LECTURE NOTES

ON

DIGITAL CIRCUITS AND SYSTEMS

B.TECH ECE III YEAR I SEMESTER

(JNTUA-R15)

Mrs. A.KUSUMA KUMARI

ASSISTANT PROFESSOR

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

VEMU INSTITUTE OF TECHNOLOGY

P.KOTHAKOTA, PAKALA, CHITTOOR-517112

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

(15A04510) Digital Circuits & Systems

Course Outcomes:

Upon completion of the course, students should possess the following skills:

Be able to manipulate numeric information in different forms, e.g. different bases, signed integers, various codes such as ASCII, Gray, and BCD.

Be able to manipulate simple Boolean expressions using the theorems and postulates of Boolean algebra and to minimize combinational functions.

Be able to design and analyze small combinational circuits and to use standard combinational functions/building blocks to build larger more complex

circuits.

Be able to design and analyze small sequential circuits and devices and to use standard sequential functions/building blocks to build larger more complex circuits.

UNIT-I

Number System and Boolean Algebra And Switching Functions: Number Systems, Base Conversion Methods, Complements of Numbers, Codes- Binary Codes, Binary Coded Decimal Code and its Properties, Unit Distance Codes, Alpha Numeric Codes, Error Detecting and Correcting Codes. Boolean algebra: Basic Theorems and Properties, Switching Functions, Canonical and Standard Form, Algebraic Simplification of Digital Logic Gates, Properties of XOR Gates, Universal Gates, Multilevel NAND/NOR

UNIT -II:

Minimization and Design of Combinational Circuits: Introduction, The Minimization with theorem, The Karnaugh Map Method, Five and Six Variable Maps, Prime and

Essential Implications, Don"t Care Map Entries, Using the Maps for Simplifying,

Tabular Method, Partially Specified Expressions, Multi-output Minimization, Minimization and Combinational Design, Arithmetic Circuits, Comparator, Multiplexers, Code Converters, Wired Logic, Tristate Bus System, Practical Aspects related to Combinational Logic Design, Hazards and Hazard Free Relations.

UNIT III

SEQUENTIAL CIRCUITS

Latches, Flip-flops - SR, JK, D, T, and Master-Slave – Characteristic table and equation – Application table – Edge triggering – Level Triggering – Realization

of one flip flop using other flip flops – serial adder/sub-tractor- Asynchronous

Ripple or serial counter – Asynchronous Up/Down counter - Synchronous counters

– Synchronous Up/Down counters – Programmable counters – Design of Synchronous counters: state diagram- State table –State minimization –State assignment - Excitation table and maps-Circuit implementation - Modulo–n counter, Registers

– shift registers - Universal shift registers – Shift register counters – Ring counter – Shift counters - Sequence generators.

UNIT IV

MEMORY DEVICES

Classification of memories – ROM - ROM organization - PROM – EPROM – EEPROM

– EAPROM, RAM – RAM organization – Write operation – Read operation – Memory cycle - Timing wave forms – Memory decoding – memory expansion – Static RAM Cell- Bipolar RAM cell – MOSFET RAM cell – Dynamic RAM cell – Programmable Logic Devices – Programmable Logic Array (PLA) - Programmable Array Logic (PAL) - Field Programmable Gate Arrays (FPGA) - Implementation of combinational logic circuits using ROM, PLA, PAL

UNIT V

SYNCHRONOUS AND ASYNCHRONOUS SEQUENTIAL CIRCUITS

Synchronous Sequential Circuits: General Model – Classification – Design – Use of Algorithmic State Machine – Analysis of Synchronous Sequential Circuits Asynchronous Sequential Circuits: Design of fundamental mode and pulse mode circuits – Incompletely specified State Machines – Problems in Asynchronous Circuits – Design of Hazard Free Switching circuits. Design of Combinational and Sequential circuits using VERILOG

TEXT BOOKS:

1. Switching and Finite Automata Theory- Zvi Kohavi & Niraj K. Jha, 3rd Edition, Cambridge.

2. Digital Design- Morris Mano, PHI, 4th Edition. Prentice Hall of India Pvt. Ltd.,

2003 / Pearson Education (Singapore) Pvt. Ltd., New Delhi, 2003.

3. S. Salivahanan and S. Arivazhagan, Digital Circuits and Design, 3rd Edition., Vikas Publishing House Pvt. Ltd.

UNIT I

Number System & Boolean Algebra

1.1 Digital Systems:

Digital electronics or digital (electronic) circuits are electronics that handle digital signals – discrete bands of analog levels – rather than by continuous ranges (as used in analogue electronics). All levels within a band of values represent the same numeric value. Because of this discretization, relatively small changes to the analog signal levels due to manufacturing tolerance, signal attenuation or parasitic noise do not leave the discrete envelope, and as a result are ignored by signal state sensing circuitry.

In most cases, the number of these states is two, and they are represented by two voltage bands: one near a reference value (typically termed as "ground" or zero volts), and the other a value near the supply voltage. These correspond to the "false" ("0") and "true" ("1") values of the Boolean domain respectively, named after its inventor,George Boole, yielding binary code.

Digital techniques are useful because it is easier to get an electronic device to switch into one of a number of known states than to accurately reproduce a continuous range of values.Digital electronic circuits are usually made from large assemblies of logic gates, simple electronic representations of Boolean logic functions.

An advantage of digital circuits when compared to analog circuits is that signals represented digitally can be transmitted without degradation due to noise.[8] For example, a continuous audio signal transmitted as a sequence of 1s and 0s, can be reconstructed without error, provided the noise picked up in transmission is not enough to prevent identification of the 1s and 0s. An hour of music can be stored on a compact disc using about 6 billion binary digits.

In a digital system, a more precise representation of a signal can be obtained by using more binary digits to represent it. While this requires more digital circuits to process the signals, each digit is handled by the same kind of hardware, resulting in an easily scalable system. In an analog system, additional resolution requires fundamental improvements in the linearity and noise characteristics of each step of the signal chain.

Computer-controlled digital systems can be controlled by software, allowing new functions to be added without changing hardware. Often this can be done outside of the factory by updating the product's software. So, the product's design errors can be corrected after the product is in a customer's hands.

Information storage can be easier in digital systems than in analog ones. The noise-immunity of digital systems permits data to be stored and retrieved without degradation. In an analog system, noise from aging and wear degrade the information stored. In a digital system, as long as the total noise is below a certain level, the information can be recovered perfectly.

Even when more significant noise is present, the use of redundancy permits the recovery of the original data provided too many errors do not occur. In some cases, digital circuits use more energy than analog circuits to accomplish the same tasks, thus producing more heat which increases the complexity of the circuits such as the inclusion of heat sinks. In portable or battery-powered systems this can limit use of digital systems.

1.2 The Binary Number System:

It is a positional weighted system. The base or radix of this no. system is 2 Hence it has two independent symbols. The basic itself can‘t be a symbol. The symbol used is 0 and 1.The binary digit is called a bit. A binary no. consists of a sequence of bits each of which is either a 0 or 1. The binary point separates the integer and fraction parts. Each digit (bit) carries a weight based on its position relative to the binary point. The weight of each bit position is on power of 2 greater than the weight of the position to its immediate right. The first bit to the left of the binary point has a weight of 2^0 & that column is called the Units Column. The second bit to the left has a weight of 2^1 & it is in the 2‘s column & the third has weight of 2^2& so on. The first bit to the right of the binary point has a weight of 2^{-1} & it is said to be in the ½ _s column , next right bit with a weight of 2^{-2} is in ¼‘s column so on..The decimal value of the binary no. is the sum of the products of all its bits multiplied by the weight of their respective positions. In general , binary no. wioth an integer part of (n+1) bits & a fraction parts of k bits can be

$d_n\ d_{n-1}\ d_{n-2}\ \ldots\ldots\ldots d_1\ d_0.d_{-1}\ d_{-2}\ d_{-3}\ \ldots\ldots d_{-k}$

In decimal equivalent is

$(d_n\ x2^n)+(d_{n-1}\ x2^{n-1})+\ \ldots\ldots\ldots(d_1\ x2^1)+(d_0\ x2^0)+(d_{-1}\ x2^{-1})(d_{-2}\ x2^{-2})\ \ldots\ldots$

The decimal equivalent of the no. system

$d_n\ d_{n-1}\ d_{n-2}$ ……$d_1\ d_0.d_{-1}\ d_{-2}\ d_{-3}$ …….d_{-k} in any system with base b is

$(d_n\ xb^n)+(d_{n-1}\ xb^{n-1})+$ ……$(d_1\ xb^1)+(d_0\ xb^0)+(d_{-1}\ xb^{-1})(d_{-2}\ xb^{-2})$ …….

The binary no. system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors , diodes etc. A transistor can be OFF or ON a switch can be OPEN or CLOSED , a diode can be OFF or ON .These two states represented by the symbols 0 & 1 respectively.

Counting in binary:

Easy way to remember to write a binary sequence of n bits is to follow 8421 code or

- The rightmost column in the binary number begins with a 0 & alternates between 0 & 1.
- Second column begins with 2(=2^1) zeros & alternates between the groups of 2 zeros & 2 ones. So on

Decmal no.	Binary no.	Decimal no.	Binary no.
0	0	20	10100
1	1	21	10101
2	10	22	10110
3	11	23	10111
4	100	24	11000
5	101	25	11001
6	110	26	11010
7	111	27	11010
8	1000		
9	1001		
10	1010		
11	1011		
12	1100		
13	1101		
14	1110		
15	1111		
16	10000		
17	10001		
18	10010		
19	10011	39	100111

1.3 Number base conversions

Conversion from any system to decimal

The number of values that can be expressed by a single digit in any number system is called the system radix, and any value can be expressed in terms of its system radix.

Octal to Decimal

For example the system radix of octal is 8, since any of the 8 values from 0 to 7 can be written as a single digit.

Convert 126_8 to decimal.

Using the values of each column, (which in an octal integer are powers of 8) the octal value 126_8 can also be written as:

$(1x8^2) + (2x8^1) + (6 \times 8^0)$

As ($8^2 = 64$), ($8^1 = 8$) and ($8^0 = 1$), this gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:

$1x64 = 64$ $2x8 = 16$ $6x1 = 6$

Then simply add these results to give the decimal value.

$64 + 16 + 6 = 86_{10}$

Therefore $126_8 = 86_{10}$.

Binary to Decimal

Convert 1101_2 to decimal.

The same method can be used to convert binary number to decimal:

$= (1x2^3)+(1x2^2)+(0x2^1)+(1x2^0)$

$= 8 + 4 + 0 + 1$

$= 13_{10}$

Therefore $1101_2 = 13_{10}$.

Hexadecimal to Decimal

Convert $B2D_{16}$ to decimal.

Using the same method to convert hexadecimal to decimal.

$= (Bx16^2)+(2x16^1)+(Dx16^0)$

$= (11x16^2)+(2x16^1)+(13x16^0)$

$= 2816 + 32 +13$

$= 2861_{10}$

Therefore $B2D_{16} = 2861_{10}$.

The same method (multiplying each digit by it's column value) can be to convert any system to decimal.

Converting from Decimal to any Radix

To convert a decimal integer number (a decimal number in which any fractional part is ignored) to any other radix, all that is needed is to continually divide the number by its radix, and with each division, write down the remainder. When read from bottom to top, the remainder will be the converted result.

Decimal to Octal

convert the decimal number 86_{10} to octal:

Divide 86_{10} by the system radix, which when converting to octal is 8. This gives the answer 10, with a remainder of 6.

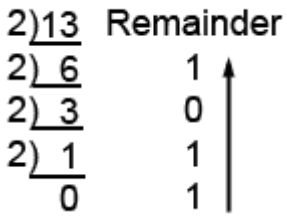
	Remainder
8)86	
8)10	6
8) 1	2
0	1

Continue dividing the answer by 8 and writing down the remainder until the answer = 0. Now simply write out the remainders, starting from the bottom, to give 126_8

Therefore $86_{10} = 126_8$

Decimal to Binary

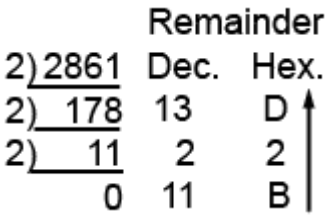
Convert the decimal number 13_{10} to binary:



This process also works to convert decimal to binary, but this time the system radix is 2:

Therefore $13_{10} = 1101_2$

Decimal to Hexadecimal



It also works to convert decimal to hexadecimal, but now the radix is 16:

As some of the remainders may be greater than 9 (and so require their alphabetic replacement), you may find it easier to use Decimal for the remainders, and then convert them to Hex.

Therefore $2861_{10} = B2D_{16}$

Numbers with Fractions :

It is very common in the decimal system to use fractions; that is any decimal number that contains a decimal point, but how can decimal numbers, such as 34.625_{10} be converted to binary fractions?

In electronics this is not normally done, as binary does not work well with fractions. However as fractions do exist, there has to be a way for binary to deal with them. The method used is to get rid of the radix (decimal) point by NORMALISING the decimal fraction

using FLOATING POINT arithmetic. As long as the binary system keeps track of the number of places the radix point was moved during the normalisation process, it can be restored to its correct position when the result of the binary calculation is converted back to decimal for display to the user.

However, for the sake of completeness, here is a method for converting decimal fractions to binary fractions. By carefully selecting the fraction to be converted, the system works, but with many numbers the conversion introduces inaccuracies; a good reason for not using binary fractions in electronic calculations.

Converting the Decimal Integer to Binary

	Remainder
2)34	
2)17	0
2) 8	1
2) 4	0
2) 2	0
2) 1	0
0	1

The radix point splits the number into two parts; the part to the left of the radix point is called the INTEGER. The part to the right of the radix point is the FRACTION. A number such as 34.625_{10} is therefore split into 34_{10} (the integer), and $.625_{10}$ (the fraction).

To convert such a fractional decimal number to any other radix, the method described above is used to convert the integer.

So $34_{10} = 100010_2$

Converting the Decimal Fraction to Binary

	Carry	
Fraction		625
x Radix		x2
Result	1	250
x Radix		x2
Result	0	500
x Radix		x2
Result	1	000

To convert the fraction, this must be MULTIPLIED by the radix (in this case 2 to convert to binary). Notice that with each multiplication a CARRY is generated from the third column. The Carry will be either 1 or 0 and these are written down at the left hand side of the result. However when each result is multiplied the carry is ignored (don't multiply the carry). Each result is multiplied in this way until the result (ignoring the carry) is 000. Conversion is now complete.

For the converted value just read the carry column from top to bottom.

So $0.625_{10} = .101_2$

Therefore the complete conversion shows that $34.62510 = 100010.101_2$

However, with binary, there is a problem in using this method, .625 converted easily but many fractions will not. For example if you try to convert .626 using this method you would find that the binary fraction produced goes on to many, many places without a result of exactly 000 being reached.

With some decimal fractions, using the above method will produce carries with a repeating pattern of ones and zeros, indicating that the binary fraction will carry on infinitely. Many decimal fractions can therefore only be converted to binary with limited accuracy. The number of places after the radix point must be limited, to produce as accurate an approximation as required.

The most commonly encountered number systems are binary and hexadecimal, and a quick method for converting to decimal is to use a simple table showing the column weights, as shown in Tables 1.2.1a and 1.2.1b.

Converting Binary to Decimal

Table 1.2.1a

Bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value (weighting) of each bit	128	64	32	16	8	4	2	1
8 Bit Binary	0	1	0	0	0	0	1	1

To convert from binary to decimal, write down the binary number giving each bit its correct 'weighting' i.e. the value of the columns, starting with a value of one for the right hand (least significant) bit. Giving each bit twice the value of the previous bit as you move left.

Example:

To convert the binary number 01000011_2 to decimal. Write down the binary number and assign a 'weighting' to each bit as in Table 1.2.1a

Now simply add up the values of each column containing a 1 bit, ignoring any columns containing 0.

Applying the appropriate weighting to 01000011 gives 64 + 2 + 1 = 67

Therefore: $01000011_2 = 67_{10}$

Binary and Hexadecimal

Converting between binary and hexadecimal is a much simpler process; hexadecimal is really just a system for displaying binary in a more readable form.

Binary is normally divided into Bytes (of 8 bits) it is convenient for machines but quite difficult for humans to read accurately. Hexadecimal groups each 8-bit byte into two 4-bit nibbles, and assigns a value of between 0 and 15 to each nibble. Therefore each hexadecimal digit (also worth 0 to 15) can directly represent one binary nibble. This reduces the eight bits of binary to just two hexadecimal characters.

BINARY	HEX
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

For example:

111010012 is split into 2 nibbles 11102 and 10012 then each nibble is assigned a hexadecimal value between 0 and F.

The bits in the most significant nibble (11102) add up to 8+4+2+0 = 1410 = E16

The bits in the least significant nibble (10012) add up to 8+0+0+1 = 910 = 916

Therefore 111010012 = E916

Converting hexadecimal to binary of course simply reverses this process.

1.4 Complements of numbers:

Ones Complement Notation

The complement (or opposite) of +5 is −5. When representing positive and negative numbers in 8-bit ones complement binary form, the positive numbers are the same as in signed binary notation. However, the complement of these numbers, that is their negative counterparts, are represented by 'complementing' each 1 bit of the positive binary number to 0 and each 0 to 1.

For example:

$+5_{10}$ is 00000101_2

-5_{10} is 11111010_2

Twos Complement Notation

Twos complement notation solves the problem of the relationship between positive and negative numbers, and achieves accurate results in subtractions.

To perform binary subtraction, the twos complement system uses the technique of complementing the number to be subtracted. In the ones complement system this produced a result that was 1 less than the correct answer, but this could be corrected by using the 'end around carry' system. This still left the problem that positive and negative versions of the same number did not produce zero when added together.

The two's complement system overcomes both of these problems by simply adding one to the ones complement version of the number before addition takes place.

9's & 10's Complements:

It is the Subtraction of decimal no.s can be accomplished by the 9's & 10's compliment methods similar to the 1's & 2's compliment methods of binary . the 9's compliment of a decimal no. is obtained by subtracting each digit of that decimal no. from 9. The 10's compliment of a decimal no is obtained by adding a 1 to its 9's compliment.

Example: 9's compliment of 3465 and 782.54 is

9999	999.99
-3465	-782.54
----------	----------
6534	217.45
----------	----------

10's complement of 4069 is

9999
- 4069

5930
+1

5931

9's compliment method of subtraction:

To perform this, obtain the 9's compliment of the subtrahend and it to the minuend now call this no. the intermediate result .if there is a carry to the LSD of this result to get the answer called **end around carry**.If there is no carry , it indicates that the answer is negative & the intermediate result is its 9's compliment.

Example: Subtract using 9's comp

(1)745.81-436.62

745.81
-436.62

309.19

745.81
+563.37 9's compliment of 436.62

1309.18 Intermediate result
+1 end around carry

309.19
---

(2)436.62-745.82

436.62
-745.81

-309.19

436.62
+254.18

690.80

If there is no carry indicating that answer is negative . so take 9's complement of intermediate result & put minus sign (-) result should ne -309.19

If carry indicates that the answer is positive +309.19

10's compliment method of subtraction:

To perform this, obtain the 10's compliment of the subtrahend& add it to the minuend. If there is a carry ignore it. The presence of the carry indicates that the answer is positive, the result is the answer. If there is no carry, it indicates that the answer is negative & the result is its 10's compliment. Obtain the 10's compliment of the result & place negative sign infront to get the answer.

Example: (a)2928.54-41673 (b)416.73-2928.54

2928.54		0416.73
-0416.73		-2928.54
2511.81		-2511.81
2928.54		0416.73
+9583.27	10‘s compliment of 436.62	+7071.46
12511.81	ignore the carry	7488.19

1.5 Signed Binary Notation

All the binary arithmetic problems looked so far, used only POSITIVE numbers. The reason for this is that it is not possible in PURE binary to signify whether a number is positive or negative. This of course would present a problem in any but the simplest of arithmetic.

There are a number of ways in which binary numbers can represent both positive and negative values, 8 bit systems for example normally use one bit of the byte to represent either + or − and the remaining 7 bits to give the value. One of the simplest of these systems is SIGNED BINARY, also often called ‘Sign and Magnitude’, which exists in several similar versions, but is commonly an 8 bit system that uses the most significant bit (msb) to indicate a positive or a negative value. By convention, a 0 in this position indicates that the number given by the remaining 7 bits is positive, and a most significant bit of 1 indicates that the number is negative.

For example:

+4510 in signed binary is $(0)0101101_2$

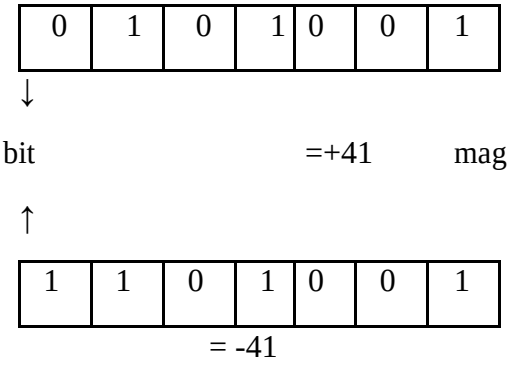
-4510 in signed binary is $(1)0101101_2$

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.

Example:



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.

Example

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

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	--	--

1.6 Binary codes:

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 610(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 610(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 is then added.

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) Subtractions:

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 from 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 difference term(b coz taking a borrow is equivalent to adding six invalid states , result is in xs-6)

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.

Gray Code					
1 bit	2 bit	3 bit	4 bit	Decimal	4 bit binary
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

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

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**:** Odd parity**,** even parity for odd 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

(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

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		

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 :

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.

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

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

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	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}

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	D_{10}	D_{11}	D_{12}

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 And Switching Functions

Boolean algebra:

Switching circuits called Logic circuits, gate circuits & digital circuits. Switching algebra called Boolean Algebra. Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of element (0.1) two binary operators called OR & AND

& One unary operator NOT. Binary Digits 0 & 1 used to represent two voltage levels. Binary 1 is for high i.e, +5v . Binary 0 for Low i.e, 0v.

A+A=A A.A=A because variable has only a logic value.
Also there are some theorems of Boolean Algebra.

1. (a) A + A = A	(b) A.A = A	Tautology Law
2. (a) A + 1 = 1	(b) A.0 = 0	Union Law
3. (A′)′ = A		Involution Law
4. (a) A + (B + C) = (A + B) + C	(b) A.(B.C) = (A.B).C	Associative Law
5. (a) (A + B)′ = A′B′	(b) (A.B)′ = A′ + B′	De Morgan's Law
6. (a) A + AB = A	(b) A(A + B) = A	Absorption Law
7. (a) A + A′B = A + B	(b) A(A′ + B) = AB	
8. (a) AB + AB′ = A	(b) (A + B) (A + B′) = A	Logical adjancy
9. (a) AB + A′C + BC = AB + A′C	(b) (A + B) (A′ + C) (B + C) = (A + B)	Consensus Law

Logic Operators:

AND, OR, NOT are 3 basic operations or functions that performed in Boolean Algebra. & derived operations as NAND, NOR, X-OR, X-NOR.

