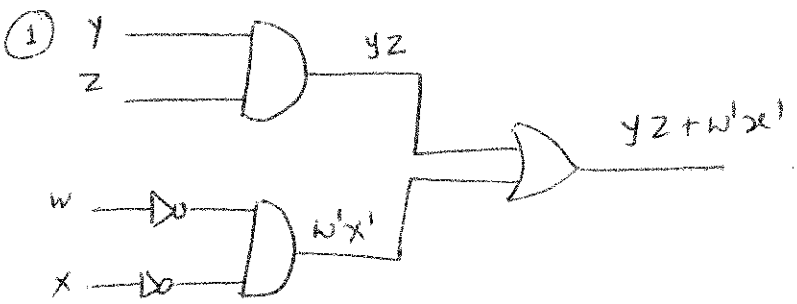
$= yz + w'x'$

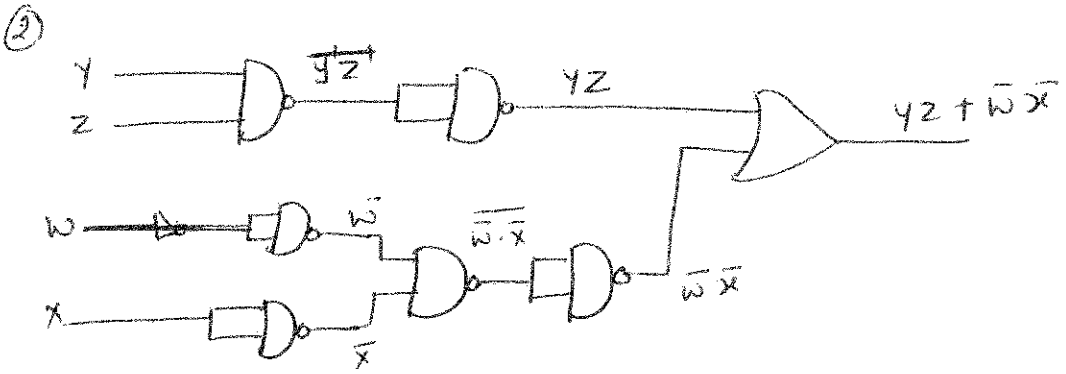
Implementation using NAND gate only.

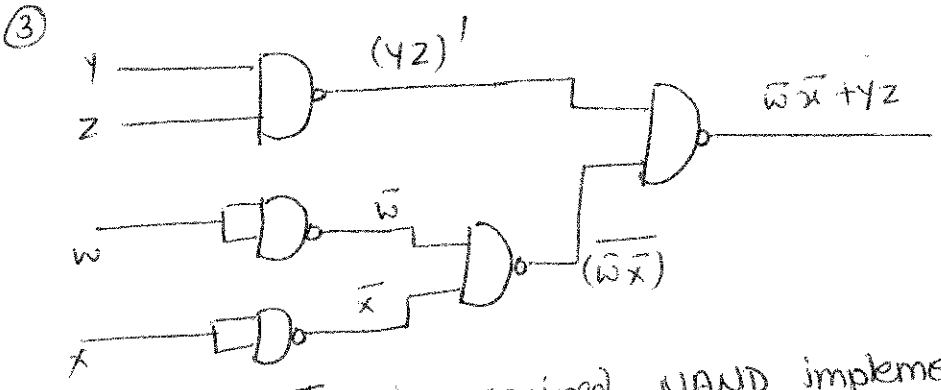
NOTE: Take double inversion.

$$F = \overline{\overline{\bar{w}\bar{x} + yz}}$$
$$= \overline{\bar{w}\bar{x} \cdot \overline{yz}}$$

step 1







This is required NAND implementation.

PU 2015 A Boolean function given by $f(A, B, C, D) = \Sigma(3, 4, 6, 8, 10, 12, 14)$ and don't care condition $d(A, B, C, D) = \Sigma(0, 2)$. Simplify it using K-map and implement using NAND gate only.

Soln: Here, $f(A, B, C, D) = \Sigma(3, 4, 6, 8, 10, 12, 14)$

$d(A, B, C, D) = \Sigma(0, 2)$

K-map

AB \ CD	00	01	11	10
00	X	0	1	X
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$F(A, B, C, D) = C'D' + AB'C$

$D' + A'B'C$

NOW, To implement using NAND gate only.

Taking double inversion,

$$f(A,B,C,D) = \overline{\overline{D' + A'B'C}}$$

$$= \overline{\overline{D} \cdot \overline{\bar{A}\bar{B}C}}$$

$$= \overline{D \cdot \overline{\bar{A}\bar{B}C}}$$

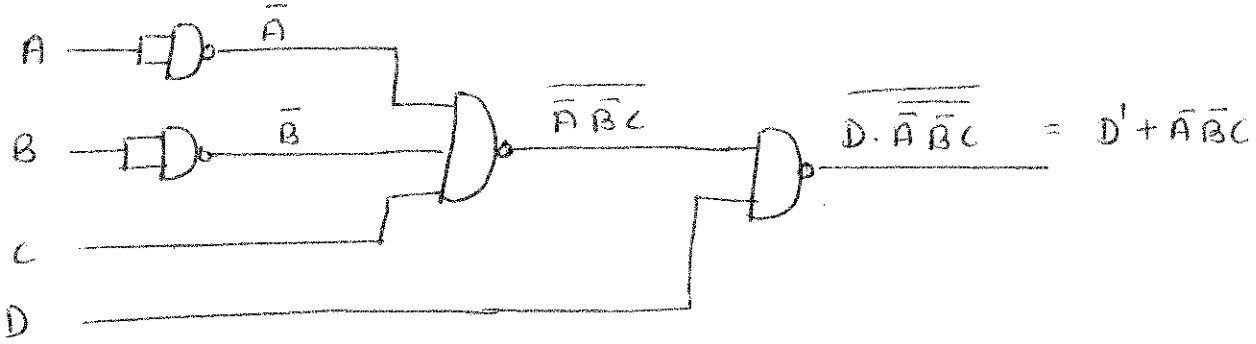


Fig: NAND implementation.

K-map for product of sum Design (as SOP)

Product of sums design uses the same principles, but applied to the zeros of the function.

Example $f(x,y,z) = \Sigma(1,4,8)$

$= \Pi(0,2,3,5,7)$

x \ yz	y+z 0 0	y+z' 0 1	y'+z' 1 1	y'+z 1 0
x 0	0	1	0	0
x' 1	1	0	0	1

$f(x,y,z) = (x+y')(x'+z')(x+z)$

Basically, the digital circuits are one of the following types:

i) Combinational circuits

ii) Sequential circuits

Combinational circuits:

A combinational circuit may be defined as a logic circuit the output of which depends only upon the combination of inputs. The output does not depend on the past value of inputs or outputs. Therefore, combinational circuits do not need any memory.

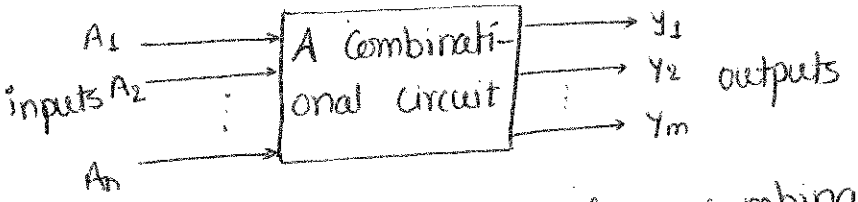


Fig: Block diagram of a combinational circuit

A combinational circuit operates in following three steps:

i) It accepts n-different inputs

ii) The combination of gates operates on the inputs

iii) m different outputs are produced as per requirement.

Sequential circuits:

The output of a sequential circuit depends upon the present time inputs, the previous output and the sequence in which the inputs are applied.

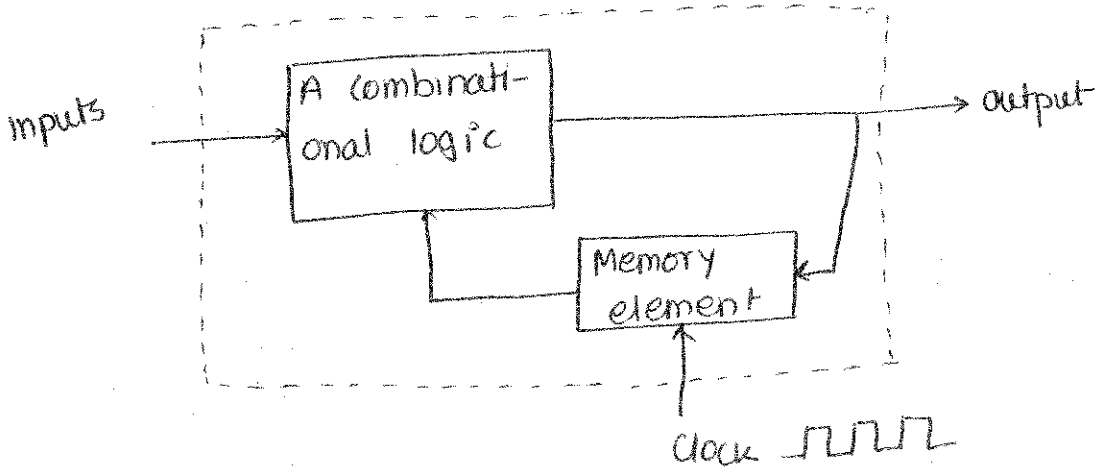


Fig: Block diagram of a sequential circuit

In order to provide the previous input or output, we require a memory element. Therefore, a sequential circuit requires a memory element.

Combinational logic design procedure:

The various steps involved in designing procedure of a combinational logic may be listed as under:

i) We will be given a problem / Problem is stated.
ii) Determine the number of inputs and outputs and assign letter symbols to input and output variables.
iii) The truth table that defines inputs and outputs relationship is derived.
iv) Draw K-map for each output and obtain the simplified Boolean expression for each output.
v) Lastly, draw the logic diagram [combinational circuit].

A practicle design methods would have to consider certain constraints as:

1. Minimum number of gates
2. Minimum number of inputs to gate
3. Minimum propagation time of the signal through the circuit
4. Minimum number of interconnection.

Example: A circuit has four inputs and two outputs. One of the outputs is high when majority of inputs are high. The second output is high only when all inputs are of same type. Design the combinational circuit.

Sol^n: Let the four inputs be A, B, C, D and the two outputs be Y_1 and Y_2.

Decimal	Inputs A	B	C	D	outputs Y_1	Y_2
0	0	0	0	0	0	1
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	0	0	1	1	0	0
4	0	1	0	0	0	0
5	0	1	0	1	0	0
6	0	1	1	0	0	0
7	0	1	1	1	1	0
8	1	0	0	0	0	0
9	1	0	0	1	0	0
10	1	0	1	0	0	0
11	1	0	1	1	1	0
12	1	1	0	0	0	0
13	1	1	0	1	1	0
14	1	1	1	0	1	0
15	1	1	1	1	1	1

For output Y_1

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

$Y_1 = ABD + BCD + ACD + ABC$

For output Y_2

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

$Y_2 = A'B'C'D' + ABCD$

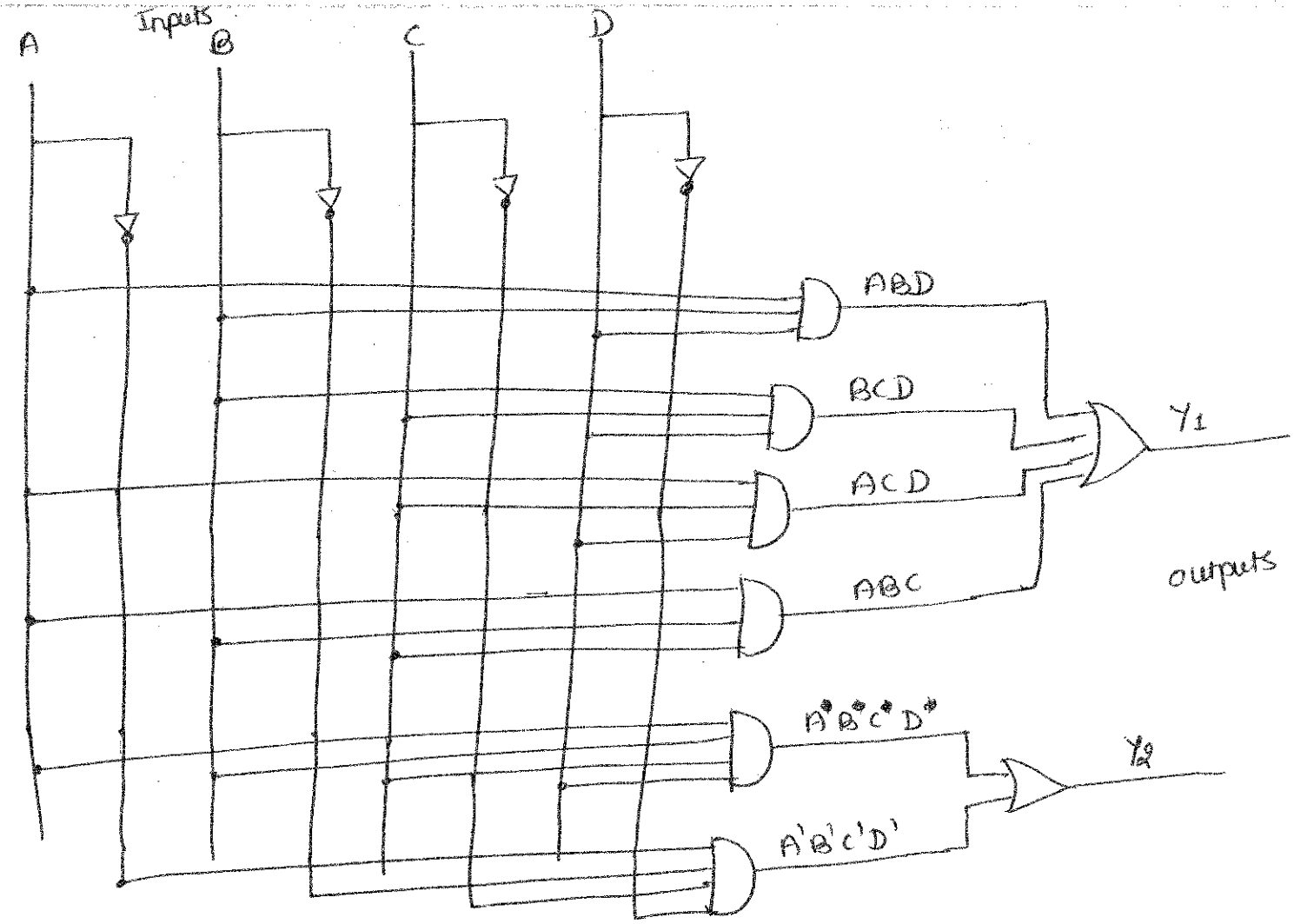


fig: Illustration of Logic diagram

* Binary adders may be of two types:
  1) Half adder
  2) Full adder

1. Half adder

Half adder is a combinational logic circuit with two inputs and two outputs. It is the basic building block for addition of two single bit numbers. This circuit has two outputs namely carry and sum.

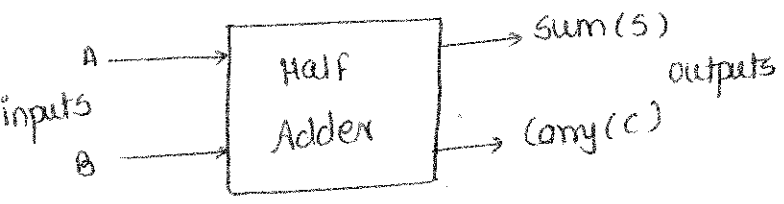


fig: Block diagram

Inputs		outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

fig: truth table

K-map for sum

A \ B	0	1
0	0	[1]
1	[1]	0

K-map for carry

A \ B	0	1
0	0	0
1	0	[1]

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

Carry = AB

Combinational circuit is:

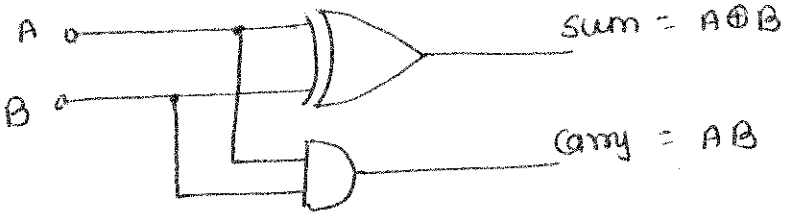


Fig: An Half adder circuit

Half adder using basic gates:

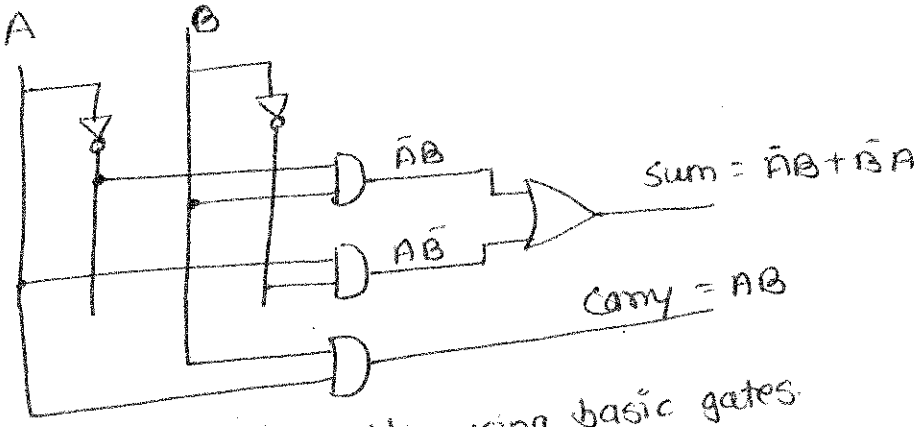


Fig: An Half adder using basic gates

Limitation:

- The addition of three bits is not possible to perform by using an half adder circuit.

Full adder

To overcome the drawback of an Half adder circuit, we develop a 3 single bit adder circuit called Full Adder. Basically, a full adder is a three input and two output combinational circuit.

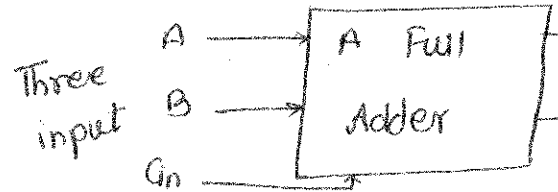


Fig: Block diagram

Inputs			Outputs	
A	B	C_{in}	S	C_o
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

K-map for sum (S)

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

$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$

$= A \oplus B \oplus C_{in}$

K-map for carry out (C_o)

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

$C_o = AC_{in} + BC_{in} + AB$

A logic diagram for full adder:

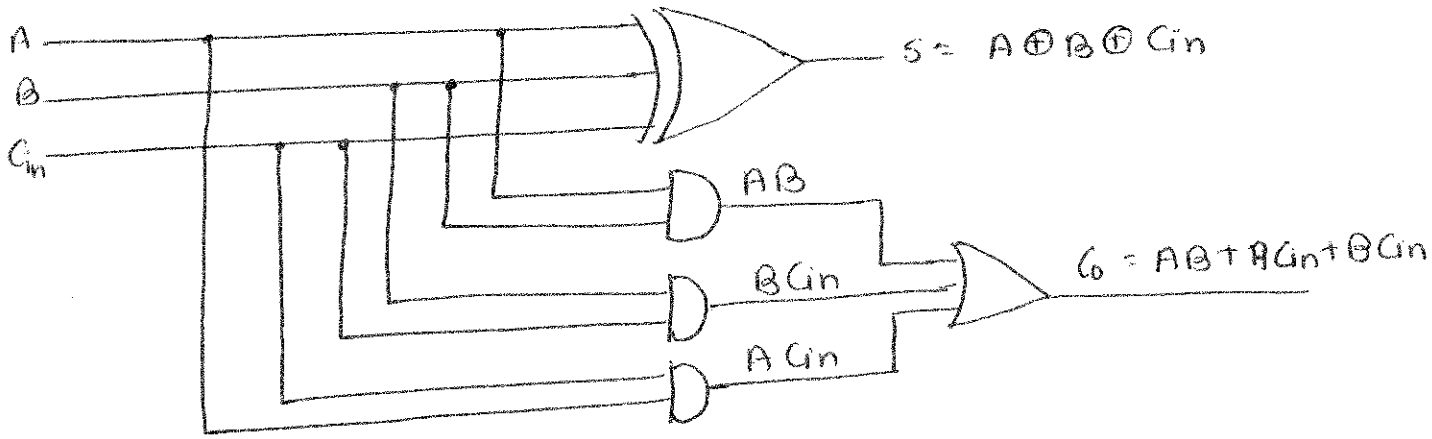


Fig: A full adder circuit

Full adder using Half adders:

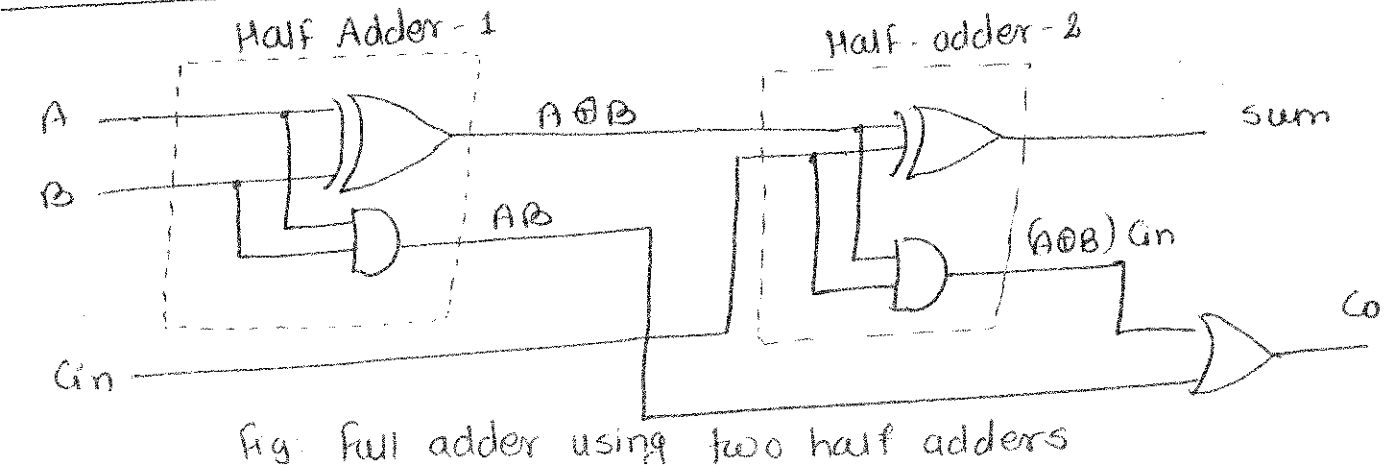


Fig: Full adder using two half adders

PROOF:

Here, Sum $= (A \oplus B) \oplus C_{in}$

$= A \oplus B \oplus C_{in}$

This is same as that obtained for a full adder.

NOW, $C_0 = (A \oplus B) C_n + AB = (A'B + AB') C_n + AB$

$= A'BC_n + AB'C_n + AB = A'BC_n + AB'C_n + AB(1 + C_n)$

$= A'BC_n + AB'C_n + AB + ABC_n = BC_n(A' + A) + AB'C_n + AB$

$= BC_n + AB'C_n + AB(1 + C_n) = BC_n + AB'C_n + AB + ABC_n$

$= BC_n + AB + AC_n(B' + B)$

$= AB + BC_n + AC_n$

This is same as that obtained for a full adder.

Application:

A full adder acts as the basic building block of the 4 bit/8 bit binary / BCD adder ICs such as 7483.

Binary subtractors may be classified as:

1) An Half subtractor
2) A full subtractor

An Half Subtractor

Half subtractor may be defined as a combinational circuit with two inputs and two outputs (i.e., difference and borrow). In subtraction (A-B), A is called as minuend bit and B is called as subtrahend bit.

Truth table

	Inputs		outputs	
S.N.	A	B	Difference D (A-B)	Borrow B_0
1.	0	0	0	0
2.	0	1	1	1
3.	1	0	1	0
4.	1	1	0	0

K-map for Difference (D)

A \ B	0	1
0	0	1
1	1	0

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

K-map for borrow o/p

A \ B	0	1
0	0	1
1	0	0

$Borrow = \bar{A}B$

Logic diagram

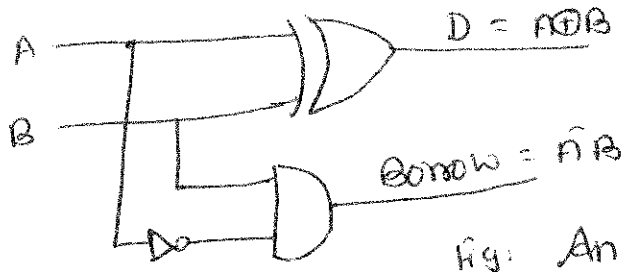


Fig: An half subtractor circuit.

Using Basic gate:

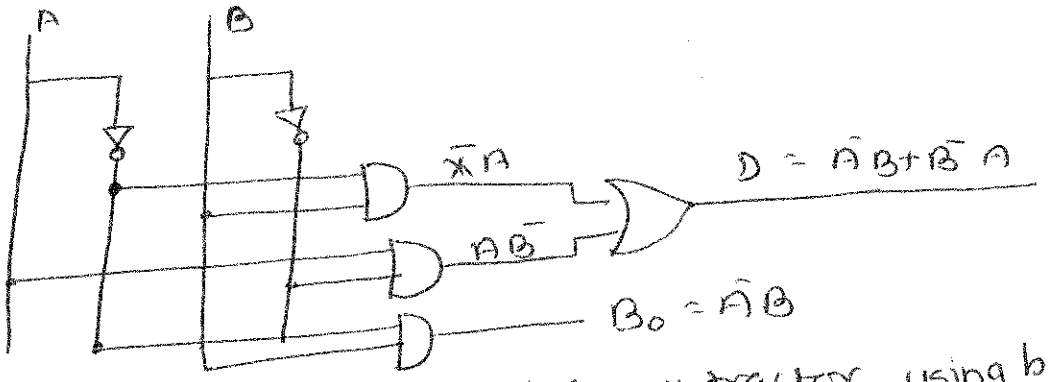


Fig: Half ~~subtracting~~ subtractor using basic gates

Full subtractor:

A full subtractor is a combinational circuit with three inputs A, B and B_{in} and two outputs D and B_o. Here, A is the minuend, B is subtrahend, B_{in} is the borrow produced by the previous stage, D is the difference output and B_o is the borrow output.

Truth Table

Inputs			Outputs	
A (minuend)	B (subtrahend)	B_{in} (Previous borrow)	D = A − B − B_{in}	B_o
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

For difference output

A \ BBin	00	01	11	10
0	0	[1]	0	[1]
1	[1]	0	[1]	0

For Borrow output

A \ BBin	00	01	11	10
0	0	[1]	[1]	[1]
1	0	0	[1]	0

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

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

$= B_{in}(\overline{A \oplus B}) + \bar{B}_{in}(A \oplus B)$

$= A \oplus B \oplus B_{in}$

$B_0 = \bar{A}B_{in} + \bar{A}B + BB_{in}$

Logic diagram for a full subtractor:

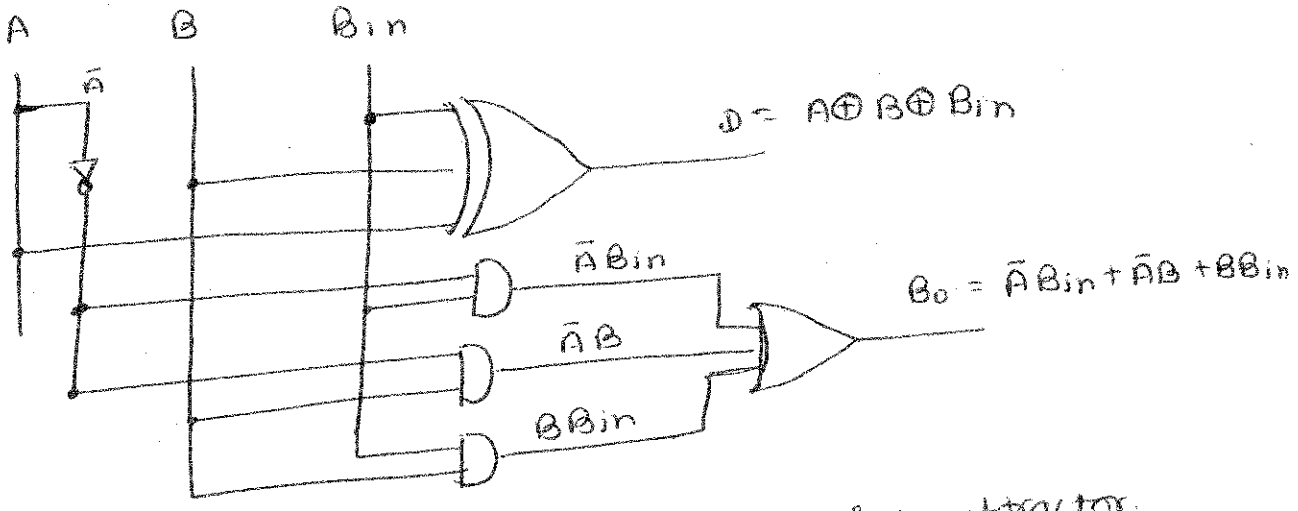


Fig Illustration of Logic diagram for a full subtractor

Construction of full subtractor using Half subtractors

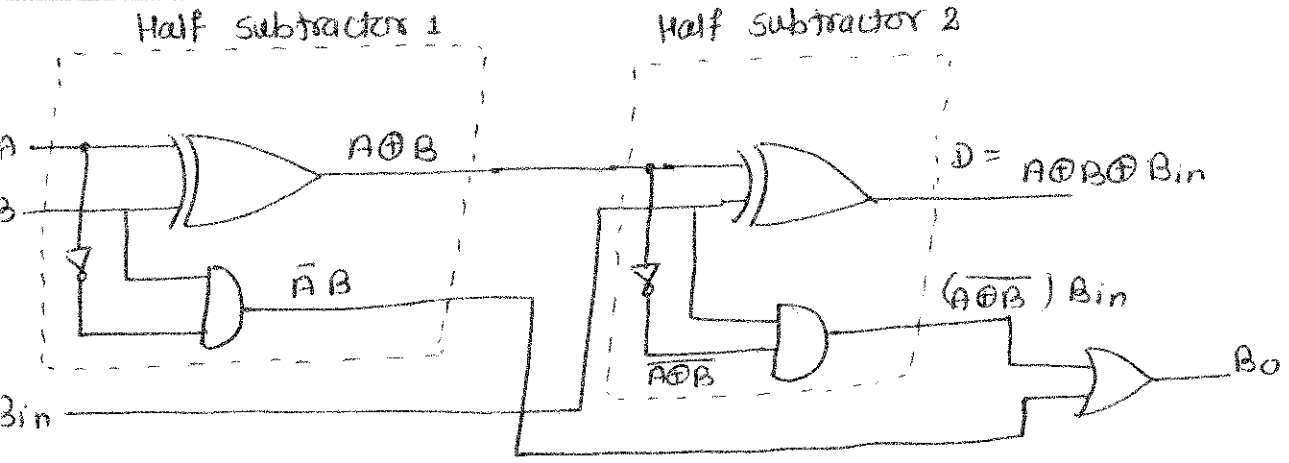


Fig: Construction of a full subtractor using half subtractors

$D = A \oplus B \oplus B_{in}$

$B_o = (\overline{A \oplus B})B_{in} + \bar{A}B = (\overline{\bar{A}B + A\bar{B}})B_{in} + \bar{A}B$

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

$= \bar{A}\bar{B}B_{in} + ABB_{in} + \bar{A}B(1 + B_{in})$

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

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

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

$= \bar{A}\bar{B}B_{in} + BB_{in} + \bar{A}B(1 + B_{in})$

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

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

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

Analysis Procedure:

The design of a combinational circuit starts from the verbal specification of a required function and culminates with a set of output Boolean functions or a logic diagram. The analysis of a combinational circuit is somewhat the reverse process. It starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or a verbal explanation of the circuit operation.

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label with arbitrary symbols all gate outputs that are a function of the input variables. Obtain the Boolean functions for each gate.
2. Label with other arbitrary symbols those gates which are a function of input variables and/or previously labeled gates. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables only.

