Digital Logic Syllabus

Number Systems:- Binary, Octal and Hexa decimal representation and their conversions; BCD,ASCII, EBCDIC, Gray codes and their conversions;
Binary number representation:- Signed binary number representation with 1’s and 2’s complement methods, Binary arithmetic.
Boolean Expression:- Various Logic gates-their truth tables and circuits; Representation in SOP and POS forms; Minimization of logic expressions by algebraic method, Kmap method,
Combinational Circuits: Adder and Subtractor circuits; Applications and circuits of Encoder ,Decoder, Comparator ,Multiplexer, De-Multiplexer and Parity Generator.
Sequential Circuits: Basic memory element-S-R, J-K,D and T Flip Flops, various types of Registers and counters and their design.
Memory circuits: RAM ,ROM, EPROM, EEPROM, Design of combinational circuits-using ROM, PLAs and PLDs. Logic familiesTTL,ECL,MOS and CMOS, their operation and specifications.

DIGITAL SYSTEM

- A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.
- Digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

Characteristics of Digital systems

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

Advantages of Digital system

1. **Ease of programmability:-** The digital systems can be used for different applications by simply changing the program without additional changes in hardware.
2. **Reduction in cost of hardware:-** The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.
3. **High speed:-** Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.
4. **High Reliability:-** Digital systems are highly reliable one of the reasons for that is use of error correction codes.
5. **Design is easy:-** The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.
6. **Result can be reproduced easily:-** Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

Disadvantages of Digital Systems

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

- Quantization error during analog signal sampling.
- Number system is a basis for counting varies items. Modern computers communicate and operate with binary numbers which use only the digits 0 &1.

Character Encoding

- Character encoding is the process of assigning numbers to graphical characters
- Character encoding tells computers how to interpret digital data into letters, numbers and symbols.
- **ASCII (American Standard Code for Information Interchange (1963):** ASCII is a 7-bit character encoding standard, representing 128 characters, including control characters, English letters, digits, and punctuation.
- **Extended ASCII:** Later, 8-bit versions of ASCII were developed, allowing for 256 characters, including additional symbols and characters for other languages.
- **EBCDIC (Extended Binary Coded Decimal Interchange Code):**
- IBM developed EBCDIC (Extended Binary-Coded Decimal Interchange Code), which is an 8-bit character encoding based on BCD.
- An 8-bit character encoding used primarily on IBM mainframe (IBM System/360) and midrange systems.
- **Unicode:** The most widely used encoding system today, capable of representing characters from nearly every writing system in the world, including emojis.
  - **UTF-8:** A variable-length encoding that uses 1 to 4 bytes per character. It is backward-compatible with ASCII and is the most common encoding on the web.
  - **UTF-16:** Uses 2 or 4 bytes per character and is commonly used in Windows and Java.
  - **UTF-32:** Uses 4 bytes per character, providing direct access to all Unicode code points, but is less common due to its inefficiency in terms of space.

Number System $(\quad)_x$

1. **Decimal Number: $(\quad)_{10}$**

   The decimal system has ten symbols: 0,1,2,3,4,5,6,7,8,9.
   It has a radix/ base of 10.
   In binary system weight is expressed as power of 10.

2. **Binary number: $(\quad)_2$**

   It has only 2 symbols: 0,1.
   The binary number has a radix of 2.
   In binary system weight is expressed as power of 2.

3. **Octal Number: $(\quad)_8$**

   Octal systems use a base or radix of 8. It uses first eight digits of decimal number system. Thus it has digits from 0,1,2,3,4,5,6,7.

4. **Hexadecimal Number: $(\quad)_{16}$**

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

Number Base conversions

1. **Decimal to Binary:** To convert decimal number in any other number system (base) divide the given number with the desired number system (base) and write down the reminder from bottom to up.

   Here we have to convert in binary so divide by base 2.

   Example: $(36)_{10} = (100100)_2$

		Reminder
2	36	0 ↑
2	18	0
2	9	1
2	4	0
2	2	0
	1 →	1

2. **Decimal to octal:** To convert decimal number in any other number system(base) divide the given number with the desired number system(base) and write down the reminder from bottom to up.

   Here we have to convert in octal so divide by base 8.

   Example: $(136)_{10} = (210)_8$

		Reminder
8	136	0 ↑
8	17	1
	2 →	2

3. **Decimal to Hexadecimal:** To convert decimal number in any other number system(base) divide the given number with the desired number system(base) and write down the reminder from bottom to up. **Here we have to convert in Hexadecimal so divide by base 16.**

   Example: $(156)_{10} = (9C)_{16}$

		Reminder
16	156	12 -> C ↑
	9 →	9

4. **Binary to Decimal:** To convert any number system (base) in Decimal number system multiply the given number with given base weight(place value) then add all the digits to get the decimal number.

   Here we are converting binary number to decimal so multiply each bit with the weight of each bits weight (place value)

   Example: $(110101)_2 = (53)_{10}$

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 53$$

5. **Octal to Decimal:** To convert any number system (base) in Decimal number system multiply the given number with given base weight(place value) then add all the digits to get the decimal number.

   Here we are converting Octal number to decimal so multiply each bit with the weight of each bits weight (place value).

   Example: $(145)_8 = (101)_{10}$

   $1 \times 8^2 + 4 \times 8^1 + 5 \times 8^0$
   $= 1 \times 64 + 4 \times 8 + 5 \times 1 = 101$

6. **Hexadecimal to Decimal:** To convert any number system (base) in Decimal number system multiply the given number with given base weight(place value) then add all the digits to get the decimal number.

   Here we are converting Hexadecimal number to decimal so multiply each bit with the weight of each bits weight (place value).

   Example: $(136)_{16} = (310)_{10}$

   $1 \times 16^2 + 3 \times 16^1 + 6 \times 16^0$
   $= 1 \times 256 + 3 \times 16 + 6 \times 1 = 310$

7. **Binary to octal:** (make pair of 3 bits from left to right and then convert each pair in decimal)

   Example: $(1100110101)_2 = (1465)_8$

$$\begin{array}{cccc} 0\ 0\ 1 & 1\ 0\ 0 & 1\ 1\ 0 & 1\ 0\ 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 4 & 6 & 5 \end{array}$$

8. **Binary to Hexadecimal:** (make pair of 4 bits from left to right and then convert each pair in decimal)

   Example: $(1100101110101)_2 = (1465)_{16}$

$$\begin{array}{cccc} 1\ 1\ 0\ 1 & 1\ 0\ 0\ 1 & 0\ 1\ 1\ 1 & 0\ 1\ 0\ 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 13 & 9 & 7 & 5 \\ \downarrow & & & \\ D & & & \end{array}$$

9. **Octal to Binary: (convert every individual decimal digit in 3bit binary number and then write together)**

   Example: $(453)_8 = (100101011)_2$

$$\begin{array}{ccc} 4 & 5 & 3 \\ \downarrow & \downarrow & \downarrow \\ 100 & 101 & 011 \end{array}$$

10. **Hexadecimal to Binary:** (convert every individual decimal digit in 4 bit binary number and then write together)

Example: $(638)_{16}$ =$(011000111000)_2$

$$\begin{array}{ccc}6 & 3 & 8\\ \downarrow & \uparrow & \downarrow\\ 0110 & 0011 & 1000\end{array}$$

11. **Octal to Hexadecimal:** (Convert the given octal in decimal or binary and then convert that Decimal or binary number to hexadecimal)

Here I have converted it in binary number for ease.