AXIOMS & Laws of Boolean algebra:

Axioms or Postulates are a set of logical expressions i.e, without proof. & also we can build a set of useful theorems. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

AND	OR	NOT
0.0=0	0+0=0	1=0
0.1=0	0+1=1	0=1
1.0=0	1+0=1	
1.1=1	1+1=1	

Complementation Laws:

Complement means invert (0 as 1 & 1 as 0)

Law1:0=1
Law2:1=0

Law3:If A=0 then =1
Law4:If A=1 then =0
Law5: =A(double complementation law)

AND laws:

Law 1: A.0=0(Null law)
Law 2:A.1=A(Identity law)
Law 3:A.A=A
Law 4:A. =0

OR laws:

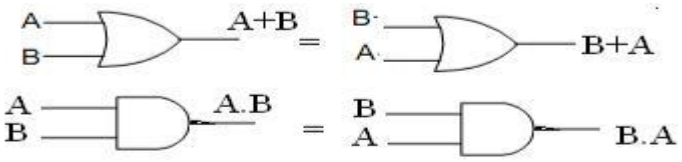
Law 1: A+0=A(Null law)
Law 2:A+1=1
Law 3:A+A=A
Law 4:A+ =0

Commutative laws: allow change in position of AND or OR variables. 2 commutative laws

Law 1: A+B=B+A
Law 2: A.B=B.A

A B	A+B	=	B A	B+A
0 0	0		0 0	0
0 1	1		0 1	1
1 0	1		1 0	1
1 1	1		1 1	1

A.B	B.A
0	0
0	0
0	0
1	1

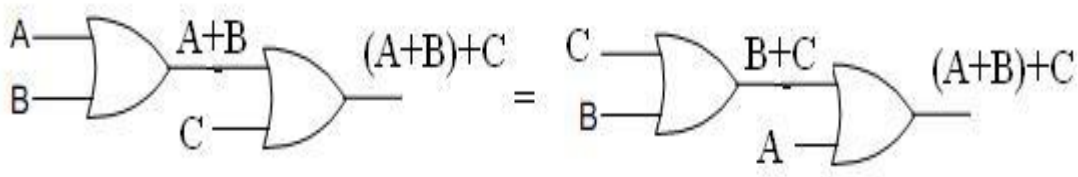


Associative laws: This allows grouping of variables. It has 2 laws.

Law 1: (A+B)+C=A+(B+C) =A OR B ORed with C

This law can be extended to any no. of variables

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



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

=

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

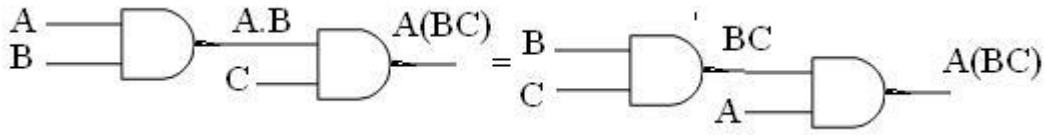
1 1 0	1	1
1 1 1	1	1

1 1 0	1	1
1 1 1	1	1

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

This law can be extended to any no. of variables

(A.B.C).D=(A.B.C).D



A B C	AB	(AB)C
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	0	0
1 0 0	0	0
1 0 1	0	0
1 1 0	1	0
1 1 1	1	1

=

A B C	BC	A(BC)
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	1	0
1 0 0	0	0
1 0 1	0	0
1 1 0	0	0
1 1 1	1	1

Distributive Laws:

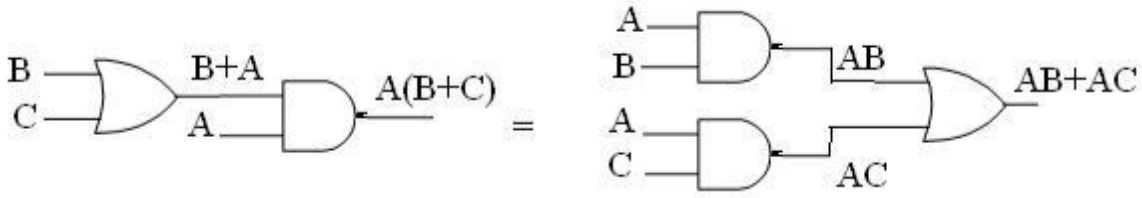
This has 2 laws

Law 1.A(B+C)=AB+AC

This law applies to single variables.

EX:ABC(D+E)=ABCD+ABCE

AB(CD+EF)=ABCD+ABEF



Law 2.A+BC=(A+B)(A+C) RHF=(A+B)(A+C)

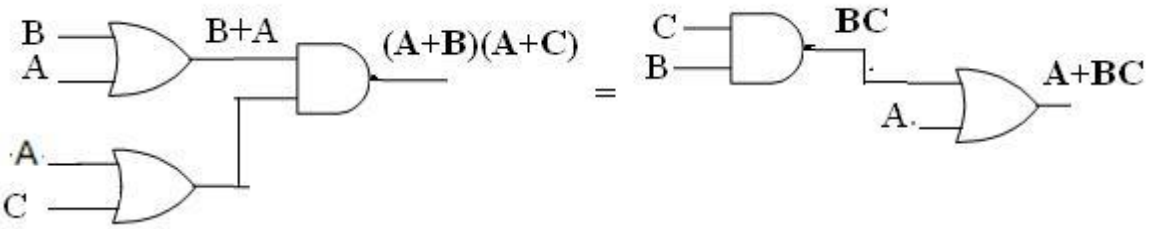
=AA+AC+BA+BC

=A+AC+AB+BC

=A(1+C+B)+BC

=A.1+BC

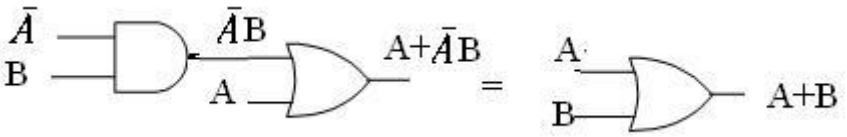
=A+BC LHF



Redundant Literal Rule(RLR):

Law 1: A+ B=A+B

LHF = (A+)(A+B) =1.(A+B) =A+B



ORing of a variable with the AND of the compliment of that variable with another variable, is equal to the ORing of the two variables.

A B	B	A+ B
0 0	0	0
0 1	1	1
1 0	0	1
1 1	0	1

=

A	B	A+B
0	0	0
0	1	1
1	1	1
1	1	1

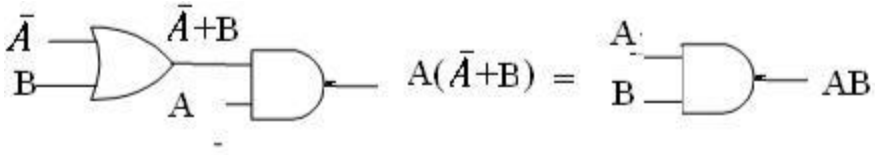
Law 2:A(A +B)=AB

LHF = A.A +A.B

=0+AB

=AB RHF

ANDing of a variable with the OR of the complement of that variable with another variable , is equal to the ANDing of the two variables.



A B	+B	A(+B)
0 0	1	0
0 1	1	0
1 0	0	0

=

A	B	A+B
0	0	0
0	1	0
1	1	0

1 1	1	1

Idempotence Laws:

Idempotence means same value. It has 2 laws.

Law 1=A.A=A
Law 2=A+A=A

Absorption Laws:

Law 1=A+A.B=A

= A(1+B)

=A.1

=A

i.e., A+A. any term=A

A B		A+ B)
0 0	0	0
0 1	0	0
1 0	0	1
1 1	1	1

Law 2=A(A+B)=A

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

= A+AB

=A(1+B)

= A.1

=A

A B	+	A(A+B)
0 0	0	0
0 1	1	0
1 0	1	1
1 1	1	1

Consensus theorem:

Theorem 1: AB+ C+BC=AB+ c

LHS: AB+ C+BC =AB+
C+BC(A+) =AB+
C+BCA+BC
=AB(1+C)+ c(1)
=AB+ C
RHS

This can be extended to any no. of variables

EX: AB+ C+BCD =AB+

Theorem 2: (A+B)(+)(B+C)=(A+B)(+C)

Transposition Theorem:

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

RHS: (A+C)(+B)

=A +C +AB+CB =0+
C+AB+BC =
C+AB+BC(A+)
=AB+ABC+ C+ BC
=AB+ C
LHS

Shannon's expansion Theorem:

This theorem states that any switching expression can be decomposed w.r.t. a variable A into two parts, one containing A &other containing A'. It is useful in decomposing complex machines into an interconnection of smaller components.

f(A,B,C---)=A.f(1,B,C---)+A'.f(0,B,C----)

f(A,B,C,---)=[A+f(0,B,C,-----)].[A'+f(1,B,C-----]

Ex: DeMorganize f=((A+B')(C+D'))', f=((A+B')(C+D'))'
=(A+B')(C+D')
= A.B' +C.D'=A'.B+C'.D

DUALITY:

In a positive Logic system the more positive of the two voltage levels is represented by a 1 & the more negative by a 0. In a negative logic system the more positive of the two voltage levels is represented by a 0 & more negative by a 1. This distinction between positive &negative logic systems is important because an OR gate in the positive logic system becomes an AND gate in the negative logic system &vice versa. Positive & Negative logics give a basic duality in Boolean identities. Procedure dual identity by changing all _+' (OR) to —.'(AND) & complementing all 0's &1's. Once a theorem or statement is proved, the dual also thus stands proved called Principle of duality.

$[f(A,B,C,-------0,1,+,.)]_d$ =f(A,B,C,----1,0,.,+)

Boolean functions & their representation:

A function of n Boolean variables denoted by $f(x_1,x_2,x_3\text{------}x_n)$ is another variable denoted by & takes one of the two possible values 0 & 1.

The various ways of representing a given function is

1. *Sum of Product(SOP) form:*

A function of n Boolean variables denoted by $f(x_1,x_2,x_3\text{------}x_n)$ is another variable denoted by & takes one of the two possible values 0 & 1.

The various ways of representing a given function is

2. *Sum of Product(SOP) form:*

It is called the Disjunctive Normal Form(DNF) Ex:f(A,B,C)=(B+ C)

2. Product of Sums (POS) form:

It is called the Conjunctive Normal Form(CNF).This is implemented using Consensus theorem.

3. Truth Table form:

The function is specified by listing all possible combinations of values assumed by the variables & the corresponding values of the function.

4. Standard Sum of Products form: Called Disjunctive Canonical form (DCF) & also called Expanded SOP form or Canonical SOP form

f(A,B,C)=(B+ C)= B(C+B)+ C(A+C) = C+ B + BC+A C

A Product term contains all the variables of the function either in complemented or Uncomplemented form is called a minterm. A minterm assumes the value 1 only for one combination of the variables. An n variable function can have in all 2^n minterms to 1 is the standard sum of products form of the function. Min terms are denoted as m_0, m1,m2-- --. Here suffixes are denoted by the decimal codes.

5. Standard Product of Sums form: It is called as Conjunctive Canonical form (CCF). It is also called Expanded POS or Canonical POS.

6. Octal designation

7. Karnaugh Map

Put the Truth Table in a compact form by labeling the row & columns of a map. It is used in the minimization of functions 3,4,5,6 variables.

m_0, m1, m2,m3 ----- are minterms

M_0, M1, M2, M3--------are Maxterms.

Expansion of a Boolean expression in SOP form to the standard SOP form:

1.Write down all the terms.

2. If one or more variables are missing in any term.Expand that term by multiplying it with the sum of each one of the missing variable and its complement.

3. Drop out redundant terms.

interms of minterms:

1. Write down all the terms.

2. Put Xs in terms where variables must be inserted to form a minterm. 3.Replace the non-complemented variables by 1s and the complemented variables by 0s, and use all combinations of Xs in terms of 0s and 1s to generate minterms.

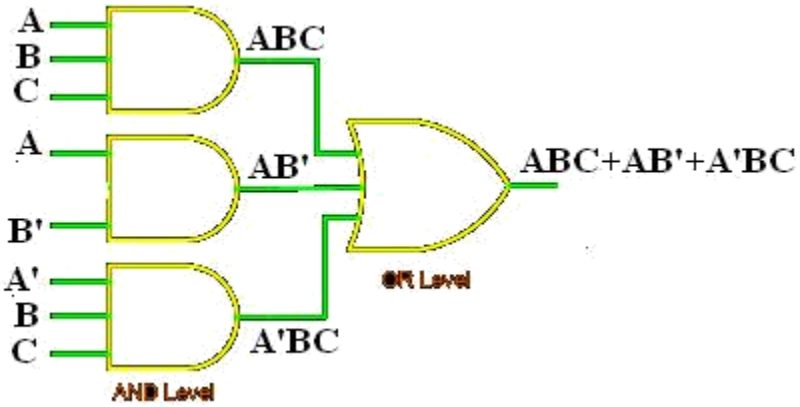
4. Drop out redundant terms.

Boolean Expression & Logic Diagrams:

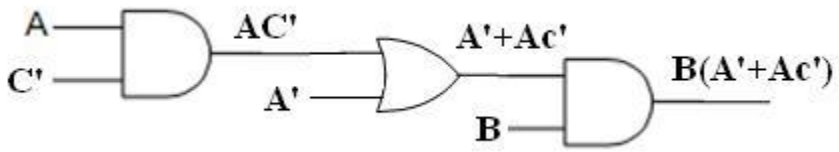
Boolean expressions can be realized as hardware using logic gates. Conversely, hardware can be translated into Boolean expressions for the analysis of existing circuits.

1. Converting Boolean Expressions to Logic:

To convert, start with the output & work towards the input.



The POS expression (A+B+C)(A+B‘)(A‘+B+C) can be implemented using OR and AND gates. The expression ABC‘+A‘B[=B(A‘+AC‘) can be implemented in hybrid form as



Hybrid Logic reduces the no. of gate inputs required for realization (from 7 to 6 in this case), but results in multilevel logic. Different inputs pass through number of gates to reach the output. It leads to non-uniform propagation delay between different numbers of gates to give rise to logic race. The SOP and POS realizations give rise two-level logic. The two-level logic provides uniform time delay between input and outputs, because each input signal has to pass through two gates to reach the output. So, it does not suffer from the problem of logic race.

Since NAND logic and NOR logic are universal logic circuits which are first computed and converted to AOI logic may then be converted to either NAND logic or NOR logic depending on the choice. The procedure is

1. Draw the circuit in AOI logic
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the AND gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates
4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram
5. Replace bubbled OR by NAND and bubbled AND by NOR
6. Eliminate double inversions.

LOGIC GATES: Logic gates are fundamental building blocks of digital systems. Logic gate produces one output level when some combinations of input levels are present. & a different

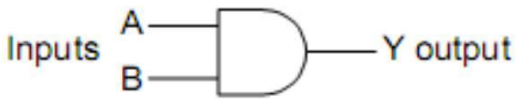
output level when other combination of input levels is present. In this, 3 basic types of gates are there. AND, OR & NOT.

The interconnection of gates to perform a variety of logical operation is called *Logic Design*. Inputs & outputs of logic gates can occur only in two levels.1,0 or High, Low or True , False or On , Off. A table which lists all the possible combinations of input variables & the corresponding outputs is called a Truth Table. It shows how the logic circuits output responds to various combinations of logic levels at the inputs. *Level Logic*, a logic in which the voltage levels represent logic 1 & logic 0.Level logic may be Positive Logic or Negative Logic. In *Positive Logic* the higher of two voltage levels represent logic 1 & Lower of two voltage levels represent logic 0.In *Negative Logic* the lower of two voltage levels represent logic 1 & higher of two voltage levels represent logic 0.

In TTl (Transistor-Transistor Logic) Logic family voltage levels are +5v, 0v.Logic 1 represent +5v & Logic 0 represent 0v.

AND Gate:

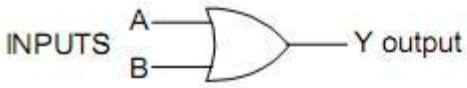
It is represented by ' **.'**(dot) It has two or more inputs but only one output. The output assume the logic 1 state only when each one of its inputs is at logic 1 state . The output assumes the logic 0 state even if one of its inputs is at logic 0 state. The AND gate is also called an All or Nothing gate.



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate:

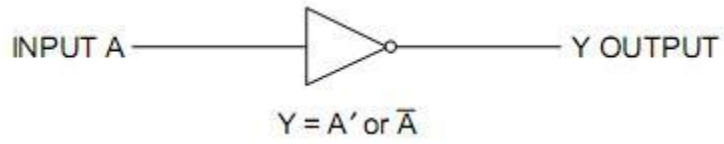
It is represented by '+' (plus) It has two or more inputs but only one output. The output assumes the logic 1 state only when one of its inputs is at logic 1 state. The output assumes the logic 0 state even if each one of its inputs is at logic 0 state. TheOR gate is also called an any or All gate. Also called an inclusive OR gate because it includes the condition both the inputs can be present.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate:

It is represented by '-'(bar).It is also called an *Inverter or Buffer*. It has only one input & one output. Whose output always the compliment its input. the output assumes logic 1 when input is logic 0 & output assume logic 0 when input is logic 1.



Truth Table

A	X
1	0
0	1

Boolean Expression:

X=A'

The Universal Gates:

The universal gates are NAND, NOR. Each of which can also realize Logic Circuits Single handedly. NAND-NOR called Universal Building Blocks.. Both NAND-NOR can perform all the three basic logic functions. AOI logic can be converted to NAND logic or NOR logic.

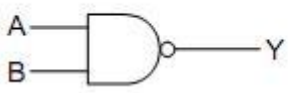
NAND Gate:

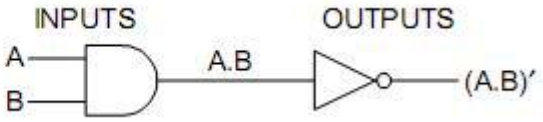
NAND gate mean NOT AND i.e, AND output is NOTed.
NAND→AND & NOT gates

Boolean Expression: $Y= \overline{A .B.C}$

NAND assumes Logic 0 when each of inputs assume logic 1.

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

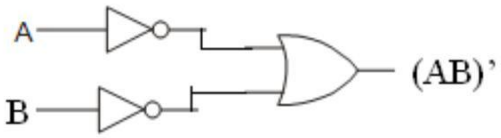




Bubbled OR gate is OR gate with inverted inputs. The output of this is same as NAND gate.

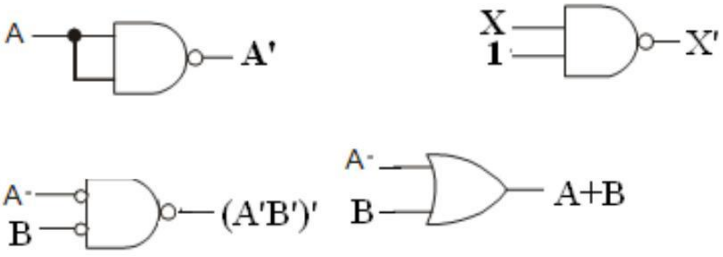
Y=A'+B'=(AB)'

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



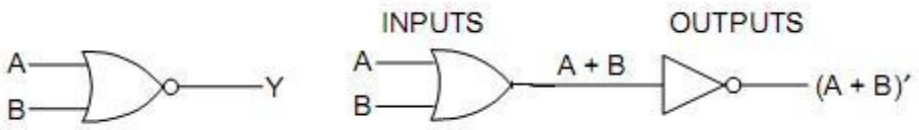
- NAND gate as an Inverter.

All its input terminals together & applying the signal to be inverted to the common terminal by connecting all input terminals except one to logic 1 & applying the signal to be inverted to the remaining terminal. It is also called Controlled Inverter.



NOR Gate:

NOR gate is NOT gate with OR gate. i.e, OR gate is NOTed.



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

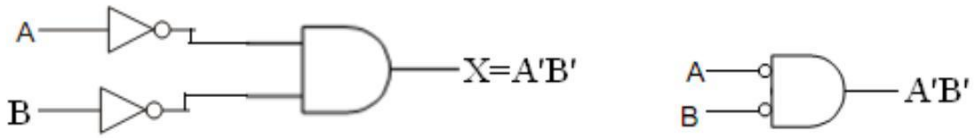
Truth Table

Bubbled AND gate:

It is AND gate with inverted inputs.The AND gate with inverted inputs is called a bubbled And gate. So a NOR gate is equivalent to a bubbled and gate.A bubbled AND gate is also called a

negative AND gate. Since its output assumes the HIGH state only when all its inputs are in LOW state , a NOR gate is also called active-LOW AND gate.Output Y is 1 only when both A & B are equal to 0.i.e, only when both A‘ and B‘ are equal to 1.

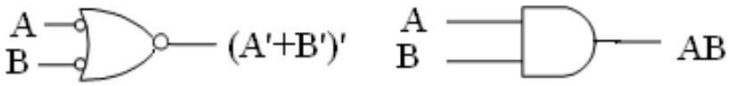
NOR can also realized by first inverting the inputs and ANDing those inverted inputs.



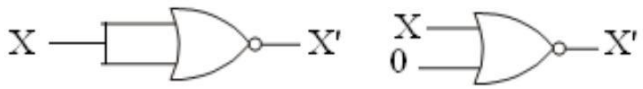
Inputs A B	Inverted Inputs A‘ B‘	Output Y
0 0	1 1	1
0 1	1 0	0
1 0	0 1	0
1 1	0 0	0

NOR gate as an inverter:

is tying all input terminals together & applying the signal to be inverted to the common terminals or all inputs set as logic 0 except one & applying signal to be inverted to the remaining terminal.

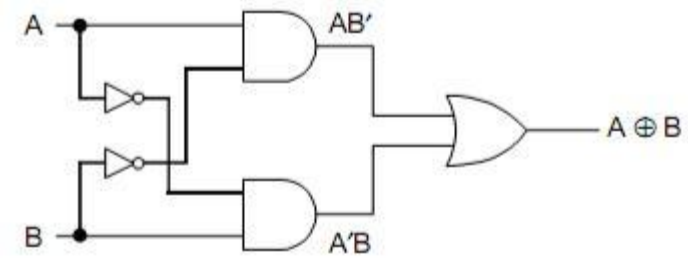


Bubbled NOR Gate: is AND gate.



