NOTES
ON
DIGITAL ELECTRONICS

B.Tech. II YEAR - II Sem.
(2022-23)

MALLADI SUNDER RAO

ASSISTANT PROFESSOR

DEPARTMENT OF ELECTRICAL&ELECTRONICS ENGINEERING
NARSIMHA REDDY ENGINEERING COLLEGE
(UGC AUTONOMOUS)

Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad
Accredited by NAAC with A Grade, Accredited by NBA

Maisammguda (V), Dhulapally (P), Medchal (M) Secunderabad-14

DIGITAL ELECTRONICS

SYLLABUS

UNIT - I

Fundamentals of Digital Systems and Logic Families: Digital signals, digital circuits, AND, OR, NOT, NAND, NOR and Exclusive-OR operations, Boolean algebra, examples of IC gates, number systems binary, signed binary, octal hexadecimal number, binary codes, characteristics of digital lCs, digital logic families, TTL, Schottky TTL and CMOS logic, interfacing CMOS and TTL, Tri-state logic.

UNIT - II

Combinational Digital Circuits: Standard representation for logic functions, K-map representation, and simplification of logic functions using K-map, minimization of logical functions. care conditions, ultiplexer, De-Multiplexer/Decoders, Adders, Subtractors, BCD arithmetic, carry look ahead adder, serial ladder, ALU, elementary ALU design, popular MSI chips, digital comparator, parity hecker/generator, code converters, priority encoders, decoders/drivers for display devices, Q-M method of function realization.

UNIT - III

Sequential Circuits and Systems: A 1-bit memory, the circuit properties of Bi-stable latch, the clocked SR flip flop, J, K, T and D types flip-flops, applications of flip-flops, shift registers, applications of shift egisters, serial to parallel converter, parallel to serial converter, ring counter, sequence generator, ripple (Asynchronous) counters, synchronous counters, asynchronous sequential counters, applications of counters.

UNIT IV

A/D and D/A Converters: Digital to analog converters: weighted resistor/converter,R-2R Ladder D/A converter, specifications for D/A converters, examples of D/A converter lCs, sample and hold circuit, analog to digital converters: quantization and encoding, parallel comparator A/D converter, successive approximation A/D converter, counting A/D converter, dual slope A/D converter, A/D converter using voltage to frequency and voltage to time conversion, specifications of A/D converters, example of A/D converter ICs.

UNIT - V

Semiconductor Memories and Programmable Logic Devices: Memory organization and operation, expanding memory size, classification and characteristics of memories, sequential memory, read only memory (ROM), read and write memory (RAM), content addressable memory (CAM), charge de coupled device memory (CCD), commonly used memory chips, ROM as a PLD, Programmable logic array, Programmable array logic, complex Programmable logic devices (CPLDS), Field Programmable Gate Array (FPGA).

UNIT - 1

Fundamentals of Digital Systems and Logic Families:

- Introduction about digital system
- Philosophy of number systems
- Complement representation of negative numbers
- Binary arithmetic
- Binary codes
- Error detecting & error correcting codes
- Hamming codes

INTRODUCTION ABOUT DIGITAL SYSTEM

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

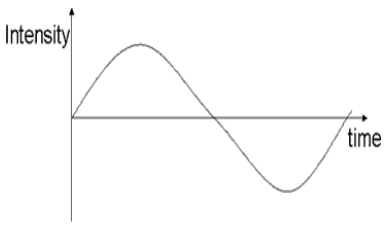
Characteristics of Digital systems

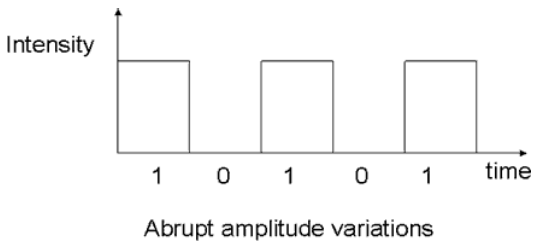
- Digital systems manipulate discrete elements of information.
- Discrete elements are nothing but the digits such as 10 decimal digits or 26 letters of alphabets and so on.
- Digital systems use physical quantities called signals to represent discrete elements.
- In digital systems, the signals have two discrete values and are therefore said to be binary.
- A signal in digital system represents one binary digit called a bit. The bit has a value either 0 or 1.

Analog systems vs Digital systems

Analog system process information that varies continuously i.e; they process time varying signals that can take on any values across a continuous range of voltage, current or any physical parameter.

Digital systems use digital circuits that can process digital signals which can take either 0 or 1 for binary system.





Abrupt amplitude variations

Advantages of Digital system over Analog system

1. Ease of programmability

The digital systems can be used for different applications by simply changing the program without additional changes in hardware.

2. Reduction in cost of hardware

The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.

3. High speed

Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.

4. High Reliability

Digital systems are highly reliable one of the reasons for that is use of error correction codes.

5. Design is easy

The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.

6. Result can be reproduced easily

Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

Disadvantages of Digital Systems

- Use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well.
- Digital circuits are often fragile, in that if a single piece of digital data is lost or misinterpreted the meaning of large blocks of related data can completely change.
- Digital computer manipulates discrete elements of information by means of a binary code.
- Quantization error during analog signal sampling.

NUMBER SYSTEM

Number system is a basis for counting varies items. Modern computers communicate and operate with binary numbers which use only the digits 0 &1. Basic number system used by humans is Decimal number system.

For Ex: Let us consider decimal number 18. This number is represented in binary as 10010.

We observe that binary number system take more digits to represent the decimal number. For large numbers we have to deal with very large binary strings. So this fact gave rise to three new number systems.

i) Octal number systems

ii) Hexa Decimal number system

iii) Binary Coded Decimal number(BCD) system

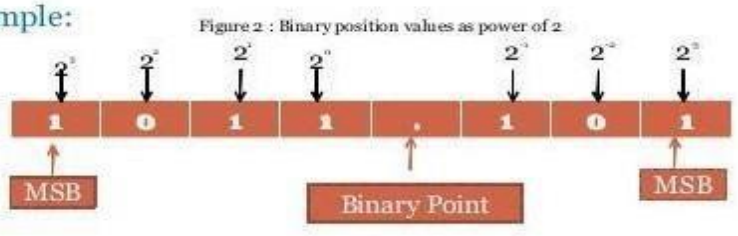
To define any number system we have to specify

- Base of the number system such as 2,8,10 or 16.
- The base decides the total number of digits available in that number system.
- First digit in the number system is always zero and last digit in the number system is always base-1.

Binary number system:

The binary number has a radix of 2. As r = 2, only two digits are needed, and these are 0 and 1. In binary system weight is expressed as power of 2.

- Example:



The left most bit, which has the greatest weight is called the Most Significant Bit (MSB). And the right most bit which has the least weight is called Least Significant Bit (LSB).

For Ex: $1001.01_2 = [(1) \times 2^3] + [(0) \times 2^2] + [(0) \times 2^1] + [(1) \times 2^0] + [(0) \times 2^{-1}] + [(1) \times 2^2]$

$1001.01_2 = [1 \times 8] + [0 \times 4] + [0 \times 2] + [1 \times 1] + [0 \times 0.5] + [1 \times 0.25]$

$1001.01_2 = 9.25_{10}$

Decimal Number system

The decimal system has ten symbols: 0,1,2,3,4,5,6,7,8,9. In other words, it has a base of 10.

Octal Number System

Digital systems operate only on binary numbers. Since binary numbers are often very long, two shorthand notations, octal and hexadecimal, are used for representing large binary numbers. Octal systems use a base or radix of 8. It uses first eight digits of decimal number system. Thus it has digits from 0 to 7.

Hexa Decimal Number System

The hexadecimal numbering system has a base of 16. There are 16 symbols. The decimal digits 0 to 9 are used as the first ten digits as in the decimal system, followed by the letters A, B, C, D, E and F, which represent the values 10, 11,12,13,14 and 15 respectively.

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Number Base conversions

The human beings use decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number system into its equivalent binary.

i) Binary to octal number conversion
ii) Binary to hexa decimal number conversion

The binary number:	001	010	011	000	100	101	110	111
The octal number:	1	2	3	0	4	5	6	7

The binary number:	0001	0010	0100	1000	1001	1010	1101	1111
The hexadecimal number:	1	2	5	8	9	A	D	F

iii) Octal to binary Conversion

Each octal number converts to 3 binary digits

Code
0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111

To convert 653_8 to binary, just substitute code:

6 → 110, 5 → 101, 3 → 011

iv) Hexa to binary conversion

4 → 0100, F → 1111, D → 1101, 7 → 0111

v) Octal to Decimal conversion
Ex: convert 4057.06_8 to octal

$=4x8^3+0x8^2+5x8^1+7x8^0+0x8^{-1}+6x8^{-2}$

$=2048+0+40+7+0+0.0937$

$=2095.0937_{10}$

vi) Decimal to Octal Conversion

Ex: convert 378.93_{10} to octal

378_{10} to octal: Successive division:

```
8 |  378
  |____
8 |47 ---   2
  |_________
8 |5  ---   7      ↑
  |_________
  0  ---   5
```

$=572_8$

0.93_{10} to octal :

0.93x8=7.44
0.44x8=3.52 ↓
0.53x8=4.16
0.16x8=1.28

$=0.7341_8$

$378.93_{10}=572.7341_8$

vii) Hexadecimal to Decimal Conversion

Ex: $5C7_{16}$ to decimal

$=(5x16^2)+(C\ x16^1)+ (7\ x16^0)$

$=1280+192+7$

$=147_{10}$

viii) Decimal to Hexadecimal Conversion

Ex: 2598.67510

```
1 6 | 2598
    |____
16  162      -6
    10       -2
```

= A26 (16)

0.67510=0.675x16 -- 10.8

=0.800x16 -- 12.8 ↓
=0.800x16 -- 12.8
=0.800x16 -- 12.8
$=0.ACCC_{16}$

$2598.675_{10} = A26.ACCC_{16}$

ix) Octal to hexadecimal conversion:

The simplest way is to first convert the given octal no. to binary & then the binary no. to hexadecimal.

Ex: 756.603_8

7	5	6	.	6	0	3
111	101	110	.	110	000	011
0001	1110	1110	.	1100	0001	1000
1	E	E	.	C	1	8

x) Hexadecimal to octal conversion:

First convert the given hexadecimal no. to binary & then the binary no. to octal.

Ex: B9F.AE16

B	9	F	.	A	E		
1011	1001	1111	.	1010	1110		
101	110	011	111	.	101	011	100
5	6	3	7	.	5	3	4

=5637.534

Complements:

In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.

i) The radix complement or r's complement
ii) The diminished radix complement or (r-1)'s complement

Representation of signed no.s binary arithmetic in computers:

- Two ways of rep signed no.s
    1. Sign Magnitude form
    2. Complemented form
- Two complimented forms
    1. 1‘s compliment form
    2. 2‘s compliment form

Advantage of performing subtraction by the compliment method is reduction in the hardware.(instead of addition & subtraction only adding ckt‘s are needed.)
i. e, subtraction is also performed by adders only.

Instead of subtracting one no. from other the compliment of the subtrahend is added to minuend. In sign magnitude form, an additional bit called the sign bit is placed in front of the no. If the sign bit is 0, the no. is +ve, If it is a 1, the no is _ve.

Ex:

0	1	0	1	0	0	1

↓

Sign bit =+41 magnitude

↑

1	1	0	1	0	0	1

= -41

Note: manipulation is necessary to add a +ve no to a –ve no

Representation of signed no.s using 2’s or 1’s complement method:

If the no. is +ve, the magnitude is rep in its true binary form & a sign bit 0 is placed in front of the MSB.I f the no is _ve , the magnitude is rep in its 2‘s or 1‘s compliment form &a sign bit 1 is placed in front of the MSB.

Ex:

Given no.	Sign mag form	2‘s comp form	1‘s comp form
01101	+13	+13	+13
010111	+23	+23	+23
10111	-7	-7	-8
1101010	-42	-22	-21

Special case in 2's comp representation:

Whenever a signed no. has a 1 in the sign bit & all 0's for the magnitude bits, the decimal equivalent is -2^n , where n is the no of bits in the magnitude .

Ex: 1000= -8 & 10000=-16

Characteristics of 2's compliment no.s:

Properties:

1. There is one unique zero
2. 2's comp of 0 is 0
3. The leftmost bit can't be used to express a quantity . it is a 0 no. is +ve.
4. For an n-bit word which includes the sign bit there are $(2^{n-1}-1)$ +ve integers, 2^{n-1} –ve integers & one 0 , for a total of 2^n uniquestates.
5. Significant information is containd in the 1's of the +ve no.s & 0's of the _ve no.s
6. A _ve no. may be converted into a +ve no. by finding its 2's comp.

Signed binary numbers:

Decimal	Sign 2's comp form	Sign 1's comp form	Sign mag form
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0011	0011	0011
+0	0000	0000	0000

-0	--	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
8	1000	--	--

Methods of obtaining 2's comp of a no:

- In 3 ways
  1. By obtaining the 1's comp of the given no. (by changing all 0's to 1's & 1's to 0's) & then adding 1.
  2. By subtracting the given n bit no N from 2^n
  3. Starting at the LSB , copying down each bit upto & including the first 1 bit encountered , and complimenting the remaining bits.

  Ex: Express -45 in 8 bit 2's comp form

  +45 in 8 bit form is 00101101

  I method:

  1's comp of 00101101 & the add 1

  00101101
  11010010
  +1
  ___ _ _ _ _ _ _ _ _ _

  11010011 is 2's comp form

  II method:

  Subtract the given no. N from 2^n

  2^n = 100000000

  Subtract 45= -00101101
  +1
  ___ _ _ _

  11010011 is 2's comp

  III method:

  Original no: 00101101

Copy up to First 1 bit 1
Compliment remaining : 1101001

_____

bits 11010011

Ex:

-73.75 in 12 bit 2's comp form

I method

01001001.1100
10110110.0011
+1

_____

10110110.0100 is 2‘s

II method:

2^8 = 100000000.0000

Sub 73.75=-01001001.1100

_____

10110110.0100 is 2‘s comp

III method :

Orginalno : 01001001.1100
Copy up to 1‘st bit 100
Comp the remaining bits: 10110110.0

_____

10110110.0100

2’s compliment Arithmetic:

- The 2‘s comp system is used to rep –ve no.s using modulus arithmetic . The word length of a computer is fixed. i.e, if a 4 bit no. is added to another 4 bit no . the result will be only of 4 bits. Carry if any , from the fourth bit will overflow called the Modulus arithmetic.
  Ex:1100+1111=1011
- In the 2‘s compl subtraction, add the 2‘s comp of the subtrahend to the minuend . If there is a carry out , ignore it , look at the sign bit I,e, MSB of the sum term .If the MSB is a 0, the result is positive.& it is in true binary form. If the MSB is a ` (carry in or no carry at all) the result is negative.& is in its 2‘s comp form. Take its 2‘s comp to find its magnitude in binary.

Ex: Subtract 14 from 46 using 8 bit 2‘s comp arithmetic:

+14 = 00001110
-14 = 11110010 2‘s comp

+46 = 00101110
-14 =+11110010 2‘s comp form of -14

-32 (1)00100000 ignore carry

Ignore carry ; The MSB is 0 . so the result is +ve. & is in normal binary

form. So the result is +00100000=+32.

EX: Add -75 to +26 using 8 bit 2‘s comp arithmetic

+75	= 01001011	
-75	=10110101	2‘s comp
+26	= 00011010	
-75	=+10110101	2‘s comp form of -75

-49	11001111	No carry

No carry , MSB is a 1, result is _ve & is in 2‘s comp. The magnitude is 2‘s comp of 11001111. i.e, 00110001 = 49. so result is -49

Ex: add -45.75 to +87.5 using 12 bit arithmetic

+87.5 = 01010111.1000
-45.75=+11010010.0100

-41.75	(1)00101001.1100 ignore carry

MSB is 0, result is +ve. =+41.75

1’s compliment of n number:

- It is obtained by simply complimenting each bit of the no,.& also , 1‘s comp of a no, is subtracting each bit of the no. form 1.This complemented value rep the – ve of the original no. One of the difficulties of using 1‘s comp is its rep o f zero. Both 00000000 & its 1‘s comp 11111111 rep zero.
- The 00000000 called +ve zero& 11111111 called –ve zero.

Ex: -99 & -77.25 in 8 bit 1‘s comp

+99	=	01100011
-99	=	10011100
+77.25 =		01001101.0100
-77.25 =		10110010.1011

1’s compliment arithmetic:

In 1‘s comp subtraction, add the 1‘s comp of the subtrahend to the minuend. If there is a carryout , bring the carry around & add it to the LSB called the **end around carry.** Look at the sign bit (MSB) . If this is a 0, the result is +ve & is in true binary. If the MSB is a 1 (carry or no carry), the result is –ve & is in its is comp form .Take its 1‘s comp to get the magnitude inn binary.

Ex: Subtract 14 from 25 using 8 bit 1‘s

25	=	00011001	+14	=	00001110
-45	=	11110001	-25	=	+11100110
+11		(1)00001010	-11		11110100

+1

No carry MSB =1

EX: ADD -25 to +14

00001011 result=-ve=-11_{10}

MSB is a 0 so result is +ve (binary)

=+11_{10}

Binary codes

Binary codes are codes which are represented in binary system with modification from the original ones.

- Weighted Binary codes
- Non Weighted Codes

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. The binary counting sequence is an example.

Decimal	BCD 8421	Excess-3	84-2-1	2421	5211	Bi-Quinary 5043210			5	0	4	3	2	1	0
0	0000	0011	0000	0000	0000	0100001		0		X					X
1	0001	0100	0111	0001	0001	0100010		1		X				X	
2	0010	0101	0110	0010	0011	0100100		2		X			X		
3	0011	0110	0101	0011	0101	0101000		3		X		X			
4	0100	0111	0100	0100	0111	0110000		4		X	X				
5	0101	1000	1011	1011	1000	1000001		5	X						X
6	0110	1001	1010	1100	1010	1000010		6	X					X	
7	0111	1010	1001	1101	1100	1000100		7	X				X		
8	1000	1011	1000	1110	1110	1001000		8	X			X			
9	1001	1111	1111	1111	1111	1010000		9	X		X				

Reflective Code

A code is said to be reflective when code for 9 is complement for the code for 0, and

so is for 8 and 1 codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211, and excess-3 are reflective, whereas the 8421 code is not.

Sequential Codes

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and Excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

Non weighted codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: Excess-3 code

Excess-3 Code

Excess-3 is a non weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

Gray Code

The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit- distance code. In digital Gray code has got a special place.

Decimal Number	Binary Code	Gray Code	Decimal Number	Binary Code	Gray Code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Binary to Gray Conversion

- Gray Code MSB is binary code MSB.
- Gray Code MSB-1 is the XOR of binary code MSB and MSB-1.
- MSB-2 bit of gray code is XOR of MSB-1 and MSB-2 bit of binary code.
- MSB-N bit of gray code is XOR of MSB-N-1 and MSB-N bit of binary code.

8421 BCD code (Natural BCD code):

Each decimal digit 0 through 9 is coded by a 4 bit binary no. called natural binary codes. Because of the 8,4,2,1 weights attached to it. It is a weighted code & also sequential . it is useful for mathematical operations. The advantage of this code is its case of conversion to & from decimal. It is less efficient than the pure binary, it require more bits.

Ex: 14→1110 in binary

But as 0001 0100 in 8421 ode.

The disadvantage of the BCD code is that , arithmetic operations are more complex than they are in pure binary . There are 6 illegal combinations 1010,1011,1100,1101,1110,1111 in these codes, they are not part of the 8421 BCD code system . The disadvantage of 8421 code is, the rules of binary addition 8421 no, but only to the individual 4 bit groups.

BCD Addition:

It is individually adding the corresponding digits of the decimal no,s expressed in 4 bit binary groups starting from the LSD . If there is no carry & the sum term is not an illegal code , no correction is needed .If there is a carry out of one group to the next group or if the sum term is an illegal code then 6_{10}(0100) is added to the sum term of that group & the resulting carry is added to the next group.

Ex: Perform decimal additions in 8421 code

(a)25+13

In BCD 25= 0010 0101

In BCD +13 =+0001 0011

38 0011 1000

No carry , no illegal code .This is the corrected sum

(b). 679.6 + 536.8

679.6	=	0110	0111	1001	.0110 in BCD
+536.8	=	+0101	0011	0010	.1000 in BCD
1216.4		1011	1010	0110	. 1110 illegal codes
		+0110	+ 0011	+0110	. + 0110 add 0110 to each
		(1)0001	(1)0000	(1)0101	. (1)0100 propagate carry
		/	/	/	/
		+1	+1	+1	+1
		0001	0010	0001	0110 . 0100
		1	2	1	6 . 4

BCD Subtraction:

Performed by subtracting the digits of each 4 bit group of the subtrahend the digits from the corresponding 4- bit group of the minuend in binary starting from the LSD . if there is no borrow from the next group , then 6_{10}(0110)is subtracted from the difference term of this group.

(a)38-15

In BCD	38=	0011	1000
In BCD	-15 =	-0001	0101
	23	0010	0011

No borrow, so correct difference.

.(b) 206.7-147.8

206.7	=	0010	0000	0110	.	0111	in BCD
-147.8	=	-0001	0100	0111	.	0110	in BCD
58.9		0000	1011	1110	.	1111	borrows are present
		-0110	-0110	.		-0110	subtract 0110
			0101	1000	.	1001	

BCD Subtraction using 9's & 10's compliment methods:

Form the 9's & 10's compliment of the decimal subtrahend & encode that no. in the 8421 code . the resulting BCD no.s are then added.

EX: 305.5 – 168.8

305.5 = 305.5
-168.8= +83.1 9's comp of -168.8
— – –
(1)136.6
+1 end around carry
136.7 corrected difference

305.5_{10} = 0011 0000 0101 . 0101
$+831.1_{10}$ = +1000 0011 0001 . 0001 9's comp of 1 68.8 in BCD
– – – ——————
+1011 0011 0110 . 0110 1011 is illegal code
+0110 add 0110

---

(1)0001 0011 0110 . 0110
+1 End around carry

0001 0011 0110 . 0111
= 136.7

Excess three(xs-3)code:

It is a non-weighted BCD code .Each binary codeword is the corresponding 8421 codeword plus 0011(3).It is a sequential code & therefore , can be used for arithmetic operations..It is a self-complementing code.s o the subtraction by the method of compliment addition is more direct in xs-3 code than that in 8421 code. The xs-3 code has six invalid states 0000,0010,1101,1110,1111.. It has interesting properties when used in addition & subtraction.

Excess-3 Addition:

Add the xs-3 no.s by adding the 4 bit groups in each column starting from the LSD. If there is no carry starting from the addition of any of the 4-bit groups , subtract 0011 from the sum term of those groups (because when 2 decimal digits are added in xs-3 & there is no carry , result in xs-6). If there is a carry out, add 0011 to the sum term of those groups(because when there is a carry, the invalid states are skipped and the result is normal binary).

EX: 37 0110 1010
+28 +0101 1011

65 1011 (1)0101 carry generated
+1 propagate carry

1100 0101 add 0011 to correct 0101 &
-0011 +0011 subtract 0011 to correct 1100

1001 1000 $=65_{10}$

Excess -3 (XS-3) Subtraction:

Subtract the xs-3 no.s by subtracting each 4 bit group of the subtrahend from the corresponding 4 bit group of the minuend starting form the LSD .if there is no borrow from the next 4-bit group add 0011 to the difference term of such groups (because when decimal digits are subtracted in xs-3 & there is no borrow , result is normal binary). I f there is a borrow , subtract 0011 from the differenceterm(b coz taking a borrow is equivalent to adding six invalid states , result is in xs-6)

Ex: 267-175

267 = 0101 1001 1010
-175= -0100 1010 1000

0000 1111 0010
+0011 -0011 +0011

0011 1100 +0011 $=92_{10}$

Xs-3 subtraction using 9's & 10's compliment methods:

Subtraction is performed by the 9's compliment or 10's compliment

Ex:687-348 The subtrahend (348) xs -3 code & its compliment are:

9's comp of 348 = 651

Xs-3 code of 348 = 0110 0111 1011

1's comp of 348 in xs-3 = 1001 1000 0100

Xs=3 code of 348 in xs=3 = 1001 1000 0100

687 687

-348 → +651 9's compl of 348

339 (1)338

+1 end around carry

339 corrected difference in decimal

1001	1011	1010	687 in xs-3
+1001	1000	0100	1's comp 348 in xs-3

(1)0010 (1)0011 1110 carry generated

//

+1 +1 propagate carry

(1)0011 0010 1110

+1 end around carry

0011 0011 1111 (correct 1111 by sub0011 and
+0011 +0011 +0011 correct both groups of 0011 by
adding 0011)

0110 0110 1100 corrected diff in xs-3 = 330_{10}

The Gray code (reflective –code):

Gray code is a non-weighted code & is not suitable for arithmetic operations. It is not a BCD code . It is a cyclic code because successive code words in this code differ in one bit position only i.e, it is a unit distance code.Popular of the unit distance code.It is also a reflective code i.e,both reflective & unit distance. The n least significant bits for 2^n through 2^{n+1}-1 are the mirror images of thosr for 0 through 2^n-1.An N bit gray code can be obtained by reflecting an N-1 bit code about an axis at the end of the code, & putting the MSB of 0 above the axis & the MSB of 1 below the axis.

Reflection of gray codes:

Gray Code				Decimal	4 bit binary
1 bit	2 bit	3 bit	4 bit		
0	00	000	0000	0	0000
1	01	001	0001	1	0001
	11	011	0011	2	0010
	10	010	0010	3	0011
		110	0110	4	0100
		111	0111	5	0101
		101	0101	6	0110
		110	0100	7	0111

			1100	8	1000
			1101	9	1001
			1111	10	1010
			1110	11	1011
			1010	12	1100
			1011	13	1101
			1001	14	1110
			1000	15	1111

Binary to Gray conversion:

N bit binary no is rep by $B_n\ B_{n-1}$ ---------- B_1

Gray code equivalent is by $G_n\ G_{n-1}$ ---------- G_1

B_n, G_n are the MSB's then the gray code bits are obtaind from the binary code as

$G_n=B_n$	$G_{n-1}=B_n \oplus B_{n-1}$	$G_{n-2}=B_{n-1} \oplus B_{n-}$	----------	$G_1=B_2 \oplus B1$	

$\oplus$ →EX-or symbol

Procedure: ex-or the bits of the binary no with those of the binary no shifted one position to the right . The LSB of the shifted no. is discarded & the MSB of the gray code no.is the same as the MSB of the original binaryno.

EX: 10001 $\oplus$ $\oplus$ $\oplus$

(a). Binary : 1 →0 →0 →1

Gray : 1 1 0 1

(b). Binary: 1 0 0 1

Shifted binary: 1 0 0 (1)

1 1 0 1→gray

Gray to Binary Conversion:

If an n bit gray no. is rep by G_n G_{n-1}---------- G_1

its binary equivalent by B_n B_{n-1} ---------- B_1 then the binary bits are obtained from gray bits as

$B_n = G_n$	$B_{n-1} = B_n \oplus G_{n-1}$	$B_{n-2} = \oplus G_{n-2}$	----------	$B_1 = B_2 \oplus G_1$

To convert no. in any system into given no. first convert it into binary & then binary to gray. To convert gray no into binary no & convert binary no into require no system.

Ex:10110010(gray) = 11011100_2= DC_{16}=334_8=220_{10}
EX:1101

Gray: 1 1 0 1

Binary:1 0 0 1

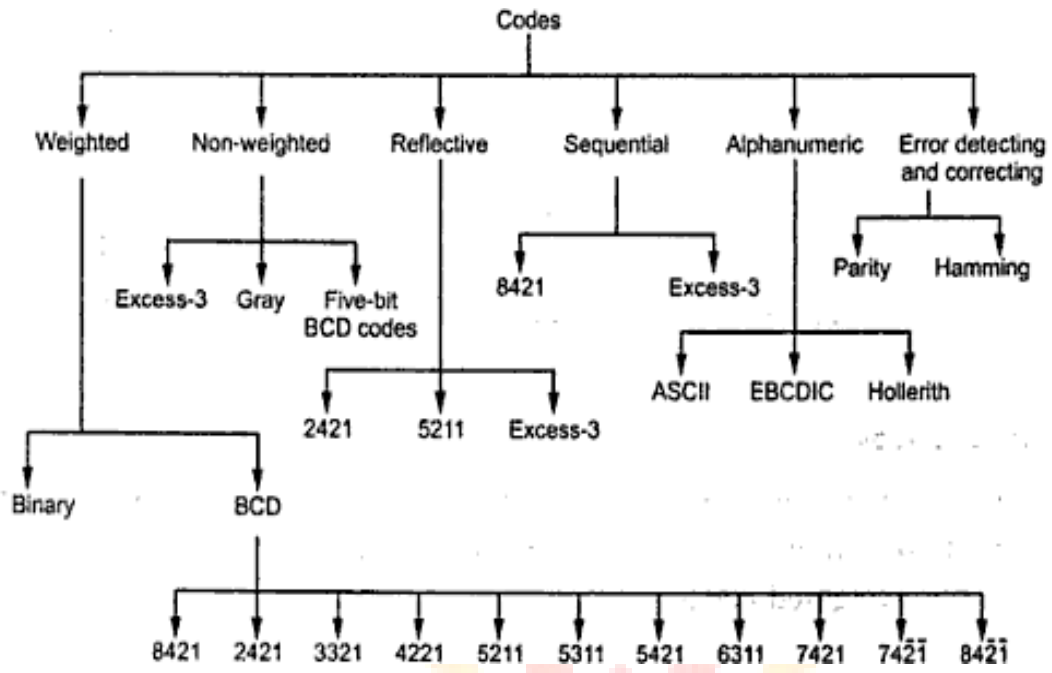
Ex: $3A7_{16}$= 0011,1010,0111_2=1001110100(gray)
527_8=101,011,011_2=111110110(gray)
652_{10}=1010001100_2= 1111001010(gray)

XS-3 gray code:

In a normal gray code , the bit patterns for 0(0000) & 9(1101) do not have a unit distance between them i.e, they differ in more than one position.In xs-3 gray code , each decimal digit is encoded with gray code patter of the decimal digit that is greater by 3. It has a unit distance between the patterns for 0 & 9.

XS-3 gray code for decimal digits 0 through 9

Decimal digit	Xs-3 gray code	Decimal digit	Xs-3 gray code
0	0010	5	1100
1	0110	6	1101
2	0111	7	1111
3	0101	8	1110
4	0100	9	1010



Binary codes block diagram

Error – Detecting codes: When binary data is transmitted & processed,it is susceptible to noise that can alter or distort its contents. The 1‘s may get changed to 0‘s & 1‘s .because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

Parity: The simplest techniques for detecting errors is that of adding an extra bit known as parity bit to each word being transmitted.Two types of parity**:** Oddparity**,** evenparity forodd parity, the parity bit is set to a _0‘ or a _1‘ at the transmitter such that the total no. of 1 bit in the word including the parity bit is an odd no.For even parity, the parity bit is set to a _0‘ or a _1‘ at the transmitter such that the parity bit is an even no.

Decimal	8421 code	Odd parity	Even parity
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0100	1	0
6	0110	1	0
7	0111	0	1
8	1000	0	1
9	1001	1	0

When the digit data is received . a parity checking circuit generates an error signal if the total no of 1‘s is even in an odd parity system or odd in an even parity system. This parity check can always detect a single bit error but cannot detect 2 or more errors with in the same word.Odd parity is used more often than even parity does not detect the situation. Where all 0‘s are created by a short ckt or some other fault condition.

Ex: Even parity scheme

(a) 10101010 (b) 11110110 (c)10111001

Ans:

(a) No. of 1‘s in the word is even is 4 so there is no error
(b) No. of 1‘s in the word is even is 6 so there is no error
(c) No. of 1‘s in the word is odd is 5 so there is error

Ex: odd parity

(a)10110111 (b) 10011010 (c)11101010

Ans:

(a) No. of 1‘s in the word is even is 6 so word has error
(b) No. of 1‘s in the word is even is 4 so word has error
(c) No. of 1‘s in the word is odd is 5 so there is no error

Checksums:

Simple parity can‘t detect two errors within the same word. To overcome this, use a sort of 2 dimensional parity. As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end. At the end of transmission, the sum called the check sum. Up to that time sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same, then no errors were detected at the receiver end. If there is an error, the receiving location can ask for retransmission of the entire data, used in teleprocessing systems.

Block parity:

Block of data shown is create the row & column parity bits for the data using odd parity. The parity bit 0 or 1 is added column wise & row wise such that the total no. of 1‘s in each column & row including the data bits & parity bit is odd as

Data	Parity bit
10110	0
10001	1
10101	0
00010	0
11000	1
00000	1
11010	0

data
10110
10001
10101
00010
11000
00000
11010

Error –Correcting Codes:

A code is said to be an error –correcting code, if the code word can always be deduced from an erroneous word. For a code to be a single bit error correcting code, the minimum distance of that code must be three. The minimum distance of that code is the smallest no. of bits by which any two code words must differ. A code with minimum distance of 3 can't only correct single bit errors but also detect (can't correct) two bit errors, The key to error correction is that it must be possible to detect & locate erroneous that it must be possible to detect & locate erroneous digits. If the location of an error has been determined. Then by complementing the erroneous digit, the message can be corrected , error correcting , code is the Hamming code , In this , to each group of m information or message or data bits, K parity checking bits denoted by P1,P2,----------pk located at positions 2^{k-1} from left are added to form an (m+k) bit code word. To correct the error, k parity checks are performed on selected digits of each code word, & the position of the error bit is located by forming an error word, & the error bit is then complemented. The k bit error word is generated by putting a 0 or a 1 in the 2^{k-1}th position depending upon whether the check for parity involving the parity bit P_k is satisfied or not.Error positions & their corresponding values :

Error Position	For 15 bit code $C_4\ C_3\ C_2\ C_1$	For 12 bit code $C_4\ C_3\ C_2\ C_1$	For 7 bit code $C_3\ C_2\ C_1$
0	0 0 0 0	0 0 0 0	0 0 0
1	0 0 0 1	0 0 0 1	0 0 1
2	0 0 1 0	0 0 1 0	0 1 0
3	0 0 1 1	0 0 1 1	0 1 1
4	0 1 0 0	0 1 0 0	1 0 0
5	0 1 0 1	0 1 0 1	1 0 1
6	0 1 1 0	0 1 1 0	1 1 0
7	0 1 1 1	0 1 1 1	1 1 1
8	1 0 0 0	1 0 0 0	
9	1 0 0 1	1 0 0 1	
10	1 0 1 0	1 0 1 0	
11	1 0 1 1	1 0 1 1	
12	1 1 0 0	1 1 0 0	
13	1 1 0 1		
14	1 1 1 0		
15	1 1 1 1		

7- bit Hamming code:

To transmit four data bits, 3 parity bits located at positions 2^0 21&2^2 from left are added to make a 7 bit codeword which is then transmitted.

The word format

P_1	P_2	D_3	P_4	D_5	D_6	D_7

D—Data bits P-

Parity bits

Decimal Digit	For BCD $P_1P_2D_3P_4D_5D_6D_7$	For Excess-3 $P_1P_2D_3P_4D_5D_6D_7$
0	0 0 0 0 0 0 0	1 0 0 0 0 1 1
1	1 1 0 1 0 0 1	1 0 0 1 1 0 0
2	0 1 0 1 0 1 1	0 1 0 0 1 0 1
3	1 0 0 0 0 1 1	1 1 0 0 1 1 0
4	1 0 0 1 1 0 0	0 0 0 1 1 1 1
5	0 1 0 0 1 0 1	1 1 1 0 0 0 0
6	1 1 0 0 1 1 0	0 0 1 1 0 0 1
7	0 0 0 1 1 1 1	1 0 1 1 0 1 0
8	1 1 1 0 0 0 0	0 1 1 0 0 1 1
9	0 0 1 1 0 0 1	0 1 1 1 1 0 0

Ex: Encode the data bits 1101 into the 7 bit even parity Hamming Code

The bit pattern is

$P_1P_2D_3P_4D_5D_6D_7$

1 1 0 1

Bits 1,3,5,7 (P_1 111) must have even parity, so P_1 =1

Bits 2, 3, 6, 7(P_2 101) must have even parity, so P_2 =0

Bits 4,5,6,7 (P_4 101)must have even parity, so P_4 =0

The final code is 1010101

EX: Code word is 1001001

Bits 1,3,5,7 (C_1 1001) →no error →put a 0 in the 1's position→C_1=0

Bits 2, 3, 6, 7(C_2 0001)) → error →put a 1 in the 2's position→C_2=1

Bits 4,5,6,7 (C_4 1001)) →no error →put a 0 in the 4's position→C_3=0

15-bit Hamming Code: It transmit 11 data bits, 4 parity bits located 2^0 2^1 2^2 2^3

Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12	D13	D14	D15

12-Bit Hamming Code: It transmit 8 data bits, 4 parity bits located at position 2^0 2^1 2^2 2^3

Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12

Alphanumeric Codes:

These codes are used to encode the characteristics of alphabet in addition to the decimal digits. It is used for transmitting data between computers & its I/O device such as printers, keyboards & video display terminals.Popular modern alphanumeric codes are ASCII code & EBCDIC code.

Boolean algebra

In 1854, George Boole developed an algebraic system now called Boolean algebra. In 1938, Claude E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0, 1), two binary operators called OR, AND, and one unary operator NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits. It is a way to express logic functions algebraically.

Axioms and laws of Boolean algebra

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

	AND Operation	OR Operation	NOT Operation
Axiom1 :	0.0=0	0+0=0	$\bar{0}$=1
Axiom2:	0.1=0	0+1=1	$\bar{1}$=0
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

Complementation law

Law1: $\bar{0}$=1

Law2: $\bar{1}$=0

Law3: if A=0,then $\bar{A}$=1

Law4: if A=1,then $\bar{A}$=0

Law5: if $\bar{\bar{A}}$=A (double inversion law)

AND Law

Law1: A.0=0 (Null law)

Law2: A.1=A (Identity law)

Law3: A.A=A (Impotence law)

Law4: A.$\bar{A}$=0

OR Law

Law1: A+0=A

Law2: A+1=1

Law3: A+A=A (Impotence law)

Law4: A+$\bar{A}$=1

Basic Theorems and Properties of Boolean algebra

Commutative law

Law1: A+B=B+A Law2: A.B=B.A

Associative law

Law1: A + (B +C) = (A +B) +C Law2: A(B.C) = (A.B)C

Distributive law

Law1: A.(B + C) = AB+ AC Law2: A + BC = (A + B).(A +C)

Absorption law

Law1: A +AB =A Law2: A(A +B) = A

Solution: A(1+B)
A

Solution: A.A+A.B
A+A.B
A(1+B)
A

DeMorgan Theorems

Theorem1: $\overline{(AB)} = \bar{A}\bar{B}$ Theorem2: $\overline{(A+B)} = \bar{A}\bar{B}$

Redundant Literal Rule

Rule1: $A+\bar{A}B=A+B$ Rule2: $A.(\bar{A}B)=AB$

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

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

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

Solution: $A.(\bar{A}B)$

$A.\bar{A}A.B$

AB

Consensus Theorem

Theorem1. AB+ A'C + BC = AB + A'C Theorem2. (A+B). (A'+C).(B+C) =(A+B).(A'+C)

The BC term is called the consensus term and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (A) and its complement (A') are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement

Consensus Theorem1 Proof:

AB+A'C+BC=AB+A'C+(A+A')BC
=AB+A'C+ABC+A'BC
=AB(1+C)+A'C(1+B)
= AB+ A'C

Principle of Duality

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations (+) and (·) as well as the identity elements 0 and 1.
Such expressions are known as duals of each other.
If some equivalence is proved, then its dual is also immediately true.
E.g. If we prove: (x.x)+(x'+x')=1, then we have by duality: (x+x)·(x'.x')=0

The Huntington postulates were listed in pairs and designated by part (a) and part (b) in below table.

Table for Postulates and Theorems of Boolean algebra

Part-A	Part-B
A+0=A	A.0=0
A+1=1	A.1=A
A+A=A (Impotence law)	A.A=A (Impotence law)
$A+\bar{A}$1	$A.\bar{A}$0
$\bar{\bar{A}}$A (double inversion law)	--
Commutative law: A+B=B+A	A.B=B.A
Associative law: A + (B +C) = (A +B) +C	A(B.C) = (A.B)C
Distributive law: A.(B + C) = AB+ AC	A + BC = (A + B).(A +C)
Absorption law: A +AB =A	A(A +B) = A
DeMorgan Theorem: $\overline{AB}$ =$\bar{A}\bar{B}$	$\overline{(A+B)}$=$\bar{A}\bar{B}$
Redundant Literal Rule: A+$\bar{A}$B=A+B	A.($\bar{A}$B)=AB
Consensus Theorem: AB+ A'C + BC = AB + A'C	(A+B). (A'+C).(B+C) =(A+B).(A'+C)

History of Digital ICs and their Classifications

Integrated circuit: A collection of one or more gates fabricated on a single silicon chip is called an Integrated Circuit(IC) or it can be defined as an Integrated Circuit(IC) is a silicon wafer or a Die that contains two or more number of active such as diodes, transistors and some of passive components such as resistors, capacitors.

NOTE: The passive element Inductor can't be fabricated using ICs, Because they have magnetic flux

History of IC:

- There are many ways are available to design electronic logic circuit. First electrically controlled logic circuits were developed in 1930s at Bell laboratories based on relays.
- The first electronic digital computer named as ENIAC was developed in mid 1940s based on vacuum tubes. It has 100 feet long, 10 feet height, 3 feet deep and consumed 140kw of power.
- By the invention of semiconductor diode and transistor after 1947 smaller, faster and more capable computers were designed.
- Better computers are designed by the invention of ICs which allowed multiples diodes, transistors and other components to be fabricated on a single chip of silicon in 1960s.
- The first IC family was introduced in 1960.

Classification of Digital ICs: Based on the size or number of logic components/gates fabricated per chip the ICs are classified into different Integrations as

- Small Scale Integration (SSI): It have less than 100 components (about 10 gates).
- Medium Scale Integration (MSI): It contains between 100-1000 components or have more than 10 but less than 100 gates.
- Large Scale Integration (LSI): Here number of components is between 1000 and 10000 or have gates between 100-1000.
- Very Large Scale Integration (VLSI): It contains components between 10000-100000 per chip or gates between 1000-10000 per chip.
- Ultra Large Scale Integration (ULSI): It contains more than 100000 components per chip.
- Giant Scale Integration (GSI): It contains much more than 2000000 components per chip.

Logic Families and their classifications:

Logic Families:

It is a collection of different IC chips that have similar input, output and internal circuit characteristics i.e. group of compatible ICs with same logic levels and supply voltages but perform different logic functions.

NOTE: 1) Chips from same family can be interconnected.

2) Chips from different family may not be compatible, means they may use different power supply voltages and input, output conditions.

Classification of Logic Families:

Logic families are mainly classified as two types as

Bipolar Logic Families: It mainly uses bipolar devices like diodes, transistors in addition to passive elements like resistors and capacitors. These are sub classified as saturated bipolar logic family and unsaturated bipolar logic family. i) *Saturated Bipolar Logic Family:* In this family the transistors used in ICs are driven into saturation.