Example:

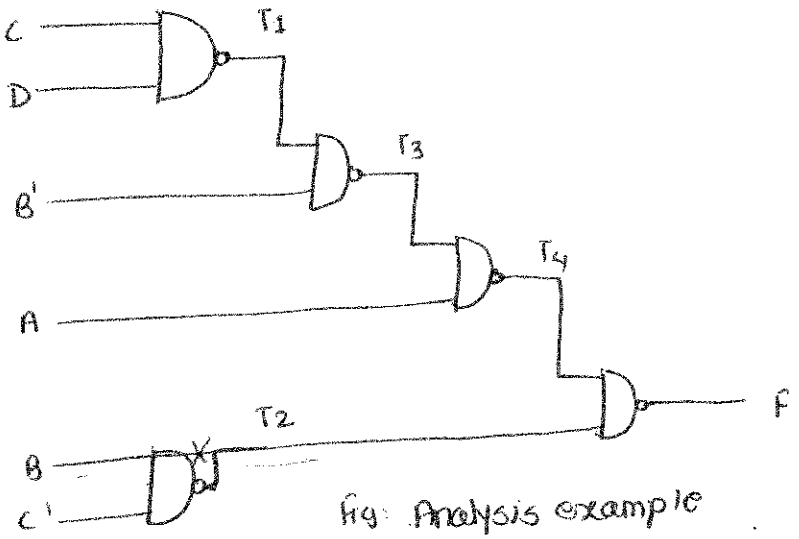


fig: Analysis example

$T_1 = (\overline{CD}) = \bar{C} + \bar{D}$

$T_2 = (\overline{B\bar{C}}) = \bar{B} + C$

$T_3 = (\overline{T_1 \cdot \bar{B}}) = \overline{(\bar{C}+\bar{D})\bar{B}} = \overline{B\bar{C} + B\bar{D}} = \overline{B\bar{C}} \cdot \overline{B\bar{D}} = (B+C)(B+D)$

$= B + CD$

$T_4 = (\overline{T_3 \cdot A}) = \overline{(B+CD)\cdot A} = \overline{AB + ACD}$

$F = \overline{T_4 \cdot T_2} = \overline{\overline{AB+ACD}\cdot(\bar{B}+C)}$

$= \overline{\overline{AB}\cdot\overline{ACD}\cdot(\bar{B}+C)}$

$= \overline{AB} + \overline{ACD} + (\overline{\bar{B}+C})$

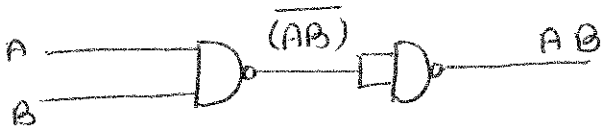
$= AB + ACD + \bar{B}\cdot\bar{C}$

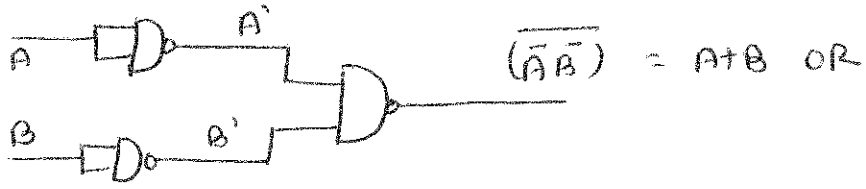
$= AB + ACD + B\bar{C}$

Multilevel NAND circuits:

Process to obtain multilevel NAND diagram from Boolean expression:

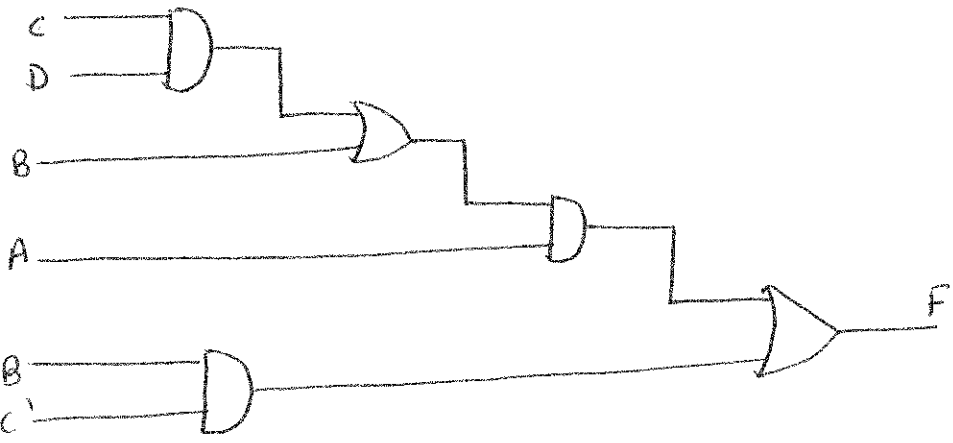
1. From the given algebric expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal and complement inputs are available.
2. Draw a second logic diagram with the equivalent NAND logic.
3. Remove any two cascaded inverters from the diagram, since double inversion does not perform a logic function. Remove inverters connected to single external inputs and complement the corresponding input variable. The new logic diagram obtained is the required NAND gate implementation.



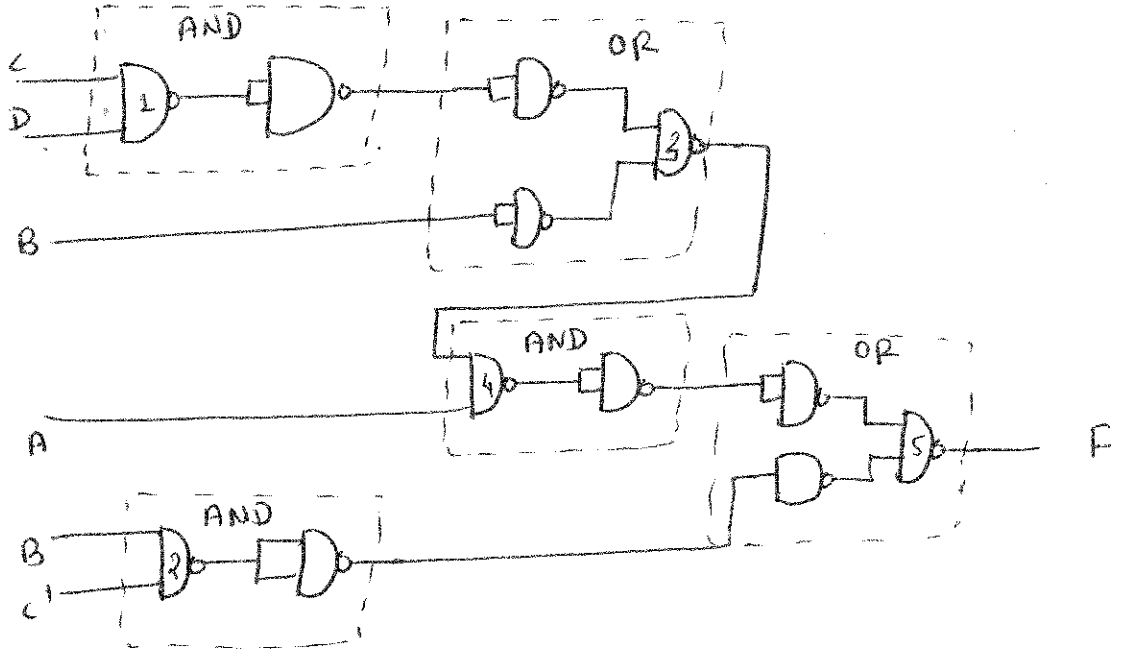


Example:

$F = A(B+CD)+BC'$



a) AND/OR implementation



b) Substituting equivalent NAND function

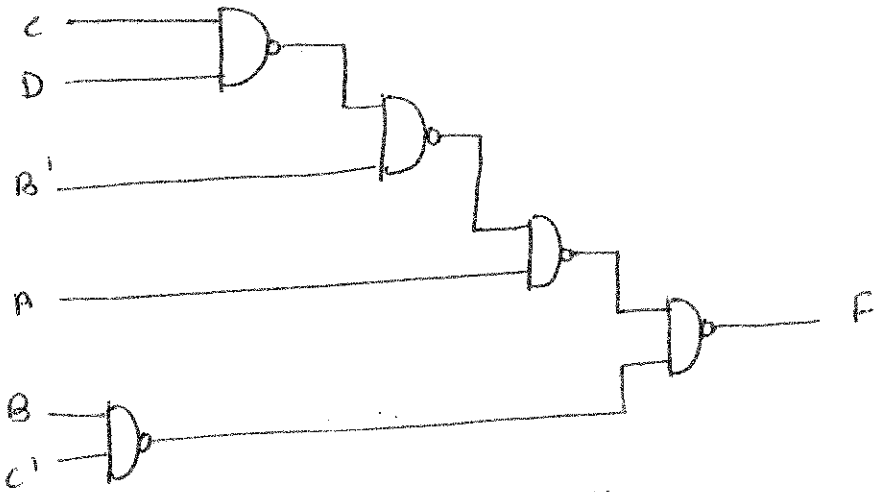


Fig: NAND implementation

Fig: Implementation of $F = A(B + CD) + BC'$ with NAND gates

Multilevel NOR Circuits:

Process to obtain multilevel NOR diagram from Boolean expression.

1. Draw the AND-OR logic diagram from the given algebric expression. Assume that both the normal and the complement inputs are available.
2. Draw a second logic diagram with equivalent NOR logic. substituted for each AND, OR and NOT gate.
3. Remove pairs of cascaded inverters from the diagram. Remove inverters connected to single external inputs and complement the corresponding

input variable

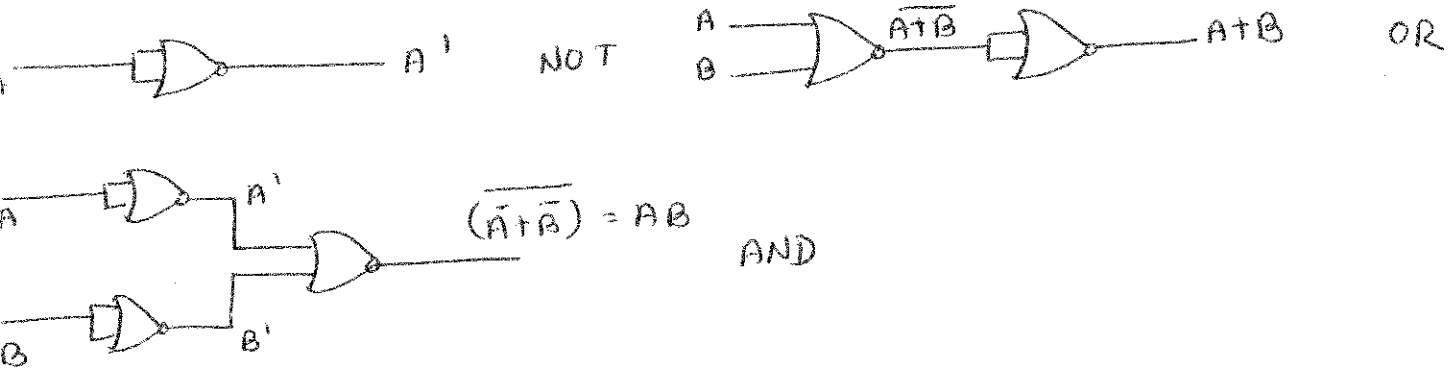
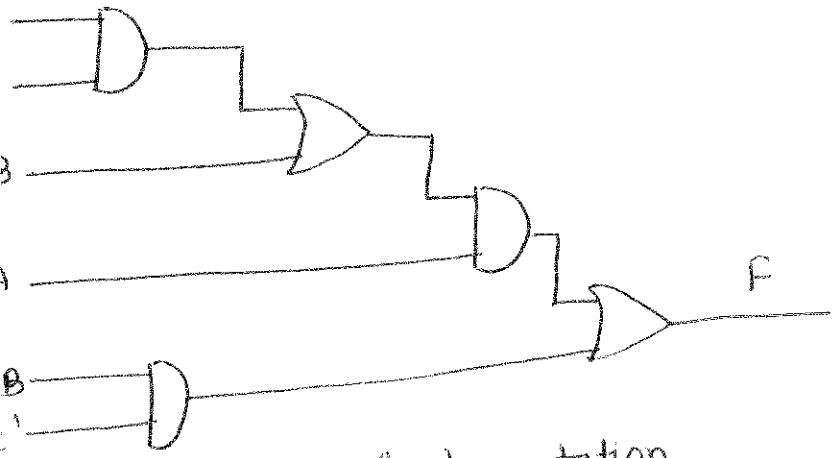


fig: Implementation of NOT, OR and AND by NOR gates.

Example

$F = A(B+(D)+BC'$



a) AND/OR implementation

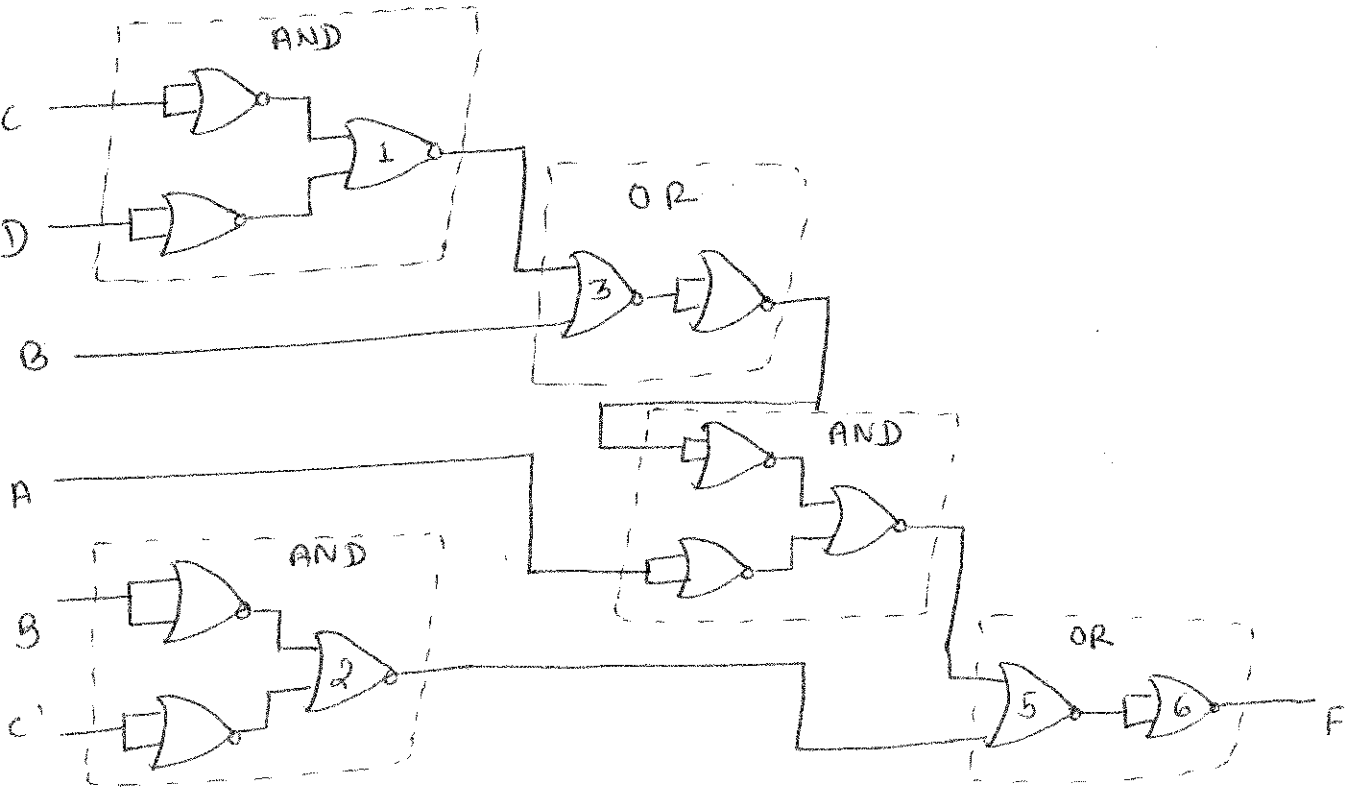


fig: Substituting equivalent NOR functions

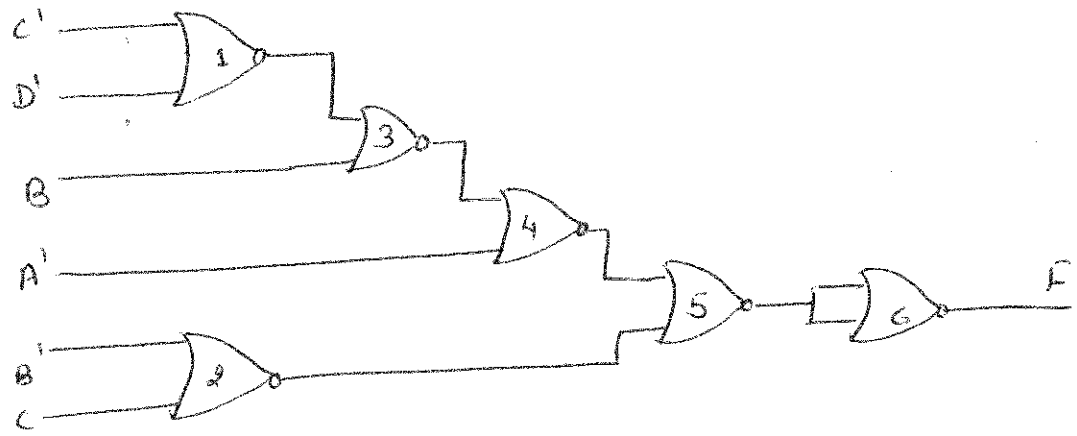


fig: NOR implementation.

Parity Generation and checking: (M. Morris Mano 1us, 1us)

Parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond to the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator; the circuit that checks the parity in the receiver is called a parity checker.

1) Parity Generator

Even parity generator [3 bit]

Truth table

x	y	z	Pg
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

K-map for Pg

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$P_g = x'y'z + x'yz' + xy'z' + xyz$

$= x \oplus y \oplus z$

$= x'(y'z + yz') + x(y'z' + yz)$

$= x'(y \oplus z) + x(\overline{y \oplus z})$

$= x \oplus y \oplus z$

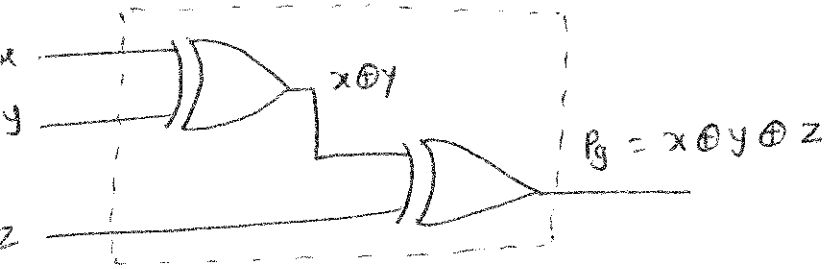


Fig: Combinational Logic for even parity generator

Parity checker:

Even parity checker [3 bit]

0 = True = no error (even parity)
1 = false = error (odd parity)

x	y	z	P_g	P_c
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

K-map for P_c

xy \ zP_g	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$P_c = x'y'z'P_g + x'y'zP_g + x'yz'P_g' + x'yzP_g + xyz'P_g + xyzP_g' + xy'z'P_g' + xy'zP_g$

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

$= P_g(\overline{x \oplus y \oplus z}) + P_g'(x \oplus y \oplus z)$

$= x \oplus y \oplus z \oplus P_g$

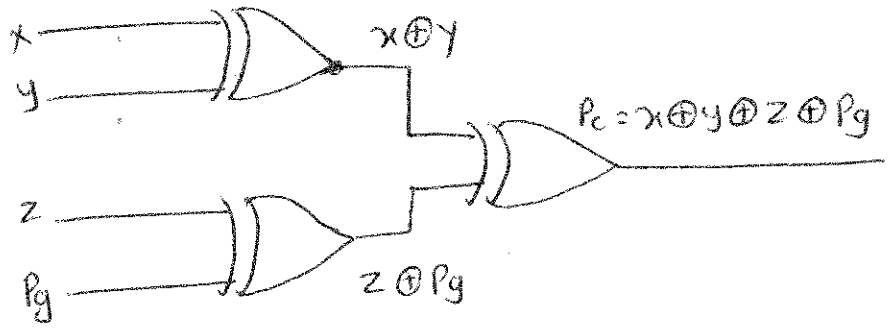


Fig: Combinational Logic for even parity checker

Code Conversion:

1) Binary to excess-3 code:

dec	Binary				excess-3 code			
	W	X	Y	Z	A	B	C	D
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 - 15 Don't care

K-map for A

WX \ YZ	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

A = W + XZ + XY

For B k-map

wx \ yz	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$B = x'y + x'z + xy'z'$

For C k-map

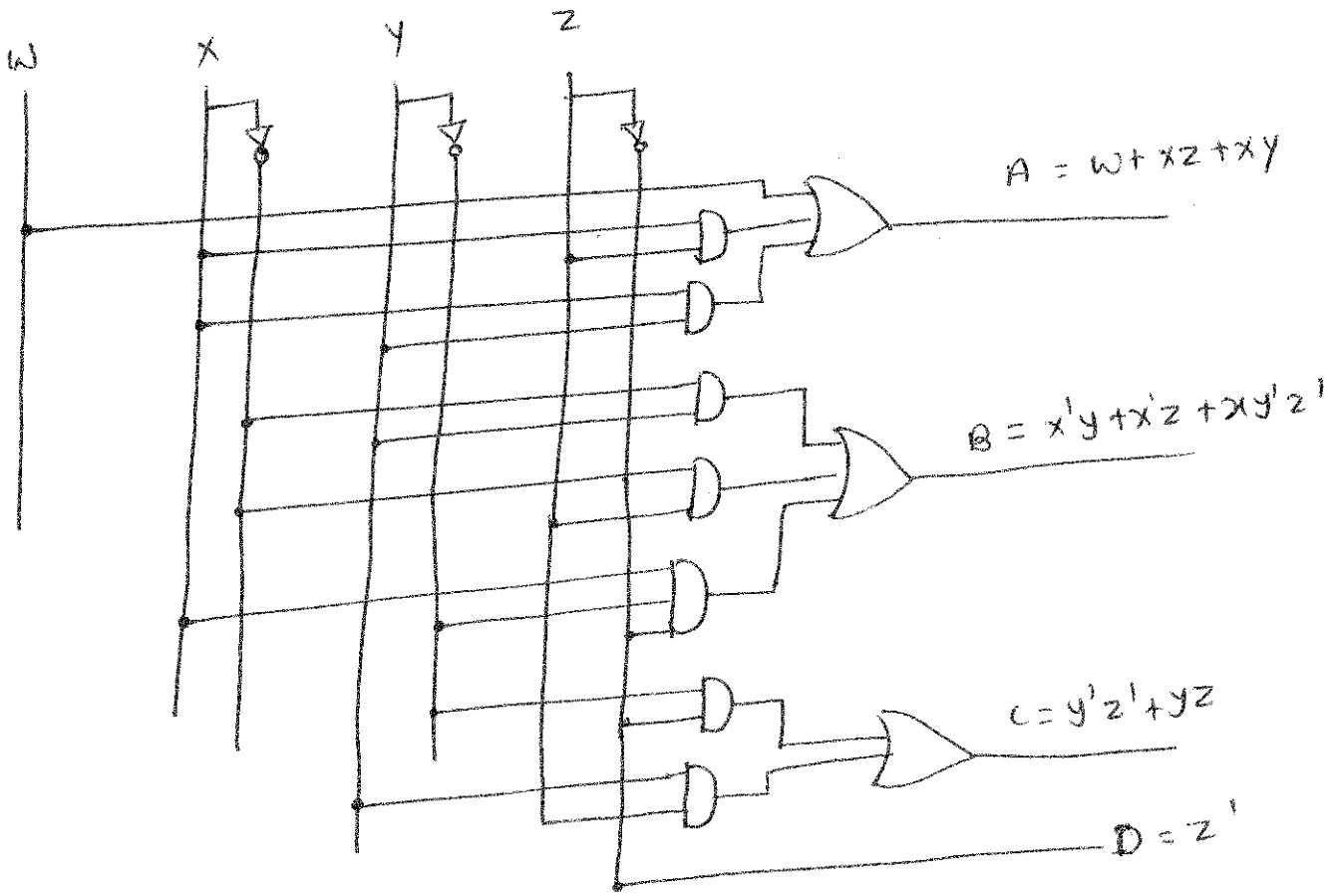
wx \ yz	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$C = y'z' + yz$

For D k-map

wx \ yz	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

$D = z'$



2) Binary to Gray Conversion:

Dec	Binary				Gray			
	W	X	Y	Z	A	B	C	D
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	1
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

K-map

For A

WX \ YZ	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

A = W

For B

WX \ YZ	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

B = $W'X + WX' = W \oplus X$

For C

wx \ yz	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$C = xy' + x'y$

$= x \oplus y$

For D

wx \ yz	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$D = y'z + yz'$

$= y \oplus z$

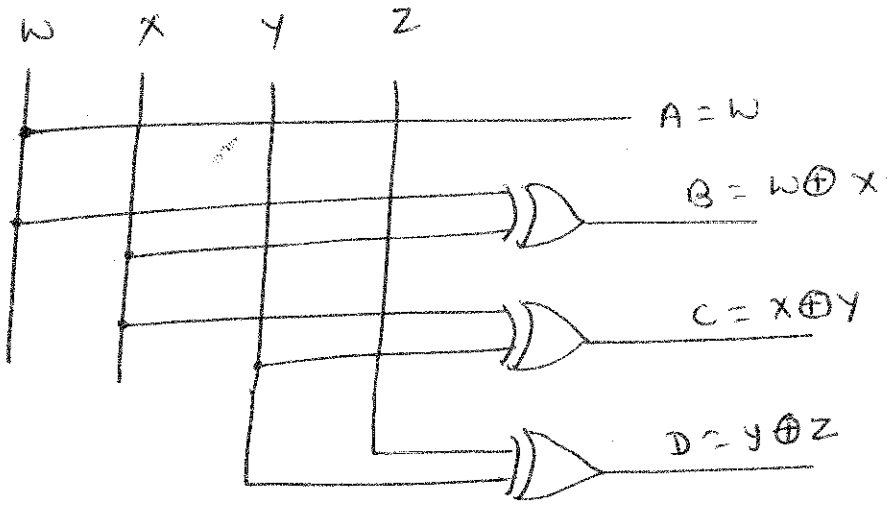


Fig: Combinational logic circuit to convert binary code to gray code.

Q. A circuit has four inputs and two outputs. One of the outputs is high when majority of inputs are high. The second output is high only when all inputs are of same type. Design the combinational circuit.

[$Y_1 = ABC + BCD + ABD + ACD$, $Y_2 = \bar{A}\bar{B}\bar{C}\bar{D} + ABCD$]

Q. Design a combinational circuit whose input is a four bit number and at the output, we obtain the two's complement of the input.

[$B_3 = \bar{A}_3(A_0 + A_1) + \bar{A}_1\bar{A}_0(A_3 \oplus A_2)$, $B_2 = A_2 \oplus (A_0 + A_1)$, $B_1 = A_1 \oplus A_0$, $B_0 = A_0$]

NOTE:

Q. use K-map to simplify the given Boolean function in POS and implement the simplified function using NOR gate only.

$F(A,B,C,D) = \Sigma(1,3,8,9,12,13,14,15)$ and don't care, $d(A,B,C,D) = \Sigma(2,7,10)$

soln: In POS form

$F(A,B,C,D) = \Pi(0,2,4,5,6,) \Sigma(1,3,8,9,12,13,14,15)$

$d(A,B,C,D) = \Sigma(2,7,10)$

K-map

AB \ CD	C+D 00	C+D' 01	C'+D' 11	C'+D 10
A+B 00	0	1	1	X
A+B' 01	0	0	X	0
A'+B' 11	1	1	1	1
A'+B 10	1	1	0	X

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

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

for NOR implementation,

$f(A,B,C,D) = \overline{(A+C+D)(A+B'+C)(A+B'+D)(A'+B+C'+D')}$

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

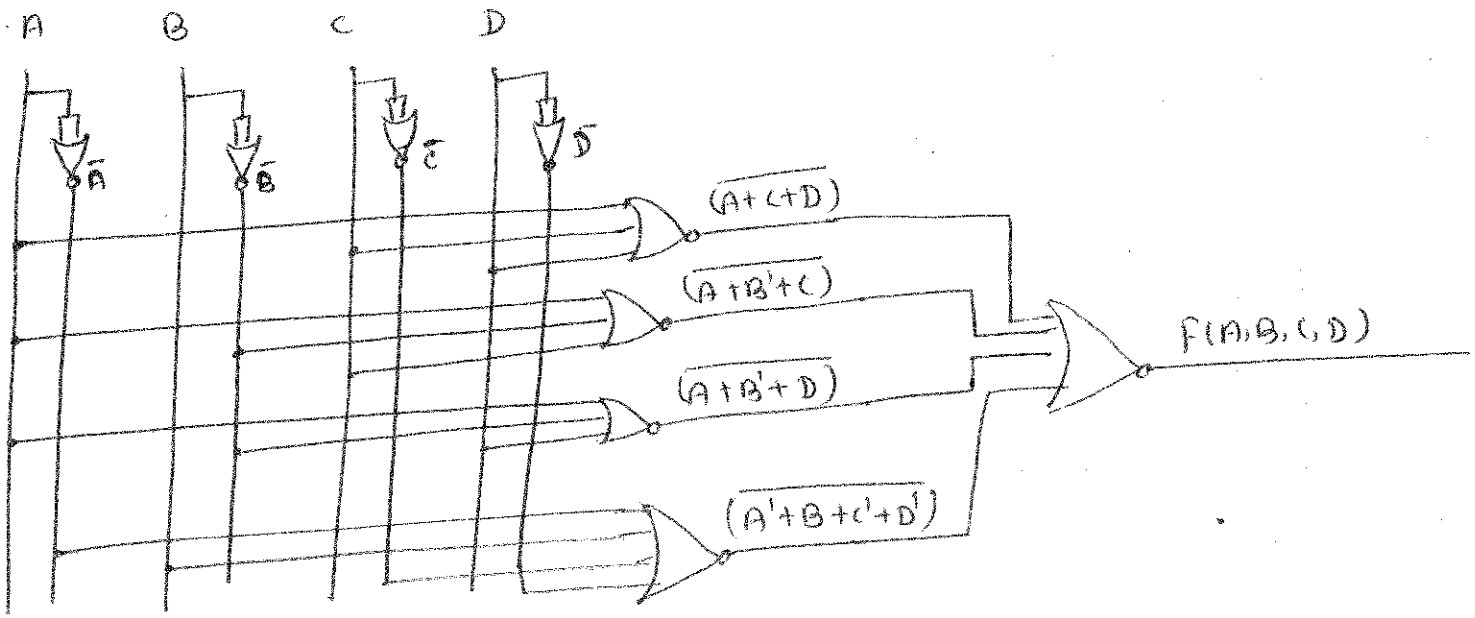


Fig: NOR gate implementation.

BCD Addition:

We add two BCD numbers A and B

- Sum ≤ 9, carry = 0 → Answer is correct. No correction is required.
- Sum ≤ 9, carry = 1 → we add 6 to the sum to get correct answer
- Sum > 9, carry = 0 → We add 6 to the sum to get the correct answer

Example Add $(569)_{10}$ and $(687)_{10}$ in BCD

soln:

				1 1 1
569	0101	0110		1001
+ 687	0110	1000		0111
1256	1011	1111	1	0000
	invalid BCD	invalid BCD		valid BCD with carry 1

Incorrect answer

We add $(0110)_2$ to only the invalid BCD numbers to get correct answer.

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

0001 0010 0101 0110 = $(1256)_{10}$ Correct answer.

Therefore, $(569)_{10} + (687)_{10} = (1256)_{10}$

BCD subtraction

① using 9's complement

Example1:

perform $(83)_{10} - (21)_{10}$ using the 9's complement method

soln: 9's complement of 21 is, 78

Now, $(83)_{10}$	1000	0011		
$+(78)_{10}$	0111	1000		
$(161)_{10}$	1111	1011		← Result is in invalid BCD form
	invalid BCD			

$\overline{78}$ ← 10's complement of 22.

ii) Next, we add $(54)_{10}$ and 10's complement of $(22)_{10}$.

$(54)_{10}$ 0 1 0 1 0 1 0 0

10's complement of $(22)_{10}$ 0 1 1 1 1 0 0 0

carry: 1 1 1

1 1 0 0 1 1 0 0

↳ Invalid BCD ↵

iii) Then, we add $(0110)_2$

1 1 0 0 1 1 0 0

carry: + 0 1 1 0 0 1 1 0

1 1 1

⊠ 0 0 1 1 0 0 1 0

3 2

Discard final carry. Answer is positive and is in true BCD form.

Therefore, $(54)_{10} - (22)_{10} = (32)_{10}$.

Now, add $(0110)_2$ to each invalid BCD number.

```
  1 1 1                    1 0 1 1
  1 1 1 1                  0 1 1 0
  0 1 1 0                  1 0 0 0 1
 ---------
 1 0 1 1 0
```

↳ final carry is 1 which indicates that the result is positive and is in its true form.

Now, Result is, $0110 \quad 0001$

$+ \quad 1$

$0110 \quad 0010$

6 2

final answer is positive and is in its true form

Therefore, $(83)_{10} - (21)_{10} = (62)_{10}$

Example2: Perform $(52)_{10} - (89)_{10}$ using 9's complement.

soln:

i) First, we obtain 9's complement of $(89)_{10}$.