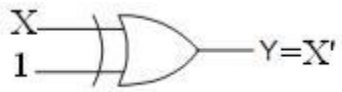
The Exclusive OR (X-OR) gate:

It has 2 inputs& only 1 output. It assumes output as 1 when input is not equal called *anti-coincidence gate or inequality detector*. The X-OR gate using AND-OR-NOT gates:

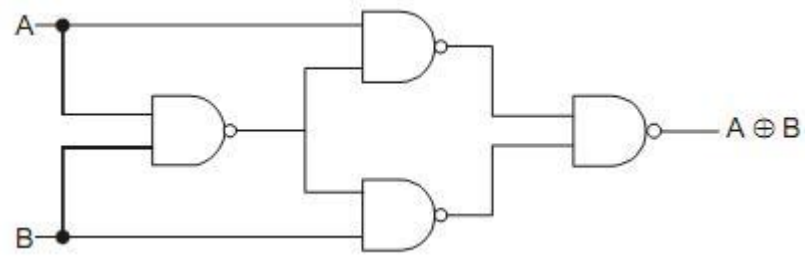


X-OR gate as an Inverter:

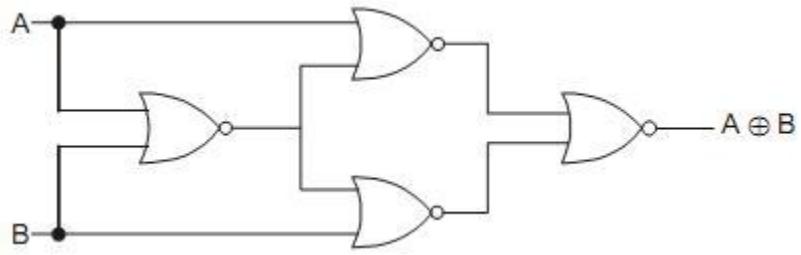
By connecting one of two input terminals to logic 1 & feeding the sequence to be inverted to other terminal



X-OR gate using NAND gates only:



X-OR gate using NOR gates only:



The EX-NOR Gate:

It is X-OR gate with a NOT gate. It has two inputs & one output logic circuit. It assumes output as 0 when one if inputs are 0 & other 1.It can be used as an equality detector because it outputs a 1 only when its inputs are equal.

$$X=A \odot B=AB+A'B'=\overline{A \oplus B}=(AB'+A'B)'$$

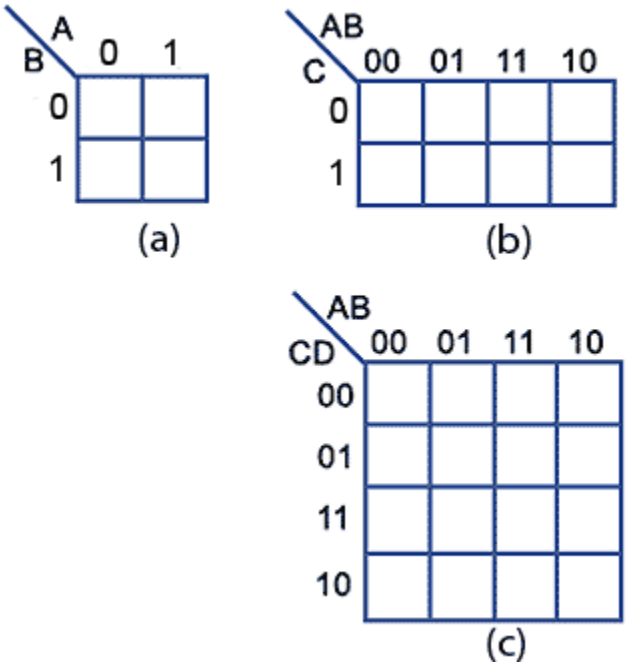
UNIT II

Gate Level Minimization

Karnaugh Maps offer a graphical method of reducing a digital circuit to its minimum number of gates. The map is a simple table containing 1s and 0s that can express a truth table or complex Boolean expression describing the operation of a digital circuit. The map is then used to work out the minimum number of gates needed, by graphical means rather than by algebra. Karnaugh maps can be used on small circuits having two or three inputs as an alternative to Boolean algebra, and on more complex circuits having up to 6 inputs, it can provide quicker and simpler minimization than Boolean algebra.

CONSTRUCTING KARNAUGH MAPS:

The shape and size of the map is dependent on the number of binary inputs in the circuit to be analyzed. The map needs one cell for each possible binary word applied to the inputs.



Therefore:

2 input circuits with inputs A and B require maps with $2^2 = 4$ cells (Fig a).

3 input circuits with inputs A B and C require maps with $2^3 = 8$ cells (Fig b).

4 input circuits with inputs A B C and D require maps with $2^4 = 16$ cells (Fig c).

The input labels are written at the top left hand corner, divided by a diagonal line. The top and left edges of the map then represent all the possible input combinations for the inputs allocated to that edge

The Karnaugh map can be populated with data from either a truth table or a Boolean equation.

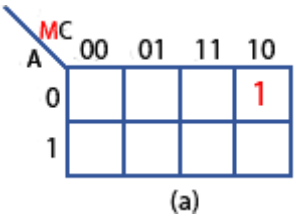
As an example, Table below shows the truth table with the Boolean expressions derived from each input combination those results in logic 1 output. This results in a Boolean equation for the un-simplified circuit:

A	M	C	X	Boolean
0	0	0	0	
0	0	1	0	
0	1	0	1	M
0	1	1	1	M•C
1	0	0	0	
1	0	1	1	A•C
1	1	0	1	A•M
1	1	1	1	A•M•C

X = M + M•C + A•C + A•M + A•C•M

Step (a)

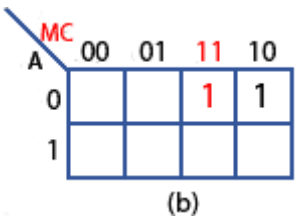
From Table , row 3, inputs AMC have values of 010, producing a logic 1 at the output (X) and giving the Boolean expression M in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=0 and MC=10 as shown at (a)



(a)

Step (b)

In Table , row 4, inputs AMC have values of 011, producing a logic 1 at the output (X) and giving the Boolean expression MC in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=0 and MC=11 as shown at (b)

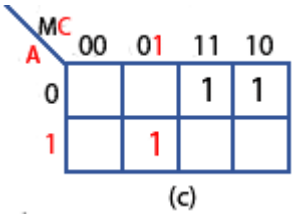


(b)

Step (c)

In Table , row 5, output (X), is 0 so this row is ignored. However, in row 6, inputs AMC have values 101, producing a logic 1 at the output (X) and giving the Boolean expression AC in the

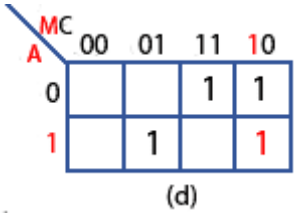
Boolean column. Therefore 1 is placed in the map cell corresponding to A=1 and MC=01 as shown at (c)



(c)

Step (d)

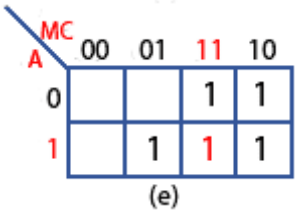
In Table, row 7, the inputs AMC have values of 110, producing a logic 1 at the output (X) and giving the Boolean expression AM in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=1 and MC=10 as shown at (d)



(d)

Step (e)

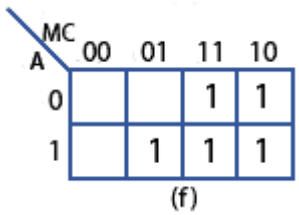
Finally, in Table ,row 8 the inputs AMC have values of 111 producing a logic 1 at the output (X) and giving the Boolean expression of AMC in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=1 and MC=11 as shown at (e)



(e)

The completed map (f)

All the truth table rows that produced a logic 1 have now been entered into the map and those lines that produced a logic 0 can be ignored, so the remaining three cells are left blank. Later it will be shown that these blank cells can be useful when mapping larger circuits, but for now the map is ready for simplification.



Simplifying Karnaugh Maps

Circuit simplification in any Karnaugh map is achieved by combining the cells containing 1 to make groups of cells. In grouping the cells it is necessary to follow six rules.

Karnaugh Map Rules

1. Groups should contain as many '1' cells (i.e. cells containing a logic 1) as possible and no blank cells.

2. Groups can only contain 1, 2, 4, 8, 16 or 32... etc. cells (powers of 2).

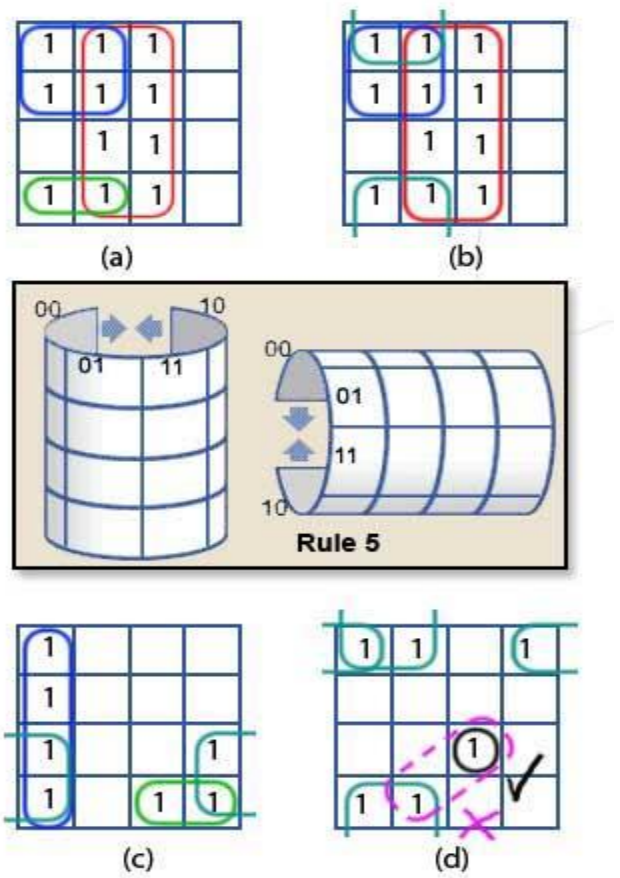
3. A '1' cell can only be grouped with adjacent '1' cells that are immediately above, below, left or right of that cell; no diagonal grouping.

4. Groups of '1' cells can overlap. This helps make smaller groups as large as possible, which is an advantage in finding the simplest solution.

5. The top/bottom and left/right edges of the map are considered to be continuous, as shown in Figure, so larger groups can be made by grouping cells across the top and bottom or left and right edges of the map.

6. There should be as few groups as possible.

Map (a) follows rules 2, 3 and 4 and shows three groups containing 8, 4 and 2 cells. This will simplify the circuit being produced, but it is not optimum.



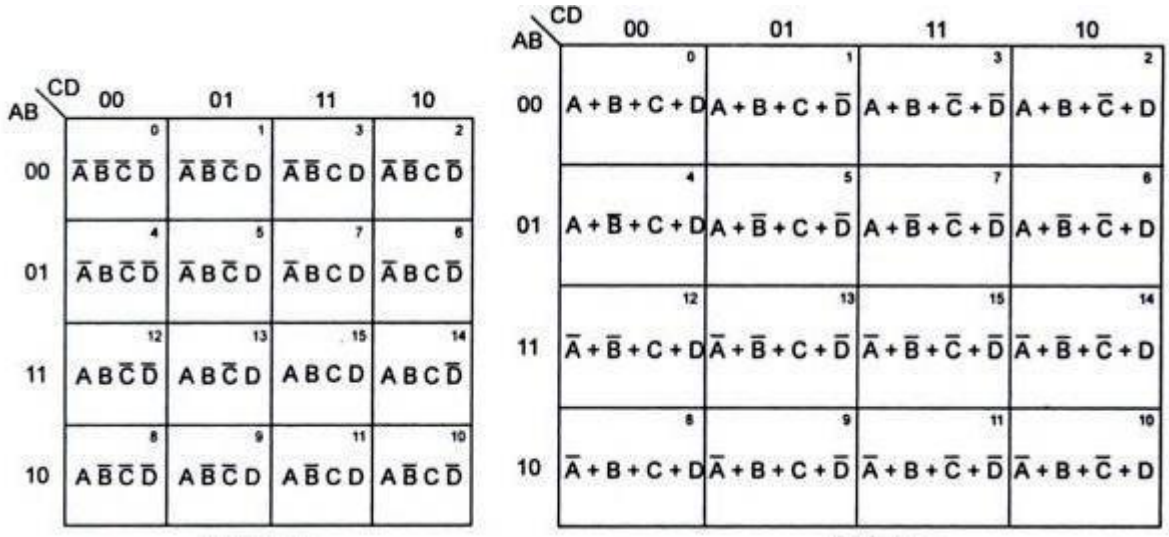
Map (b) shows an improvement, still with 3 groups but they now contain 8, 4 and 4 cells. This map takes advantage of rule 5 by joining the 2 cells ringed in green in Map (a) with the top two cells in the blue group, see Map (b) to form a group of 4 (ringed in cyan) instead of a group of 2. The map now conforms to all 6 rules.

Map (c) (for a different circuit) shows how a potentially single '1' cell (second cell from the bottom in the right hand column) can be grouped with two other cells in the blue group, and one cell in the green group, to make a (cyan) group of 4.

Sometimes however there may be a single cell that cannot be joined with other groups, as shown in map (d). Rule 3 prohibits diagonal grouping so there is no alternative other than to leave a group of 1. This is permissible, but in map (d), which represents a four input circuit, the simplified Boolean equation will contain an un-simplified expression relating to the single cell, which will have all four possible terms e.g. A•B•C•D.

Four variable k-maps:

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



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 , E,-------- ABCDE with minterms m_0, m_1-----m_{31} respectively in SOP & A+B+C+D+E, A+B+C+ ,---------- 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}.

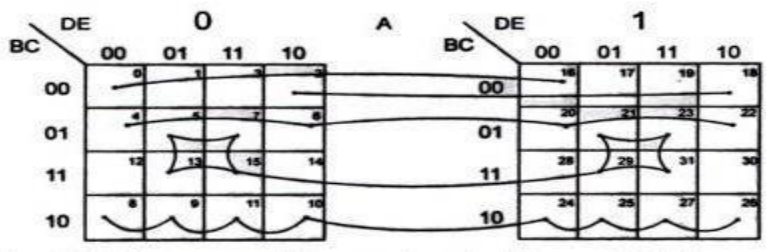
Some possible 8-squares are m_0, m_1, m_3, m_2, m_{16}, m_{17}, m_{19}, m_{18}; m_0, m_4, m_{12}, m_8, m_{16}, m_{20}, m_{28}, m_{24}; m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31}.

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

Grouping S is

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

$M_0, M_{16} = B + C + D + E$
$M_2, M_{18} = B + C + \overline{D} + E$
$M_4, M_6, M_{20}, M_{22} = B + \overline{C} + E$
$M_5, M_7, M_{13}, M_{15}, M_{21}, M_{23}, M_{29}, M_{31} = \overline{C} + \overline{E}$
$M_8, M_9, M_{10}, M_{11}, M_{24}, M_{25}, M_{26}, M_{27} = \overline{B} + C$



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)

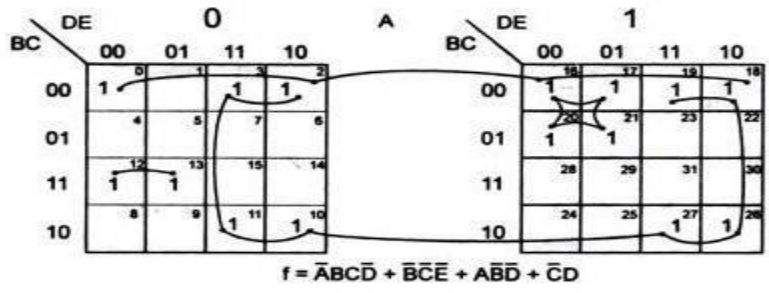
The real minimal expression is the minimal of the SOP and POS forms.

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)



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

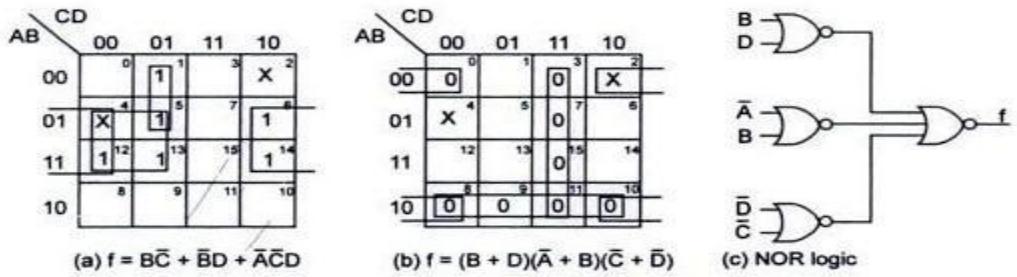
combinations for which the value of expressions is not specified are called don‘t care combinations or Optional Combinations, such expressions stand incompletely specified. The output is a don‘t care for these invalid combinations.

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) = > f=π M(0,3,7,9,10,11,15).πd(2,4)

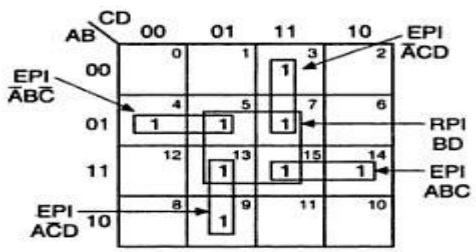


Prime implicants, Essential Prime implicants, Redundant prime implicants:

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)= CD+ABC+A D + B



<u>OTHER TWO LEVEL IMPLEMENTATIONS:</u>

- 2^{2n} Possible Switching Functions of n Variables (actually much fewer types obtained by permuting and/or complementing input variables)
- 1-Level Forms
  - Cannot Represent all Possible Functions
- 2-Level Forms
  - Can Represent all Possible Functions
  - With Additional Restrictions – ***CANONICAL***
- k-Level Forms ($k \geq 2$)
  - Many Different Ways to Represent a Given Functions
- If a multi-input Gate Represents a (binary or greater) Boolean Operator
  - Expression can Represent a Netlist
  - k Indicates "depth" of Netlist
- ***1-Level Forms***
  - Due to Restriction of Representable Functions, Not as Useful
- ***2-Level Forms***
  - Can be Canonical
  - Longest PI to PO Path is Always 2 Related to Delay
  - Minimization Usually Means
    1. Minimum Number of "Terms"
    2. Minimum Number of "Literals" in Expression
- **k-Level Forms (k≥2)**
  - Many More Possible Representations
  - Can be Optimized for Area
  - Delay is More Complicated
    1. False Path Problem
    2. Spatio-temporal Correlations
    3. Typically an Area versus Delay Tradeoff (e.g., Ripple versus CL Adder)

Common Terms for 2-Level Minimization

- Literal – A variable in complemented or uncomplemented form
- Product – The disjunction (AND) of a set of literals; also represents a cube
- Support Set – Set of all variables that define the domain of a switching function
- Minterm – A disjunction (AND) containing an instance of each literal corresponding to a variable in the support set that is in the on-set, f^{on},of a function
- Don't Care – The absence of a supporting variable in a product term

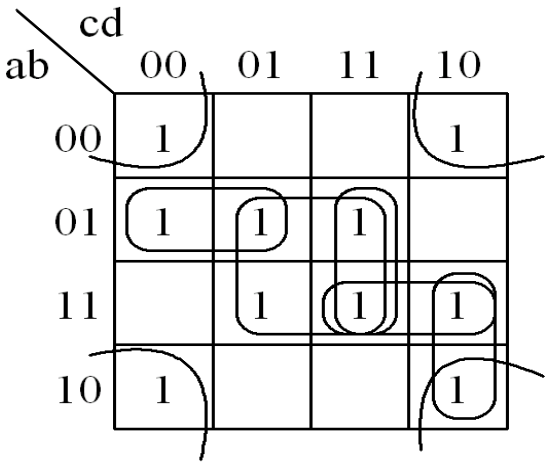
- Implicant – A product term that covers one or more minterms in the on-set, f^{on}, of a function
- Prime Implicant – An implicant in the on-set, f^{on}, of a function such that it is not a subproduct of any other possible implicant in the set.

Essential Prime Implicant – A prime implicant that covers at least one minterm ***NOT*** covered by any other implicant in the on-set, f^{on}.

SOP Minimization – Terms Example

Consider, $f^{on} = \{\bar{b}\bar{d}, \bar{a}b\bar{c}, bd, ac\bar{d}, abc, bcd\}$

- ✓ Each Element of f^{on} is an Implicant
- ✓ Prime Implicants are: $\{\bar{b}\bar{d}, \bar{a}b\bar{c}, bd, ac\bar{d}, abc\}$
- ✓ abc is a Prime Implicant but **NOT** an Essential Prime Implicant
- ✓ bcd is an Implicant but **NOT** Prime Implicant
- ✓ bd is an Essential Prime Implicant
- ✓ Product *abc* covers minterms: $m_4=abcd$ and $m_5=abcd$, disjunction of literals *a*, *b*, *c*, variable *d* is a Don't Care
- ✓ *F* Represented by f^{on} has Support Set: {*a, b, c, d*}



TABULAR METHOD:

The tabular method which is also known as the Quine-McCluskey method is particularly useful when minimising functions having a large number of variables, e.g. The six-variable functions. Computer programs have been developed employing this algorithm. The method reduces a function in standard sum of products form to a set of prime implicants from which as many

variables are eliminated as possible. These prime implicants are then examined to see if some are redundant. The tabular method makes repeated use of the law A + $\overline{A}$= 1. Note that Binary notation is used for the function, although decimal notation is also used for the functions. As usual a variable in true form is denoted by 1, in inverted form by 0, and the abscence of a variable by a dash (-).

RULES OF TABULAR METHOD:

Consider a function of three variables f(A, B, C):

$\overline{A}$ B $\overline{C}$ is represented by 011 ← Binary notation, where A = 0, B= 1 and C= 1
A $\overline{B}$ $\overline{C}$ is represented by 100
A $\overline{C}$ is represented by 1–0
B C is represented by –11

Consider the function:

$$f(A, B, C, D) = \sum(1110, 1111) = A\ B\ C\ \overline{D} + A\ B\ C\ D = A\ B\ C$$

Listing the two minterms shows they can be combined

```
A B C D
1 1 1 0 ←┐
1 1 1 1 ←┘ Can combine (Differs in one digit position)
1 1 1 –
```

Now consider the following:

$$f(A, B, C, D) = \sum(1101, 1110) = A\ B\ \overline{C}\ D + A\ B\ C\ \overline{D}$$

Note that these variables cannot be combined

```
A B C D
1 1 0 1 ←┐
1 1 1 0 ←┘ Cannot combine (Differs in more than one digit position)
    ?
```

This is because the *FIRST RULE* of the Tabular method for two terms to combine, and thus eliminate one variable, is that they must differ in only **one digit** position.