Examples: a) Transistor-Transistor Logic (TTL)
b) Resistor-Transistor Logic (RTL)
c) Direct Coupled Transistor Logic (DCTL)
d) Diode Transistor Logic (DTL)
e)High Threshold Logic(HTL)
f) Integrated Injection Logic (IIL or I2L)

ii)Unsaturated bipolar logic family: In this family the transistors used in ICs are not driven into saturation.

Examples: a) Schottky TTL
b) Emitter Coupled Logic(ECL)

B) Unipolar Logic Families: It mainly uses Unipolar devices like MOSFETs in addition to passive elements like resistors and capacitors. These logic families have the advantages of high speed and lower power consumption
than Bipolar families. These are classified as
i) PMOS or P-Channel MOS Logic Family
ii) NMOS or N-Channel MOS Logic Family
iii) CMOS Logic Family

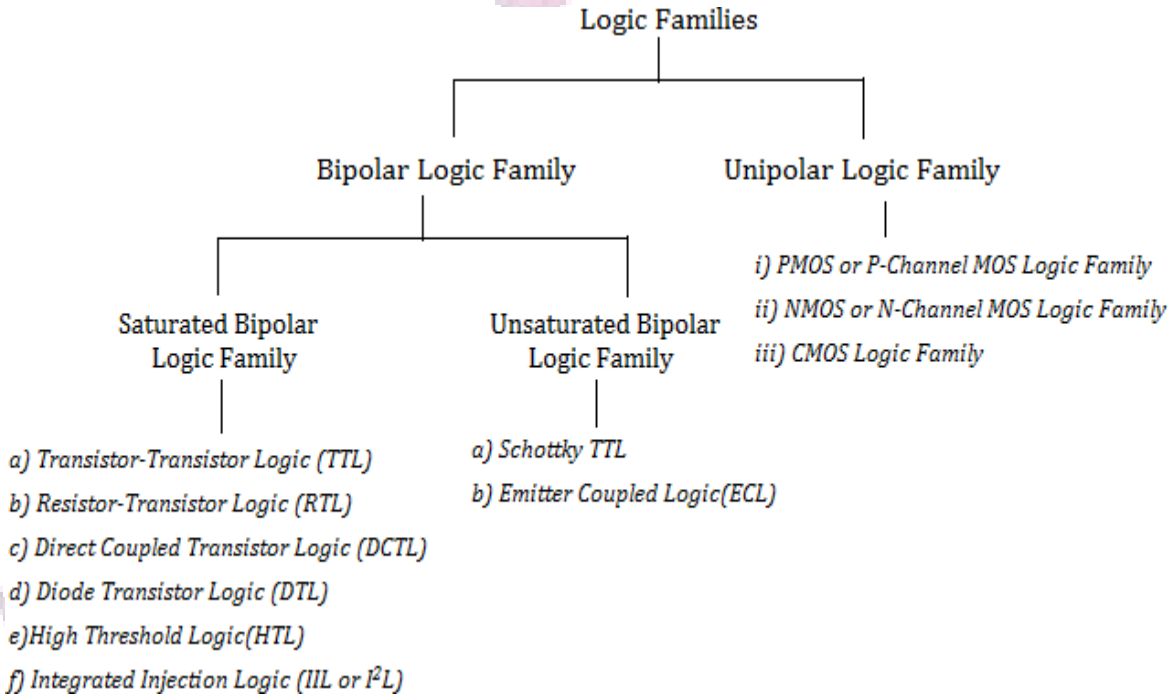


Fig.1 Classification of Logic Families

CMOS Logic: The basic building blocks in CMOS logic circuits are MOS transistors. All circuits that are implemented by CMOS logic have Basic CMOS circuit which will form by the Complementary connection of NMOS and PMOS transistors. So that this logic is named as *Complementary Metal Oxide Semiconductor Logic.*

Metal Oxide Semiconductor transistor:

 A MOS transistor contains 4 terminals named as Gate(G), Source(S), Drain(D) and Substrate(Sb).

 Among 4 terminals Gate is an insulating terminal. So no conduction will takes place between remaining two(S and D) terminals. Hence it has highest resistance between Source and Drain.

 The voltage applied at the Gate terminal may create electric field that enhances or retards the flow of current between Source and Drain. Due to this field effect MOS transistors are called *Field Effect Transistors (FET)*.

 Here the Resistance between Source and Drain also controlled by voltage applied at Gate terminal hence MOSFETs also called as *Voltage Controlled Resistors*.

 MOS transistors are classified into two types based on the use of the channel type as N-Channel or NMOS transistor and P-Channel or PMOS transistor.

 Both N-Channel or NMOS transistor and P-Channel or PMOS transistor are again sub divided into two types based on their mode of operation as Enhancement mode and Deletion mode transistors

 In Enhancement mode of operation a channel is developed between two terminals Source and Drain of respective MOS transistors.ie. If it is NMOS transistor then N- Channel and it is PMOS then PChannel transistor by applying required voltage at Gate terminal.

 In Depletion mode of operation the already existed channel will be removed between two terminals Source and Drain of respective MOS transistors.ie. If it is NMOS transistor then NChannel and it is PMOS then P-Channel transistor by applying required voltage at Gate terminal.

 In N-MOS Transistor if input voltage V_{gs} is Zero then resistance between Source and Drain R_{ds} is very high in terms of MΩ and if V_{gs} is more positive voltage then R_{ds} is very low in terms of (0-10) Ω.

 In P-MOS Transistor if input voltage V_{gs} is Zero then resistance between Source and Drain R_{ds} is very high in terms of MΩ and if V_{gs} is more negative voltage then R_{ds} is very low in terms of (0-10) Ω.

Note: A small current will flows through high resistance between Gate and Source or Drain junctions.

 Explain the Operation of basic CMOS circuit?

Basic CMOS

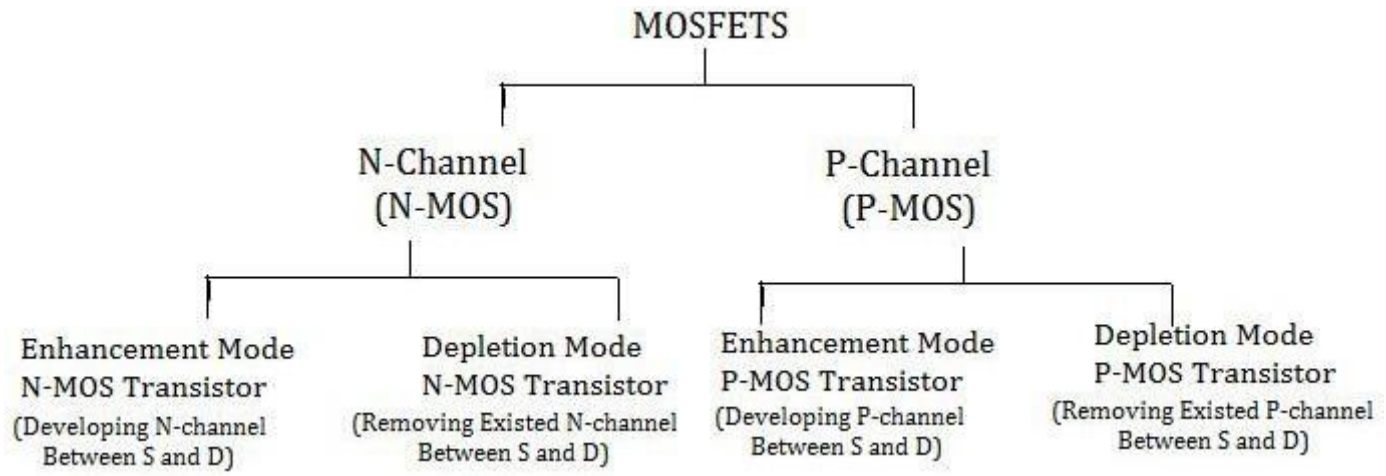


Fig.Classification of MOSFETs

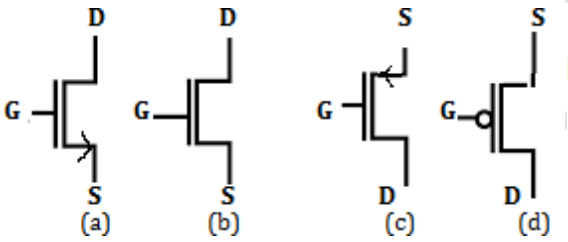
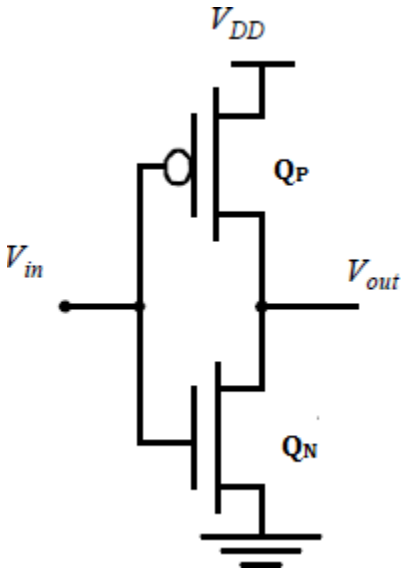


Fig.Symbols of NMOS(a,b) and PMOS(c,d) Transistors

Basic NMOS and PMOS transistors that connected in a complementary connection to form CMOS logic circuit.

- It contains pull-up and pull-down networks.
- Pull-up network contains PMOS transistor and pull-down network consists of NMOS Transistor.
- When input applied as logic '0'(L) the PMOS transistor is in ON condition and that translates output to logic '1'(H), i.e., applied voltage is pulled up to 5V(H) from 0V(L) by PMOS transistor. Hence it is called as *Pull-Up transistor*.
- When input applied as logic '1'(H) the NMOS transistor is in ON condition and that translates output to logic '0'(L), i.e., applied voltage is pulled down to 0V(L) from 5V(H) by NMOS transistor. Hence it is called as *Pull-Down transistor*.



Operation:

Case (i): When V_{in}= 0 V, then NMOS transistor is OFF state, since its V_{GS} = 0 V. PMOS is in ON state Since its V_{GS} is large negative (-5V). So PMOS presents only a small resistance between V_{DD} and output. Hence output is 5 V.

Case (ii): When V_{in}= 5 V, then PMOS transistor is in OFF state, since its V_{GS} = 0 V. NMOS is in ON state Since its V_{GS} is large positive (+5 V). So NMOS presents only a small resistance between output and ground. Hence output is 0 V.

Note: From case (i) & (ii) we can conclude that the operation of a basic CMOS circuit gives INVERTING operation.

2 Input NAND gate using CMOS logic

NAND gate is one of the basic logic gates to perform the digital operation on the input signals.

 It is the combination of AND Gate followed by NOT gate i.e. it is the opposite operation of AND gate where the Logic NAND gate is complementary of AND gate.

 The logic output of NAND gate is low(FALSE) only when the inputs are high (TRUE).

 To implementation 2 Input NAND gate using CMOS logic we require 2 pull-up PMOS and 2 pull-down NMOS transistors.

Operation:

Case (i): When A=B=0 V, then both NMOS transistors are in OFF state, since its V_{GSA} = V_{GSB} = 0 V. Both PMOS transistors (P_A, P_B) are in ON state. Since its V_{GSA} and V_{GSB} voltage is large negative (-5V). So PMOS transistors presents only a small resistance between V_{DD} and output. Hence output is 5 V.

Case (ii): When A=B=5 V, then both PMOS transistors are in OFF state, since its input voltages V_{GSA} = V_{GSB} = 0 V. Both NMOS transistors (N_A, N_B) are in ON state Since its V_{GSA} and V_{GSB} is large positive (+5V). So NMOS transistors presents only a small resistance between Output and ground. Hence output is 0 V.

Case (iii): When A=0 V and B=5 V, then NMOS transistor (N_A) is in OFF state, since its input voltage V_{GSA}= 0 V and NMOS transistor (N_B) is in ON state since its V_{GSB}=5V. PMOS transistor (P_A) is in ON state, since its V_{GSA}= -5V and PMOS transistor (P_B) is in OFF state since its V_{GSB}=0V. So PMOS transistor P_A presents only a small resistance between V_{DD} and output. Hence output is 5 V.

Case (iv): When A=5 V and B=0 V, then NMOS transistor (N_A) is in ON state, since its input voltage V_{GSA}= +5V and NMOS transistor (N_B) is in OFF state since its V_{GSB}=0V. PMOS transistor (P_A) is in OFF state, since its V_{GSA}= 0V and PMOS transistor (P_B) is in ON state since its V_{GSB}=-5V. So PMOS transistor P_B presents only a small resistance between V_{DD} and output. Hence output is 5 V.

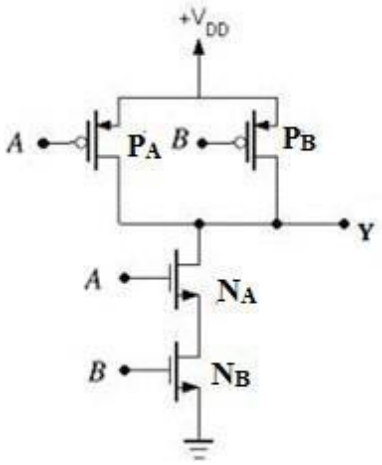


Fig: Two Input NOR gate circuit using CMOS logic

NOTE: *For a given Si Area N-Channel Transistor has lower ON resistance than P-Channel Transistor. So, the circuit having series N-Channel connection is more faster than Parallel PChannel, Hence NAND circuit is faster than NOR.*

Transistor-Transistor Logic Families are classified as following:

(a) Early TTL families
- 74L Low power
- 74H High speed

(b) Schottky TTL families
- 74S Schottky
- 74LS Low power Schottky
- 74AS Advanced Schottky
- 74ALS Advanced Low power Schottky
- 74F Fast

(a) Early TTL families:

- The original TTL family of logic gates was introduced by Sylvania in 1963. It was popularized by Texas Instruments, whose "7400-series" part numbers for gates and other TTL components quickly became an industry standard.
- As in 7400-series CMOS, devices in a given TTL family have part numbers of the form 74FAMnn, where "FAM" is an alphabetic family mnemonic and nn is a numeric function designator. Devices in different families with the same value of nn perform the same function.

In the original TTL family, "FAM" is null and the family is called 74-series TTL.

- The 74H (High speed TTL) family used lower resistor values to reduce propagation delay at the expense of increased power consumption.
- The 74L (Low-power TTL) family used higher resistor values to reduce power consumption at the expense of propagation delay.

(b) Schottky TTL families:

- The first family to make use of Schottky transistors was 74S (Schottky TTL). With Schottky

transistors and low resistor values, this family has much higher speed, but higher power consumption, than the original 74-series TTL.

- 74LS (Low-power Schottky TTL), introduced shortly after 74S. By combining Schottky transistors with higher resistor values, 74LS TTL matches the speed of 74- series TTL but has about one-fifth of its power consumption.
- Thus, 74LS is a preferred logic family for new TTL designs. Subsequent IC processing and circuit innovations gave rise to two more Schottky logic families as 74AS (Advanced Schottky TTL) and 74ALS (Advanced Low-power Schottky TTL).
- The 74AS (Advanced Schottky TTL) family offers speeds approximately twice as fast as 74S with approximately the same power consumption.
- The 74ALS (Advanced Low-power Schottky TTL) family offers both lower power and higher speeds than 74LS, and rivals 74LS in popularity for general-purpose requirements in new TTL designs.
- The 74F (Fast TTL) family is positioned between 74AS and 74ALS in the speed/power tradeoff, and is probably the most popular choice for high-speed requirements in new TTL designs.

Characteristics of TTL logic families:

- The important characteristics of TTL families are summarized in Table 4.17.
- The first two rows of the table list the propagation delay (in nanoseconds) and the power consumption (in milliWatts) of a typical 2-input NAND gate in each family.
- One figure of merit of a logic family is its *speed-power product* listed in the third row of the table.
- The remaining rows have values of voltage levels for all TTL families.

	74	74S	74LS	74AS	74ALS	74F
Performance ratings						
Propagation delay (ns)	9	3	9.5	1.7	4	3
Power dissipation (mW)	10	20	2	8	1.2	6
Max. clock rate (MHz)	35	125	45	200	70	100
Fan-out (same series)	10	20	20	40	20	33
Speed-Power product(pj)	90	60	19	13.6	4.8	18
Voltage parameters						
V_{OH}(min) (V)	2.4	2.7	2.7	2.5	2.5	2.5
V_{OL}(max) (V)	0.4	0.5	0.5	0.5	0.5	0.5
V_{IH}(min) (V)	2.0	2.0	2.0	2.0	2.0	2.0
V_{II} (max) (V)	0.8	0.8	0.8	0.8	0.8	0.8

Characteristics of TTL logic families.

INTERFACING: Connecting outputs of one circuit to inputs of another circuit that may have

different electrical characteristics.

- If two circuits that are going to interface have different Electrical characteristics, then direct contact can't be made.
- In such cases Driver and Load circuits are connected through INTERFACE. Interface circuitry shifts levels of voltage & current for compatibility.
- Driver output signal must satisfy the requirements of load circuit.
- If both driver and load require different power supplies, then outputs of both circuits must swing between its specified voltage ranges
- The interfacing may done in between two different logic families or with in the same logic families.
- Interfacing in between CMOS and TTL logic families is achieved in ways as

(a) TTL Driving CMOS circuits.
(b) CMOS driving TTL circuits.

Boolean Function

Boolean algebra is an algebra that deals with binary variables and logic operations.

A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.

For a given value of the binary variables, the function can be equal to either 1 or 0.

F(vars) = *expression*

Set of binary Variables

Operators (+, •, ')
Constants (0, 1)
Groupings (parenthesis)
Variables

Consider an example for the Boolean function

$$F1 = x + y'z$$

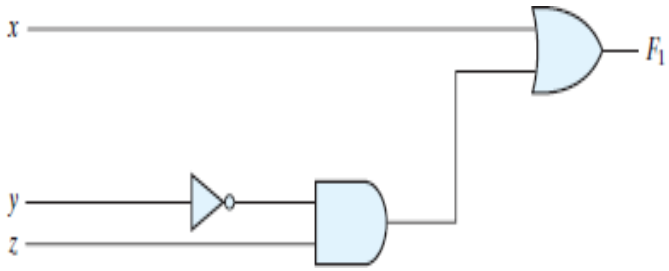
The function F1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F1 is equal to 0 otherwise. The complement operation dictates that when y' = 1, y = 0. Therefore, F1 = 1 if x = 1 or if y = 0 and z = 1.

A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n, where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

Truth Table for F1

x	y	z	F_1
0	0	0	0
0	0	1	1



Gate Implementation of F1 = x + y'z

Note:

Q: Let a function F() depend on n variables. How many rows are there in the truth table of F() ?

A: 2^n rows, since there are 2^n possible binary patterns/combinations for the n variables.

Truth Tables

- Enumerates all possible combinations of variable values and the corresponding function value
- Truth tables for some arbitrary functions
  F1(x,y,z), F2(x,y,z), and F3(x,y,z) are shown to the below.

x	y	z	F_1	F_2	F_3
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	1	0	1

- Truth table: a unique representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- Truth tables can be used to prove equality theorems.
- However, the size of a truth table grows exponentially with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

Boolean expressions-NOT unique

Unlike truth tables, expressions epresenting a Boolean function are NOT unique.

- Example:
  - F(x,y,z) = x'•y'•z' + x'•y•z' + x•y•z'
  - G(x,y,z) = x'•y'•z' + y•z'
- The corresponding truth tables for F() and G() are to the right. They are identical.
- Thus, F() = G()

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

Algebraic Manipulation (Minimization of Boolean function)

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- Example: Simplify F = x'yz + x'yz' + xz.

  F= x'yz + x'yz' + xz
  = x'y(z+z') + xz
  = x'y•1 + xz
  = x'y + xz

- Example: Prove

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

- **Proof:**

  x'y'z'+ x'yz'+ xyz'
  = x'y'z' + x'yz' + x'yz' + xyz'
  = x'z'(y'+y) + yz'(x'+x)
  = x'z'•1 + yz'•1
  = x'z' + yz'

Complement of a Function

- The complement of a function is derived by interchanging (• and +), and (1 and 0), and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The *complement* of a function IS NOT THE SAME as the *dual* of a function.

Example

- Find G(x,y,z), the complement of F(x,y,z) = xy'z' + x'yz

  Ans: G = F' = (xy'z' + x'yz)'
  = (xy'z')' • (x'yz)' *DeMorgan*
  = (x'+y+z) • (x+y'+z') *DeMorgan* again

Note: The complement of a function can also be derived by finding the function's *dual,* and then complementing all of the literals

Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.
Identical functions will have exactly the same canonical form.

- Minterms and Maxterms
- Sum-of-Minterms and Product-of- Maxterms
- Product and Sum terms
- Sum-of-Products (SOP) and Product-of-Sums (POS)

Definitions

Literal: A variable or its complement
Product term: literals connected by •
Sum term: literals connected by +
Minterm: a product term in which all the variables appear exactly once, either complemented or uncomplemented.
Maxterm: a sum term in which all the variables appear exactly once, either complemented or uncomplemented.
Canonical form: Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

Minterm

- Represents exactly one combination in the truth table.
- Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

- Represents exactly one combination in the truth table.
- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.

Example: Assume 3 variables x,y,z (order is fixed)

x	y	z	Minterm	Maxterm
0	0	0	$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1	$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0	$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1	$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0	$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1	$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0	$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1	$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms

- Every function F() has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Sum-Of-Products:

The minterms included are those m_j such that F( ) = 1 in row j of the truth table for F( ).

Canonical Product-Of-Sums:

The maxterms included are those M_j such that F( ) = 0 in row j of the truth table for F( ).

Example

Consider a Truth table for $f_1(a,b,c)$ at right

The canonical sum-of-products form for f_1 is

$f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

$= a'b'c + a'bc' + ab'c' + abc'$

The canonical product-of-sums form for f_1 is

$f_1(a,b,c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$

$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c').$

- Observe that: $m_j = M_j'$

a	b	c	f_1
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	1
1	1	1	0

Shorthand: ∑ and ∏

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where ∑ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where ∏ indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$
-

Conversion between Canonical Forms

- Replace ∑ with ∏ (or *vice versa*) and replace those *j*'s that appeared in the original form with those that do not.
    - Example:

$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$

$= m_1 + m_2 + m_4 + m_6$

$= \sum(1,2,4,6)$

$= \prod(0,3,5,7)$

$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c')$

Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

$F1 = y' + xy + x'yz'$

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

$F2 = x(y' + z)(x' + y + z')$

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

Conversion of SOP from standard to canonical form

Example-1.

Express the Boolean function F = A + B'C as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

A = AB(C + C') + AB' (C + C')

= ABC + ABC' + AB'C + AB'C'

The second term B'C is missing one variable; hence,

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

Combining all terms, we have

F = A + B'C

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

But AB'C appears twice, and according to theorem (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

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

= m1 + m4 + m5 + m6 + m7

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

F(A, B, C) = ∑m (1, 4, 5, 6, 7)

Example-2.

Express the Boolean function F = xy + x'z as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

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

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

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

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

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

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

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

Combining all the terms and removing those which appear more than once, we finally obtain

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

F= M0M2M4M5

A convenient way to express this function is as follows:

F(x, y, z) = πM(0, 2, 4, 5)

The product symbol, π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Digital Logic Gates

Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR		$F = x + y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inverter		$F = x'$	$x \mid F$ $0 \mid 1$ $1 \mid 0$
Buffer		$F = x$	$x \mid F$ $0 \mid 0$ $1 \mid 1$
NAND		$F = (xy)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR		$F = (x + y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

Properties of XOR Gates

- XOR (also ⊕) : the "not-equal" function
- XOR(X,Y) = X ⊕ Y = X'Y + XY'
- Identities:
  - X ⊕ 0 = X
  - X ⊕ 1 = X'
  - X ⊕ X = 0
  - X ⊕ X' = 1
- Properties:
  - X ⊕ Y = Y ⊕ X
  - (X ⊕ Y) ⊕ W = X ⊕ (Y ⊕ W)

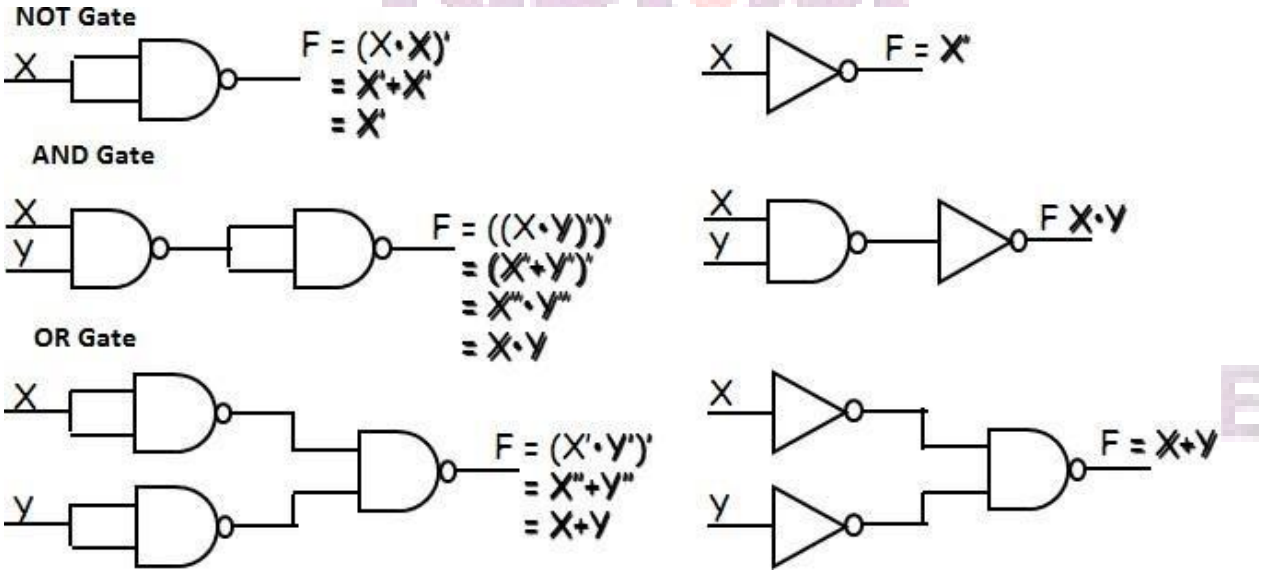
Universal Logic Gates

NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

NAND as a Universal Gate

NAND Known as a "universal" gate because ANY digital circuit can be implemented with NAND gates alone.

To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.



Unit-II

Combinational circuits

Two-variable k-map:

A two-variable k-map can have 2^2=4 possible combinations of the input variables A and B. Each of these combinations, $\bar{A}\bar{B}$,$\bar{A}$ B,A$\bar{B}$,AB(in the SOP form) is called a minterm. The minterm may be represented in terms of their decimal designations – m0 for $\bar{A}\bar{B}$, m1 for $\bar{A}$ B,m2 for A$\bar{B}$ and m3 for AB, assuming that A represents the MSB. The letter m stands for minterm and the subscript represents the decimal designation of the minterm. The presence or absence of a minterm in the expression indicates that the output of the logic circuit assumes logic 1 or logic 0 level for that combination of input variables.

The expression f=$\bar{A}\bar{B}$,+$\bar{A}$ B+A$\bar{B}$+AB , it can be expressed using min

term as F= m0+m2+m3=$\sum$m(0,2,3)

Using Truth Table:

Minterm	Inputs A	B	Output F
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

A 1 in the output contains that particular minterm in its sum and a 0 in that column indicates that the particular mintermdoes not appear in the expression for output . this information can also be indicated by a two-variable k-map.

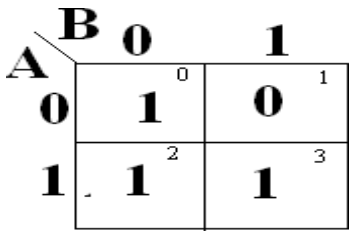
Mapping of SOP Expresions:

A two-variable k-map has 22=4 squares .These squares are called cells. Each square on the k-map represents a unique minterm. The minterm designation of the squares are placed in any square, indicates that the corresponding minterm does output expressions. And a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

A \ B	0	1
0	A'B'	A'B
1	AB'	AB

The minterms of a two-variable k-map

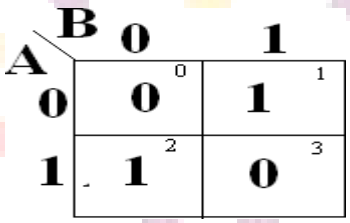
The mapping of the expressions =∑m(0,2,3)is



k-map of ∑m(0,2,3)

EX: Map the expressions f=*A* B+A*B*

F= $m_1+m_2=\sum m(1,2)$The k-map is



Minimizations of SOP expressions:

To minimize Boolean expressions given in the SOP form by using the k-map, look for adjacent adjacent squares having 1‘s minterms adjacent to each other, and combine them to form larger squares to eliminate some variables. Two squares are said to be adjacent to each other, if their minterms differ in only one variable. (i.e, *A* B & A*B* differ only in one variable. so they may be combined to form a 2-square to eliminate the variable B.similarly all other.

The necessary condition for adjacency of minterms is that their decimal designations must differ by a power of 2. A minterm can be combined with any number of minterms adjacent to it to form larger squares. Two minterms which are adjacent to each other can be combined to form a bigger square called a 2-square or a pair. This eliminates one variable – the variable that is not common to both the minterms. For EX:

m0 and m1 can be combined to yield,

f_1 = m0+m1=*AB*+*A* B=*A* (B+*B*

)=*A* m0 and m2 can be combined to yield,

f_2 = m0+m2=*A B*+*AB*=*B*(*A* + *A*)=*B*

m1 and m3 can be combined to yield,

f_3= m1+m3=A B+AB=B($A + A$)=B

m2 and m3 can be combined to yield,

f_4 = m2+m3=AB+AB=A(B+B)=A

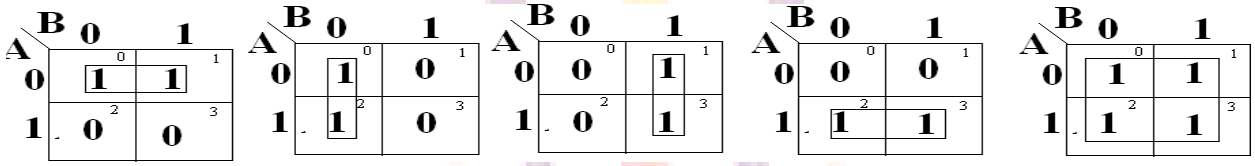
m_0 ,m_1 ,m_2 and m_3 can be combined to yield,

=A B+AB +AB+AB

=A (B+B) +A(B+B)

=A +A

=1

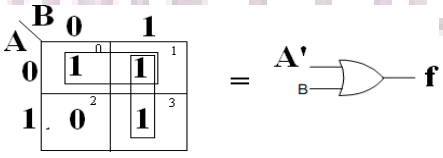


f1=A f2=B f3=B f4=A f5=1

The possible minterm groupings in a two-variable k-map.

Two 2-squares adjacent to each other can be combined to form a 4-square. A 4-square eliminates 2 variables. A 4-square is called a quad. To read the squares on the map after minimization, consider only those variables which remain constant through the square, and ignore the variables which are varying. Write the non complemented variable if the variable is remaining constant as a 1, and the complemented variable if the variable is remaining constant as a 0, and write the variables as a product term. In the above figure f_1 read as$A$, because, along the square , A remains constant as a 0, that is , as A , where as B is changing from 0 to 1.

EX: Reduce the minterm f=A B +AB +AB using mapping Expressed in terms of minterms, the given expression is F=m_0+m_1+m_2+ m_3=m$\sum$(0,1,3)& the figure shows the k-map for f and its reduction . In one 2-square, A is constant as a 0 but B varies from a 0 to a 1, and in the other 2-square, B is constant as a 1 but A varies from a 0 to a 1. So, the reduced expressions is A +B.



It requires two gate inputs for realization as

f=A +B (k-map in SOP form, and logic diagram.)

The main criterion in the design of a digital circuit is that its cost should be as low as possible. For that the expression used to realize that circuit must be minimal.Since the cost is proportional to number of gate inputs in the circuit in the circuit, an expression is considered minimal only if it corresponds to the least possible number of gate inputs. & there is no guarantee for that k-map in SOP is the real minimal. To obtain real minimal expression, obtain the minimal expression both in SOP & POS form form by using k-maps and take the minimal of these two minimals.

The 1‘s on the k-map indicate the presence of minterms in the output expressions, where as the 0s indicate the absence of minterms .Since the absence of a minterm in the SOP expression means the presense of the corresponding maxterm in the POS expression of the same .when a SOP expression is plotted on the k-map, 0s or no entries on the k-map represent the maxterms. To obtain the minimal expression in the POS form, consider the 0s on the k-map and follow the procedure used for combining 1s. Also, since the absence of a maxterm in the POS expression means the presence of the corresponding minterm in the SOP expression of the same , when a POS expression is plotted on the k-map, 1s or no entries on the k-map represent the minterms.

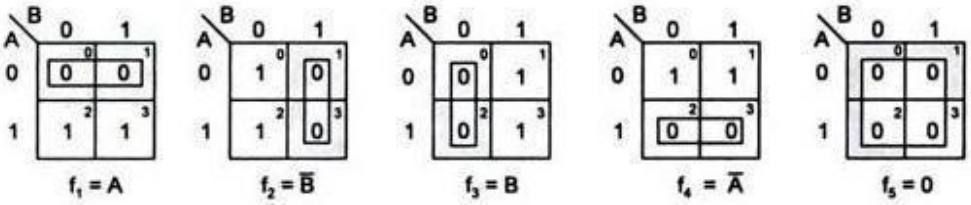
Mapping of POS expressions:

Each sum term in the standard POS expression is called a maxterm. A function in two variables (A, B) has four possible maxterms, A+B,A+$\overline{B}$,$\overline{A}$ +B,$\overline{A}$ +$\overline{B}$

. They are represented as M_0, M_1, M2, and M3respectively. The uppercase letter M stands for maxterm and its subscript denotes the decimal designation of that maxterm obtained by treating the non-complemented variable as a 0 and the complemented variable as a 1 and putting them side by side for reading the decimal equivalent of the binary number so formed.

For mapping a POS expression on to the k-map, 0s are placed in the squares corresponding to the maxterms which are presented in the expression an d1s are placed in the squares corresponding to the maxterm which are not present in the expression. The decimal designation of the squares of the squares for maxterms is the same as that for the minterms. A two-variable k-map & the associated maxterms are asthe maxterms of a two-variable k-map

The possible maxterm groupings in a two-variable k-map



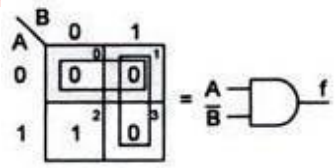
Minimization of POS Expressions:

To obtain the minimal expression in POS form, map the given POS expression on to the K-map and combine the adjacent 0s into as large squares as possible. Read the squares putting the complemented variable if its value remains constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square (ignoring the variables which do not remain constant throughout the square) and then write them as a sum term.

Various maxterm combinations and the corresponding reduced expressions are shown in figure. In this f_1 read as A because A remains constant as a 0 throughout the square and B changes from a 0 to a 1. f_2 is read as B' because B remains constant along the square as a 1 and A changes from a 0 to a 1. f_5
Is read as a 0 because both the variables are changing along the square.

Ex: Reduce the expression f=(A+B)(A+B')(A'+B') using mapping.

The given expression in terms of maxterms is $f=\pi M(0,1,3)$. It requires two gates inputs for realization of the reduced expression as



F=AB'

K-map in POS form and logic diagram

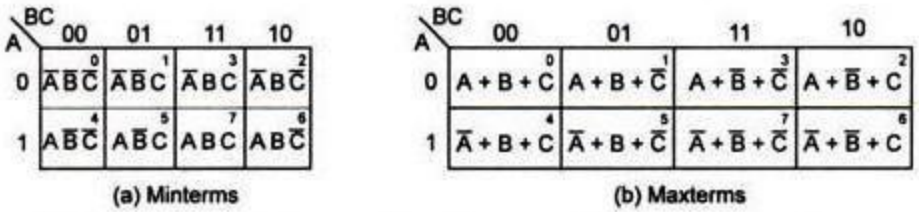
In this given expression ,the maxterm M_2 is absent. This is indicated by a 1 on the k-map. The corresponding SOP expression is $\sum m_2$ or AB'. This realization is the same as that for the POS form.

Three-variable K-map:

A function in three variables (A, B, C) expressed in the standard SOP form can have eight possible combinations: A B C , AB C,A BC ,A BC,AB C ,AB C,ABC , and ABC. Each one of these combinations designate d by m0,m1,m2,m3,m4,m5,m6, and m7, respectively, is called a minterm. A is the MSB of the minterm designator and C is the LSB.

In the standard POS form, the eight possible combinations are:A+B+C, A+B+C , A+B +C,A+B +C ,A + B+ C,A +B +C ,A + B + C,A +B +C . Each oneof these combinations designated by M_0, M1, M2, M3, M4, M5, M6, and M7respectively is called a maxterm. A is the MSB of the maxterm designator and C is the LSB.

A three-variable k-map has, therefore, $8(=2^3)$ squares or cells, and each square on the map represents a minterm or maxterm as shown in figure. The small number on the top right corner of each cell indicates the minterm or maxterm designation.



The three-variable k-map.

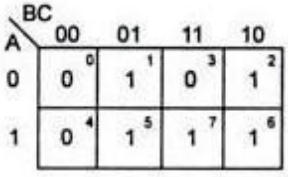
The binary numbers along the top of the map indicate the condition of B and C for each column. The binary number along the left side of the map against each row indicates the condition of A for that row. For example, the binary number 01 on top of the second column in fig indicates that the variable B appears in complemented form and the variable C in non-complemented form in all the minterms in that column. The binary number 0 on the left of the first row indicates that the variable A appears in complemented form in all the minterms in that row, the binary numbers along the top of the k-map are not in normal binary order. They are, infact, in the Gray code. This is to ensure that twophysically adjacent squares are really adjacent, i.e., their minterms or maxterms differ by only one variable.

Ex: Map the expression f=: $A\,B$C+$A\,BC$ +ABC+ABC +ABC

In the given expression , the minterms are : $A\,B$C=001=m_1 ; ABC=101=m_5; $A\,BC$ =010=m_2;

ABC =110=m_6;ABC=111=m_7.

So the expression is f=∑m(1,5,2,6,7)= ∑m(1,2,5,6,7). The corresponding k-map is

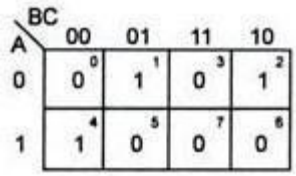


K-map in SOP form

Ex: Map the expression f= (A+B+C),$(A+B+C)$ $(A+B+C)$(A + $B+C$)$(A+B+C)$

In the given expression the maxterms are :A+B+C=000=M_0;$A+B+C$ =101=M_5;$A+B+C=111$=M_7; A + $B+C$ =011=M_3;$A+B+C$=110=M_6.

So the expression is f = π M (0,5,7,3,6)= π M (0,3,5,6,7). The mapping of the expression is



K-map in POS form.

Minimization of SOP and POS expressions:

For reducing the Boolean expressions in SOP (POS) form plotted on the k-map, look at the 1s (0s) present on the map. These represent the minterms (maxterms). Look for the minterms (maxterms) adjacent to each other, in order to combine them into larger squares. Combining of adjacent squares in a k-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS)expression is called *looping*. Some of the minterms (maxterms) may have many adjacencies. Always start with the minterms (maxterm) with the least number of adjacencies and try to form as large as large a square as possible. The larger must form a geometric square or rectangle. They can be formed even by wrapping around, but cannot be formed by using diagonal configurations. Next consider the minterm (maxterm) with next to the least number of adjacencies and form as large a square as possible. Continue this till all the minterms (maxterms) are taken care of . A minterm (maxterm) can be part of any number of squares if it is helpful in reduction. Read the minimal expression from the k-map, corresponding to the squares formed. There can be more than one minimal expression.

Two squares are said to be adjacent to each other (since the binary designations along the top of the map and those along the left side of the map are in Gray code), if they are physically adjacent to each other, or can be made adjacent to each other by wrapping around. For squares to be combinable into bigger squares it is essential but not sufficient that their minterm designations must differ by a power of two.

General procedure to simplify the Boolean expressions:

1. Plot the k-map and place 1s(0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
2. Check the k-map for 1s(0s) which are not adjacent to any other 1(0). They are isolated minterms(maxterms) . They are to be read as they are because they cannot be combined even into a 2-square.
3. Check for those 1s(0S) which are adjacent to only one other 1(0) and make them pairs (2 squares).
4. Check for quads (4 squares) and octets (8 squares) of adjacent 1s (0s) even if they contain some 1s(0s) which have already been combined. They must geometrically form a square or a rectangle.
5. Check for any 1s(0s) that have not been combined yet and combine them into bigger squares if possible.
6. Form the minimal expression by summing (multiplying) the product the product (sum) terms of all the groups.

Reading the K-maps:

While reading the reduced k-map in SOP (POS) form, the variable which remains constant as 0 along the square is written as the complemented (non-complemented) variable and the one which remains constant as 1 along the square is written as non-complemented (complemented) variable and the term as a product (sum) term. All the product (sum) terms are added (multiplied).