9's complement of $(89)_{10} = 99 - 89 = 10$

ii) Then, we add $(52)_{10}$ and 9's complement of $(89)_{10}$.

Thus,

```
   0 1 0 1   0 0 1 0     BCD of (52)10
 + 0 0 0 1   0 0 0 0     9's complement of (89)10
Carry:   1
---------------------
Sum: 0 1 1 0   0 0 1 0   final carry is 0, so the sum is -ve.
```

iii) Next, we take 9's complement of sum.

```
   1 0 0 1   1 0 0 1     BCD of 99 → [To find 9's complement subtract sum from 99].
 - 0 1 1 0   0 0 1 0
---------------------
   0 0 1 1   0 1 1 1     final answer
     3         7
```

Therefore, $(52)_{10} - (89)_{10} = -(37)_{10}$

(ii) using 10's complement

Example3:

perform $(54)_{10} - (22)_{10}$ in BCD using 10's complement

soln:

i) First, let us obtain 10's complement of 22.

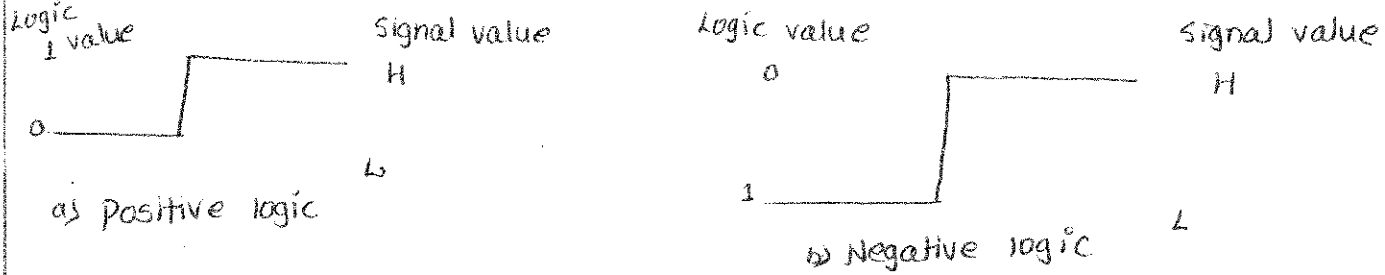
9's complement of 22, $99 - 22 = 77$

$+ \quad 1$

positive and negative logic system:

We designate the higher level by H and the lower level by L.

There are two choices for logic-value assignment.



(a) positive logic (b) Negative logic

Choosing the high-level H to represent logic 1, as shown in fig(a). defines a positive-logic system. Choosing the low-level L to represent logic-1, as shown in fig(b), defines a negative-logic system.

It is not signal polarity that determines the type of logic, but rather the assignment of logic values according to the relative amplitudes of the signals.

Digital circuits are easier to design than analog circuit:

Digital systems are quite easy to design than analog systems, because digital design involves logic design which does not require special maths skills and its behaviour may be visualized part-by-part.

To visualize the behaviour of analog systems is quite difficult. Actually, it requires the special insights about the analog components such as capacitors, inductors and transistors used in the analog system.

some other factors:

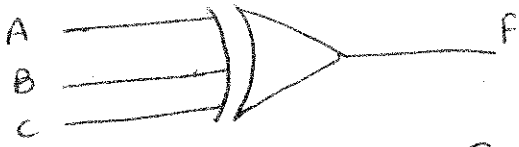
- Reliability and Reproducibility of Results
- flexibility
- functionality
- Programmability
- Device speed
- Economy in manufacturing

-upgrading technology

Factors to be considered while constructing the Logic gates:

1. Feasibility and economy of producing the gate with physical components.
2. The possibility of extending the gate to more than two inputs.
3. The basic properties of the binary operator such as commutativity and associativity
4. The ability of the gate to implement Boolean functions alone or in conjuction with other gates.

Three input X-OR



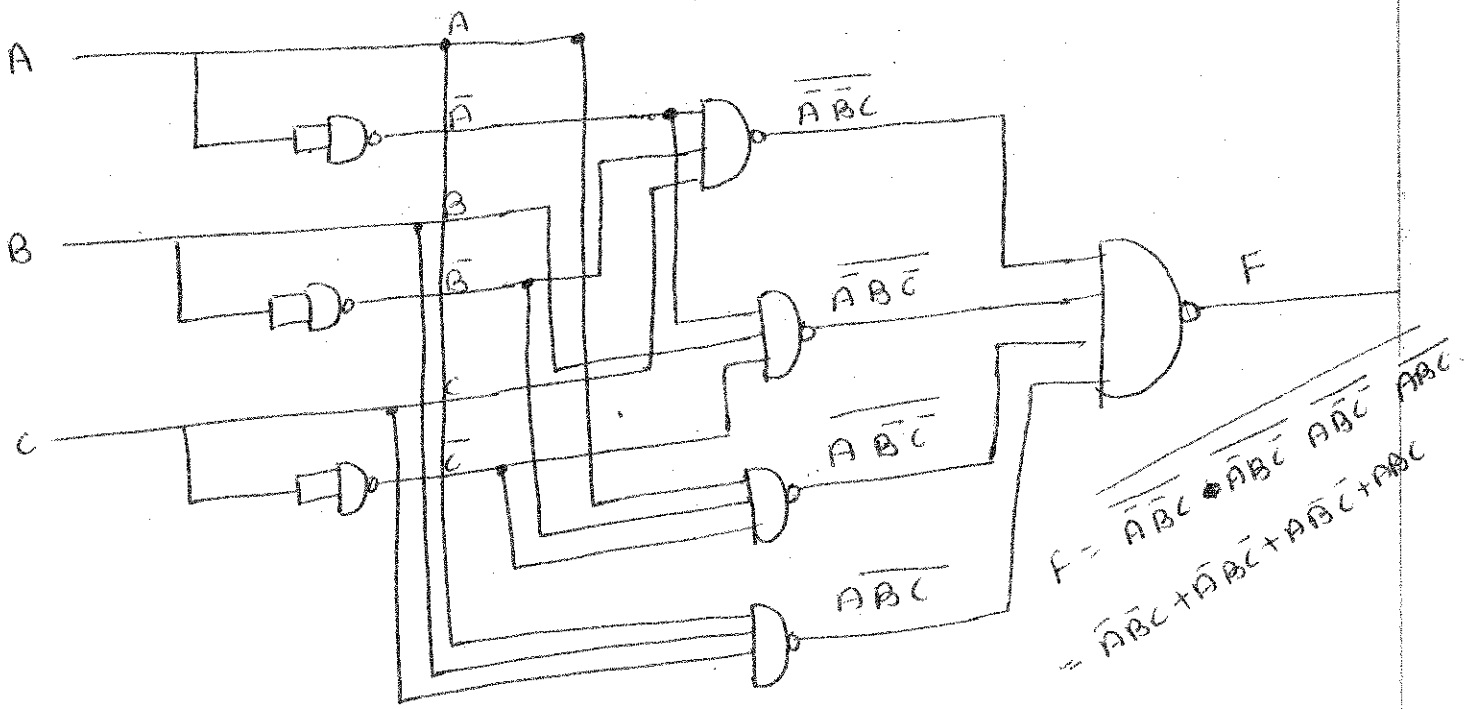
$$F = \bar{A}B\bar{C} + \bar{A}\bar{B}C + AB\bar{C} + ABC$$

To implement using NAND gate

Take double inversion:

$$F = \overline{\bar{A}B\bar{C} + \bar{A}\bar{B}C + AB\bar{C} + ABC}$$

$$= \overline{\bar{A}B\bar{C}} \;\; \overline{\bar{A}\bar{B}C} \;\; \overline{AB\bar{C}} \;\; \overline{ABC}$$



Binary Parallel Adder:

A Full adder is capable of adding only two single ~~bit~~ digit binary numbers along with a carry input. But, in practise, we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers, we need to use the n-bit parallel adder. It makes use of a number of full adders in cascade. The carry output of the previous full adder is connected to the carry input of the next full adder.

A 4-bit Binary Parallel Adder:

Input Carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
output carry	0	0	1	1	C_{i+1}

A binary parallel adder is a digital function that produces the arithmetic sum of two binary numbers in parallel. It consists of full-adders connected in ~~series~~ cascade, with the output carry from one full adder connected to the input carry of the next full-adder.

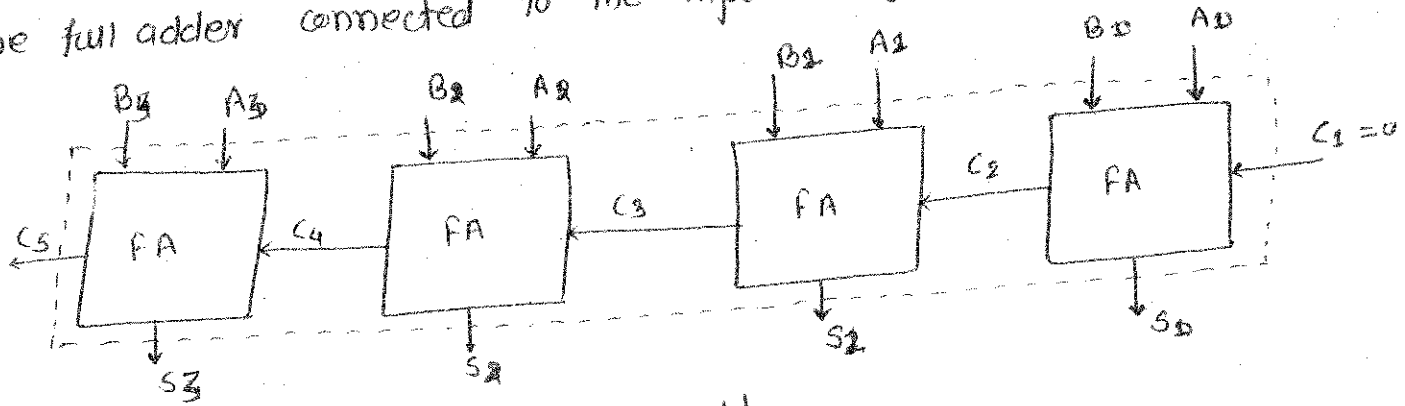


Fig: 4-bit full-adder

Example: Design a BCD-to-excess-3 code converter.

Excess-3 equivalent code can be obtained from the BCD code by the addition of binary 0011. This addition can be easily implemented by means of a 4-bit full-adders MSI circuit, as shown in figure below

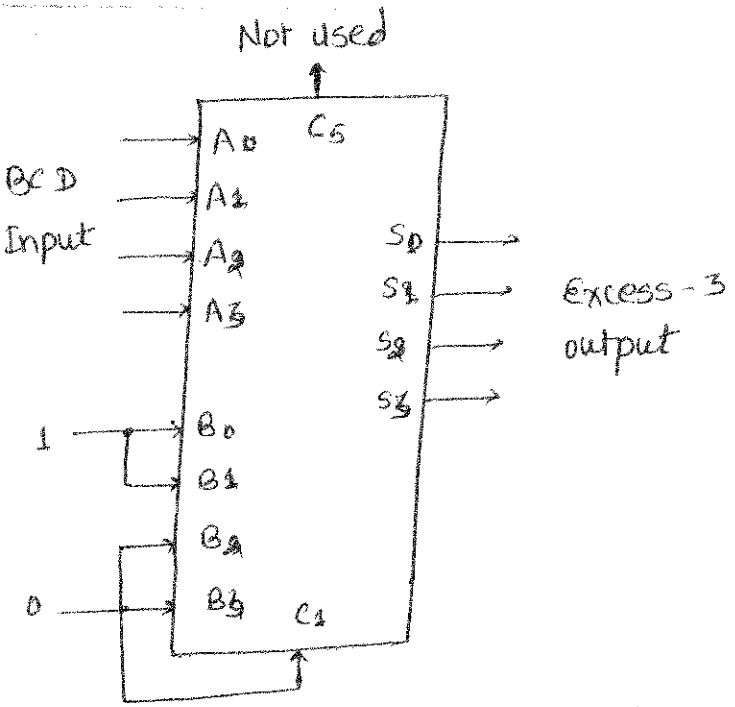


Fig: BCD-to excess-3 code converter

Example: Design a 4-bit adder using three full adders and one half adder.

Sol": Figure below shows the block diagram of the required 4-bit adder. The Half adder is used in the least significant position to add A_0 and B_0. The three full adders are used thereafter.

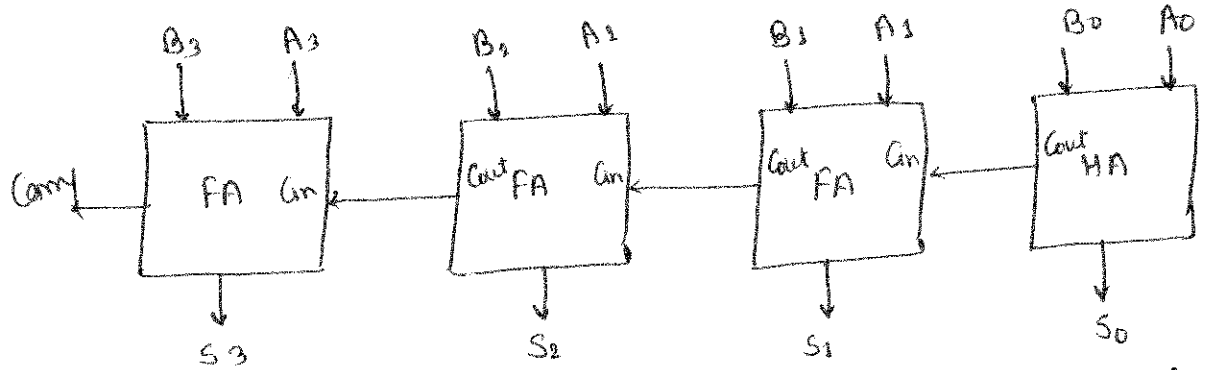


Fig: A 4-bit adder using full adders and half adder.

4-bit parallel subtractor using 2's complement:

A 4-bit parallel subtractor using a 4-bit parallel adder is shown in figure below.

The number to be subtracted (B) is first passed through inverters to obtain its 1's complement. Then 1 is added to 1's complement of B, by making $C_{in} = 1$. Thus we obtain the 2's

complement of B. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. Also, $S_3S_2S_1S_0$ represent the result of binary subtraction (A-B) and carry output Cout represents the polarity of the result. If A>B, then Cout = 0 and the result is in true binary form but if A<B then Cout = 1 and the result is in the 2's complement form.

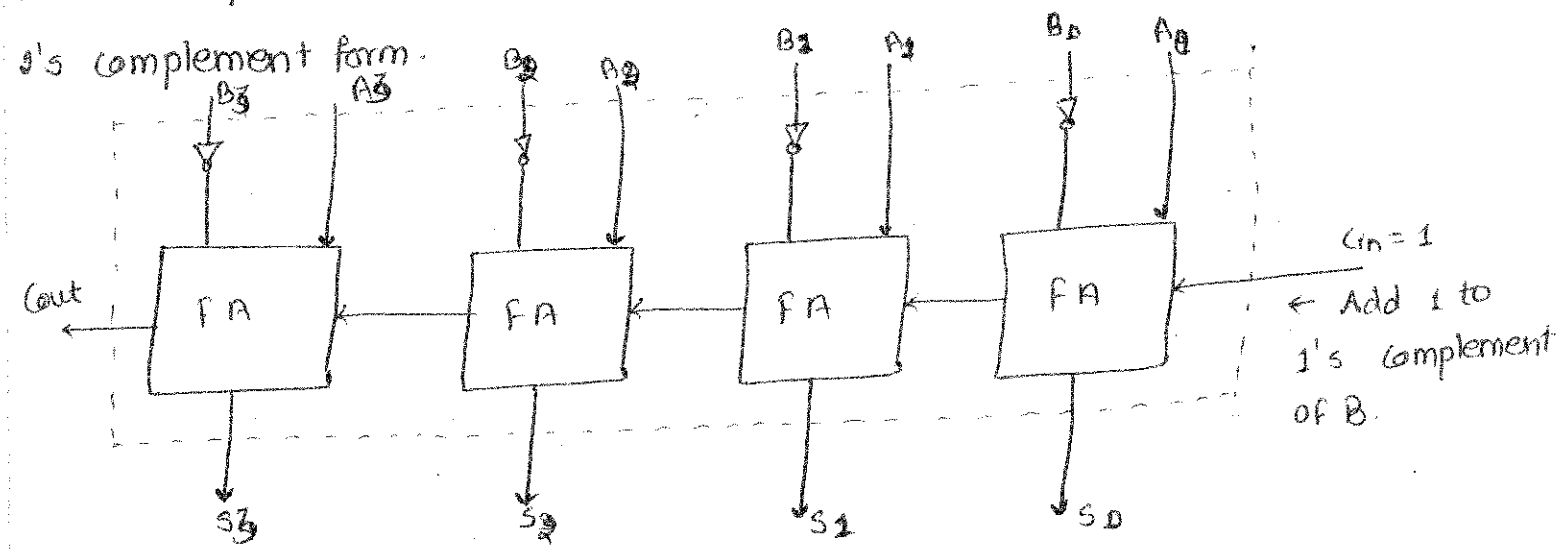


fig: 4-bit parallel subtractor

4-bit Binary parallel Adder/subtractor

The addition or subtraction of two 4-bit binary numbers can be obtained using the same circuit shown in figure below. The operation performed by this circuit (Addition or subtraction) depends on the state of the mode select input.

Here, the number B is applied to the adder through a set of EX-OR gates. One input of each EX-OR gate is connected to the mode select input (M.)

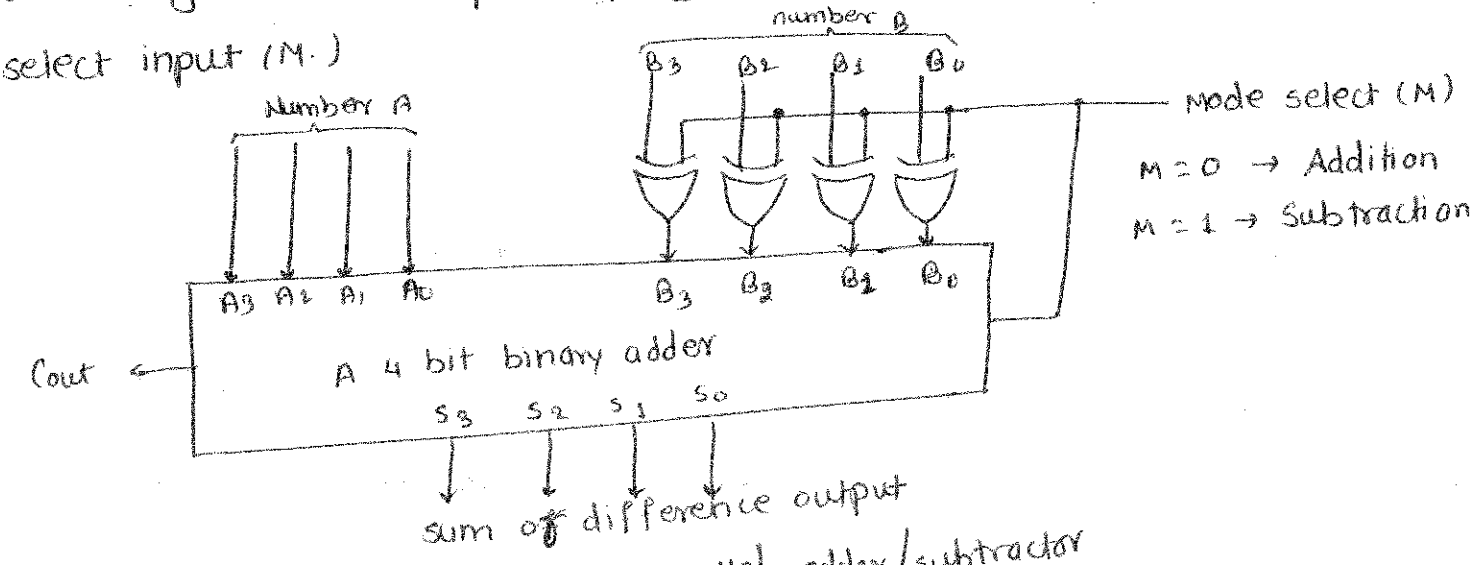


fig: A 4-bit binary parallel adder/subtractor

Working operation as Adder (M = 0)

The mode select (M) input is connected to ground. Therefore, M = 0, Since, M = 0, C_{in} = 0 and the output of the EX-OR gates will be the same number B applied at their inputs. Thus is because $0 \oplus 0 = 0$ and $0 \oplus 1 = 1$. Hence, B_3, B_2, B_1 and B_0 will pass unchanged through the EX-OR gates. The adder then adds $A + B + C_{in}$ = A + B since $C_{in} = 0$. Thus, with M = 0, addition will take place.

Working operation as subtractor (M = 1)

The mode select (M) input is connected to V_{cc}. Therefore M = 1. Since, M = 1, C_{in} = 1 and one input of each X-OR gate is now 1. Hence, each X-OR gate acts as an inverter. This is because, $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$. Hence, each bit of word B is inverted by the X-OR inverters. Thus, we get the 1's complement of number B at the output of X-OR gates. The inverted number B adds with C_{in} = 1 to give the 2's complement of B. Hence, the adder will add A with the 2's complement of B and the result is actually the subtraction A - B. Hence, with M = 1, this circuit works as a 2's complement subtractor.

4 bit BCD adder

A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD.

Suppose we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary and produce a result which may range from 0 to 19. These binary numbers are listed in table below and are labeled by symbols K, Z_8, Z_4, Z_2, Z_1. K is carry.

The table for Binary sum to BCD sum converter is

DCC	Binary					BCD				
	K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1

4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

K-map for c

Z_8Z_4 \ Z_2Z_1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$C = Z_8Z_4 + Z_8Z_2$$

Therefore, The condition for a correction and an output carry can be expressed by the boolean function:

$$c = K + Z_8Z_4 + Z_8Z_2$$

When, c = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

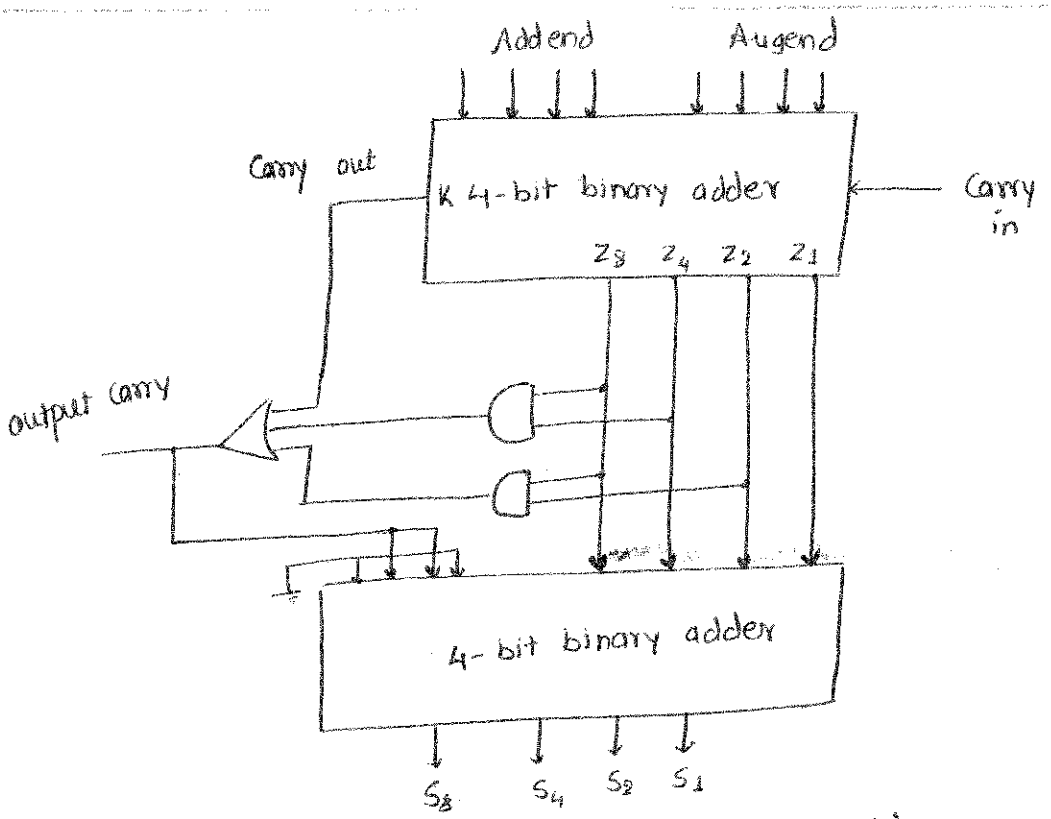


fig: Block diagram of a BCD adder

Magnitude comparator:

A magnitude comparator is a combinational circuit that compares two numbers, A and B, and determines their relative magnitudes. The outcome of the comparision is specified by three binary variables that indicate whether $A > B$, $A = B$ or $A < B$.

Consider two number, A and B, with four digits each.

$A = A_3 A_2 A_1 A_0$

$B = B_3 B_2 B_1 B_0$

Case 1:

$A = B$

The two numbers are equal if all pairs of significant digits are equal i.e if $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$. The equality relation of each pair of bits can be expressed logically with an equivalence function:

$$x_i = A_i B_i + A_i' B_i' \quad , \quad i = 0, 1, 2, 3$$

Where, $x_i = 1$ only if the pair of bits in position i are equal. i.e, if both are 1's or both are 0's.

For equality conditions to exist, all x_i variables must be equal to 1. This dictates an AND operation of all variables:

$$(A = B) = x_3 x_2 x_1 x_0$$

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

Case 2:

$A > B$ or $A < B$

To determine if A is greater than or less than B, we inspect the relative magnitude of pairs of significant digits starting from the most significant position. If the two digits are equal, we compare the next lower significant pair of digits. This comparision continues until a pair of unequal digits is reached. If the corresponding digit of A is 1 and that of B is 0, we conclude that $A > B$. If the corresponding digit of A is 0 and that of B is 1, we have that $A < B$. The sequential comparision can be expressed logically by the following two Boolean functions

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$

NOTE:

for 2-bit $x_i = A_i B_i + A_i' B_i'$ for $i = 0, 1$

$$(A = B) = x_1 \cdot x_0$$

$$(A > B) = A_1 B_1' + x_1 A_0 B_0'$$

$$(A < B) = A_1' B_1 + x_1 A_0' B_0$$

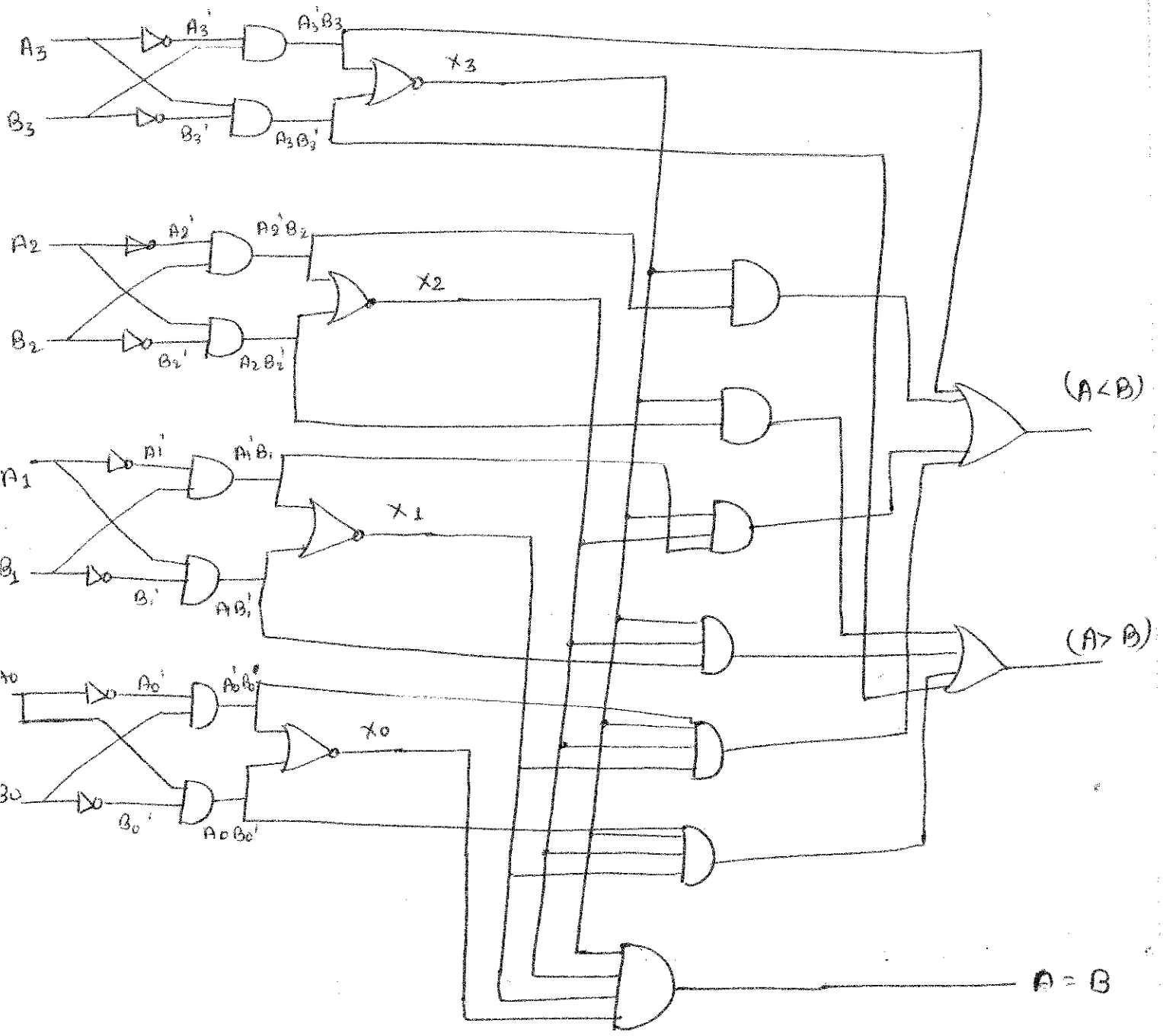


Fig: 4-bit magnitude comparator

Decoders

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit decoded information has unused or don't care combinations, the decoder output will have less than 2^n outputs

3 to 8 bit decoders:

The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. In order to decode all possible combination of 3 bit, 8 decoder gates are required i.e. $2^3 = 8$

This type of decoder is commonly called 3 to 8 line decoder. There are 3 inputs and 8 outputs. A list of binary codes and their corresponding decoding function is given below:

Decimal digit	Binary i/p			Outputs								Decoding function
	x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
0	0	0	0	1	0	0	0	0	0	0	0	$\bar{x}\bar{y}\bar{z} = D_0$
1	0	0	1	0	1	0	0	0	0	0	0	$\bar{x}\bar{y}z = D_1$
2	0	1	0	0	0	1	0	0	0	0	0	$\bar{x}y\bar{z} = D_2$
3	0	1	1	0	0	0	1	0	0	0	0	$\bar{x}yz = D_3$
4	1	0	0	0	0	0	0	1	0	0	0	$x\bar{y}\bar{z} = D_4$
5	1	0	1	0	0	0	0	0	1	0	0	$x\bar{y}z = D_5$
6	1	1	0	0	0	0	0	0	0	1	0	$xy\bar{z} = D_6$
7	1	1	1	0	0	0	0	0	0	0	1	$xyz = D_7$

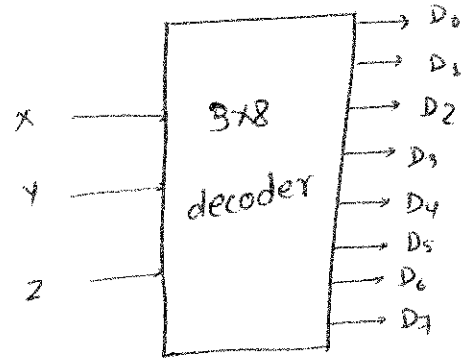


fig: 3×8 decoder

A logic symbol for 3×8 line decoder with inputs and outputs is shown in figure above. The binary decimal label indicates that the binary input makes the corresponding output active. The input lebel 4, 2, 1 represents the binary weights of ~~three~~ the input bits.