Example: $(2341)_8$ =$(010011100001)_2$ = $(4E1)_{16}$

12. **Hexadecimal to Octal:** (Convert the given hexadecimal in Decimal or Binary and then convert that Decimal or binary number to octal)

Here I have converted it in binary number for ease.

Example: $(3B69)_{16}$ = $(0011101101101001)_2$ =$(35551)_8$

Number Base conversions of fraction number

1. **Fraction Decimal to Binary:**

Example:

$(36.65)_{10}$ = $(100100.\ 1010)_2$

Intregal Part **To solve fraction part we multiply with given base**

0.65 x 2= 1.30

		Reminder
2	36	0
2	18	0
2	9	1
2	4	0
2	2	0
	1	1

0.30 x 2 = 0.60
0.60 x 2 = 1.20
0.20 x 2 = 0.40

2. **Fraction Decimal to octal:** To solve fraction part we multiply with given base. Either encounter 0 or get repeated pattern or options given in the question matched.

Example: $(136.123)_{10}$ = $(210.0767\ )_8$

		Reminder
8	136	0
8	17	1
	2	2

0.123 x 8 = 0.984
0.984 x 8 = 7.872
0.872 x 8 = 6.976
0.976 x 8 =7.808

3. Fraction Decimal to Hexadecimal:

Example: $(156.25)_{10} = (9C.4)_{16}$

		Reminder
16	**156**	**12 -> C** ↑
	9 →	**9**

0.25 x 16 =4.00

4. Fraction Binary to Decimal:

Example: $(110101.101)_2 = (53.625)_{10}$

$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\ .\ 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1\ .\ 1 \times 1/2 + 1 \times 1/8 = 53.625$

5. Fraction Octal to Decimal:

Example: $(145.24)_8 = (101.3125)_{10}$

$1 \times 8^2 + 4 \times 8^1 + 5 \times 8^0\ .\ 2 \times 8^{-1} + 4 \times 8^{-2}$

$= 1 \times 64 + 4 \times 8 + 5 \times 1\ .\ 2 \times 1/8 + 4 \times 1/64 = 101.3125$

6. Fraction Hexadecimal to Decimal:

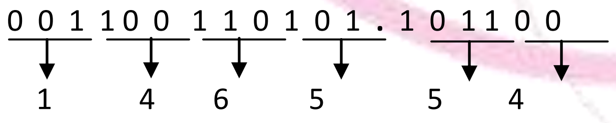
Example: $(136.29)_{16} = (310.1601)_{10}$

$1 \times 16^2 + 3 \times 16^1 + 6 \times 16^0\ .\ 2 \times 16^{-1} + 9 \times 16^{-2}$

$= 1 \times 256 + 3 \times 16 + 6 \times 1\ .\ 2 \times 1/16 + 9 \times 1/256 = 310.1601$

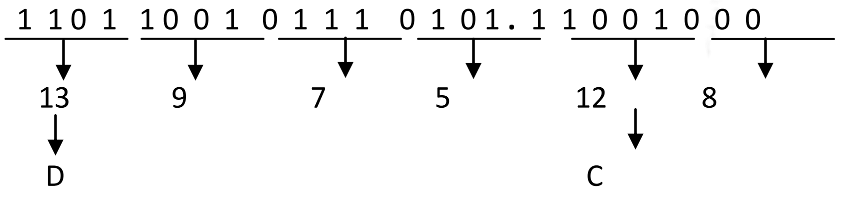
7. Fraction Binary to octal:

Example: $(1100110101.1011)_2 = (1465.54)_8$



8. Fraction Binary to Hexadecimal:

Example: $(1100101110101.11001)_2 = (1465.C\ 8)_{16}$



9. **Fraction Octal to Binary:**

Example: $(453.21)_8 = (100101011.010001)_2$

```
 4     5     3  .  2     1
 ↓     ↓     ↓     ↓     ↓
100   101   011 . 010   001
```

10. **Fraction Hexadecimal to Binary:** (convert every individual decimal digit in 4 bit binary number and then write together)

Example: $(638.92)_{16} = (011000111000.10010010)_2$

```
 6      3      8  .   9      2
 ↓      ↓      ↓      ↓      ↓
0110   0011   1000 . 1001   0010
```

11. **Fraction Octal to Hexadecimal:**

Example: $(2341.11)_8 = (010011100001.001001)_2 = (4E1.24)_{16}$

12. **Fraction Hexadecimal to Octal:**

Example: $(3B69.4C)_{16} = (0011101101101001.010011)_2 = (35551.23)_8$

Addition and subtraction of binary number

Carry 1 11
1101011
+ 1001011
10110110

Carry 111
100111
+ 100101
1001100

Borrow
1101011
- 1001011
0100000

Borrow 1 1 2
1100011
- 1001111
0010100

Addition of any number system

Carry 111
$(4576)_8$
+ $(3245)_8$
$(10043)_8$

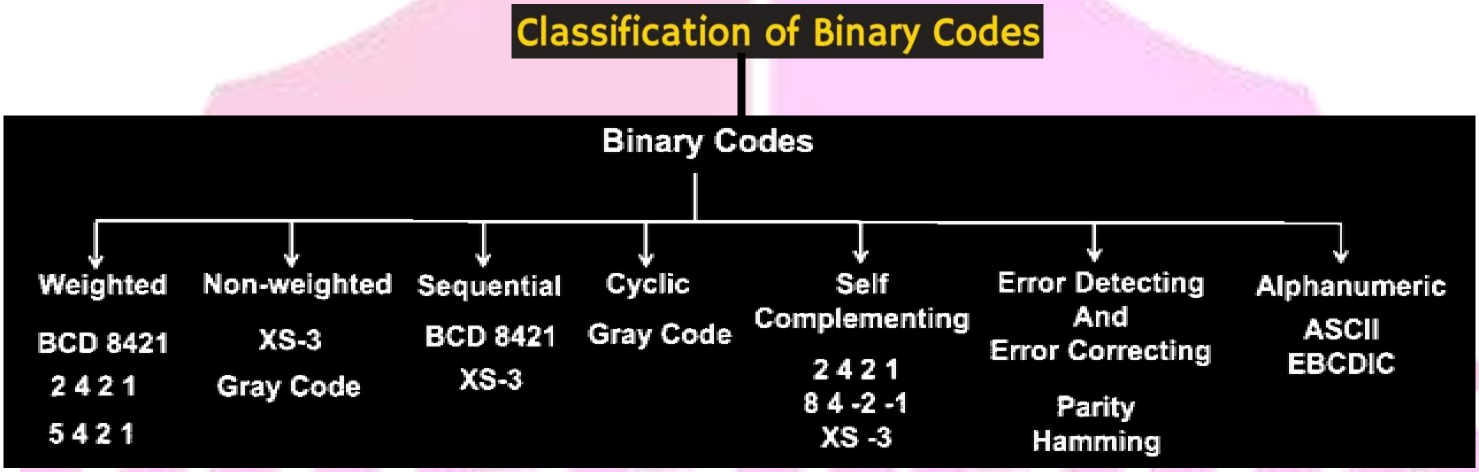
Carry 11
$(3215)_6$
+ $(5243)_6$
$(12502)_6$

Carry 11
$(X1Y2)_8$
+ $(41XY)_8$
$(7X20)_8$
X=3, Y=6

Carry 1 1
$(X7Y6)_8$
+ $(4X33)_8$
$(605Y)_8$
X=1, y=1

Binary Codes

- In the coding, when numbers or letters are represented by a specific group of symbols, it is said to be that number or letter is being encoded.
- The group of symbols is called as code.
- The digital data is represented, stored and transmitted as group of bits.
- This group of bits is also called as binary code.