Some possible combinations of minterms and the corresponding minimal expressions readfrom the k-maps are shown in fig: Here f_6 is read as 1, because along the 8-square no variable remains constant. F_5 is read as A , because, along the 4-square formed by0,m_1,m_2 and m_3 , the variables B and C are changing, and A remains constant as a 0. Algebraically,

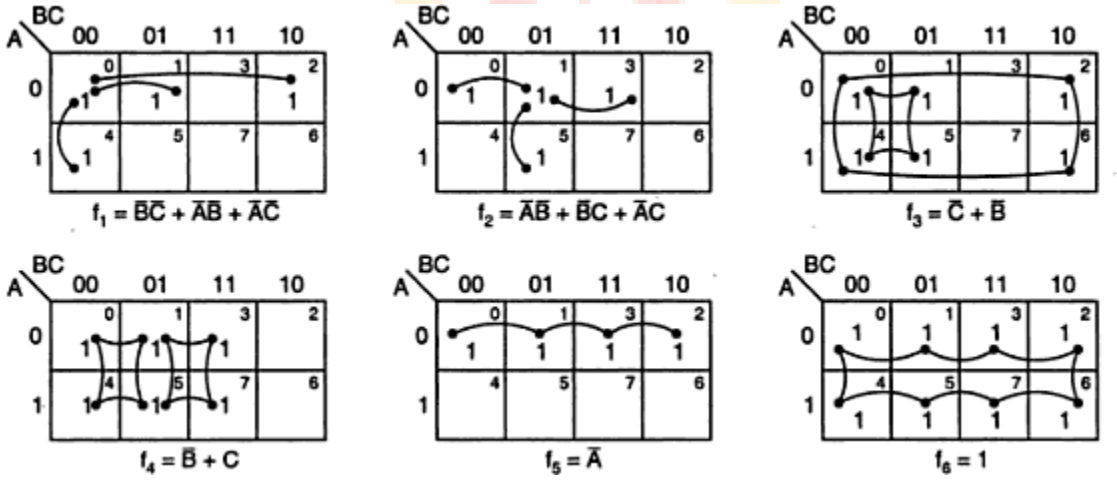
$f_5 = m_0+m_1+m_2+m_3$

$= A\,BC + A\,B\mathrm{C} + A\,BC + A\,BC$

$= A\,B(C + \mathrm{C}) + A\,\mathrm{B}(\mathrm{C} + C)$

$= A\,B + A\,\mathrm{B}$

$= A\,(B + \mathrm{B}) = A$

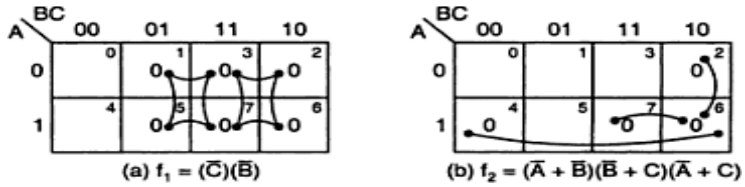


f_3 is read as $C + B$, because in the 4-square formed by m0,m2,m6, and m4, the variable A and B are changing , where as the variable C remains constant as a 0. So it is read as C . In the 4-square formed by m_0, m1, m4, m_5, A and C are changing but B remains constant as a 0. So it is read as$B$. So, the resultant expression for f_3 is the sum of these two, i.e., $C + B$.

f_1 is read as $BC + A\,B + A\,C$,because in the 2-square formed by m_0 and m_4 , A is changing from a 0 to a 1. Whereas B and C remain constant as a 0. So it s read as $B\,C$. In the 2-square formed by m_0 and m_1, C is changing from a 0 to a 1, whereas A and B remain constant as a 0. So it is read as $A\,B$ .In the 2-square formed by m_0 and m_2 , B is changing from a 0 to a 1 whereas A and C remain constant as a 0. So, it is read as$A\,C$. Therefore, the resultant SOP expression is

$BC + A\,B + BC$

Some possible maxterm groupings and the corresponding minimal POS expressions read from the k-map are



In this figure, along the 4-square formed by M_1, M3, M7, M5, A and B are changing from a 0 to a 1, where as C remains constant as a 1. SO it is read as C . Along the 4-squad formed by M_3, M2, M7, and M_6, variables A and C are changing from a 0 to a 1. But B remains constant as a 1. So it is read as B. The minimal expression is the product of these two terms , i.e., $f_1 = (C)(B)$ .also in this figure, along the 2-square formed by M_4 and M_6 , variable B is changing from a 0 to a 1, while variable A remains constant as a 1 and variable C remains constant as a 0. SO, read it as

A +C. Similarly, the 2-square formed by M_7 andM_6 is read as $A + B$, while the 2-square formed by M_2 and M_6 is read as B+C. The minimal expression is the product of these sum terms, i.e, f_2 $=(A + C)+(A + B)+(B +C)$

Ex:Reduce the expression f=∑m(0,2,3,4,5,6) using mapping and implement it in AOI logic as well as in NAND logic.The Sop k-map and its reduction , and the implementation of the minimal expression using AOI logic and the corresponding NAND logic are shown in figures below

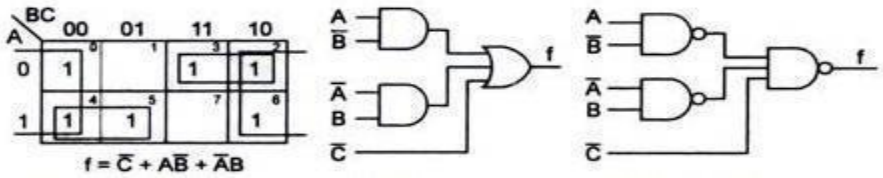
In SOP k-map, the reduction is done as:

1 m_5 has only one adjacency m_4 , so combine m_5 and m_4 into a square. Along this 2-square A remains constant as 1 and B remains constant as 0 but C varies from 0 to 1. So read it as AB.

2 m_3 has only one adjacency m_2 , so combine m_3 and m_2 into a square. Along this 2-square A remains constant as 0 and B remains constant as 1 but C varies from 1 to 0. So read it as A B.

3 m_6 can form a 2-square with m_2 and m_4 can form a 2-square with m_0, but observe that by wrapping the map from left to right m_0, m_4 ,m_2 ,m_6 can form a 4-square. Out of these m_2 andm_4 have already been combined but they can be utilized again. So make it. Along this 4-square, A is changing from 0 to 1 and B is also changing from 0 to 1 but C is remaining constant as 0. so read it as C .

4 Write all the product terms in SOP form. So the minimal SOP expression is



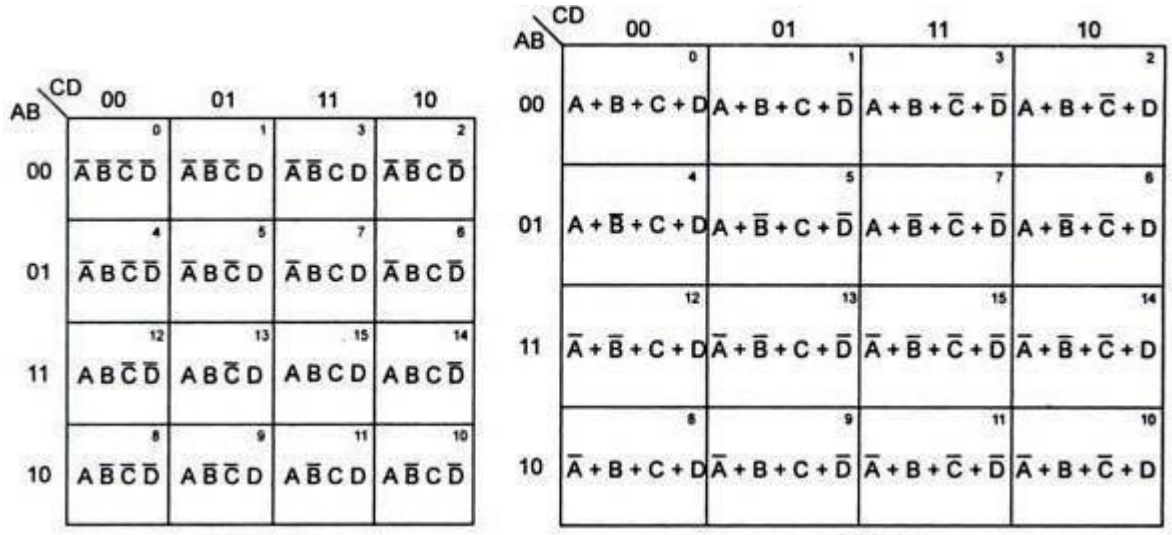
f_{min}=

k-map **AOI logic** **NAND logic**

Four variable k-maps:

Four variable k-map expressions can have 2^4=16 possible combinations of input variables such as $A\,BC\,D$,$A\,BC\,D$,----------ABCD with minterm designations m_0,m_1 ---------- m_{15} respectively in SOP form & A+B+C+D, A+B+C+D ,----------$A+B+C+D$ with maxterms M_0,M_1, ----------

-M_{15} respectively in POS form. It has 2^4=16 squares or cells.The binary number designations of rows & columns are in the gray code. Here follows 01 & 10 follows 11 called Adjacency ordering.



AB \ CD	00	01	11	10
00	0 $\bar{A}\bar{B}\bar{C}\bar{D}$	1 $\bar{A}\bar{B}\bar{C}D$	3 $\bar{A}\bar{B}CD$	2 $\bar{A}\bar{B}C\bar{D}$
01	4 $\bar{A}B\bar{C}\bar{D}$	5 $\bar{A}B\bar{C}D$	7 $\bar{A}BCD$	6 $\bar{A}BC\bar{D}$
11	12 $AB\bar{C}\bar{D}$	13 $AB\bar{C}D$	15 $ABCD$	14 $ABC\bar{D}$
10	8 $A\bar{B}\bar{C}\bar{D}$	9 $A\bar{B}\bar{C}D$	11 $A\bar{B}CD$	10 $A\bar{B}C\bar{D}$

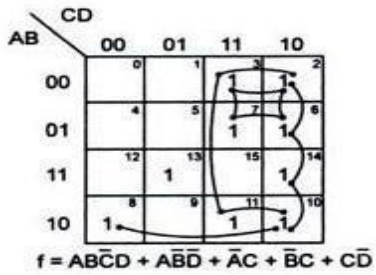
SOP form

AB \ CD	00	01	11	10
00	0 $A+B+C+D$	1 $A+B+C+\bar{D}$	3 $A+B+\bar{C}+\bar{D}$	2 $A+B+\bar{C}+D$
01	4 $A+\bar{B}+C+D$	5 $A+\bar{B}+C+\bar{D}$	7 $A+\bar{B}+\bar{C}+\bar{D}$	6 $A+\bar{B}+\bar{C}+D$
11	12 $\bar{A}+\bar{B}+C+D$	13 $\bar{A}+\bar{B}+C+\bar{D}$	15 $\bar{A}+\bar{B}+\bar{C}+\bar{D}$	14 $\bar{A}+\bar{B}+\bar{C}+D$
10	8 $\bar{A}+B+C+D$	9 $\bar{A}+B+C+\bar{D}$	11 $\bar{A}+B+\bar{C}+\bar{D}$	10 $\bar{A}+B+\bar{C}+D$

POS form

EX: Reduce using mapping the expression Σ m(2, 3, 6, 7, 8, 10, 11, 13, 14).

Start with the minterm with the least number of adjacencies. The minterm m_{13} has no adjacency. Keep it as it is. The m_8 has only one adjacency, m_{10}. Expand m_8 into a 2-square with m_{10}. The m_7 has two adjacencies, m_6 and m_3. Hence m_7 can be expanded into a 4-square with m_6, m_3 and m_2. Observe that, m_7, m_6, m_2, and m_3 form a geometric square. The m_{11} has 2 adjacencies, m_{10} and m_3. Observe that, m_{11}, m_{10}, m_3, and m_2 form a geometric square on wrapping the K-map. So expand m_{11} into a 4-square with m_{10}, m_3 and m_2. Note that, m_2 and m_3, have already become a part of the 4-square m_7, m_6, m_2, and m_3. But if m_{11} is expanded only into a 2-square with m_{10}, only one variable is eliminated. So m_2 and m_3 are used again to make another 4-square with m_{11} and m_{10} to eliminate two variables. Now only m_6 and m_{14} are left uncovered. They can form a 2-square that eliminates only one variable. Don't do that. See whether they can be expanded into a larger square. Observe that, m_2, m_6, m_{14}, and m_{10} form a rectangle. So m_6 and m_{14} can be expanded into a 4-square with m_2 and m_{10}. This eliminates two variables.



Five variable k-map:

Five variable k-map can have 2^5 =32 possible combinations of input variable as $A\ BC\ DE$,$A\ BC\ D$E,--------ABCDE with minterms m_0, m_1-----m_{31} respectively in SOP & A+B+C+D+E, A+B+C+$D\ E$,----------$A+B+C+D+E$ with maxterms M_0,M_1, ---------- M_{31} respectively in POS form. It has 2^5=32 squares or cells of the k-map are divided into 2 blocks of

16 squares each.The left block represents minterms from m_0 to m_{15} in which A is a 0, and the right block represents minterms from m_{16} to m_{31} in which A is 1.The 5-variable k-map may contain 2-squares, 4-squares , 8-squares , 16-squares or 32-squares involving these two blocks. Squares are also considered adjacent in these two blocks, if when superimposing one block on top of another, the squares coincide with one another.

Some possible 2-squares in a five-variable map are m_0, m_{16}; m_2, m_{18}; m_5, m_{21}; m_{15}, m_{31}; m_{11}, m_{27}.

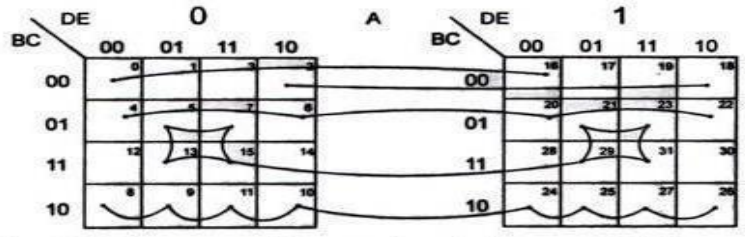
Some possible 4-squares are m_0, m_2, m_{16}, m_{18}; m_0, m_1, m_{16}, m_{17}; m_0, m_4, m_{16}, m_{20}; m_{13}, m_{15}, m_{29}, m_{31}; m_5, m_{13}, m_{21}, m_{29}.

Some possible 8-squares are m_0, m_1, m_3, m_2, m_{16}, m_{17}, m_{19}, m_{18}; m_0, m_4, m_{12}, m_8, m_{16}, m_{20}, m_{28}, m_{24}; m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31}.

The squares are read by dropping out the variables which change. Some possible

Grouping s is

(a) m_0, $m_{16} = \overline{B}\overline{C}\overline{D}\overline{E}$	M_0, $M_{16} = B + C + D + E$
(b) m_2, $m_{18} = \overline{B}\overline{C}D\overline{E}$	M_2, $M_{18} = B + C + \overline{D} + E$
(c) m_4, m_6, m_{20}, $m_{22} = \overline{B}C\overline{E}$	M_4, M_6, M_{20}, $M_{22} = B + \overline{C} + E$
(d) m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, $m_{31} = CE$	M_5, M_7, M_{13}, M_{15}, M_{21}, M_{23}, M_{29}, $M_{31} = \overline{C} + \overline{E}$
(e) m_8, m_9, m_{10}, m_{11}, m_{24}, m_{25}, m_{26}, $m_{27} = B\overline{C}$	M_8, M_9, M_{10}, M_{11}, M_{24}, M_{25}, M_{26}, $M_{27} = \overline{B} + C$



Ex: F=∑m(0,1,4,5,6,13,14,15,22,24,25,28,29,30,31) is SOP

POS is F=πM(2,3,7,8,9,10,11,12,16,17,18,19,20,21,23,26,27)

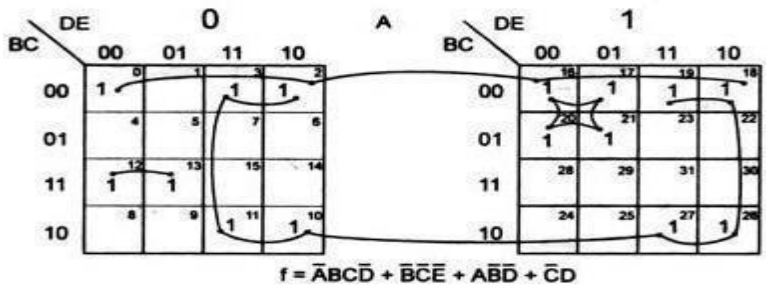
The real minimal expression is the minimal of the SOP and POS forms.

The reduction is done as

1. There is no isolated 1s
2. M_{12} can go only with m_{13}. Form a 2-square which is read as A‘BCD‘
3. M_0 can go with m_2,m_{16} and m_{18} . so form a 4-square which is read as B‘C‘E‘
4. M_{20},m_{21},m_{17} and m_{16} form a 4-square which is read as AB‘D‘
5. M2,m3,m18,m19,m10,m11,m26 and m27 form an 8-square which is read as C‘d
6. Write all the product terms in SOP form.

So the minimal expression is

F_{min}= A‘BCD‘+B‘C‘E‘+AB‘D‘+C‘D(16 inputs)



In the POS k-map ,the reduction is done as:

1. There are no isolated 0s

3. **M_1 can go only with M_5. So, make a 2-square, which is read as $(A + B + D + \bar{E})$.**
   M_4 can go with M_5, M_7, and M_6 to form a 4-square, which is read as $(A + B + \bar{C})$.

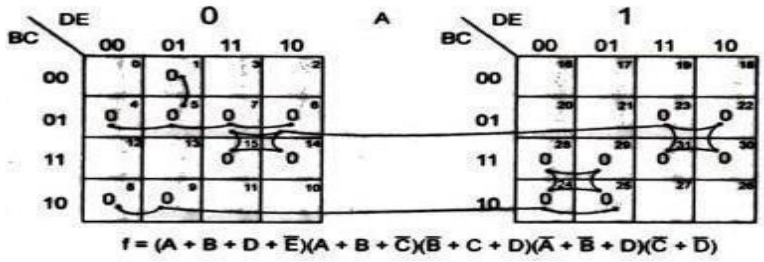
4.M_8

5. M_{28}

6.M_{30}

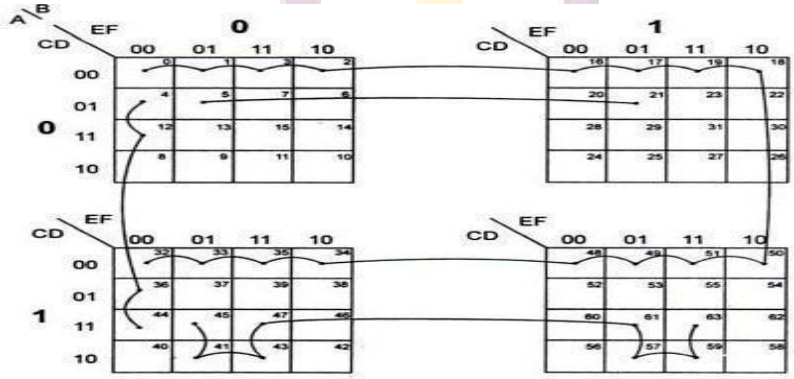
7. Sum terms in POS form. So the minimal expression in POS is

F_{min}= A‘BcD‘+B‘C‘E‘+AB‘D‘+C‘D



Six variable k-map:

Six variable k-map can have 2^6 =64 combinations as $A\,BC\,DEF, A\,BC\,DEF$,---------- ---ABCDEF with minterms m_0, m_1-----m_{63} respectively in SOP & (A+B+C+D+E+F), ---------- $(A + B + C + D + E + F)$ with maxterms M_0,M_1, ----------M_{63} respectively in POS form. It has 2^6=64 squares or cells of the k-map are divided into 4 blocks of 16 squares each.



Some possible groupings in a six variable k-map

Don't care combinations: For certain input combinations, the value of the output is unspecified either because the input combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the value of experiments are not specified are called don't care combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the value of expressions is not specified are called don't care combinations or Optional Combinations, such expressions stand incompletely specified. The output is a don't care for these invalid combinations.

Ex:In XS-3 code system, the binary states 0000, 0001, 0010,1101,1110,1111 are unspecified. & never occur called don't cares.

A standard SOP expression with don't cares can be converted into a standard POS form by keeping the don't cares as they are & writing the missing minterms of the SOP form as the maxterms of the POS form viceversa.

Don't cares denoted by _X' or _φ'

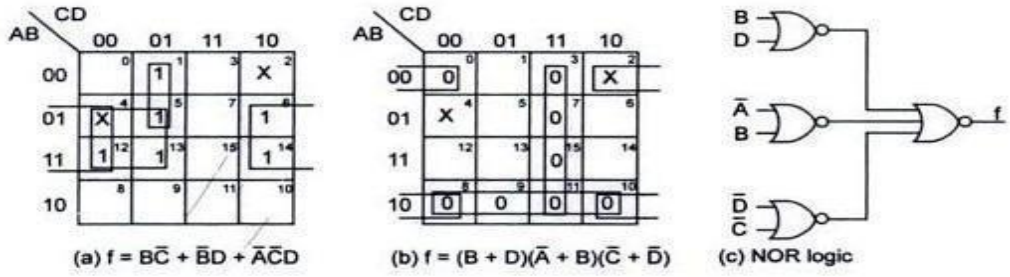
Ex:f=∑m(1,5,6,12,13,14)+d(2,4)

Or f=π M(0,3,7,9,10,11,15).πd(2,4)

SOP minimal form f_{min}= BC +BD+A CD

POS minimal form f_{min}=(B+D)(A +B)(C +D)

$$= B + D + A + B + (C + D$$



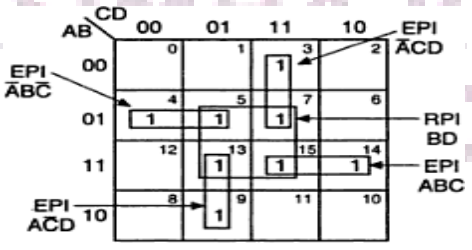
Prime implicants, Essential Prime implicants, Redundant prime implicants:

Each square or rectangle made up of the bunch of adjacent minterms is called a subcube. Each of these subcubes is called a Prime implicant (PI). The PI which contains at leastone which cannot be covered by any other prime implicants is called as Essential Prime implicant (EPI).The PI whose each 1 is covered at least by one EPI is called a Redundant Prime implicant (RPI). A PI which is neither an EPI nor a RPI is called a Selective Prime implicant (SPI).

The function has unique MSP comprising EPI is

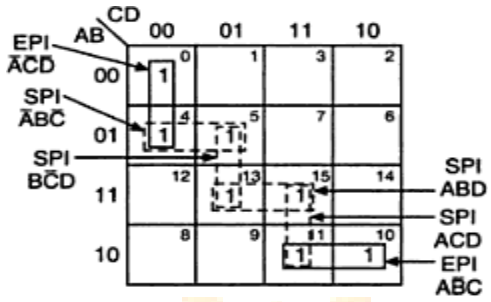
F(A,B,C,D)=A CD+ABC+AC D +A BC

The RPI _BD' may be included without changing the function but the resulting expression would not be in minimal SOP(MSP) form.



Essential and Redundant Prime Implicants

F(A,B,C,D)=∑m(0,4,5,10,11,13,15) SPI are marked by dotted squares, shows MSP form of a function need not be unique.



Essential and Selective Prime Implicants

Here, the MSP form is obtained by including two EPI‘s & selecting a set of SPI‘s to cover remaining uncovered minterms 5,13,15. & these can be covered as

(A) (4,5) &(13,15) --------- A BC +ABD
(B) (5,13) & (13,15) ------- BC D+ABD
(C) (5,13) & (15,11) ------ BC D+ACD

F(A,B,C,D)=A C D+ABC-------- EPI‘s +A BC +ABD

(OR) F(A,B,C,D)=A C D+ABC-------- EPI‘s + BC D+ABD

(OR) F(A,B,C,D)=A C D+ABC -------- EPI‘s + BC D+ACD

False PI’s Essential False PI’s, Redundant False PI’s & Selective False PI’s:

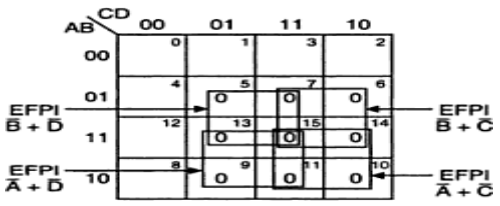
The maxterms are called falseminterms. The PI‘s is obtained by using the maxterms are called False PI‘s (FPI). The FPI which contains at least one _0‘ which can‘t be covered by only other FPI is called an Essential False Prime implicant (ESPI)

F(A,B,C,D)= ∑m(0,1,2,3,4,8,12)

=π M(5,6,7,9,10,11,13,14,15)

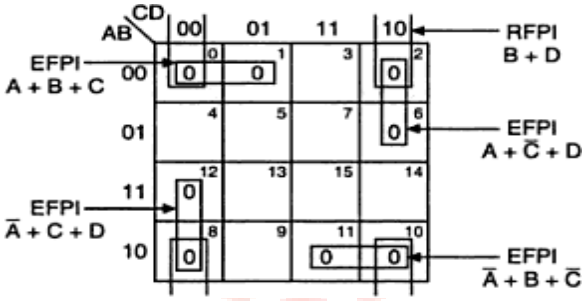
F_{min}= $(B+C)(A+C)(A+D)(B+D)$

All the FPI, EFPI‘s as each of them contain atleast one _0‘ which can‘t be covered by any other FPI



Essential False Prime implicants

Consider Function F(A,B,C,D)= π M(0,1,2,6,8,10,11,12)



Essential and Redundant False Prime Implicants

Mapping when the function is not expressed in minterms (maxterms):

An expression in k-map must be available as a sum (product) of minterms (maxterms). However if not so expressed, it is not necessary to expand the expression algebraically into its minterms (maxterms). Instead, expansion into minterms (maxterms) can be accomplished in the process of entering the terms of the expression on the k-map.

Limitations of Karnaugh maps:

- Convenient as long as the number of variables does not exceed six.
- Manual technique, simplification process is heavily dependent on the humanabilities.

Quine-Mccluskey Method:

It also known as *Tabular method.* It is more systematic method of minimizing expressions of even larger number of variables. It is suitable for hand computation as well as computation by machines i.e., programmable. . The procedure is based on repeated application of the combining theorem.

PA+P$\bar{A}$ =P (P is set of literals) on all adjacent pairs of terms, yields the set of all PI's from which a minimal sum may be selected.

Consider expression

$\sum$m(0,1,4,5)= $A\,BC$ +$A\,B$C+ABC +ABC

First, second terms & third, fourth terms can be combined

$A\,B(C + C\,)+A\,B(\mathrm{C}+C\,)=A\,B\,+\mathrm{A}B$

Reduced to

$B(A\, + A)=B$

The same result can be obtained by combining m_0& m_4 & m_1&m_5 in first step & resulting terms in the second step .

Procedure:

- Decimal Representation
- Don‘t cares
- PI chart
- EPI
- Dominating Rows & Columns
- Determination of Minimal expressions in comples cases.

Branching Method:

EXAMPLE 3.29 Obtain the set of prime implicants for the Boolean expression $f = \Sigma\, m(0, 1, 6, 7, 8, 9, 13, 14, 15)$ using the tabular method.

Solution

Group the minterms in terms of the number of 1s present in them and write their binary designations. The procedure to obtain the prime implicants is shown in Table 3.3.

Table 3.3 Example 3.29

	Column 1		Column 2		Column 3
	Minterm	**Binary designation**		**A B C D**	**A B C D**
Index 0	0	0 0 0 0 ✓	0, 1 (1)	0 0 0 – ✓	0, 1, 8, 9 (1, 8) – 0 0 – Q
Index 1	1	0 0 0 1 ✓	0, 8 (8)	– 0 0 0 ✓	... ...
	8	1 0 0 0 ✓	1, 9 (8)	– 0 0 1 ✓	... ...
Index 2	6	0 1 1 0 ✓	8, 9 (1)	1 0 0 – ✓	6, 7, 14, 15 (1, 8) – 1 1 – P
	9	1 0 0 1 ✓	6, 7 (1)	0 1 1 – ✓	
Index 3	7	0 1 1 1 ✓	6, 14 (8)	– 1 1 0 ✓	
	13	1 1 0 1 ✓	9, 13 (4)	1 – 0 1 S	
	14	1 1 1 0 ✓	7, 15 (8)	– 1 1 1 ✓	
Index 4	15	1 1 1 1 ✓	13, 15 (2)	1 1 – 1 R	
			14, 15 (1)	1 1 1 – ✓	

Comparing the terms of index 0 with the terms of index 1 of column 1, $m_0(0000)$ is combined with $m_1(0001)$ to yield 0, 1 (1), i.e. 000 –. This is recorded in column 2 and 0000 and 0001 are checked off in column 1. $m_0(0000)$ is combined with $m_8(1000)$ to yield 0, 8 (8), i.e. – 000. This is recorded in column 2 and 1000 is checked off in column 1. Note that 0000 of column 1 has already been checked off. No more combinations of terms of index 0 and index 1 are possible. So, draw a line below the last combination of these groups, i.e. below 0, 8 (8), – 000 in column 2. Now 0, 1 (1), i.e. 000 – and 0, 8 (8), i.e. – 000 are the terms in the first group of column 2.

Comparing the terms of index 1 with the terms of index 2 in column 1, $m_1(0001)$ is combined with $m_9(1001)$ to yield 1, 9 (8), i.e. – 001. This is recorded in column 2 and 1001 is checked off in column 1 because 0001 has already been checked off. $m_8(1000)$ is combined with $m_9(1001)$ to yield 8, 9 (1), i.e. 100 –. This is recorded in column 2. 1000 and 1001 of column 1 have already been checked off. So, no need to check them off again. No more combinations of terms of index 1 and index 2 are possible. So, draw a line below the last combination of these groups, i.e. 8, 9 (1), -- 001 in column 2. Now 1, 9 (8), i.e. – 001 and 8, 9 (1), i.e. 100– are the terms in the second group of column 2.

Similarly, comparing the terms of index 2 with the terms of index 3 in column 1,

$m_6(0110)$ and $m_7(0111)$ yield 6, 7 (1), i.e. 011–. Record it in column 2 and check off 6(0110) and 7(0111).

$m_6(0110)$ and $m_{14}(1110)$ yield 6, 14 (8), i.e. –110. Record it in column 2 and check off 6(0110) and 14(1110).

$m_9(1001)$ and $m_{13}(1101)$ yield 9, 13 (4), i.e. 1–01. Record it in column 2 and check off 9(1001) and 13(1101).

So, 6, 7 (1), i.e. 011–, and 6, 14 (8), i.e. –110 and 9, 13 (4), i.e. 1–01 are the terms in group 3 of column 2. Draw a line at the end of 9, 13 (4), i.e. 1–01.

Also, comparing the terms of index 3 with the terms of index 4 in column 1,

$m_7(0111)$ and $m_{15}(1111)$ yield 7, 15 (8), i.e. –111. Record it in column 2 and check off 7(0111) and 15(1111).

$m_{13}(1101)$ and $m_{15}(1111)$ yield 13, 15 (2), i.e. 11–1. Record it in column 2 and check off 13 and 15.

$m_{14}(1110)$ and $m_{15}(1111)$ yield 14, 15 (1), i.e. 111–. Record it in column 2 and check off 14 and 15.

So, 7, 15 (8), i.e. –111, and 13, 15 (2), i.e. 11–1 and 14, 15 (1), i.e. 111– are the terms in group 4 of column 2. Column 2 is completed now.

Comparing the terms of group 1 with the terms of group 2 in column 2, the terms 0, 1 (1), i.e. 000– and 8, 9 (1), i.e. 100– are combined to form 0, 1, 8, 9 (1, 8), i.e. –00–. Record it in group 1 of column 3 and check off 0, 1 (1), i.e. 000–, and 8, 9 (1), i.e. 100– of column 2. The terms 0, 8 (8), i.e. –000 and 1, 9 (8), i.e. –001 are combined to form 0, 1, 8, 9 (1, 8), i.e. –00–. This has already been recorded in column 3. So, no need to record again. Check off 0, 8 (8), i.e. –000 and 1, 9 (8), i.e. –001 of column 2. Draw a line below 0, 1, 8, 9 (1, 8), i.e. –00–. This is the only term in group 1 of column 3. No term of group 2 of column 2 can be combined with any term of group 3 of column 2. So, no entries are made in group 2 of column 2.

Comparing the terms of group 3 of column 2 with the terms of group 4 of column 2, the terms 6, 7 (1), i.e. 011–, and 14, 15 (1), i.e. 111– are combined to form 6, 7, 14, 15 (1, 8), i.e. –11–. Record it in group 3 of column 3 and check off 6, 7 (1), i.e. 011– and 14, 15 (1), i.e. 111– of column 2. The terms 6, 14 (8), i.e. –110 and 7, 15 (8), i.e. –111 are combined to form 6, 7, 14, 15 (1, 8), i.e. –11–. This has already been recorded in column 3; so, check off 6, 14 (8), i.e. –110 and 7, 15 (8), i.e. –111 of column 2.

Observe that the terms 9, 13 (4), i.e. 1–01 and 13, 15 (2), i.e. 11–1 cannot be combined with any other terms. Similarly in column 3, the terms 0, 1, 8, 9 (1, 8), i.e. –00– and 6, 7, 14, 15 (1, 8), i.e. –11– cannot also be combined with any other terms. So, these 4 terms are the prime implicants.

The terms, which cannot be combined further, are labelled as P, Q, R, and S. These form the set of prime implicants.

EX:

Obtain the minimal expression for $f = \Sigma\, m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$ using the tabular method.

Solution

The procedure to obtain the set of prime implicants is illustrated in Table 3.4.

Table 3.4 Example 3.30

Step 1		Step 2	Step 3	
Index 1	1 ✓	1, 3 (2) ✓	1, 3, 5, 7 (2, 4)	T
	2 ✓	1, 5 (4) ✓	1, 5, 9, 13 (4, 8)	S
	8 ✓	1, 9 (8) ✓	2, 3, 6, 7 (1, 4)	R
Index 2	3 ✓	2, 3 (1) ✓	8, 9, 12, 13 (1, 4)	Q
	5 ✓	2, 6 (4) ✓	5, 7, 13, 15 (2, 8)	P
	6 ✓	8, 9 (1) ✓		
	9 ✓	8, 12 (4) ✓		
	12 ✓	3, 7 (4) ✓		
Index 3	7 ✓	5, 7 (2) ✓		
	13 ✓	5, 13 (8) ✓		
Index 4	15 ✓	6, 7 (1) ✓		
		9, 13 (4) ✓		
		12, 13 (1) ✓		
		7, 15 (8) ✓		
		13, 15 (2) ✓		

The non-combinable terms P, Q, R, S and T are recorded as prime implicants.

$$P \rightarrow 5, 7, 13, 15\ (2, 8) = X\ 1\ X\ 1 = BD$$

(Literals with weights 2 and 8, i.e. C and A are deleted. The lowest minterm is $m_5(5 = 4 + 1)$. So, literals with weights 4 and 1, i.e. B and D are present in non-complemented form. So, read it as BD.)

$$Q \rightarrow 8, 9, 12, 13\ (1, 4) = 1\ X\ 0\ X = A\overline{C}$$

(Literals with weights 1 and 4, i.e. D and B are deleted. The lowest minterm is m_8. So, literal with weight 8 is present in non-complemented form and literal with weight 2 is present in complemented form. So, read it as $A\overline{C}$.)

$$R \rightarrow 2, 3, 6, 7\ (1, 4) = 0\ X\ 1\ X = \overline{A}C$$

(Literals with weights 1 and 4, i.e. D and B are deleted. The lowest minterm is m_2. So, literal with weight 2 is present in non-complemented form and literal with weight 8 is present in complemented form. So, read it as $\overline{A}C$.)

$$S \rightarrow 1, 5, 9, 13\ (4, 8) = X\ X\ 0\ 1 = \overline{C}D$$

(Literals with weights 4 and 8, i.e. B and A are deleted. The lowest minterm is m_1. So, literal with weight 1 is present in non-complemented form and literal with weight 2 is present in complemented form. So, read it as $\overline{C}D$.)

$$T \rightarrow 1, 3, 5, 7\ (2, 4) = 0\ X\ X\ 1 = \overline{A}D$$

(Literals with weights 2 and 4, i.e. C and B are deleted. The lowest minterm is 1. So, literal with weight 1 is present in non-complemented form and literal with weight 8 is present in complemented form. So, read it as $\overline{A}D$.)

The prime implicant chart of the expression

$$f = \Sigma\, m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$$

is as shown in Table 3.5. It consists of 11 columns corresponding to the number of minterms and 5 rows corresponding to the prime implicants P, Q, R, S, and T generated. Row R contains four ×s at the intersections with columns 2, 3, 6, and 7, because these minterms are covered by the prime implicant R. A row is said to cover the columns in which it has ×s. The problem now is to select a minimal subset of prime implicants, such that each column contains at least one × in the rows corresponding to the selected subset and the total number of literals in the prime implicants selected is as small as possible. These requirements guarantee that the number of unions of the selected prime implicants is equal to the original number of minterms and that, no other expression containing fewer literals can be found.

Table 3.5 Example 3.30: Prime implicant chart

		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	1	2	3	5	6	7	8	9	12	13	15
*P → 5, 7, 13, 15 (2, 8)				×		×				×	×
*Q → 8, 9, 12, 13 (1, 4)							×	×	×	×	
*R → 2, 3, 6, 7 (1, 4)		×	×		×	×					
S → 1, 5, 9, 13 (4, 8)	×			×				×		×	
T → 1, 3, 5, 7 (2, 4)	×		×	×		×					

In the prime implicant chart of Table 3.5, m_2 and m_6 are covered by R only. So, R is an essential prime implicant. So, check off all the minterms covered by it, i.e. m_2, m_3, m_6, and m_7. Q is also an essential prime implicant because only Q covers m_8 and m_{12}. Check off all the minterms covered by it, i.e. m_8, m_9, m_{12}, and m_{13}. P is also an essential prime implicant, because m_{15} is covered only by P. So check off m_{15}, m_5, m_7, and m_{13} covered by it. Thus, only minterm 1 is not covered. Either row S or row T can cover it and both have the same number of literals. Thus, two minimal expressions are possible.

$$P + Q + R + S = BD + A\bar{C} + \bar{A}C + \bar{C}D$$

or

$$P + Q + R + T = BD + A\bar{C} + \bar{A}C + \bar{A}D$$

Combinational Logic Design

Logic circuits for digital systems may be combinational or sequential. The output of a combinational circuit depends on its present inputs only .Combinational circuit processing operation fully specified logically by a set of Boolean functions .A combinational circuit consists of input variables, logic gates and output variables.Both input and output data are represented by signals, i.e., they exists in two possible values. One is logic –1 and the other logic 0.

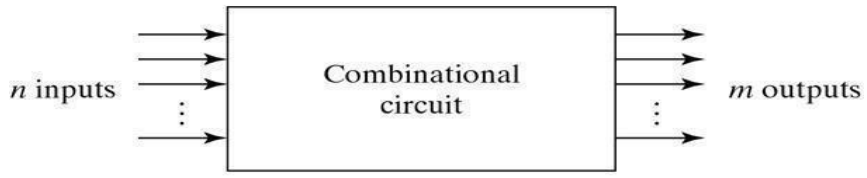


Fig. Block Diagram of Combinational Circuit

For n input variables,there are 2^n possible combinations of binary input variables .For each possible input Combination ,there is one and only one possible output combination.A combinational circuit can be described by m Boolean functions one for each output variables.Usually the input s comes from flip-flops and outputs goto flip-flops.

Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

Adders:

Digital computers perform variety of information processing tasks,the one is arithmetic operations.And the most basic arithmetic operation is the addition of two binary digits.i.e, 4 basic possible operations are:

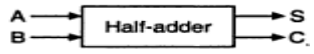
0+0=0,0+1=1,1+0=1,1+1=10

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1,the binary sum consists of two digits.The higher significant bit of this result is called a carry.A combinational circuit that performs the addition of two bits is called a half-adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder.& 2 half adder can employ as a full-adder.

The Half Adder: A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

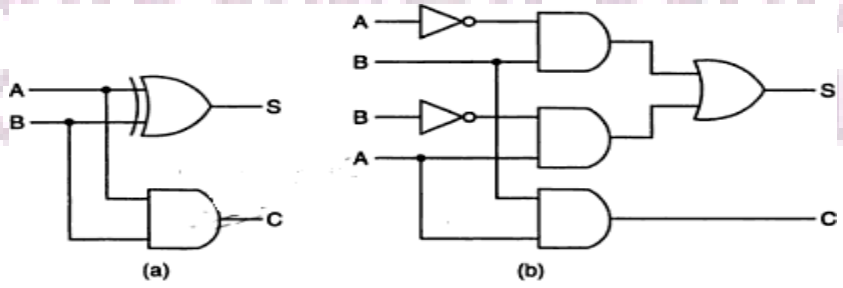
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

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

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1).Therefore,

C=AB

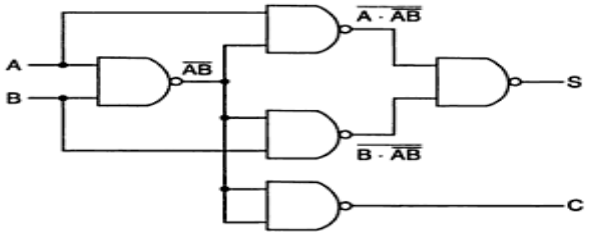
A half-adder can be realized by using one X-OR gate and one AND gate a



Logic diagrams of half-adder

NAND LOGIC:

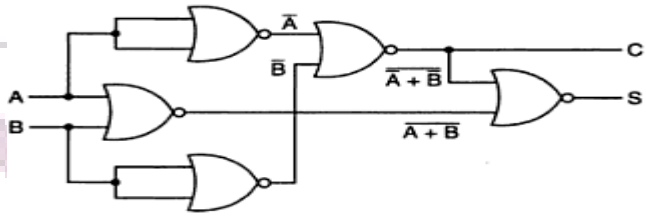
$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= A \cdot \overline{AB} + B \cdot \overline{AB}$$
$$= \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$
$$C = AB = \overline{\overline{AB}}$$



Logic diagram of a half-adder using only 2-input NAND gates.

NOR Logic:

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



Logic diagram of a half-adder using only 2-input NOR gates.

The Full Adder:

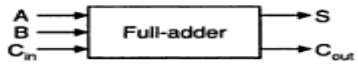
A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry.The

eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s , the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-adder.

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A,B and C_{in} is described by

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

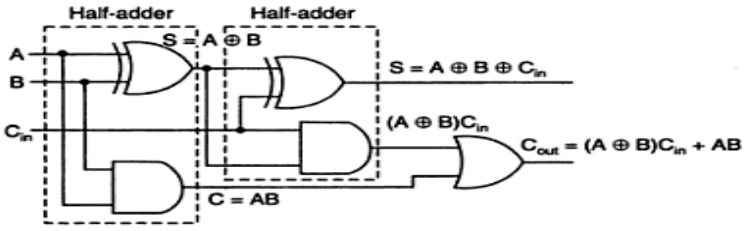
$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$

and

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

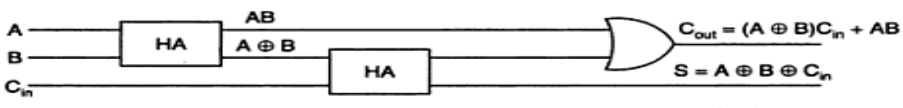
$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A,B, and C_{in}, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic.

The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as

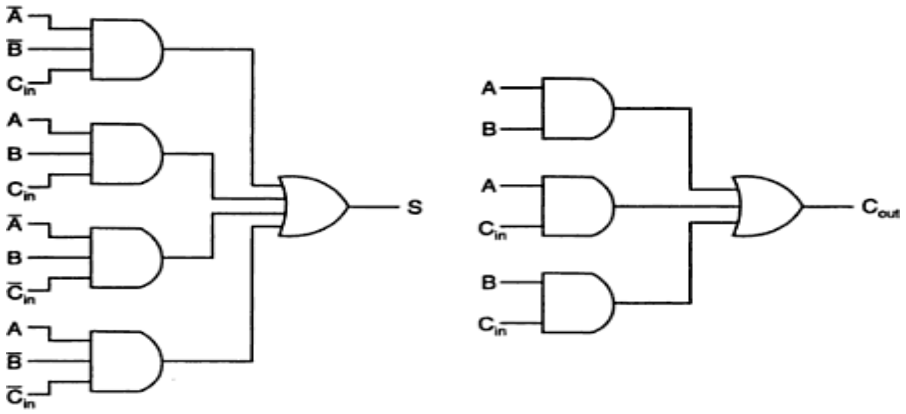
$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

Then

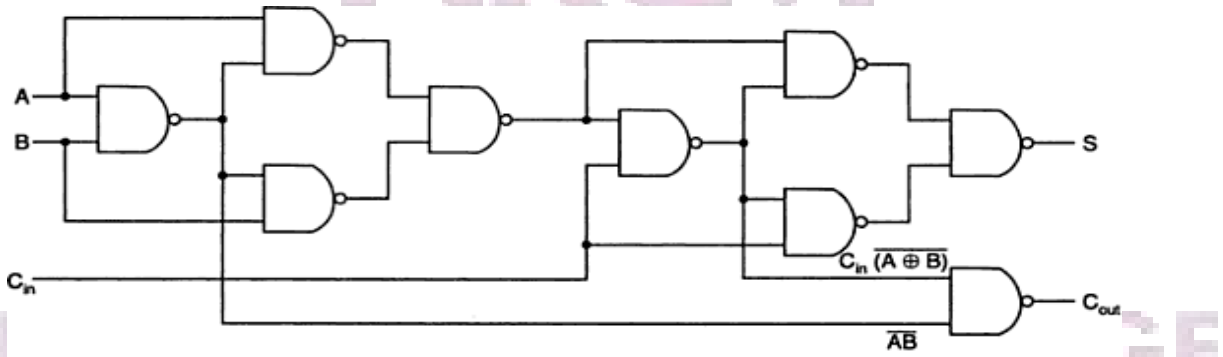
$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)C_{in}}}}$$

NAND Logic:

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$



Sum and carry bits of a full-adder using AOI logic.