Bear in mind that when two terms are combined, one of the combined terms has one digit more at logic 1 than the other combined term. This indicates that the number of 1's in a term is significant and is referred to as its index.

For example: f(A, B, C, D)

0000.....Index 0
0010, 1000.....Index 1
1010, 0011, 1001.....Index 2
1110, 1011.....Index 3
1111.....Index 4

The necessary condition for combining two terms is that the indices of the two terms must differ by one logic variable which must also be the same.

Example 1:

Consider the function: Z = f(A,B,C) = $\overline{A}\overline{B}\overline{C}$+ $\overline{A}\overline{B}$C + A$\overline{B}\overline{C}$ + A$\overline{B}$C

To make things easier, change the function into binary notation with index value and decimal value.

f(A, B , C) = $\sum$ (000, 001, 100, 101)————Binary notation

0 1 1 2————Index

0 1 4 5————Decimal value

Tabulate the index groups in a colunm and insert the decimal value alongside.

	First List				Second List				Third List			
		A	B	C		A	B	C		A	B	C
Index 0 →	0	0	0	0 ✓	0,1	0	0	– ✓	0,1,4,5	–	0	–
	1	0	0	1 ✓	0,4	–	0	0 ✓	0,4,1,5	–	0	–
Index 1 →	4	1	0	0 ✓	1,5	–	0	1 ✓				
Index 2 →	5	1	0	0 ✓	4,5	1	0	– ✓				

From the first list, we combine terms that differ by 1 digit only from one index group to the next. These terms from the first list are then seperated into groups in the second list. Note that the ticks are just there to show that one term has been combined with another term. From the second list we can see that the expression is now reduced to: Z = $\overline{A}\overline{B}$+ $\overline{B}\overline{C}$+ $\overline{B}$C + A$\overline{B}$

From the second list note that the term having an index of 0 can be combined with the terms of index 1. Bear in mind that the dash indicates a missing variable and **must** line up in order to get a third list. The final simplified expression is: Z = $\overline{B}$

Bear in mind that any unticked terms in any list must be included in the final expression (none occured here except from the last list). Note that the only prime implicant here is Z = $\overline{B}$.

The tabular method reduces the function to a set of prime implicants.

Note that the above solution can be derived algebracially. Attempt this in your notes.

Example 2:

Consider the function f(A, B, C, D) = $\overline{\sum}$(0,1,2,3,5,7,8,10,12,13,15), note that this is in decimal form.

$\sum$(0000,0001,0010,0011,0101,0111,1000,1010,1100,1101,1111) in binary form.

(0,1,1,2,2,3,1,2,2,3,4) in the index form.

The chart is used to remove redundant prime implicants. A grid is prepared having all the prime implicants listed at the left and all the minterms of the function along the top. Each minterm covered by a given prime implicant is marked in the appropriate position.

First List

	A	B	C	D	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
8	1	0	0	0	✓
3	0	0	1	1	✓
5	0	1	0	1	✓
10	1	0	1	0	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
13	1	1	1	1	✓
15	1	1	1	1	✓

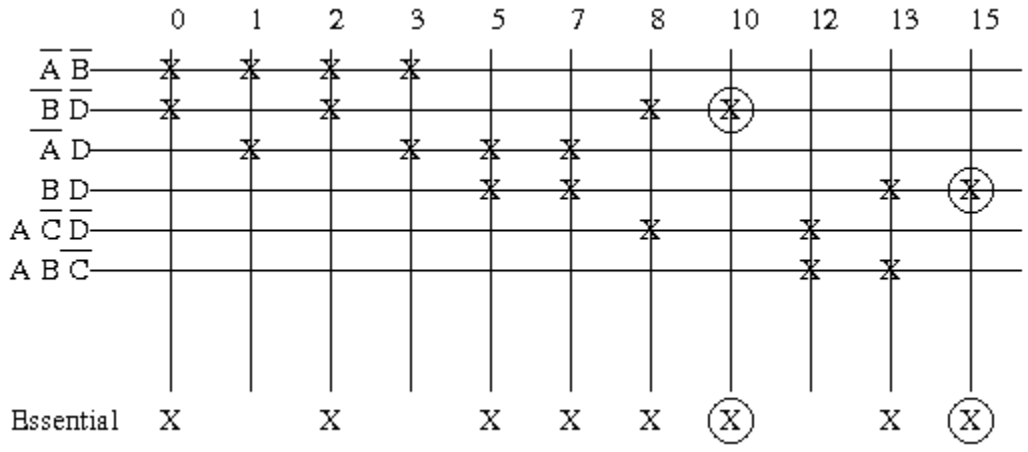
Second List

	A	B	C	D	
0,1	0	0	0	–	✓
0,2	0	0	–	0	✓
0,8	–	0	0	0	✓
1,3	0	0	–	1	✓
1,5	0	–	0	1	✓
2,3	0	0	1	–	✓
2,10	–	0	1	0	✓
8,10	1	0	–	0	✓
8,12	1	–	0	0	$A\overline{C}\overline{D}$
3,7	0	–	1	1	✓
5,7	0	1	–	1	✓
5,13	–	1	0	1	✓
12,13	1	1	0	–	$AB\overline{C}$
7,15	–	1	1	1	✓
13,15	1	1	–	1	✓

Third List

	A	B	C	D	
0, 1 2 3	0	0	–	–	$\overline{A}\,\overline{B}$
0, 2 1 3	0	0	–	–	
0, 2, 8, 10	–	0	–	0	$\overline{B}\,\overline{D}$
0, 8, 2, 10	–	0	–	0	
1, 3, 5, 7	0	–	–	1	$\overline{A}D$
1, 5, 3, 7	0	–	–	1	
5, 7, 13, 15	–	1	–	1	BD
5, 13, 7, 15	–	1	–	1	

The prime implicants are: $\overline{A}\overline{B}+\overline{B}\overline{D}+\overline{A}D+BD+A\overline{C}\overline{D}+AB\overline{C}$



From the above chart, BD is an essential prime implicant. It is the only prime implicant that covers the minterm decimal 15 and it also includes 5, 7 and 13. $\overline{B}\overline{D}$is also an essential prime implicant. It is the only prime implicant that covers the minterm denoted by decimal 10 and it also includes the terms 0, 2 and 8. The other minterms of the function are 1, 3 and 12. Minterm 1 is present in $\overline{A}\overline{B}$and $\overline{A}D$. Similarly for minterm 3. We can therefore use either of these prime

implicants for these minterms. Minterm 12 is present in $A\overline{C}\overline{D}$ and $AB\overline{C}$, so again either can be used.

Thus, one minimal solution is: $Z = \overline{B}\overline{D} + BD + \overline{A}\overline{B} + A\overline{C}\overline{D}$

UNIT III

Combinational Logic Circuits

3.1 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.

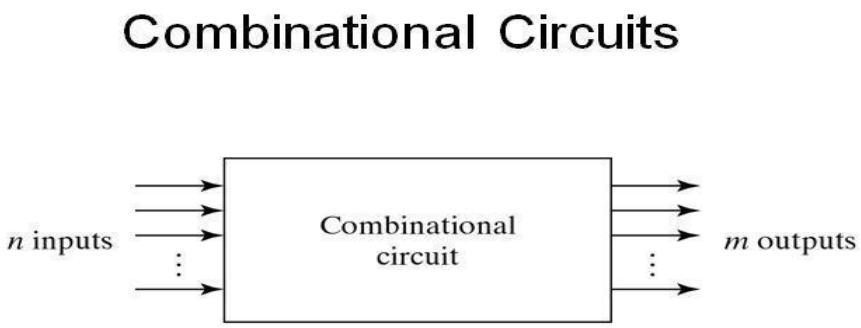


Fig. Block Diagram of Combinational Circuit

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

Binary Addition:

Rules:

0+0=0
0+1=1
1+0=1
1+1=10 i.e, 0 with a carry of 1.

Example: add binary no.s 1101.101 & 111.011

8421 2^{-1} 2^{-2} 2^{-3}
1101.101
111.011
_____
10101.000

In 2^{-3} column 1+1=0 with a carry of 1 to the 2^{-2} column
In 2^{-2} column 0+1+1=0 2^{-1}
1 1+0+1=0 1‘s
2 1+1+1=1 2‘s
4 0+1+1=0 4‘s
8 1+1+1=1 8‘s
16 1+1 =0 16‘s

Binary Subtraction:

Rules: 0-0=0
1-1=0
1-0=1
0-1=1with a borrow of 1

Example: subtract binary no.s

8421 2^{-1} 2^{-2} 2^{-3}
1010.010
111.111

0010.011

In 2^{-3} column 10-1=1

DECIMAL ADDER:

Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary coded form. An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code. For binary addition, it is sufficient to consider a pair of significant bits together with a previous carry. A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input and output carry , the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry. Suppose we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19. These binary numbers are labeled by symbols K, Z8, Z4, Z2, and Z1. K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

When the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit that detects the necessary correction can be derived from the entries in the table.

It is obvious that a correction is needed when the binary sum has an output carry K = 1.

The other six combinations from 1010 through 1111 that need a correction have a 1 in position Z8. To distinguish them from binary 1000 and 1001, which also have a 1 in position Z8, we specify further that either Z4 or Z2 must Have a 1.

The condition for a correction and an output carry can be expressed by the Boolean function

$C = K + Z8.Z4 + Z8.Z2$

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

MAGNITUDE COMPARATOR

Another common and very useful combinational logic circuit is Digital **Comparator** circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean algebra. There are two main types of Digital Comparator available and these are.

- Identity Comparator – an Identity Comparator is a digital comparator that has only one output terminal for when A = B either "HIGH" A = B = 1 or "LOW" A = B = 0
- Magnitude Comparator – a Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …. An, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. Bn, etc) and produce an output condition or flag depending upon the result

of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$$A > B, \quad A = B, \quad A < B$$

Which means: A is greater than B, A is equal to B, and A is less than B

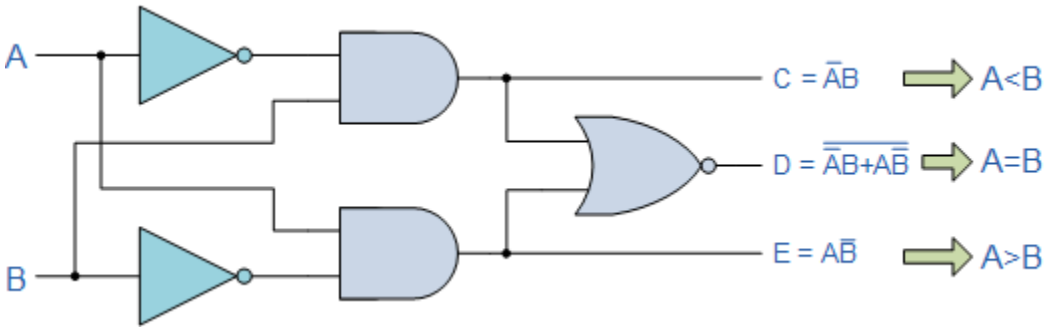
This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

1-BIT DIGITAL COMPARATOR CIRCUIT:

Notice the two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = Bresembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the n-bits giving: $Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the "magnitude" of these values, a logic "0" against a logic "1" which is where the term **Magnitude Comparator** comes from.

As well as comparing individual bits, we can design larger bit comparators by cascading together n of these and produce a n-bit comparator just as we did for the n-bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

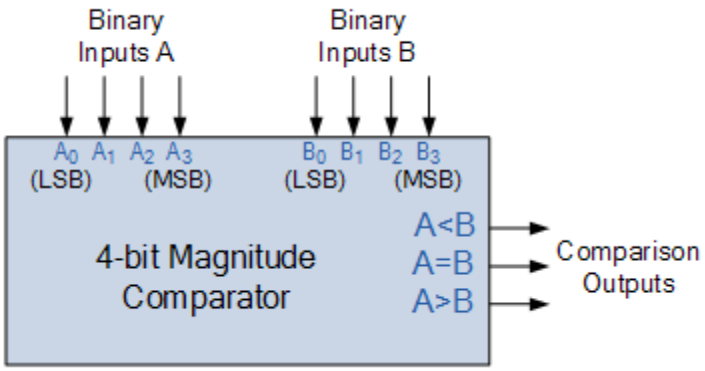


Then the operation of a 1-bit digital comparator is given in the following Truth Table.

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

4-bit Magnitude Comparator



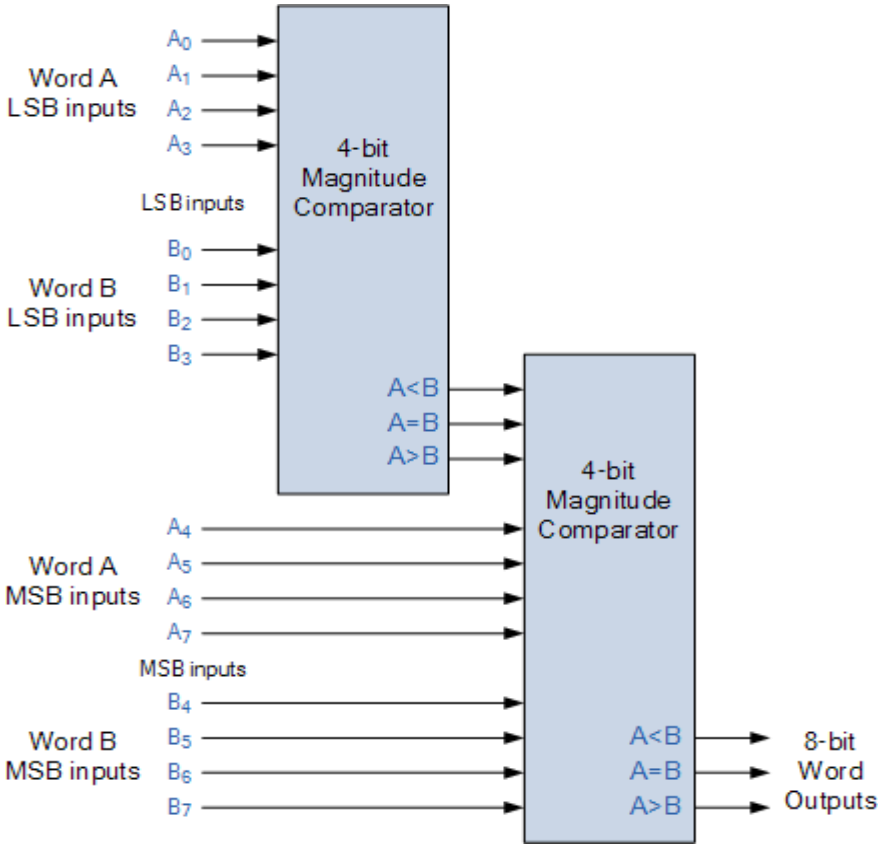
Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

8 BIT WORD COMPARATOR:

When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = B then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.

If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. **Digital**

Comparator is used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.



DECODERS:

The basic function of a decoder is to detect the presence of a specified combination of bits on its inputs and to indicate that presence by a specified output level. A decoder has n input lines o handles n bits and from one to 2^noutput lines to indicate the presence of one or more n – bit combinations.

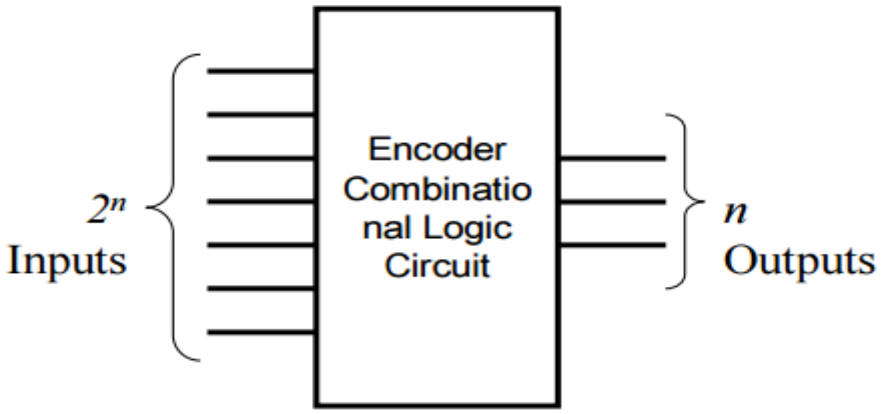
The truth table for 3 lines to 8 line decoder:

Inputs			Outputs							
a	b	c	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

ENCODERS:

An encoder is the combinational circuit which performs a reverse function that of decoder. Encode inputs are decimal digits and/or alphabetic characters and outputs are coded representation of these inputs.

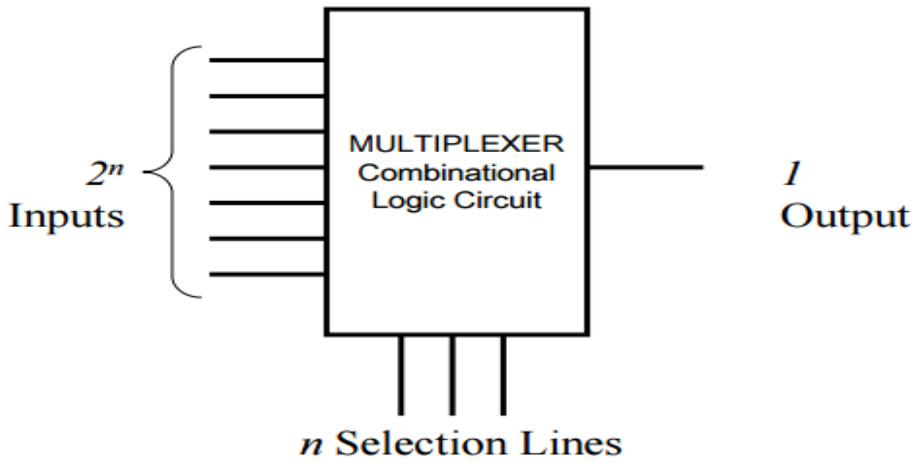
The decimal to BCD encoder has 10 inputs, one representing each decimal digit and four outputs. The truth table for 10 lines to 4 line



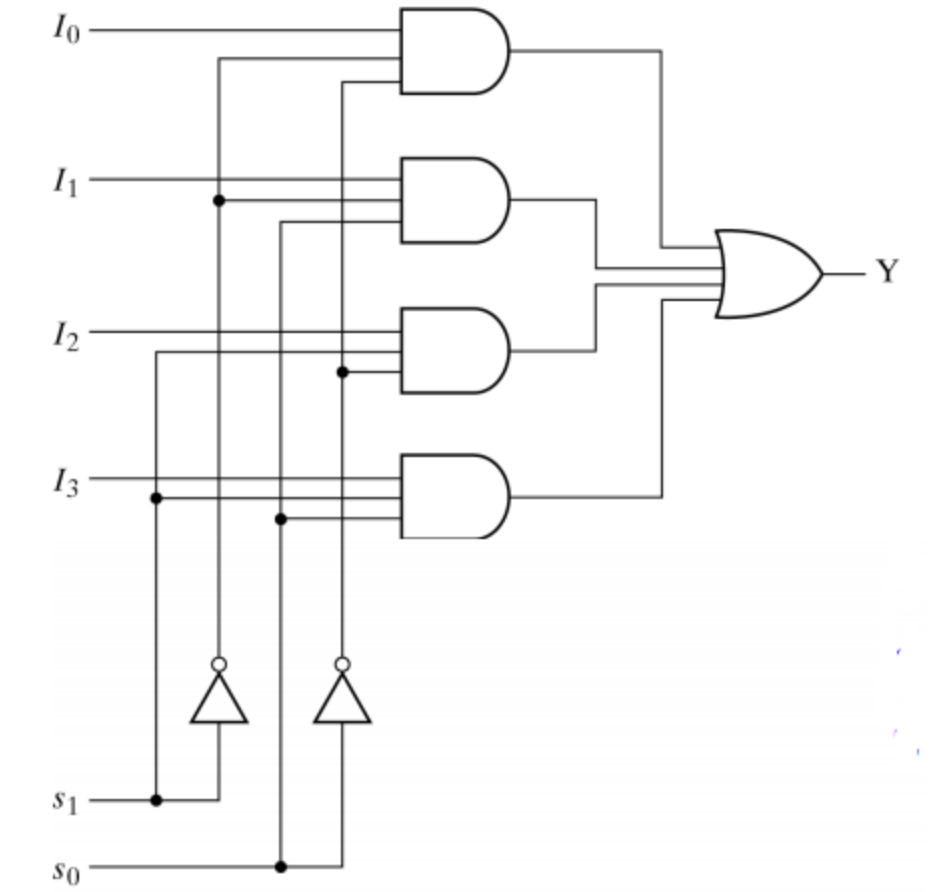
Decimal Digits	BCD Code			
	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

MULTIPLEXERS:

A multiplexer or Data Selector is a combinational circuit that selects binary information from one of many input lines and directs the information to a single output line. The selection of a particular input line is controlled by a set of selection lines. For 2^n data line we have n selection lines.



4 TO 1 LINE MULTIPLEXER:



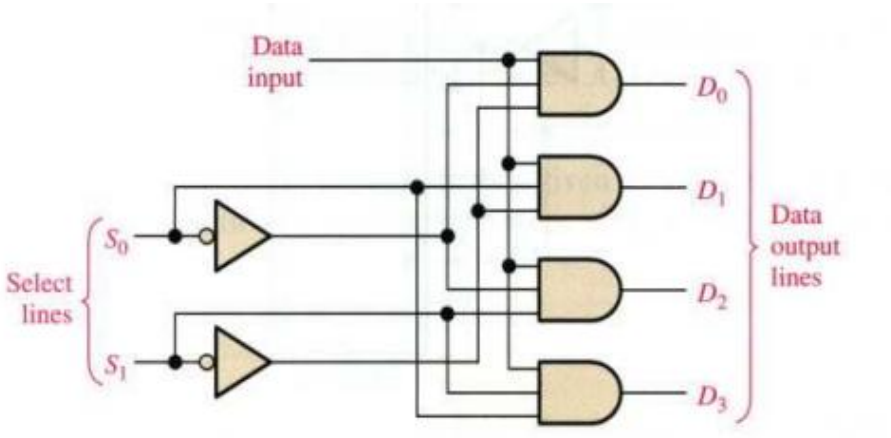
s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

An electronic multiplexer can be considered as a multiple-input, single-output switch, and a demultiplexer as a single-input, multiple-output switch.The schematic symbol for a multiplexer is an isosceles trapezoid with the longer parallel side containing the input pins and the short parallel side containing the output pin.