Classification of Binary Codes

Binary Codes

Weighted: BCD 8421, 2 4 2 1, 5 4 2 1

Non-weighted: XS-3, Gray Code

Sequential: BCD 8421, XS-3

Cyclic: Gray Code

Self Complementing: 2 4 2 1, 8 4 -2 -1, XS -3

Error Detecting And Error Correcting: Parity, Hamming

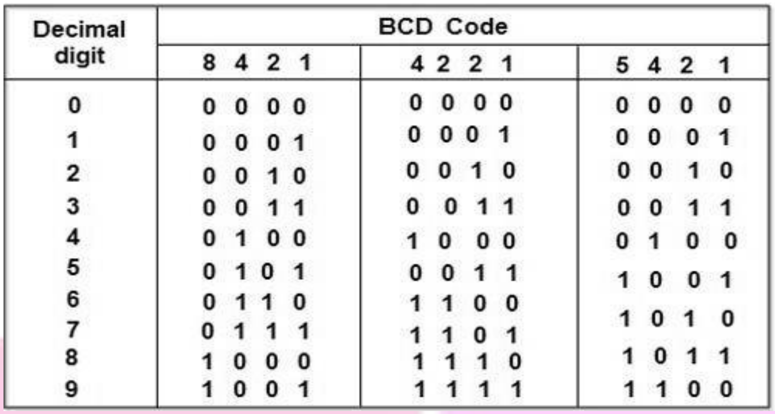
Alphanumeric: ASCII, EBCDIC

Weighted code

- Each binary bit is assigned by a “weight” and values depend on the position of the binary bit.
- The sum of the weights of these binary bits, whose value is 1 is equal to the decimal digit which they represent.
- In other words, if w1, w2, w3 and w4 are the weights of the binary digits, and x1, x2, x3 and x4 are the corresponding bit values, then the decimal digit N=w4x4 + w3x3+w2x2+w1x1 is represented by the binary sequence x4x3x2x1.
- **Example of Weighted codes:** BCD, 8421, 2421, 7421, 5421, 4221, 5211, 3321 etc.
- **Weighted codes are used in:**
  a) Data manipulation during arithmetic operation.
  b) For input/output operations in digital circuits.
  c) To represent the decimal digits in calculators, volt meters etc.

Binary coded Decimal Code (8421)

- Binary-coded decimal is a code to represent a given decimal number in an equivalent binary form.
- Generally BCD assigns a four-digit binary code to each digit 0 to 9 in a decimal (base 10) number.
- BCD number which is representing the decimal number greater than 9 is called packed BCD. Ex:- $(12)_{10}$=(10010), $(148)_{10}$=(101001000) etc
- There are several BCD codes like 8421, 2421, 3321, 4221, 5211, 5311, 5421, etc.
- The most common BCD code is the 8421 BCD code.
- We can convert a binary number to BCD by using “Shift ADD-3” method.
- BCD code is a Sequential code. In sequential codes, each succeeding code is one binary number greater than its preceding code,



Decimal digit	BCD Code		
	8 4 2 1	4 2 2 1	5 4 2 1
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1	0 0 1 1
4	0 1 0 0	1 0 0 0	0 1 0 0
5	0 1 0 1	0 0 1 1	
6	0 1 1 0	1 1 0 0	1 0 0 1
7	0 1 1 1	1 1 0 1	1 0 1 0
8	1 0 0 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 0 0

Conversion of decimal number to BCD

Convert each digit to 4 bit binary code

Example: **$(49)_{10}$ = $(1001001)_{BCD}$**

Conversion of BCD to Decimal

Pair 4 bits of given binary and convert in equivalent Decimal.

Example: **$(10011101011001)_{BCD}$ = $(2759)_{10}$**

BCD Addition

Perform BCD addition by following the binary addition and check result with some following cases:-

- **Case1:-** If the resultant sum is less than or equal to 9 and final carry is 0 then the result is correct.

  Example: 0100 + 0101 = 1001 (<=9 and no carry occurs) correct

- **Case2:-** If the resultant sum is less than 9 and final carry is 1 then the result is incorrect. To find the correct result we need to add 6 (0110) in the resultant sum.

  Example: 1001 + 1001 = 10010 (<=9 and carry occurs) incorrect

  Therefore add 0110 (6) in the result

  10010+ 0110 =11000 (Correct)

- **Case3:-** If the resultant sum is greater than 9 and final carry is 0 then the result is incorrect. To find the correct result we need to add 6 (0110) in the resultant sum.

  Example: 1001 + 0101 = 1110 (>=9 and no carry occurs) incorrect

  Therefore add 0110 (6) in the result

  1110+ 0110 =10100 (Correct)

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is more complicated than binary number.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

Non-Weighted Codes

- In non-weighted or un-weighted codes, the digit value does not depend upon their position i.e., each digit position within the number is not assigned fixed value.
- Examples of non-weighted codes: Excess-3 code and gray code.
- Non weighted codes are used in:

a) To perform certain arithmetic operations.
b) Shift position encodes.
c) Used for error detecting purpose.

Excess-3 code

- The excess-3 code of a decimal number is achieved by adding the number 3 to the 8421 code.
- In other words we can say excess-3 code is a 4 bit code which can be achieved by adding 3 in 8421 code.
- Excess-3 code is non weighted code.
- It is also a reflective or self complementing and sequential code.
- It is represented as xs-3 or x-3 code.

DECIMAL NUMBER	BINARY NUMBER	8421 CODE	EXCESS-3
0	0	0000	0011
1	1	0001	0100
2	10	0010	0101
3	11	0011	0110
4	100	0100	0111
5	101	0101	1000
6	110	0110	1001
7	111	0111	1010
8	1000	1000	1011
9	1001	1001	1100

GRAY Code

- This code doesn’t have any weights. So, it is an un-weighted code.
- In the gray code the successive values are differed in one bit position only. Hence, this code is called as unit distance code.
- It is also called as reflected binary code or minimum error code or cyclic code.
- Binary code is converted to gray code to reduce switching operation.

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

Binary to Gray code conversion

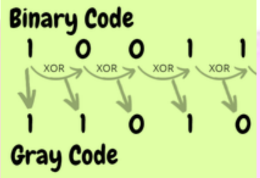
Step1:- Write MSB bit as given.