Logic diagram of a full-adder using only 2-input NAND gates.

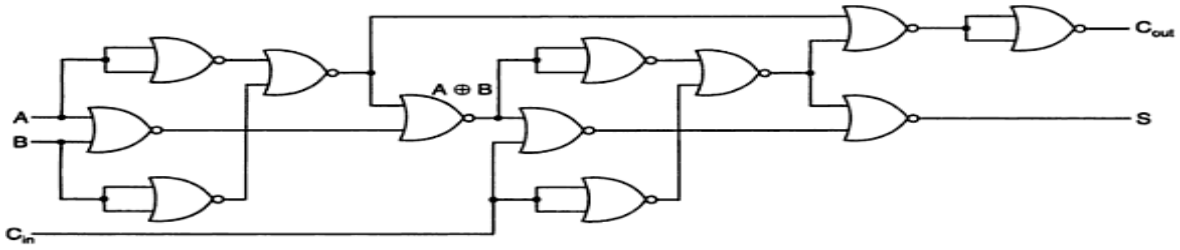
NOR Logic:

$$A \oplus B = \overline{\overline{(A + B)} + \overline{\overline{A} + \overline{B}}}$$

Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) + C_{in}} + \overline{(A \oplus B) + \overline{C_{in}}}}$$

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{A} + \overline{B}} + \overline{\overline{C_{in}} + \overline{A \oplus B}}$$



Logic diagram of a full-adder using only 2-input NOR gates.

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

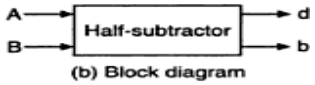
The Half-Subtractor:

A Half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. . It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

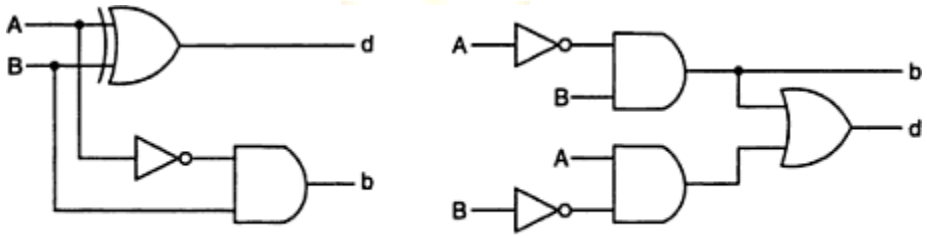
Half-subtractor.

The output borrow b is a 0 as long as A≥B. It is a 1 for A=0 and B=1. The d output is the result of the arithmetic operation 2b+A-B.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is , therefore ,

$$d=AB+A\ B=A\oplus B \text{ and } b=A\ B$$

That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.Note that logic for this exactly the same as the logic for output S in the half-adder.



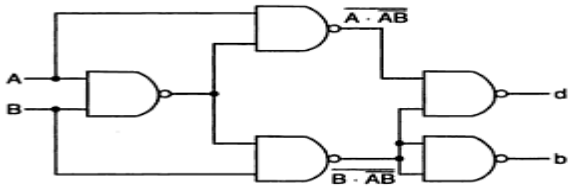
Logic diagrams of a half-subtractor.

A half-substractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:

$$d = A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

$$b = \overline{A}B = B(\overline{A} + \overline{B}) = B(\overline{AB}) = \overline{\overline{B \cdot \overline{AB}}}$$



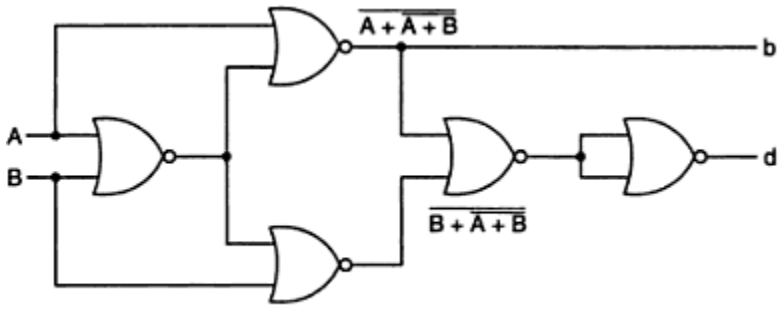
Logic diagram of a half-subtractor using only 2-input NAND gates.

NOR Logic:

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

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

$$d = \overline{A}B = \overline{A}(A + B) = \overline{\overline{A}(A + B)} = \overline{A + (\overline{A + B})}$$



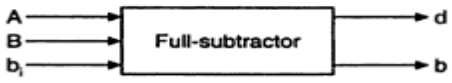
Logic diagram of a half-subtractor using only 2-input NOR gates.

The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A) , when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of A-B-b_i.

Inputs			Difference	Borrow
A	B	b_i	d	b
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

(a) Truth table



(b) Block diagram

Full-subtractor.

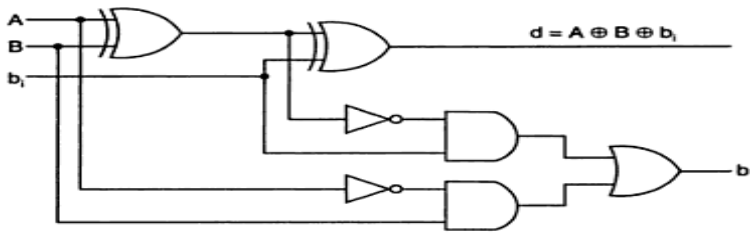
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possiblecombinations of A,B and b_i is

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B)$$
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i$$

and

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$
$$= \bar{A}B + (\overline{A \oplus B})b_i$$

A full-subtractor can be realized using X-OR gates and AOI gates as



Logic diagram of a full-subtractor.

The full subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

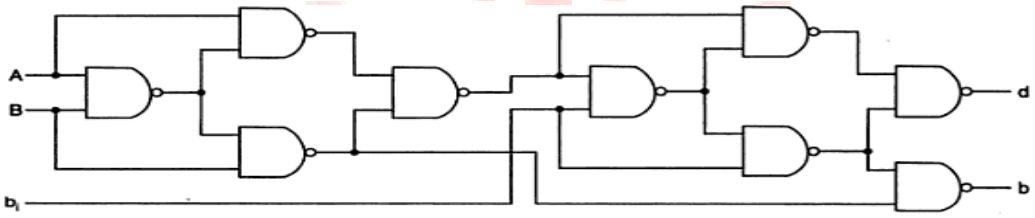
NAND Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}} = \overline{\overline{(A \oplus B)\overline{(A \oplus B)b_i}} \cdot \overline{b_i\overline{(A \oplus B)b_i}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\bar{A}B + b_i\overline{(A \oplus B)}}$$

$$= \overline{\overline{\bar{A}B} \cdot \overline{b_i\overline{(A \oplus B)}}} = \overline{B(\bar{A} + \bar{B}) \cdot b_i(\bar{b}_i + (A \oplus B))}$$

$$= \overline{\overline{B \cdot \overline{AB}} \cdot \overline{b_i[\bar{b}_i \cdot (A \oplus B)]}}$$



Logic diagram of a full-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}}$$

$$= \overline{\overline{(A \oplus B)b_i} + \overline{(A \oplus B)}\bar{b}_i}$$

$$= \overline{\overline{[(A \oplus B) + \overline{(A \oplus B)}\bar{b}_i]}[b_i + \overline{(A \oplus B)}\bar{b}_i]}$$

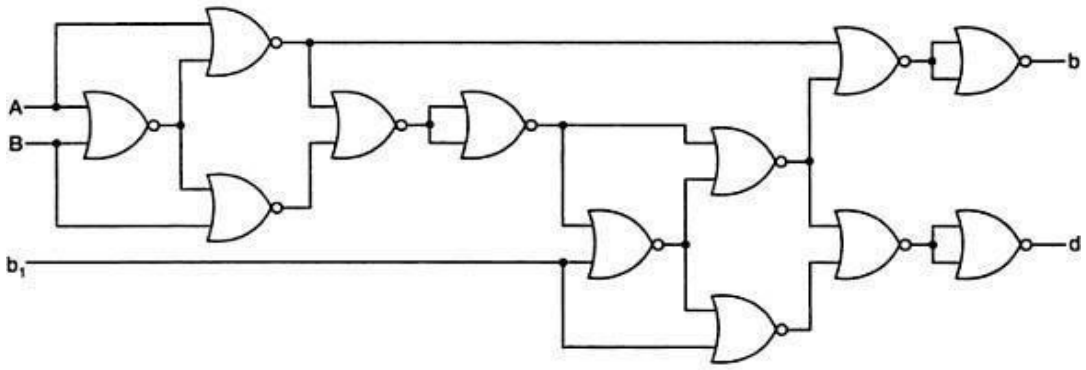
$$= \overline{\overline{(A \oplus B) + \overline{(A \oplus B)} + b_i} + \overline{b_i + \overline{(A \oplus B)} + b_i}}$$

$$= \overline{\overline{(A \oplus B) + \overline{(A \oplus B)} + b_i} + \overline{b_i + \overline{(A \oplus B)} + b_i}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B})$$

$$= \bar{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + b_i]$$

$$= \overline{\overline{A + (\overline{A + B})} + \overline{(A \oplus B) + (\overline{A \oplus B}) + b_i}}$$

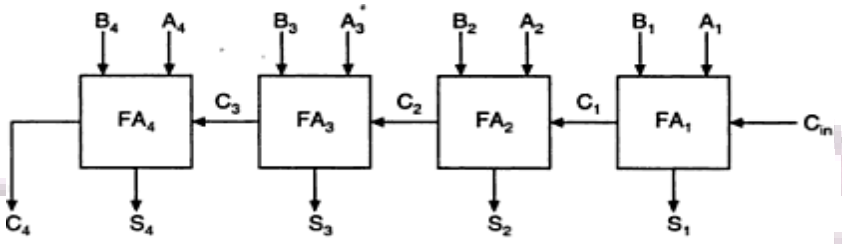


Logic diagram of a full subtractor using only 2-input NOR gates.

Binary Parallel Adder:

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain , with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augends bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower –order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} and the output carry is C_4. The S output generates the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. AN n-bit parallel adder requires n-full adders. It can be constructed from 4-bit, 2-bit and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher –order bits. The 4-bit full adder is a typical example of an MSI function.



Logic diagram of a 4-bit binary parallel adder.

Ripple carry adder:

In the parallel adder, the carry –out of each stage is connected to the carry-in of the next stage. The sum and carry-out bits of any stage cannot be produced, until sometime after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry,

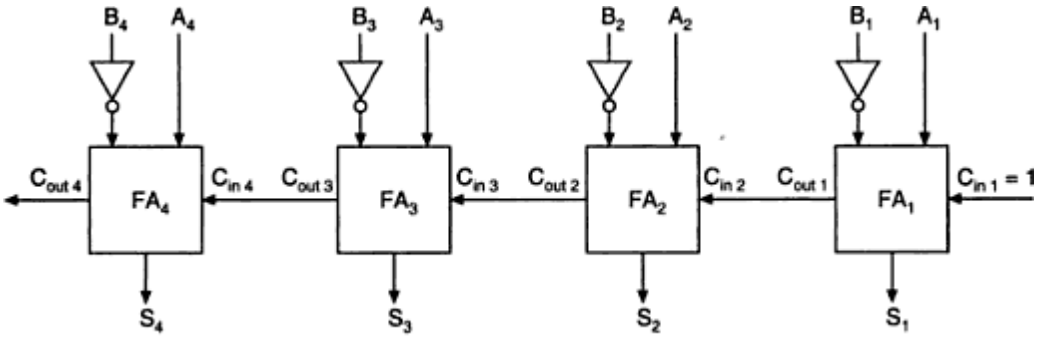
which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

The 4-bit parallel adder, the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1. Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2) , and so on. At each stage ,the sum bit is not valid until after the carry bits in all the preceding stages are valid. Carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a ripple carry adder.. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

4- Bit Parallel Subtractor:

The subtraction of binary numbers can be carried out most conveniently by means of complements , the subtraction A-B can be done by taking the 2‘s complement of B and adding it to A . The 2‘s complement can be obtained by taking the 1‘s complement and adding 1 to the least significant pair of bits. The 1‘s complement can be implemented with inverters as

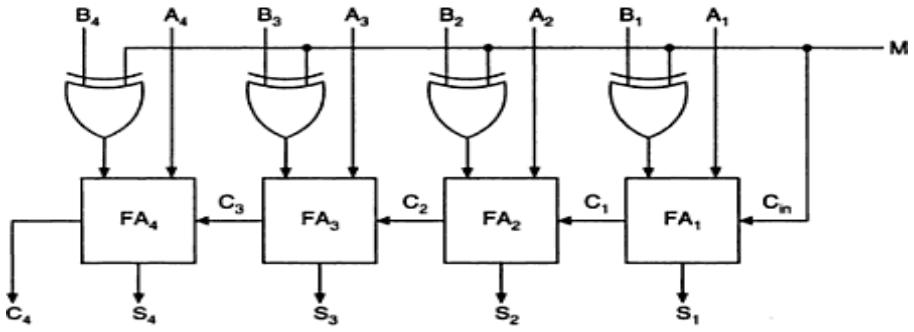


Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit adder-subtractor, the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor. Each X-OR gate receives input M and one of the inputs of B. When M=0, $B \oplus 0 = B$ .The full-adder receives the value of B , the input carry is 0

and the circuit performs A+B. when $B \oplus 1 = B'$ and C_1=1. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2's complement of B.



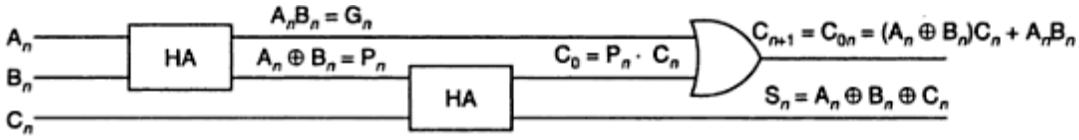
Logic diagram of a 4-bit binary adder-subtractor.

The Look-Ahead –Carry Adder:

In parallel-adder,the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Consider one full adder stage; say the nth stage of a parallel adder as shown in fig. we know that is made by two half adders and that the half adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called generated carry, expressed as $G_n = A_n.B_n$ which has to appear at the output through the OR gate as shown in fig.



A full adder (*n*th stage of a parallel adder).

Thereis another possibility of producing a carry out. X-OR gate inside the half-adder at the input produces an intermediary sum bit- call it P_n –which is expressed as $P_n = A_n \oplus B_n$. Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final

sum bit and $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$ and output carry$C_0 = P_n.C_n = (A_n \oplus B_n)C_n$ which becomes carry for the (n+1) th stage.

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the and gate in the second half-adder will inhibit C_n . the carry out of the nth stage is 1 when either $G_n=1$ or $P_n.C_n = 1$ or both G_n and $P_n.C_n$ are equal to 1.

For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$
$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n+1)st stage .it is possible to express the output carry of a higher significant stage is the carry-out of the previous stage.

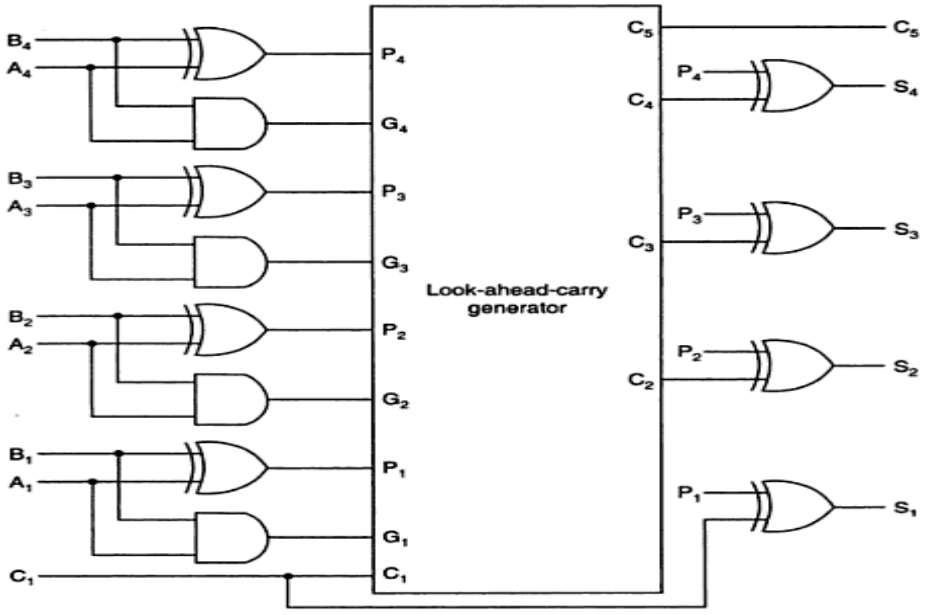
Based on these , the expression for the carry-outs of various full adders are as follows,

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for n stages designated as 0 through $(n-1)$ would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \ldots + P_{n-1} \cdot \ldots P_0 \cdot C_0$$

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).



Logic diagram of a 4-bit look-ahead-carry adder.

2's complement Addition and Subtraction using Parallel Adders:

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction operations of signed numbers can be performed using only the addition operation ,if we use the 2's complement form to represent negative numbers.

The circuit shown can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/SUB'. When the ADD/SUB' level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/Sub' level is LOW, the circuit subtract the number in register B from the number in register A. The operation is:

When ADD/SUB' is a 1:

1. AND gates 1,3,5 and 7 are enabled , allowing B_0,B_1,$B_2$and B_3 to pass to the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are disabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates 9,10,11 and 12.
2. The two levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3. The sum appears at the output S_0 to S_3
3. Add/SUB' =1 causes no carry into the adder.

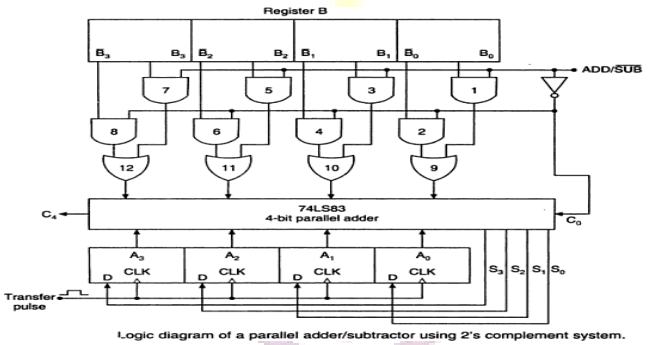
When ADD/SUB' is a 0:

1. AND gates 1,3,5 and 7 are disabled , allowing B_0,B_1,$B_2$and B_3 from reaching the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are enabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates.

2. The two levels B_0‘ to B_3‘ pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3.The C_0 is now 1.thus the number in register B is converted to its 2‘s complement form.

3. The difference appears at the output S_0 to S_3.

Adders/Subtractors used for adding and subtracting signed binary numbers. In computers , the output is transferred into the register A (accumulator) so that the result of the addition or subtraction always end up stored in the register A This is accomplished by applying a transfer pulse to the CLK inputs of register A.



Logic diagram of a parallel adder/subtractor using 2's complement system.

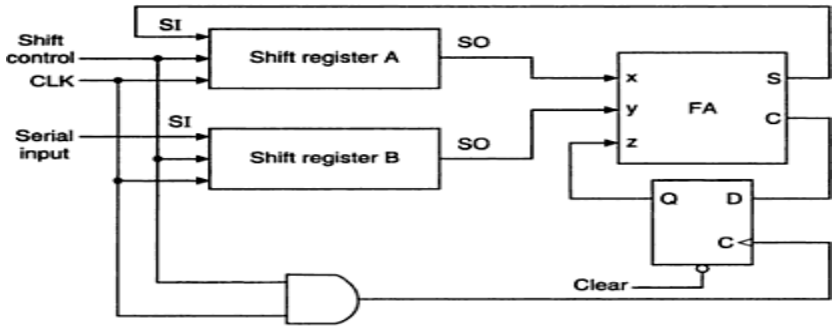
Serial Adder:

A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full-adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is:

Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at x and y. The shift control enables both registers and carry flip-flop , so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A , and the output carry is transferred into flip-flop Q . The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.

Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.



Logic diagram of a serial adder.

Difference between Serial and Parallel Adders:

The parallel adder registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

BCD Adder:

The BCD addition process:

1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:

1. Add two 4-bit BCD code groups, using straight binaryaddition.

2. Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is , add 0110 (decimal 6) to this sum and generate a carry to the next decimalposition.

The first requirement is easily met by using a 4- bit binary parallel adder such as the 74LS83 IC .For example , if the two BCD code groups $A_3A_2A_1A_0$and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4 , the carry –out of the MSB bits.

The sum outputs $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 100109when both the BCD code groups are 1001=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases , where the sum is greater than 1001 are listed as:

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e, for the cases in table). If examine these cases ,see that X will be HIGH for either of the following conditions:

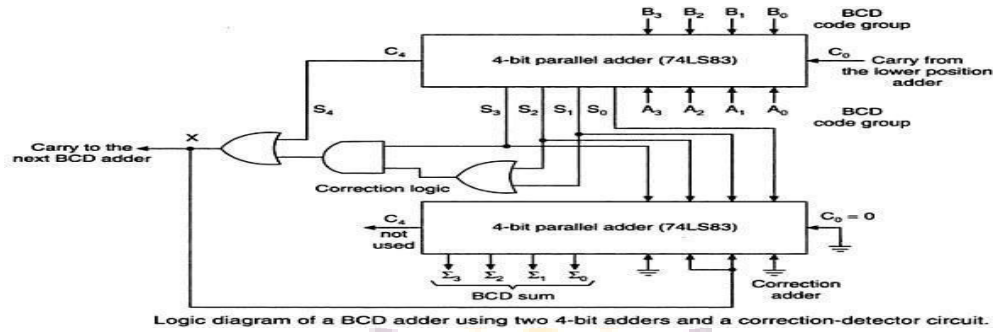
1. Whenever S_4 =1(sum greater than 15)
2. Whenever S_3 =1 and either S_2 or S_1 or both are 1 (sum 10 to 15)

This condition can be expressed as

$$X=S_4+S_3(S_2+S_1)$$

Whenever X=1, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are added together in the upper 4-bit adder, to produce the sum $S_4S_3S_2S_1S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when X=1, producing the final BCD sum output represented by $\sum_3\sum_2\sum_1\sum_0$. The X is also the carry-out that is produced when the sum is greater than 01001. When X=0, there is no carry and no addition of 0110. In such cases, $\sum_3\sum_2\sum_1\sum_0= S_3S_2S_1S_0$.

Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.



Logic diagram of a BCD adder using two 4-bit adders and a correction-detector circuit.

EXCESS-3(XS-3) ADDER:

To perform Excess-3 additions,

1. Add two xs-3 code groups
2. If carry=1, add 0011(3) to the sum of those two code groups
   If carry =0, subtract 0011(3) i.e., add 1101 (13 in decimal) to the sum of those two code groups.
   Ex: Add 9 and 5

	1100	9 in Xs-3
	+1000	5 in xs-3

1	0100	there is a carry
+0011	0011	add 3 to each group
---	---	
0100	0111	14 in xs-3
(1)	(4)	

EX:

(b)	0 1 1 1	4 in XS-3
	+ 0 1 1 0	3 in XS-3
	1 1 0 1	no carry
	+ 1 1 0 1	Subtract 3 (i.e. add 13)
Ignore carry	1 1 0 1 0	7 in XS-3
	(7)	

Implementation of xs-3 adder using 4-bit binary adders is shown. The augend (A_3 $A_2A_1A_0$) and addend ($B_3B_2B_1B_0$) in xs-3 are added using the 4-bit parallel adder. If the carry is a 1, then 0011(3) is added to the sum bits $S_3S_2S_1S_0$ of the upper adder in the lower 4-bit parallel

adder. If the carry is a 0, then 1101(3) is added to the sum bits (This is equivalent to subtracting 0011(3) from the sum bits. The correct sum in xs-3 is obtained

Excess-3 (XS-3) Subtractor:

To perform Excess-3 subtraction,

1. Complement the subtrahend
2. Add the complemented subtrahend to the minuend.
3. If carry =1, result is positive. Add 3 and end around carry to the result . If carry=0, the result is negative. Subtract 3, i.e, and take the 1‘s complement of the result.

Ex: Perform 9-4

	1100	9 in xs-3
	+1000	Complement of 4 n Xs-3
	---	
(1)	0100	There is a carry
	+0011	Add 0011(3)

	0111	
	1	End around carry

	1000	5 in xs-3

The minuend and the 1‘s complement of the subtrahend in xs-3 are added in the upper 4-bit parallel adder. If the carry-out from the upper adder is a 0, then 1101 is added to the sum bits of the upper adder in the lower adder and the sum bits of the lower adder are complemented to get the result. If the carry-out from the upper adder is a 1, then 3=0011 is added to the sum bits of the lower adder and the sum bits of the lower adder give the result.

Binary Multipliers:

In binary multiplication by the paper and pencil method, is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time.
In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0,0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this process is

Multiplicand 1110		
Multiplier	1001	
	1110	The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (1 1 1 0 0)
	1110	The second multiplier bit is a 0; write down the previous result 1110; shift the multiplicand to the left again (1 1 1 0 0 0)

+1110000 The fourth multiplier bit is a 1 write down the new multiplicand add it to the first partial product to obtain the final product.

1111110

This multiplication process can be performed by the serial multiplier circuit , which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of following elements

X register: A 4-bit shift register that stores the multiplier --- it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

B register: An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

A register: An 8-bit register, i.e, the accumulator that accumulates the partial products.

Adder: An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs S_7 through S_0 are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

1. **Before the first clock pulse***:* Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. Assume that each of these registers is loaded using its asynchronous inputs(i.e., PRESET and CLEAR). The output of the adder will be the sum of A and B,i.e., 00001110.
2. **First Clock pulse***:*Since the LSB of the multiplier (X_0) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This produces a new sum of A and B.
3. **Second Clock Pulse:** The second bit of the original multiplier is now in X_0 . Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.
4. **Third Clock Pulse**:The third bit of the original multiplier is now in X_0;since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.
5. **Fourth Clock Pulse:** The last bit of the original multiplier is now in X_0 , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.

Code converters:

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus a

code converter is a logic circuit whose inputs are bit patterns representing numbers (or character) in one cod and whose outputs are the corresponding representation in a different code. Code converters are usually multiple output circuits.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

For example, a binary –to-gray code converter has four binary input lines B_4, B_3,B_2,B_1 and four gray code output lines G_4,G_3,G_2,G_1. When the input is 0010, for instance, the output should be 0011 and so forth. To design a code converter, we use a code table treating it as a truth table to express each output as a Boolean algebraic function of all the inputs.

In this example, of binary –to-gray code conversion, we can treat the binary to the gray code table as four truth tables to derive expressions for G_4, G3, G2, and G1. Each of these four expressions would, in general, contain all the four input variables B_4, B3,B_2,and B1. Thus,this code converter is actually equivalent to four logic circuits, one for each of the truth tables.

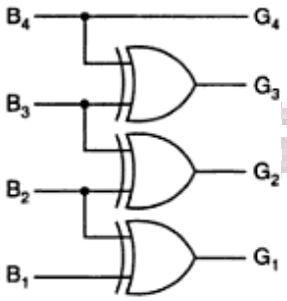
The logic expression derived for the code converter can be simplified using the usual techniques, including _don‘t cares‘ if present. Even if the input is an unweighted code, the same cell numbering method which we used earlier can be used, but the cell numbers --must correspond to the input combinations as if they were an 8-4-2-1 weighted code. s

Design of a 4-bit binary to gray code converter:

$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15) \qquad G_4 = B_4$$
$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11) \qquad G_3 = \bar{B}_4 B_3 + B_4 \bar{B}_3 = B_4 \oplus B_3$$
$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13) \qquad G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2 = B_3 \oplus B_2$$
$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14) \qquad G_1 = \bar{B}_2 B_1 + B_2 \bar{B}_1 = B_2 \oplus B_1$$

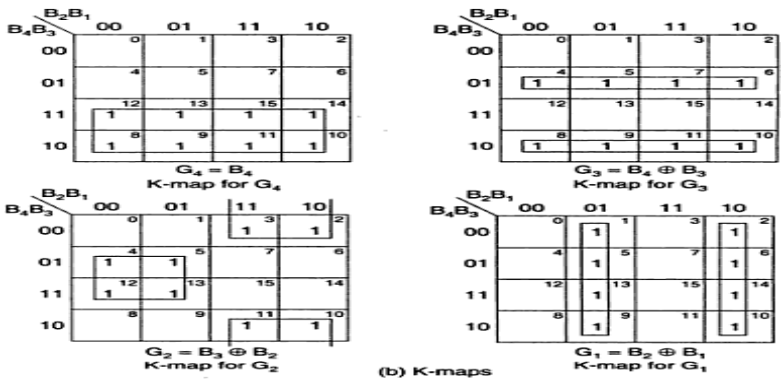
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

(a) Conversion table



(c) Logic diagram

4-bit binary-to-Gray code converter



(b) K-maps

4-bit binary-to-Gray code converter.

Design of a 4-bit gray to Binary code converter:

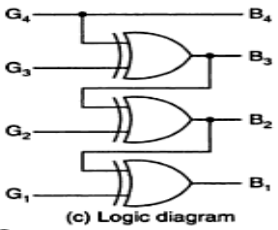
$$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$$
$$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$$

$$B_4 = G_4$$
$$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$$
$$B_2 = \bar{G}_4 G_3 \bar{G}_2 + \bar{G}_4 \bar{G}_3 G_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$$
$$= \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = \bar{G}_4 \bar{G}_3 \bar{G}_2 G_1 + \bar{G}_4 \bar{G}_3 G_2 \bar{G}_1 + \bar{G}_4 G_3 G_2 G_1 + \bar{G}_4 G_3 \bar{G}_2 \bar{G}_1 + G_4 G_3 \bar{G}_2 G_1$$
$$+ G_4 G_3 G_2 \bar{G}_1 + G_4 \bar{G}_3 G_2 G_1 + G_4 \bar{G}_3 \bar{G}_2 \bar{G}_1$$

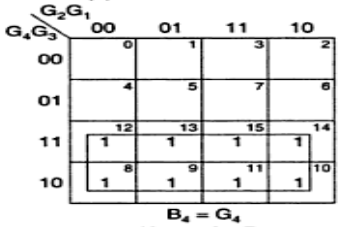
$$= \bar{G}_4 \bar{G}_3 (G_2 \oplus G_1) + G_4 G_3 (G_2 \oplus G_1) + \bar{G}_4 G_3 (\overline{G_2 \oplus G_1}) + G_4 \bar{G}_3 (\overline{G_2 \oplus G_1})$$
$$= (G_2 \oplus G_1)(\overline{G_4 \oplus G_3}) + (\overline{G_2 \oplus G_1})(G_4 \oplus G_3)$$
$$= G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

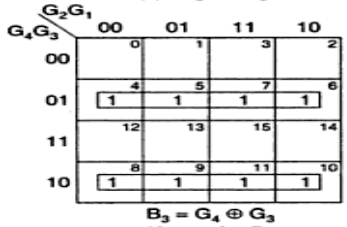
4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

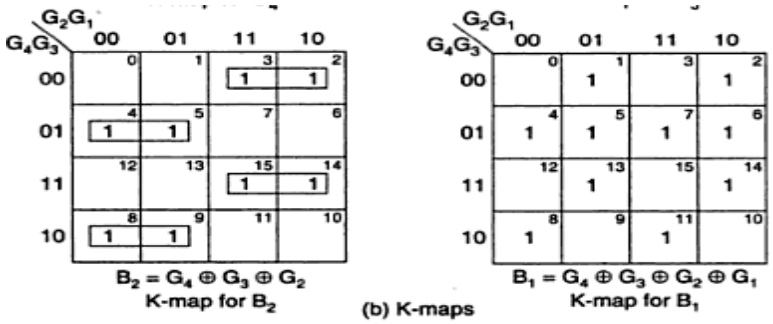
(a) Conversion table



(c) Logic diagram







(b) K-maps

4-bit Gray-to-binary code converter.

Design of a 4-bit BCD to XS-3 code converter:

8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

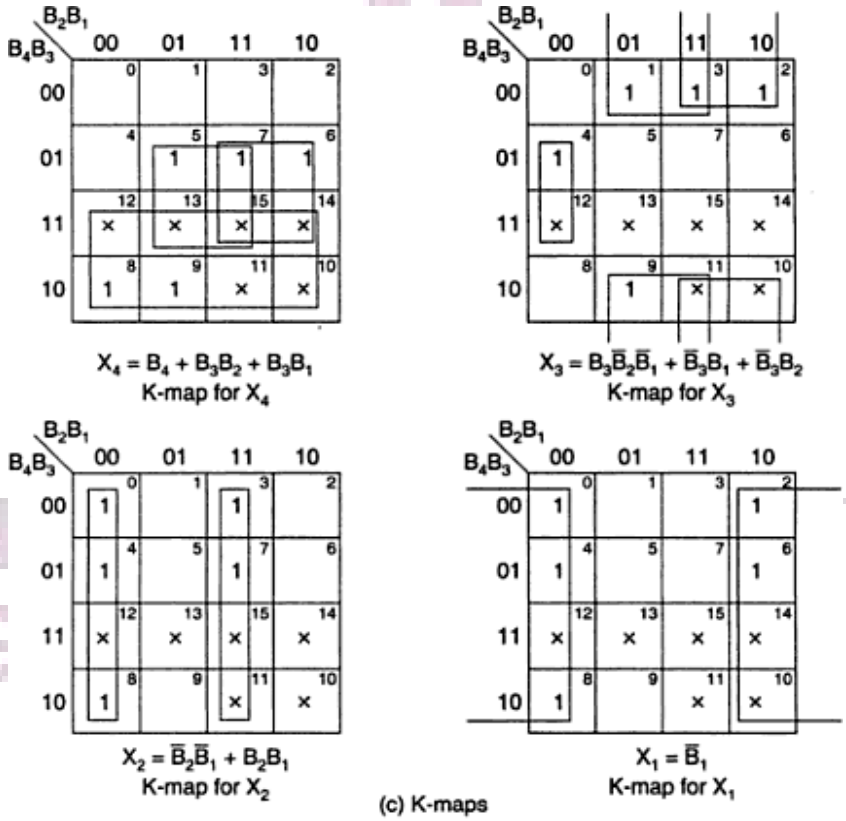
(a) Conversion table

$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$
$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$
$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$
$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$

The minimal expressions are
$X_4 = B_4 + B_3B_2 + B_3B_1$
$X_3 = B_3\overline{B}_2\overline{B}_1 + \overline{B}_3B_1 + \overline{B}_3B_2$
$X_2 = \overline{B}_2\overline{B}_1 + B_2B_1$
$X_1 = \overline{B}_1$

(b) Minimal expressions

4-bit BCD-to-XS-3 code converter



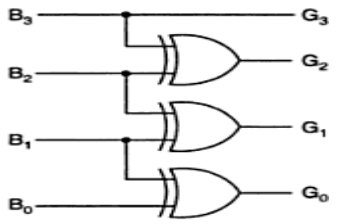
(c) K-maps

4-bit BCD-to-XS-3 code converter.

Design of a BCD to gray code converter:

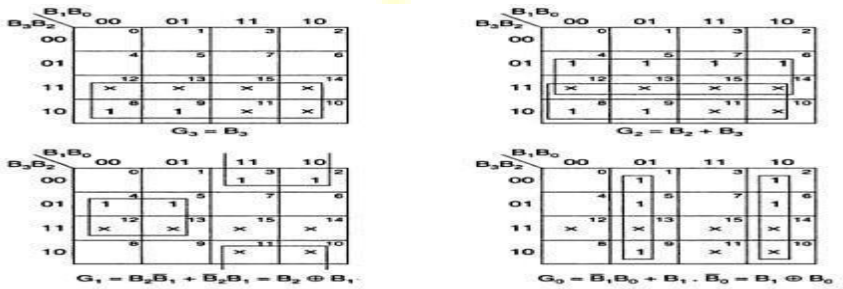
BCD code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

(a) BCD-to-Gray code conversion table



(b) Logic diagram

BCD-to-Gray code converter.

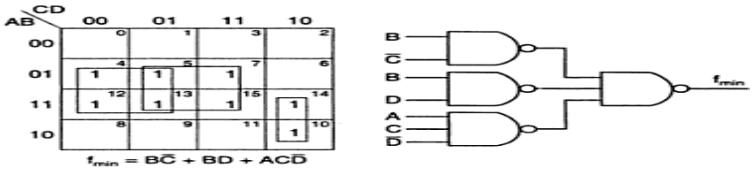


K-maps for a BCD-to-Gray code converter.

Design of a SOP circuit to Detect the Decimal numbers 5 through 12 in a 4-bit gray code Input:

Decimal number	4-bit Gray code A	B	C	D	Output f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



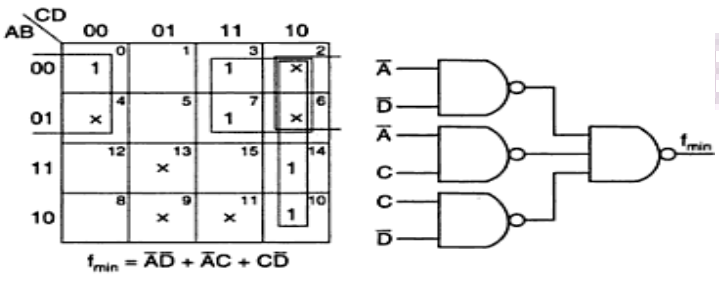
(b) K-map (c) NAND logic

Truth table, K-map and logic diagram for the SOP circuit.

Design of a SOP circuit to detect the decimal numbers 0,2,4,6,8 in a 4-bit 5211 BCD code input:

Decimal number	5211 code A	B	C	D	Output f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0

(a) Truth table



(b) K-map (c) Logic diagram

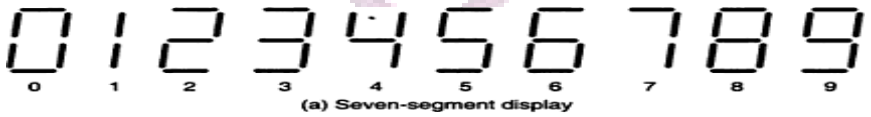
Truth table, K-map and logic diagram for the SOP circuit.

Design of a Combinational circuit to produce the 2's complement of a 4-bit binary number:

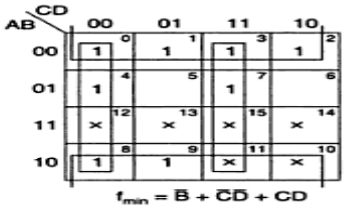
Input				Output			
A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

(a) Conversion table

Conversion table and K-maps for the circuit

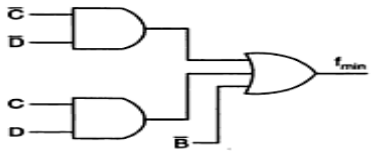


(a) Seven-segment display



$f_{min} = \bar{B} + \bar{C}\bar{D} + CD$

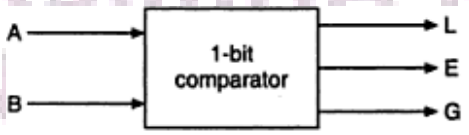
(b) K-map



(c) Logic diagram

Comparators:

$$EQUALITY = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



Block diagram of a 1-bit comparator.

1. Magnitude Comparator:

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\overline{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

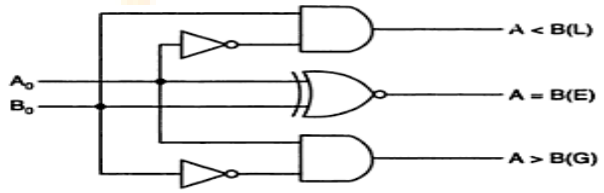
$$A < B: L = \overline{A}_0 B_0$$

If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

1-bit comparator.

1- bit Magnitude Comparator:

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

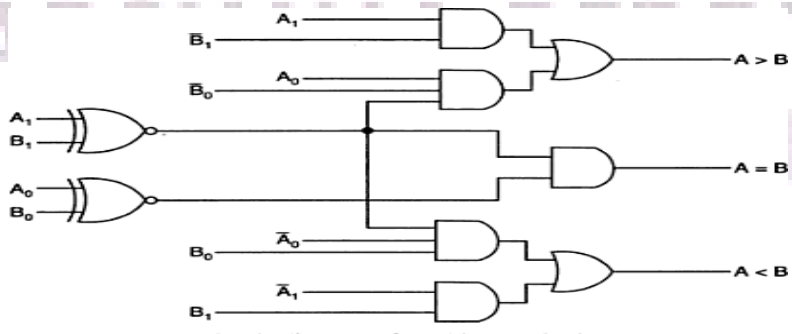
$$A > B : G = A_1\overline{B}_1 + (A_1 \odot B_1)A_0\overline{B}_0$$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$. So the expression for $A < B$ is

$$A < B : L = \overline{A}_1 B_1 + (A_1 \odot B_1)\overline{A}_0 B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$



Logic diagram of a 2-bit magnitude comparator.