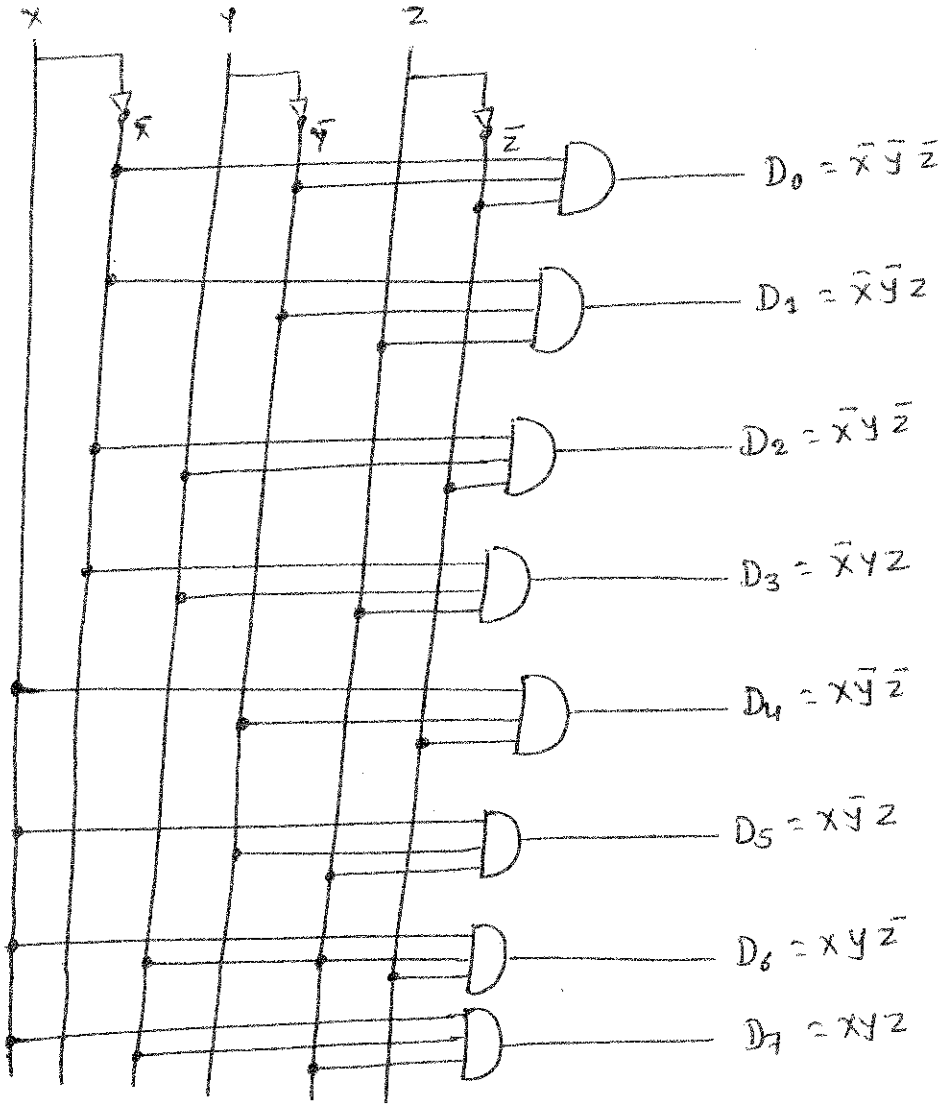


Fig: A 3 to 8 line decoder

Combinational logic implementation of decoder:

Full adder circuit with a decoder:

$$\text{sum}(S) = x \oplus y \oplus z = x \oplus (y'z + yz')$$

$$= x'(y'z + yz') + x(y'z + yz')'$$

$$= x'y'z + x'yz' + x(yz + y'z')$$

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

$$= m_1 + m_2 + m_7 + m_4$$

$$\therefore S = \Sigma(1, 2, 4, 7)$$

and, $\text{Carry}(C) = xy + yz + xz$

$$= xy(z + z') + yz(x + x') + xz(y + y')$$

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

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

$$= m_7 + m_6 + m_3 + m_5$$

$$\therefore C = \Sigma(3, 5, 6, 7)$$

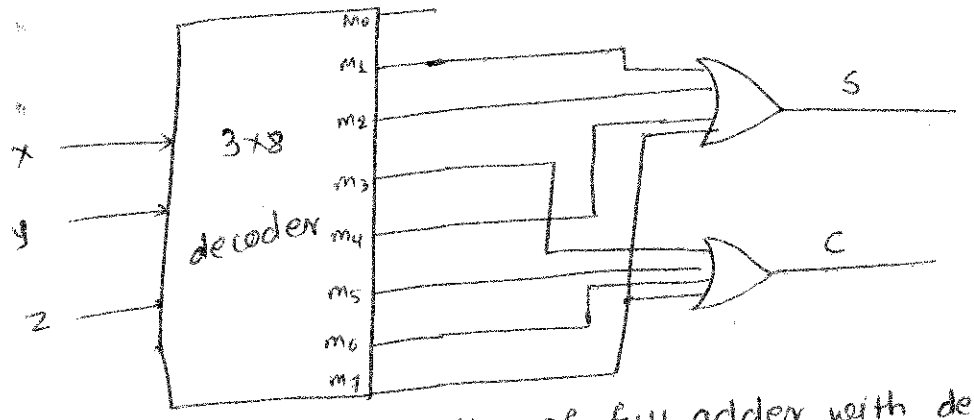


fig: implementation of full adder with decoder

Full subtractor circuit with a decoder:

Difference (D) = $x \oplus y \oplus z$

$= x \oplus (yz' + y'z)$

$= x'(yz' + y'z) + x(yz' + y'z)'$

$= x'(yz' + y'z) + x(yz + y'z')$

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

$= m_2 + m_1 + m_7 + m_4$

$D = \Sigma(1, 2, 4, 7)$

Borrow (B) = $x'z + yz + x'y$

$= x'z(y + y') + yz(x + x') + x'y(z + z')$

$= x'yz + x'y'z + xyz + x'yz + x'yz + x'yz'$

$= m_3 + m_1 + m_7 + m_3 + m_3 + m_2$

$= m_1 + m_2 + m_3 + m_7$

$B = \Sigma(1, 2, 3, 7)$

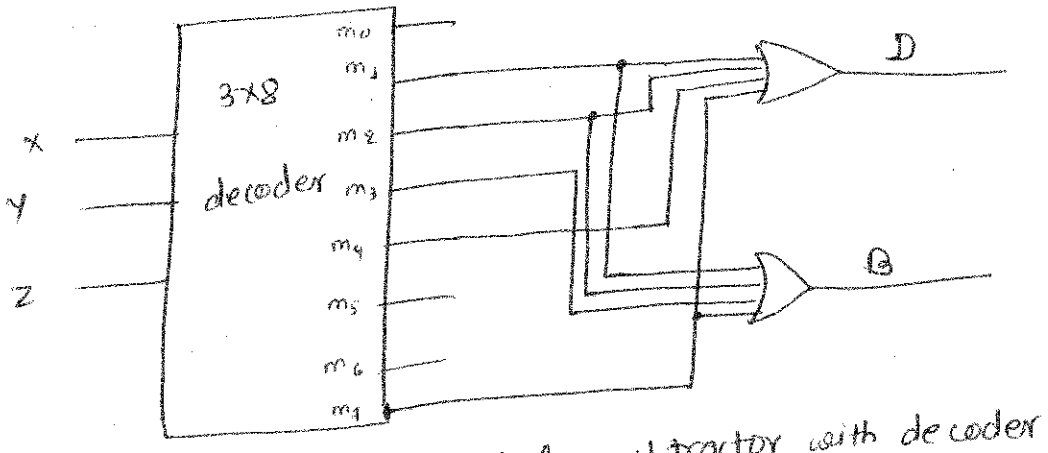


fig: Implementation of full subtractor with decoder

Demultiplexers:

A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines.

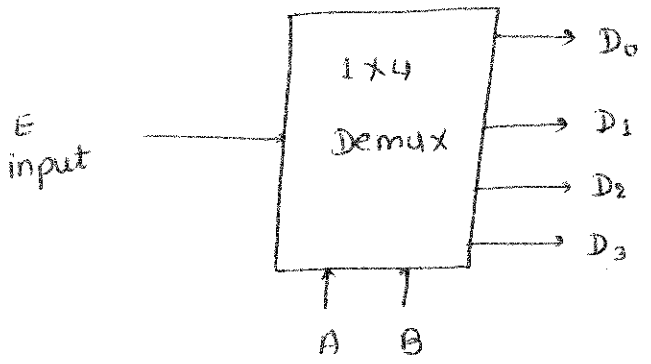


fig: 1×4 demultiplexer

Truth table

input	selection line		output				function
E	A	B	D_0	D_1	D_2	D_3	
0	x	x	0	0	0	0	$D_0 =$
1	0	0	1	0	0	0	$D_0 = E\bar{A}\bar{B}$
1	0	1	0	1	0	0	$D_1 = E\bar{A}B$
1	1	0	0	0	1	0	$D_2 = EA\bar{B}$
1	1	1	0	0	0	1	$D_3 = EAB$

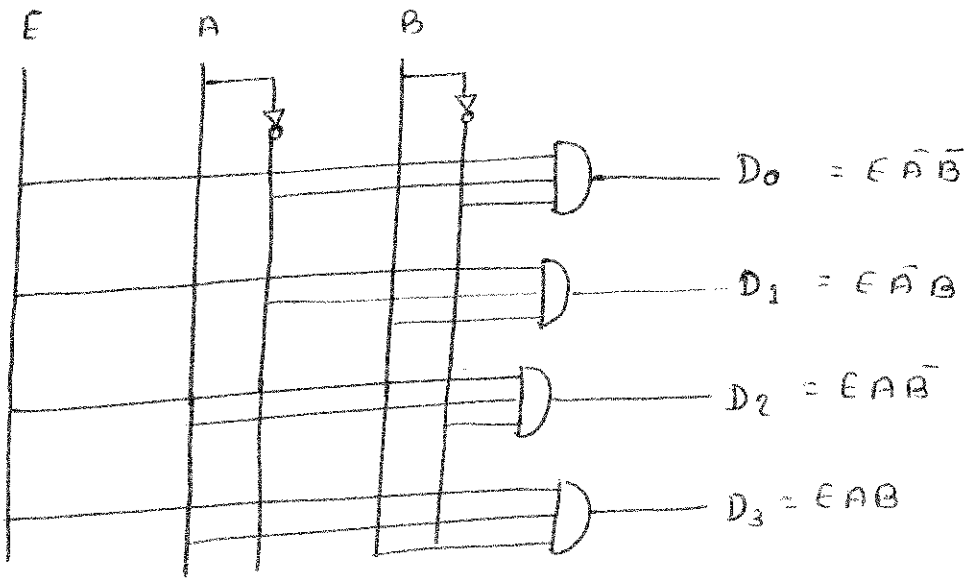


fig: ~~implementation~~ combinational logic diagram of demux

Encoders:

An encoder is a digital function that produces a reverse operation from that of a decoder. An encoder has 2^n (or less) input lines and n output lines. The output line generate the binary code for the 2^n input variables.

The octal-to-binary encoder consists of eight inputs, one for each of the eight digits, and three outputs that generate the corresponding binary number.

Truth table

Dec	Inputs								outputs		
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	1	0	1
6	0	0	0	0	0	0	1	0	1	1	0
7	0	0	0	0	0	0	0	1	1	1	1

$$X = D_4 + D_5 + D_6 + D_7$$

$$Y = D_2 + D_3 + D_6 + D_7$$

$$Z = D_1 + D_3 + D_5 + D_7$$

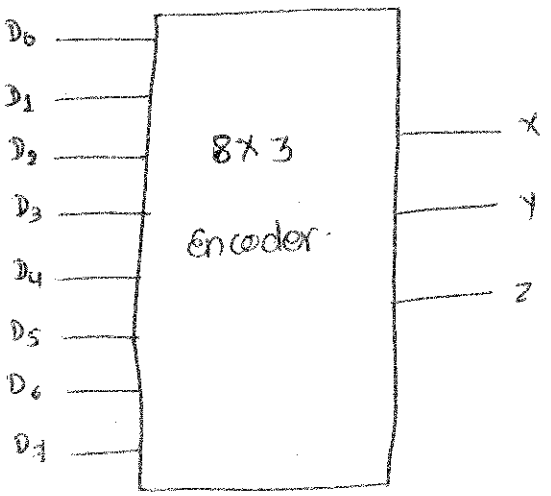


fig: 8X3 encoder

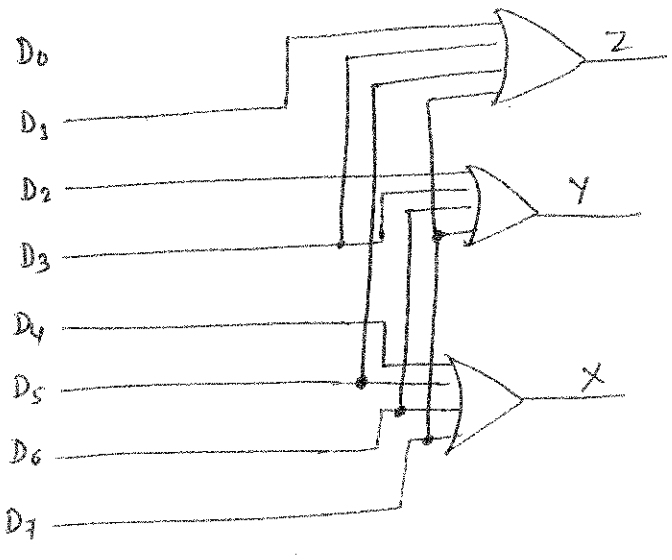
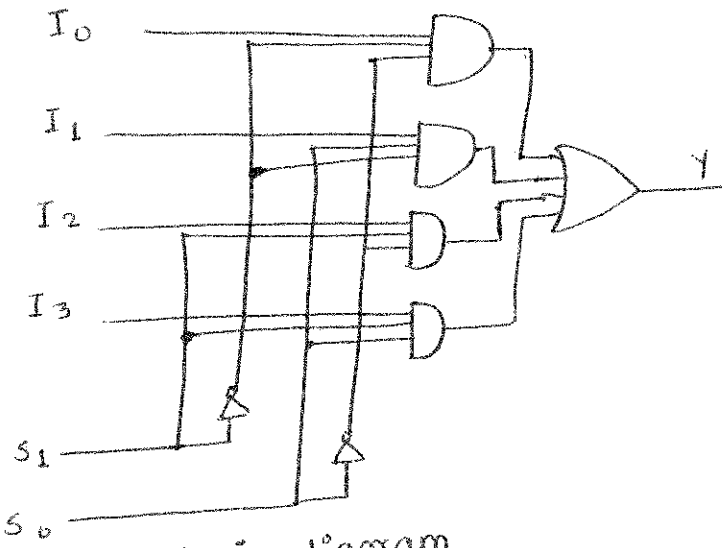


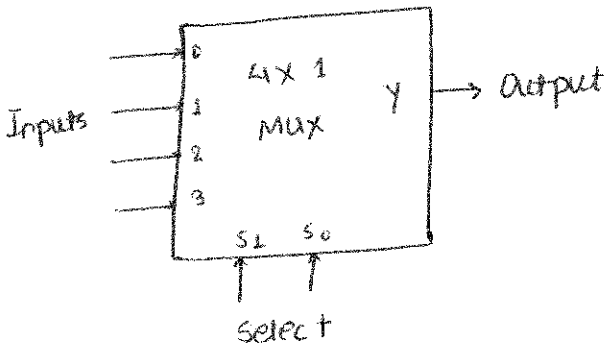
fig: octal to binary encoder

MULTIPLEXERS:

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.



a) Logic diagram



b) Block diagram

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

c) function table

$$Y = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0$$

Fig: A 4-to-1 line multiplexer

Application

- It is used as a data selector to select one out of many data inputs
- It is used for simplification of logic design
- In the data acquisition system
- In designing the combinational circuits
- In the D/A converters
- To minimize the number of connections.

Example1 Implement a 4:1 multiplexer using 2:1 multiplexers

soln: Data inputs : ~~D_0 D_1 D_2 D_3~~ I_0, I_1, I_2, I_3

Select inputs : S_1, S_0

output : Y

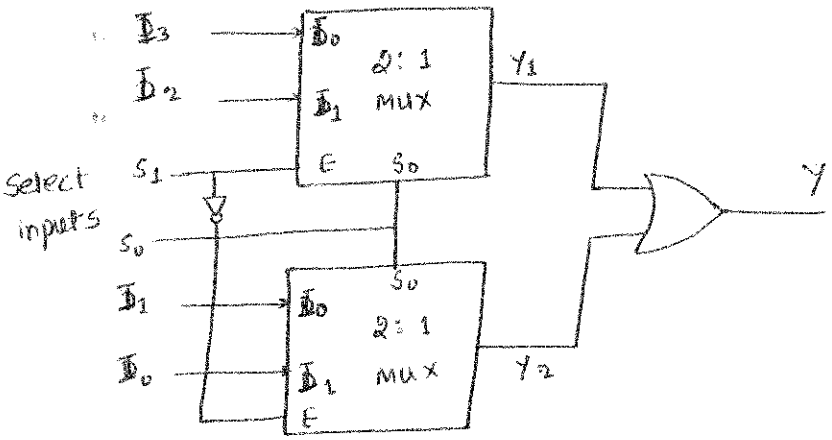


Fig: 4:1 Multiplexer using 2:1 multiplexers

Example 2

Obtain the 8:1 multiplexer using two 4:1 multiplexers

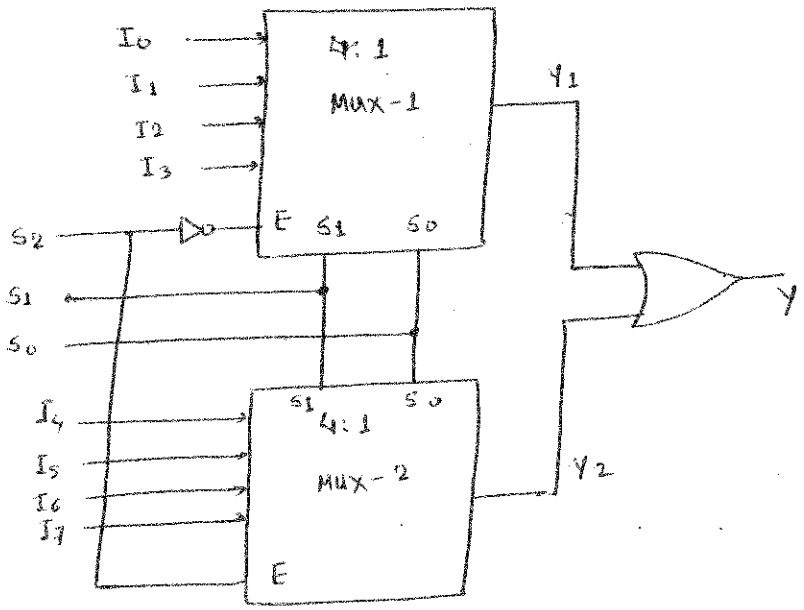


Fig: 8:1 multiplexer by cascading two 4:1 multiplexers

Select inputs			output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Mux 1 is enable

Mux 2 is enable

Fig: Truth table

There are in all eight data inputs (I_0 through I_7). The select lines S_1 and S_0 of both 4:1 multiplexers are connected in parallel whereas a third select input S_2 is used for enabling one multiplexer at a time.

Example 3: Implement a 16:1 multiplexer using 4:1 multiplexers

Description: The select input S_1 and S_0 of the multiplexers 1, 2, 3 and 4 are connected together. The select inputs S_3 and S_2 are applied to the select input S_1 and S_0 of Mux-5. The output Y_1, Y_2, Y_3, Y_4 are applied to the data inputs I_0, I_1, I_2 and I_3 of Mux-5 as shown in fig below.

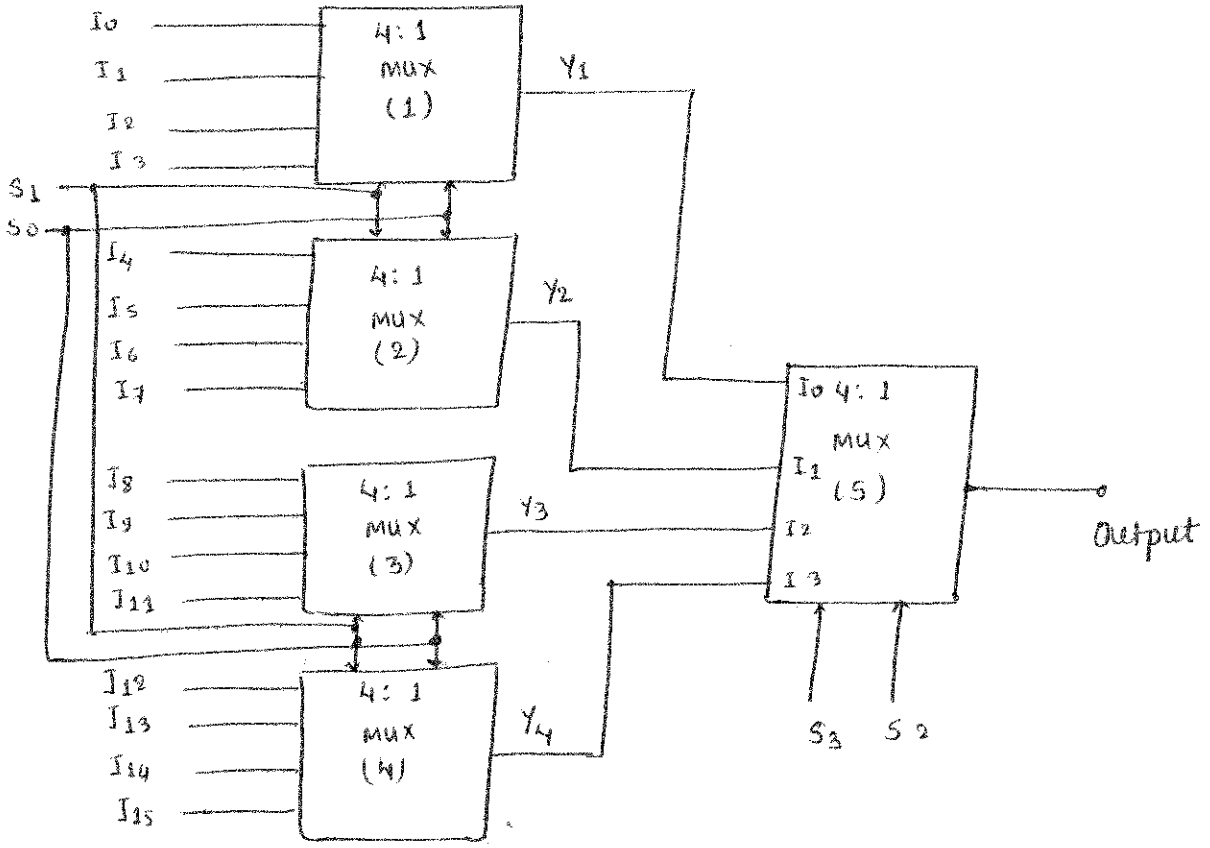


Fig: 16:1 multiplexer using 4:1 multiplexers

Select inputs				Mux. outputs				Final output	
S_3	S_2	S_1	S_0	Y_1	Y_2	Y_3	Y_4	Y	
0	0	0	0	I_0	I_4	I_8	I_{12}	I_0	$S_3S_2 = 00$
0	0	0	1	I_1	I_5	I_9	I_{13}	I_1	∴ MUX-5 select Y_1
0	0	1	0	I_2	I_6	I_{10}	I_{14}	I_2	
0	0	1	1	I_3	I_7	I_{11}	I_{15}	I_3	
0	1	0	0	I_0	I_4	I_8	I_{12}	I_4	$S_3S_2 = 01$
0	1	0	1	I_1	I_5	I_9	I_{13}	I_5	∴ MUX-5 select Y_2
0	1	1	0	I_2	I_6	I_{10}	I_{14}	I_6	
0	1	1	1	I_3	I_7	I_{11}	I_{15}	I_7	
1	0	0	0	I_0	I_4	I_8	I_{12}	I_8	$S_3S_2 = 10$
1	0	0	1	I_1	I_5	I_9	I_{13}	I_9	∴ MUX-5 select
1	0	1	0	I_2	I_6	I_{10}	I_{14}	I_{10}	Y_3
1	0	1	1	I_3	I_7	I_{11}	I_{15}	I_{11}	
1	1	0	0	I_0	I_4	I_8	I_{12}	I_{12}	$S_3S_2 = 11$
1	1	0	1	I_1	I_5	I_9	I_{13}	I_{13}	∴ MUX-5 select
1	1	1	0	I_2	I_6	I_{10}	I_{14}	I_{14}	Y_4
1	1	1	1	I_3	I_7	I_{11}	I_{15}	I_{15}	

Fig: Summary of operation

Read - only memory (ROM)

A ROM is essentially a memory (or storage) device in which a fixed set of binary information is stored. The binary information must first be specified by the user and is then embedded in the unit to form the required interconnection pattern. Once a pattern is established for a ROM, it remains fixed even when power is turned off and then on again.

A block diagram of a ROM is shown in figure below. It consists of n input lines and m output lines. Each bit combination of the input variable is called a word. The number of bits per word is equal to the number of output lines m. An address is essentially a binary number that denotes one of the minterms of n variable. The number of distinct addresses possible with n input variables is 2^n. An output word can be selected by a unique address, and since there are 2^n distinct addresses in a ROM, there are 2^n distinct words which are said to be stored in the unit. The word available on the output lines at any given time depends on the address value applied to the input lines. A ROM is characterized by the number of words 2^n and the number of bits per word m.

example: 32X8 ROM, 32X4 ROM

Internally, the ROM is a combinational circuit with AND gates connected as a decoder and a number of OR gates equal to the number of outputs in the unit

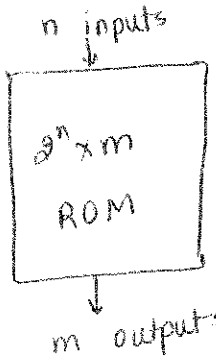


Fig: ROM block diagram

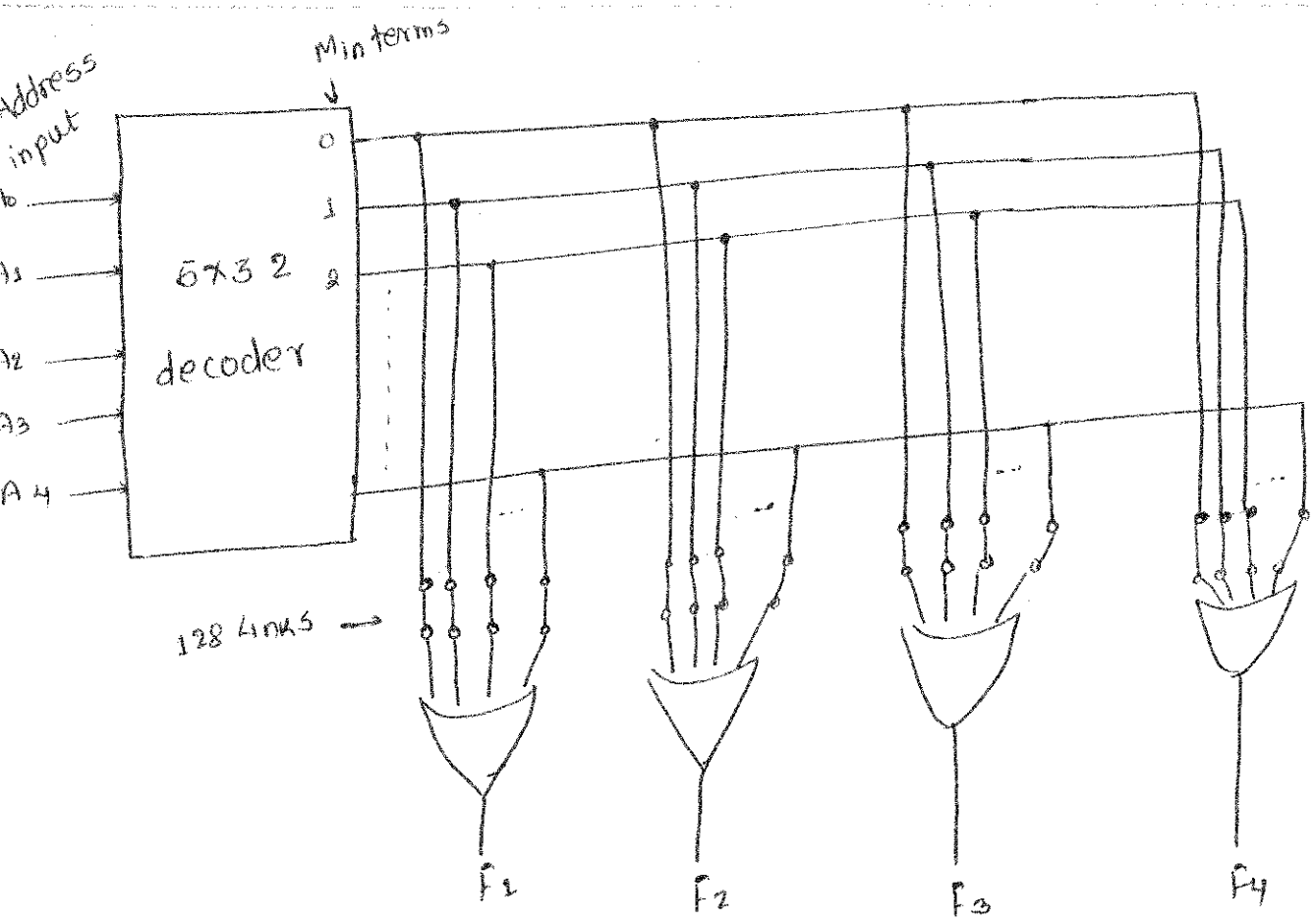


Fig: Logic construction of a 32×4 ROM

Implementation of ROM

Eg: Implement

$F_1(A_1, A_0) = \Sigma(1,2,3)$

$F_2(A_1, A_0) = \Sigma(0,2)$ with 4×2 bit ROM

Sol^n:

Here,

We have 4 words

So, $2^n = 4$

$n = 2$ inputs

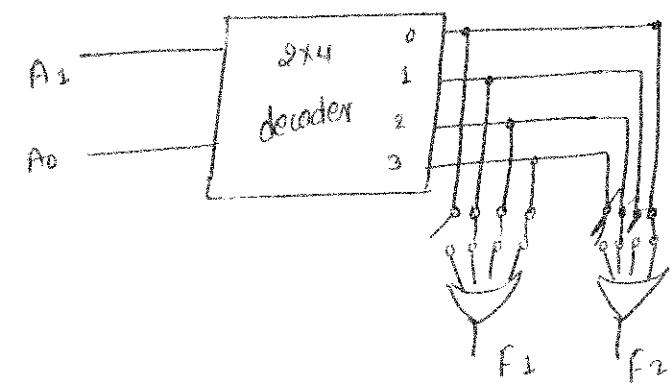


Fig 4×2 bit ROM

Programmable Logic Array (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 PLA, the decoder is replaced by a group of AND gates, each of which can be programmed to generate a ~~program~~ product term of the input variables. The AND and OR gates inside the PLA are initially fabricated with links among them. The specific Boolean functions are implemented in sum of product form by opening appropriate links and leaving the desired connections.

A block diagram of the PLA is shown in figure below. 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.

The size of the PLA is specified by the number of inputs, the number of product terms, and the number of outputs. A Typical PLA has 16 inputs, 48 product terms, and 8 outputs. The number of programmed links is $2nxk + kxm + m$, whereas that of a ROM is $2^n x m$.