Step2:- add the MSB to the next bit, write the sum and ignore the carry.

Step3:- Repeat step2 till last bit.

Or use Xor for conversion

Example:



Gray to Binary conversion

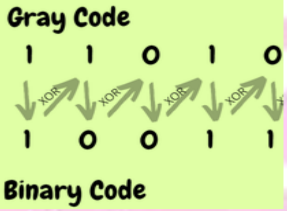
Step1:- Write MSB bit as given.

Step2:- add the MSB to the next bit, write the sum and neglect the carry.

Step3:- Repeat step2 till last bit.

Or use Xor for conversion

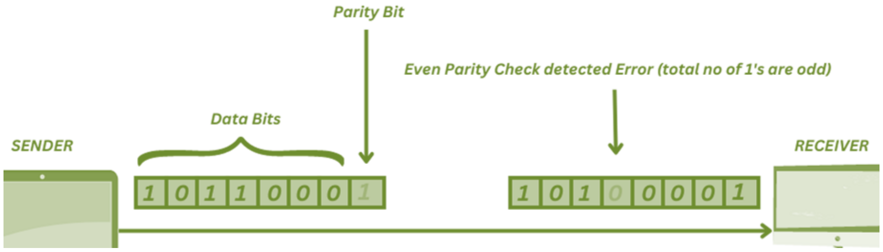
Example:



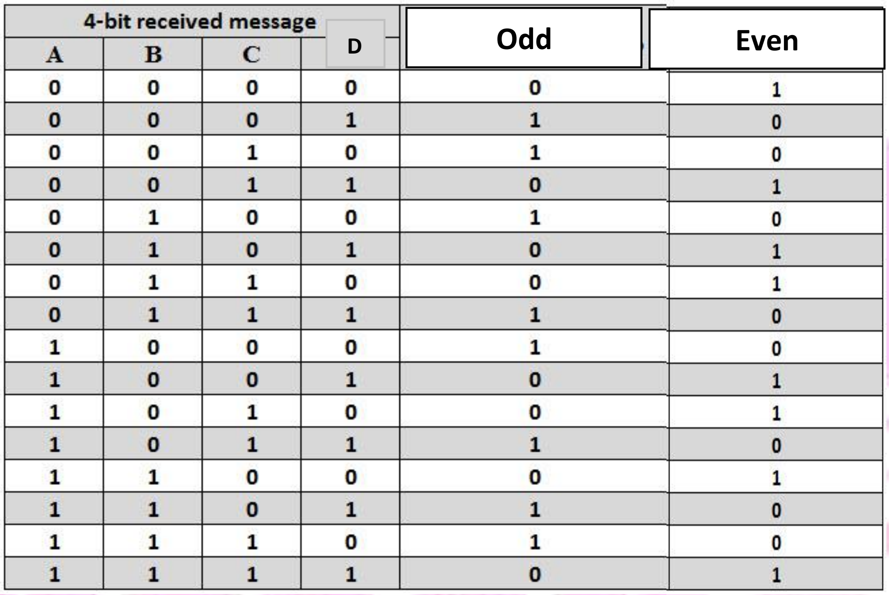
Parity

- Parity is used to check error in data transmission. Parity bit is an extra bit added to the message to detect error during data transmission.
- During data transmission the external noise can change bits from 1 to 0 or 0 to 1 which changes the meaning of the actual message and is called error.
- For efficient data transfer, there should be an error detection and correction codes.
- There are three error detection techniques Parity Bit, CheckSum and Cyclic Redundancy Check (CRC).
- A Parity Generator is a combinational logic circuit that generates the parity bit in the transmitter.
- On the other hand, a circuit that checks the parity in the receiver is called Parity Checker.
- A combined circuit or device of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data.
- There are two parity bit checker:-
  - ✓ Even parity checks
  - ✓ Odd parity checks

Even Parity:- In even parity, the parity bit is set (1) or reset (0) to ensure that the total number of 1's in the code, including the parity bit, is an even number. If any single bit error occurs during transmission, the parity check will detect it. However, it cannot detect errors where an even number of bits are flipped (e.g., two bits).



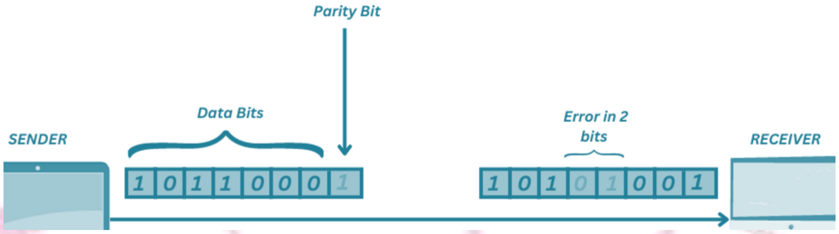
Odd Parity:- In odd parity, the parity bit is set (1) or cleared (0) to ensure that the total number of 1s in the code, including the parity bit, is an odd number. Similar to even parity, it can detect single bit errors but not errors where an odd number of bits are flipped.



4-bit received message			D	Odd	Even
A	B	C			
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

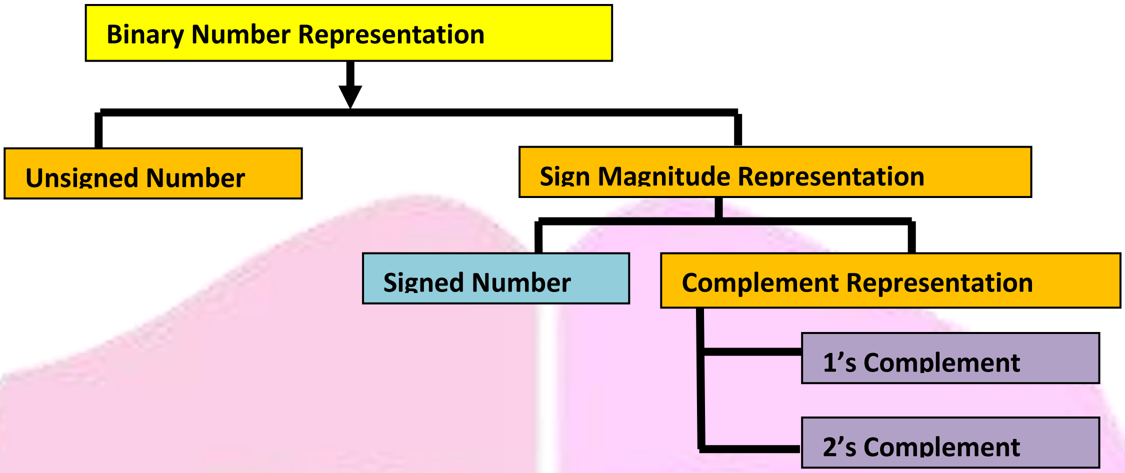
Disadvantages of parity bit check

1. It detect error in a single bit only and it also cannot determine the exact location of error in the data bit.
2. If the number of bits in even or odd parity check increase or decrease (data changed) but remained to be even or odd respectively then it won't be able to detect error as the number of bits are still even or odd.