4-Bit Magnitude Comparator:

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) = A_3\overline{B}_3 + (A_3 \odot B_3)A_2\overline{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\overline{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\overline{B}_0$$

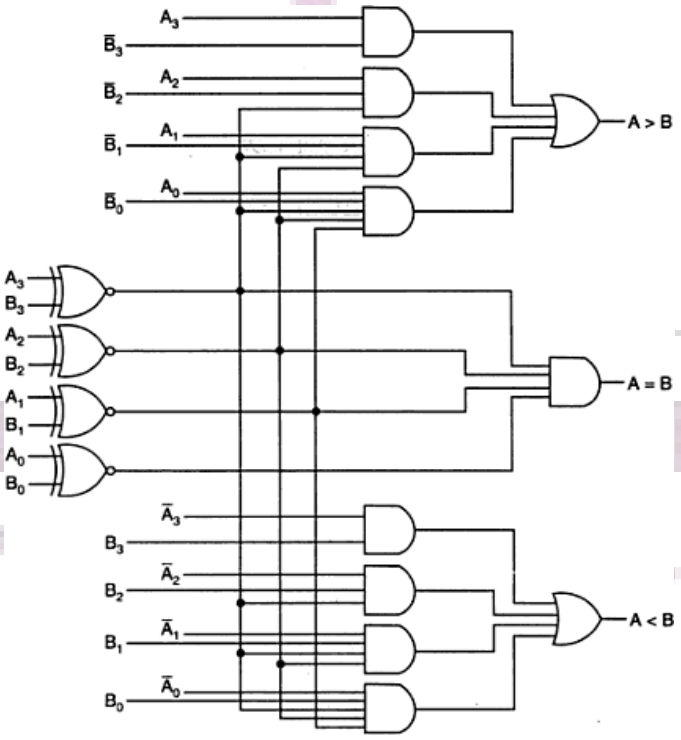
Similarly, the logic expression for $A < B$ can be written as

$$A < B = \overline{A}_3B_3 + (A_3 \odot B_3)\overline{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A}_0B_0$$

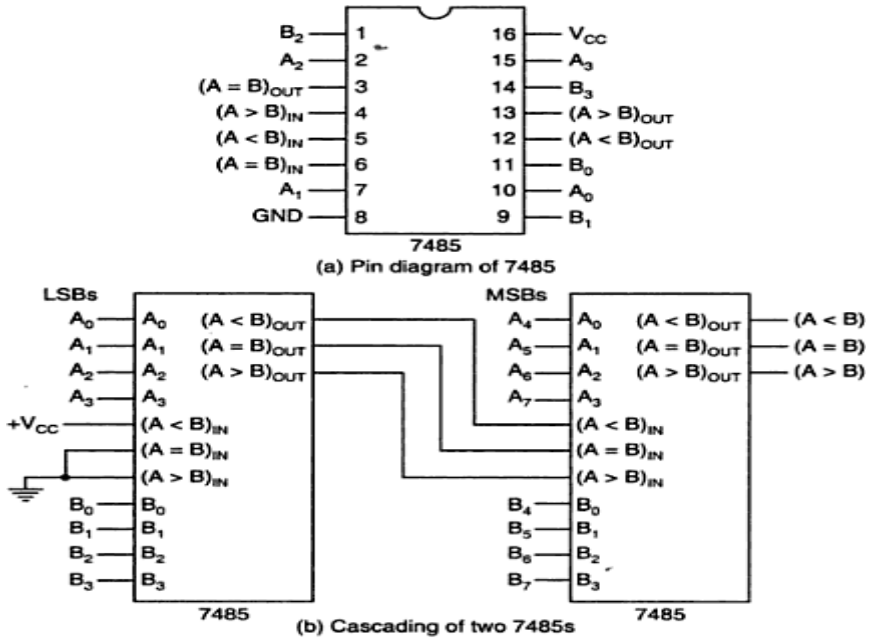
If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then $A = B$.

So the expression for $A = B$ can be written as

$$(A = B) = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



IC Comparator:

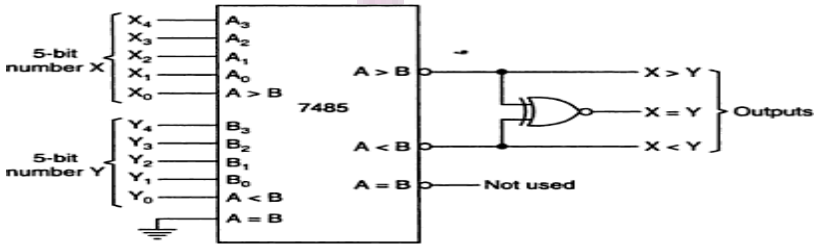


(a) Pin diagram of 7485

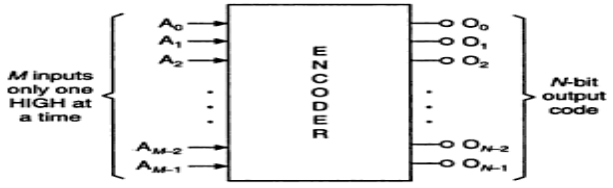
(b) Cascading of two 7485s

Pin diagram and cascading of 7485 4-bit comparators.

ENCODERS:



Use of 7485 as a 5-bit comparator.

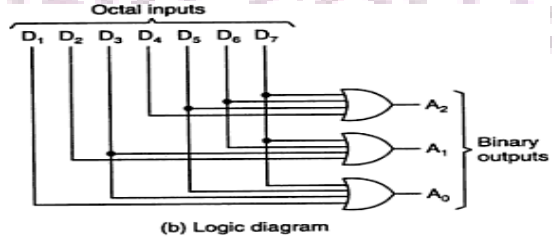


Block diagram of encoder.

Octal to Binary Encoder:

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

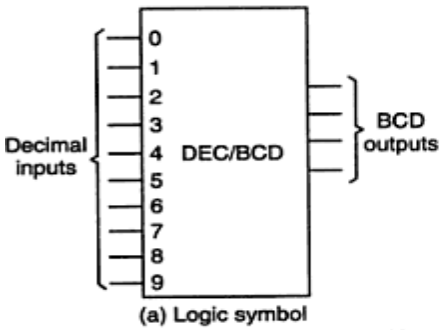
(a) Truth table



(b) Logic diagram

Octal-to-binary encoder.

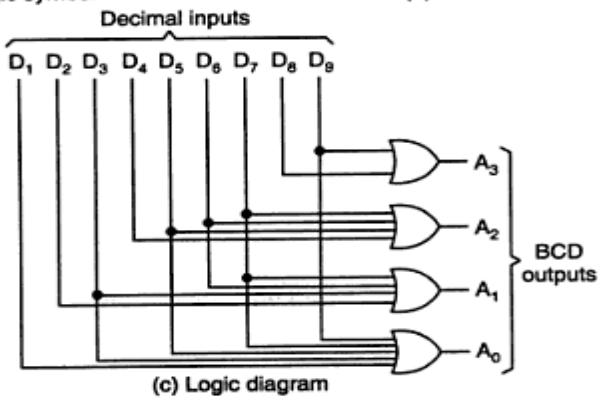
Decimal to BCD Encoder:



(a) Logic symbol

Decimal inputs		Binary			
		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

(b) Truth table



(c) Logic diagram

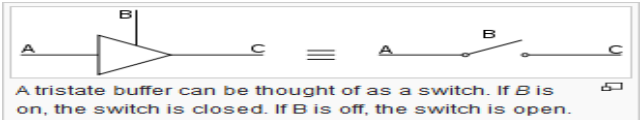
Decimal-to-BCD encoder.

Tristate bus system:

In digital electronics**three-state**, **tri-state**, or **3-state**logic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit.

This allows multiple circuits to share the same output line or lines (such as a bus which cannot listen to more than one device at a time).

Three-state outputs are implemented in many registers, bus drivers, and flip-flops in the 7400 and 4000 series as well as in other types, but also internally in many integrated circuits. Other typical uses are internal and external buses in microprocessors, computer memory, and peripherals. Many devices are controlled by an active-low input called OE (Output Enable) which dictates whether the outputs should be held in a high-impedance state or drive their respective loads (to either 0- or 1-level).



A tristate buffer can be thought of as a switch. If *B* is on, the switch is closed. If B is off, the switch is open.

INPUT		OUTPUT
A	B	C
0	1	0
1		1
X	0	Z (high impedance)

Truth Table

State	Q0	Q1	Q2
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	0	0	1

UNIT - IV

A/D and D/A Converters:

Digital-to-analogue (*D/A*) and analogue-to-digital (*A/D*) converters constitute an essential link when digital

devices interface with analogue devices, and vice versa. They are important building blocks of any digital system, including both communication and noncommunication systems, besides having other applications. A D/A converter is important not only because it is needed at the output of most digital systems, where it converts a digital signal into an analogue voltage or current so that it can be fed to a chart recorder, for instance, for measurement purposes, or a servo motor in a control application; it is also important because it forms an indispensable part of the majority of A/D converter types. An A/D converter, too, has numerous applications. When it comes to transmitting analogue data, it forms an essential interface with a digital communication system where the analogue signal to be transmitted is digitized at the sending end with an A/D converter. It is invariably used in all digital read-out test and measuring equipment. Whether it is a digital multimeter or a digital storage oscilloscope or even a pH meter, an A/D converter is an important and essential component of all of them. In this chapter, we will discuss the operational fundamentals, the major performance specifications, along with their significance, and different types and applications of digital-to-analogue and analogue-to-digital converters, in addition to application-relevant information of some of the popular devices. A large number of solved examples is also included to illustrate the concepts.

Digital-to-Analogue Converters

D/A converter takes digital data at its input and converts them into analogue voltage or current that is proportional to the weighted sum of digital inputs. In the following paragraphs it is briefly explained. how different bits in the digital input data contribute a different quantum to the overall output analogue voltage or current, and also that the LSB has the least and the MSB the highest weight.

Simple Resistive Divider Network for D/A Conversion

Simple resistive networks can be used to convert a digital input into an equivalent analogue output. Figure 12.1 shows one such resistive network that can convert a three-bit digital input into an analogue output. This network, however, can be extended further to enable it to perform digital-to-analogue conversion of digital data with a larger number of bits. In the network of Fig. 12.1, if RL is much larger than R_ it can be proved with the help of simple network theorems that the output analogue voltage is given by

$$V_A = \frac{[V_1/R]+[V_2/(R/2)]+[V_3/(R/4)]}{[1/R]+[1/(R/2)]+[1/(R/4)]} \tag{12.1}$$

$$= \frac{[V_1/R]+[2V_2/R]+[4V_3/R]}{[1/R]+[2/R]+[4/R]} \tag{12.2}$$

$$= \frac{V_1+2V_2+4V_3}{7} \tag{12.3}$$

which can be further expressed as

$$V_A = \frac{V_1\times 2^0+V_2\times 2^1+V_3\times 2^2}{2^3-1} \tag{12.4}$$

The generalized expression of Equation (12.4) can be extended further to an n-bit D/A converter to get the following expression:

$$V_A = \frac{V_1\times 2^0+V_2\times 2^1+V_3\times 2^2+\cdots+V_n\times 2^{n-1}}{2^n-1} \tag{12.5}$$

In expression (12.5), if $V_1 = V_2 = \ldots = V_n = V$, then a logic '1' at the LSB position would contribute $V/(2^n-1)$ to the analogue output, and a logic '1' in the next adjacent higher bit position would

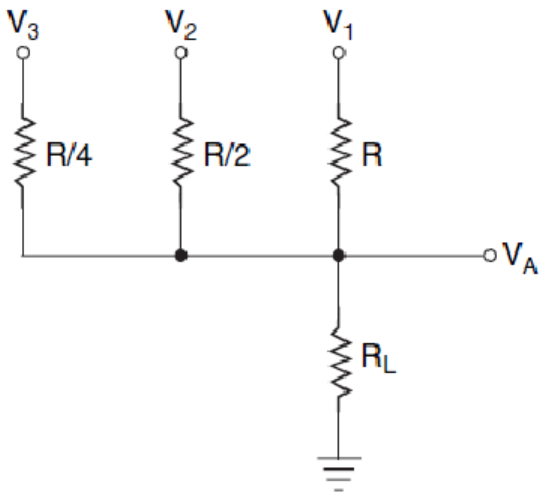


Figure 12.1 Simple resistive network for D/A conversion.

contribute $2V/(2^n - 1)$ to the output. The contributions of successive higher bit positions in the case of a logic '1' would be $4V/(2^n - 1)$, $8V/(2^n - 1)$, $16V/(2^n - 1)$ and so on. That is, the contribution of any given bit position owing to the presence of a logic '1' is twice the contribution of the adjacent lower bit position and half that of the adjacent higher bit position. When all input bit positions have a logic '1', the analogue output is given by

$$V_A = \frac{V(2^0 + 2^1 + 2^2 + \cdots + 2^{n-1})}{2^n - 1} = V \tag{12.6}$$

In the case of all inputs being in the logic '0' state, $V_A = 0$. Therefore, the analogue output varies from 0 to V volts as the digital input varies from an all 0s to an all 1s input.

12.1.2 Binary Ladder Network for D/A Conversion

The simple resistive divider network of Fig. 12.1 has two serious drawbacks. One, each resistor in the network is of a different value. Since these networks use precision resistors, the added expense becomes unattractive. Two, the resistor used for the most significant bit (MSB) is required to handle a much larger current than the LSB resistor. For example, in a 10-bit network, the current through the MSB resistor will be about 500 times the current through the LSB resistor.

To overcome these drawbacks, a second type of resistive network called the *binary ladder* (or *R*/2*R* ladder) is used in practice. The binary ladder, too, is a resistive network that produces an analogue output equal to the weighted sum of digital inputs. Figure 12.2 shows the binary ladder network for a four-bit D/A converter. As is clear from the figure, the ladder is made up of only two different values of resistor. This overcomes one of the drawbacks of the resistive divider network. It can be proved with the help of simple mathematics that the analogue output voltage V_A in the case of binary ladder network of Fig. 12.2 is given by

$$V_A = \frac{V_1 \times 2^0 + V_2 \times 2^1 + V_3 \times 2^2 + V_4 \times 2^3}{2^4} \tag{12.7}$$

In general, for an n-bit D/A converter using a binary ladder network

$$V_A = \frac{V_1 \times 2^0 + V_2 \times 2^1 + V_3 \times 2^2 + \cdots + V_n \times 2^{n-1}}{2^n} \tag{12.8}$$

For $V_1 = V_2 = V_3 = \cdots = V_n = V$, $V_A = [(2^n - 1)/2^n]V$. For $V_1 = V_2 = V_3 = \cdots = V_n = 0$, $V_A = 0$.

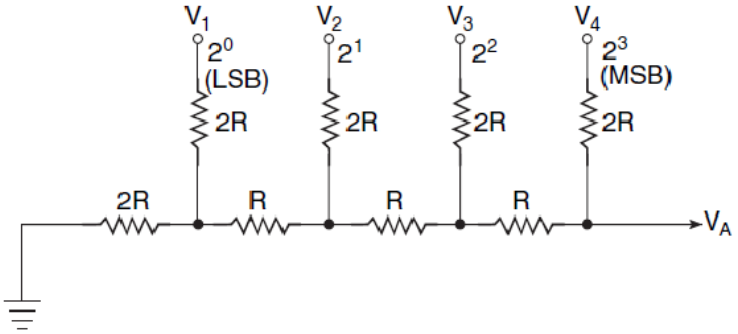


Figure 12.2 Binary ladder network for D/A conversion.

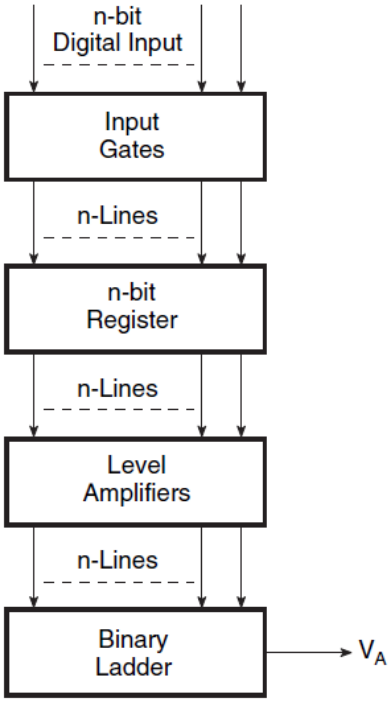


Figure 12.3 Block schematic representation of a D/A converter.

The analogue output voltage in this case varies from 0 (for an all 0s input) to $[(2^n - 1)/2^n]V$ (for an all 1s input).

The analogue output voltage in this case varies from 0 (for an all 0s input) to $[(2^n - 1)/2^n]V$ (for an all 1s input).

Also, in the case of a resistive divider network, the LSB contribution to the analogue output is $[1/(2^n - 1)]V$. This is also the minimum possible incremental change in the analogue output voltage. The same in the case of a binary ladder network would be $(1/2^n)V$.

A binary ladder network is the most widely used network for digital-to-analogue conversion, for obvious reasons. Although actual D/A conversion takes place in this network, a practical D/A converter device has additional circuitry such as a register for temporary storage of input digital data and level amplifiers to ensure that the digital signals presented to the resistive network are all of the same level. Figure 12.3 shows a block schematic representation of a complete n-bit D/A converter. D/A converters of different sizes (eight-bit, 12-bit, 16-bit, etc.) are available in the form of integrated circuits.

12.2 D/A Converter Specifications

The major performance specifications of a D/A converter include resolution, accuracy, conversion speed, dynamic range, nonlinearity (NL) and differential nonlinearity (DNL) and monotonocity.

12.2.1 Resolution

The *resolution* of a D/A converter is the number of states (2^n) into which the full-scale range is divided or resolved. Here, n is the number of bits in the input digital word. The higher the number of bits, the better is the resolution. An eight-bit D/A converter has 255 resolvable levels. It is said to

have a percentage resolution of $(1/255) \times 100 = 0.39\,\%$ or simply an eight-bit resolution. A 12-bit D/A converter would have a percentage resolution of $(1/4095) \times 100 = 0.0244\,\%$. In general, for an n-bit D/A converter, the percentage resolution is given by $(1/2^n - 1) \times 100$. The resolution in millivolts for the two cases for a full-scale output of 5 V is approximately 20 mV (for an eight-bit converter) and 1.2 mV (for a 12-bit converter).

12.2.2 Accuracy

The *accuracy* of a D/A converter is the difference between the actual analogue output and the ideal expected output when a given digital input is applied. Sources of error include the *gain error* (or full-scale error), the *offset error* (or zero-scale error), *nonlinearity errors* and a drift of all these factors. The gain error [Fig. 12.4(a)] is the difference between the actual and ideal output voltage, expressed as a percentage of full-scale output. It is also expressed in terms of LSB. As an example, an accuracy of $\pm 0.1\,\%$ implies that the analogue output voltage may be off by as much as ± 5 mV for a full-scale output of 5 V throughout the analogue output voltage range. The offset error is the error at analogue zero [Fig. 12.4(b)].

12.2.3 Conversion Speed or Settling Time

The *conversion speed* of a D/A converter is expressed in terms of its settling time. The *settling time* is the time period that has elapsed for the analogue output to reach its final value within a specified error band after a digital input code change has been effected. General-purpose D/A converters have a settling time of several microseconds, while some of the high-speed D/A converters have a settling

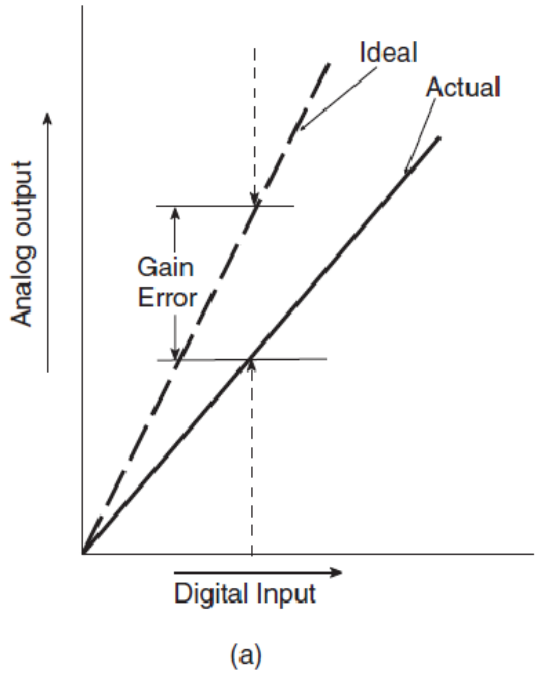
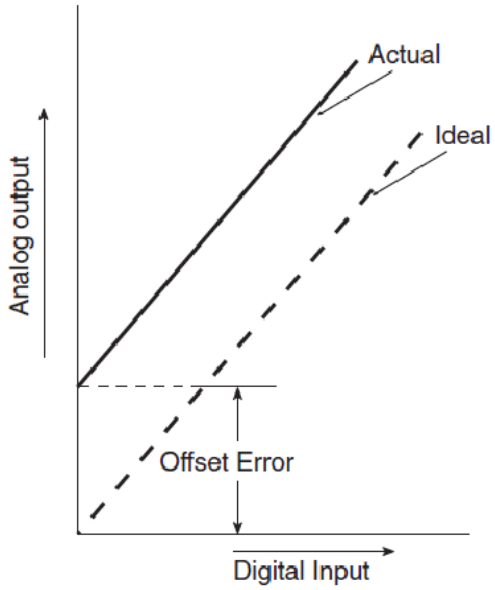


Figure 12.4 (a) Gain error and (b) offset error.



(b)

Figure 12.4 (*continued*).

time of a few nanoseconds. The settling time specification for D/A converter type number AD 9768 from Analog Devices USA, for instance, is 5 ns.

time of a few nanoseconds. The settling time specification for D/A converter type number AD 9768 from Analog Devices USA, for instance, is 5 ns.

12.2.4 Dynamic Range

This is the ratio of the largest output to the smallest output, excluding zero, expressed in dB. For linear D/A converters it is $20 \times \log 2^n$, which is approximately equal to $6n$. For companding-type D/A converters, discussed in Section 12.3, it is typically 66 or 72 dB.

12.2.5 Nonlinearity and Differential Nonlinearity

Nonlinearity (NL) is the maximum deviation of analogue output voltage from a straight line drawn between the end points, expressed as a percentage of the full-scale range or in terms of LSBs. *Differential nonlinearity* (DNL) is the worst-case deviation of any adjacent analogue outputs from the ideal one-LSB step size.

12.2.6 Monotonocity

In an ideal D/A converter, the analogue output should increase by an identical step size for every one-LSB increment in the digital input word. When the input of such a converter is fed from the output of a counter, the converter output will be a perfect staircase waveform, as shown in Fig. 12.5. In such cases, the converter is said to be exhibiting perfect monotonocity. A D/A converter is considered as monotonic if its analogue output either increases or remains the same but does not decrease as the digital input code advances in one-LSB steps. If the DNL error of the converter is less than or equal to twice its worst-case nonlinearity error, it guarantees monotonocity.

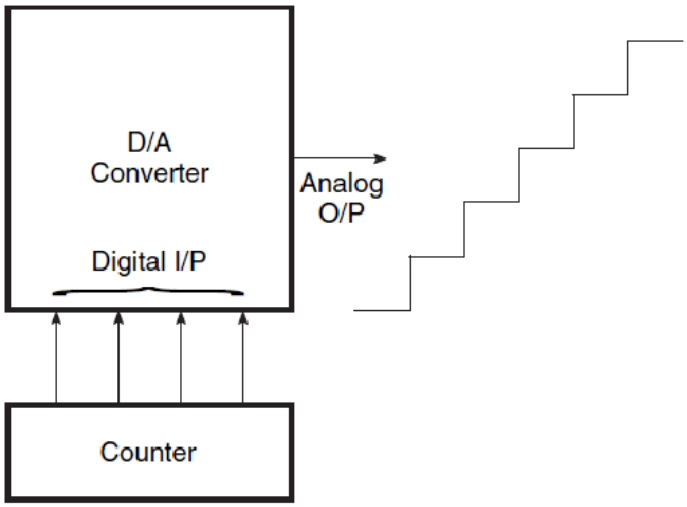


Figure 12.5 Monotonocity in a D/A converter.

12.3 Types of D/A Converter

The D/A converters discussed in this section include the following:

1. Multiplying-type D/A converters.
2. Bipolar-output D/A converters.
3. Companding D/A converters.

12.3.1 Multiplying D/A Converters

In a *multiplying-type D/A converter*, the converter multiplies an analogue reference by the digital input. Figure 12.6 shows the circuit representation. Some D/A converters can multiply only positive digital words by a positive reference. This is known as single quadrant (QUAD-I) operation. Two-quadrant operation (QUAD-I and QUAD-III) can be achieved in a D/A converter by configuring the output for bipolar operation. This is accomplished by offsetting the output by a negative MSB (equal to the analogue output of 1/2 of the full-scale range) so that the MSB becomes the sign bit.

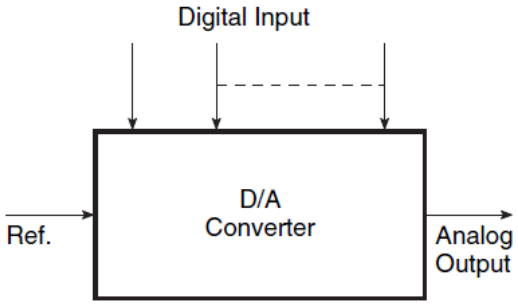


Figure 12.6 Multiplying-type D/A converter.

Some D/A converters even provide four-quadrant operation by allowing the use of both positive and negative reference. Multiplying D/A converters are particularly useful when we are looking for digitally programmable attenuation of an analogue input signal.

12.3.2 Bipolar-Output D/A Converters

In *bipolar-output D/A converters* the analogue output signal range includes both positive and negative values. The transfer characteristics of an ideal two-quadrant bipolar-output D/A converter are shown in Fig. 12.7.

12.3.3 Companding D/A Converters

Companding-type D/A converters are so constructed that the more significant bits of the digital input have a larger than binary relationship to the less significant bits. This decreases the resolution of the more significant bits, which in turn increases the analogue signal range. The effect of this is to compress more data into more significant bits.

12.4 Modes of Operation

D/A converters are usually operated in either of the following two modes of operation:

1. Current steering mode.
2. Voltage switching mode.

12.4.1 Current Steering Mode of Operation

In the *current steering mode* of operation of a D/A converter, the analogue output is a current equal to the product of a reference voltage and a fractional binary value D of the input digital word. D is equal to the sum of fractional binary values of different bits in the digital word. Also, fractional binary values of different bits in an n-bit digital word starting from the LSB are $2^0/2^n$, $2^1/2^n$, $2^2/2^n$, . . . , $2^{n-1}/2^n$.

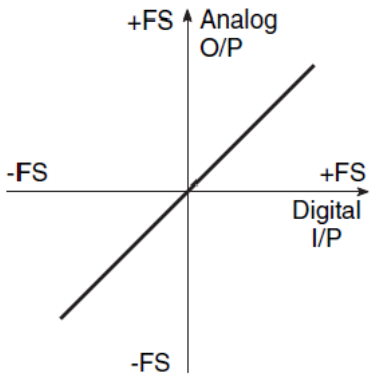


Figure 12.7 Bipolar-output D/A converter transfer characteristics.

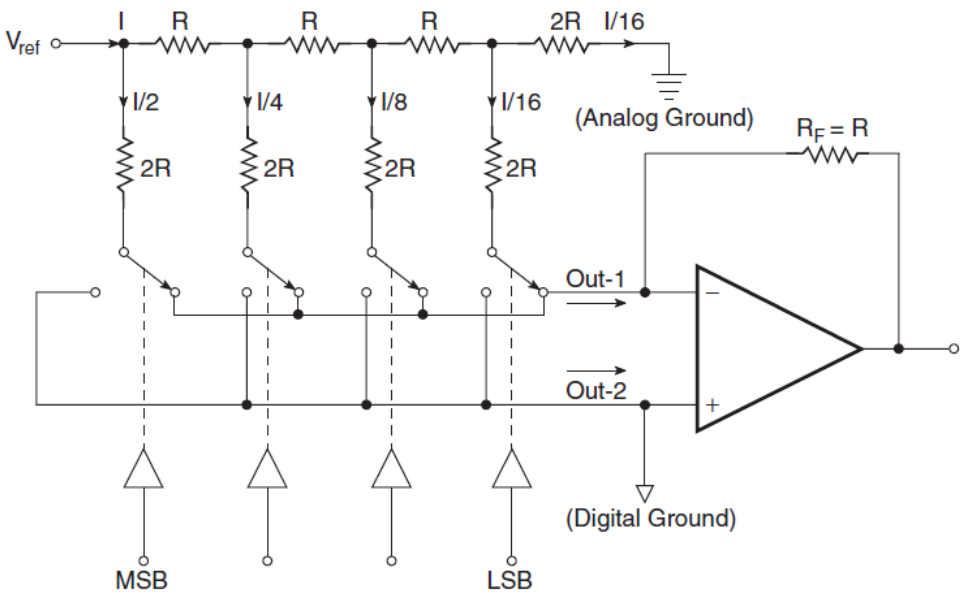


Figure 12.8 Current steering mode of operation of a D/A converter.

The output current is often converted into a corresponding voltage using an external opamp wired as a current-to-voltage converter. Figure 12.8 shows the circuit arrangement. The majority of D/A converters in IC form have an in-built opamp that can be used for current-to-voltage conversion. For the circuit arrangement of Fig. 12.8, if the feedback resistor R_F equals the ladder resistance R, the analogue output voltage at the opamp output is $-(D.V_{ref})$.

The output current is often converted into a corresponding voltage using an external opamp wired as a current-to-voltage converter. Figure 12.8 shows the circuit arrangement. The majority of D/A converters in IC form have an in-built opamp that can be used for current-to-voltage conversion. For the circuit arrangement of Fig. 12.8, if the feedback resistor R_F equals the ladder resistance R, the analogue output voltage at the opamp output is $-(D.V_{ref})$.

The arrangement of the four-bit D/A converter of Fig. 12.8 can be conveniently used to explain the operation of a D/A converter in the current steering mode. The $R/2R$ ladder network divides the input current I due to a reference voltage V_{ref} applied at the reference voltage input of the D/A converter into binary weighted currents, as shown. These currents are then steered to either the output designated Out-1 or Out-2 by the current steering switches. The positions of these current steering switches are controlled by the digital input word. A logic '1' steers the corresponding current to Out-1, whereas a logic '0' steers it to Out-2. For instance, a logic '1' in the MSB position will steer the current $I/2$ to Out-1. A logic '0' steers it to Out-2, which is the ground terminal. In the four-bit converter of Fig. 12.8, the analogue output current (or voltage) will be maximum for a digital input of 1111. The analogue output current in this case will be $I/2 + I/4 + I/8 + I/16 = (15/16)I$. The analogue output voltage will be $(-15/16)IR_F = (-15/16)IR$. Also, $I = V_{ref}/R$ as the equivalent resistance of the ladder network across V_{ref} is also R. The analogue output voltage is then $[(-15/16)(V_{ref})/R] \times R = (-15/16)V_{ref}$. Here, 15/16 is nothing but the fractional binary value of digital input 1111. In general, the maximum analogue output voltage is given by $-(1-2^{-n}) \times V_{ref}$, where n is the number of bits in the input digital word.

12.4.2 Voltage Switching Mode of Operation

In the *voltage switching mode* of operation of a $R/2R$ ladder type D/A converter, the reference voltage is applied to the Out-1 terminal and the output is taken from the reference voltage terminal. Out-2 is joined to analogue ground. Figure 12.9 shows a four-bit D/A converter of the $R/2R$ ladder type in

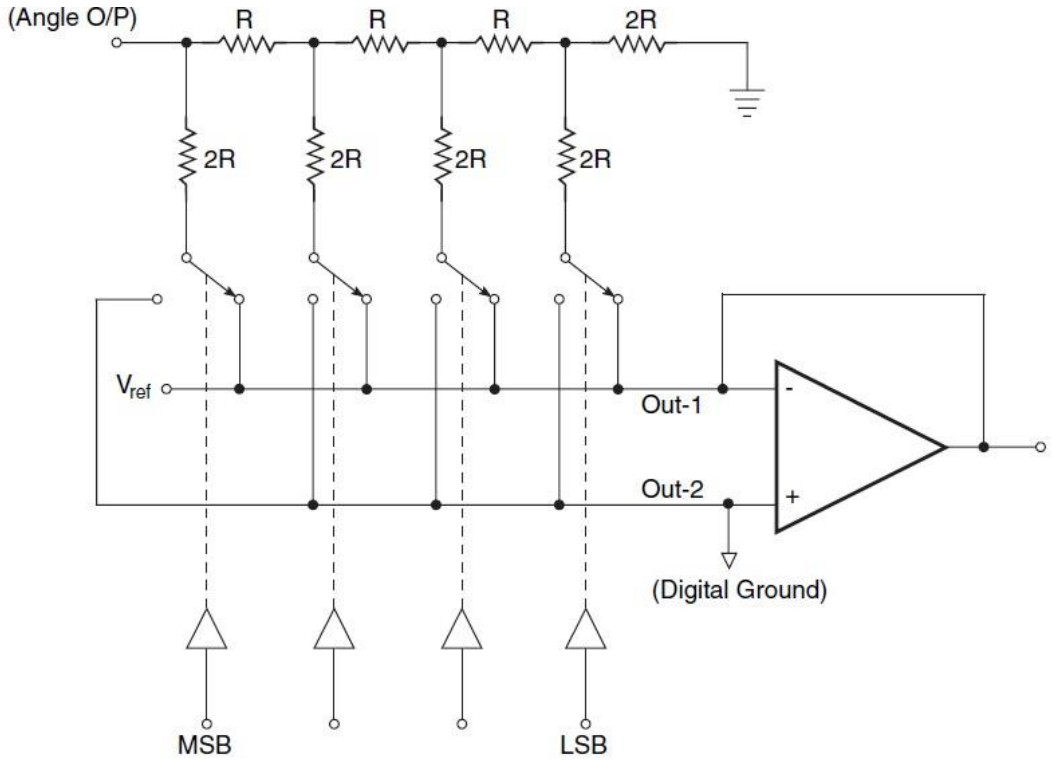


Figure 12.9 Voltage switching mode of operation of a D/A converter.

voltage switching mode of operation. The output voltage is the product of the fractional binary value

voltage switching mode of operation. The output voltage is the product of the fractional binary value of the digital input word and the reference voltage applied at the Out-1 terminal, i.e. D_V_{ref} _ As the positive reference voltage produces a positive analogue output voltage, the voltage switching mode of operation is possible with a single supply. As the circuit produces analogue output voltage, it obviates the need for an opamp and the feedback resistor. However, the reference voltage applied to the Out-1 terminal in this case will see different input impedances for different digital inputs. For this reason, the source of the input is buffered.

UNIT - V

Semiconductor Memories and Programmable Logic Devices:

Introduction:

A memory unit is a device to which binary information is transferred for storage and from which information is retrieved when needed for processing. When data processing takes place, information from memory is trans
ferred to selected registers in the processing unit.
Intermediate and final results obtained in the processing unit are transferred back to be stored in memory. Binary information received from an input device is stored in memory, and information transferred to an output device is taken from memory. A memory unit is a collection of cells capable of storing a large quantity of binary information.

There are two types of memories that are used in digital systems: *random access memory* (RAM) and *read only memory* (ROM). RAM stores new information for later use. The process of storing new information into memory is referred to as a memory *write* operation. The process of

transferring the stored information out of memory is referred to as a memory *read* operation. RAM can perform both write and read operations.

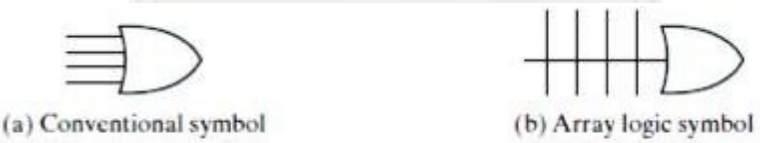
ROM can perform only the read operation. This means that suitable binary information is already stored inside memory and can be retrieved or read at any time. However, that information cannot be altered by writing.

ROM is a *programmable logic device* (PLD). The binary information that is stored within such a device is specified in some fashion and then embedded within the hardware in a process is referred to as *programming* the device. The word "programming" here refers to a hardware procedure which specifies the bits that are inserted into the hardware configuration of the device. ROM is one example of a PLD. Other such units are the programmable logic array (PLA), programmable array logic (PAL), and the field-programmable gate array (FPGA).

A PLD is an integrated circuit with internal logic gates connected through electronic intact. Programming the device involves blowing those fuses along the paths that must be removed in order to obtain the particular configuration of the desired logic function.

We introduce the configuration of PLDs and indicate procedures for their use in the design of digital systems. We also present CMOS FPGAs, which are configured by downloading a stream of bits into the device to configure transmission gates to establish the internal connectivity required by a specified logic function (combinational or sequential).

A typical PLD may have hundreds to millions of gates interconnected through hundreds to thousands of internal paths. In order to show the internal logic diagram of such a device in a concise form, it is necessary to employ a special gate symbology applicable to array logic. Figure shows the conventional and array logic symbols for a multiple input OR gate. Instead of having multiple input lines into the gate, we draw a single line entering the gate. The input lines are drawn perpendicular to this single line and are connected to the gate through internal fuses. In a similar fashion, we can draw the array logic for an AND gate. This type of graphical representation for the inputs of gates will be used throughout the chapter in array logic diagrams.



Conventional and array logic diagrams for OR gate

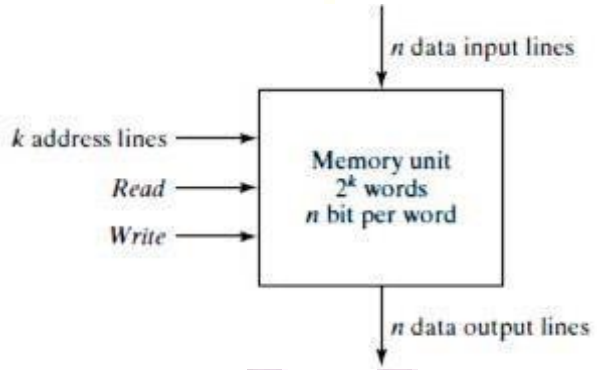
RANDOM-ACCESS MEMORY

A memory unit is a collection of storage cells, together with associated circuits needed to transfer information into and out of a device. The architecture of memory is such that information can be selectively retrieved from any of its internal locations. The time it takes to transfer information to or from any desired random location is always the same—hence the name *random access memory*, abbreviated RAM. In contrast, the time required to retrieve information that is stored on magnetic tape depends on the location of the data.

A memory unit stores binary information in groups of bits called *words*. A word in memory is an entity of bits that move in and out of storage as a unit. A memory word is a group of 1's and 0's and may represent a number, an instruction, one or more alphanumeric characters, or any other binary-coded information. A group of 8 bits is called a *byte*. Most computer memories use words that are multiples of 8 bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is

made up of four bytes. The capacity of a memory unit is usually stated as the total number of bytes that the unit can store.

Communication between memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer. A block diagram of a memory unit is shown in Fig. below. The *n* data input lines provide the information to be stored in memory, and the *n* data output lines supply the information coming out of memory. The *k* address lines specify the particular word chosen among the many available. The two control inputs specify the direction of transfer desired: The *Write* input causes binary data toe transferred into the memory, and the *Read* input causes binary data to be transferred out of memory.



Block Diagram of a memory unit

The memory unit is specified by the number of words it contains and the number of bits in each word. The address lines select one particular word. Each word in memory is assigned an identification number, called an *address*, starting from 0 up to 2*k* - 1, where *k* is the number of address lines. The selection of a specific word inside memory is done by applying the *k* -bit address to the address lines. An internal decoder accepts this address and opens the paths needed to select the word specified. Memories vary greatly in size and may range from 1,024 words, requiring an address of 10 bits, to 232 words, requiring 32 address bits. It is customary to refer to the number of words (or bytes) in memory with one of the letters K (kilo), M (mega), and G (giga). K is equal to 210, M is equal to 220, and G is equal to 230. Thus, 64K = 216, 2M = 221, and 4G = 232.

Consider, for example, a memory unit with a capacity of 1K words of 16 bits each. Since 1K = 1,024 = 210 and 16 bits constitute two bytes, we can say that the memory can accommodate 2,048 = 2K bytes. Below figure shows possible contents of the first three and the last three words of this memory. Each word contains 16 bits that can be divided into two bytes. The words are recognized by their decimal address from 0 to 1,023. The equivalent binary address consists of 10 bits. The first address is specified with ten 0's; the last address is specified with ten 1's, because 1,023 in binary is equal to 1111111111. A word in memory is selected by its binary address. When a word is read or written, the memory operates on all 16 bits as a single unit.

Memory address		Memory content
Binary	Decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
⋮	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Contents of a 1024 * 16 memory

Write and Read Operations

The two operations that RAM can perform are the write and read operations. As alluded to earlier, the write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals, the internal circuits inside the memory provide the desired operation.

The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data input lines.
3. Activate the *write* input.

The memory unit will then take the bits from the input data lines and store them in the word specified by the address lines.

The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Apply the binary address of the desired word to the address lines.
2. Activate the *read* input

The memory unit will then take the bits from the word that has been selected by the address and apply them to the output data lines. The contents of the selected word do not change after the read operation, i.e., the word operation is nondestructive.

Commercial memory components available in integrated-circuit chips sometimes provide the two control inputs for reading and writing in a somewhat different configuration. Instead of having separate read and write inputs to control the two operations, most integrated circuits provide two other control inputs: One input selects the unit and the other determines the operation. The memory operations that result from these control inputs are specified in Table below.

Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

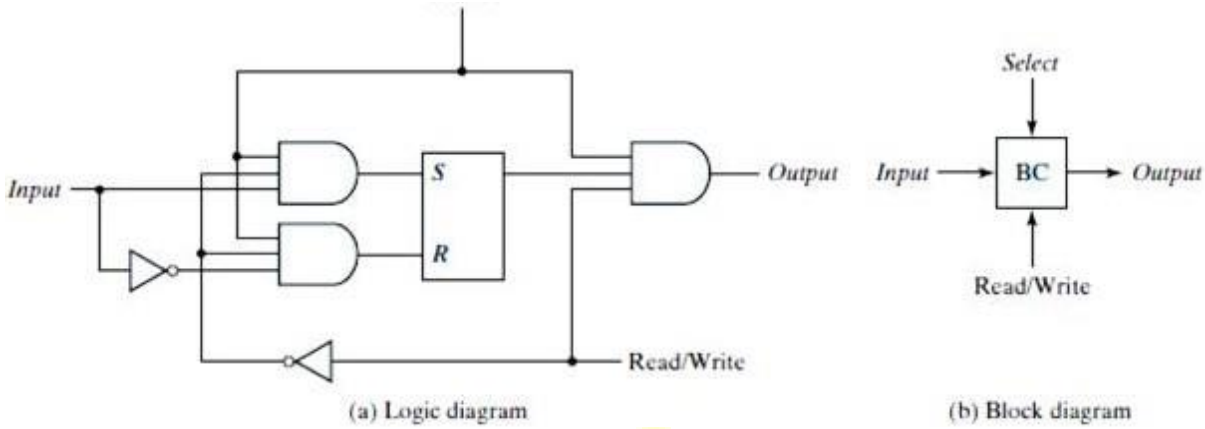
The memory enable (sometimes called the chip select) is used to enable the particular memory chip in a multichip implementation of a large memory. When the memory enable is inactive, the memory chip is not selected and no operation is performed. When the memory enable input is active, the read/write input determines the operation to be performed.

Memory Decoding

In addition to requiring storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address. In this section, we present the internal construction of a RAM and demonstrate the operation of the decoder. To be able to include the entire memory in one diagram, the memory unit presented here has a small capacity of 16 bits, arranged in four words of 4 bits each. An example of a two-dimensional coincident decoding arrangement is presented to show a more efficient decoding scheme that is used in large memories. We then give an example of address multiplexing commonly used in DRAM integrated circuits.

Internal Construction

The internal construction of a RAM of *m* words and *n* bits per word consists of *m* * *n* binary storage cells and associated decoding circuits for selecting individual words. The binary storage cell is the basic building block of a memory unit. The equivalent logic of a binary cell that stores one bit of information is shown in Fig. below. The storage part of the cell is modeled by an *SR* latch with associated gates to form a *D* latch. Actually, the convenient to model it in terms of logic symbols. A binary storage cell must be very small in order to be able to pack as many cells as possible in the small area available in the integrated circuit chip. The binary cell stores one bit in its internal latch. The select input enables the cell for reading or writing, and the read/write input determines the operation of the cell when it is selected. A 1 in the read/write input provides the read operation by forming a path from the latch to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the latch.