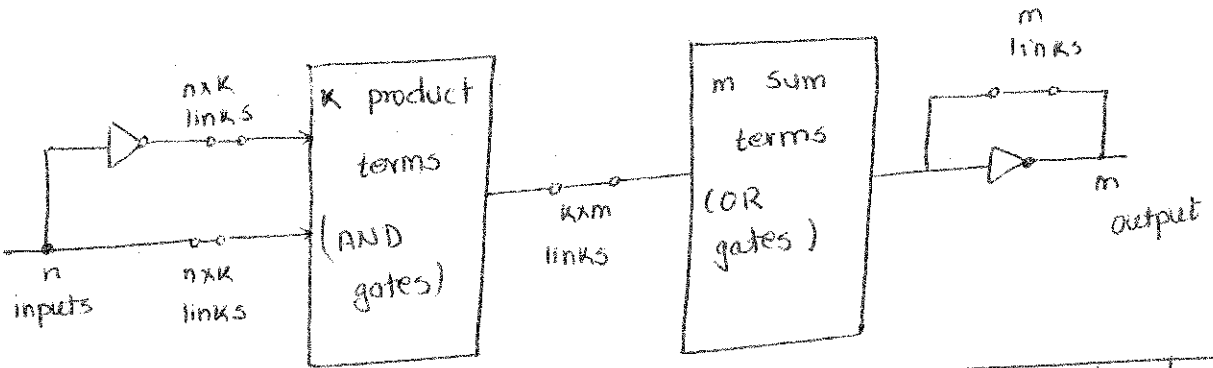


Fig: PLA block diagram

Implementation:

Implement $F_1 = AB' + AC$

$F_2 = AC + BC$ using PLA

	product term	Inputs A	B	C	outputs F_1	F_2
AB'	1	1	0	-	1	-
AC	2	1	-	1	1	1
BC	3	-	1	1	-	1
					T	T	T/C

Fig: PLA program table

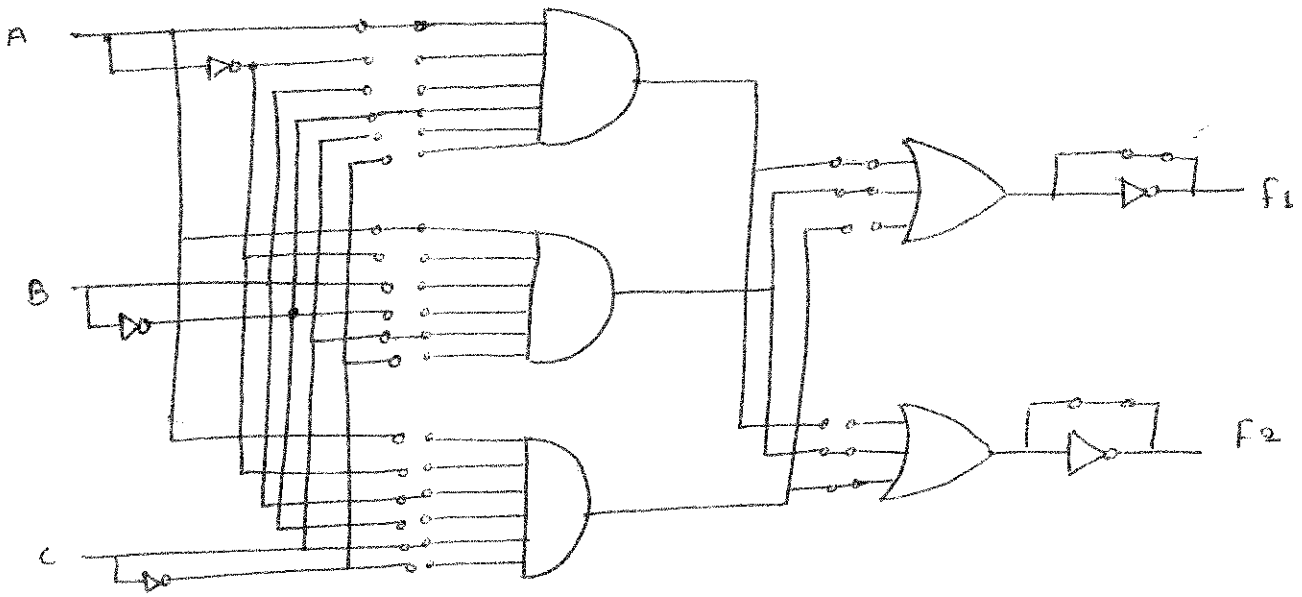


Fig: PLA implementation.

Boolean function implementation of MUX:

1. $F(A, B, C) = \Sigma(1, 3, 5, 6)$

SOP: No. of variables = 3 i.e. $n+1 = 3$

So, selection line = 2

Now, Implementation table

	I_0	I_1	I_2	I_3
$\bar{A}$	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	$\bar{A}$

2. $F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$

SOP: No. of variables = 4 i.e. $n+1=4$

So, selection line = 3

Now, Implementation table

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\bar{A}$	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

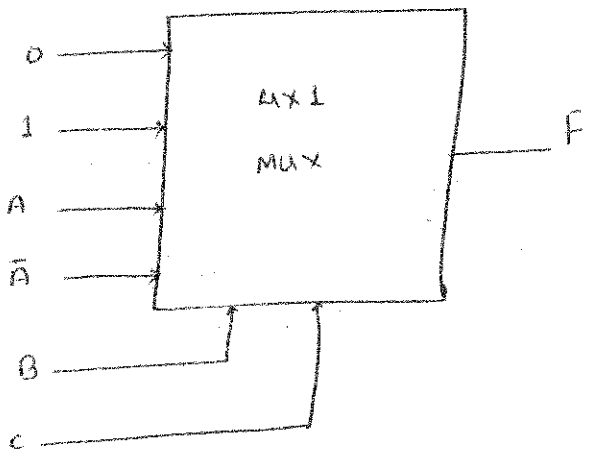


Fig: Multiplexer implementation

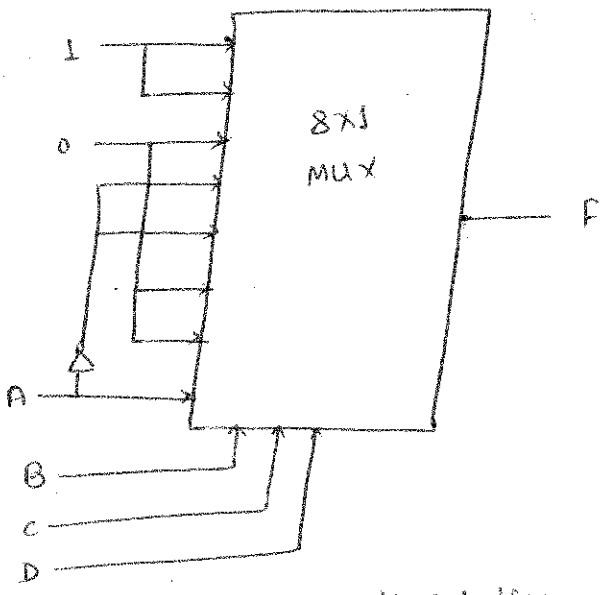


Fig: Multiplexer implementation

A block diagram of a sequential circuit is shown in figure below. It consists of a combinational circuit to which memory elements are connected to form a feedback path.

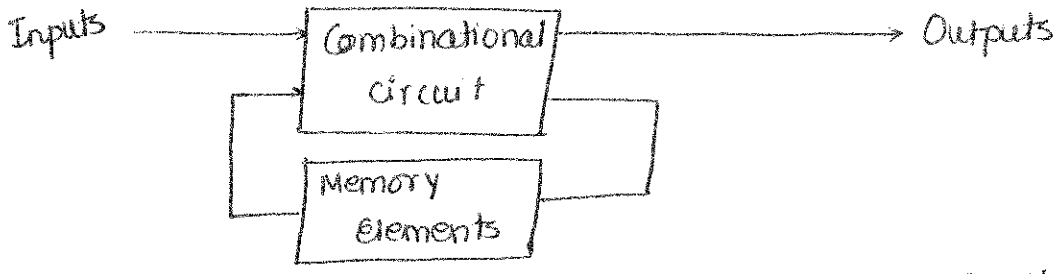


Fig: Block diagram of a sequential circuit

There are two main types of sequential circuits. Their classification depends on the timing of their signals.

1) Synchronous sequential circuit
2) Asynchronous sequential circuit

1) Synchronous Sequential circuit:

A synchronous sequential logic system, by definition must employ signals that affect the memory elements only at discrete instants of time. Synchronization is achieved by a timing device called a master-clock generator which generates a periodic train of clock pulses. Synchronous sequential circuits that use clock pulses in the inputs of memory elements are called clocked sequential circuits. The memory elements used in clocked sequential circuits are called flip-flops.

2) Asynchronous sequential circuit:

The behavior of an asynchronous sequential circuit depends upon the order in which its input signals change and can be affected at any instant of time. The memory elements commonly used in asynchronous sequential circuits are time-delay devices. The memory capability of a time-delay device is due to the fact that it takes a finite time for the signal to propagate through the device

Clock:

It is a control signal that periodically makes a transition from zero to one (0 to 1) and then back to zero (0) again. We usually denote the clock by symbol $\phi(t)$, CP, CLK.

Flip-Flops

The memory elements used in clocked sequential circuits are called 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 gives rise to different types of flip-flops.

A flip-flop circuit can maintain a binary state indefinitely (as long as power is delivered to the circuit) until directed by an input signal to switch states. The major difference among various types of flip-flops are in the number of inputs they posses and in the manner in which the inputs affect the binary state.

Basic flip-flop circuit: [R-S flip-flop circuit]

flip-flop circuit can be constructed from two NAND or NOR gates. Each circuit forms a basic flip-flop upon which other more complicated types can be built. The cross-coupled connection from the output of one gate to the input of the other gate constitutes a feedback path. For this reason, the circuits are classified as asynchronous sequential circuits. Each flip-flop has two outputs, Q and Q', and two inputs, Set and reset. This type of flip-flop is sometimes called a direct-coupled RS flip-flop or SR latch.

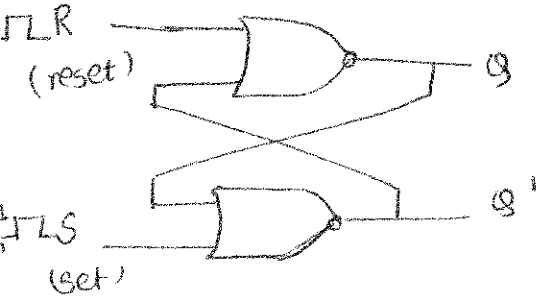


Fig: Active high input SR Latch.

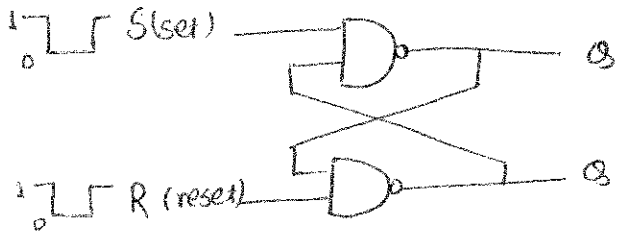


Fig. Active low input SR latch.

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

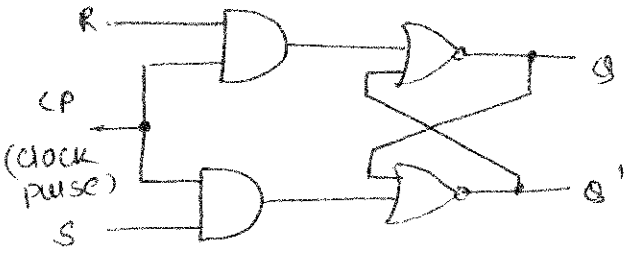
b) Truth table for R-S Flip-Flop with NOR

S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

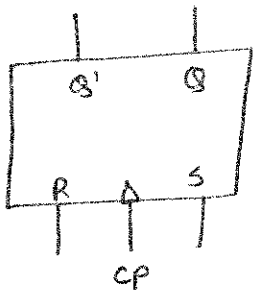
b) Truth table for R-S Flip Flop using NAND.

Clocked RS Flip Flop

By adding gates to the inputs of the basic circuit, the flip-flop can be made to respond to input levels during the occurance of a clock pulse.



a) Logic diagram



b) Graphic symbol

Q(t)	S	R	Q(t+1)	
0	0	0	0	Not changed
0	0	1	0	Reset
0	1	0	1	Set
0	1	1	indeterminate	
1	0	0	1	Not changed
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	indeterminate	

Fig: a characteristic table

Q \ SR	S'R' 00	S'R 01	SR 11	SR' 10
Q' 0	0	0	X	1
Q 1	1	0	X	X

$Q(t+1) = S + Q R'$

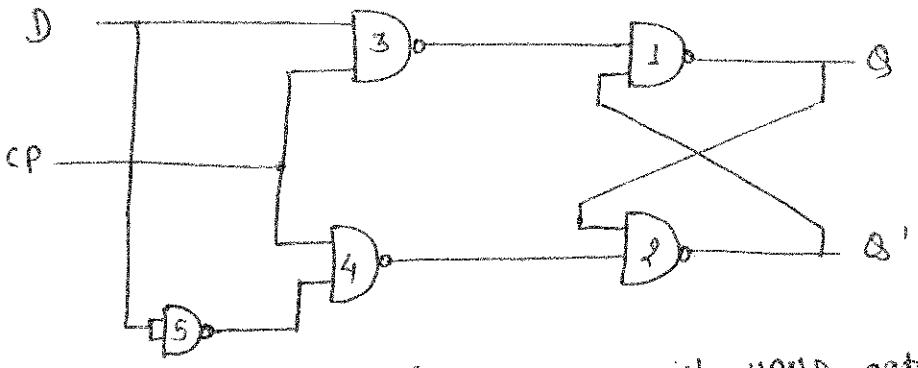
Fig d) characteristic equation

Fig: clocked RS flip-flop

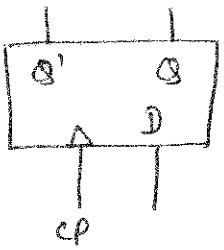
D flip flop

The D-flip flop shown in figure below is a modification of the clocked RS flip-flop NAND gates 1 and 2 form a basic flip-flop and gates 3 and 4 modify it into a clocked RS flip-flop. The D input goes directly to the S input

and its complement, through gate 5, is applied to the R input.



a) Logic diagram with NAND gates



b) Graphic symbol

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

c) characteristic table

Q \ D	D' 0	D 1
Q' 0	0	1
Q 1	0	1

$Q(t+1) = D$

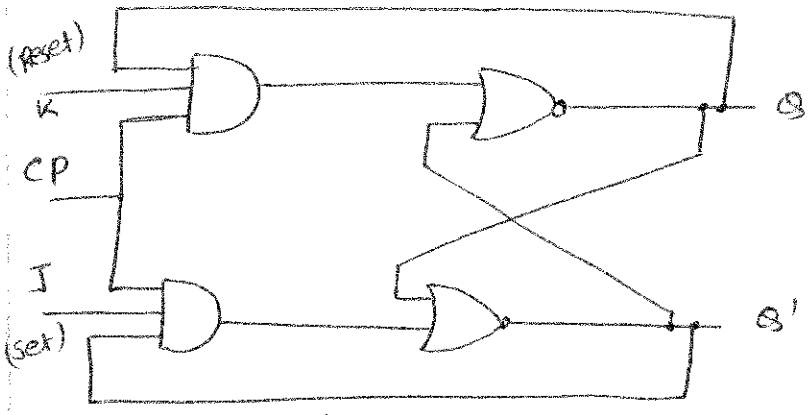
d) characteristic equation

Fig: clocked D flip-flop

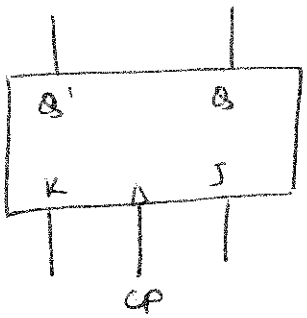
JK flip flop

A JK flip-flop is a refinement of the RS flip-flop in that the indeterminate state of the RS type is defined in the JK type. Inputs J and K behaves like inputs S and R to set and clear the flip-flop (~~note that~~ In JK flip-flop, the letter J is for set and the letter K is for clear). When inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, i.e. if Q = 1, it switches to Q = 0, and vice versa.

When bot J and K are 1, the clock pulse is transmitted through one AND gate only - the one whose input is connected to the flip-flop output ~~switch~~ which is presently equal to 1. Thus, if Q = 1, the output of the upper AND gates becomes 1 upon application of a clock pulse, and the flip-flop is cleared. If Q' = 1, the output of the lower AND gate becomes a 1 and the flip-flop is set.



a) Logic diagram



b) Graphic Symbol

Q(t)	J	K	Q(t+1)	Comment
0	0	0	0	not changed
0	0	1	0	Reset
0	1	0	1	Reset
0	1	1	1	Invert
1	0	0	1	not changed
1	0	1	0	Reset
1	1	0	1	set
1	1	1	0	Invert

c) characteristic table

Q \ JK	J'K' 00	J'K 01	JK 11	JK' 10
Q' 0	0	0	1	1
Q 1	1	0	0	1

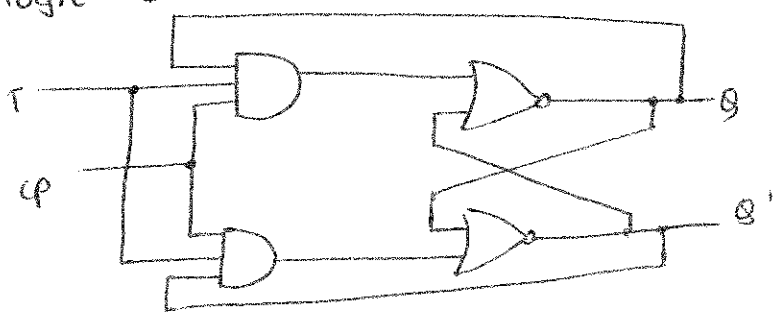
$Q(t+1) = Q'J + QK'$

d) characteristic equation

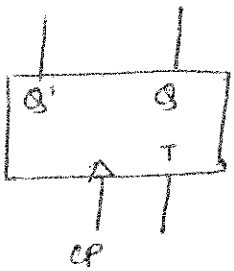
Fig: clocked JK flip-flop

T-Flip flop

The T flip-flop is a single-input version of the JK flip-flop. The T flip-flop is obtained from a JK type if both inputs are tied together. The designation T comes from the ability of the flip-flop to "toggle", or change state. Regardless of the present state of the flip-flop, it assumes the complement state when the clock pulse occurs while input T is in logic-1.



a) Logic diagram



b) Graphic symbol

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

c) characteristic table

Q \ T	T'	T
Q	0	1
Q' 0	0	[1]
Q 1	[1]	0

$Q(t+1) = Q'T + T'Q$

d) characteristic equation

fig: clocked T flip-flop

Triggering of flip-flop

The state of a flip-flop is switched by a momentary change in the input signal. This momentary change is called a trigger and the transition it causes is said to trigger the flip-flop. Asynchronous flip-flops require an input trigger defined by a change of signal level. Clocked flipflop are triggered by pulses.

There are two types of triggering:

1. Pulse width triggering
2. Edge triggering

1. Pulse width triggering:

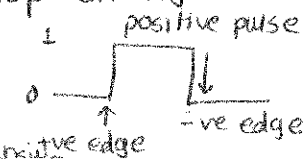
In pulse width triggering the flip-flop changes its state when clock pulse is equal to logic 1 and does not change its state when clock pulse is equal to logic 0.

fig: pulse width trigger

2. Edge triggering:

A clock pulse may be either positive or negative. The pulse goes through two signal transitions: from 0 to 1 and the return from 1 to 0. The positive transition is defind as the positive edge and the negative transition as the negative edge.

An edge triggering flip flop changes its state either at positive edge or at the -ve edge.



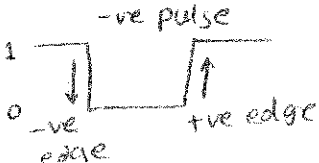


fig: defination of clock pulse transitive edge

Master-slave flip flop

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a master-slave flip-flop.

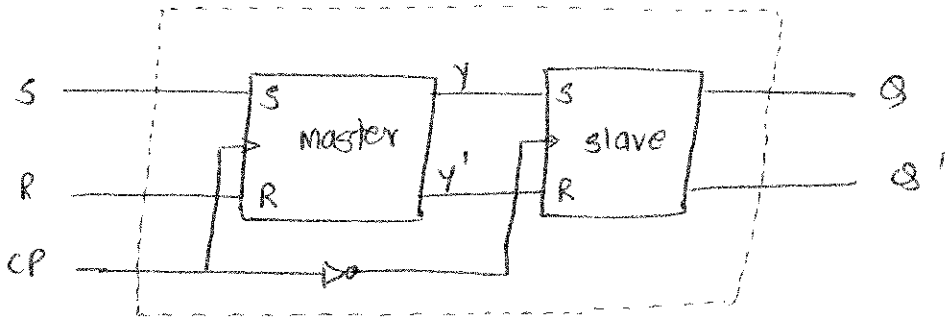


fig: Logic diagram of master-slave flip-flop

When cp = 0, the master flip-flop is disabled and the slave flip-flop is enabled and output Q is equal to Y, while Q' is equal to Y'

When cp = 1, the information then at the external R and S inputs is transmitted to the master flip-flop. The slave flip-flop, however, is isolated as long as the pulse is at its 1 level. When the pulse return to 0, the master flip-flop is isolated, when prevents the external inputs from affecting it. The slave flip-flop then goes to the same state as the master flip-flop.

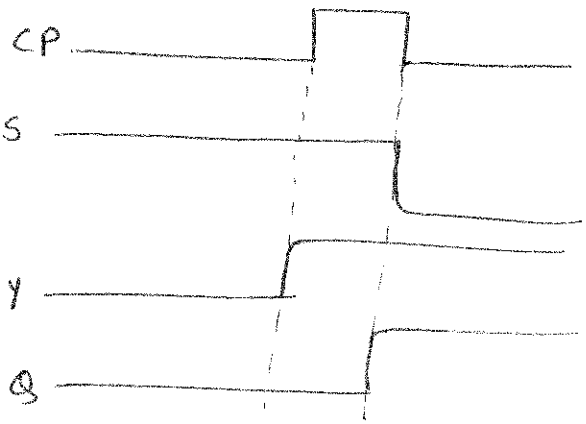


fig: timing relationship in a master slave flip flop

Edge Triggered flip-flop

Another type of flip-flop that synchronizes the state changes during a clock pulse transition is the edge-triggered flip-

flop. In this type of flip-flop, output transitions occur at a specific level of the clock pulse when the pulse input level exceeds this threshold level, the inputs are locked out and the flip-flop is therefore unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs. Some edge-triggered flip-flops cause a transition on the positive edge of the pulse, and others cause a transition on the negative edge of the pulse.

Analysis of clocked sequential circuits

The analysis of sequential circuits consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states. It is also possible to write Boolean expressions that describes the behaviour of sequential circuits.

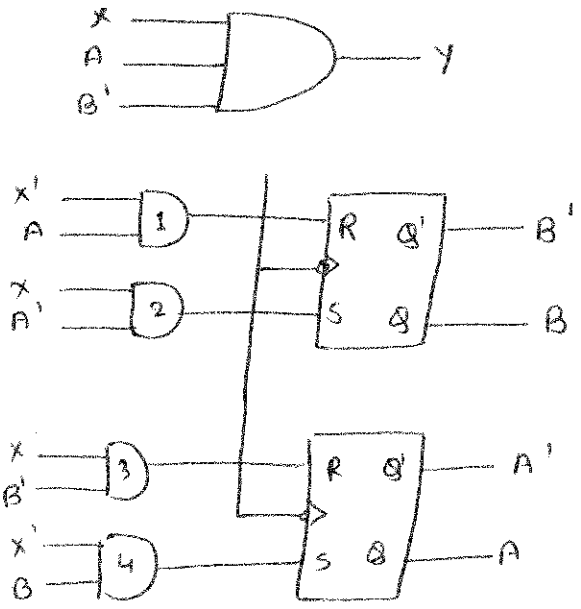


fig: Example of a clocked sequential circuit

For analysis, we first draw:

1) state table

The time sequence of inputs, outputs, and flip-flop states may be enumerated in a state table. It consists of three section labeled present state, next state, and output. The 'present state' designates the state of flip-flops before the occurrence of clock pulse. The 'next state' shows the states of flip-flops after the application of a clock pulse and the output section lists the values of the output variables during the present state.

Both the next state and output section have two columns, one for x = 0 and the other for x = 1.

present state	Next state		output	
	x = 0	x = 1	x = 0	x = 1
AB	AB	AB	Y	Y
00	00	01	0	0
01	11	01	0	0
10	10	00	0	1
11	10	11	0	0

State Diagram:

The information available in a state table may be represented graphically in a state diagram. In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles. The binary number inside each circle identifies the state the circle represents. The directed lines are labeled with two binary numbers represented by a /. The input value that causes the state transition is labeled first; the number after the symbol '/' gives the value of the output during the present state.

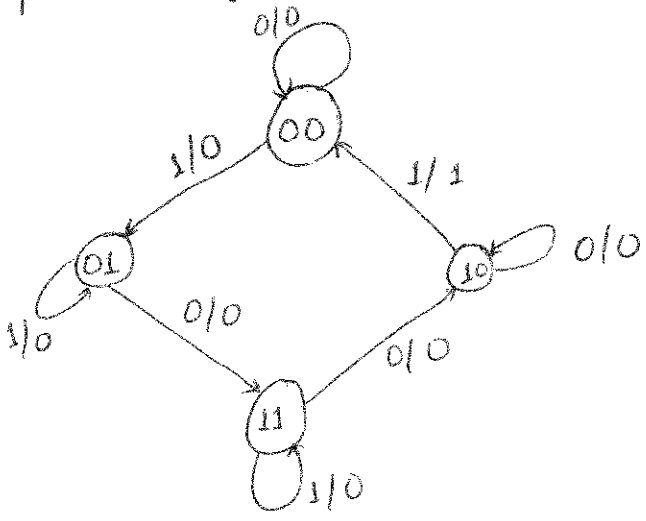


fig state diagram

3. State equation

A state equation (also known as an application equation) is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation denotes the next state of a flip-flop and the right side, a Boolean function that specifies the present state conditions that make the next state equal to 1.

From the state table, we see that

$A = 1$ in next state for combination,

$x = 0$, $AB = 01, 10, 11$

$x = 1$, $AB = 11$

$$\therefore A(t+1) = (A'B + AB' + AB)x' + ABx$$

$B = 1$ in next state for combination,

$x = 0$, $AB = 01$

$x = 1$, $AB = 00, 01, 11$

$$\therefore B(t+1) = x'A'B + x(A'B' + A'B + AB)$$

State reduction

This section discuss certain properties of sequential circuit that may be used to reduce the number of gates and flipflops during the design. The reduction of number of flipflop on the sequential circuit is referred to as state reduction problem. State reduction algorithms are concerned with procedure for reducing the number of states in a state table while keeping the external input output requirement unchanged.

Example

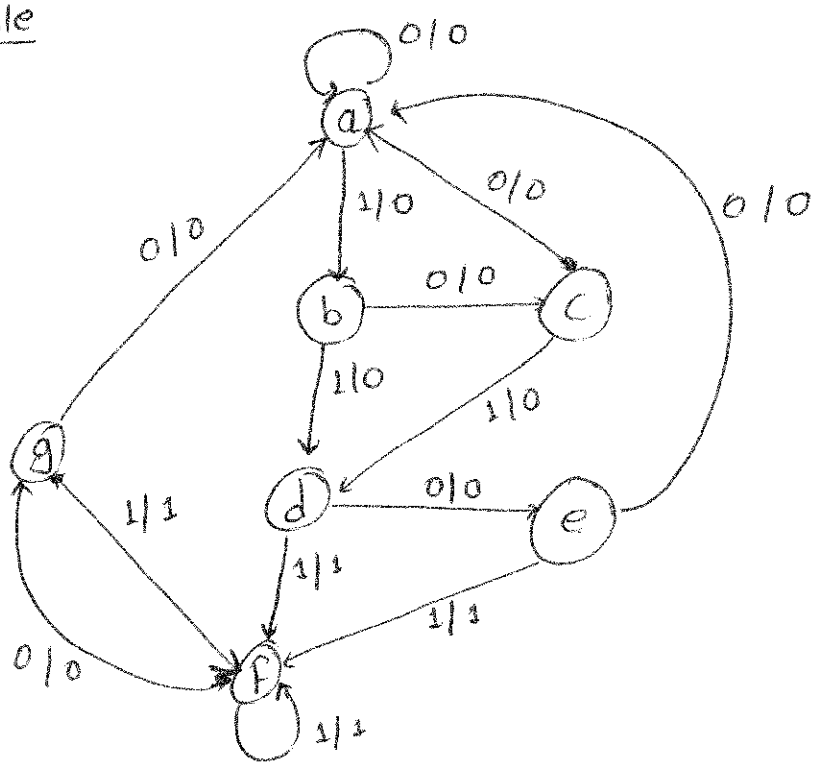


fig: state diagram

state table

present state	next state		output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	F	0	1
g	a	f	0	1

For step reduction:

step 1 : Look for the two present state that goes to same next state and have same output for both input combination.

Example: Stage g and e both goes to state a and f and have outputs 0 for x=0 and 1 for x=1 resp.

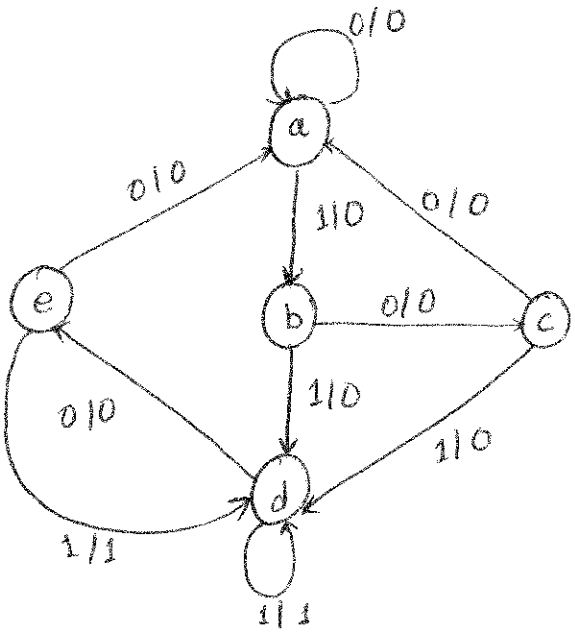
step2: The row with present state g is crossed out and state g is replaced by state e each time it occurs in the next state columns.

step3: Repeat step 1 and 2 until the next two similar step are obtained.

step4: Draw the final state table and state diagram final state table.

present state	next state		output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Final state diagram:



State Assignment:

The cost of the combinational circuit part of a sequential circuit can be reduced by using the known simplification methods for combinational circuits. However, there is another factor, known as the state assignment problem, that comes into play in minimizing the combinational gates.

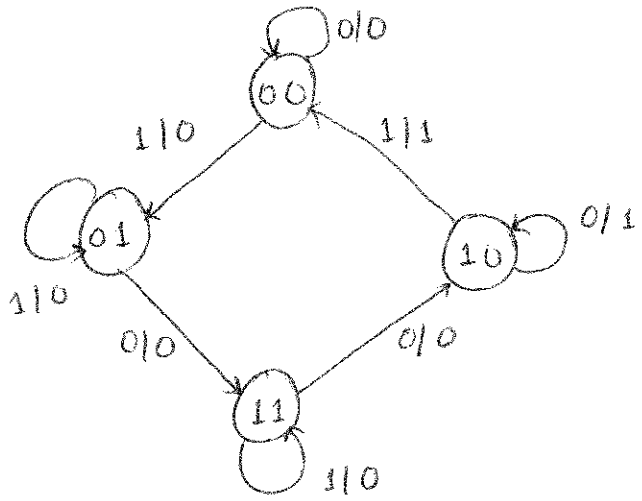
State assignment procedures are concerned with methods for assigning binary values to state in such a way as to reduce the cost of the combinational circuit that drives the flip-flops. This is particularly helpful when a sequential circuit is viewed from its external input-output terminals.

state	Assignment
a	0 0 1
b	0 1 0
c	0 1 1
d	1 0 0
e	1 0 1

Fig. Binary state assignment

present state	next state		output	
	x=0	x=1	x=0	x=1
001	0 0 1	0 1 0	0	0
010	0 1 1	1 0 0	0	0
011	0 0 1	1 0 0	0	0
100	1 0 1	1 0 0	0	1
101	0 0 1	1 0 0		

Example: For the state diagram given in figure, draw the clocked sequential circuit using T flip-flops.



soln:

i) state table

present state		Next state		output	
		x = 0	x = 1	x = 0	x = 1
A	B	A_{n+1} B_{n+1}	A_{n+1} B_{n+1}		
0	0	0 0	0 1	0	0
0	1	1 1	0 1	0	0
1	0	1 0	0 0	1	1
1	1	1 0	1 1	0	0

ii) The circuit goes through the four states. Therefore we need to use 2 flip-flops.

iii) Circuit excitation table

present state		Input	Next state		FF input		output
A	B	x	A_{n+1}	B_{n+1}	T_A	T_B	Y
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	1	0	0	0
1	0	0	1	0	0	0	1
1	0	1	0	0	1	0	1
1	1	0	1	0	0	1	0
1	1	1	1	1	0	0	0

v) K-map

For T_A

A \ BX	00	01	11	10
0	0	0	0	[1]
1	0	[1]	0	0

$T_A = A'BX' + AB'X$

For T_B

A \ BX	00	01	11	10
0	0	[1]	0	0
1	0	0	0	[1]

$T_B = A'B'X + ABX'$

For Y

A \ BX	00	01	11	10
0	0	0	0	0
1	[1]	[1]	0	0

$Y = A\bar{B}$

vi) Logic diagram

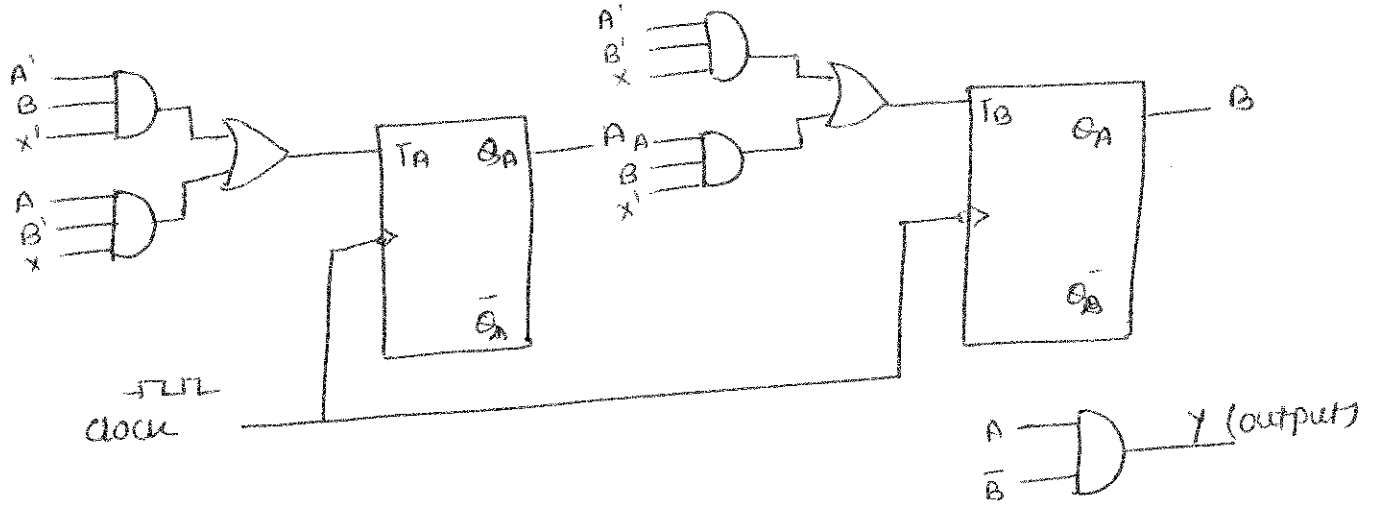


fig: Logic diagram of the required clocked sequential circuit.