Hamming code- error detection and correction

- Richard W. hamming invented Hamming codes to detect and correct the single bit error in data. later on it is extended up to 2-error detecting codes.
- Hamming codes are created because parity check cannot correct error in the data. The Hamming codes are inserted to any block length of data between actual data and redundancy bits.
- In the hamming code we add some redundant bits with the original message (data bits) to detect and correct the error.
- First we encode the data by finding and adding the value of redundant bits.
- After that we decode the data to find the single bit error and if error occurs we can correct it.


Binary Number Representation
Unsigned Number
Sign Magnitude Representation
Signed Number
Complement Representation
1's Complement
2's Complement

Unsigned Binary Numbers

- Only positive binary numbers can be represented.
- For n-bit unsigned binary numbers, all n-bits are used to represent the magnitude of the number. Need not to take extra bit.
- In unsigned binary number representation, using n-bits, we can represent the numbers from 0 to $2^n - 1$.

For example:

Represent $(18)_{10}$ in 6-bit unsigned number form.

$(18)_{10} = (010010)_2$

Sign Magnitude Representation

- In sign-magnitude representation, the Most Significant bit of the number is a sign bit and the remaining bit represents the magnitude of the number in binary form.
- For positive numbers, the sign bit is 0 and for negative number, the sign bit is 1.
- Using signed binary number representation both positive and negative numbers can be represented.

For example: (+10 and -10)   00001010 and 10001010

Representation of 8-bit sign-magnitude number, the MSB is a sign bit and the remaining 7 bits represent the magnitude.

There are three different ways the signed binary numbers can be represented.

a) Signed Magnitude Representation
b) 1's Complement Representation
c) 2's Complement Representation

Signed Number

Using n-bits, the range of numbers that can be represented in Sign Magnitude Representation is from $-(2^{n-1} - 1)$ to $(2^{n-1} - 1)$.

1's Complement Representation

- We represent negative binary number using 1's complement form.
- 1's complement of a binary number is obtained by inverting the binary bits from 0 to 1 and 1 to 0.

For example: $(-24)_{10}$ to represent a negative number, first write the binary representation of its positive parts.

$(11000)_2 = (00111)_2$ (1's complement representation of -24)

Subtraction Using 1’s complement

Step 1: Represent the Numbers in Binary
Step 2: Find the 1's Complement of the Subtrahend
Step 3: Add the Minuend to the 1's Complement of the Subtrahend.
Step 4: If the result has a carryover, then add that carry over in the least significant bit
Step 5: If there is no carryover, then take the 1’s complement of the resultant, and it is negative.

2’s Complement Representation

- In 2’s complement representation also, the representation of the positive number is same as 1’s complement and sign-magnitude form.
- But the representation of the negative number is different.
  For example: Representation of -25 in 2’s complement form
  Rules:
- Write the number corresponding to +25.
- Find 1’s complement of the +25 then add 1 to get 2’s complement.

Or

- After the first ‘1’ is encountered, invert all the 1s in the number with 0s and 0s in the number with 1s (including the sign bit)
- The resultant number is 2’s complement representation of the number -25.

Subtraction Using 2’s complement

Step 1: Represent the Numbers in Binary
Step 2: Find the 2's Complement of the Subtrahend
Step 3: Add the Minuend to the 2's Complement of the Subtrahend.
Step4: If the final carry over of the sum is 1, then it is dropped and the result is positive.
Step5: If there is no carry over, then 2's complement of the sum is the final result and it is negative.

Difference between 1’s complement and 2’s complement

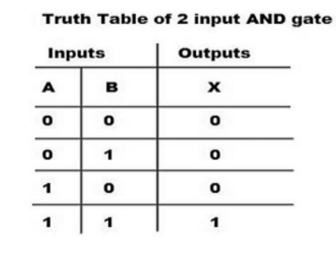
S.No.	1’s complement	2’s complement
1.	To get 1’s complement of a binary number, simply invert the given number.	To get 2’s complement of a binary number, simply invert the given number and add 1 to the least significant bit (LSB) of given result.
2.	1’s complement of binary number 110010 is 001101	2’s complement of binary number 110010 is 001110
3.	Simple implementation which uses only NOT gates for each input bit.	Uses NOT gate along with full adder for each input bit.
4.	Can be used for signed binary number representation but not suitable as ambiguous representation for number 0.	Can be used for signed binary number representation and most suitable as unambiguous representation for all numbers.
5.	0 has two different representation one is -0 (e.g., 1 1111 in five bit register) and second is +0 (e.g., 0 0000 in five bit register).	0 has only one representation for -0 and +0 (e.g., 0 0000 in five bit register). Zero (0) is considered as always positive (sign bit is 0)
6.	For k bits register, positive largest number that can be stored is ($2^{(k-1)}$ -1) and negative lowest number that can be stored is -($2^{(k-1)}$ - 1).	For k bits register, positive largest number that can be stored is ($2^{(k-1)}$ -1) and negative lowest number that can be stored is -($2^{(k-1)}$).
7.	end-around-carry-bit addition occurs in 1’s complement arithmetic operations. It added to the LSB of result.	end-around-carry-bit addition does not occur in 2’s complement arithmetic operations. It is ignored.
8.	1’s complement arithmetic operations are not easier than 2’s complement because of addition of end-around-carry-bit.	2’s complement arithmetic operations are much easier than 1’s complement because of there is no addition of end-around-carrybit.
9.	Sign extension is used for converting a signed integer from one size to another.	Sign extension is used for converting a signed integer from one size to another.

Logic Gate

- An elementary building block of a digital circuit.
- Used to create simple to complex logic circuit or integrated circuits.
- Used by complex microprocessor or ICs
- Accept one or more inputs and produce one output with some logical condition between them.
- Deal with binary signals i.e. True or False, ON or OFF, 1 or 0;
- The state of the output is dependent on the input states.
- All logic gates implements some Boolean function which correlates output with input through some logical operation
- Logic gates are mainly designed with the electronic switches using diodes and transistors.
- There are three basic gates AND, OR and NOT.
- XOR, XNOR, NAND and NOR gates are derived gates.
- NAND and NOR gates are also known as universal logic gates.
- An XOR gate is inequality detector gate and can be used in comparator, adders, parity generators etc.
- Digital systems are constructed using logic gates.