Memory cell

The logical construction of a small RAM is shown in Fig. below. This RAM consists of four words of four bits each and has a total of 16 binary cells. The small blocks labeled BC represent the binary cell with its three inputs and one output, as specified in Fig. above. A memory with four words needs two address lines. The two address inputs go through a 2 * 4 decoder to select one of the four words. The decoder is enabled with the memory-enable input. When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected. With the memory select at 1, one of the four words is selected, dictated by the value in the two address lines. Once a word has been selected, the read/write input determines the operation. During the read operation, the four bits of the selected word go through OR gates to the output terminals. During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word. The binary cells that are not selected are disabled, and their previous binary values remain unchanged. When the memory select input that goes into the decoder is equal to 0, none of the words are selected and the contents of all cells remain unchanged regardless of the value of the read/write input.

Commercial RAMs may have a capacity of thousands of words, and each word may range from 1 to 64 bits. The logical construction of a large-capacity memory would be a direct extension of the configuration shown here. A memory with $2k$ words of n bits per word requires k address lines that go into a $k * 2k$ decoder. Each one of the decoder outputs selects one word of n bits for reading or writing.

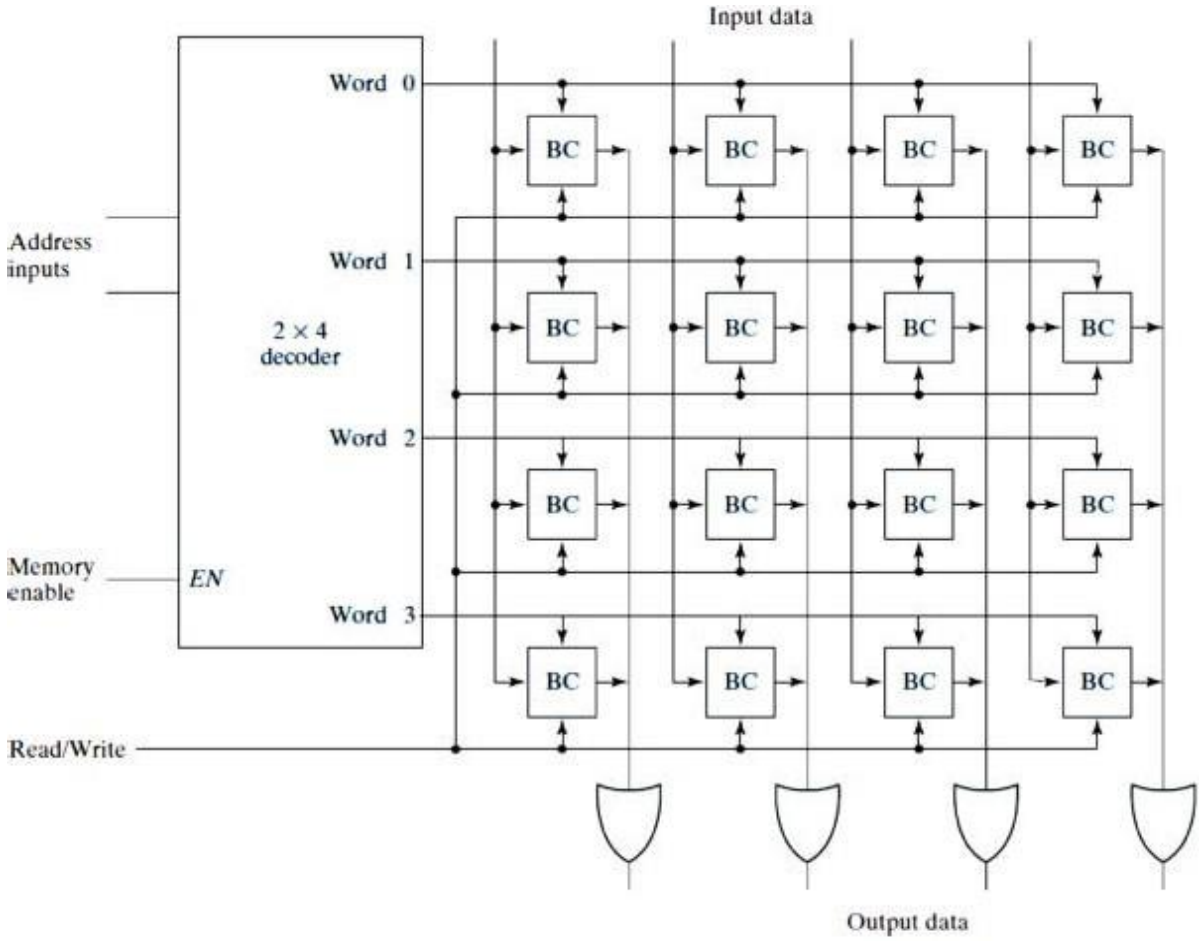
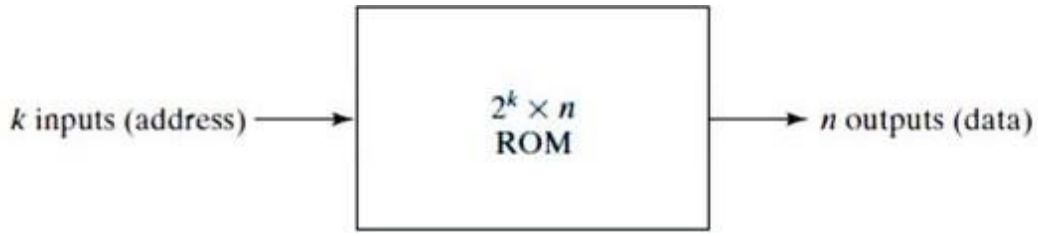


Diagram of a 4 * 4 RAM

READ ONLY MEMORY:

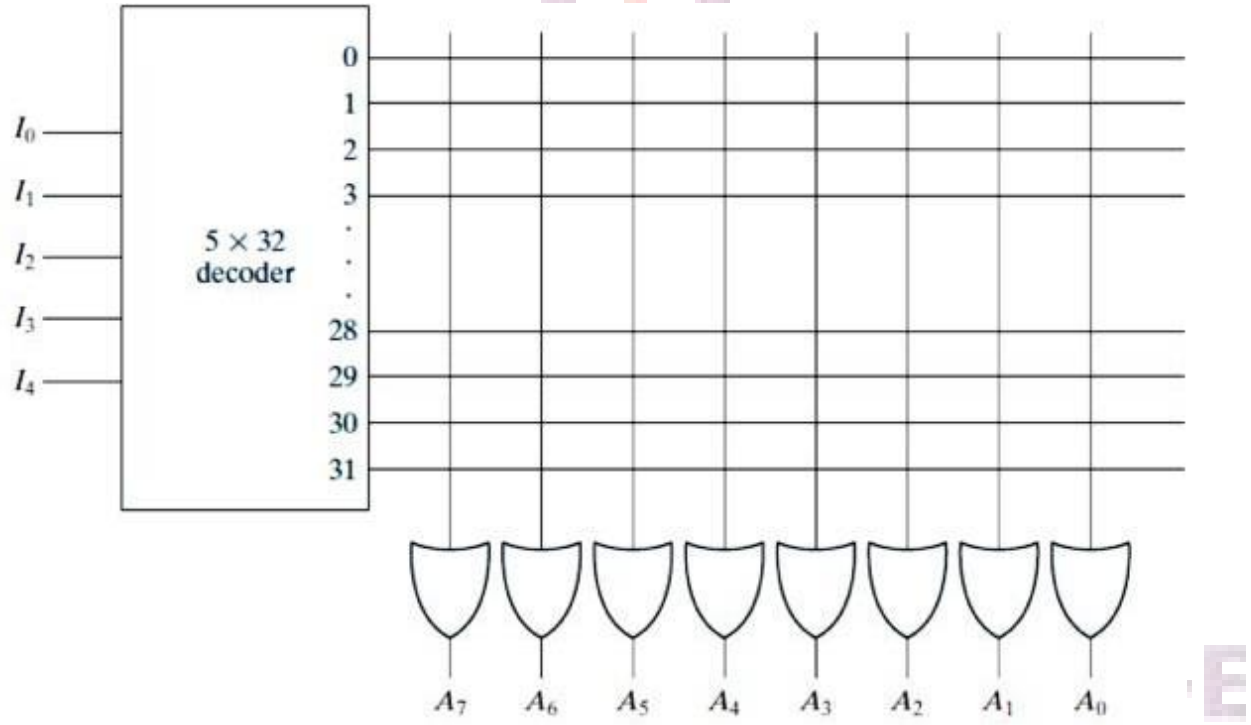
A read-only memory (ROM) is essentially a memory device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established, it stays within the unit even when power is turned off and on again.

A block diagram of a ROM consisting of k inputs and n outputs is shown in Fig. below. The inputs provide the address for memory, and the outputs give the data bits of the stored word that is selected by the address. The number of words in a ROM is determined from the fact that k address input lines are needed to specify $2k$ words. Note that ROM does not have data inputs, because it does not have a write operation. Integrated circuit ROM chips have one or more enable inputs and sometimes come with three-state outputs to facilitate the construction of large arrays of ROM.



ROM block diagram

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



Internal logic of a 32: 8 ROM

Combinational Circuit Implementation

It was shown that a decoder generates the 2^k minterms of the *k* input variables. By inserting OR gates to sum the minterms of Boolean functions, we were able to generate any desired

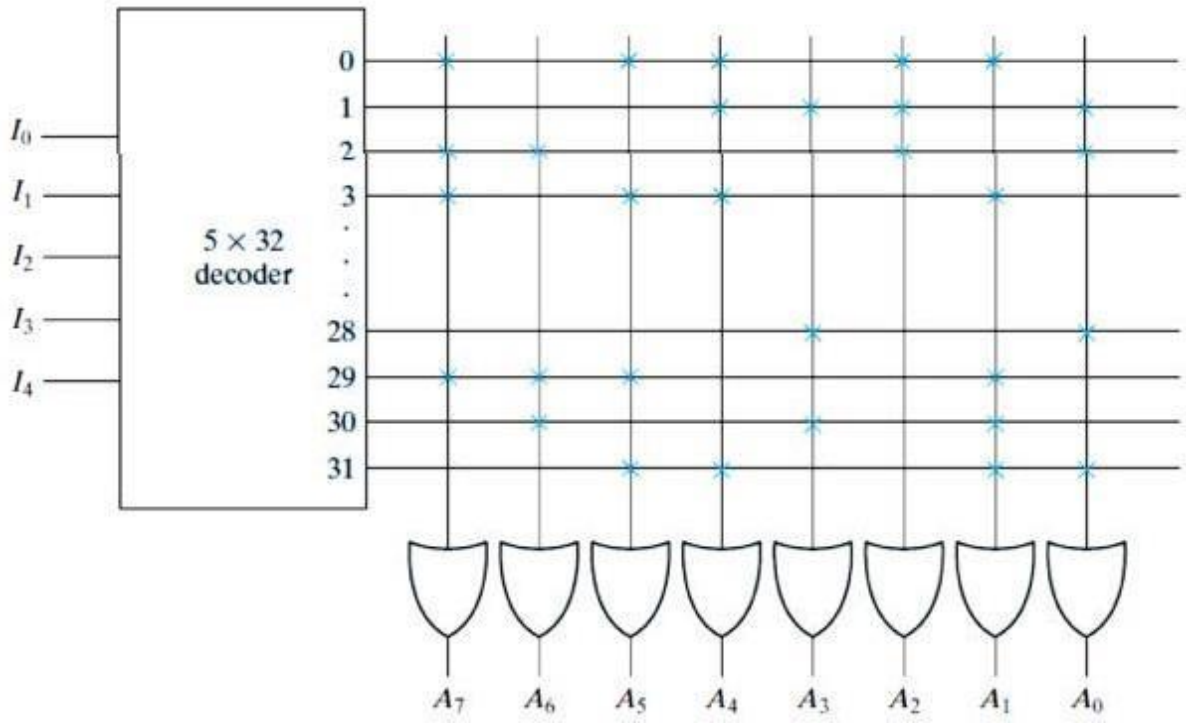
combinational circuit. The ROM is essentially a device that includes both the decoder and the OR gates within a single device to form a minterm generator. By choosing connections for those minterms which are included in the function, the ROM outputs can be programmed to represent the Boolean functions of the output variables in a combinational circuit.

The internal operation of a ROM can be interpreted in two ways. The first interpretation is that of a memory unit that contains a fixed pattern of stored words. The second interpretation is that of a unit which implements a combinational circuit. From this point of view, each output terminal is considered separately as the output of a Boolean function expressed as a sum of minterms. For example, the ROM may be considered to be a combinational circuit with eight outputs, each a function of the five input variables. Output A_7 can be expressed in sum of minterms as

$$A_7(I_4, I_3, I_2, I_1, I_0) = \sum m(0, 2, 3, \ldots, 29)$$

(The three dots represent minterms 4 through 27, which are not specified in the figure.) A connection marked with * in the figure produces a minterm for the sum. All other crosspoints are not connected and are not included in the sum. In practice, when a combinational circuit is designed by means of a ROM, it is not necessary to design the logic or to show the internal gate connections inside the unit. All that the designer has to do is specify the particular ROM by its IC number and provide the applicable truth table. The truth table gives all the information for programming the ROM. No internal logic diagram is needed to accompany the truth table.

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



Programming the ROM according to Table given above

Types of ROMs

The required paths in a ROM may be programmed in four different ways. The first is called *mask programming* and is done by the semiconductor company during the last fabrication process of the unit. The procedure for fabricating a ROM requires that the customer fill out the truth table he or she wishes the ROM to satisfy. The truth table may be submitted in a special form provided by the manufacturer or in a specified format on a computer output medium. The manufacturer makes the corresponding mask for the paths to produce the 1's and 0's according to the customer's truth table. This procedure is costly because the vendor charges the customer a special fee for custom masking the particular ROM. For this reason, mask programming is economical only if a large quantity of the same ROM configuration is to be ordered.

For small quantities, it is more economical to use a second type of ROM called *programmable read only memory,* or PROM. When ordered, PROM units contain all the fuses intact, giving all 1's in the bits of the stored words. The fuses in the PROM are blown by the application of a high-voltage pulse to the device through a special pin. A blown fuse defines a binary 0 state and an intact fuse gives a binary 1 state. This procedure allows the user to program the PROM in the laboratory to achieve the desired relationship between input addresses and stored words. Special instruments called PROM programmers are available commercially to facilitate the procedure. In any case, all procedures for programming ROMs are hardware procedures, even though the word *programming* is used.

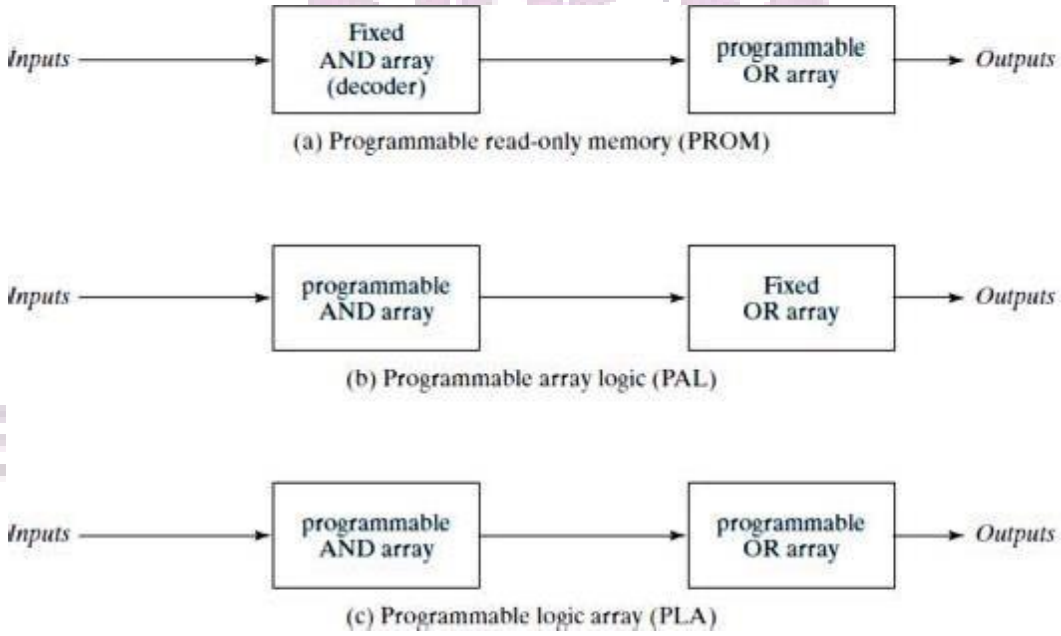
The hardware procedure for programming ROMs or PROMs is irreversible, and once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed. A third type of ROM is

the *erasable PROM*, or EPROM, which can be restructured to the initial state even though it has been programmed previously. When the EPROM is placed under a special ultraviolet light for a given length of time, the shortwave radiation discharges the internal floating gates that serve as the programmed connections. After erasure, the EPROM returns to its initial state and can be reprogrammed to a new set of values.

The fourth type of ROM is the electrically erasable PROM (EEPROM or E2PROM). This device is like the EPROM, except that the previously programmed connections can be erased with an electrical signal instead of ultraviolet light. The advantage is that the device can be erased without removing it from its socket.

Combinational PLDs

The PROM is a combinational programmable logic device (PLD)—an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND–OR sum-of-product implementation. There are three major types of combinational PLDs, differing in the placement of the programmable connections in the AND– OR array. Below figure shows the configuration of the three PLDs. The PROM has a fixed AND array constructed as a decoder and a programmable OR array. The programmable OR gates implement the Boolean functions in sum-of-minterms form. The PAL has a programmable AND array and a fixed OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate. The most flexible PLD is the PLA, in which both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum-of-products implementation. The names PAL and PLA emerged from different vendors during the development of PLDs. The implementation of combinational circuits with PROM was demonstrated in this section. The design of combinational circuits with PLA and PAL is presented in the next two sections.



Basic configuration of three PLDs

PROGRAMMABLE LOGIC ARRAY

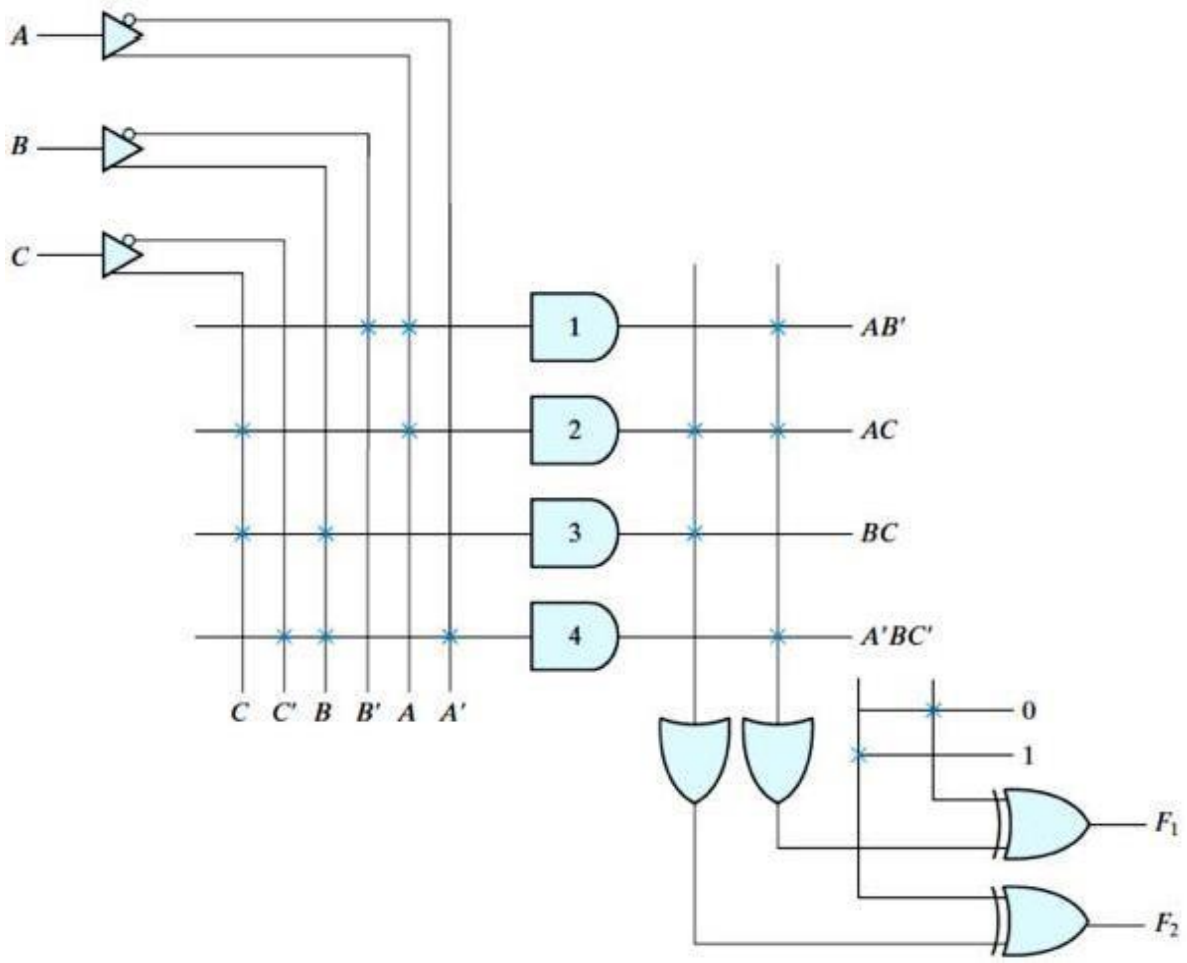
The PLA is similar in concept to the PROM, except that the PLA does not provide full decoding of the variables and does not generate all the minterms. The decoder is replaced by an array of AND gates that can be programmed to generate any product term of the input variables. The product terms are then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

The internal logic of a PLA with three inputs and two outputs is shown in Fig. below. Such a circuit is too small to be useful commercially, but is presented here to demonstrate the typical logic configuration of a PLA. The diagram uses the array logic graphic symbols for complex circuits. Each input goes through a buffer–inverter combination, shown in the diagram with a composite graphic symbol, that has both the true and complement outputs. Each input and its complement is connected to the inputs of each AND gate, as indicated by the intersections between the vertical and horizontal lines. The outputs of the AND gates are connected to the inputs of each OR gate. The output of the OR gate goes to an XOR gate, where the other input can be programmed to receive a signal equal to either logic 1 or logic 0. The output is inverted when the XOR input is connected to 1 (since x XOR 1 = x'). The output does not change when the XOR input is connected to 0 (since x *XOR* 0 = x).

The particular Boolean functions implemented in the PLA of below Fig. are

$$F1 = AB' + AC + A'BC'$$
$$F2 = (AC + BC)'$$

The product terms generated in each AND gate are listed along the output of the gate in the diagram. The product term is determined from the inputs whose crosspoints are connected and marked with a *. The output of an OR gate gives the logical sum of the selected product terms. The output may be complemented or left in its true form, depending on the logic being realized.



PLA with three inputs, four product terms, and two outputs

The fuse map of a PLA can be specified in a tabular form. For example, the programming table that specifies the PLA of above Fig. is listed in above Table. The PLA programming table consists of three sections. The first section lists the product terms numerically. The second section specifies the required paths between inputs and AND gates. The third section specifies the paths between the AND and OR gates. For each output variable, we may have a T (for true) or C (for complement) for programming the XOR gate. The product terms listed on the left are not part of the table; they are included for reference only. For each product term, the inputs are marked with 1, 0, or — (dash). If a variable in the product term appears in the form in which it is true, the corresponding input variable is marked with a 1. If it appears complemented, the corresponding input variable is marked with a 0. If the variable is absent from the product term, it is marked with a dash.

The paths between the inputs and the AND gates are specified under the column head "Inputs" in the programming table. A 1 in the input column specifies a connection from the input variable to the AND gate. A 0 in the input column specifies a connection from the complement of the variable to the input of the AND gate. A dash specifies a blown fuse in both the input variable and its complement. It is assumed that an open terminal in the input of an AND gate behaves like a 1.

The paths between the AND and OR gates are specified under the column head "Outputs." The output variables are marked with 1's for those product terms which are included in the function. Each product term that has a 1 in the output column requires a path from the output of the AND gate to the input of the OR gate. Those marked with a dash specify a blown fuse. It is assumed that an open terminal in the input of an OR gate behaves like a 0. Finally, a T (true) output dictates that the other input of the corresponding XOR gate be connected to 0, and a C (complement) specifies a connection to 1.

The size of a PLA is specified by the number of inputs, the number of product terms, and the number of outputs. A typical integrated circuit PLA may have 16 inputs, 48 product terms, and eight outputs. For *n* inputs, *k* product terms, and *m* outputs, the internal logic of the PLA consists of *n* buffer–inverter gates, *k* AND gates, *m* OR gates, and *m* XOR gates. There are $2n * k$ connections between the inputs and the AND array, $k * m$ connections between the AND and OR arrays, and *m* connections associated with the XOR gates.

In designing a digital system with a PLA, there is no need to show the internal connections of the unit as was done in Fig. above. All that is needed is a PLA programming table from which the PLA can be programmed to supply the required logic. As with a ROM, the PLA may be mask programmable or field programmable. With mask programming, the customer submits a PLA program table to the manufacturer. This table is used by the vendor to produce a custom-made PLA that has the required internal logic specified by the customer. A second type of PLA that is available is the field programmable logic array, or FPLA, which can be programmed by the user by means of a commercial hardware programmer unit.

In implementing a combinational circuit with a PLA, careful investigation must be undertaken in order to reduce the number of distinct product terms, since a PLA has a finite number of AND gates. This can be done by simplifying each Boolean function to a minimum number of terms. The number of literals in a term is not important, since all the input variables are available anyway. Both the true value and the complement of each function should be simplified to see which one can be expressed with fewer product terms and which one provides product terms that are common to other functions.

PAL

The PAL is a programmable logic device with a fixed OR array and a programmable AND array. Because only the AND gates are programmable, the PAL is easier to program than, but is not as flexible as, the PLA. Figure 7.16 shows the logic configuration of a typical PAL with four inputs and four outputs. Each input has a buffer–inverter gate, and each output is generated by a fixed OR gate. There are four sections in the unit, each composed of an AND–OR array that is *three wide*, the term used to indicate that there are three programmable AND gates in each section and one fixed OR gate. Each AND gate has 10 programmable input connections, shown in the diagram by 10 vertical lines intersecting each horizontal line. The horizontal line symbolizes the multiple-input configuration of the AND gate. One of the outputs is connected to a buffer–inverter gate and then fed back into two inputs of the AND gates.

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

The number of product terms in each section is fixed, and if the number of terms in the function is too large, it may be necessary to use two sections to implement one Boolean function.

As an example of using a PAL in the design of a combinational circuit, consider the following Boolean functions, given in sum-of-minterms form:

$$w(A, B, C, D) = g(2, 12, 13)$$
$$x(A, B, C, D) = g(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = g(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = g(1, 2, 8, 12, 13)$$

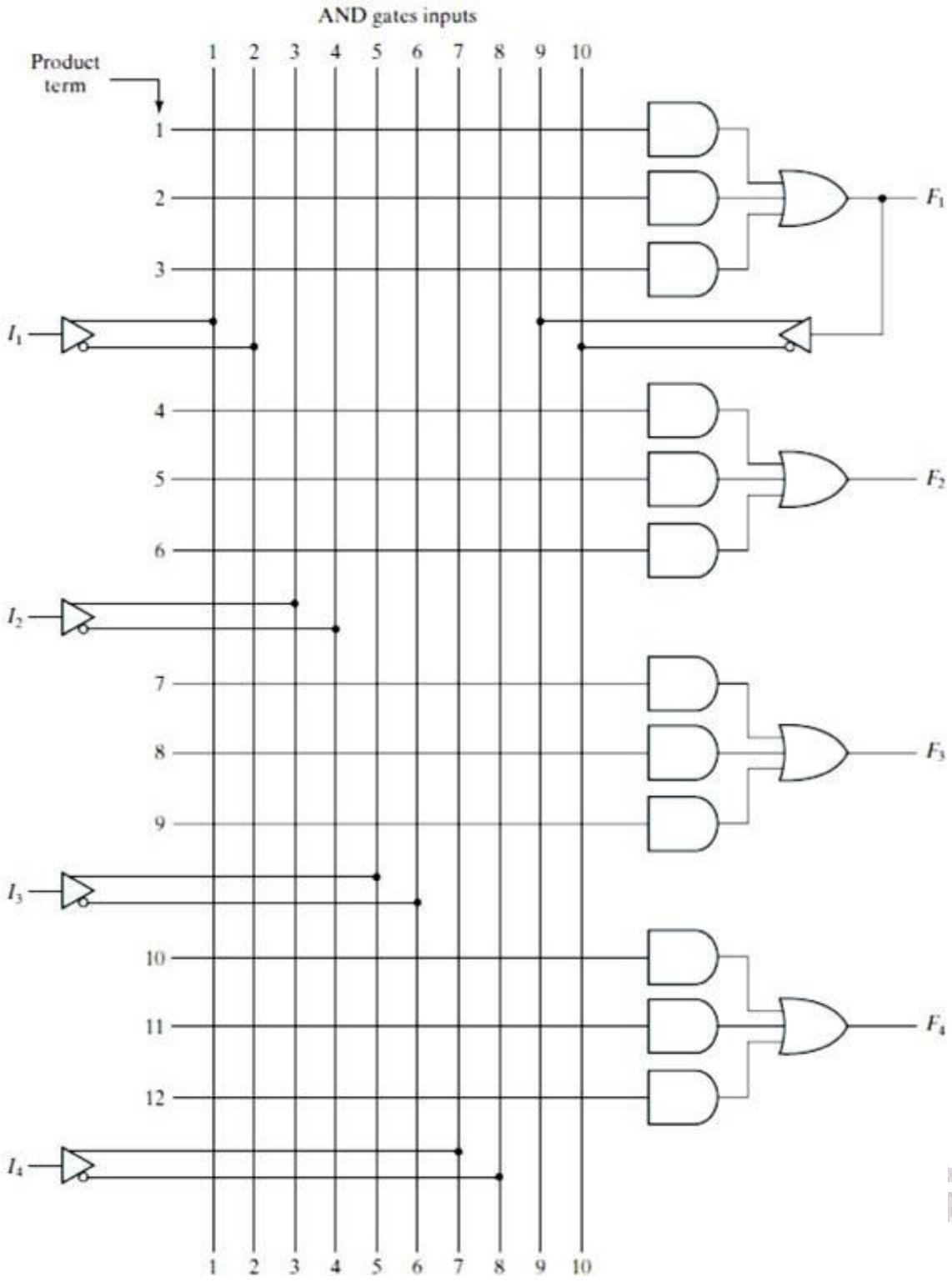
Simplifying the four functions to a minimum number of terms results in the following Boolean functions:

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$
$$= w + AC'D' + A'BC'D$$

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

PAL Programming Table

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



PAL with four inputs, four outputs, and a three-wide AND–OR structure

Unit III

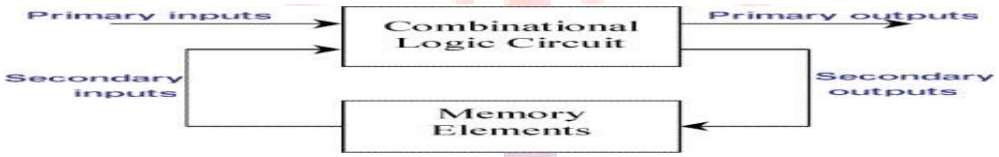
Sequential Circuits-I

Sequential circuits

Classification of sequential circuits: Sequential circuits may be classified as two types.

1. Synchronous sequential circuits
2. Asynchronous sequential circuits

Combinational logic refers to circuits whose output is strictly depended on the present value of the inputs. As soon as inputs are changed, the information about the previous inputs is lost, that is, combinational logics circuits have no memory. Although every digital system is likely to have combinational circuits, most systems encountered in practice also include memory elements, which require that the system be described in terms of sequential logic. Circuits whose output depends not only on the present input value but also the past input value are known as **sequential logic circuits**. The mathematical model of a sequential circuit is usually referred to as a **sequential machine**.



Comparison between combinational and sequential circuits

Combinational circuit	Sequential circuit
1. In combinational circuits, the output variables at any instant of time are dependent only on the present input variables	1. in sequential circuits the output variables at any instant of time are dependent not only on the present input variables, but also on the present state
2.memory unit is not requires in combinational circuit	2.memory unit is required to store the past history of the input variables
3. these circuits are faster because the delay between the i/p and o/p due to propagation delay of gates only	3. sequential circuits are slower than combinational circuits
4. easy to design	4. comparatively hard to design

Level mode and pulse mode asynchronous sequential circuits:

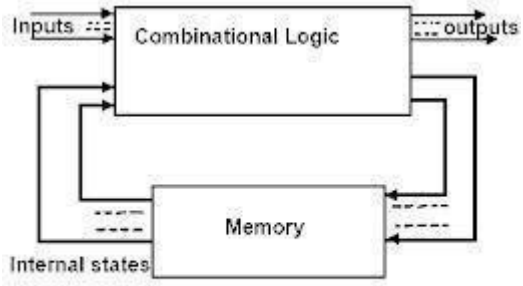


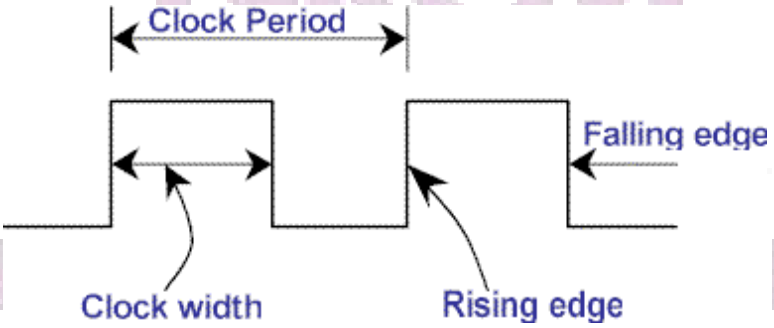
Figure 1: Asynchronous Sequential Circuit

Fig shows a block diagram of an asynchronous sequential circuit. It consists of a combinational circuit and delay elements connected to form the feedbackloops. The present state and next state variables in asynchronous sequential circuits called secondary variables and excitation variables respectively..

There are two types of asynchronous circuits: fundamental mode circuits and pulse mode circuits.

Synchronous and Asynchronous Operation:

Sequential circuits are divided into two main types: **synchronous** and **asynchronous**. Their classification depends on the timing of their signals.***Synchronous*** sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a free-running **clock signal**. The clock signal is generally some form of square wave as shown in Figure below.



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

The reciprocal of the clock period is referred to as the **clock frequency**. The **clock width** is defined as the time during which the value of the clock signal is equal to 1. The ratio of the clock width and clock period is referred to as the duty cycle. A clock signal is said to

be **active high** if the state changes occur at the clock's rising edge or during the clock width. Otherwise, the clock is said to be **active low**. Synchronous sequential circuits are also known as **clocked sequential circuits**.

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

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

Fundamental Mode Circuits assumes that:

1. The input variables change only when the circuit is stable
2. Only one input variable can change at a given time
3. Inputs are levels are not pulses

A pulse mode circuit assumes that:

1. The input variables are pulses instead of levels
2. The width of the pulses is long enough for the circuit to respond to the input
3. The pulse width must not be so long that is still present after the new state is reached.

Latches and flip-flops

Latches and flip-flops are the basic elements for storing information. One latch or flip-flop can store one bit of information. The main difference between latches and flip-flops is that for latches, their outputs are constantly affected by their inputs as long as the enable signal is asserted. In other words, when they are enabled, their content changes immediately when their inputs change. Flip-flops, on the other hand, have their content change only either at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal. After the rising or falling edge of the clock, the flip-flop content remains constant even if the input changes.

There are basically four main types of latches and flip-flops: SR, D, JK, and T. The major differences in these flip-flop types are the number of inputs they have and how they change state. For each type, there are also different variations that enhance their operations. In this chapter, we

will look at the operations of the various latches and flip-flops.the flip-flops has two outputs, labeled Q and Q‘. the Q output is the normal output of the flip flop and Q‘ is the inverted output.

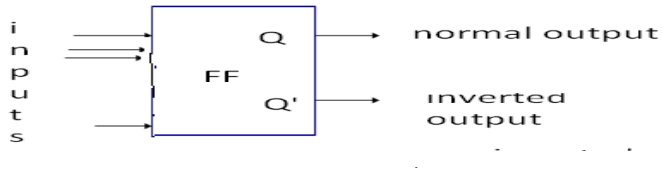


Figure: basic symbol of flipflop

A latch may be an active-high input latch or an active –LOW input latch.active –HIGH means that the SET and RESET inputs are normally resting in the low state and one of them will be pulsed high whenever we want to change latch outputs.

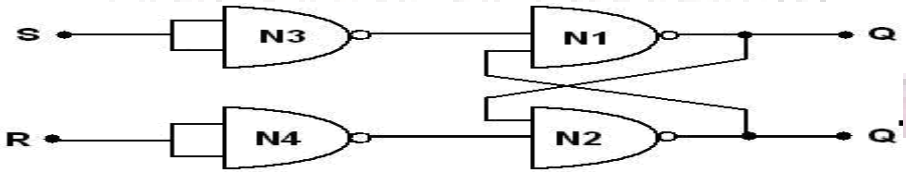
SR latch:

The latch has two outputs Q and Q‘. When the circuit is switched on the latch may enter into any state. If Q=1, then Q‘=0, which is called SET state. If Q=0, then Q‘=1, which is called RESET state. Whether the latch is in SET state or RESET state, it will continue to remain in the same state, as long as the power is not switched off. But the latch is not an useful circuit, since there is no way of entering the desired input. It is the fundamental building block in constructing flip-flops, as explained in the following sections

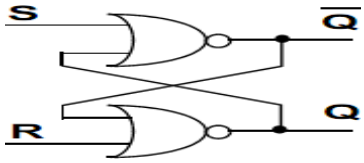
NAND latch

NAND latch is the fundamental building block in constructing a flip-flop. It has the property of holding on to any previous output, as long as it is not disturbed.

The opration of NAND latch is the reverse of the operation of NOR latch.if 0‘s are replaced by 1‘s and 1‘s are replaced by 0‘s we get the same truth table as that of the NOR latch shown



NOR latch



S	R	Q	$\overline{Q}$	Function
0	0	Q^+	$\overline{Q}^+$	Storage State
0	1	0	1	Reset
1	0	1	0	Set
1	1	0-?	0-?	Indeterminate State

The analysis of the operation of the active-HIGHNOR latch can be summarized as follows.

1. SET=0, RESET=0: this is normal resting state of the NOR latch and it has no effect on the output state. Q and Q‘ will remain in whatever stste they were prior to the occurrence of this input condition.
2. SET=1, RESET=0: this will always set Q=1, where it will remain even after SET returns to 0
3. SET=0, RESET=1: this will always reset Q=0, where it will remain even after RESET returns to 0
4. SET=1,RESET=1; this condition tries to SET and RESET the latch at the same time, and it produces Q=Q‘=0. If the inputs are returned to zero simultaneously, the resulting output stste is erratic and unpredictable. This input condition should not be used.

The SET and RESET inputs are normally in the LOW state and one of them will be pulsed HIGH. Whenever we want to change the latch outputs..

RS Flip-flop:

The basic flip-flop is a one bit memory cell that gives the fundamental idea of memory device. It constructed using two NAND gates. The two NAND gates N1 andN2 are connected such that, output of N1 is connected to input of N2 and output of N2 to input of N1. These form the feedback path the inputs are S and R, and outputs are Q and Q‘. The logic diagram and the block diagram of R-S flip-flop with clocked input

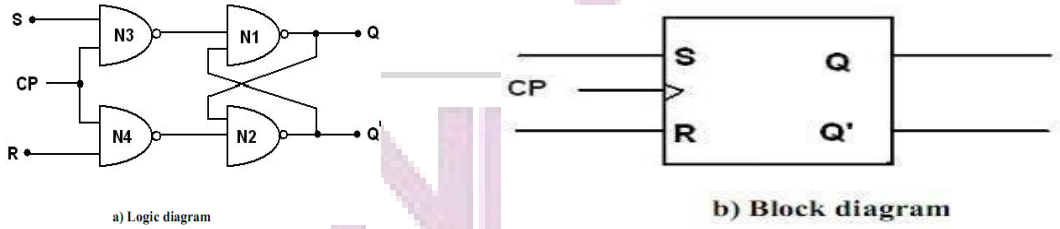


Figure: RS Flip-flop

The flip-flop can be made to respond only during the occurrence of clock pulse by adding two NAND gates to the input latch. So synchronization is achieved. i.e., flip-flops are allowed to change their states only at particular instant of time. The clock pulses are generated by a clock pulse generator. The flip-flops are affected only with the arrival of clock pulse**.**

Operation:

1. When CP=0 the output of N3 and N4 are 1 regardless of the value of S and R. This is given as input to N1 and N2. This makes the previous value of Q and Q‘unchanged.

2. When CP=1 the information at S and R inputs are allowed to reach the latch and change of state in flip-flop takes place.

3. CP=1, S=1, R=0 gives the SET state i.e., Q=1, Q‘=0.

4. CP=1, S=0, R=1 gives the RESET state i.e., Q=0, Q‘=1.

5. CP=1, S=0, R=0 does not affect the state of flip-flop.

6. CP=1, S=1, R=1 is not allowed, because it is not able to determine the next state. This condition is said to be a ―race condition‖.

In the logic symbol CP input is marked with a triangle. It indicates the circuit responds to an input change from 0 to 1. The characteristic table gives the operation conditions of flip-flop. Q(t) is the present state maintained in the flip-flop at time ‗t‘. Q(t+1) is the state after the occurrence of clock pulse.

Truth table

S	R	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	*	Not allowed

Edge triggered RS flip-flop:

Some flip-flops have an RC circuit at the input next to the clock pulse. By the design of the circuit the R-C time constant is much smaller than the width of the clock pulse. So the output changes will occur only at specific level of clock pulse. The capacitor gets fully charged when clock pulse goes from low to high. This change produces a narrow positive spike. Later at the trailing edge it produces narrow negative spike. This operation is called edge triggering, as the flip-flop responds only at the changing state of clock pulse. If output transition occurs at rising edge of clock pulse (0□1), it is called positively edge triggering. If it occurs at trailing edge (1□0) it is called negative edge triggering. Figure shows the logic and block diagram.