Demultiplexers

It performs the inverse operation of a multiplexer .A combinational circuit that receives input from a single line and transmits it to one of 2 n possible output lines. The selection of the specific output is controlled by the bit combination of n selection lines



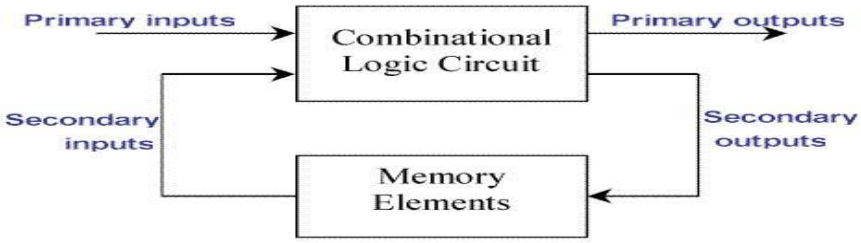
UNIT IV

Sequential Logic Circuits

4.1 Sequential circuits

Combinational logic refers to circuits whose output strictly depends on present value of the inputs. As soon as inputs are changed, the information about the previous inputs is lost, i.e.,

combinational logics circuits have no memory. Although every digital system is likely to have combinational circuits, most systems 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 referred as **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

Classification of sequential circuits:

Sequential circuits may be classified as two types.

1. Synchronous sequential circuits
2. Asynchronous sequential circuits

Level mode and pulse mode asynchronous sequential circuits:

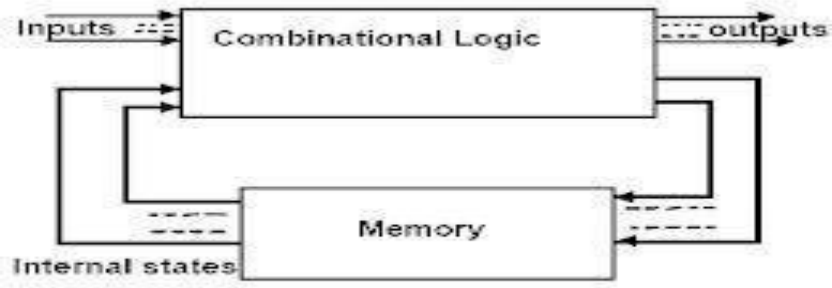
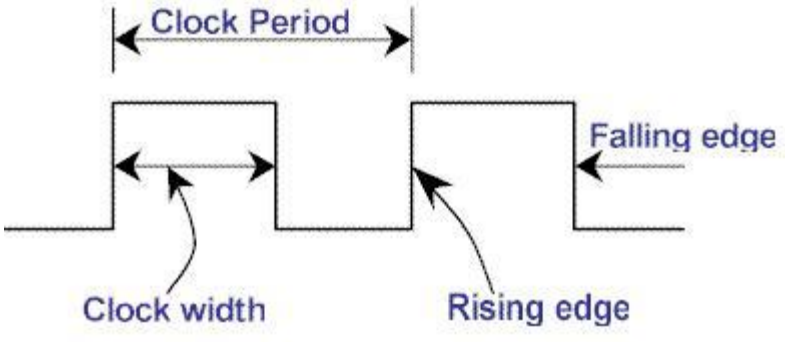


Figure 1: Asynchronous Sequential Circuit

Figure shows a block diagram of an asynchronous sequential circuit. It consists of a combinational circuit and delay elements connected to form the feedback loops. 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, 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

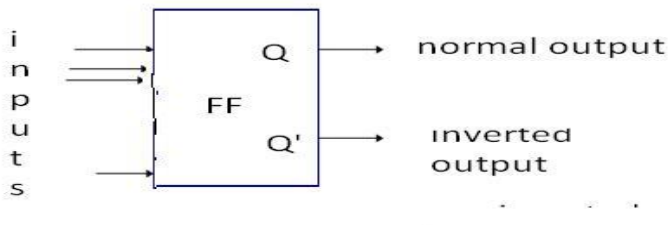
high if the state changes occur at the clock's rising edge or during the clock width. Otherwise, the clock is said to be active low. Synchronous sequential circuits are also known as clocked sequential circuits.

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 is 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.

4.2 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.



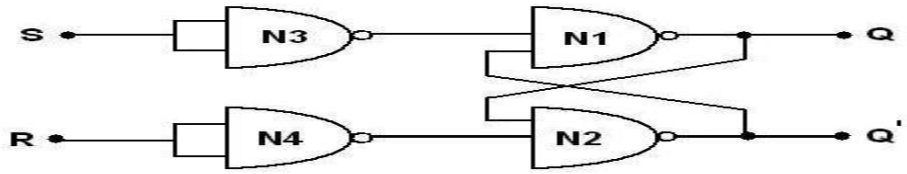
A latch may be an active-high input latch or an active –LOW input latch.active –HIGH means that the SET and RESET inputs are normally resting in the low state and one of them will be pulsed high whenever we want to change latch outputs.

SR latch:

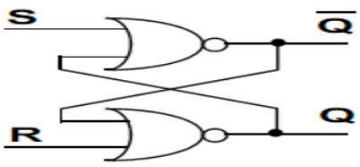
The latch has two outputs Q and Q‘. When the circuit is switched on the latch may enter into any state. If Q=1, then Q‘=0, which is called SET state. If Q=0, then Q‘=1, which is called RESET state. Whether the latch is in SET state or RESET state, it will continue to remain in the same state, as long 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 operation of NAND latch is the reverse of the operation of NOR latches. 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

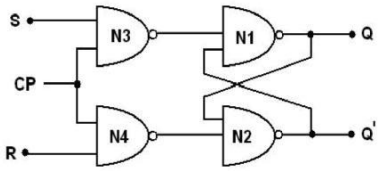
The analysis of the operation of the active-HIGHNOR latch can be summarized as follows.

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

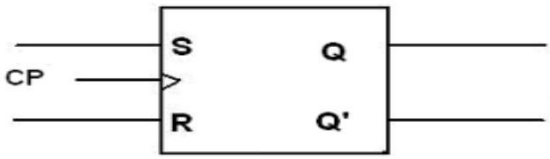
3. SET=0, RESET=1: this will always reset Q=0, where it will remain even after RESET returns to 0
4. SET=1,RESET=1; this condition tries to SET and RESET the latch at the same time, and it produces Q=Q‘=0. If the inputs are returned to zero simultaneously, the resulting output stste is erratic and unpredictable. This input condition should not be used.
5. 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.



a) Logic diagram



b) Block diagram

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:

- 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.
- 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.
- CP=1, S=1, R=0 gives the SET state i.e., Q=1, Q‘=0.
- CP=1, S=0, R=1 gives the RESET state i.e., Q=0, Q‘=1.
- CP=1, S=0, R=0 does not affect the state of flip-flop.
- 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‖.

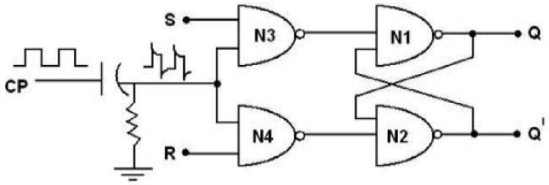
In the logic symbol CP input is marked with a triangle. It indicates the circuit responds to an input change from 0 to 1. The characteristic table gives the operation conditions of flip-flop. Q(t) is the present state maintained in the flip-flop at time _t'. Q(t+1) is the state after the occurrence of clock pulse.

Truth table

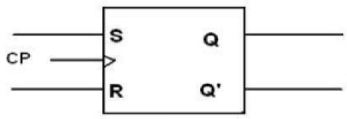
S	R	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	*	Not allowed

Edge triggered RS flip-flop:

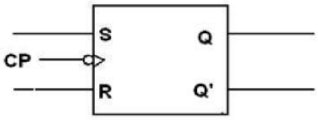
Some flip-flops have an RC circuit at the input next to the clock pulse. By the design of the circuit the R-C time constant is much smaller than the width of the clock pulse. So the output changes will occur only at specific level of clock pulse. The capacitor gets fully charged when clock pulse goes 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.



a) Logic diagram of edge triggered RS flip-flop



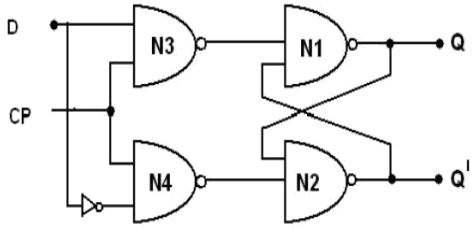
b) Block diagram of positive edge triggered flip-flop



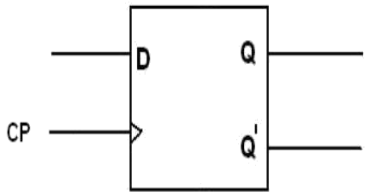
c) Block diagram of negative edge triggered flip-flop

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



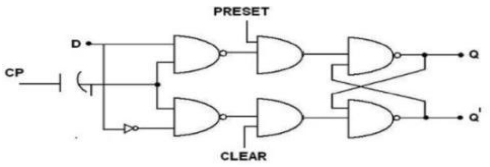
b) Block diagram

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

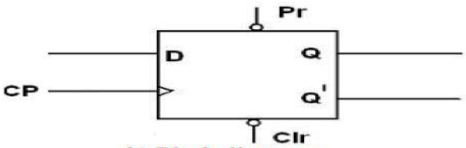
Truth table

CP	D	Q
0	x	Previous state
1	0	0
1	1	1

Edge Triggered D Flip-flop:



a) Logic diagram

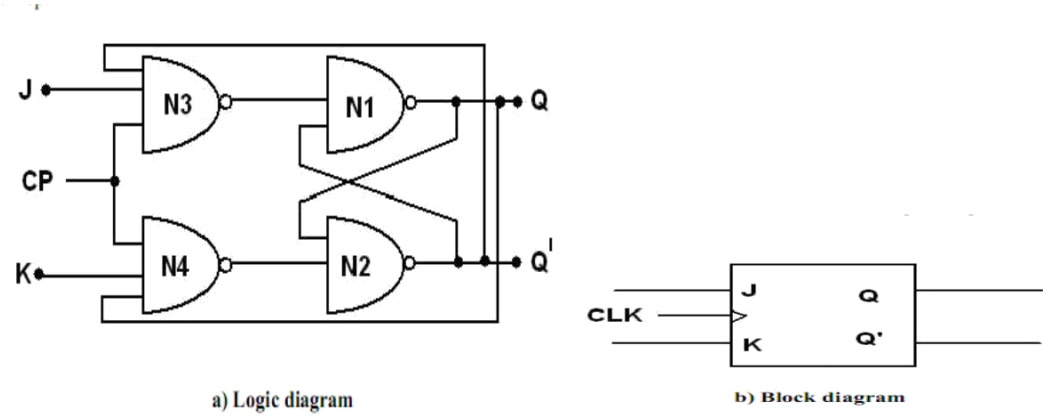


b) Block diagram

Truth table

PRESET	CLEAR	CP	D	Q
0	0	X	X	*(forbidden)
0	1	X	X	1
1	0	X	X	0
1	0	0	X	NC
1	1	1	X	NC
1	1	↓	X	NC
1	1	↑	0	0
1	1	↑	1	1

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 below 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.

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 N4is 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

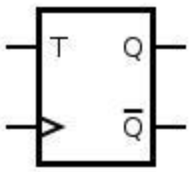
$$Q_{next} = J\overline{Q} + \overline{K}Q$$

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



$$Q_{next} = T \oplus Q = T\overline{Q} + \overline{T}Q$$

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).

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).

$P=V_{cc}.I_{cc}$

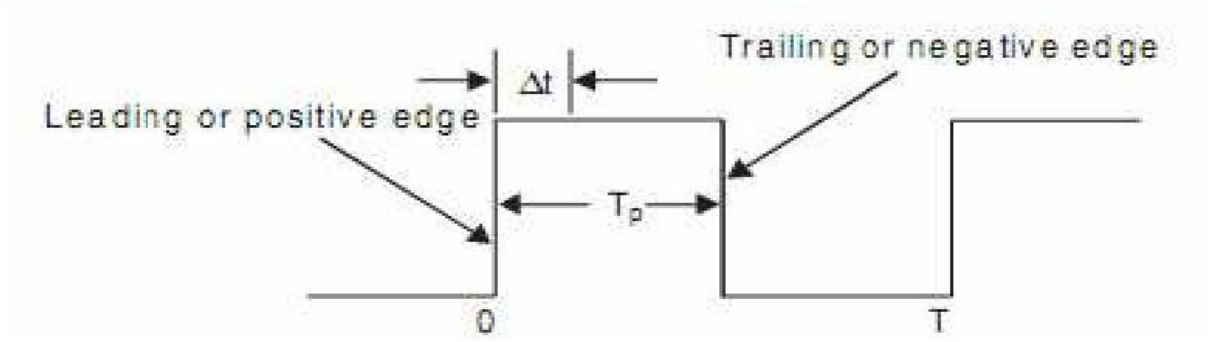
The power dissipation of a flip flop is usually in mW.

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.

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.

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

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 of the circuit.

Mealy machine: The outputs of the circuit depend upon both the current state of the circuit and the inputs.

An example to go through the steps

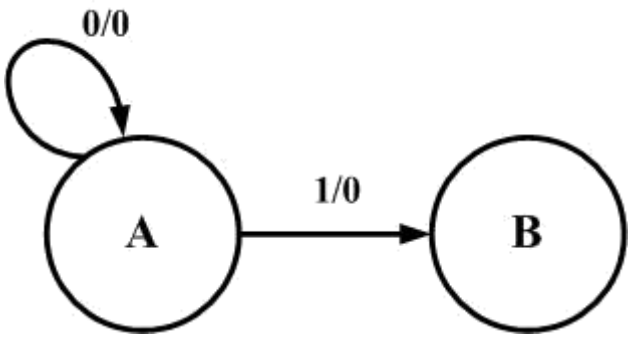
The specification: The circuit will have one input, X, and one output, Z. The output Z will be 0 except when the input sequence 1101 are the last 4 inputs received on X.

Generation of a state diagram

- Create states and meaning for them.

State A – the last input was a 0 and previous inputs unknown. State B – the last input was a 1 and the previous input was a 0. Possibly.

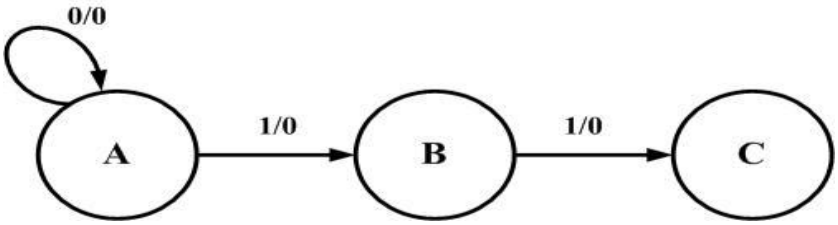
Capture this in a state diagram



- Circles represent the states
- Lines and arcs represent the transition between states.

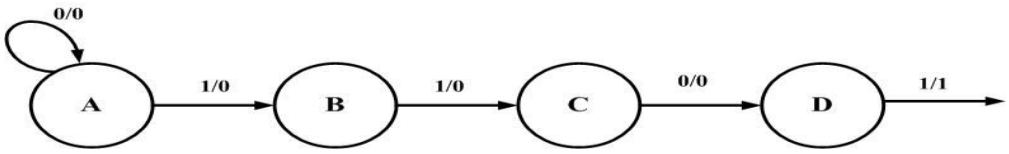
- The notation Input/output on the line or arc specifies the input that causes this transition and the output for this change of state.

Add a state C – Have detected the input sequence 11 which is the start of the sequence

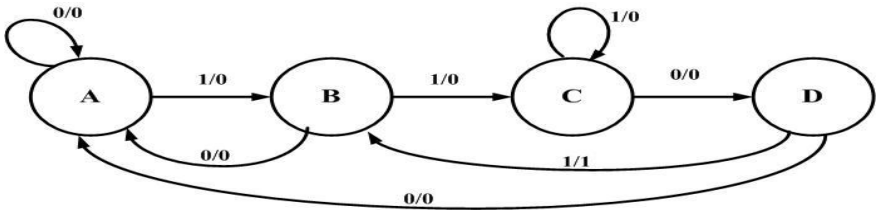


- Add a state D

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



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

Shift registers:

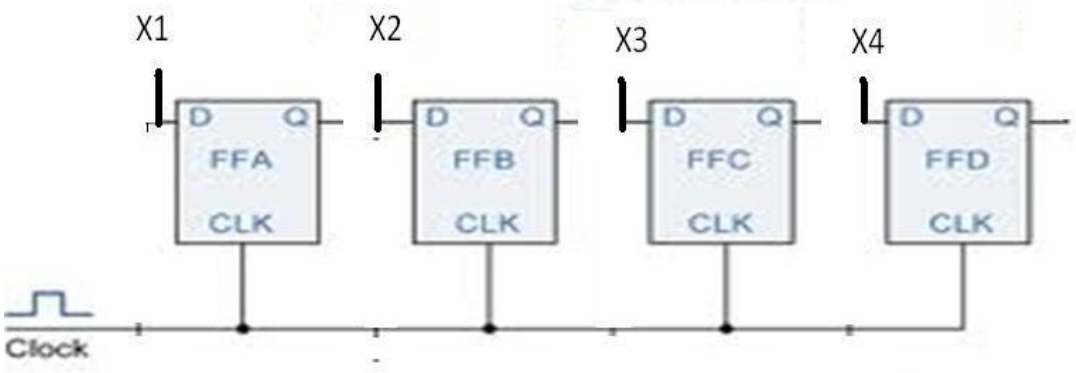
In digital circuits, a **shift register** is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, *shifting in* the data 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.

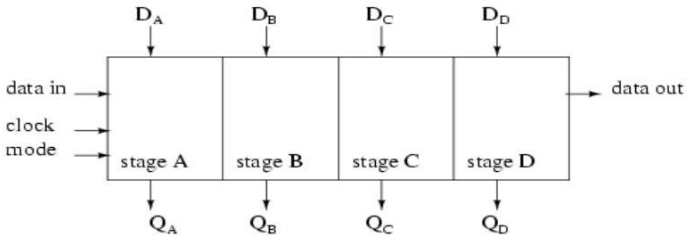


The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.

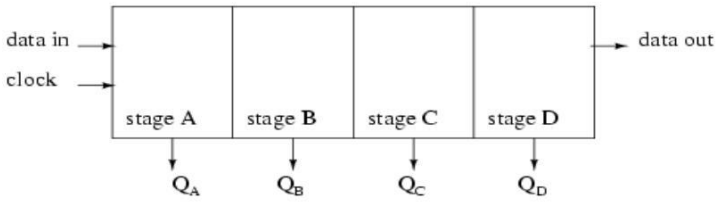
Controlled buffer register:

If goes LOW, all the FFs are RESET and the output becomes, Q=0000. When 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.

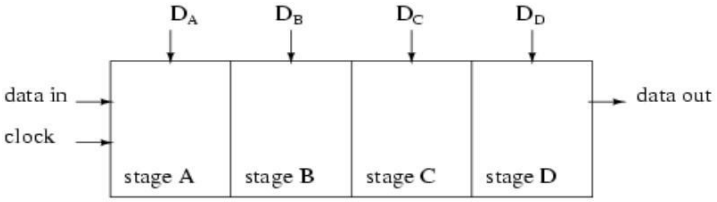
Data transmission in shift registers:



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



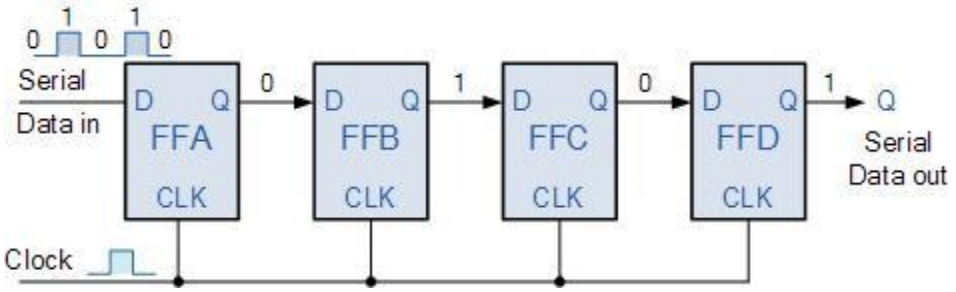
Parallel-in, serial-out shift register with 4-stages

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

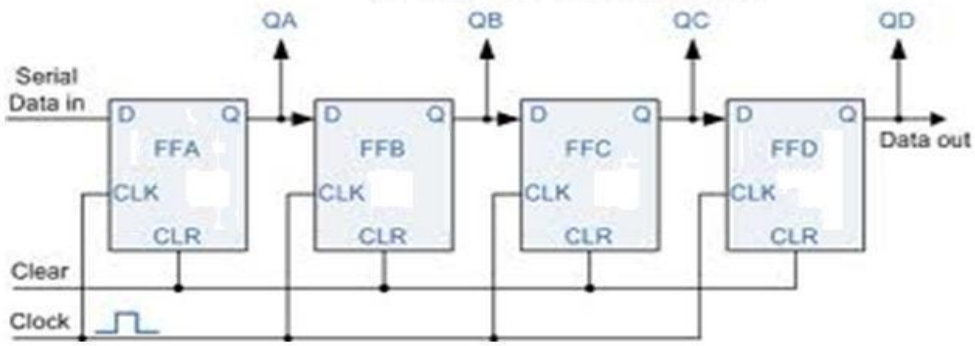
Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages is as shown. 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.



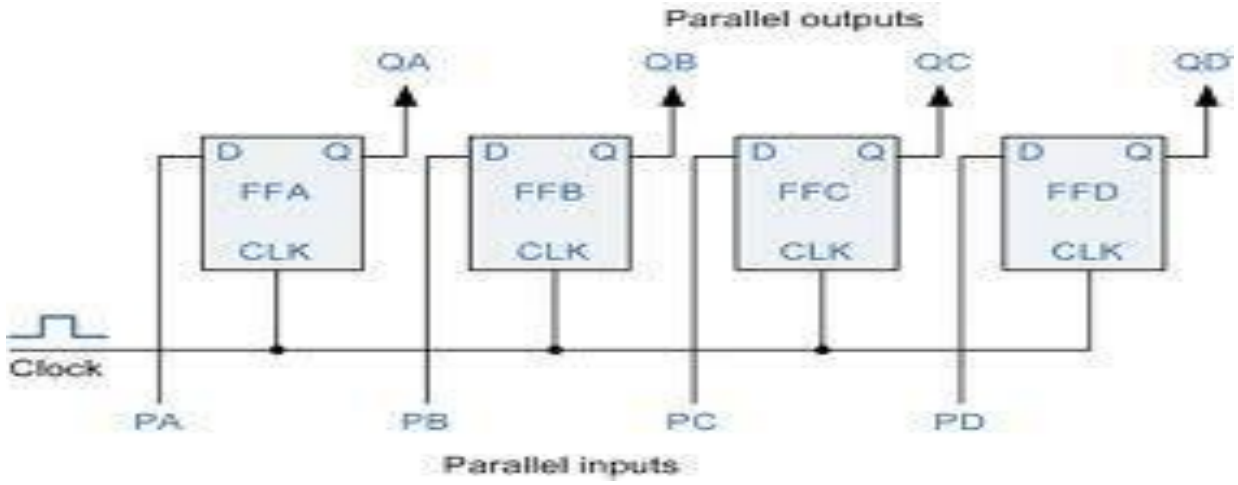
When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

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.