AND Gate

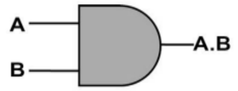
- It will produce a high output when all the inputs are high otherwise the output is low . (X=A.B)



Truth Table of 2 input AND gate

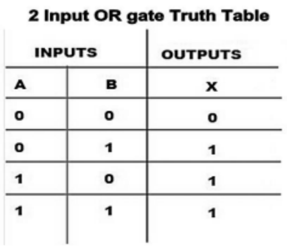
Inputs		Outputs
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Symbol :



OR Gate

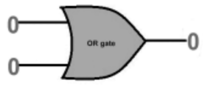
- An OR gate produces a high output when any one
- of the input is high .It produces a low output when all the
- inputs are low. (X=A+B)



2 Input OR gate Truth Table

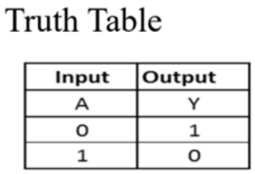
INPUTS		OUTPUTS
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Symbol :



NOT Gate

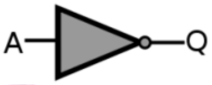
It produces high output when the input is low and vice versa. The NOT gate is also called as an inverter. (Q=A')



Truth Table

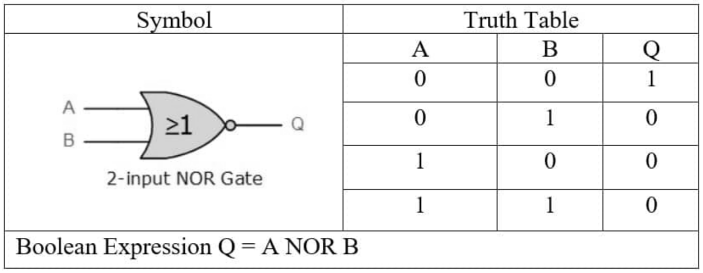
Input	Output
A	Y
0	1
1	0

Symbol :



NOR Gate

NOR gate is OR gate followed by NOT gate. (X=(A+B)')

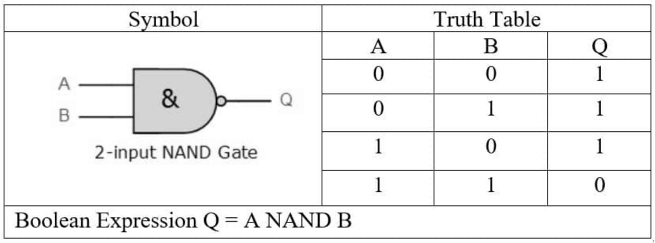

A
B
≥1
Q
2-input NOR Gate

Symbol	Truth Table		
	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	0

Boolean Expression Q = A NOR B

NAND Gate

NAND gate is combination of AND and NOT gates. The NAND gate provides AND functions with inverted output.

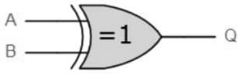

A
B
&
Q
2-input NAND Gate

Symbol	Truth Table		
	A	B	Q
	0	0	1
	0	1	1
	1	0	1
	1	1	0

Boolean Expression Q = A NAND B

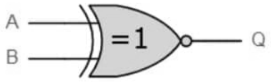
XOR Gate

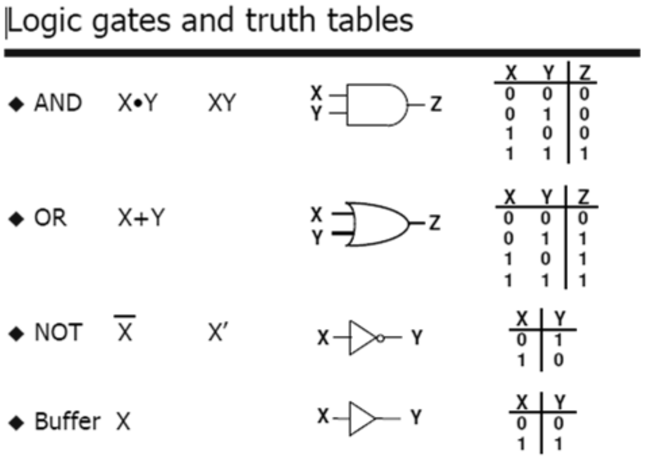
- Exclusive OR gate is basically designed to exclude the condition of standard OR gate so as to generate real binary addition.
- The output of an XOR gate is at logic ‘1’ when the inputs are dissimilar and at logic ‘0’ when the inputs are similar.
- As XOR gate generates logic ‘1’ only when inputs are not equal, hence this gate is also known as “inequality detector” gate.

Symbol	Truth Table		
	A	B	Q
 2-input Ex-OR Gate	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression Q = A XOR B			

XNOR Gate

- XNOR is obtained by the combination of NOT and XOR gates.
- The output of an XNOR gate is at logic ‘1’ when the inputs are similar and at logic ‘0’ when the inputs are dissimilar. The logic equation for two input XOR gate is given by
- As XNOR gate generates logic ‘1’ only when both the inputs are equal, hence this gate is also known as “equality detector” gate.

Symbol	Truth Table		
	A	B	Q
 2-input Ex-NOR Gate	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression Q = A XNOR B			



Logic gates and truth tables

◆ AND X•Y XY

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

◆ OR X+Y

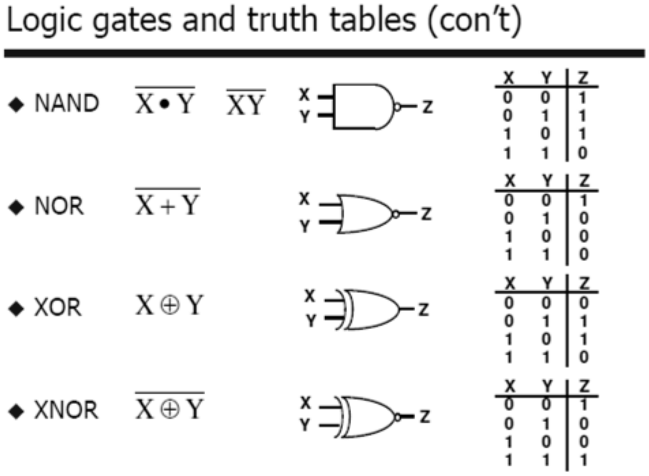
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

◆ NOT $\overline{X}$ X′

X	Y
0	1
1	0

◆ Buffer X

X	Y
0	0
1	1



Logic gates and truth tables (con’t)