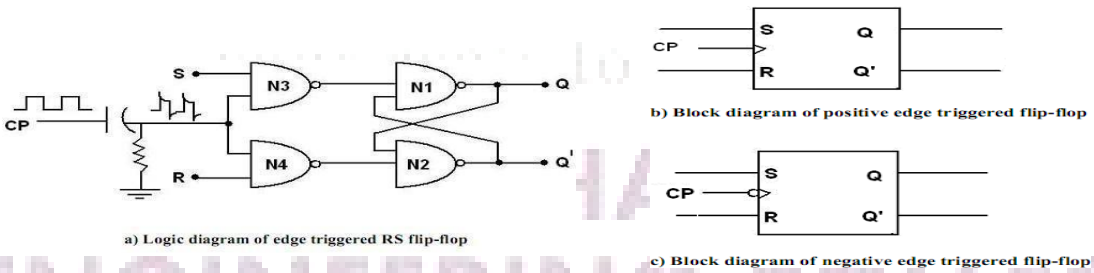
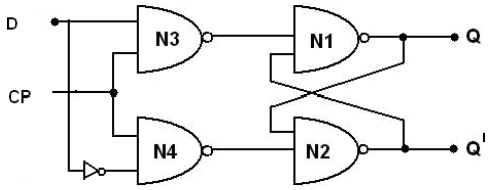


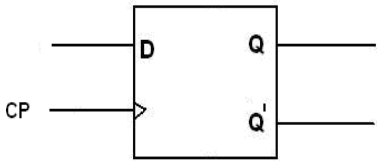
Figure: Edge triggered RS flip-flop

D flip-flop:

The D flip-flop is the modified form of R-S flip-flop. R-S flip-flop is converted to D flip-flop by adding an inverter between S and R and only one input D is taken instead of S and R. So one input is D and complement of D is given as another input. The logic diagram and the block diagram of D flip-flop with clocked input



a) Logic diagram



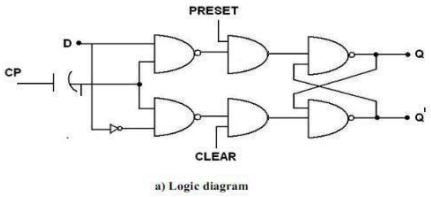
b) Block diagram

When the clock is low both the NAND gates (N1 and N2) are disabled and Q retains its last value. When clock is high both the gates are enabled and the input value at D is transferred to its output Q. D flip-flop is also called –Data flip-flop‖.

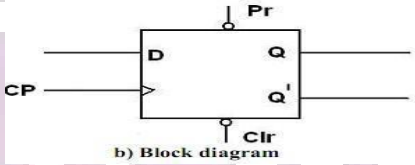
Truth table

CP	D	Q
0	x	Previous state
1	0	0
1	1	1

Edge Triggered D Flip-flop:



a) Logic diagram



b) Block diagram

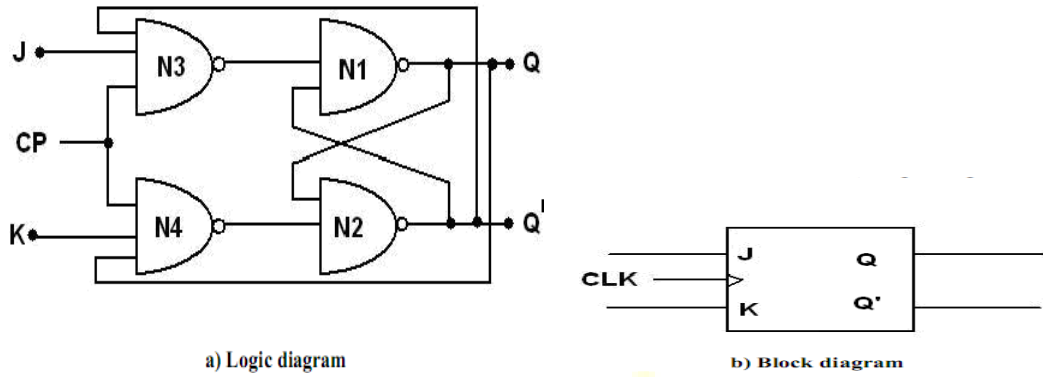
Truth table

PRESET	CLEAR	CP	D	Q
0	0	X	X	*(forbidden)
0	1	X	X	1
1	0	X	X	0
1	1	0	X	NC
1	1	1	X	NC
1	1	↓	X	NC
1	1	↑	0	0
1	1	↑	1	1

Figure: truth table, block diagram, logic diagram of edge triggered flip-flop

JK flip-flop (edge triggered JK flip-flop)

The race condition in RS flip-flop, when R=S=1 is eliminated in J-K flip-flop. There is a feedback from the output to the inputs. Figure 3.4 represents one way of building a JK flip-flop.



a) Logic diagram

b) Block diagram

Truth table

J	K	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	Q'_t	Complement/ toggle.

Figure: JK flip-flop

The J and K are called control inputs, because they determine what the flip-flop does when a positive clock edge arrives.

Operation:

1. When J=0, K=0 then both N3 and N4 will produce high output and the previous value of Q and Q‘ retained as it is.

2. When J=0, K=1, N3 will get an output as 1 and output of N4 depends on the value of Q. The final output is Q=0, Q‘=1 i.e., reset state

3. When J=1, K=0 the output of N4 is 1 and N3 depends on the value of Q‘. The final output is Q=1 and Q‘=0 i.e., set state

4. When J=1, K=1 it is possible to set (or) reset the flip-flop depending on the current state of output. If Q=1, Q‘=0 then N4 passes ‘0‘to N2 which produces Q‘=1, Q=0 which is reset state. When J=1, K=1, Q changes to the complement of the last state. The flip-flop is said to be in the toggle state.

The characteristic equation of the JK flip-flop is:

$$Q_{next} = J\overline{Q} + \overline{K}Q$$

JK flip-flop operation[28]								
Characteristic table				Excitation table				
J	**K**	**Q_{next}**	**Comment**	**Q**	**Q_{next}**	**J**	**K**	**Comment**
0	0	Q	hold state	0	0	0	X	No change
0	1	0	reset	0	1	1	X	Set
1	0	1	set	1	0	X	1	Reset
1	1	Q	toggle	1	1	X	0	No change

T flip-flop:

If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation

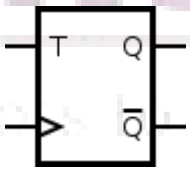


Figure : symbol for T flip flop

$Q_{next} = T \oplus Q = T\overline{Q} + \overline{T}Q$ (expanding the XOR operator

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or D flip-flop (T input and $P_{revious}$ is connected to the D input through an XOR gate).

T flip-flop operation[28]							
Characteristic table				**Excitation table**			
T	Q	Q_{next}	**Comment**	Q	Q_{next}	T	**Comment**
0	0	0	hold state (no clk)	0	0	0	No change
0	1	1	hold state (no clk)	1	1	0	No change
1	0	1	toggle	0	1	1	Complement
1	1	0	toggle	1	0	1	Complement

Flip flop operating characteristics:

The operation characteristics specify the performance, operating requirements, and operating limitations of the circuits. The operation characteristics mentions here apply to all flip-flops regardless of the particular form of the circuit.

Propagation Delay Time: is the interval of time required after an input signal has been applied for the resulting output change to occur.

Set-up Time: is the minimum interval required for the logic levels to be maintained constantly on the inputs (J and K, or S and R, or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

Hold Time: is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

Maximum Clock Frequency: is the highest rate that a flip-flop can be reliably triggered.

Power Dissipation: is the total power consumption of the device. It is equal to product of supply voltage (Vcc) and the current (Icc).

$P=V_{cc}.I_{cc}$

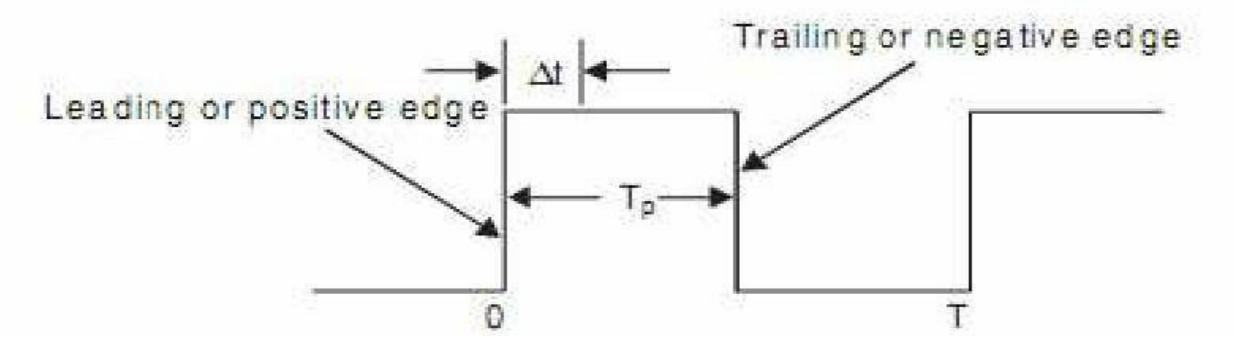
The power dissipation of a flip flop is usually in mW.

Pulse Widths: are the minimum pulse widths specified by the manufacturer for the Clock, SET and CLEAR inputs.

Clock transition times: for reliable triggering, the clock waveform transition times should be kept very short. If the clock signal takes too long to make the transitions from one level to other, the flip flop may either triggering erratically or not trigger at all.

Race around Condition

The inherent difficulty of an S-R flip-flop (i.e., S = R = 1) is eliminated by using the feedback connections from the outputs to the inputs of gate 1 and gate 2 as shown in Figure. Truth tables in figure were formed with the assumption that the inputs do not change during the clock pulse (CLK = 1). But the consideration is not true because of the feedback connections



- Consider, for example, that the inputs are J = K = 1 and Q = 1, and a pulse as shown in Figure is applied at the clock input.
- After a time interval t equal to the propagation delay through two NAND gates in series, the outputs will change to Q = 0. So now we have J = K = 1 and Q = 0.
- After another time interval of t the output will change back to Q = 1. Hence, we conclude that for the time duration of tP of the clock pulse, the output will oscillate between 0 and 1. Hence, at the end of the clock pulse, the value of the output is not certain. This situation is referred to as a race-around condition.
- Generally, the propagation delay of TTL gates is of the order of nanoseconds. So if the clock pulse is of the order of microseconds, then the output will change thousands of times within the clock pulse.
- This race-around condition can be avoided if tp< t < T. Due to the small propagation delay of the ICs it may be difficult to satisfy the above condition.
- A more practical way to avoid the problem is to use the master-slave (M-S) configuration as discussed below.

Applications of flip-flops:

Frequency Division: When a pulse waveform is applied to the clock input of a J-K flip-flop that is connected to toggle, the Q output is a square wave with half the frequency of the clock input. If more flip-flops are connected together as shown in the figure below, further division of the clock frequency can be achieved

. **Parallel data storage:** a group of flip-flops is called register. To store data of N bits, N flip-flops are required. Since the data is available in parallel form. When a clock pulse is applied to all flip-flops simultaneously, these bits will transfer will be transferred to the Q outputs of the flip flops.

Serial data storage: to store data of N bits available in serial form, N number of D-flip-flops is connected in cascade. The clock signal is connected to all the flip-flops. The serial data is applied to the D input terminal of the first flip-flop.

Transfer of data: data stored in flip-flops may be transferred out in a serial fashion, i.e., bit-by-bit from the output of one flip-flops or may be transferred out in parallel form.

Excitation Tables:

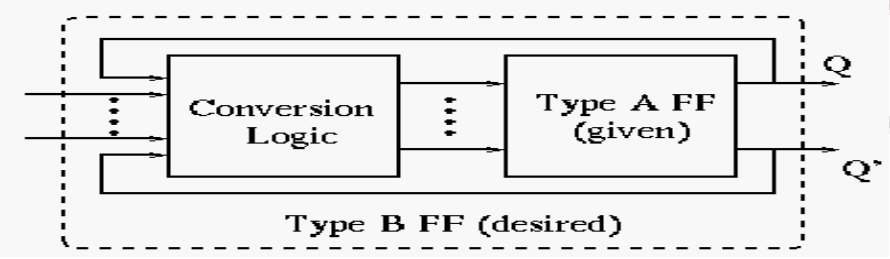
Previous State -> Present State	D
0 -> 0	0
0 -> 1	1
1 -> 0	0
1 -> 1	1

Previous State -> Present State	J	K
0 -> 0	0	X
0 -> 1	1	X
1 -> 0	X	1
1 -> 1	X	0

Previous State -> Present State	S	R
0 -> 0	0	X
0 -> 1	1	0
1 -> 0	0	1
1 -> 1	X	0

Previous State -> Present State	T
0 -> 0	0
0 -> 1	1
1 -> 0	1
1 -> 1	0

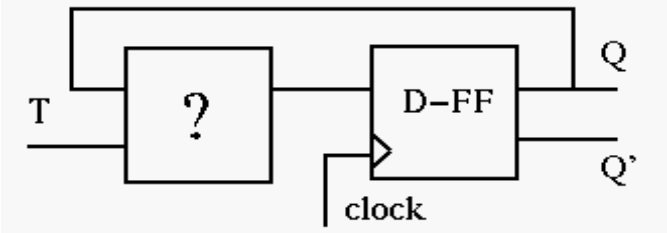
Conversions of flip-flops:



The key here is to use the excitation table, which shows the necessary triggering signal (S,R,J,K, D and T) for a desired flip-flop state transition :

Q_t	Q_{t+1}	S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

Convert a D-FF to a T-FF:

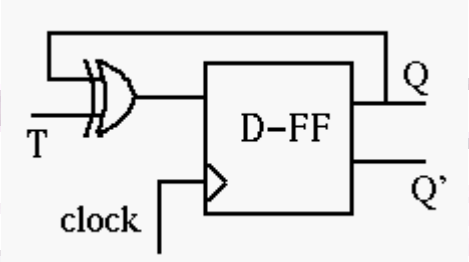


We need to design the circuit to generate the triggering signal D as a function of T and Q:
. Consider the excitation table:

$D = f(T, Q)$.

Q_t	Q_{t+1}	T	D
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

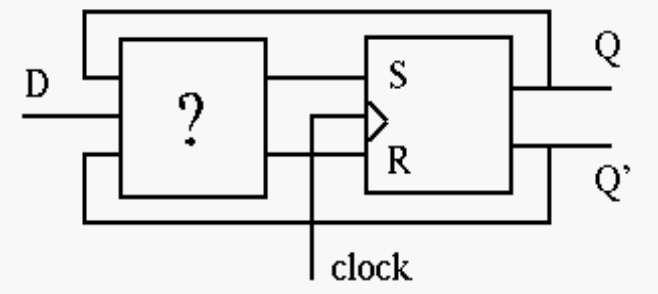
Treating as a function of and current FF state , we have



$D = T'Q + TQ' = T \oplus Q$

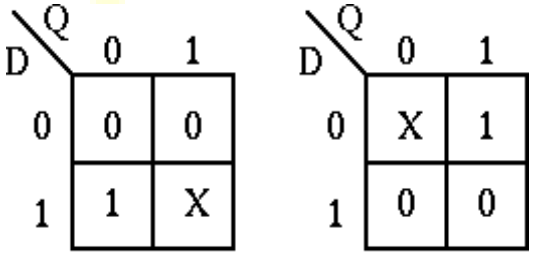
Convert a RS-FF to a D-FF:

We need to design the circuit to generate the triggering signals S and R as functions of and consider the excitation table:

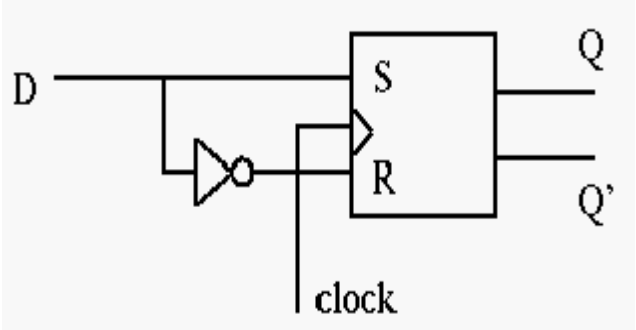


Q_t	Q_{t+1}	D	S	R
0	0	0	0	x
0	1	1	1	0
1	0	0	0	1
1	1	1	x	0

The desired signal and can be obtained as functions of and current FF state from the Karnaugh maps:



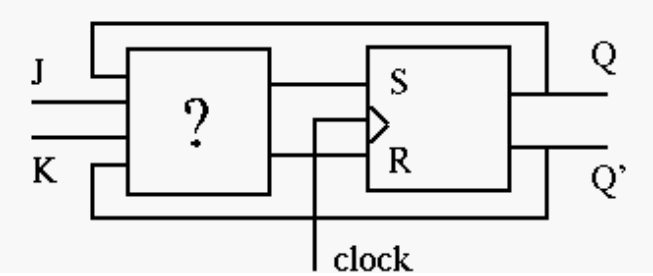
$S = D, \quad R = D'$



Convert a RS-FF to a JK-FF:

We need to design the circuit to generate the triggering signals S and R as functions of, J, K.

Consider the excitation table**:** The desired signal and as functions of, and current FF state can be obtained from the Karnaugh maps:



Q_t	Q_{t+1}	J	K	S	R
0	0	0	x	0	x
0	1	1	x	1	0
1	0	x	1	0	1
1	1	x	0	x	0

K-maps:

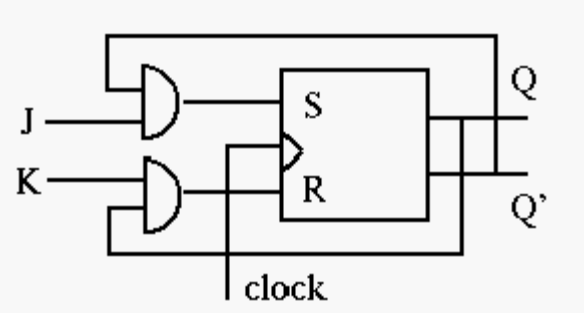
K \ QJ	00	01	11	10
0	0	1	X	X
1	0	1	0	0

S=Q'J

K \ QJ	00	01	11	10
0	X	0	0	0
1	X	0	1	1

R=QK

$S = Q'J, \quad R = QK$



The Master-Slave JK Flip-flop:

The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip-flop being connected to the two inputs of the "Slave" flip-flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip-flop as shown below.

The input signals J and K are connected to the gated "master" SR flip-flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip-flop is the inverse (complement) of the "master" clock input, the "slave" SR flip-flop does not toggle. The outputs from the "master" flip-flop are only "seen" by the gated "slave" flip-flop when the clock input goes "LOW" to logic level "0". When the clock is "LOW", the outputs from the "master" flip-flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip-flop now responds to the state of its inputs passed over by the "master" section. Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip-flop are fed through to the gated inputs of the "slave" flip-flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip-flop edge or pulse-triggered. Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal. In other words, the Master-Slave JK Flip-flop is a "Synchronous" device as it only passes data with the timing of the clock signal.

Sequential Circuit Design

- Steps in the design process for sequential circuits
- State Diagrams and State Tables
- Examples

- Steps in Design of a Sequential Circuit

1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
3. State Assignment: From a state table assign binary codes to the states.
4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
8. Verification: Use a HDL to verify the design.

Mealy and Moore

- Sequential machines are typically classified as either a Mealy machine or a Moore machine implementation.
- Moore machine: The outputs of the circuit depend only upon the current state ofthe circuit.
- Mealy machine: The outputs of the circuit depend upon both the current state ofthe circuit and the inputs.

An example to go through the steps

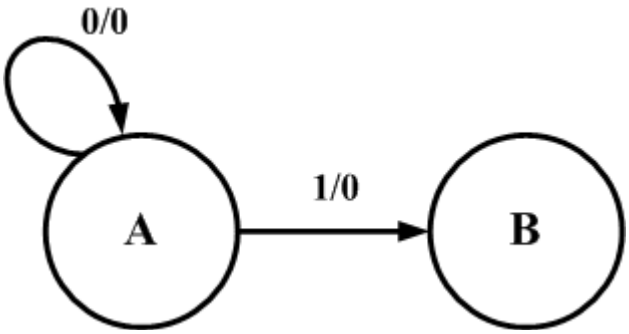
The specification: The circuit will have one input, X, and one output, Z. The output Z will be 0 except when the input sequence 1101 are the last 4 inputs received on X. In that case it will be a 1

Generation of a state diagram

- Create states and meaning for them.

State A – the last input was a 0 and previous inputs unknown. Can also be the reset state.
State B – the last input was a 1 and the previous input was a 0. The start of a new sequence possibly.

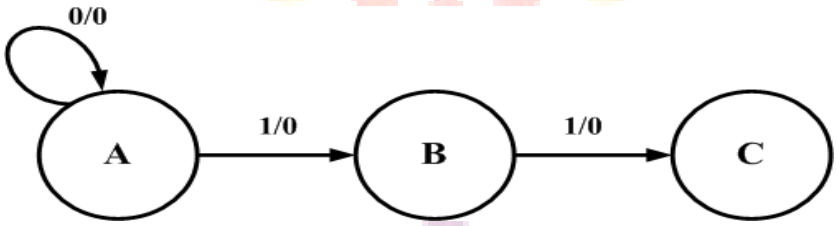
- Capture this in a state diagram



- Capture this in a state diagram

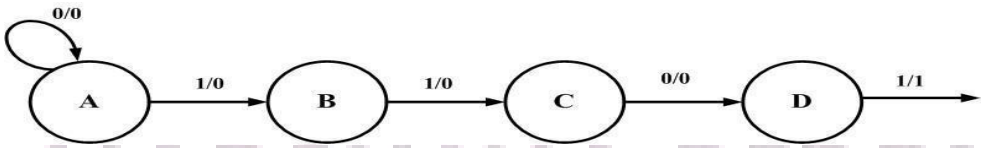
- Circles represent the states
- Lines and arcs represent the transition between states.
- The notation Input/output on the line or arc specifies the input that causes this transition and the output for this change of state.

- Add a state C – Have detected the input sequence 11 which is the start of the sequence



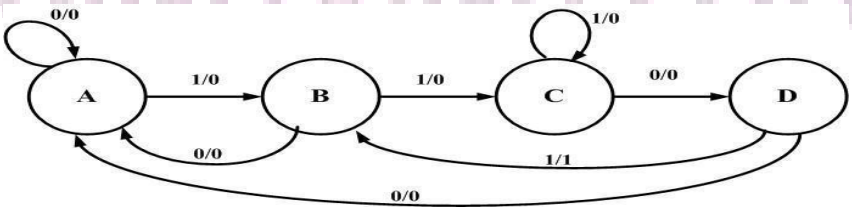
- Add a state D

  State D – have detected the 3rd input in the start of a sequence, a 0, now having 110. From State D, if the next input is a 1 the sequence has been detected and a 1 is output.



- The previous diagram was incomplete.

- In each state the next input could be a 0 or a 1. This must be included



- The state table
- This can be done directly from the state diagram

Prresent State	Next State X =0	Next State X=1	Output X=0	Output X=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

- Now need to do a state assignment

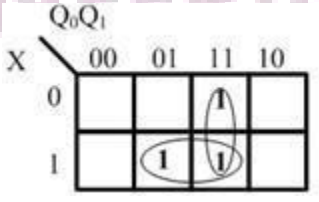
Select a state assignment

- Will select a gray encoding
- For this state A will be encoded 00, state B 01, state C 11 and state D 10

Prresent State	Next State X =0	Next State X=1	Output X=0	Output X=1
00	00	01	0	0
01	00	11	0	0
11	10	11	0	0
10	00	01	0	1

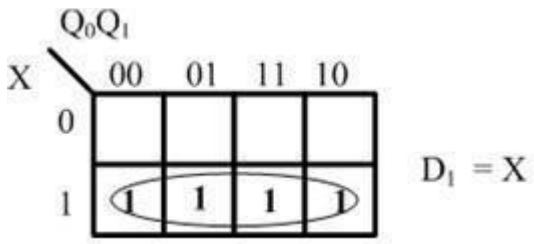
Flip-flop input equations

- Generate the equations for the flip-flop inputs
- Generate the D_0 equation



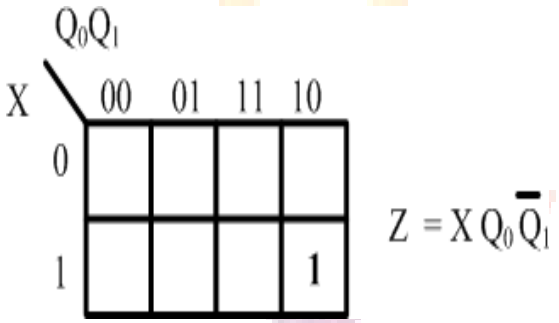
$D_0 = Q_0 Q_1 + X Q_1$

- Generate the D_1 equation



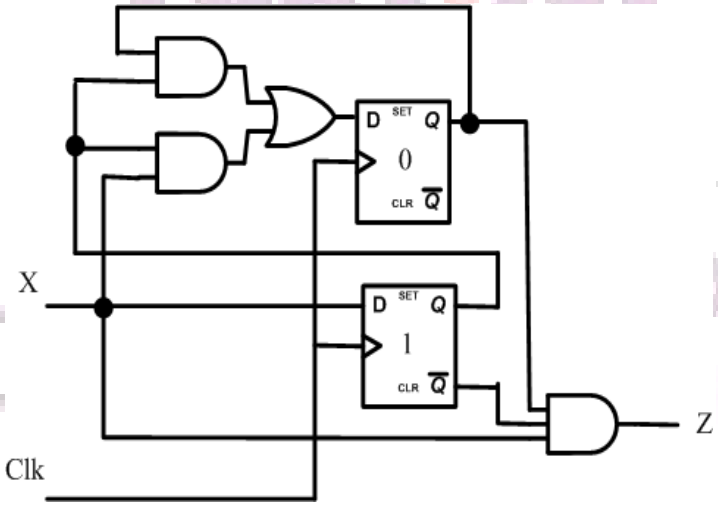
The output equation

- The next step is to generate the equation for the output Z and what is needed to generate it.
- Create a K-map from the truth table.



Now map to a circuit

- The circuit has 2 D type F/Fs



Shift registers:

In digital circuits, a **shift register** is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, *shifting in* the data present at its input and *shifting out* the last bit in the array, at each transition of the clock input. More generally, a **shift register** may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as **serial-in, parallel-out** (SIPO) or as **parallel-in, serial-out** (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also **bi-directional** shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a **circular shift register**

Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

Buffer register:

The buffer register is the simple set of registers. It is simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.

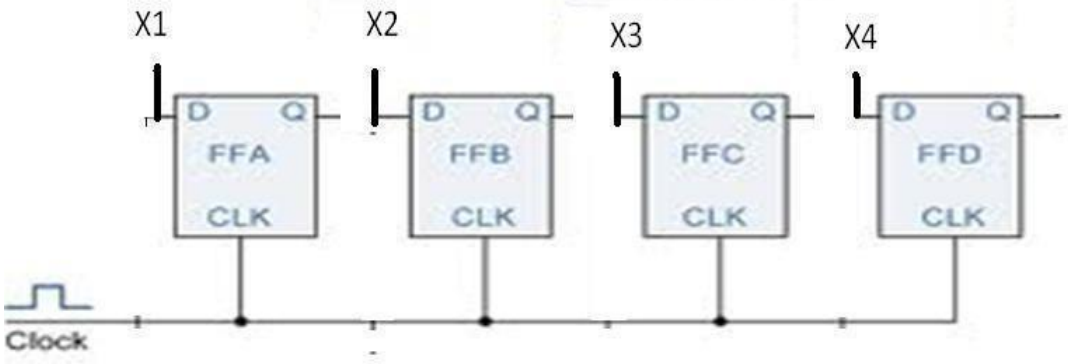


Figure: logic diagram of 4-bit buffer register

The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes:

$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$

$Q=X$

Controlled buffer register:

If CLRgoes LOW, all the FFs are RESET and the output becomes, Q=0000.

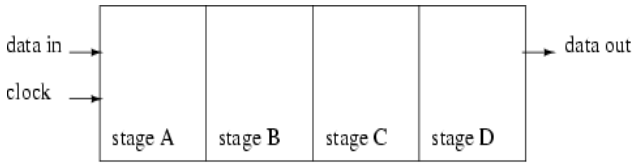
When CLR is HIGH, the register is ready for action. LOAD is the control input. When LOAD is HIGH, the data bits X can reach the D inputs of FF‘s.

$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$

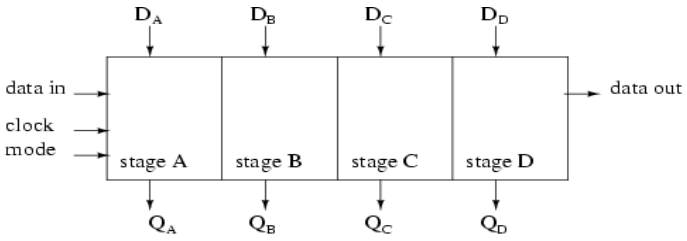
$Q=X$

When load is low, the X bits cannot reach the FF‘s.

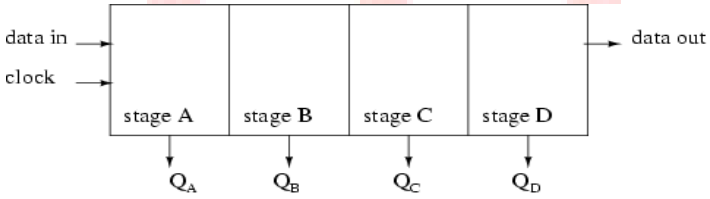
Data transmission in shift registers:



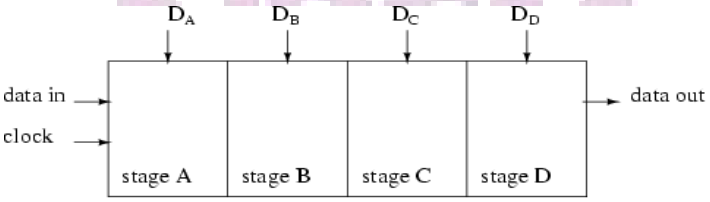
Serial-in, serial-out shift register with 4-stages



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



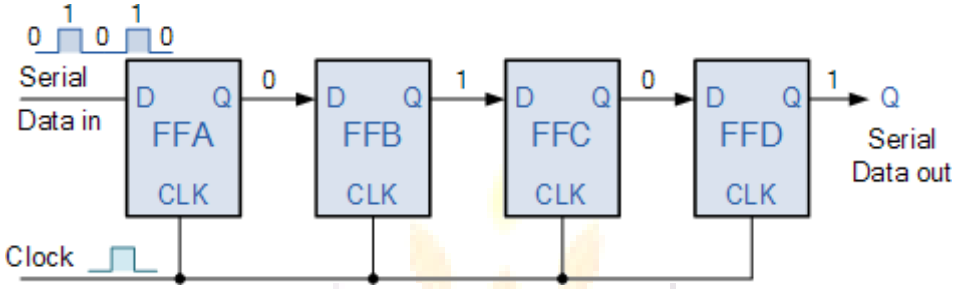
Parallel-in, serial-out shift register with 4-stages

A number of ff`s connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.

1. Serial in, serial out, shift right, shift registers
2. Serial in, serial out, shift left, shift registers
3. Parallel in, serial out shift registers
4. Parallel in, parallel out shift registers

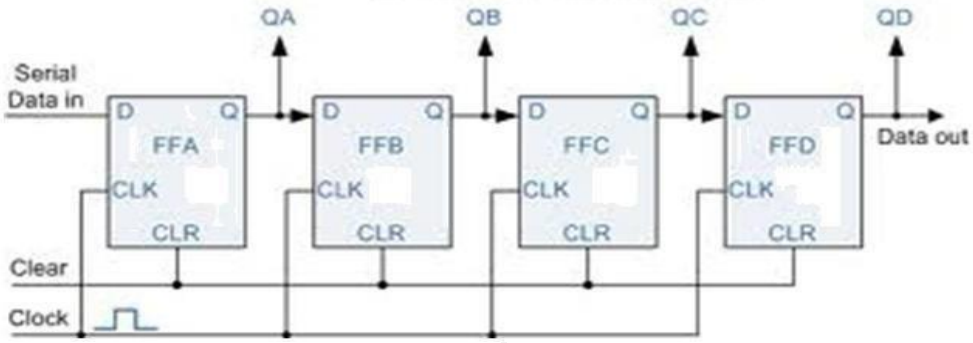
Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.



When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

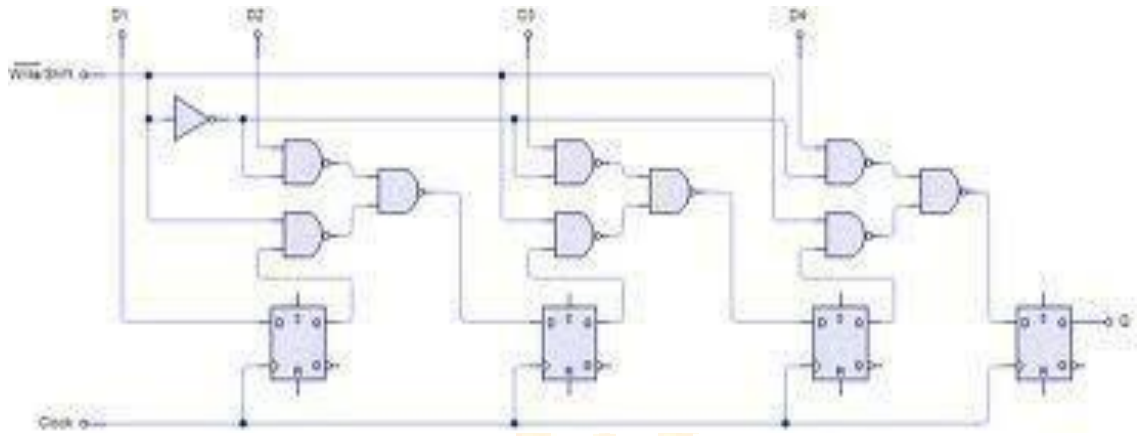
Serial-in, parallel-out, shift register:



In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.

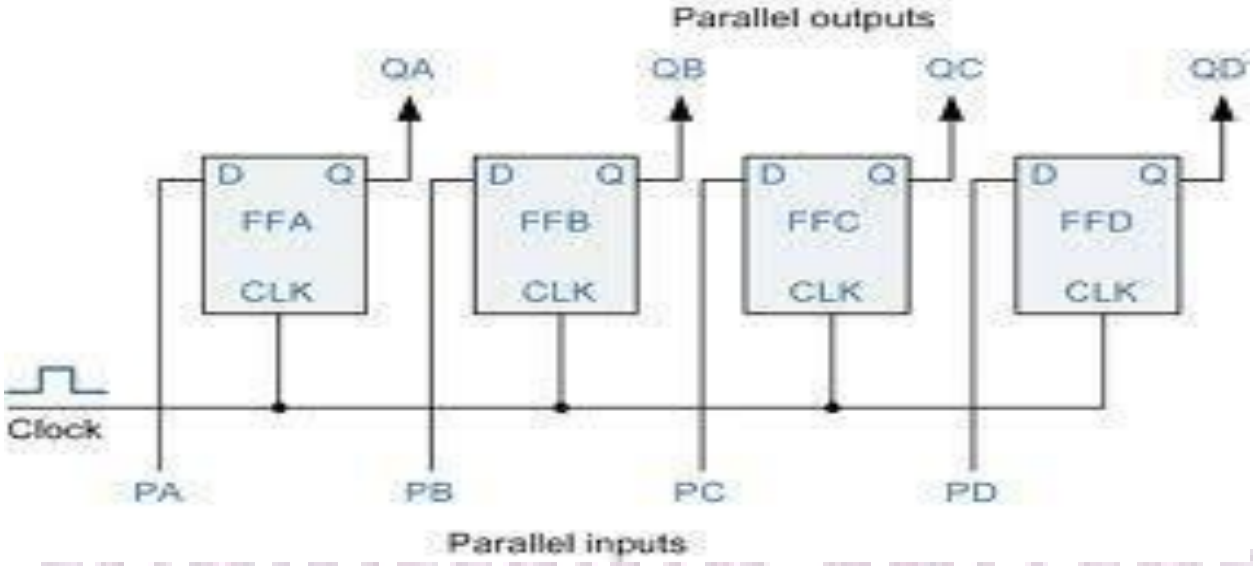
Parallel-in, serial-out, shift register:



For a parallel-in, serial out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.

There are four data lines A,B,C,D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminalQ4

Parallel-in, parallel-out, shift register



In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF‘s. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional shift register:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register.the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.

A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5,G6,G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register

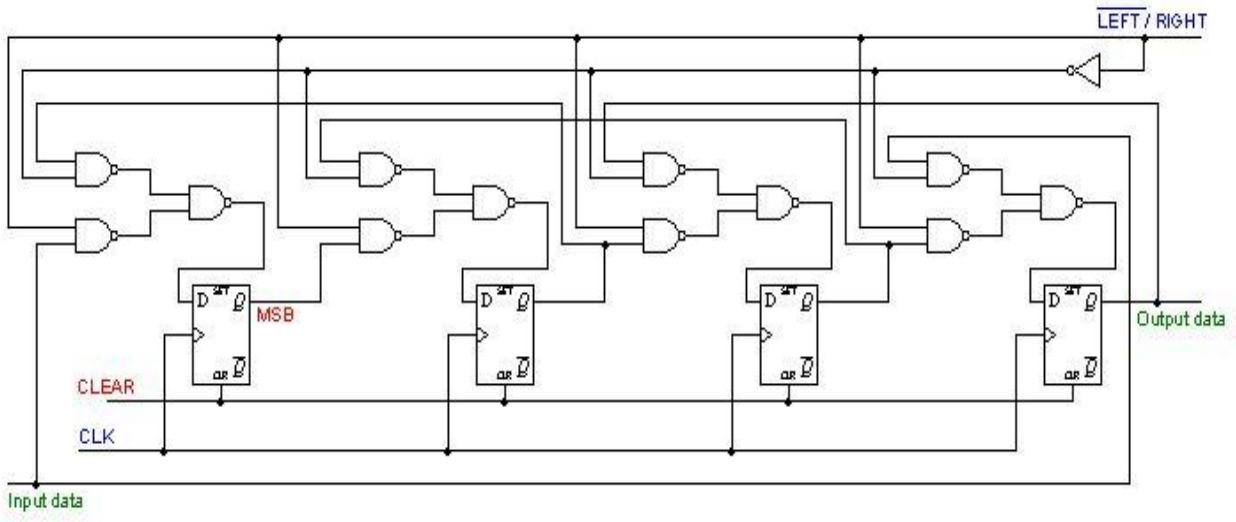


Figure: logic diagram of a 4-bit bidirectional shift register

Universal shift register:

A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form.

The most general shift register has the following capabilities.

1. A clear control to clear the register to 0
2. A clock input to synchronize the operations
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right

4. A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left
5. A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer
6. N parallel output lines
7. A control state that leaves the information in the register unchanged in the presence of the clock.

A universal shift register can be realized using multiplexers. The below fig shows the logic diagram of a 4-bit universal shift register that has all capabilities. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0. Input 0 in each multiplexer is selected when S1S0=00, input 1 is selected when S1S0=01 and input 2 is selected when S1S0=10 and input 4 is selected when S1S0=11. The selection inputs control the mode of operation of the register according to the functions entries. When S1S0=0, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S1S0=01, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flopA4. When S1S0=10, a shift left operation results with the other serial input going into flip-flop A1. Finally when S1S0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle

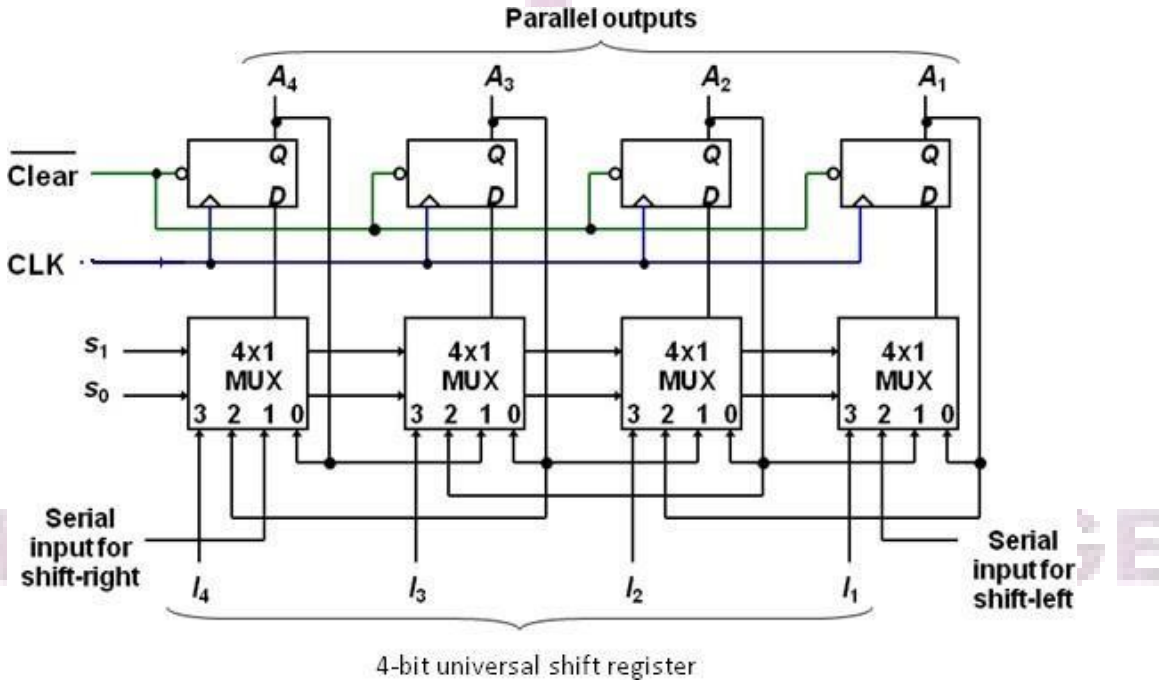


Figure: logic diagram 4-bit universal shift register

Function table for theregister

mode control		
S0	**S1**	**register operation**
0	0	No change
0	1	Shift Right
1	0	Shift left
1	1	Parallel load

Counters:

Counter is a device which stores (and sometimes displays) the number of times particular event or process has occurred, often in relationship to a clock signal. A Digital counter is a set of flip flops whose state change in response to pulses applied at the input to the counter. Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters

In electronics counters can be implemented quite easily using register-type circuits such as the flip-flops and a wide variety of classifications exist:

- Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops
- Synchronous counter – all state bits change under control of a singleclock
- Decade counter – counts through ten states per stage
- Up/down counter – counts both up and down, under command of a control input
- Ring counter – formed by a shift register with feedback connection in a ring
- Johnson counter – a *twisted* ring counter
- Cascaded counter
- Modulus counter.

Each is useful for different applications. Usually, counter circuits are digital in nature, and count in natural binary Many types of counter circuits are available as digital building blocks, for example a number of chips in the 4000 series implement different counters.

Occasionally there are advantages to using a counting sequence other than the natural binary sequence such as the binary coded decimal counter, a linear feed-back shift register counter, or a gray-code counter.

Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.

Asynchronous counters:

An asynchronous (ripple) counter is a single JK-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), one will get another 1 bit counter that counts half as fast. Putting them together yields a two-bit counter:

Two-bit ripple up-counter using negative edge triggered flip flop:

Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.

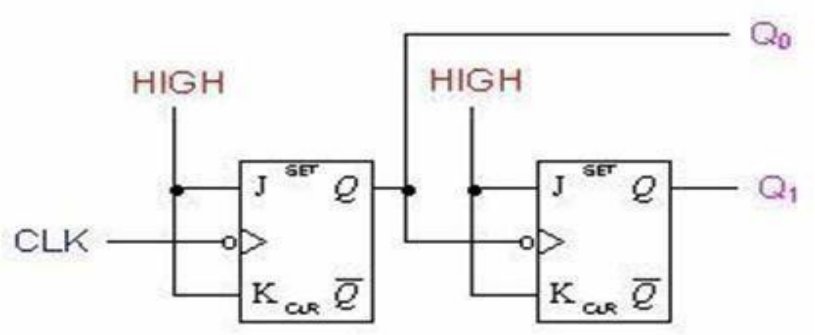
· The counter is initially assumed to be at a state 00 where the outputs of the tow flip-flops are noted as Q_1Q_0. Where Q_1 forms the MSB and Q_0 forms the LSB.