- A register is a group of binary storage cells suitable for holding binary information. A group of flip-flops constitutes a register, since each flip-flop is a binary cell capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing any binary information containing n bits. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.

Counters :

A counter is essentially a register that goes through a predetermined sequence of states upon the application of input pulses. The gates in a counter are connected in such a way as to produce a prescribed sequence of binary states in the register.

Memory :

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of storage. A random-access memory (RAM) differs from a read-only memory (ROM) in that a RAM can transfer the stored information out (read) and is also capable of receiving new ~~important~~ information in for storage (write). A more appropriate name for such a memory would be read-write memory.

Registers

A group of flip-flops sensitive to pulse duration is usually called a latch, where as a group of flip-flops sensitive to pulse transition is called a register.

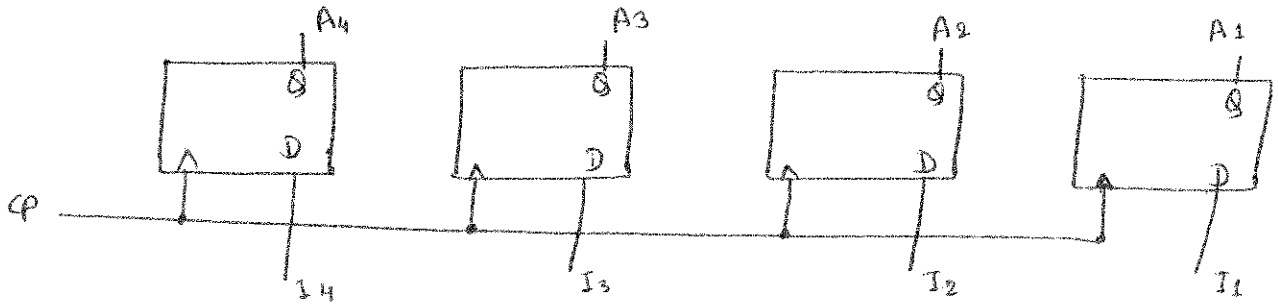


fig: 4-bit register

Figure shows the register constructed with 4 D-type flip-flops and a common clock pulse input. The clock pulse input, CP, enables all flip-flops so that the information presently available at the four inputs can be transferred into the 4-bit register. The four outputs can be sampled to obtain the information presently stored in the register.

Register with parallel load:

The transfer of new information into a register is referred to as loading the register. If all the bits of the register are loaded simultaneously with a single clock pulse, we say that the loading is done in parallel.

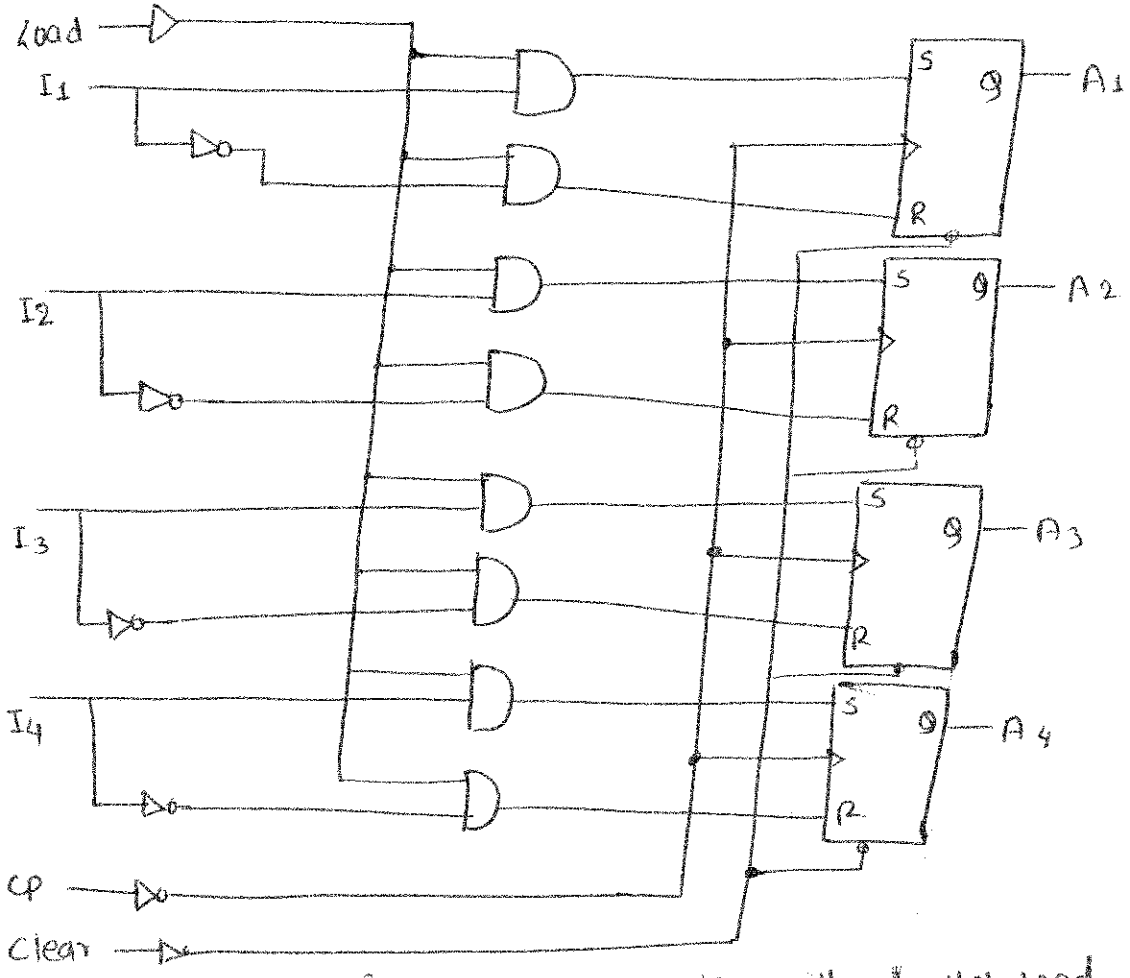


fig: 4-bit register with parallel load

A 4-bit register with a load control input using RS flip-flop is shown in figure above. The CP input of the register receives continuous synchronized pulses which are applied to all flip-flops. The inverter in the CP path causes all flip-flops to be triggered by the negative edge of the incoming pulses.

Shift register

A register capable of shifting its binary information either to the right or to the left is called a shift register. The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive a common clock pulse which causes the shift from one stage to the next.

The simplest possible shift register is one that uses only flip-flops, as shown in figure below. The Q output of a given flip-flop is connected to the D input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right. The serial input determines what goes into the leftmost flip-flop during the shift. The serial output is taken from the output of the rightmost flip-flop prior to the application of a pulse.

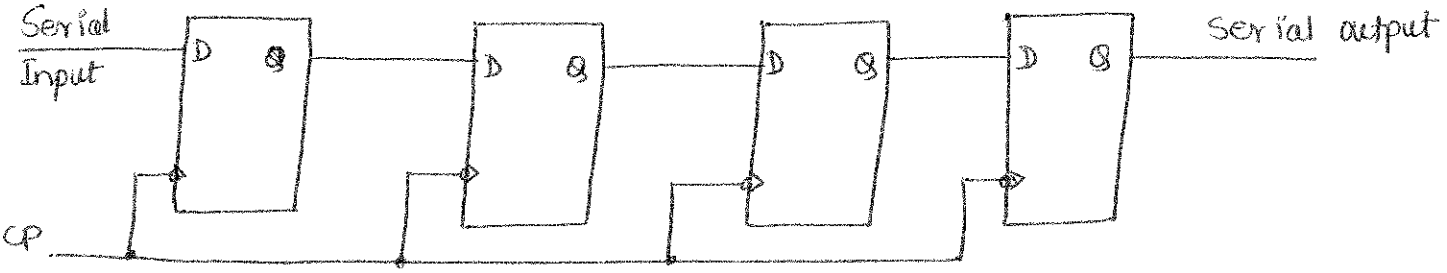


Fig: 4-bit right shift register

Bidirectional shift Register with parallel Load :

A register capable of shifting both right and left is called a bidirectional shift register. One that can shift in only one direction is called a unidirectional shift register. If the register has both shift and parallel-load capabilities, it is called a shift register with parallel load.

S_1 and S_0 inputs in the figure, control the mode of operation of the register.

mode control		Register operation
S_1	S_0	
0	0	No change
0	1	shift right

1 0 shift left

1 1 Parallel load

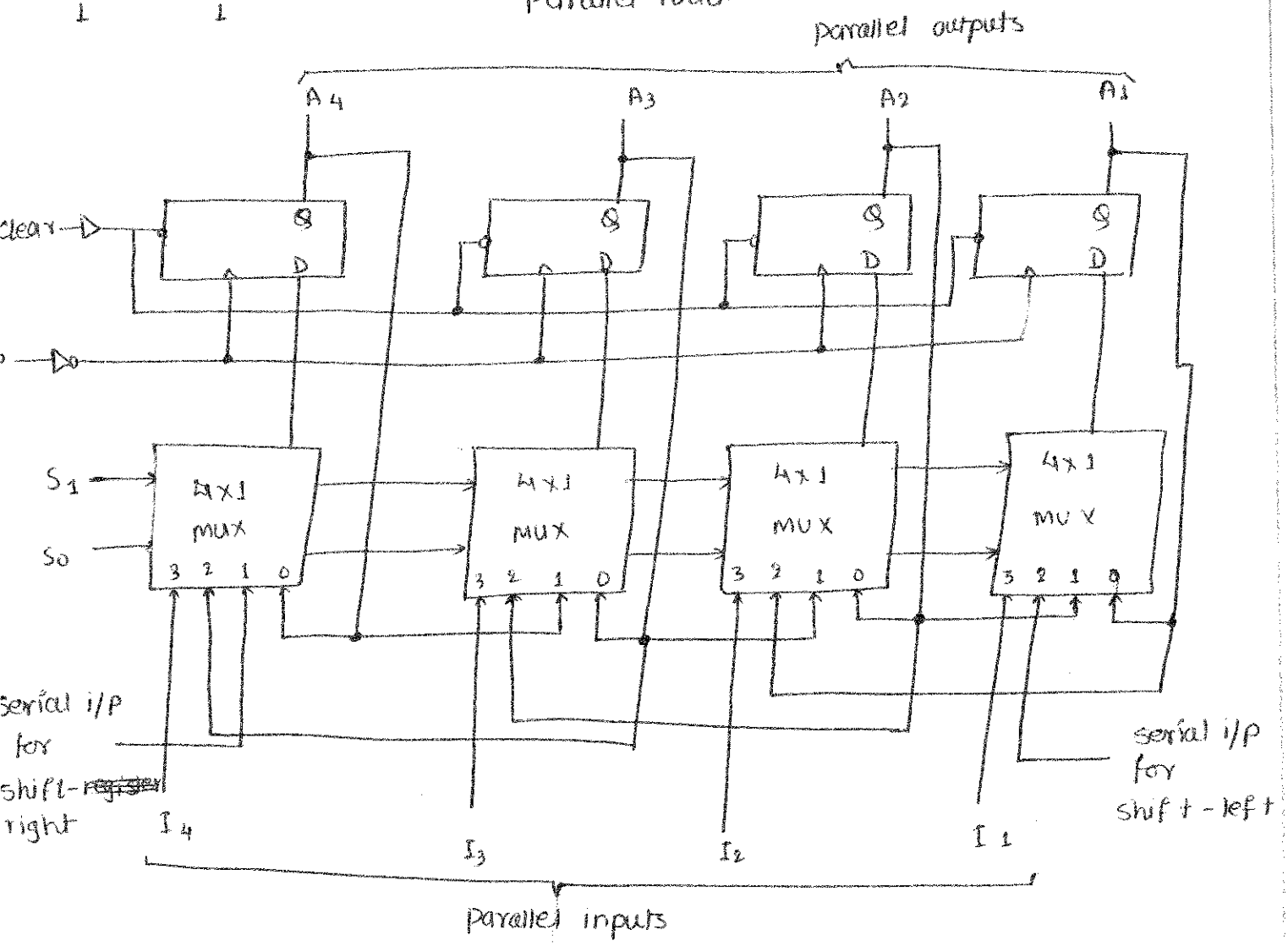


fig: 4-bit bidirectional shift register with parallel load

Modes of operation of a shift Register :

The various modes in which a shift register can be operate may be listed as under:

i) serial input serial output
ii) serial input parallel output
iii) Parallel in serial out
iv) parallel in parallel out

1) serial in serial out (SISO)

It accepts data serially and produce the information on its output in serial basis.

Let us consider that all the flip-flops initially are in the reset condition. This means that $Q_3 = Q_2 = Q_1 = Q_0 = 0$. Let us illustrate the

entry of a four bit binary number 1 1 1 1 into the register. When this is to be done, this number must be applied to Din bit-by-bit with the LSB bit applied first. This means that the D_{in} of FF-3, i.e. D_3 is connected to serial data input (D_{in}). Output of FF-3 i.e., Q_3 is connected to the next flip-flop, i.e D_2 and so on.

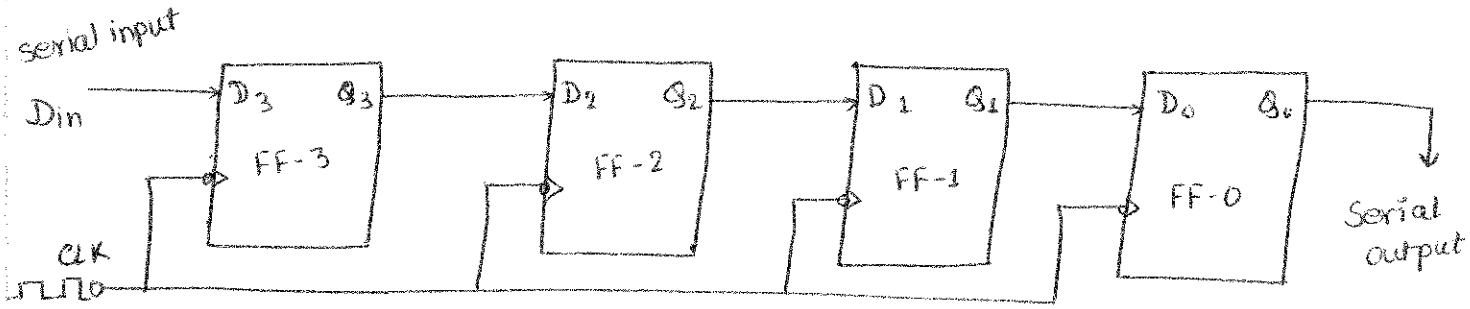


fig Serial shift right register

2. Serial in parallel out [SIPO]

Data is entered serially and then taken out in parallel. In this mode data is loaded bit-by-bit and outputs are disabled as long as the loading is taking place. As soon as the loading is complete, and all the flip-flops consist of their required data, the outputs are enabled so that all the loaded data is made available over all the output lines simultaneously. Also, number of clock cycles required to load a four bit word is 4. Therefore, the speed of operation of SIPO mode will remain same as that of SISO mode.

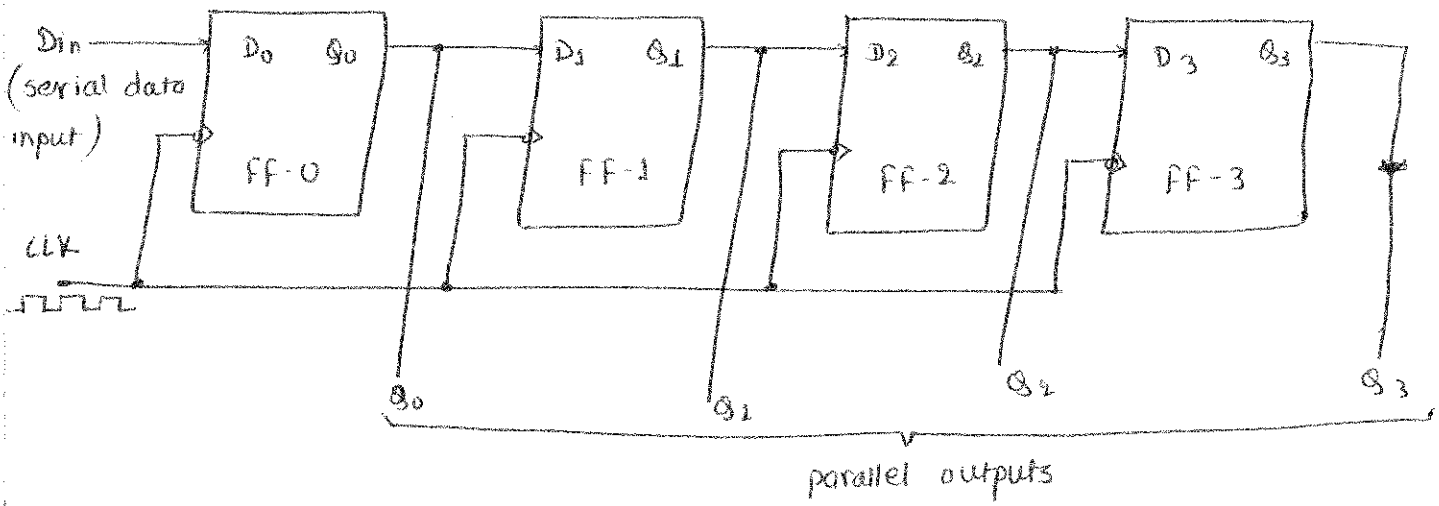


fig: Illustration of serial input parallel output mode

3. Parallel in serial out mode (PISO)

Figure below shows the four bit parallel input serial output register. Output of previous FF is connected to the input of the next one with the help of a combinational circuit. Then, the binary input word B_0, B_1, B_2, B_3 is applied through the same combinational circuit. There are two modes in which this circuit can work. These modes are shift mode and load mode.

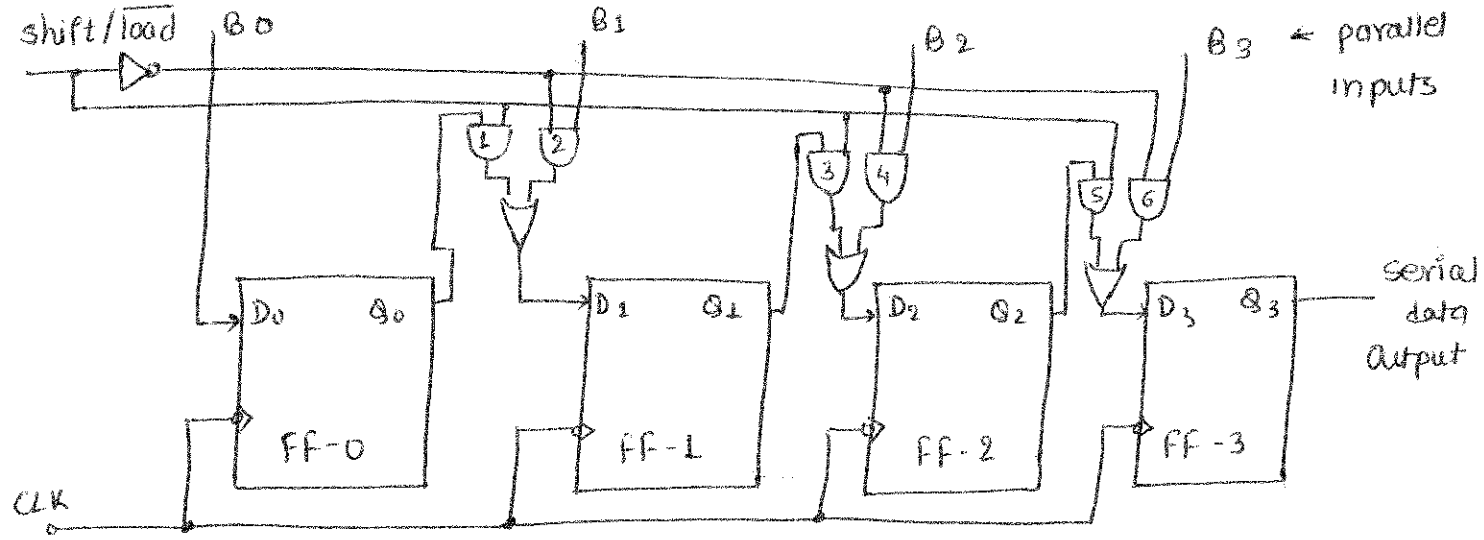


Fig: parallel in serial out shift register

Load mode

When shift/$\overline{load}$ line is low (0), the AND gates 2,4,6 becomes active. They pass B_1, B_2, B_3 bits to the corresponding flip-flops. On low going edge of clock, the Binary inputs $B_0\ B_1\ B_2\ B_3$ get loaded into the corresponding flip-flops. Therefore, parallel loading take place.

shift mode

When the $\overline{shift}$/load line is high(1), the AND gate 2,4,6 become inactive. Hence, the parallel loading of the data becomes impossible. But, the AND gates 1,3,5 becomes active. Therefore, the shifting of data from left to right bit-by-bit on application of clock pulses. Thus, the parallel in serial out operation takes place.

4. parallel in Parallel out (PIPO)

The 4-bit binary input B_0, B_1, B_2, B_3 is applied to the data inputs D_0, D_1, D_2 and D_3 respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits shall be loaded into the flip-flops

simultaneously. The loaded bits appear simultaneously to the output side. Here, only one clock pulse is essential to load all the bits.

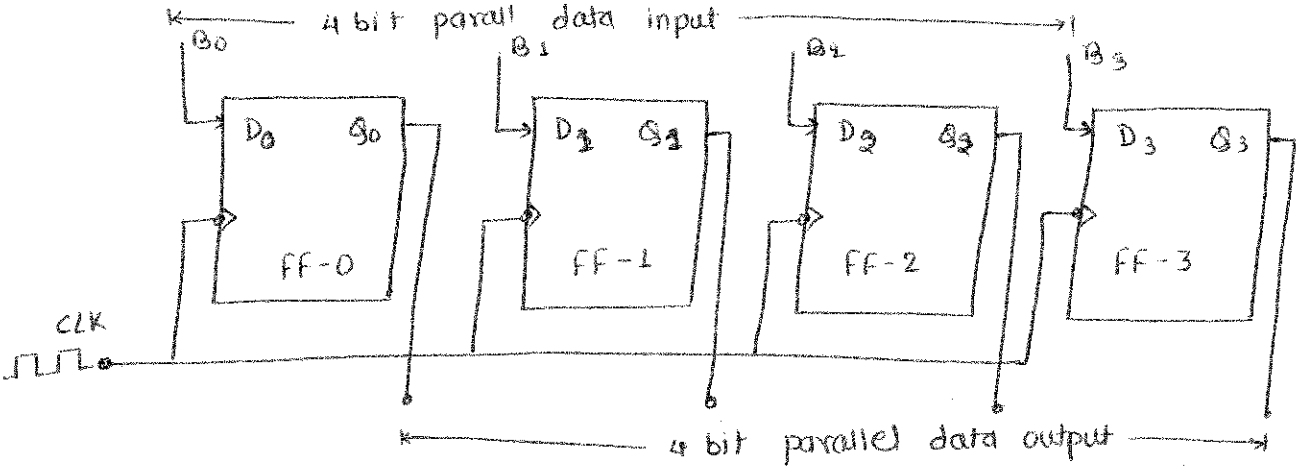


fig. Parallel in parallel out shift register

Counters

MSI counters come in two categories: ripple counters and Synchronous counters.

In a ripple counter, the flip-flop output transition serves as a source for triggering other flip-flops. In other words, the CP inputs of all flip-flops (except the first) are triggered not by the incoming pulses but rather by the transition that occur in other flip-flops.

In a synchronous counter, the input pulses are applied to all CP inputs of all flip-flops. The change of state of a particular flip-flop is dependent on the present state of other flip-flops.

Ripple Counters

The flip-flops change one at a time in rapid succession, and the signal propagates through the counter in a ripple fashion. Ripple counters are sometimes called asynchronous counter.

A binary ripple counter consists of a series connection of complementing flip-flops (T or JK type), with the output of each flip-flop connected to the CP input of the next higher-order flip-flop.

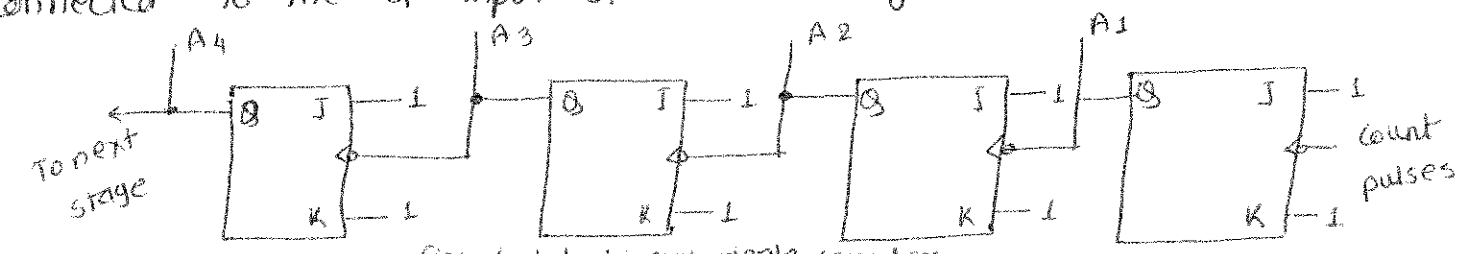


fig: 4-bit binary ripple counter

BCD Ripple counter

A decimal counter follows a sequence of ten states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits.

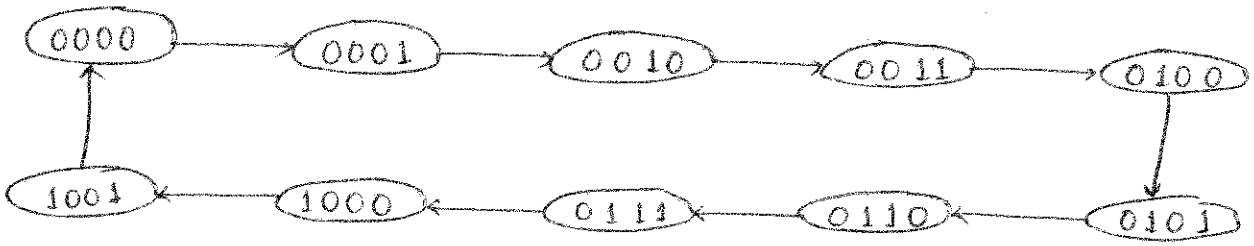


Fig: state diagram of a decimal BCD counter

The following are the conditions for each flip-flop state transition:

1. Q_1 is complemented on the negative edge of every count pulse.
2. Q_2 is complemented if $Q_8 = 0$ and Q_1 goes from 1 to 0. Q_2 is cleared if $Q_8 = 1$ and Q_1 goes from 1 to 0.
3. Q_4 is complemented when Q_2 goes from 1 to 0.
4. Q_8 is complemented when $Q_4 Q_2 = 11$ and Q_1 goes from 1 to 0. Q_8 is cleared if either Q_4 or Q_2 is 0 and Q_1 goes from 1 to 0.

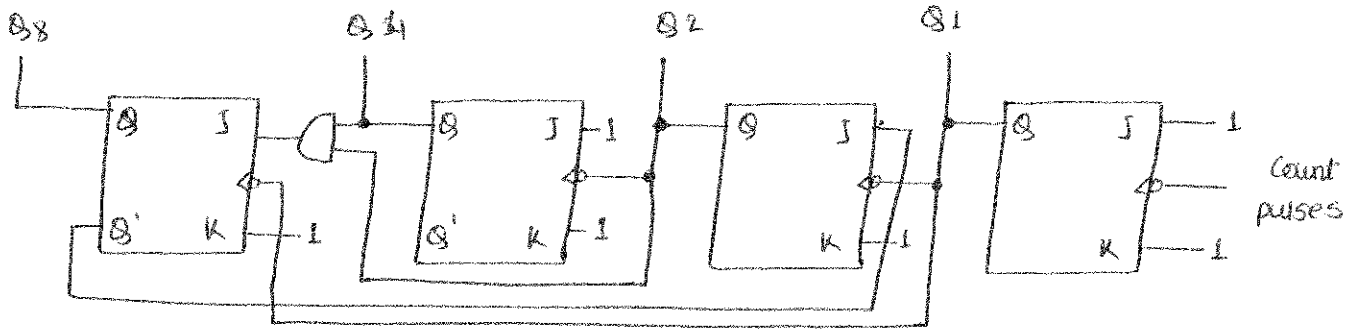


Fig: Logic diagram of a BCD ripple counter [BCD decade counter].

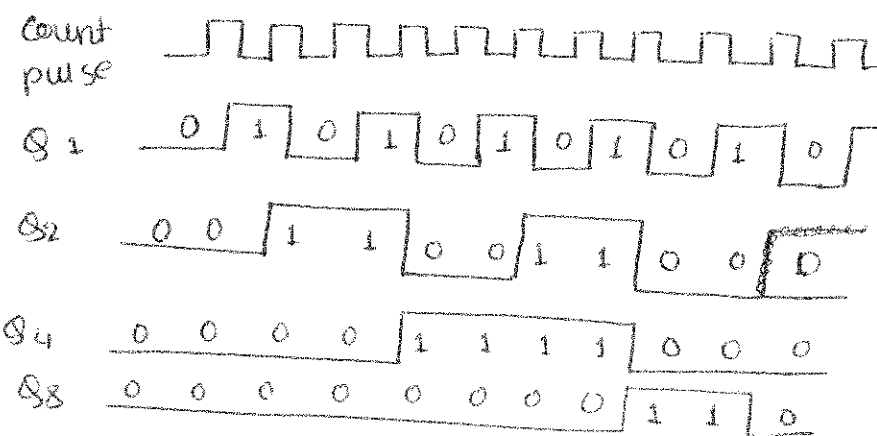


Fig. Timing diagram for the decimal counter

Synchronous Counters:

Synchronous counters are distinguished from ripple counters in that clock pulses are applied to the cp inputs of all flip-flops. The common pulse triggers all the flip-flops simultaneously, rather than one at a time in succession as in a ripple counter.

Types:

1. Binary counter
2. Binary Up-Down Counter
3. BCD decade synchronous counter

1. Binary synchronous counter

It is a counter which count binary sequence rather than BCD sequence. For eg. we can use 3-bit binary counter to count from 0 to 7 decimal number.

Design

1. State diagram

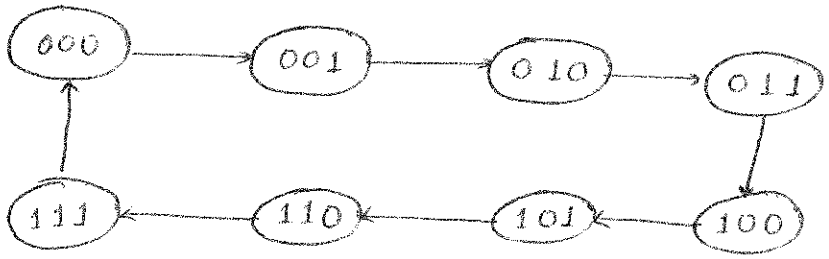


fig: state diagram of a 3 bit binary counter

2. Excitation table for T-flip flop input

Count Sequence			Flip-flop I/P		
A_3	A_2	A_1	TA_3	TA_2	TA_1
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

3. K-map

K-map for T_{A3}

A_3 \ A_2A_1	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$T_{A3} = A_2 A_1$

K-map for T_{A2}

A_3 \ A_2A_1	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$T_{A2} = A_1$

and, $T_{A1} = 1$.