◆ NAND $\overline{X \bullet Y}$ $\overline{XY}$

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

◆ NOR $\overline{X + Y}$

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

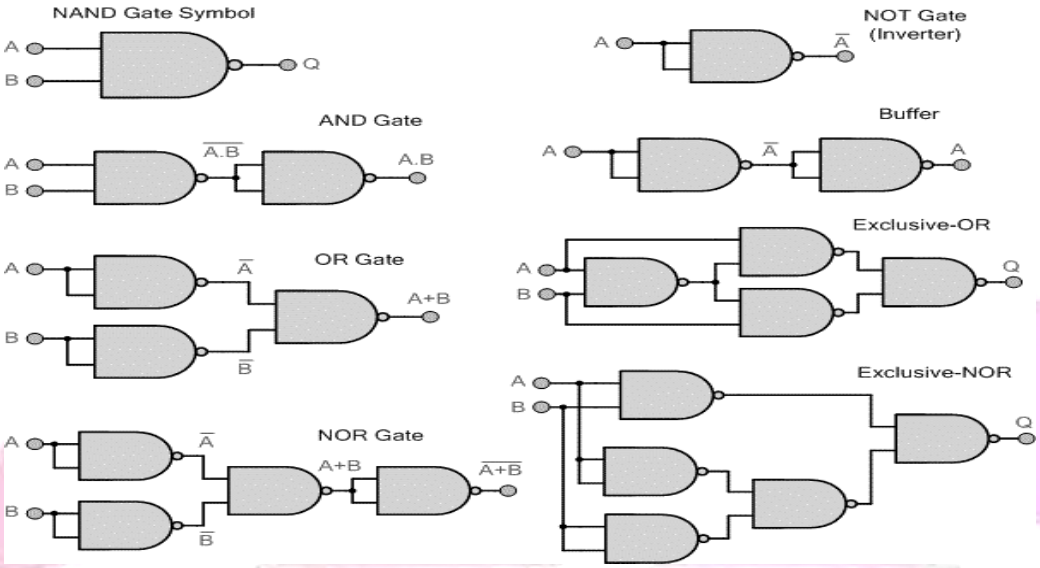
◆ XOR $X \oplus Y$

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

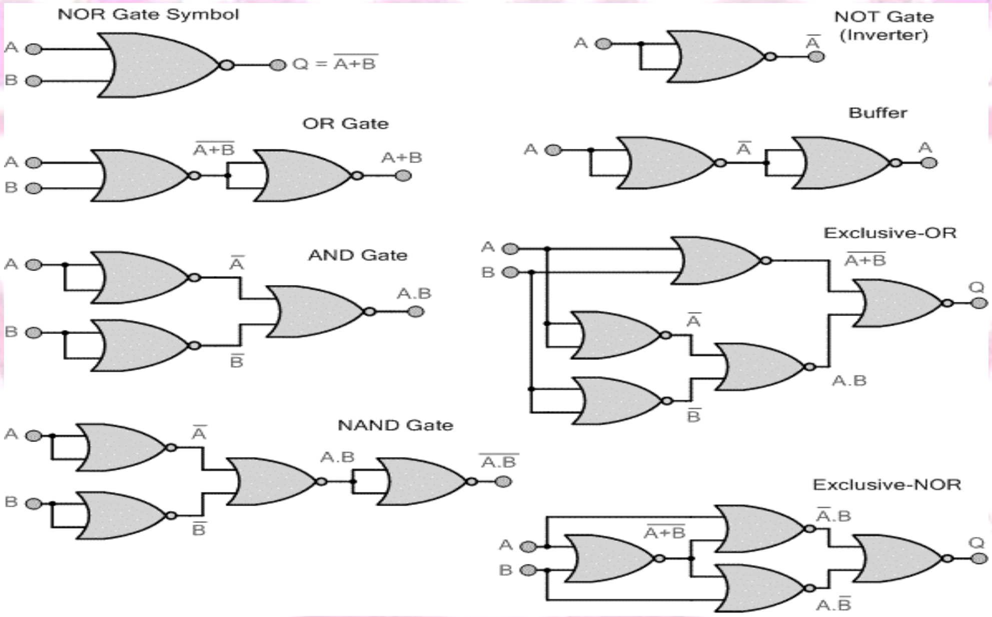
◆ XNOR $\overline{X \oplus Y}$

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Logic Gates by using NAND Gate



Logic gates using NOR Gate



Implementation of All GATES using universal GATES

Implemented gate	NAND GATE	NOR GATE
NOT	1	1
OR	3	2
AND	2	3
EXOR	4	5
EXNOR	5	4
NOR	4	---
NAND	---	4

Boolean Algebra

- Boolean algebra is a mathematical function to represent the logical conditions.
- To process the output of a complex circuit we use boolean algebra.
- All arithmetic operations performed with Boolean quantities have one of two possible outcomes: either 1 or 0.

Laws of Boolean Algebra

1. **Commutative Law:-**

   $A+B = B+A$

   $A.B = B.A$

2. **Associative Law:-**

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

   $(A.B).C = A.(B.C)$

3. **Distributive Law:-**

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

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

4. **Absorption Law:-**

   $A+AB = A$

   $A(A+B) = B$

5. **Idempotence Law:-**

   $A+A = A$

   $A.A = A$

6. **Ivolutionary Law:-**

   $(A')' = A$

Some basic rules of Boolean algebra:-

- $A+0 = A$
- $A+1 = 1$
- $A.0 = 0$
- $A.1 = A$
- $A+A = A$
- $A+A' = 1$
- $A.A = A$
- $A.A' = 0$
- $(A')' = A$
- $A+A'B = A+B$
- $(A+B)(A+C) = A+BC$

Boolean Algebra Theorms

DeMorgan Theorms:-

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

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

In general form

$(A1.A2.A3.......An)' = A1'+A2'+A3'+........+An'$

$(A1+A2+A3+........+An)' = A1'A2'A3'......An'$

Consensus Theorem/Redundancy Theorem

A variable is associated with some variable and its compliment is associated with some other variable and the next term is formed by the left over variables, then the term becomes redundant.
This theorem is used to eliminate redundant term.

It is applicable only if a Boolean function,

1. Contains 3-variables.
2. Each variable used two times.
3. Only one variable is in complemented or uncomplemented form.

Then, the related terms to that complemented and uncomplemented variable is the answer.
Consensus theorem can be extended to any number of variables.

For Example:-

1. AB+A'B+BC =AB+A'B

2. (A+B) (B'+ C) (C + A) = (A+B) (B'+ C) = AB' +BC

Duality Theorem

"Dual expression" is equivalent to write a negative logic of the given Boolean relation.
For this we have to:-
(i) change each OR sign by an AND sign and vice-versa
(ii) complement any '0' or '1' appearing in expression
(iii) keep literals/variables as it is.

Example:- A'BC+ABC'+AB'C' = (A'+B+C) (A+B+C') (A+B'+C')

Self Dual Theorm

A boolean function is said to be Self dual if and only if its dual is equivalent to the given boolean function.
The given function is neutral i.e., (the number of minterms is equal to the number of maxterms).
The function does not contain two mutually exclusive terms.
Ex:- f(A,B,C) = AB+BC+AC
It dual (A+B).(B+C).(A+C) is equivanent to the same function.

Important Points

- Every Self-dual function is neutral but every neutral function is not Self-dual.
- Self-duality is closed under complement i.e, the complement of a Self-dual function is also Self-dual.
- Total number of combinations with n variables:- $\mathbf{2^n}$
- Total number of boolean function with n variables:- $\mathbf{2^{2^n}}$
- Total number of dual with n variables: $\mathbf{2^{2^{n-1}}}$

Complementary Theorem

For obtaining complement expression we have to
(i) change each OR sign by AND sign and vice-versa
(ii) complement any '0' or '1' appearing in expression.
(ii) complement the individual literals/variables.
Ex:- The compliment of the function- AB'C+A'BC'+ AB'C'= (A'+B+C') (A+B'+C) (A'+B+C)

Representation of Boolean Functions

Representation of Boolean Functions is called boolean expression.

A truth table of a boolean function can be represented as a boolean expression.

A function of 'n’ Boolean variables denoted by (A1,A2, A3.......An) is another variable of algebra and takes one of the two possible values either 0 or 1.