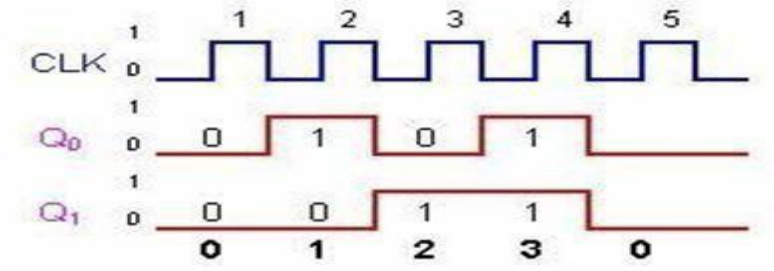
· For the negative edge of the first clock pulse, output of the first flip-flop FF_1 toggles its state. Thus Q_1 remains at 0 and Q_0 toggles to 1 and the counter state are now read as 01.

· During the next negative edge of the input clock pulse FF_1 toggles and $Q_0 = 0$. The output Q0 being a clock signal for the second flip-flop FF_2 and the present transition acts as a negative edge for FF_2 thus toggles its state $Q_1 = 1$. The counter state is now read as 10.

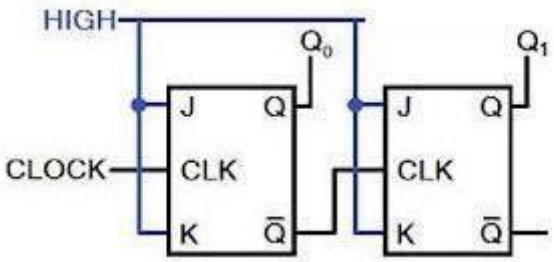
· For the next negative edge of the input clock to FF_1 output Q0 toggles to 1. But this transition from 0 to 1 being a positive edge for FF_2 output Q_1 remains at 1. The counter state is now read as 11.

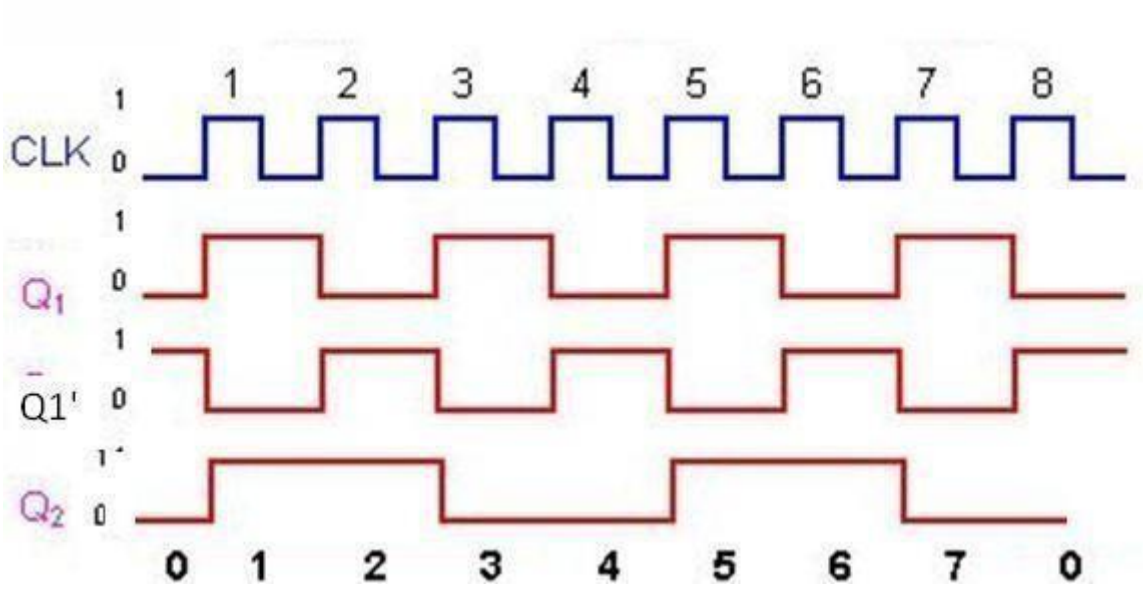
· For the next negative edge of the input clock, Q_0 toggles to 0. This transition from 1 to 0 acts as a negative edge clock for FF_2 and its output Q_1 toggles to 0. Thus the starting state 00 is attained. Figure shown below





Two-bit ripple down-counter using negative edge triggered flip flop:





A 2-bit down-counter counts in the order 0,3,2,1,0,1…….,i.e, 00,11,10,01,00,11 …..,etc. the above fig. shows ripple down counter, using negative edge triggered J-K FFs and its timing diagram.

- For down counting, Q1‘ of FF1 is connected to the clock of Ff2. Let initially all the FF1 toggles, so, Q1 goes from a 0 to a 1 and Q1‘ goes from a 1 to a 0.

- The negative-going signal at Q1‘ is applied to the clock input of FF2, toggles Ff2 and, therefore, Q2 goes from a 0 to a 1.so, after one clock pulse Q2=1 and Q1=1, I.e., the state of the counter is 11.
- At the negative-going edge of the second clock pulse, Q1 changes from a 1 to a 0 and Q1‘ from a 0 to a 1.
- This positive-going signal at Q1‘ does not affect FF2 and, therefore, Q2 remains at a 1. Hence , the state of the counter after second clock pulse is 10
- At the negative going edge of the third clock pulse, FF1 toggles. So Q1, goes from a 0 to a 1 and Q1‘ from 1 to 0. This negative going signal at Q1‘ toggles FF2 and, so, Q2 changes from 1 to 0, hence, the state of the counter after the third clock pulse is 01.
- At the negative going edge of the fourth clock pulse, FF1 toggles. So Q1, goes from a 1 to a 0 and Q1‘ from 0 to 1. . This positive going signal at Q1‘ does not affect FF2 and, so, Q2 remains at 0, hence, the state of the counter after the fourth clock pulse is 00.

Two-bit ripple up-down counter using negative edge triggered flip flop:

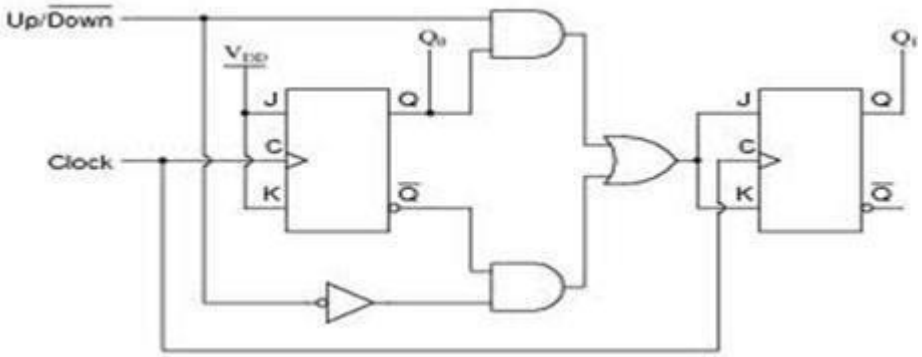


Figure: asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop:

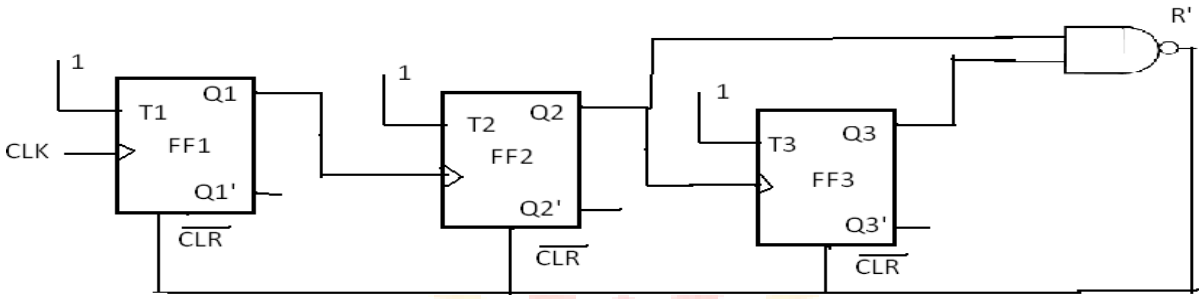
- As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1 is transmitted to clock of FF2 and when M=0 for down counting, Q1‘ is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.
- Clock signal to FF2= (Q1.Up)+(Q1‘. Down)= Q1m+Q1‘M‘

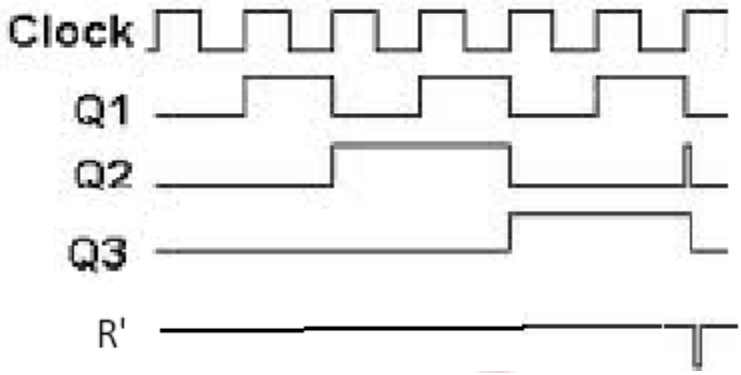
Design of Asynchronous counters:

To design a asynchronous counter, first we write the sequence , then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R and R‘ using K-Map or any other method. Provide a feedback such that R and R‘ resets all the FF‘s after the desired count

Design of a Mod-6 asynchronous counter using T FFs:

A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. it is –divide by-6-counter‖, in the sense that it divides the input clock frequency by 6.it requires three FFs, because the smallest value of n satisfying the conditionN≤2^n is n=3; three FFs can have 8 possible states, out of which only six are utilized and the remaining two states 110and 111, are invalid. If initially the counter is in 000 state, then after the sixth clock pulse, it goes to 001, after the second clock pulse, it goes to 010, and so on.





After sixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q3, Q2 and Q1 as the variables, and reset R as the output and obtain an expression for R in terms of Q3, Q2, and Q1that decides the feedback into be provided. From the truth table, R=Q3Q2. For active-low Reset, R‘ is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.

R=0 for 000 to 101, R=1 for 110, and R=X=for111

Therefore,

R=Q3Q2Q1‘+Q3Q2Q1=Q3Q2

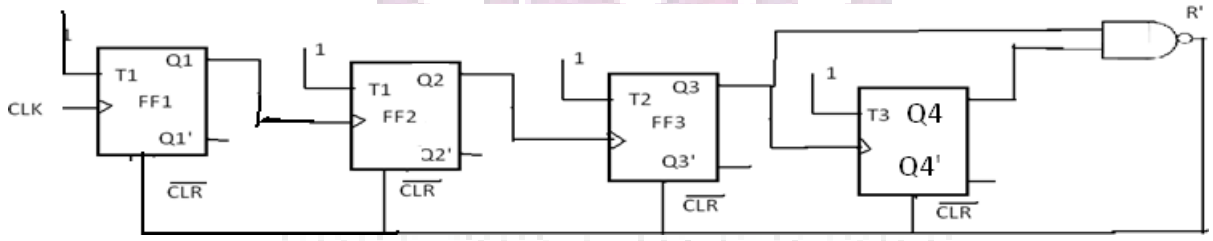
The logic diagram and timing diagram of Mod-6 counter is shown in the above fig.

The truth table is as shown in below.

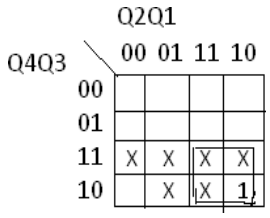
After pulses	States Q3	Q2	Q1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	0	0

Design of a mod-10 asynchronous counter using T-flip-flops:

A mod-10 counter is a decade counter. It also called a BCD counter or a divide-by-10 counter. It requires four flip-flops (condition $10 \leq 2^n$ is n=4). So, there are 16 possible states, out of which ten are valid and remaining six are invalid. The counter has ten stable state, 0000 through 1001, i.e., it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of Q2. The state 1010 is a temporary state for which the reset signal R=1, R=0 for 0000 to 1001, and R=C for 1011 to 1111.



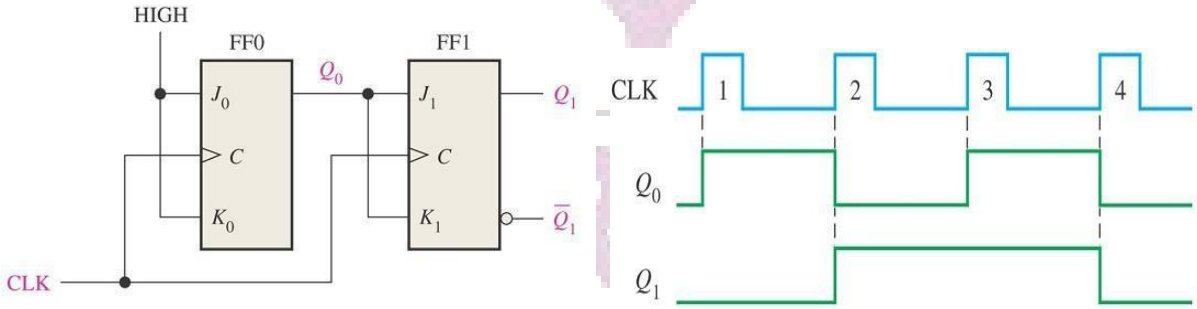
The count table and the K-Map for reset are shown in fig. from the K-Map R=Q4Q2. So, feedback is provided from second and fourth FFs. For active –HIGH reset, Q4Q2 is applied to the clear terminal. For active-LOW reset $Q4Q2$ is connected CLR isof all Flip=flops.



After pulses	Count Q4	Q3	Q2	Q1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	0	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	0	1	0	1
10	0	0	0	0

Synchronous counters:

Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. if the clock frequency is very high, the asynchronous counter may skip some of the states. This problem is overcome in synchronous counters or parallel counters. Synchronous counters are counters in which all the flip flops are triggered simultaneously by the clock pulses Synchronous counters have a common clock pulse applied simultaneously to all flip-flops. A 2-Bit Synchronous Binary Counter



Design of synchronous counters:

For a systematic design of synchronous counters. The following procedure is used.

Step 1: State Diagram: draw the state diagram showing all the possible states state diagram which also be called nth transition diagrams, is a graphical means of depicting the sequence of states through which the counter progresses.

Step2: number of flip-flops: based on the description of the problem, determine the required number n of the flip-flops- the smallest value of n is such that the number of states $N \leq 2^n$--- and the desired counting sequence.

Step3: choice of flip-flops excitation table: select the type of flip-flop to be used and write the excitation table. An excitation table is a table that lists the present state (ps) , the next state(ns) and required excitations.

Step4: minimal expressions for excitations: obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step5: logic diagram: draw a logic diagram based on the minimal expressions

Design of a synchronous 3-bit up-down counter using JK flip-flops:

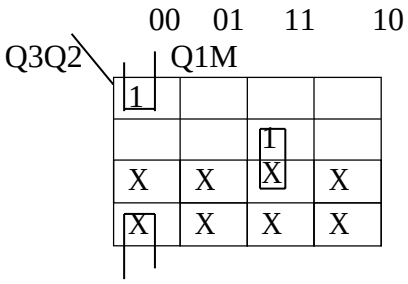
Step1: determine the number of flip-flops required. A 3-bit counter requires three FFs. It has 8 states (000,001,010,011,101,110,111) and all the states are valid. Hence no don‘t cares. For selecting up and down modes, a control or mode signal M is required. When the mode signal M=1 and counts down when M=0. The clock signal is applied to all the FFs simultaneously.

Step2: draw the state diagrams: the state diagram of the 3-bit up-down counter is drawn as

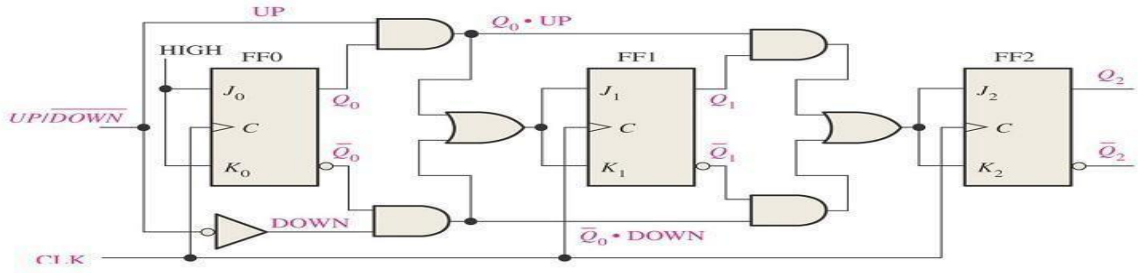
Step3: select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in fig.

PS			mode	NS			required excitations					
Q3	Q2	Q1	M	Q3	Q2	Q1	J3	K3	J2	K2	J1	K1
0	0	0	0	1	1	1	1	x	1	x	1	x
0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	0	0	0	x	0	x	x	1
0	0	1	1	0	1	0	0	x	1	x	x	1
0	1	0	0	0	0	1	0	x	x	1	1	x
0	1	0	1	0	1	1	0	x	x	0	1	x
0	1	1	0	0	1	0	0	x	x	0	x	1
0	1	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	0	1	1	x	1	1	x	1	x
1	0	0	1	1	0	1	x	0	0	x	1	x
1	0	1	0	1	0	0	x	0	0	x	x	1
1	0	1	1	1	1	0	x	0	1	x	x	1
1	1	0	0	1	0	1	x	0	x	1	1	x
1	1	0	1	1	1	1	x	0	x	0	1	x
1	1	1	0	1	1	0	x	0	x	0	x	1
1	1	1	1	0	0	0	x	1	x	1	x	1

Step4: obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3,J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in fig.



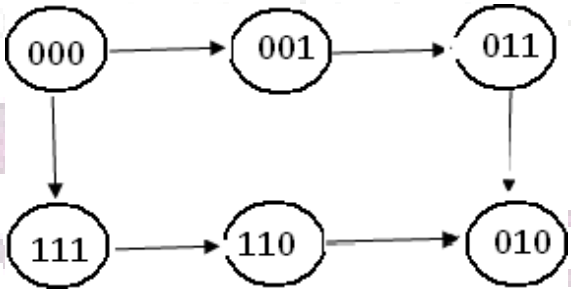
Step5: draw the logic diagram: a logic diagram using those minimal expressions can be drawn as shown in fig.



Design of a synchronous modulo-6 gray cod counter:

Step 1: the number of flip-flops: we know that the counting sequence for a modulo-6 gray code counter is 000, 001, 011, 010, 110, and 111. It requires n=3FFs ($N \leq 2^n$, i.e., $6 \leq 2^3$). 3 FFs can have 8 states. So the remaining two states 101 and 100 are invalid. The entries for excitation corresponding to invalid states are don‘t cares.

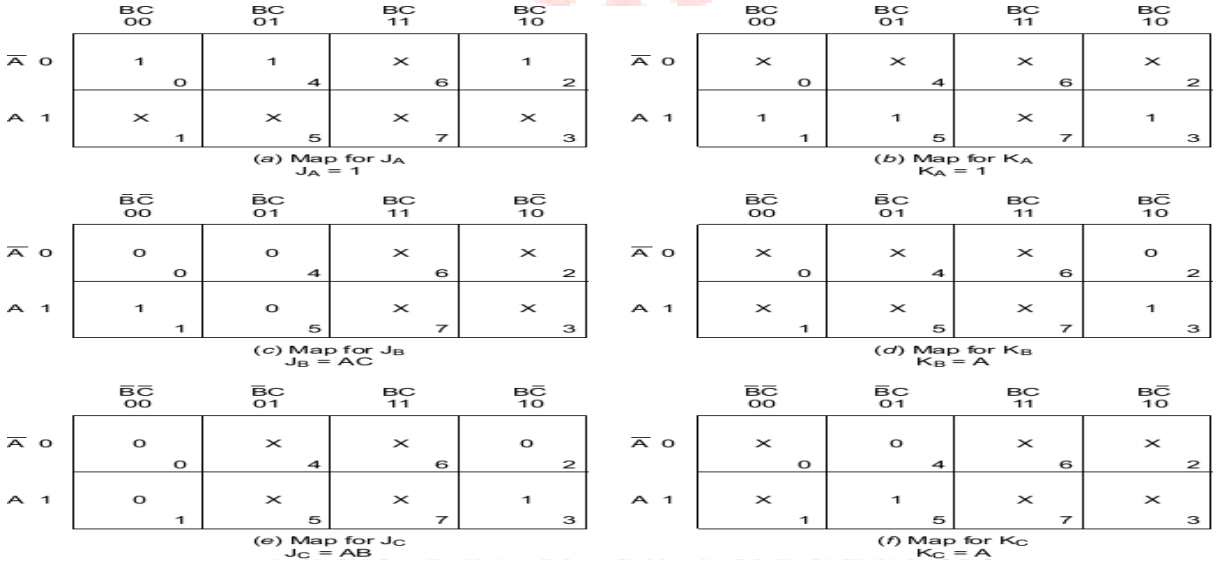
Step2: the state diagram: the state diagram of the mod-6 gray code converter is drawn as shown in fig.



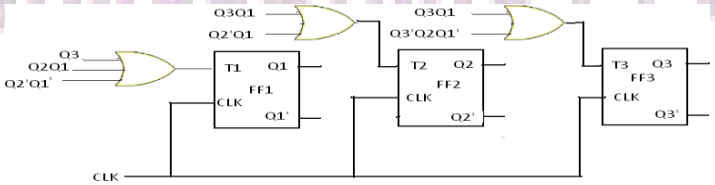
Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.

PS			NS			required excitations		
Q3	**Q2**	**Q1**	**Q3**	**Q2**	**Q1**	**T3**	**T2**	**T1**
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Step4: The minimal expressions: the K-maps for excitations of FFs T3,T2,and T1 in terms of outputs of FFs Q3,Q2, and Q1, their minimization and the minimal expressions for excitations obtained from them are shown if fig



Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.



Design of a synchronous BCD Up-Down counter using FFs:

Step1: the number of flip-flops: a BCD counter is a mod-10 counter has 10 states (0000 through 1001) and so it requires n=4FFs($N \leq 2^n$,, i.e., $10 \leq 2^4$). 4 FFS can have 16 states. So out of 16 states, six states (1010 through 1111) are invalid. For selecting up and down mode, a control or mode signal M is required. , it counts up when M=1 and counts down when M=0. The clock signal is applied to all FFs.

Step2: the state diagram: The state diagram of the mod-10 up-down counter is drawn as shown in fig.

Step3: types of flip-flops and excitation table: T flip-flops are selected and the excitation table of the modulo-10 up down counter using T flip-flops is drawn as shown in fig.

The remaining minterms are don't cares($\sum$d(20,21,22,23,24,25,26,37,28,29,30,31)) from the excitation table we can see that T1=1 and the expression for T4,T3,T2 are asfollows.

T4=$\sum$m(0,15,16,19)+d(20,21,22,23,24,25,26,27,28,29,30,31)

T3=$\sum$m(7,15,16,8)+d(20,21,22,23,24,25,26,27,28,29,30,31)

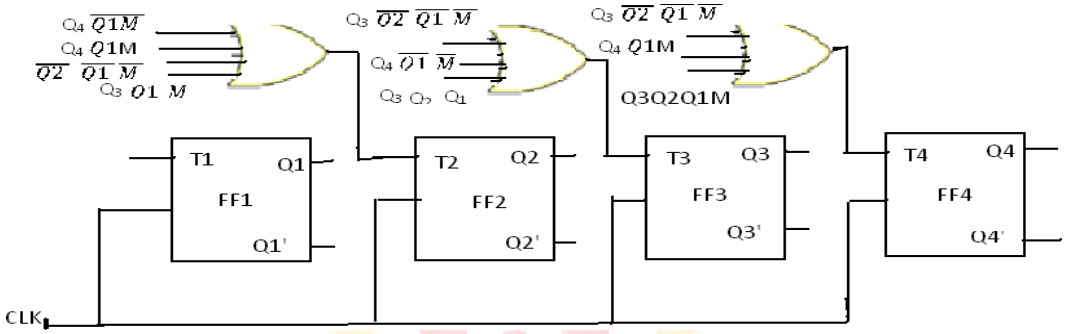
T2=$\sum$m(3,4,7,8,11,12,15,16)+d(20,21,22,23,24,25,26,27,28,29,30,31)

PS				**mode**	**NS**				**required excitations**			
Q4	**Q3**	**Q2**	**Q1**	**M**	**Q4**	**Q3**	**Q2**	**Q1**	**T4**	**T3**	**T2**	**T1**
0	0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	0	0	0	0	1
0	1	0	1	1	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	1	0	0	1

Step4: The minimal expression: since there are 4 state variables and a mode signal, we require 5 variable kmaps. 20 conditions of Q4Q3Q2Q1M are valid and the remaining 12 combinations are invalid. So the entries for excitations corresponding to those invalid combinations are don‘t cares. Minimizing K-maps for T2 we get

T 2= Q4Q1‘M+Q4‘Q1M+Q2Q1‘M‘+Q3Q1‘M‘

Step5: the logic diagram: the logic diagram based on the above equation is shown in fig.



Shift register counters:

One of the applications of shift register is that they can be arranged to form several types of counters. The most widely used shift register counter is ring counter as well as the twisted ring counter.

Ring counter: this is the simplest shift register counter. The basic ring counter using D flip-flops is shown in fig. the realization of this counter using JK FFs. The Q output of each stage is connected to the D flip-flop connected back to the ring counter.

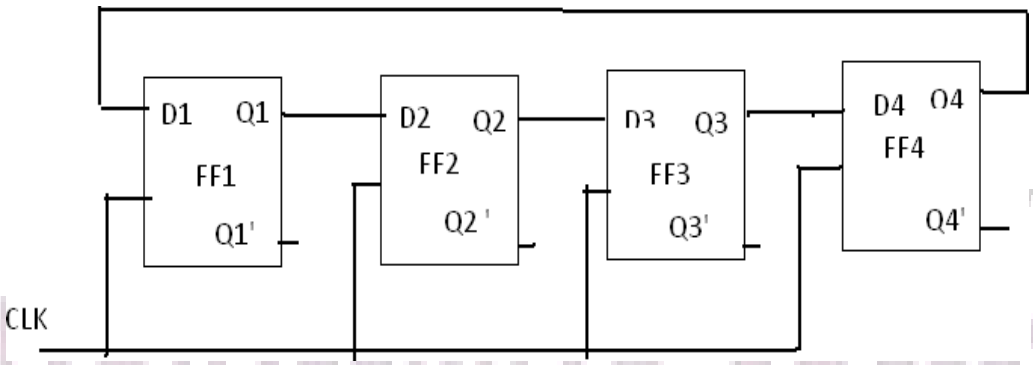
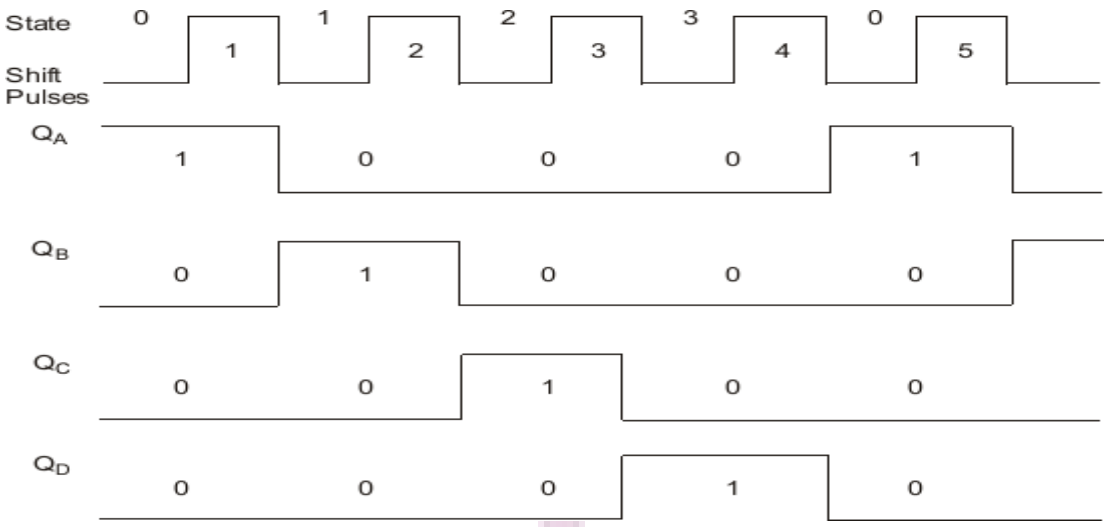


FIGURE: logic diagram of 4-bit ring counter using D flip-flops

Only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially the first FF is present to a 1. So, the initial state is 1000, i.e., Q1=1, Q2=0,Q3=0,Q4=0. After each clock pulse, the contents of the register are shifted to the right by one bit and Q4 is shifted back to Q1. The sequence repeats after four clock pulses. The number

of distinct states in the ring counter, i.e., the mod of the ring counter is equal to number of FFs used in the counter. An n-bit ring counter can count only n bits, where as n-bit ripple counter can count 2^n bits. So, the ring counter is uneconomical compared to a ripple counter but has advantage of requiring no decoder, since we can read the count by simply noting which FF is set. Since it is entirely a synchronous operation and requires no gates external FFs, it has the further advantage of being very fast.

Timing diagram:



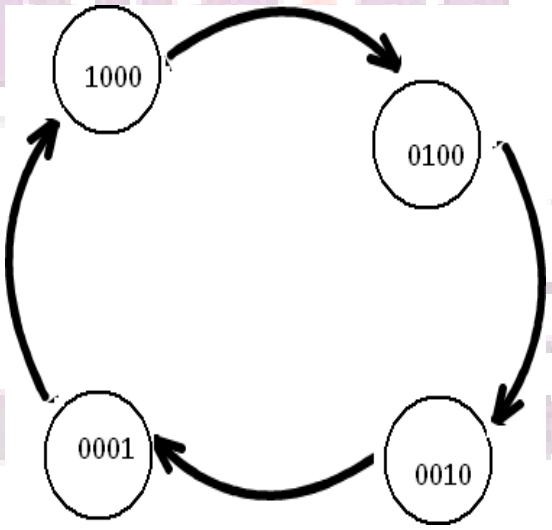


Figure: state diagram

Twisted Ring counter (Johnson counter):

This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. the Q output of each is connected to the D input of the next stage, but the Q‘ output of the last stage is connected to the D input of the first stage, therefore, the name twisted ring counter. This feedback arrangement produces a unique sequence of states.

The logic diagram of a 4-bit Johnson counter using D FF is shown in fig. the realization of the same using J-K FFs is shown in fig.. The state diagram and the sequence table are shown in figure. The timing diagram of a Johnson counter is shown in figure.

Let initially all the FFs be reset, i.e., the state of the counter be 0000. After each clock pulse, the level of Q1 is shifted to Q2, the level of Q2to Q3, Q3 to Q4 and the level of Q4‘to Q1 and the sequences given in fig.

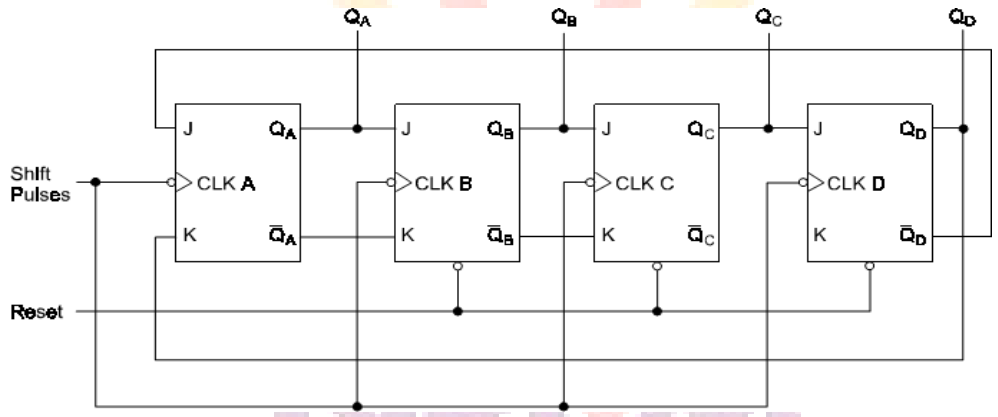


Figure: Johnson counter with JK flip-flops

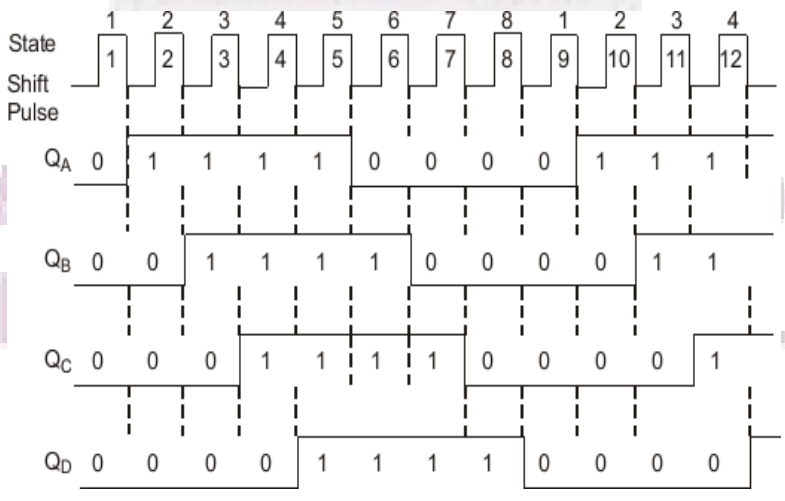
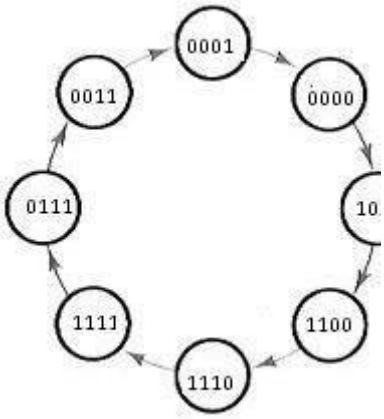


Figure: timing diagram

State diagram:



Q1	Q2	Q3	Q4	after clock pulse
0	0	0	0	0
1	0	0	0	1
1	1	0	0	2
1	1	1	0	3
1	1	1	1	4
0	1	1	1	5
0	0	1	1	6
0	0	0	1	7
0	0	0	0	8
1	0	0	0	9

Excitation table

Synthesis of sequential circuits:

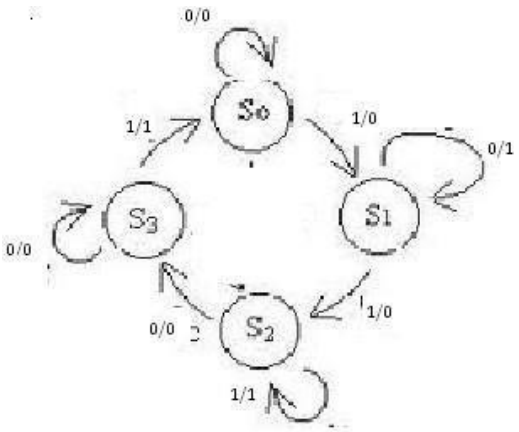
The synchronous or clocked sequential circuits are represented by two models.

1. Moore circuit: in this model, the output depends only on the present state of the flip-flops
2. Meelay circuit: in this model, the output depends on both present state of the flip-flop. And the inputs.

Sequential circuits are also called finite state machines (FSMs). This name is due to the fast that the functional behavior of these circuits can be represented using a finite number of states.

State diagram: the state diagram or state graph is a pictorial representation of the relationships between the present state, the input, the next state, and the output of a sequential circuit. The state diagram is a pictorial representation of the behavior of a sequential circuit.

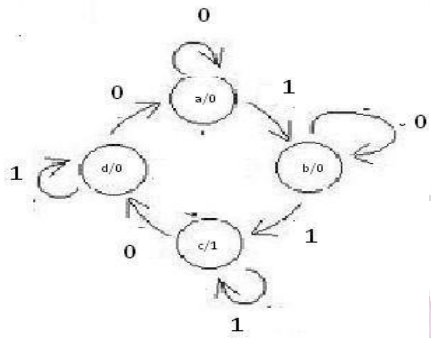
The state represented by a circle also called the node or vertex and the transition between states is indicated by directed lines connecting circle. a directed line connecting a circle with itself indicates that the next state is the same as the present state. The binary number inside each circle identifies the state represented by the circle. The direct lines are labeled with two binary numbers separated by a symbol. The input value is applied during the present state is labeled after the symbol.



	NS,O/P	
	INPUT X	
PS	X=0	X=1
a	a,0	b,0
b	b,1	c,0
c	d,0	c,1
d	d,0	a,1

Fig :a) state diagram (meelay circuit) fig: b) state table

In case of moore circuit ,the directed lines are labeled with only one binary number representing the input that causes the state transition. The output is indicated with in the circle below the present state, because the output depends only on the present state and not on the input.



	NS		
	INPUT X		
PS	X=0	X=1	O/P
a	a	b	0
b	b	c	0
c	d	c	1
d	a	d	0

Fig: a) state diagram (moore circuit) fig:b) state table

Serial binary adder:

Step1: word statement of the problem: the block diagram of a serial binary adder is shown in fig. it is a synchronous circuit with two input terminals designated X1and X2 which carry the two binary numbers to be added and one output terminal Z which represents the sum. The inputs and outputs consist of fixed-length sequences 0s and 1s.the output of the serial Z_i at time $t_i$is a function of the inputs $X_1(t_i)$ and $X_2(t_i)$ at that time t_{i-1} and of carry which had been generated at t_{i-1}. The carry which represent the past history of the serial adder may be a 0 or 1. The circuit has two states. If one state indicates that carry from the previous addition is a 0, the other state indicates that the carry from the previous addition is a 1

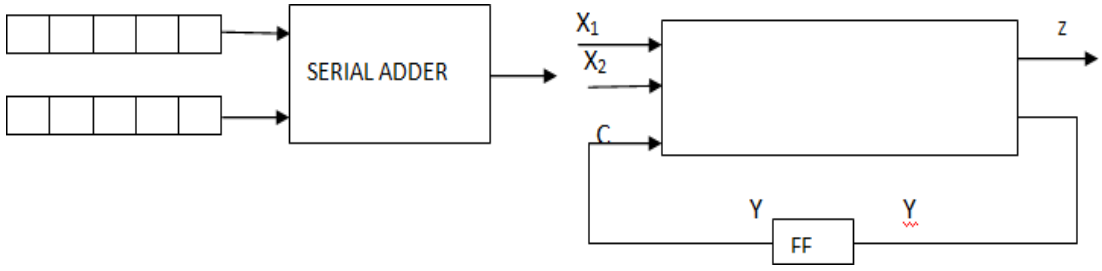
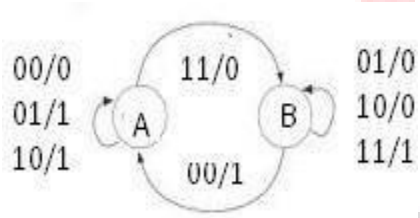


Figure: block diagram of serial binary adder

Step2 and 3: state diagram and state table: let a designate the state of the serial adder at t_i if a carry 0 was generated at t_{i-1}, and let b designate the state of the serial adder at t_i if carry 1 was generated at t_{i-1} .the state of the adder at that time when the present inputs are applied is referred to as the present state(PS) and the state to which the adder goes as a result of the new carry value is referred to as next state(NS).

The behavior of serial adder may be described by the state diagram and state table.



PS	NS ,O/P			
	X1 X2			
	0 0		1	1
	0 1		0	1
A	A,0	B,0	B,1	B,0
B	A,1	B,0	B,0	B,1

Figures: serial adder state diagram and state table

If the machine is in state B, i.e., carry from the previous addition is a 1, inputs X_1=0 and X_2=1 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs X_1=1 and X_2=0 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs X_1=1 and X_2=1 gives sum, 1 and carry 0. So the machine remains in state B and outputs a 1. Inputs X_1=0 and X_2=0 gives sum, 1 and carry 0. So the machine goes to state A and outputs a 1. The state table also gives the same information.

Setp4: reduced standard from state table: the machine is already in this form. So no need to do anything

Step5: state assignment and transition and output table:

The states, A=0 and B=1 have already been assigned. So, the transition and output table is as shown.

PS	NS				O/P			
	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1
0	0	0	0	1	0	1	1	1
1	0	1	1	1	1	0	0	1

STEP6: choose type of FF and excitation table: to write table, select the memory element the excitation table is as shown in fig.

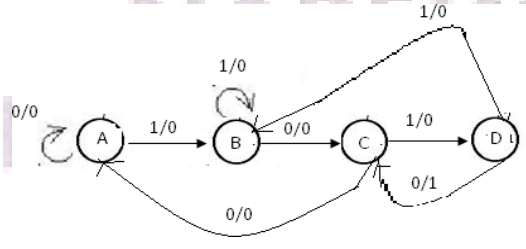
PS	I/P		NS	I/P-FF	O/P
y	x1	x2	Y	D	Z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Sequence detector:

Step1: word statement of the problem: a sequence detector is a sequential machine which produces an output 1 every time the desired sequence is detected and an output 0 at all other times

Suppose we want to design a sequence detector to detect the sequence 1010 and say that overlapping is permitted i.e., for example, if the input sequence is 01101010 the corresponding output sequence is 00000101.

Step2 and 3: state diagram and state table: the state diagram and the state table of the sequence detector. At the time t_1, the machine is assumed to be in the initial state designed arbitrarily as A. while in this state, the machine can receive first bit input, either a 0 o r a 1. If the input bit is 0, the machine does not start the detection process because the first bit in the desired sequence is a 1. If the input bit is a 1 the detection process starts.



PS	NS,Z	
	X=0	X=1
A	A,0	B,0
B	C,0	B,0
C	A,0	D,0
D	C,1	B,0

Figure: state diagram and state table of sequence detector

So, the machine goes to state B and outputs a 0. While in state B, the machinery may receive 0 or 1 bit. If the bit is 0, the machine goes to the next state, say state c, because the previous two bits are 10 which are a part of the valid sequence, and outputs 0.. if the bit is a 1, the two bits become 11 and this not a part of the valid sequence

Step4: reduced standard form state table: the machine is already in this form. So no need to do anything.

Step5: state assignment and transition and output table: there are four states therefore two states variables are required. Two state variables can have a maximum of four states, so, all states are utilized and thus there are no invalid states. Hence, there are no don‘t cares. Let a=00, B=01, C=10 and D=11 be the state assignment.

	NS(Y1Y2)		O/P(z)	
PS(y1y2	X=0	X=1	X=0	X=1
A= 0 0	0 0	0 1	0	0
B=0 1	1 0	0 1	0	0
C=1 0	0 0	1 1	0	0
D=1 1	1 1	0 1	1	0

Step6: choose type of flip-flops and form the excitation table: select the D flip-flops as memory elements and draw the excitation table.

PS	I/P	NS	INPUTS - FFS	O/P

y1	Y2	X	Y1	Y2	D1	D2	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	0

Step7: K-maps and minimal functions: based on the contents of the excitation table , draw the k-map and simplify them to obtain the minimal expressions for D1 and D2 in terms of y1, y2 and x as shown in fig. The expression for z (z=y1,y2) can be obtained directly from table

Step8: implementation: the logic diagram based on these minimal expressions