4. Final diagram

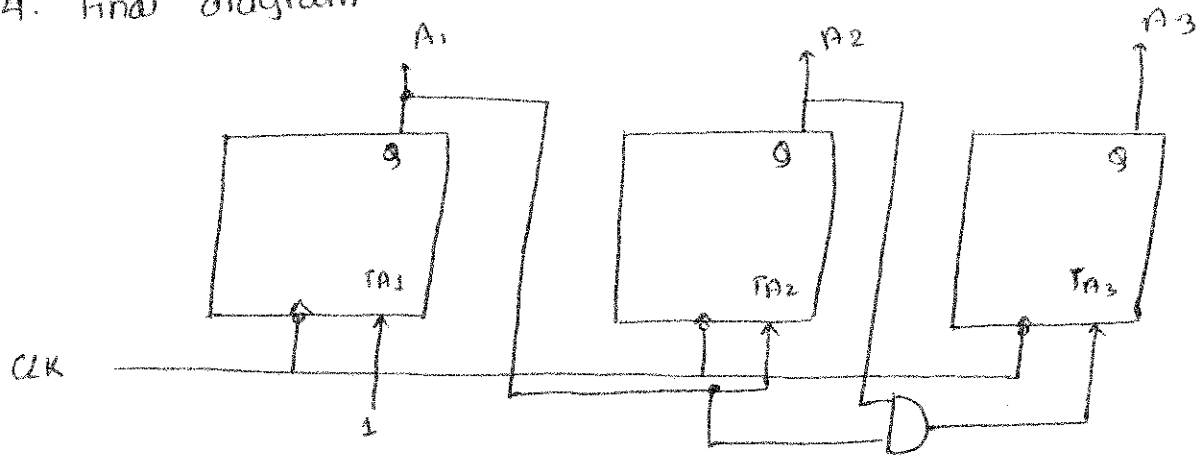


Fig: 3-bit synchronous binary counter

Binary up-down counter :

A binary counter capable of counting either up or down is shown in figure below. When the up input control is 1, the circuit counts up, since the T inputs are determined from the previous values of the normal outputs in Q. When the down input control is 1, the circuit counts down. Since the complement outputs Q' determine the states of the T inputs. When both the up and down signals are 0's the register does not change state but remains in the same count.

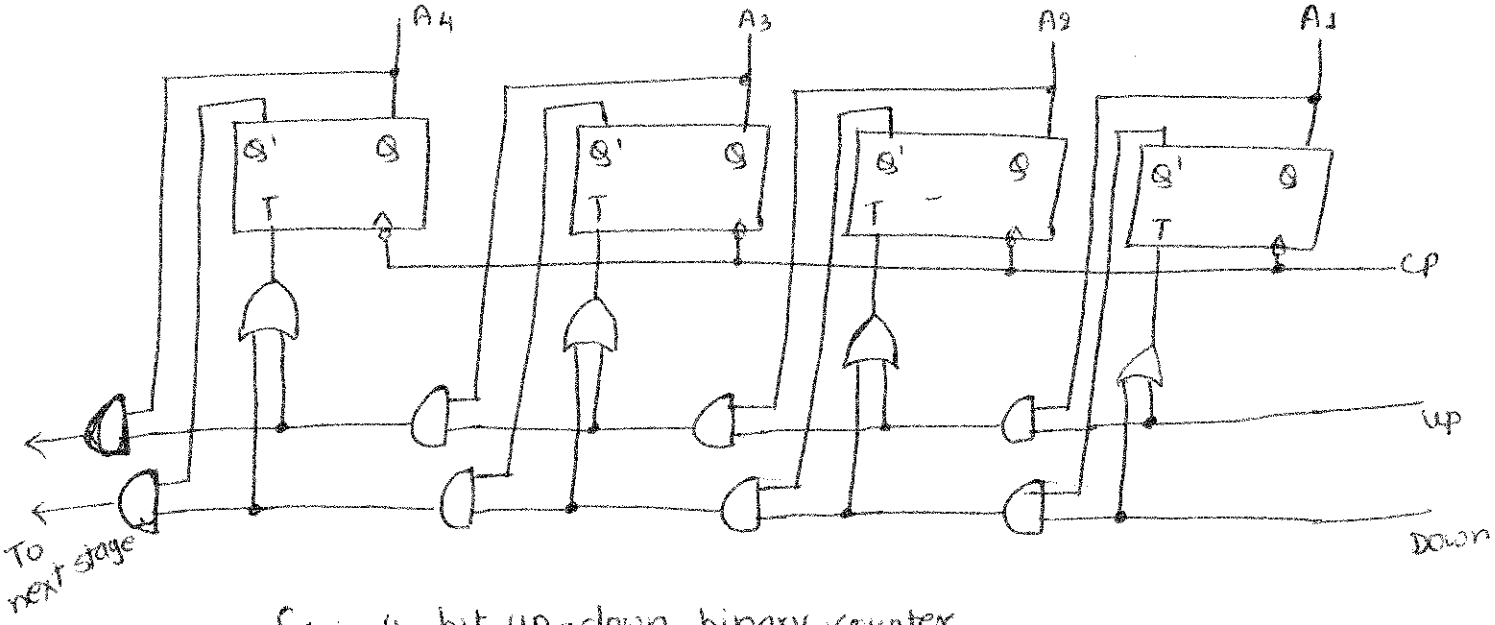


Fig: 4-bit up-down binary counter

3. BCD decade synchronous counter

A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000.

i) state diagram

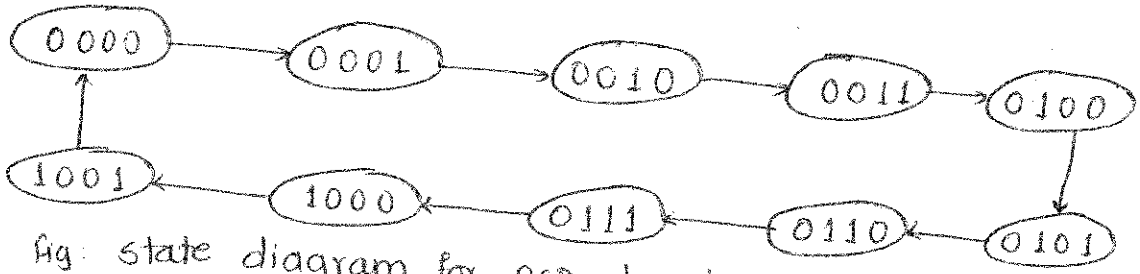


Fig: state diagram for BCD decade synchronous counter

ii) Excitation table for T flip-flop inputs

Counter sequence				Flip-Flop i/p			
A_4	A_3	A_2	A_1	TA_4	TA_3	TA_2	TA_1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	0	0	0	1
0	0	1	1	0	1	1	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	0	1

1	0	0	1	1	0	0	1

iii) K-map

K-map for TA_4

A_4A_3 \ A_2A_1	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	x	x	x	x
10	0	1	x	x

$TA_4 = A_4A_1 + A_1A_2A_3$

K-map for TA_3

A_4A_3 \ A_2A_1	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	x	x	x	x
10	0	0	x	x

$TA_3 = A_2A_1$

K-map for TA_2

A_4A_3 \ A_2A_1	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	x	x	x	x
10	0	0	x	x

$TA_2 = A_4'A_1$

K-map for TA_1

A_4A_3 \ A_2A_1	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	x	x	x	x
10	1	1	x	x

$TA_1 = 1$

iv) Final logic diagram

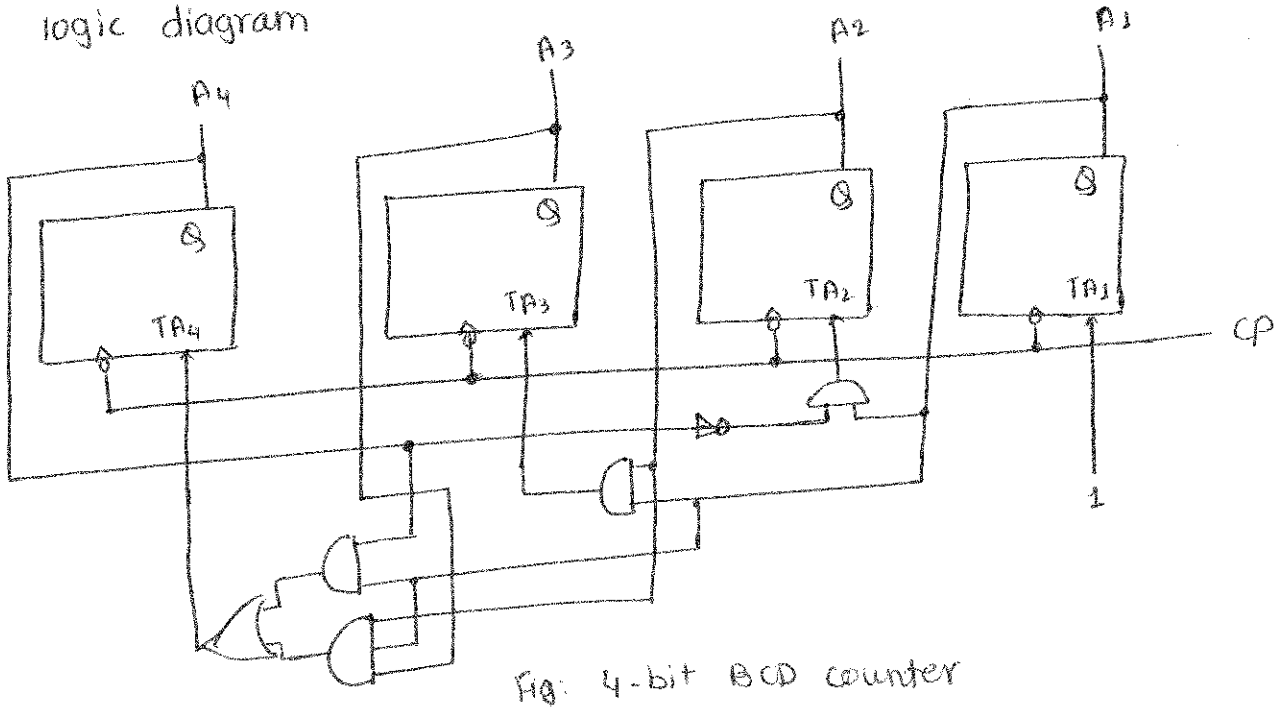


Fig: 4-bit BCD counter

A 3 bit up / Down synchronous counter

i) Circuit excitation table

Here, M is mode select. M = 0 for up counting and M = 1 for down counting.

Mode Control	Present state			Next state			Flip flop input		
M	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

ii) K-map

K-map for T_C

MQ_C \ Q_BQ_A	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	0	0	0
10	1	0	0	0

$T_C = M'Q_BQ_A + MQ_B'Q_A'$

For T_B

$Q_C Q_B$ \ MQ_A	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

$T_B = \bar{M} Q_A + M \bar{Q}_A$

$= M \oplus Q_A$

For T_A

MQ_C \ $Q_B Q_A$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$T_A = 1$

(ii) Logic diagram

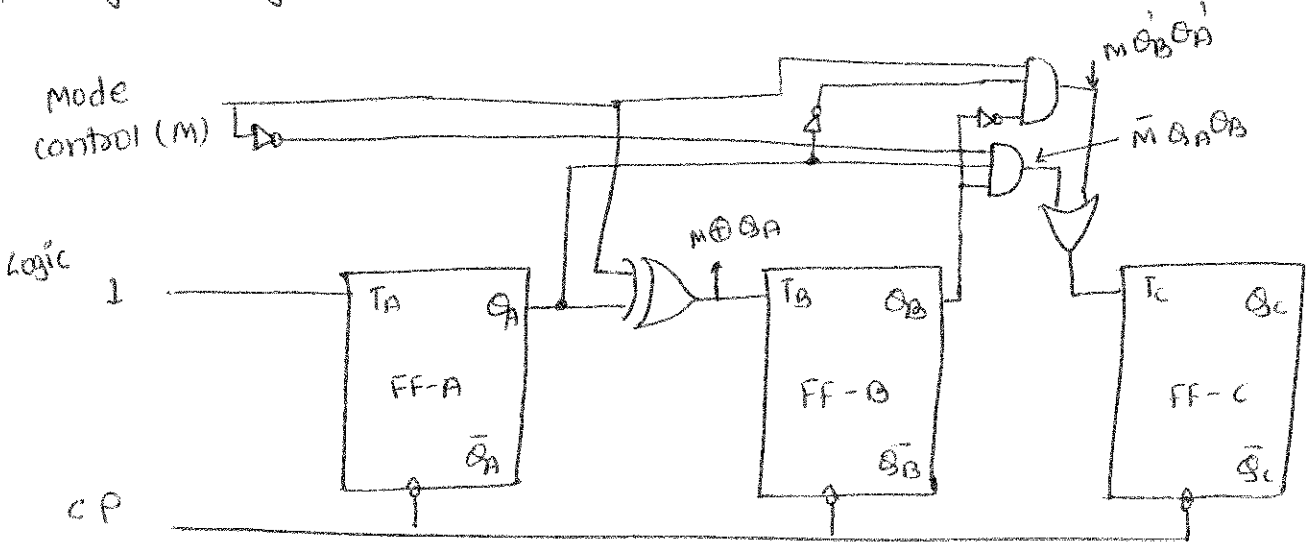


Fig: Logic diagram of a 3-bit synchronous up/down counter

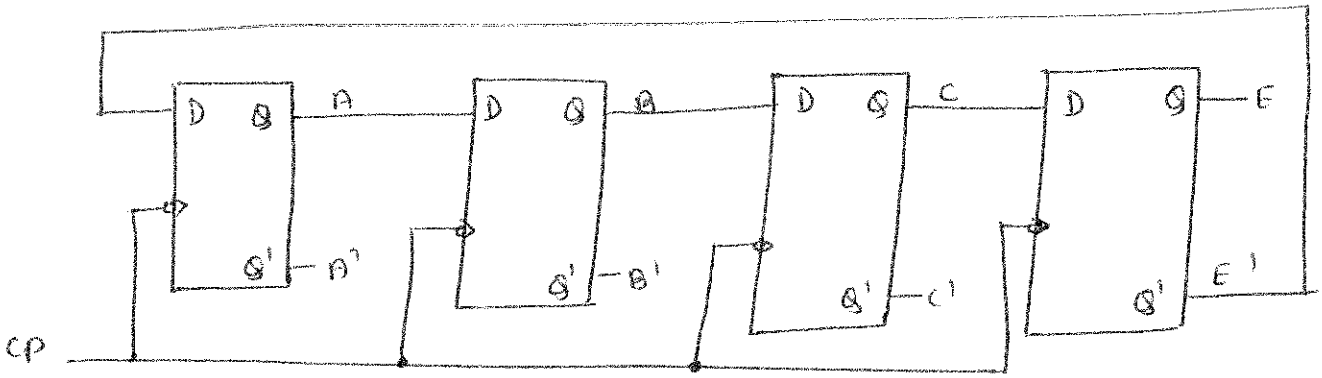
Johnson Counter

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

In general, a k-bit switch-tail ring counter will go through a sequence of 2k states. Starting from all 0's, each shift operation inserts 1's from the left until the register is filled with all 1's. In the

Following sequence, 0's are inserted from the left until the register is again filled with all 0's.

The all-0's state is decoded by taking the complement of the two extreme flip-flops outputs. The all-1's state is decoded by taking the normal outputs of the two extreme flip-flops. All other states are decoded from the adjacent 1,0 or 0,1 pattern in the sequence.



a) 4-stage johnson counter

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

b) Count sequence and required decoding

Fig. Construction of a johnson counter

The memory unit

A memory unit is a collection of storage registers together with the associated circuits needed to transfer information in and out of the registers. The storage registers in a memory unit are called memory registers.

The component that forms the binary cells of registers in a

memory unit must have certain basic properties, the most important of which are:

1) It must have a reliable two-state property for binary representation
2) It must be small in size
3) The cost per bit of storage should be as low as possible
4) The time of access to a memory register should be reasonably fast

examples: magnetic cores, Semiconductor ICs, drums, disks etc.

A memory unit stores binary information in groups called words, each word being stored in a memory register.

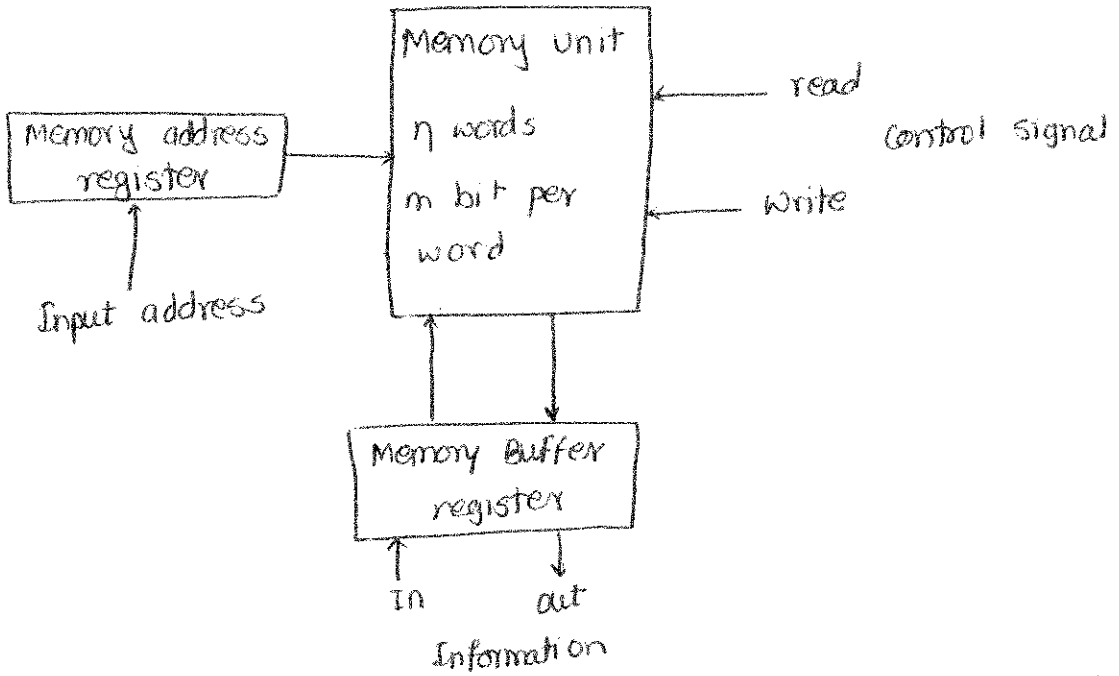


fig: Block diagram of a memory unit showing communication with environment

Memory address register

The memory address register specifies the memory word selected. To communicate with a specific memory word, its location number, or address is transferred to the address register. An address register with n bits can specify up to 2^n memory words.

Control signal

The two control signals applied to the memory unit are called read and write. A write signal specifies a transfer-in function; a read control signal specifies a transfer-out function.

Memory Buffer register

The information transfer to and from register in memory

and the external environment is communicated through one common register called the memory buffer register [information register or storage register].

Read and write operation

The sequence of operations needed to communicate with the memory unit for the purpose of transferring a word out to the MBR is:

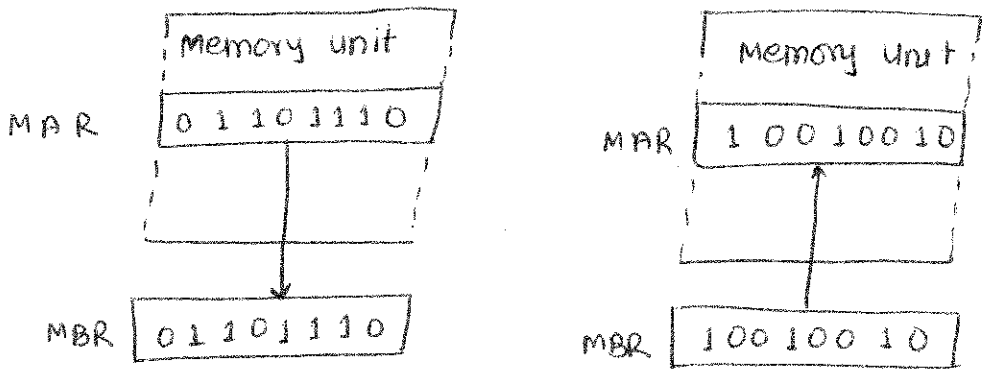
1. Transfer the address bits of the selected word into MAR
2. Activate the read control unit

The binary information presently stored in memory register is transferred into MBR.

The sequence of operations needed to store a new word into memory is:

1. Transfer the address bits of the selected word into MAR
2. Transfer the data bits of the word into MBR
3. Activate the write control input.

The data bits from MBR are stored in memory register.

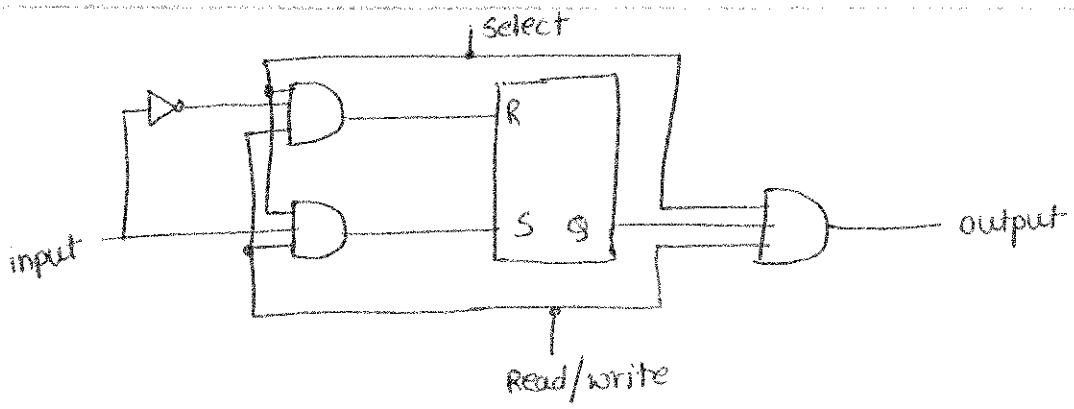


a) Read operation b) write operation

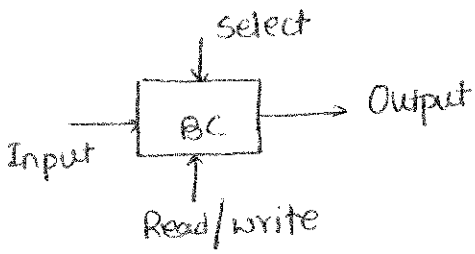
fig: Information transfer during read and write operations

Integrated circuit memory

The internal construction of a random-access memory of m words with n bits per word consists of mxn binary storage cells and the associated logic for selecting individual words. The binary storage cell is the basic building block of a memory unit.



a) Logic diagram



b) Block diagram

Fig: Memory cell

IC RAMs are constructed internally with cells having a wired-OR capability. Figure shows a IC RAM which consists of 2 words of 3 bits each, for a total of 6 binary cells.

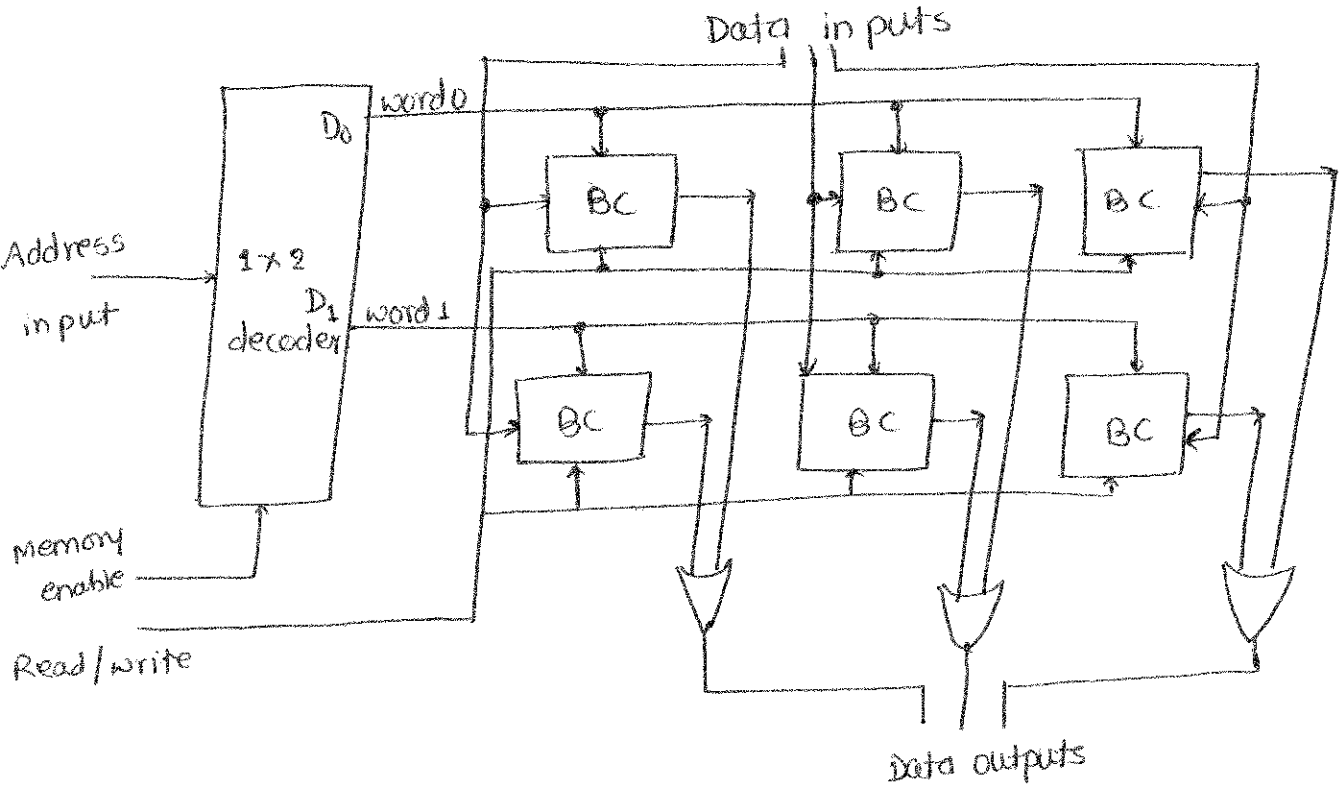


Fig: Integrated-circuit memory

Error - correcting code:

An error - correcting code (ECC) or forward error correction (FEC) code is a process of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors [up to the capability of the code being used] were introduced, either during the process of transmission, or on storage. Since the receiver does not have to ask the sender for retransmission of the data, a backchannel is not required in forward error correction, and it is therefore suitable for simplex communication such as broadcasting. Error correcting codes are frequently used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks and RAM.

Output hazards races:

Race hazard is the behavior of an electronic system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended.

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.

Memory decoding:

- A microprocessor with n address line has a 2^n address memory space
- physical memory space is the actual amount of memory implemented in terms of memory modules (chips).

Memory map : In a computer system the physical address space is allocated to various memory chips and other subsystems.

Memory Decoding places the block of memory in a module into a particular location in the address space

- Partial address decoding uses only some of the remaining address lines to drive the CS.
- Full address decoding uses all of the remaining address lines to drive the CS signal.

An arithmetic logic unit (ALU) is a multioperation, combinational-logic digital function. It can perform a set of basic arithmetic operations and a set of logic operations. The ALU has a number of selection lines to select a particular operation in the unit. The selection lines are decoded within the ALU so that k selection variables can specify up to 2^k distinct operations.

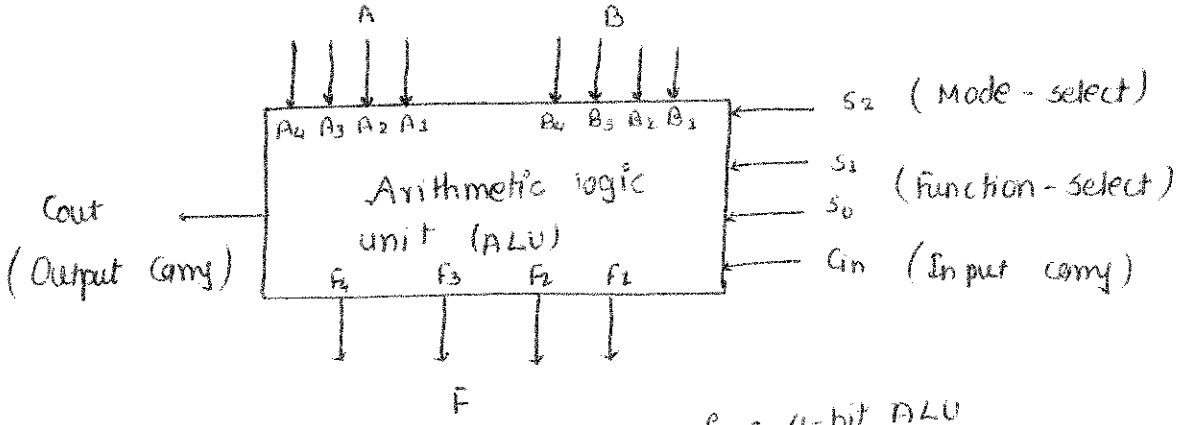


Fig: Block diagram of a 4-bit ALU

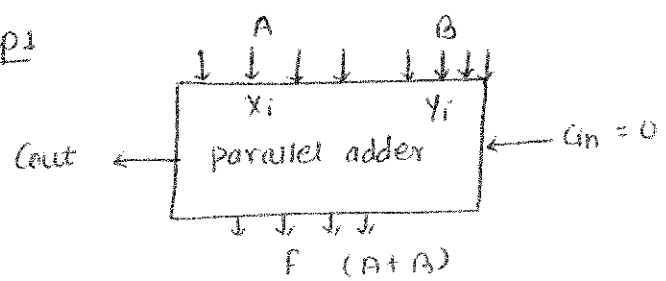
The four data inputs from A are combined with the four inputs from B to generate an operation at the F outputs. The mode select input S_2 distinguishes between arithmetic and logic operations. The two function select inputs S_1 and S_0 specify the particular arithmetic or logic operation to be generated. With three selection variables, it is possible to specify four arithmetic operations (with S_2 in one state) and four logic operations (with S_2 in the other state). The input and output carries have meaning only during an arithmetic operation.