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.

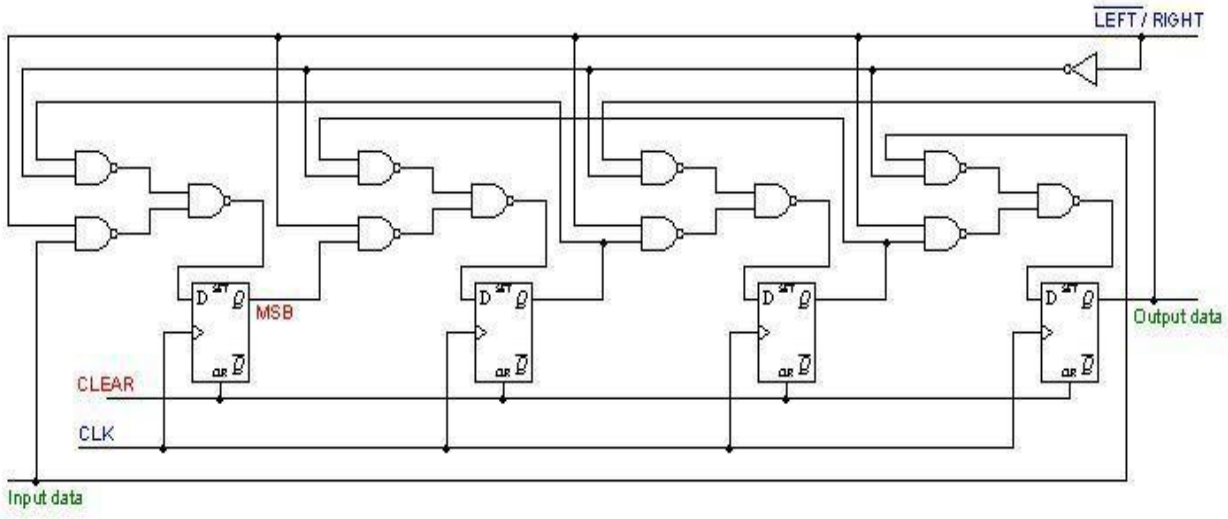
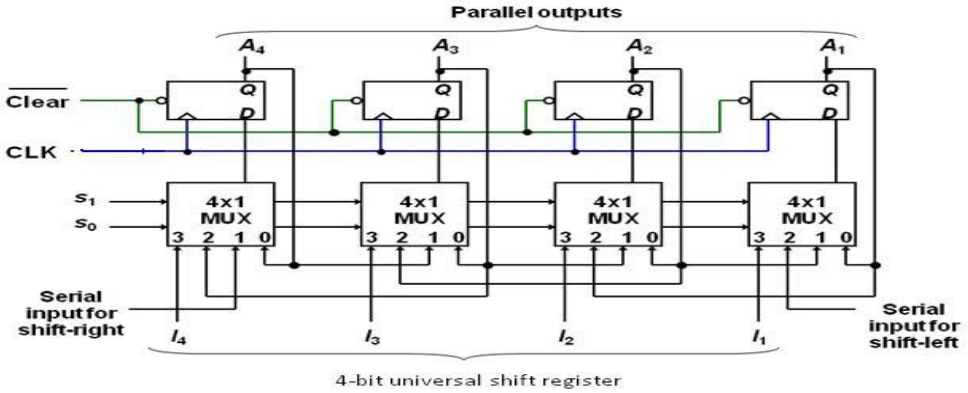


Figure: logic diagram of a 4-bit bidirectional shift register

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.

A universal shift register can be realized using multiplexers. Figure shows the logic diagram



Function table for the register

		mode control
S0	**S1**	**register operation**
0	0	No change
0	1	Shift Right
1	0	Shift left
1	1	Parallel load

of a 4-bit universal shift register. 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.

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 single clock
- 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:

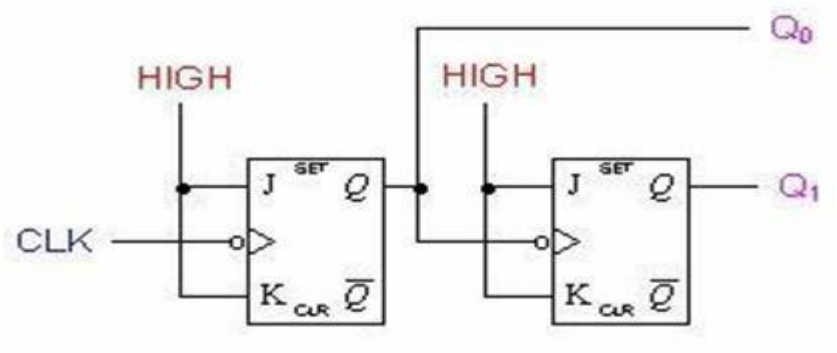
An asynchronous (ripple) counter is a single JK-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two 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:

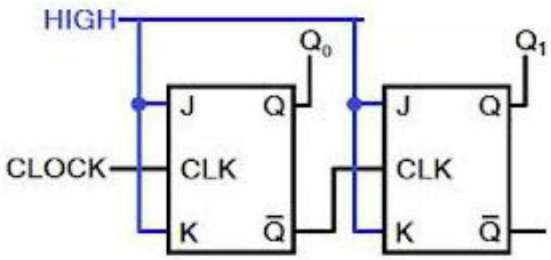
Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.The counter is initially assumed to be at a state 00 where the outputs of the tow flip-flops are noted as Q1Q0. Where Q1 forms the MSB and Q0 forms the LSB.

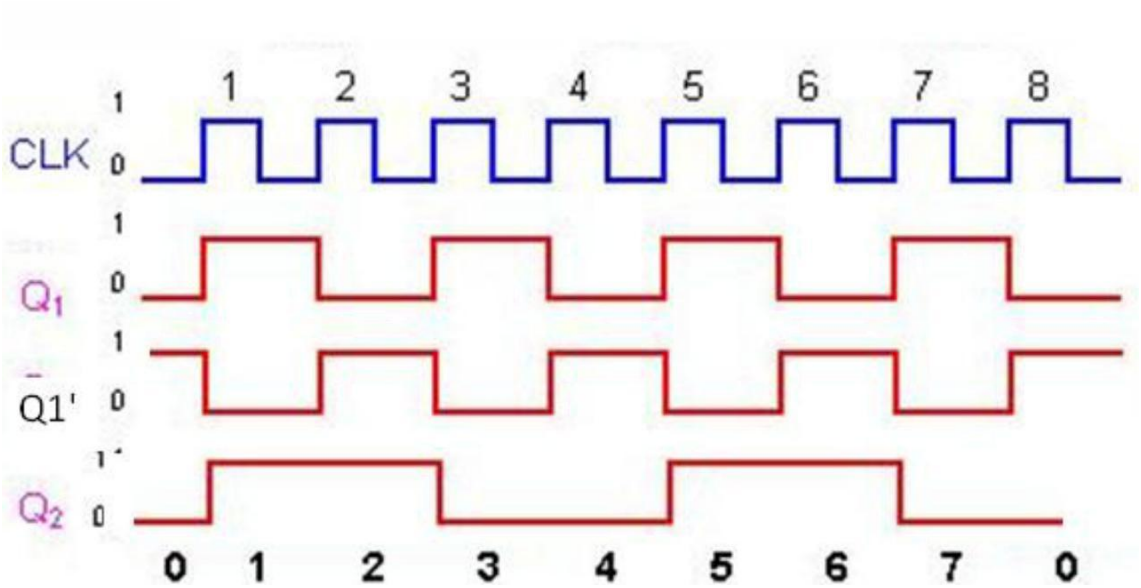
For the negative edge of the first clock pulse, output of the first flip-flop FF1 toggles its state. Thus Q1 remains at 0 and Q0 toggles to 1 and the counter state are now read as 01. During the next negative edge of the input clock pulse FF1 toggles and Q0 = 0. The output Q0 being a clock signal for the second flip-flop FF2 and the present transition acts as a negative edge for FF2 thus toggles its state Q1 = 1. The counter state is now read as 10. For the next negative edge of the input clock to FF1 output Q0 toggles to 1. But this transition from 0 to 1 being a positive edge for FF2 output Q1 remains at 1. The counter state is now read as 11. For the next negative edge of the input clock, Q0 toggles to 0. This transition from 1 to 0 acts as a negative edge clock for FF2 and its output Q1 toggles to 0. Thus the starting state 00 is attained. Figure shown below



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 figure 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 1.



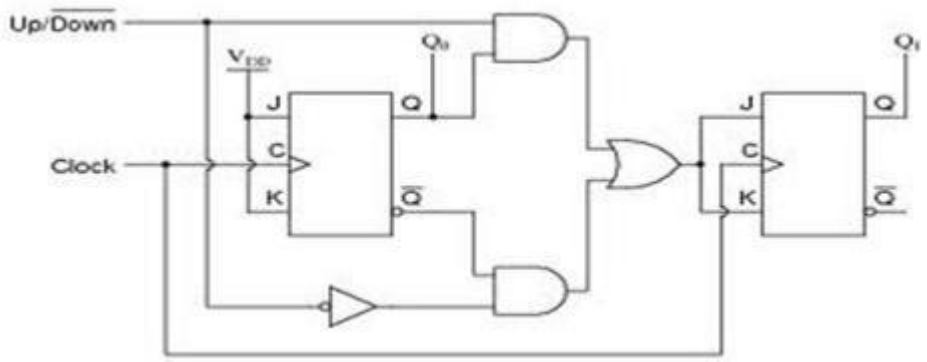


1. This positive-going signal at Q1‘ does not affect FF2 and, therefore, Q2 remains at a 1. Hence , the state of the counter after second clock pulse is 10.

   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.

2. 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:



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.

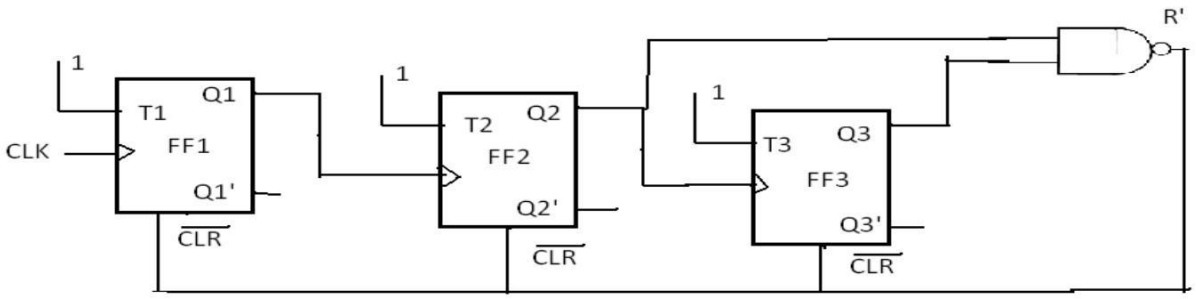
Clock signal to FF2= (Q1.Up)+(Q1‘. Down)= Q1m+Q1‘M‘

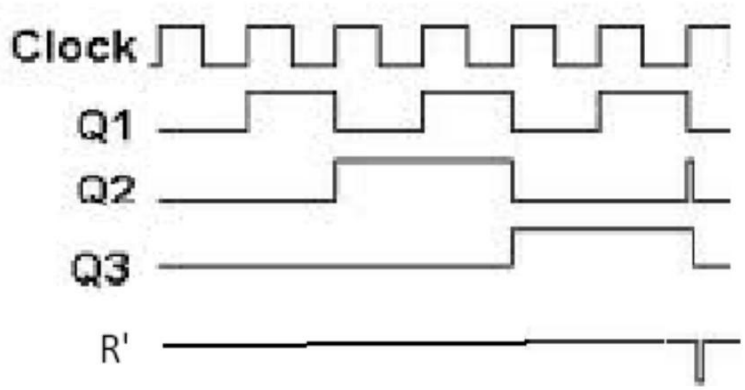
Design of Asynchronous counters:

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 condition$N \leq 2^n$ 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.

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

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.

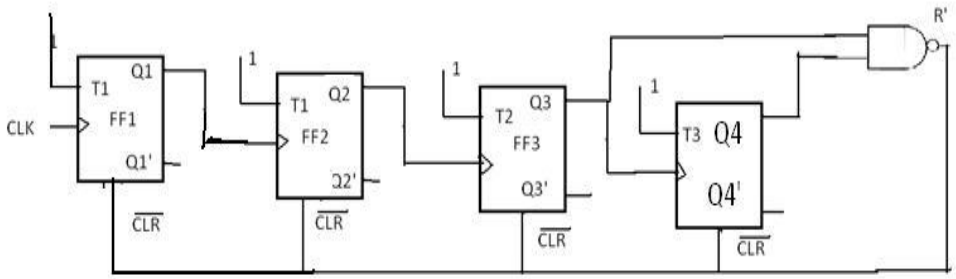
R=0 for 000 to 101, R=1 for 110, and R=X=for111

Therefore,

R=Q3Q2Q1‘+Q3Q2Q1=Q3Q2

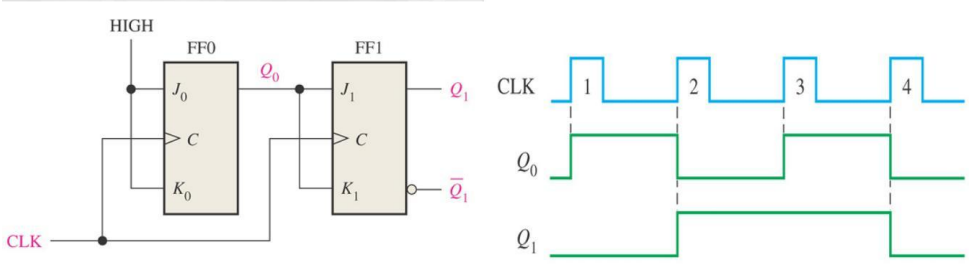
Design of a mod-10 asynchronous counter using T-flip-flops:

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.



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 can be 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 designing 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.

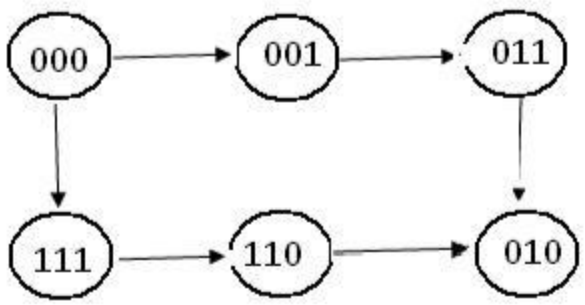
Step4: minimal expressions for excitations: obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step5: logic diagram: draw a logic diagram based on the minimal expressions.

Design of a synchronous 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.

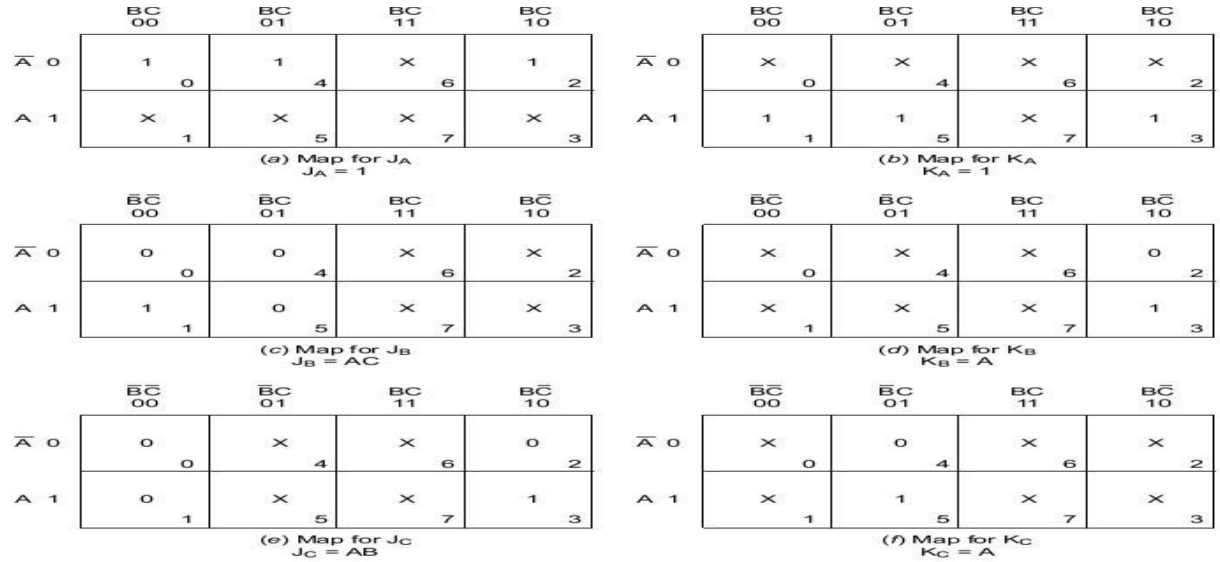


Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.

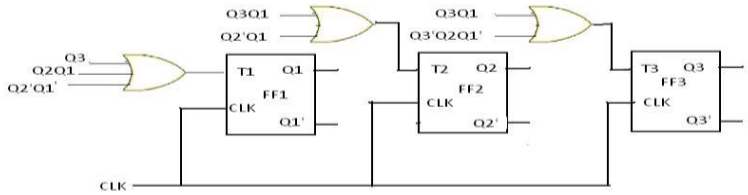
PS			**NS**			**required excitations**		
Q3	**Q2**	**Q1**	**Q3**	**Q2**	**Q1**	**T3**	**T2**	**T1**

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



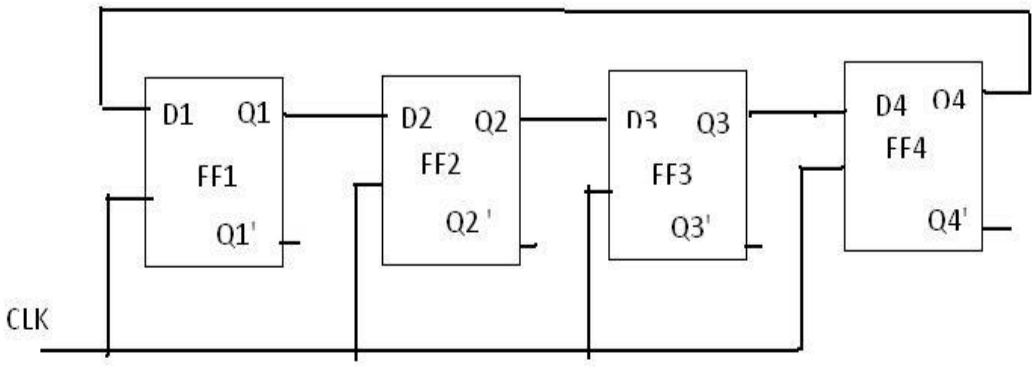
Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.



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.



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.

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 figure.

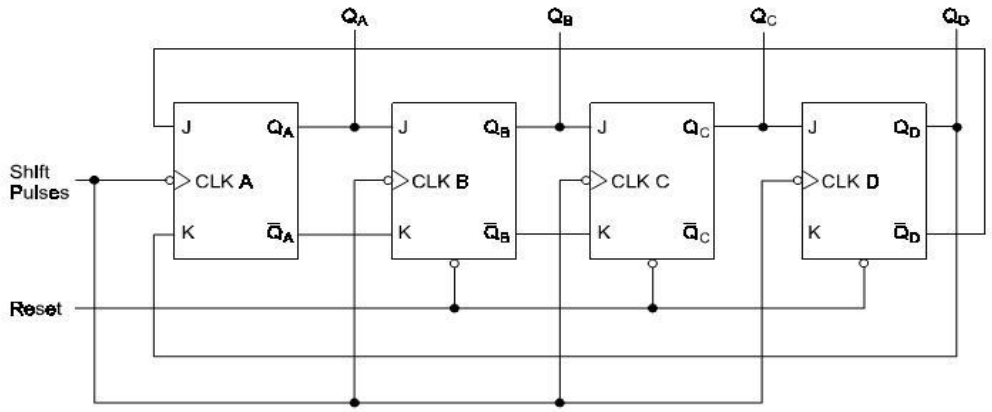
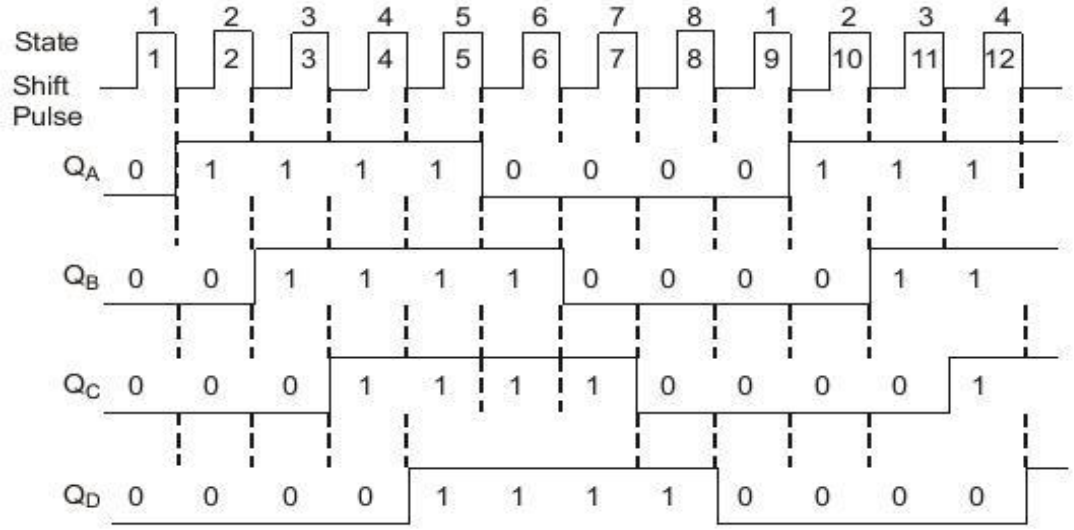


Figure: Johnson counters with JK flip-flops



ASYNCHRONOUS SEQUENTIAL CIRCUITS

Digital logic circuits can be divided into combinational logic, in which the output signals depend only on the current input signals, and sequential logic, in which the output depends both on current input and the past history of inputs. In other words, sequential logic is combinational logic with memory. Virtually all practical digital devices require sequential logic. Sequential logic can be divided into two types, synchronous logic and asynchronous logic.

SYNCHRONOUS CIRCUITS:

In synchronous logic circuits, an electronic oscillator generates a repetitive series of equally spaced pulses called the *clock signal*. The clock signal is applied to all the memory elements in the circuit, called flip-flops. The output of the flip-flops only change when triggered by the edge of the clock pulse, so changes to the logic signals throughout the circuit all begin at the same time, at regular intervals synchronized by the clock. The outputs of all the memory elements in a circuit are called the *state* of the circuit. The state of a synchronous circuit changes

only on the clock pulse. The changes in signal require a certain amount of time to propagate through the combinational logic gates of the circuit. This is called propagation delay. The period of the clock signal is made long enough so the output of all the logic gates have time to settle to stable values before the next clock pulse. As long as this condition is met, synchronous circuits will operate stably, so they are easy to design.

However a disadvantage of synchronous circuits is that they can be slow. The maximum possible clock rate is determined by the logic path with the longest propagation delay, called the *critical path*. So logic paths that complete their operations quickly are idle much of the time. Another problem is that the widely distributed clock signal takes a lot of power, and must run whether the circuit is receiving inputs or not.

ASYNCHRONOUS CIRCUITS:

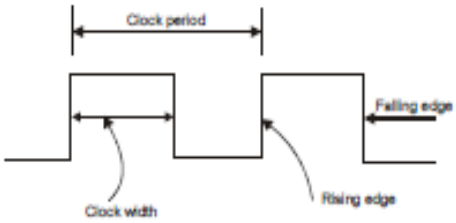
In asynchronous circuits, there is no clock, and the state of the circuit changes as soon as the input changes. Since they don't have to wait for a clock pulse to begin processing inputs, asynchronous circuits can be faster than synchronous circuits, and their speed is theoretically limited only by the propagation delays of the logic gates. However, asynchronous circuits are more difficult to design and subject to problems not found in synchronous circuits. This is because the resulting state of an asynchronous circuit can be sensitive to the relative arrival times of inputs at gates. If transitions on two inputs arrive at almost the same time, the circuit can go into the wrong state depending on slight differences in the propagation delays of the gates. This is called a race condition. In synchronous circuits this problem is less severe because race conditions can only occur due to inputs from outside the synchronous system, *asynchronous inputs*. Although some fully asynchronous digital systems have been built (see below), today asynchronous circuits are typically used in a few critical parts of otherwise synchronous systems where speed is at a premium, such as signal processing circuits.

Why Asynchronous Circuits?

- Used when speed of operation is important ,,- Response quickly without waiting for a clock pulse
- Used in small independent systems -,, Only a few components are required
- Used when the input signals may change independently of internal clock - Asynchronous in nature
- Used in the communication between two units that have their own independent clocks - Must be done in an asynchronous fashion

DIFFERENCE BETWEEN SYNCHRONOUS AND ASYNCHRONOUS

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.



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.

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 differences between synchronous and asynchronous sequential circuits are:

• In a clocked sequential circuit a change of state occurs only in response to a synchronizing clock pulse. All the flip-flops are clocked simultaneously by a common clock pulse. In an

asynchronous sequential circuit, the state of the circuit can change immediately when an input change occurs. It does not use a clock.

• In clocked sequential circuits input changes are assumed to occur between clock pulses. The circuit must be in the stable state before next clock pulse arrives. In asynchronous sequential circuits input changes should occur only when the circuit is in a stable state.

• In clocked sequential circuits, the speed of operation depends on the maximum allowed clock frequency. Asynchronous sequential circuits do not require clock pulses and they can change state with the input change. Therefore, in general the asynchronous sequential circuits are faster than the synchronous sequential circuits.

• In clocked sequential circuits, the memory elements are clocked flip-flops. In asynchronous Sequential circuits, the memory elements are either un clocked flip-flops (latches) or gate circuits with feedback producing the effect of latch operation.

In clocked sequential circuits, any number of inputs can change simultaneously (during the absence of the clock). In asynchronous sequential circuits only one input is allowed to change at a time in the case of the level inputs and only one pulse input is allowed to be present in the case of the pulse inputs. If more than one level inputs change simultaneously or more than one pulse input is present, the circuit makes erroneous state transitions due to different delay paths for each input variable.

ANALYSIS OF ASYNCHRONOUS SEQUENTIAL MACHINES

Analysis of asynchronous sequential circuits operation in fundamental mode and pulse mode will help in clearly understanding the asynchronous sequential circuits.

Fundamental Mode Circuits

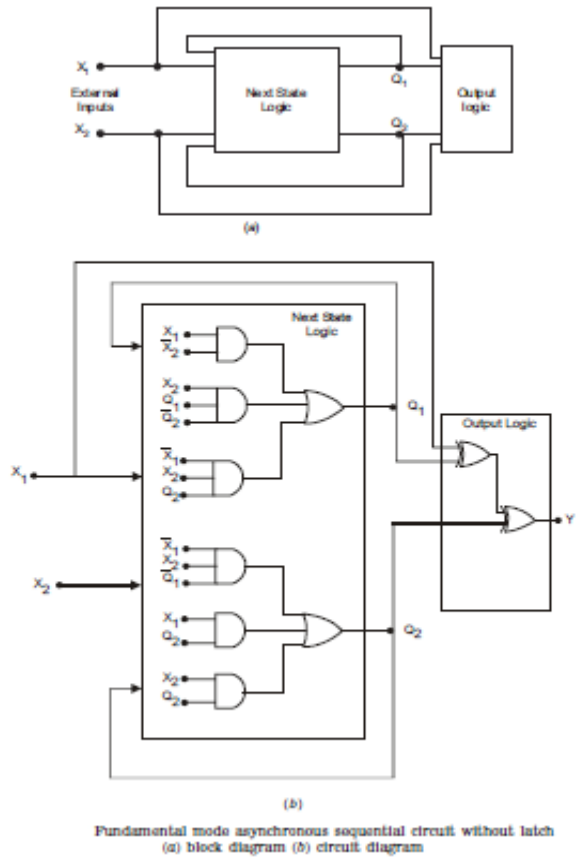
Fundamental mode circuits are of two types:

• Circuits without latches

• Circuits with latches

Circuits without Latches

Consider a fundamental mode circuit shown in Figure



Fundamental mode asynchronous sequential circuit without latch
(a) block diagram (b) circuit diagram

This circuit has only gates and no explicit memory elements are present. There are two feedback paths from Q1 and Q2 to the next-state logic circuit. This feedback creates the latching effect due to delays, necessary to produce a sequential circuit. It may be noted that a memory element latch is created due to feedback in gate circuit. The first step in the analysis is to identify the states and the state variables. The combination of level signals from external sources X1, X2 is referred to as the input state and X1, X1 are the input state variables. The combination of the outputs of memory elements are known as secondary, or internal states and these variables are known as internal or secondary state variables. Here, Q1 and Q2 are the internal variables since no explicit elements are present. The combination of both, input state and the secondary state (Q1, Q2, X1, X2) is known as the total state. Y is the output variable. The next secondary state and output logic equations are derived from the logic circuit in the next-state logic block. The next-secondary state variables are denoted by Q1.

$$Q_1^+ = X_1\overline{X}_2 + \overline{X}_1 X_2 Q_2 + X_2 Q_1 \overline{Q}_2$$
$$Q_2^+ = \overline{X}_1 X_2 \overline{Q}_1 + X_1 Q_2 + X_2 Q_2$$
$$Y = X_1 \oplus Q_1 \oplus Q_2$$

Here, Q1 and Q2 are the present secondary state variables when X1, X 2 input-states

Variables occur; the circuit goes to next secondary state. A state table shown in Table is constructed using these logic equations. If the resulting next secondary state is same as the Present state, i.e. The total state Q1, Q2, X1, X2 is said to be stable.

Otherwise it is unstable. The stability of the next total state is also shown in Table.

Table State Table

Present total state				*Next total state*				*Stable total state*	*Output*
Q_1	Q_2	X_1	X_2	Q_1^+	Q_2^+	X_1	X_2	*Yes/No*	Y
0	0	0	0	0	0	0	0	Yes	0
0	0	0	1	0	1	0	1	No	0
0	0	1	1	0	0	1	1	Yes	1
0	0	1	0	1	0	1	0	No	1
0	1	0	0	0	0	0	0	No	1
0	1	0	1	1	1	0	1	No	1
0	1	1	1	0	1	1	1	Yes	0
0	1	1	0	1	1	1	0	No	0
1	1	0	0	0	0	0	0	No	0
1	1	0	1	1	1	0	1	Yes	0
1	1	1	1	0	1	1	1	No	1
1	1	1	0	1	1	1	0	Yes	1
1	0	0	0	0	0	0	0	No	1
1	0	0	1	1	0	0	1	Yes	1
1	0	1	1	1	0	1	1	Yes	0
1	0	1	0	1	0	1	0	Yes	0

ASYNCHRONOUS SEQUENTIAL CIRCUIT DESIGN

Design of asynchronous sequential circuits is more difficult than that of synchronous sequential circuits because of the timing problems involved in these circuits. Designing an asynchronous sequential circuit requires obtaining logic diagram for the given design specifications. Usually the design problem is specified in the form of statements of the desired circuit performance precisely specifying the circuit operation for every applicable input sequence.

Design Steps

1. Primitive flow table is obtained from the design specifications. When setting up a primitive flow table it is not necessary to be concerned about adding states which may ultimately turn out to be redundant. A sufficient number of states are to be included to completely specify the circuit performance for every allowable input sequence. Outputs are specified only for stable states.

2. Reduce the primitive flow table by eliminating the redundant states, which are likely to be present. These redundant states are eliminated by merging the states. Merger diagram is used for this purpose.

3. Binary numbers are assigned to the states in the reduced flow table. The binary state assignment must be made to ensure that the circuit will be free of critical races. The output values are to be chosen for the unstable states with unspecified output entries. These must be chosen in such a way so that momentary false outputs do not occur when the circuit switches from one stable state to another stable state.

4. Transition table is obtained next.

5. From the transition table logic diagram is designed by using the combinational design methods. The logic circuit may be a combinational circuit with feedback or a circuit with S-R latches

ESSENTIAL HAZARDS

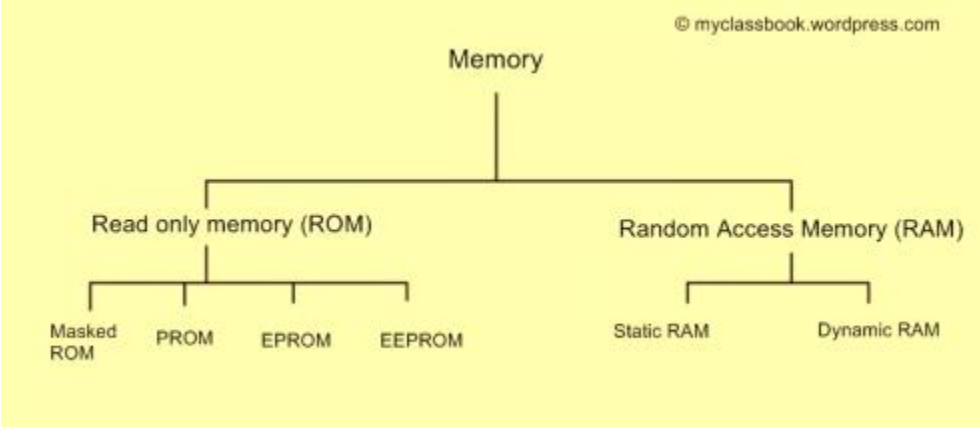
Similar to static and dynamic hazards in a combinational circuits, essential hazards occur in sequential circuits. Essential hazard is a type of hazard that exists only in asynchronous sequential circuits with two or more feedbacks. Essential hazard occurs normally in toggling type circuits. It is an error generally caused by an excessive delay to a feedback variable in response to an input change, leading to a transition to an improper state. For example, an excessive delay through an inverter circuit in comparison to the delay associated with the feedback path many cause essential hazard. Such hazards cannot be eliminated by adding redundant gates as in static hazards. To avoid essential hazard, each feedback loop must be designed with extra care to ensure that the delay in the feedback path is long enough compared to the delay of other signals that originate from the input terminals. Even though an asynchronous sequential circuit (network) is free of critical races and the combinational part of the network is fee of static and dynamic hazards, timing problems due to propagation delays may still cause the network to malfunction and go to the wrong state.

UNIT V

Programmable Devices

Classification Of Memory:

This section provides classification of memories. There are two main types of memories i.e. RAM and ROM. Following tree diagram shows the classification of Memory:



ROM (Read Only Memory):

First classification of memory is ROM. The data in this memory can only be read, no writing is allowed. It is used to store permanent programs. It is nonvolatile type of memory. The classification of ROM memory is as follows:

- Masked ROM
- PROM
- EPROM
- EEPROM

a) Masked ROM: the program or data are permanently installed at the time of manufacturing as per requirement. The data cannot be altered.

The process of permanent recording is expensive but economic for large quantities.

b) PROM (Programmable Read Only Memory): The basic function is same as that of masked ROM. but in PROM, we have fuse links. Depending upon the bit pattern, fuse can be burnt or kept intact. This job is performed by PROM programmer.

To do this, it uses high current pulse between two lines. Because of high current, the fuse will get burnt; effectively making two lines open. Once a PROM is programmed we cannot change connections, only a facility provided over masked ROM is, user can load his program in it. The disadvantage is a chance of regrowing of fuse and changes the programmed data because of aging.

c) EPROM (Erasable Programmable Read Only Memory): the EPROM is programmable by the user. It uses MOS circuitry to store data. They store 1's and 0's in form of charge. The information stored can be erased by exposing the memory to ultraviolet light which erases the data stored in all memory locations. For ultraviolet light a quartz window is provided which is covered during normal operation. Upon erasing it can be reprogrammed by using EPROM programmer. This type of memory is used in project developed and for experiment use. The

advantage is it can be programmed erased and reprogrammed. The disadvantage is all the data get erased even if you want to change single data bit.

d) EEPROM: EEPROM stands for electrically erasable programmable read only memory. This is similar to EPROM except that the erasing is done by electrical signals instead of ultraviolet light. The main advantage is the memory location can be selectively erased and reprogrammed. But the manufacturing process is complex and expensive so do not commonly used.

RAM (Random Access Memory):

Second classification of memory is RAM. The RAM is also called as read/write memory. The RAM is a volatile type of memory. It allows programmer to read or write data. If the user wants to check execution of any program, user feeds the program in RAM memory and executes it. The result of execution is then checked by either reading memory location contents or by register contents.

Following is the classification of RAM memory. It is available in two types:

a) SRAM (Static RAM): SRAM consists of flip-flop; using either transistor or MOS. for each bit we require one flip-flop. Bit status will remain as it is; unless and until you perform next write operation or power supply is switched off.

- **Advantages of SRAM:**

1) Fast memory (less access time)
2) Refreshing circuit is not required.

- **Disadvantages of SRAM:**

1) Low package density
2) Costly

b) DRAM (Dynamic RAM): In this type of memory a data is stored in form of charge in capacitors. When data is 1, the capacitor will be charged and if data is 0, the capacitor will not be charged. Because of capacitor leakage currents the data will not be hold by these cells. So the DRAMs require refreshing of memory cells. It is a process in which same data is read and written after a fixed interval.

- **Advantages of DRAM:**

1) High package density
2) Low cost

- **Disadvantages of DRAM:**

1) Required refreshing circuit to maintain or refresh charge on capacitor, every after few milliseconds.

PLD:

A PLD, or programmable logic device, is an electronic component that is used in order to build digital circuits that are reprogrammable. A programmable logic device does not have a defined function once manufactured, unlike a logic gate and has to be programmed before it can be used.

Types of Programmable Logic Devices

There are several different kinds of programmable logic devices at Future Electronics. We stock many of the most common types categorized by several parameters including Programmable Type, Number of I/O Lines, Supply Voltage, Memory Density, System Gates, Packaging Type and many other parameters specific to the type of programmable logic device. Our parametric filters will allow you to refine your search results according to the required specifications.

Applications for Programmable Logic Devices:

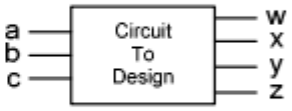
There are many applications that use programmable logic devices including:

- Networking
- Aerospace and defence
- Automotive
- Consumer electronics
- Computing
- Distributed monetary systems
- Communications
- Audio
- Computing
- Avionics

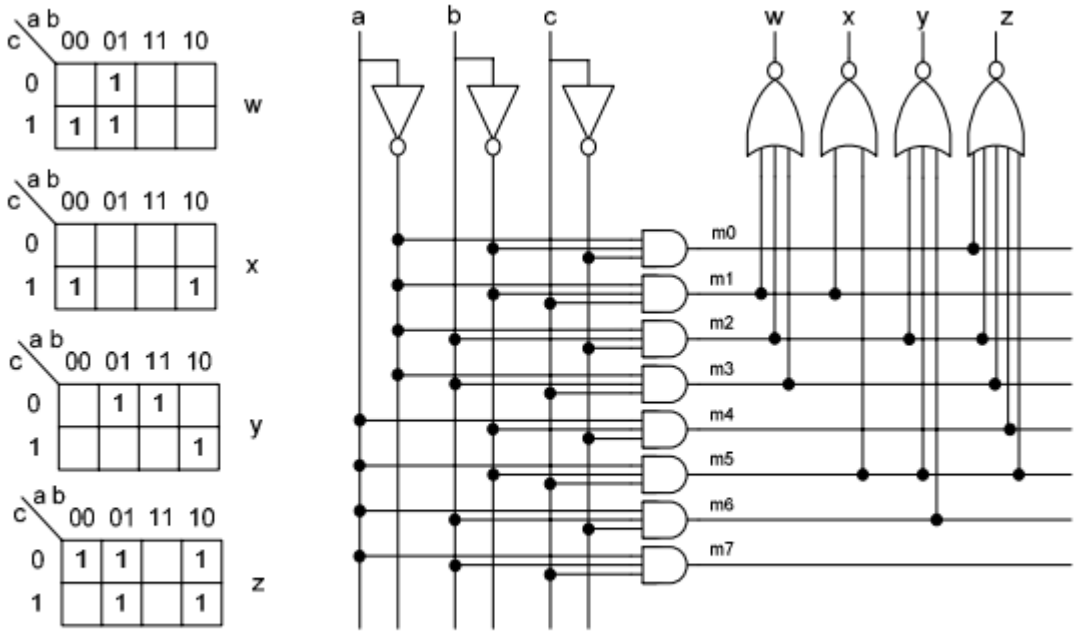
Read Only Memories:

The simplest way to implement the circuit of Figure is to form its minterms using AND gates and then OR the appropriate minterms for formation of the four circuit outputs. The circuit requires eight 3-input AND gates and four OR gates that can take up-to eight inputs. It is easiest to draw this structure in an array format as shown in Figure.

m:	a	b	c	w	x	y	z
0:	0	0	0	0	0	0	1
1:	0	0	1	1	1	0	0
2:	0	1	0	1	0	1	1
3:	0	1	1	1	0	0	1
4:	1	0	0	0	0	0	1
5:	1	0	1	0	1	1	1
6:	1	1	0	0	0	1	0
7:	1	1	1	0	0	0	0



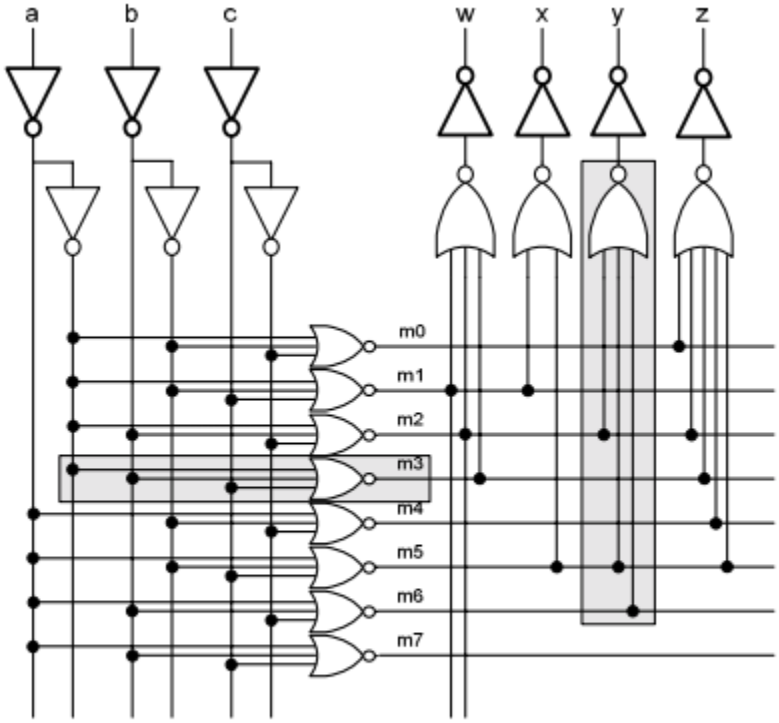
The circuit shown has an array of AND gates and an array of OR gates, that are referred to as the AND-plane and the OR-plane. In the AND-plane all eight minterms for the three inputs, a, b, and c are generated. The OR plane uses only the minterms that are needed for the outputs of the circuit. See for example minterm 7 that is generated in the AND-plane but not used in the ORplane. Figure shows the block diagram of this array structure.



AND-OR Implementation

NOR Implementation

Since realization of AND and OR gates in most technologies are difficult and generally use more delays and chip area than NAND or NOR implementations, we implement our example circuit using NOR gates. Note that a NOR gate with complemented outputs is equivalent to an OR, and a NOR gate with complemented inputs is equivalent to an AND gate. Our all NOR implementation of Figure uses NOR gates for generation of minterms and circuit outputs. To keep functionality and activity levels of inputs and outputs intact, extra inverters are used on the circuit inputs and outputs. These inverters are highlighted in Figure. Although NOR gates are used, the left plane is still called the AND-plane and the right plane is called the OR-plane.