Arithmetic circuit design

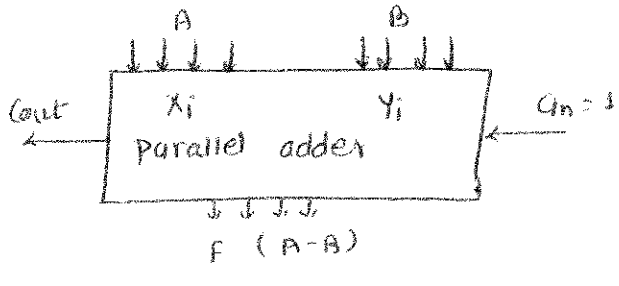
Q. Design and adder subtractor circuit with one selection variables and two inputs A and B where S=0, circuit performs A+B, when S=1, circuit performs A-B.

sol^n:

Step 1



a) addition



b) subtraction

step 2:

S	x_i	y_i	C_{in}
0	A	B	0
1	A	$\bar{B}$	1

Fig: function table

step3:

Truth table

S	A	B	x_i	y_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

K-map for y_i

S \ AB	00	01	11	10
0	0	[1]	[1]	0
1	[1]	0	0	[1]

$y_i = s'B + sB'$

$= s \oplus B$

$x_i = A$

$S = C_{in}$

step 4 Logic diagram

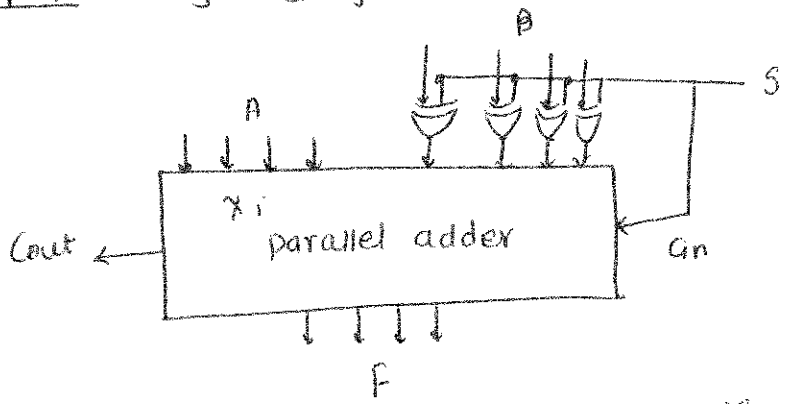


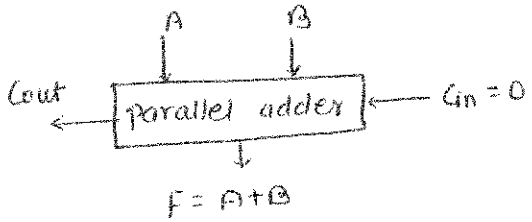
Fig: Logic diagram of arithmetic circuit

8. Design Arithmetic circuit that performs following operations.

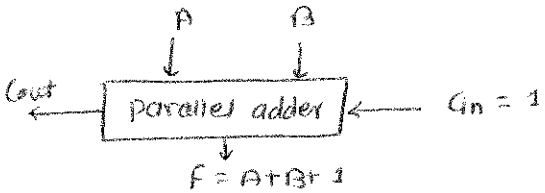
1. Addition (A+B)
2. Addition with carry (A+B+1)
3. A plus 1's complement of B (A+$\bar{B}$)
4. Subtraction (A+$\bar{B}$+1)
5. Transfer A (using B = 0)
6. Increment A (A+1), B = 0
7. Decrement A (A-1), B = 1
8. Transfer A (A), using B = all 1's

Soln:

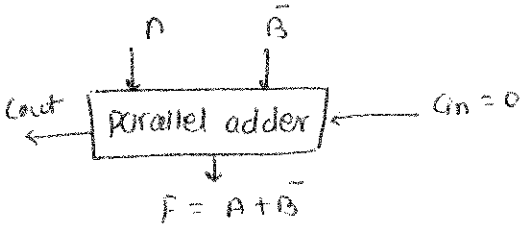
step 1 problem statement



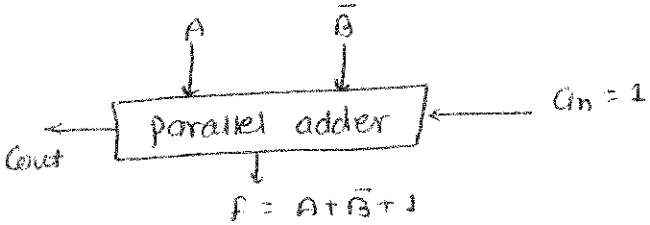
1) Addition



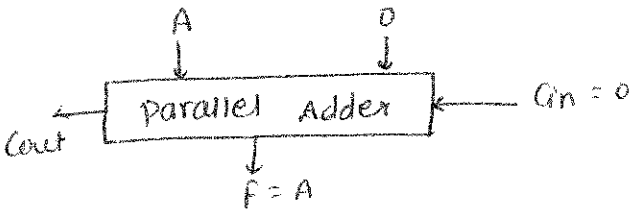
2) Addition with carry



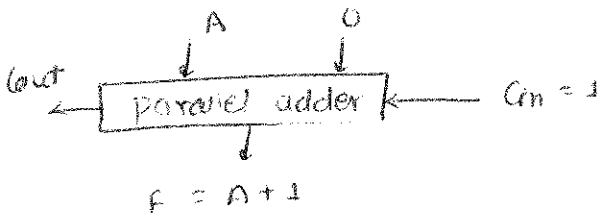
3) A plus 1's complement of B



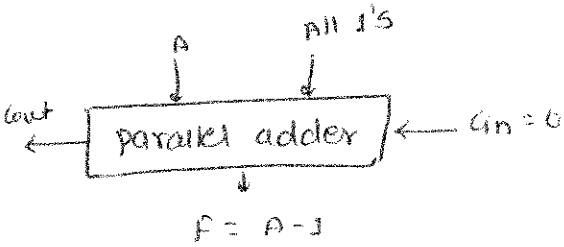
4) subtraction



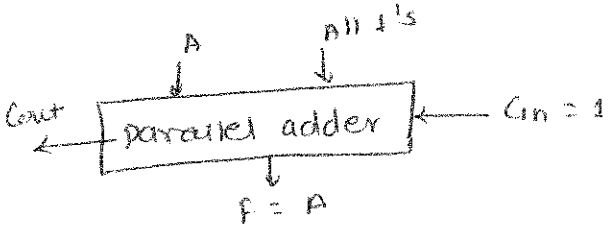
5) Transfer A



6) Increment A



7) Decrement A



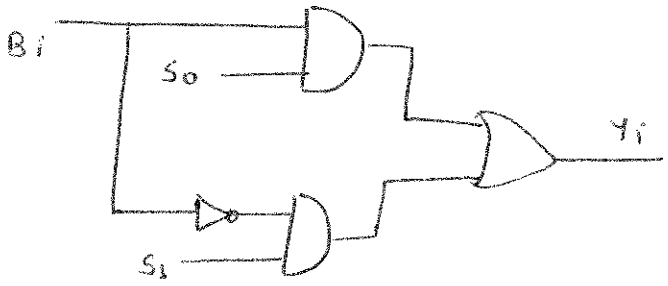
8) Transfer A

Fig: Operations obtained by controlling one set of inputs to a parallel adder.

step 2 Function Table

Function select			X equals	Y equals	Output equals function	Function
S_1	S_0	Cin				
0	0	0	A	0	F = A	Transfer A
0	0	1	A	0	F = A+1	Increment A
0	1	0	A	B	F = A+B	Add B to A
0	1	1	A	B	F = A+B+1	Add B to A plus 1
1	0	0	A	$\bar{B}$	F = A+$\bar{B}$	Add 1's complement of B to A
1	0	1	A	$\bar{B}$	F = A+$\bar{B}$+1	Add 2's complement of B to A
1	1	0	A	All 1's	F = A-1	Decrement A
1	1	1	A	All 1's	F = A	Transfer A

The circuit that controls input B to provide the function illustrate in step 1 is called a true / complement, one / zero element.



S_1	S_0	Y_i
0	0	0
0	1	B_i
1	0	$\overline{B_i}$
1	1	1

step 3 Truth table

S_1	S_0	B_i	Y_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

K- map for Y_i

S_1 \ S_0B_i	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$Y_i = S_0 B_i + S_1 B_i'$

$X_i = A_i$

$G_n =$

step 4 final logic diagram

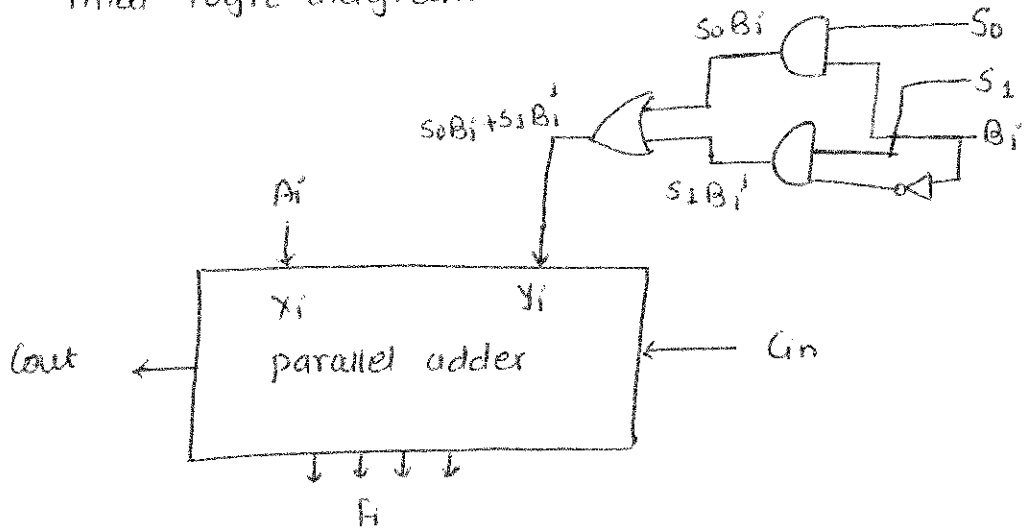


Fig: logic diagram of Arithmetic circuit

Design of Logic unit

The logic microoperations manipulate the bits of the operands separately and treat each bit as a binary variable. With 'n' variables we can create 2^{2^n} functions. If n = 2, we can create 16 function but all these 16 function can be generated using 'AND', 'OR' & 'NOT' operation.

Example :

Design a logic circuit, which can perform following logic operation.

a) OR b) AND c) NOT A g) X-OR

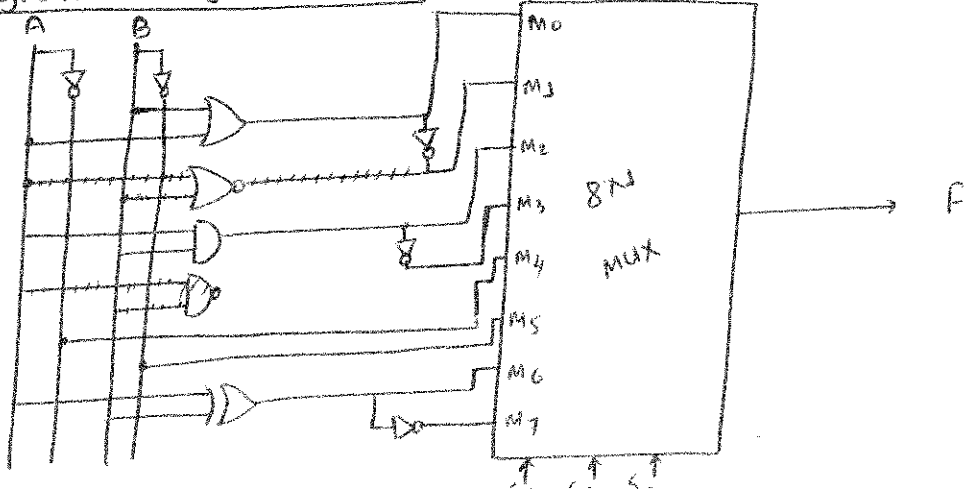
b) NOR d) NAND f) NOT B h) X-NOR

Sol^n: For 8 function we use 3 selection line i.e S_2 S_1 S_0 and Input A, B

Function table

Function select			output equals	Function
S_2	S_1	S_0		
0	0	0	$A + B$	OR
0	0	1	$\overline{A + B}$	NOR
0	1	0	$A \cdot B$	AND
0	1	1	$\overline{A \cdot B}$	NAND
1	0	0	$\bar{A}$	Not A
1	0	1	$\bar{B}$	NOT B
1	1	0	$A \oplus B$	X-OR
1	1	1	$A \odot B$	X-NOR

Logic diagram using 8x1 mux



The circuit must be repeated 'n' times for n bit logic circuit. The figure above generates 8 logic operation with 3 selection variables

Design of Arithmetic logic unit:

Here, We design an ALU with eight arithmetic operations and four logic operations. Three selection variables S_2, S_1, and S_0 select eight different operations, and the input carry C_{in} is used to select four additional arithmetic operations. With $S_2 = 0$, selection variables S_2 and S_0 together with C_{in} will select the eight arithmetic operation with $S_2 = 1$, variables S_1 and S_0 will select the four logic operations OR, XOR, AND and NOT.

The steps involved in the design of An ALU are as follows:

1. Design the arithmetic section independent of the logic section.
2. Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carries to all stages are 0.
3. Modify the arithmetic circuit to obtain the required logic operations.

The function Table

selection				output	Function
S_2	S_1	S_0	C_{in}		
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A+1	Increment A
0	0	1	0	F = A+B	Addition
0	0	1	1	F = A+B+1	Add with carry
0	1	0	0	F = A-B-1	subtract with borrow
0	1	0	1	F = A-B	subtraction
0	1	1	0	F = A-1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	x	F = A ∨ B	OR
1	0	1	x	F = A ⊕ B	XOR
1	1	0	x	F = A ∧ B	AND
1	1	1	x	F = $\bar{A}$	complement A

The inputs to each full adder circuit are specified by the Boolean functions.

$$X_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0' B_i'$$

$$Y_i = S_0 B_i + S_1 B_i'$$

$$Z_i = S_2' C_{in}$$

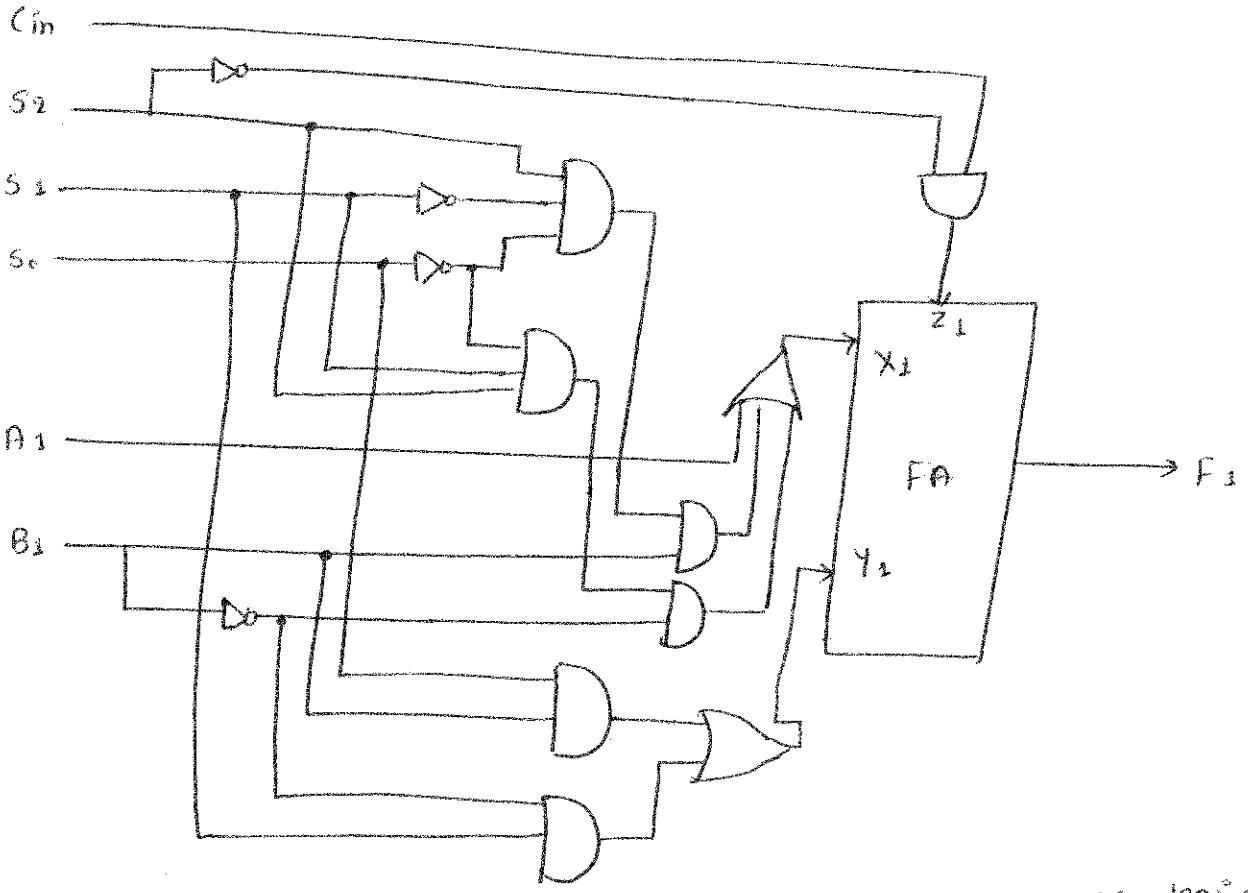


Fig: One bit Logic diagram of arithmetic logic unit (ALU)

Status Register:

It is sometimes convenient to supplement the ALU with a status register where these status-bit conditions are stored for further analysis. Status-bit conditions are sometimes called condition-code bits or flag bits.

Figure below shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z & V. The bits are set or cleared as a result of an operation performed in the ALU.

1. Bit C is set if the output carry of the ALU is 1. It is cleared if the

utput carry is 0.

Bit S is set if the highest-order bit of the result in the output of the ALU is 1. It is cleared if the highest-order bit is 0.

Bit Z is set if the output of the ALU contains all 0's and cleared otherwise.

. Bit V is set if exclusive-OR of carries C_8 and C_9 is 1, and cleared otherwise This is the condition for overflow when the numbers are in sign-2's-complement representation for the 8-bit ALU, V is set if the result is greater than 127 or less than -128.

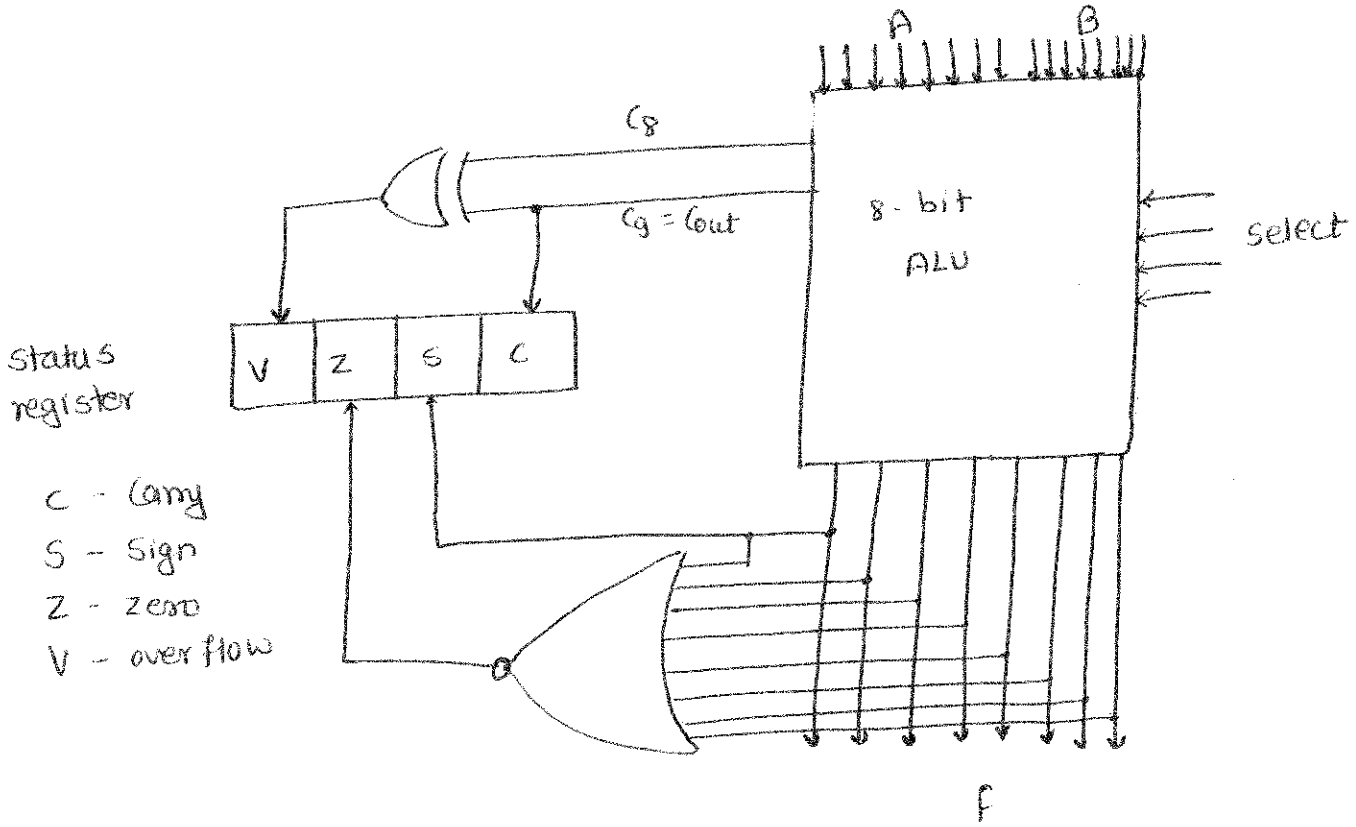


fig: setting bits in a status register

Shifter:

The shift unit attached to a processor transfers the output of the ALU onto the output bus. The shifter may transfer the information directly without a shift, or it may shift the information to the right or left. Provision is sometimes made for no transfer from the ALU to the output bus. The shifter provides the shift micro operations commonly not available in an ALU.

Design:

A combinational logic shifter can be constructed with multiplexers as shown in figure below. The two selection variables, S_1 and S_0, applied to all four multiplexers select the type of operation in the shifter. when $S_1S_0 = 00$, no shift is executed and the signals from A go directly to the F lines. The next two selection variable causes a shift-right operation and a shift-left operation. When $S_1S_0 = 11$, the multiplexers select the inputs attached to 0 and as a consequence the F outputs are also equal to 0, blocking the transfer of information from the ALU to the output bus.

Function table for shifter:

S_1	S_0	operation	function
0	0	$F \leftarrow A$	Transfer A to S (no shift)
0	1	$F \leftarrow shrA$	shift right A into F
1	0	$F \leftarrow shlA$	shift left A into F
1	1	$F \leftarrow 0$	Transfer 0's into S

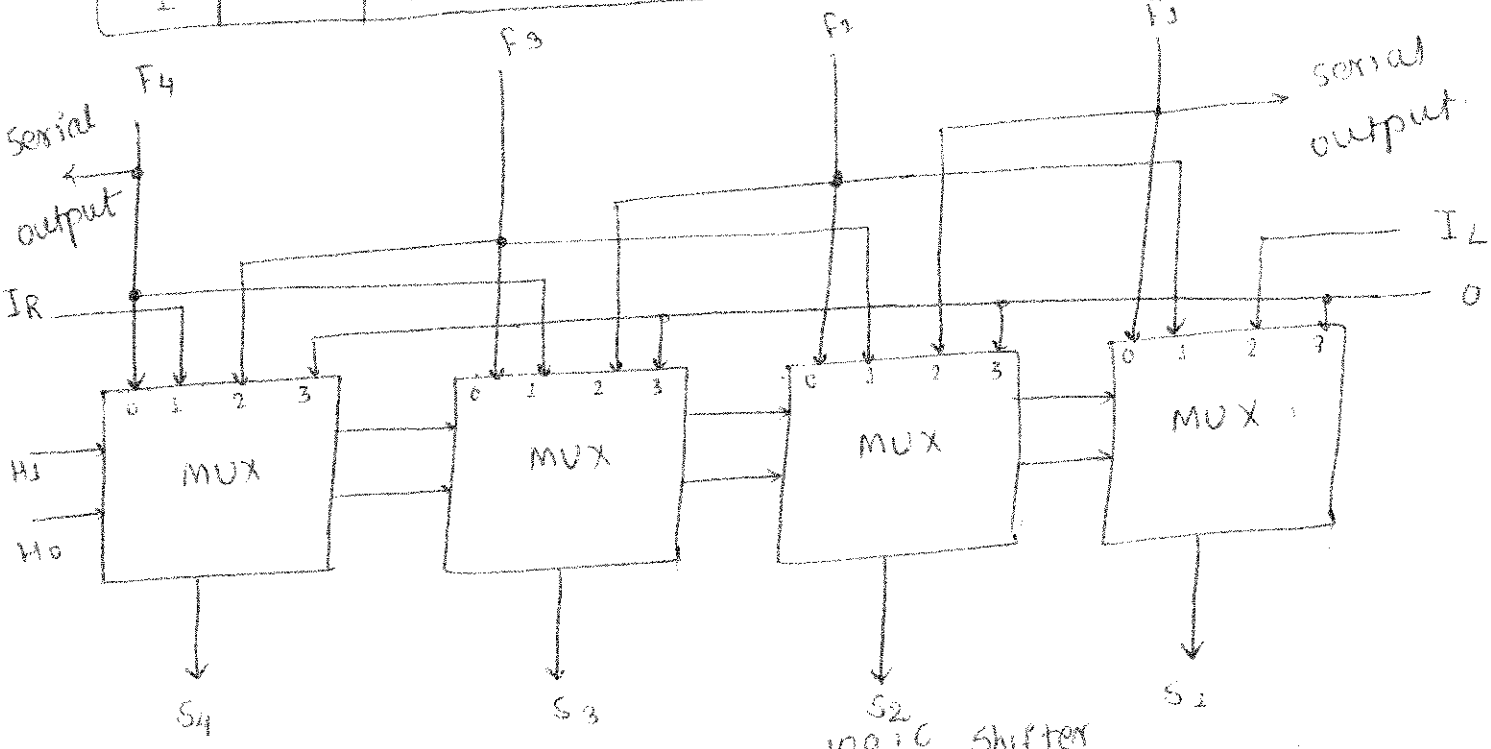


Fig 4-bit combination logic shifter

Accumulator:

Some processor unit distinguish one register from all other and called t accumulator register. Accumulator register is essentially bi-directional hift register with parallel load which is connected to an ALU. Because of eedback connection from the register to the input of ALU, the accumulator egister and its associated logic when taken as one unit constitute a equential circuit. The register A in the figure is referred to as ccumulator and is sometimes denoted by the symbol AC. The external nput to the accumulator are the data inputs and control variables etermine the micro-operation for the register. An accumulator is a ultifunction register that by itself can be made to perform all the icro-operation of processor unit.

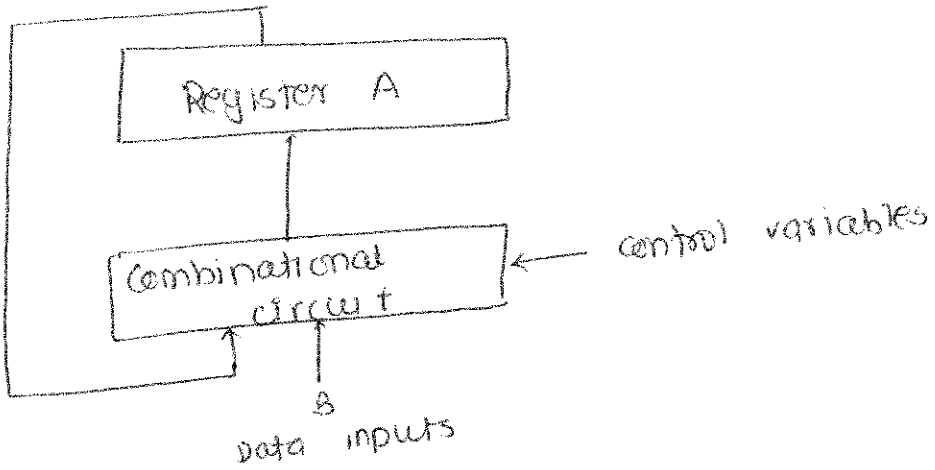


fig: Block diagram of accumulator

Design:

The set of micro-operation for the accumulator is given in table below. Control variable S_1 to S_4 are generated by control logic circuits and should be considered as control functions that initiate the corresponding register transfer operation. Register A is source register and it represents the present state of the sequential circuit. The B register is used as a second source register for microoperation. The four control variables sent to sequential circuit is also considered as input to circuit.

List of microoperations:

Control variable	Micro-operation	Name
S_1	$A \leftarrow A+B$	add
S_2	$A \leftarrow 0$	clear
S_3	$A \leftarrow \bar{A}$	complement
S_4	$A \leftarrow A \cdot B$	AND

① Add $(A+B)$

The excitation Table for inputs for the J-K flip flop are listed below:

Present state	Inputs		Next state	Flip-flop input		output
A_i	B_i	C_i	A_{i+1}	J_{Ai}	K_{Ai}	C_{i+1}
0	0	0	0	0	x	0
0	0	1	1	1	x	0
0	1	0	1	1	x	0
0	1	1	0	0	x	1
1	0	0	1	x	0	0
1	0	1	0	x	1	1
1	1	0	0	x	1	1
1	1	1	1	x	0	1

K-map for J_{Ai}

A_i \ B_iC_i	00	01	11	10
0	0	1	0	1
1	x	x	x	x

$J_{Ai} = B_i'C_i + B_iC_i'$

$= B_i \oplus C_i$

K-map for K_{Ai}

A_i \ B_iC_i	00	01	11	10
0	x	x	x	x
1	0	1	0	1

$K_{Ai} = B_i'C_i + B_iC_i'$

$= B_i \oplus C_i$

K-map for C_{i+1}

A_i \ B_iC_i	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C_{i+1} = B_iC_i + A_iC_i + A_iB_i$

$= A_iB_i + A_iC_i + B_iC_i$

The J input of flip flop Ai, designated J_{Ai}, and K input of flip flop Ai designated by K_{Ai}, do not include the control variable S_1. These two equation should affect the flipflop only when S_1 is enable; therefore they should be ANDed with control veriable S_1. So,

$J_{Ai} = (B_i \oplus C_i) S_1$ $\quad C_i = A_iB_i + A_iC_i + B_iC_i$

$K_{Ai} = (B_i \oplus C_i) S_1$

Clear (S_2)

$J_{Ai} = 0$

$K_{Ai} = S_2$

It clears all the flipflops in register A. To cause this transition in a J-K flipflop, we need to apply control variable S_2 only to K_{Ai}, input of flip flop. The J input will be assumed to be zero if nothing is applied to it.

3. Complement (S_3)

$J_{Ai} = S_3$

$K_{Ai} = S_3$

for this we need to apply S_3 to both J and K input such that $J_{Ai} = S_3$ and $K_{Ai} = S_3$

1. AND (S_4)

present state		Next state	Flip - Flop	
		A_{i+1}		
A_i	B_i		J_{Ai}	K_{Ai}
0	0	0	0	x
0	1	0	0	x
1	0	1	x	1
1	1	0	x	0

K-map for J_{Ai}

A_i \ B_i	0	1
0	0	0
1	x	x

$J_{Ai} = 0$

K-map for K_{Ai}

A_i \ B_i	0	1
0	x	x
1	1	0

$K_{Ai} = \bar{B}_i$

Since, AND operation is enabled when $S_4 = 1$. Thus, the input must be AND with S_4.

$J_{Ai} = 0$

$K_{Ai} = B_i' S_4$

Thus, the final logic statement for input J_{Ai} and K_{Ai} is,

$J_{Ai} = A_i B_i' S_1 + A_i' B_i S_1 + S_3$ $\qquad C_{i+1} = A_i B_i + A_i C_i + B_i C_i$

$K_{Ai} = A_i B_i' S_1 + A_i' B_i S_1 + S_2 + S_3 + B_i S_4$

Final logic circuit :

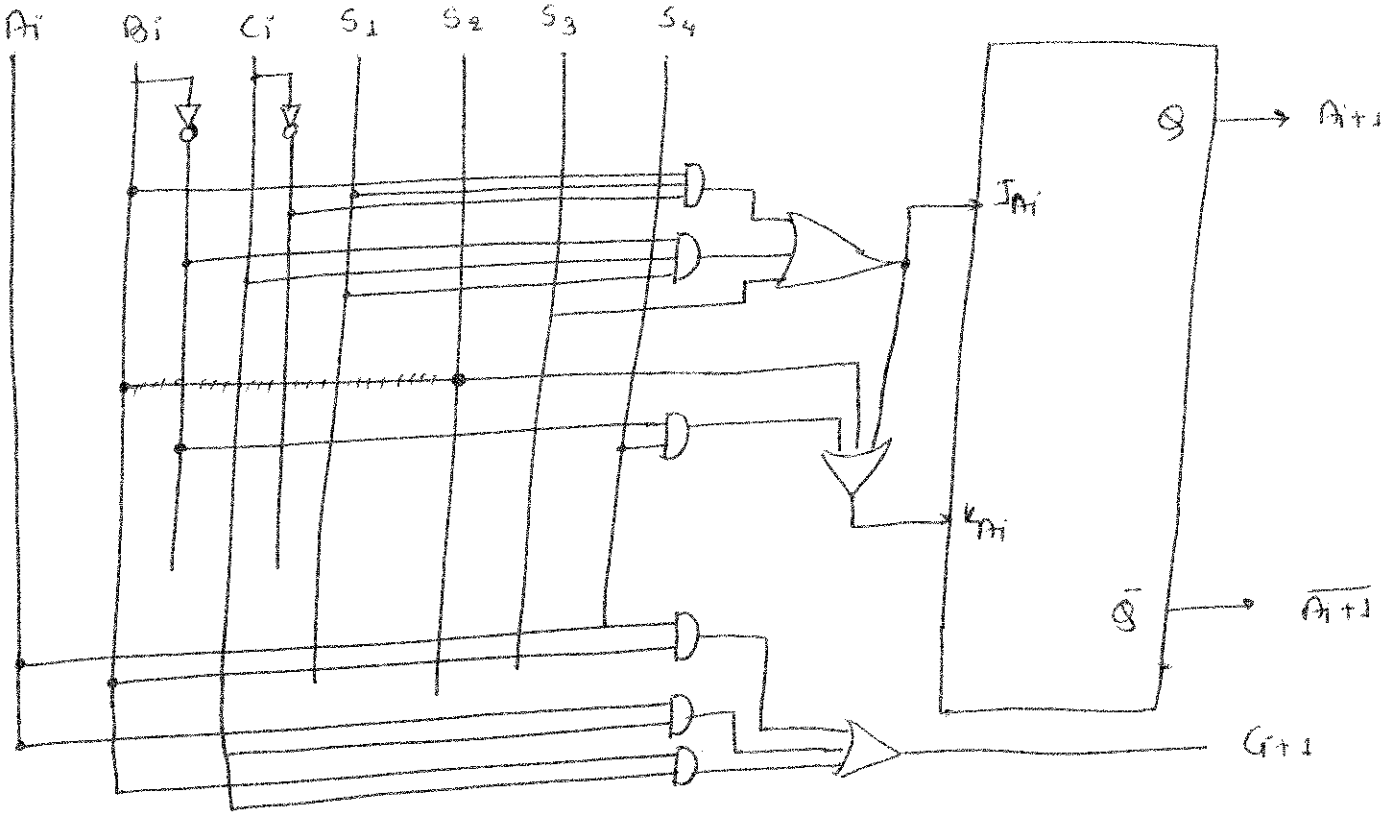


Fig: Accumulator circuit (1 stage)