All NOR Implementation

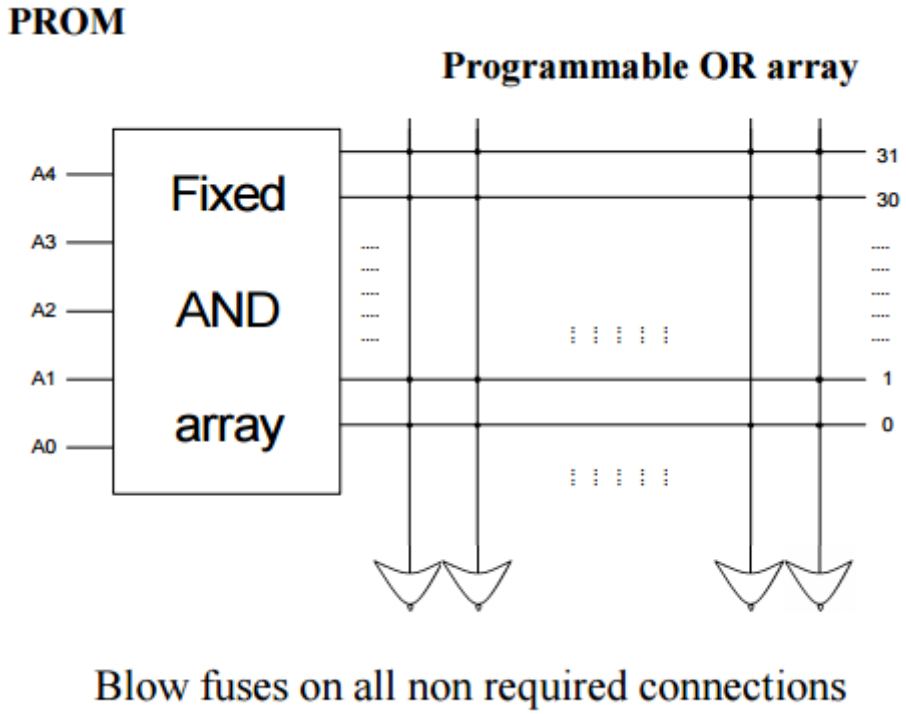
ROM Variations:

The acronym, ROM is generic and applies to most read only memories. What is today implied by ROM may be ROM, PROM, EPROM, EEPROM or even flash memories

ROM: ROM is a mask-programmable integrated circuit, and is programmed by a mask in IC manufacturing process. The use of mask-programmable ROMs is only justified when a large volume is needed. The long wait time for manufacturing such circuits makes it a less attractive choice when time-to market is an issue.

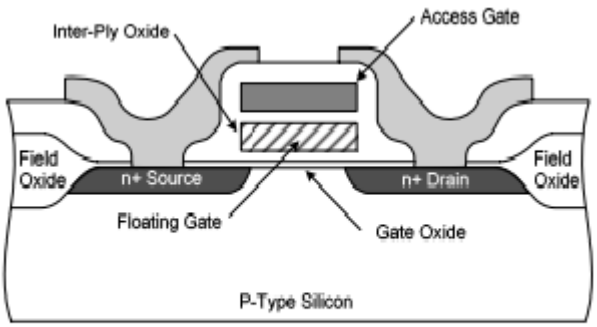
PROM: Programmable ROM is a one-time programmable chip that, once programmed, cannot be erased or altered. In a PROM, all minterms in the AND-plane are generated, and connections

of all AND-plane outputs to ORplane gate inputs are in place. By applying a high voltage, transistors in the OR-plane that correspond to the minterms that are not needed for a certain output are burned out. A fresh PROM has all transistors in its OR-plane connected. When programmed, some will be fused out permanently.



Blow fuses on all non required connections

EPROM: An Erasable PROM is a PROM that once programmed, can be completely erased and reprogrammed. Transistors in the OR-plane of an EPROM have a normal gate and a floating gate as shown in Figure below. The non-floating gate is a normal NMOS transistor gate, and the floating- gate is surrounded by insulating material that allows an accumulated charge to remain on the gate for a long time.



Floating Gate

When not programmed, or programmed as a '1', the floating gate has no extra charge on it and the transistor is controlled by the non-floating gate (access gate). To fuse-out a transistor, or program a '0' into a memory location, a high voltage is applied to the access gate of the transistor which causes accumulation of negative charge in the floating-gate area. This negative charge prevents logic 1 values on the access gate from turning on the transistor. The transistor, therefore, will act as an unconnected transistor for as long as the negative charge remains on its floating-gate. To erase an EPROM it must be exposed to ultra-violate light for several minutes. In this case, the insulating materials in the floating-gates become conductive and these gates start losing their negative charge. In this case, all transistors return to their normal mode of operation. This means that all EPROM memory contents become 1, and ready to be reprogrammed. Writing data into an EPROM is generally about a 1000 times slower than reading from it. This is while not considering the time needed for erasing the entire EPROM.

AND	OR	DEVICE
Fixed	Fixed	Not Programmable
Fixed	Programmable	PROM
Programmable	Fixed	PAL
Programmable	Programmable	PLA

EEPROM: An EEPROM is an EPROM that can electrically be erased, and hence the name: Electrically Erasable Programmable ROM. Instead of using ultraviolate to remove the charge on the non-floating gate of an EPROM transistor, a voltage is applied to the opposite end of the transistor gate to remove its accumulated negative charge. An EEPROM can be erased and reprogrammed without having to remove it. This is useful for reconfiguring a design, or saving system configurations. As in EPROMs, EEPROMs are non-volatile memories. This means that they save their internal data while not powered. In order for memories to be electrically erasable, the insulating material surrounding the floating-gate must be much thinner than those of the EPROMS. This makes the number of times EEPROMs can be reprogrammed much less than that of EPROMs and in the order of 10 to 20,000. Writing into a byte of an EEPROM is about 500 times slower than reading from it.

Flash Memory: Flash memories are large EEPROMs that are partitioned into smaller fixed-size blocks that can independently be erased. Internal to a system, flash memories are used for saving system configurations. They are used in digital cameras for storing pictures. As external devices, they are used for temporary storage of data that can be rapidly retrieved. Various forms of ROM are available in various sizes and packages. The popular 27xxx series EPROMs come in packages that are organized as byte addressable memories. For example, the 27256 EPROM has 256K bits of memory that are arranged into 32K bytes.

Programmable Logic Arrays

Compared to a ROM and a PAL, a PLA is the most flexible having a programmable set of ANDs combined with a programmable set of ORs.

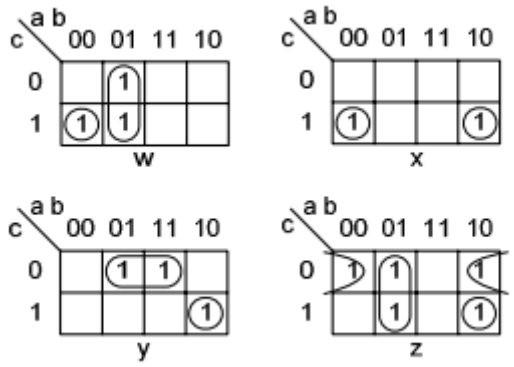
Advantages

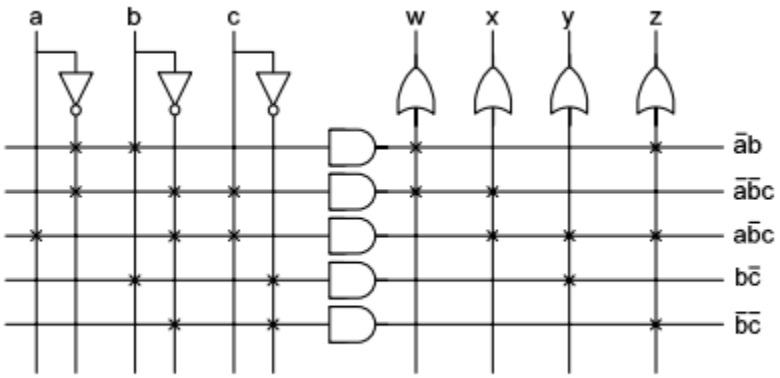
- A PLA can have large N and M permitting implementation of equations that are impractical for a ROM (because of the number of inputs, N, required
- A PLA has all of its product terms connectable to all outputs, overcoming the problem of the limited inputs to the PAL Ors
- Some PLAs have outputs that can be complemented, adding POS functions

Disadvantages

- Often, the product term count limits the application of a PLA.
- Two-level multiple-output optimization is required to reduce the number of product terms in an implementation, helping to fit it into a PLA.
- Multi-level circuit capability available in PAL not available in PLA. PLA requires external connections to do multi-level circuits.

For illustrating the PLA structure, we use the 3-input, 4-output example circuit of Figure(fig: All NOR implementation). The AND-OR implementation of this circuit that is shown in Figure led to the ROM structure.



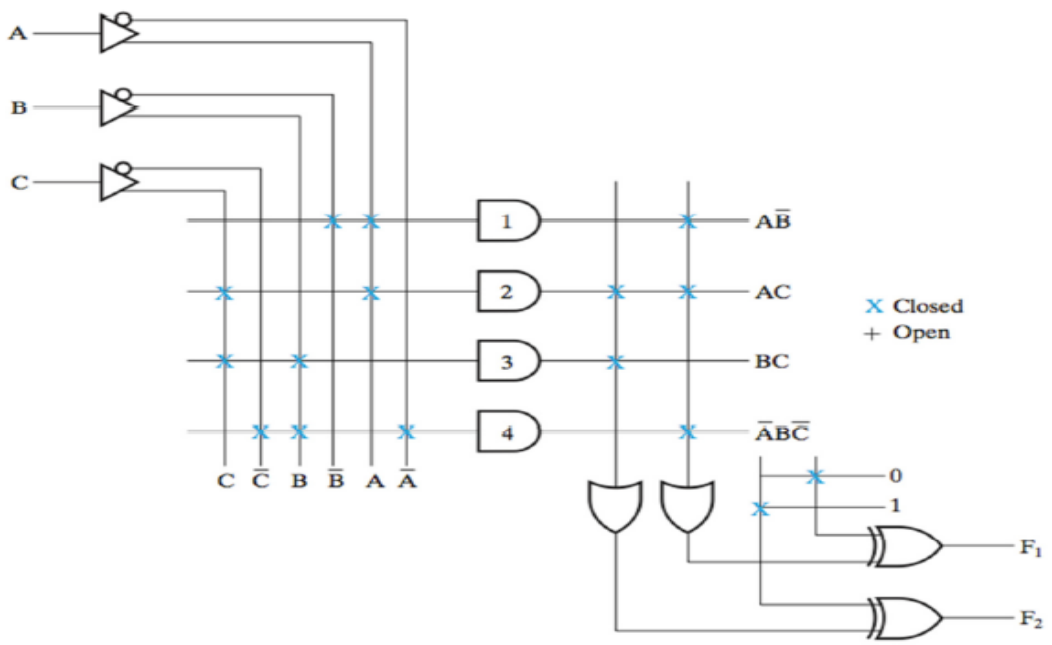


PLA Implementation

Programmable Logic Array Example

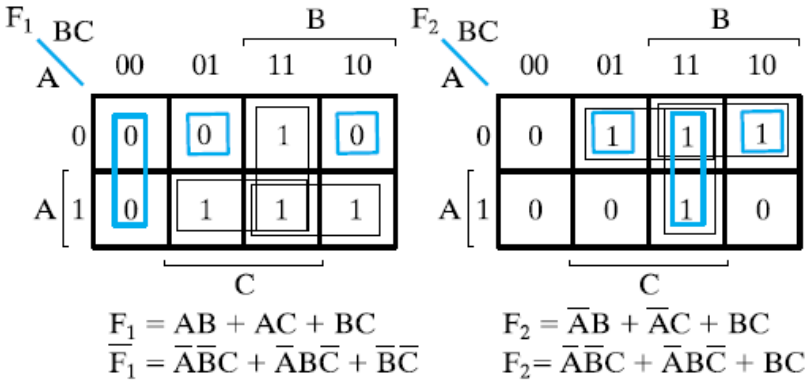
F1=AB’ + AC + A’BC’

F2= (AC+BC)’



Implementing a Combinational Circuit Using a PLA

F1(A,B,C)= Σm(3,5,6,7)
F2(A,B,C)= Σm(1,2,3,7)



The solution is:

$$F_1 = \overline{\bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{B}\bar{C}}$$

$$F_2 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + BC$$

PAL:

A PAL (programmable array logic) is a programmable logic device in which each output is computed as a two-level "sum of products" (an OR of ANDs). Modern PALs use a programmable "macro cell" on each output. These macro cells contains a D flip-flop and programmable switches (multiplexers) that configure the macro cell as either a registered or combinational output and as either a positive-true or negative-true logic. Each of the first-level AND (product) terms can be programmed to include any combination of the inputs, the outputs,

or their complements. Each of the second-level OR (sum) terms is connected to a fixed number of product terms (AND gate outputs) (typically 8 to 12). The advantages of the PAL architecture include low and fixed (two gate) propagation delays (typically down to 5 ns), and simple, low-cost (free), design tools. However, the PAL architecture limits the design to simple state machines and simple combinational circuits.

Modern PALs typically have EEPROM configuration memories and are programmed with device programmers similar to EPROM programmers

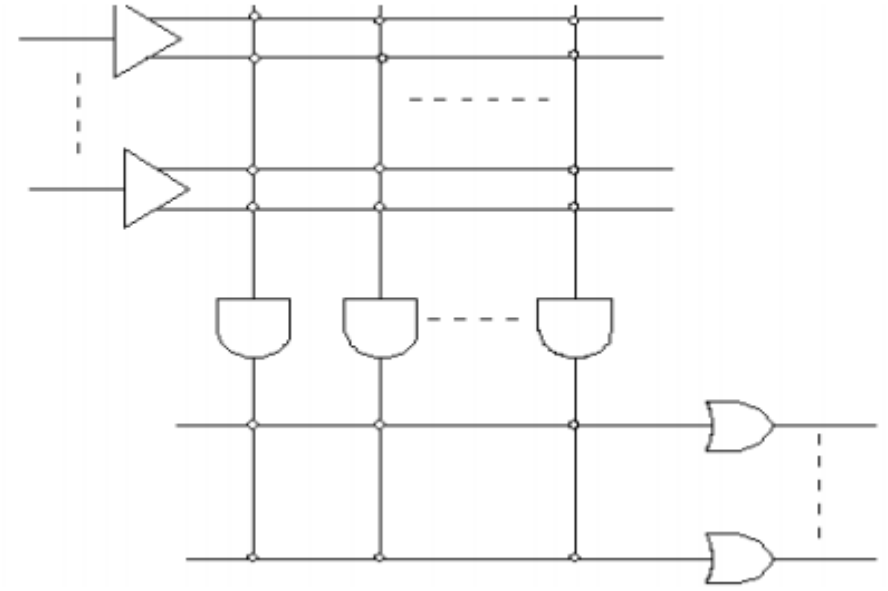
Advantages

- For given internal complexity, a PAL can have larger N and M
- Some PALs have outputs that can be complemented, adding POS functions
- No multilevel circuit implementations in ROM (without external connections from output to input). PAL has outputs from OR terms as internal inputs to all AND terms, making implementation of multi-level circuits easier.

Disadvantage

- ROM guaranteed to implement any M functions of N inputs. PAL may have too few inputs to the OR gates.

PAL has programmable AND-array, but fixed OR-array.



Programmable Array Logic Example

- 4-input, 3-output PAL with fixed, 3-input OR terms
- What are the equations for F1 through F4?

$W(A,B,C,D) = \Sigma m\,(2,12,13)$
$X(A,B,C,D) = \Sigma m\,(7,8,9,10,11,12,13,14,15)$
$Y(A,B,C,D) = \Sigma m\,(0,2,3,4,5,6,7,8,10,11,15)$
$Z(A,B,C,D) = \Sigma m\,(1,2,8,12,13)$

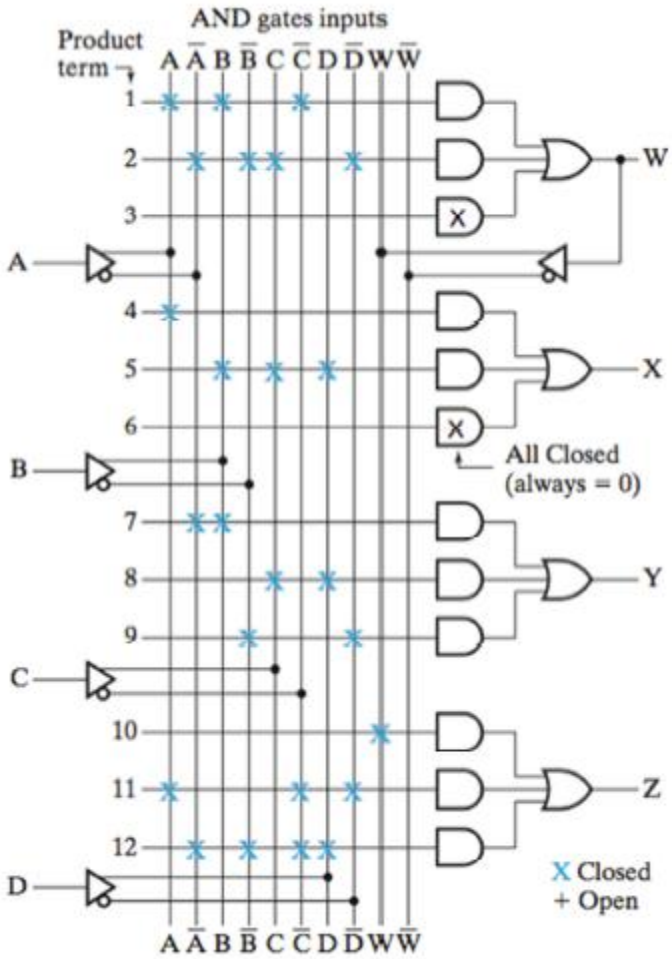
Simplifying the four function to a minimum number of terms results in the following Boolean functions

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

$X = A+BCD$

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

$Z = ABC'+A'B'CD'+AC'D'+A'B'C'D = W+AC'D'+A'B'C'D$

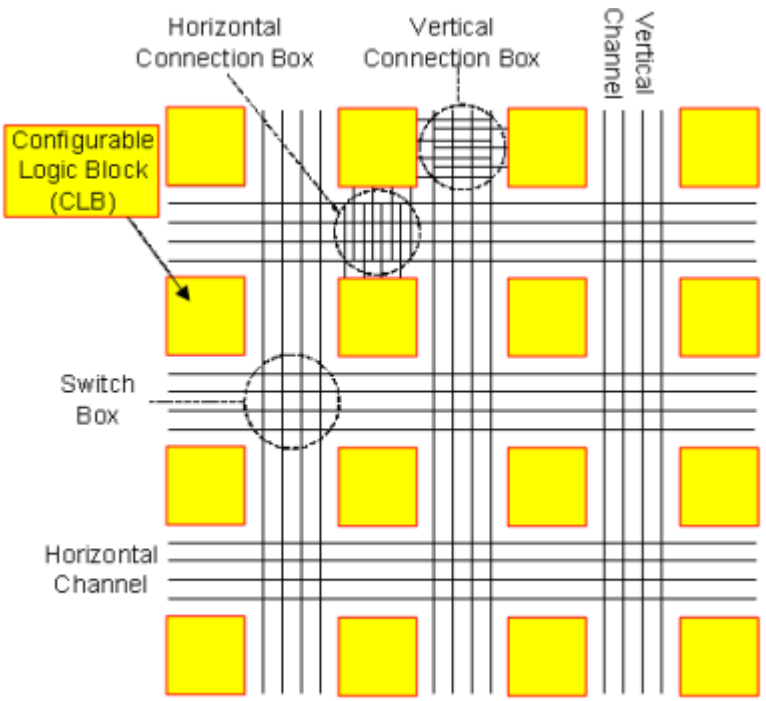


FPGA:

Field Programmable Gate Arrays (FPGAs): The FPGA consists of 3 main structures:

1. Programmable logic structure,
2. Programmable routing structure, and
3. Programmable Input/Output (I/O).

1. **Programmable logic structure:** The programmable logic structure FPGA consists of a 2-dimensional array of configurable logic blocks (CLBs).



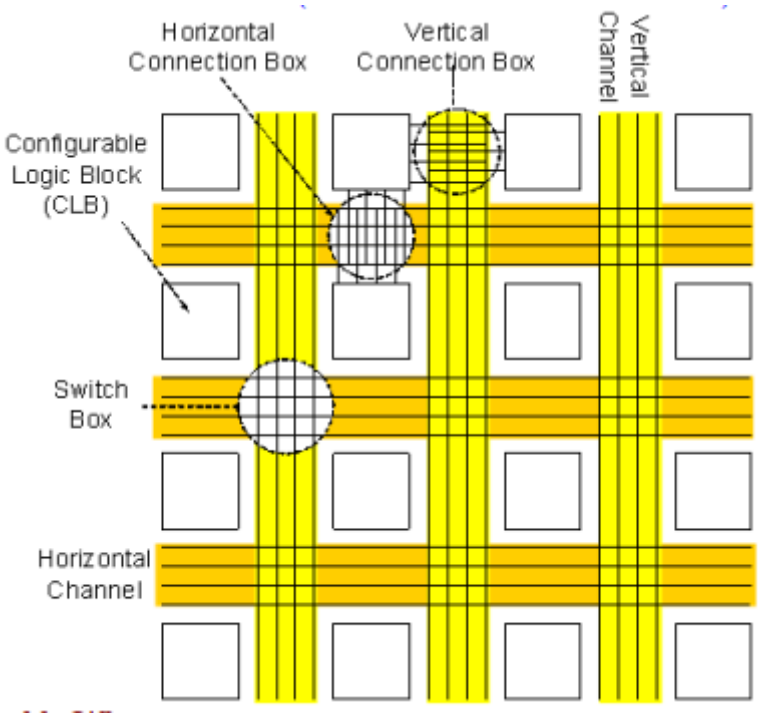
Each CLB can be configured (programmed) to implement any Boolean function of its input variables. Typically CLBs have between 4-6 input variables. Functions of larger number of variables are implemented using more than one CLB. In addition, each CLB typically contains 1 or 2 FFs to allow implementation of sequential logic. Large designs are partitioned and mapped to a number of CLBs with each CLB configured (programmed) to perform a particular function. These CLBs are then connected together to fully implement the target design. Connecting the CLBs is done using the FPGA programmable routing structure.

2. Programmable routing structure To allow for flexible interconnection of CLBs, FPGAs have 3 programmable routing resources:

--Vertical and horizontal routing channels which consist of different length wires that can be connected together if needed. This channel runs vertically and horizontally between columns and rows of CLBs as shown in the Figure.

--Connection boxes, which are a set of programmable links that can connect input and output pins of the CLBs to wires of the vertical or the horizontal routing channels.

-- Switch boxes, located at the intersection of the vertical and horizontal channels. These are a set of programmable links that can connect wire segments in the horizontal and vertical channels.



Programmable I/O :

These are mainly buffers that can be configured either as input buffers, output buffers or input/output buffers. They allow the pins of the FPGA chip to function either as input pins, output pins or input/output pins.