Boolean Function can be represented as:-

a) **Canonical form:-** All the terms contain all the variables either in complementary or in uncomplentary form
   Example: F(A, B, C) = A’BC + ABC + A’BC’

b) **Minimal form:-** Minimum numbers of literal/variables
   Example: F(A, B, C)=A+ABC+A’BC=A

Minterms and Maxterms

- As we know n-binary variables have 2^n possible combinations.
- Minterm is a product term, it contains all the variables either complementary or uncomplimentary form for that combination the function output must be '1'.
- Maxterm is a sum term, It contains all the variables either complementary or uncomplimentary form for that combination the function output must be '0'.

Sum of Product (SOP) Form

- The SOP expression usually takes the form of two or more variables ANDed together. Each product term may be minterm or implicant.
- Y = A’BC+ AB’+ AC
- Y = AB’+BC’
- This form is also called the "disjunctive normal form".
- The SOP expression is used most often because it tends itself nicely to the development of truth tables and timing diagrams.
- SOP circuits can also be constructed easily by using a special combinational logic gates called the AND-OR-INVERTER gate.
- SOP forms are used to write logical expression for the output becoming Logic 1.

Notation of SOP expression:-

- f(A, B, C) = Σ(3, 5, 6, 7)
- Y = $m_3 + m_5 + m_6 + m_7$
- Y= A’BC+ AB’C + ABC’ + ABC

Input(3 variables)			Output (Y)
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Product of Sum (POS) Form

- The POS expression usually takes the form of two or more order variables within parentheses, ANDed with two or more such terms.
- Ex:- Y = (A+B’+C) (BC’ + D)
- This form is also called the "Conjunctive normal form".
- Each individual term in standard POS form is called Maxterm.
- POS forms are used to write logical expression for output be coming Logic '0’.

Input(3 variables)			Output (Y)
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

From the above truth table,
we get:-
f(A B C) = π m(0, 1, 2, 4)
Y = m_0 X m_1 X m_2 X m_4
or
Y = (A+B+C)(A+B+C’)(A+B’+C)(A’+B+C)

We also conclude that From the above truth table, and from above equations:-
If Y = Σm(3, 5, 6, 7)
Then Y = πm(0, 1, 2, 4)

Standard Sum of Product Form

- In this form the function is the sum of number of product terms where each product term contains all the variables of the function, either in complemented or uncomplemented form.
- It is also called canonical SOP form or expanded SOP form.
- The function Y = A + BC’ can be represented as canonical form as:-
  Y = ABC+AB’C+AB’C’+ABC’+A’BC’

Standard Product of Sum Form

This form is also called canonical POS form or expanded POS form.
The function Y=(B+C’)(A+B’) can be represented in canonical form as:
Y = (B+C’+AA’)(A+B’+CC’) = (B+C’+A) (B + C’ + A’)(A+B’+C)(A+B’+C’)

Truth Table Form

A truth table is a tabular form representation of all possible combinations of given function.
Y = A’B+B’C is represented as truth table:

Input(3 variables)			Output (Y)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Simplification of Boolean expressions

There are three methods to simplify the Boolean expression

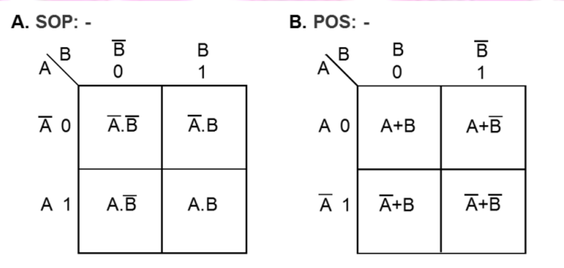
1. Boolean Laws and theorems
2. K maps
3. Quine Mc-Cluskey or Tabulation Method

Karnaugh map

- The "Karnaugh map" is a graphical method which provides a systematic method for simplifying and manipulating the Boolean expressions or to convert a truth table to its corresponding logic circuit in a simple, orderly process.
- In this technique, the information contained in a truth table or available in SOP or POS form is represented on K-map.
- Although this technique may be used for any number of variables, it is generally used up to 6-variables beyond which it becomes very cumbersome.
- In n-variable K-map there are 2^n cells.
- "Gray code" has been used for the identification of cells:

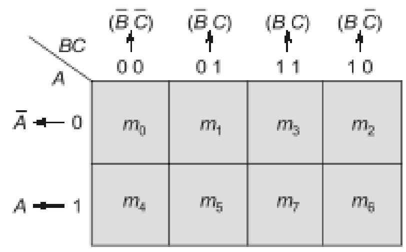
Two-variable K-Map

- Four cells
- Four minterms (maxterms)

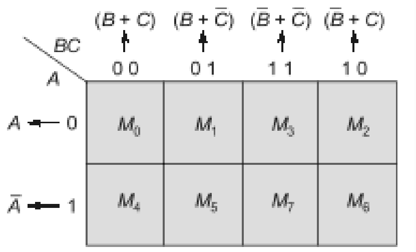


Three variable K-map

- Eight cells
- Eight minterms(maxterms)



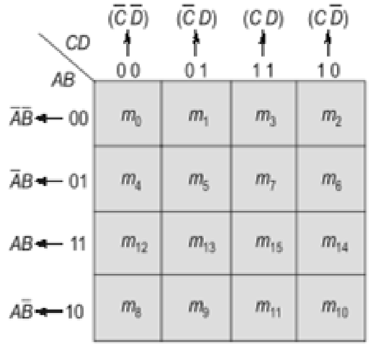
(For SOP)



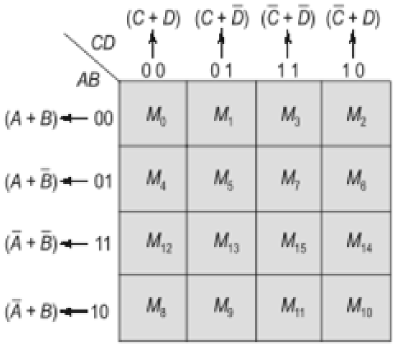
(For POS)

4- variable K-map

- Sixteen cells
- sixteen minterms (maxterms)



(For SOP)



(For POS)

K-MAP Simplification Rules

- Construct the K-map and place 1s in those cells corresponding to the 1's in the truth table.Place 0's in the other cells.
- Examine the map for adjacent 1's and loop those 1's which are not adjacent to any other 1's.
- These are called isolated 1's.
- Next, look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
- Loop any octet even it contains some 1's that have already been looped.
- Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum number of loops.
- Loop any pairs necessary to include any 1's that have not yet been looped, making sure to use the minimum number of loops.
- Form the OR sum of all the terms generated by each loop

Don't Care Condition

- Some logic circuits can be designed so that there are certain input combinations for which there are no specified output levels, usually because these input combinations will never occur.
- So, a circuit designer is free to make the output for any "don't care" condition either a 0 or 1 in order to produce the simplest output expression.
- The two conditions can be better understood by constructing the K-map for the given two conditions.

(i) In terms of SOP and don't care conditions.

$f(A,B,C,D) = \Sigma m(1, 3, 7, 11, 15)+d(0,2,5)$

(ii) In terms of POS and don't care conditions.

$f(A, B, C, D) = \pi m(4, 5, 6, 7, 8, 12). d(1, 2, 3, 9, 11,14)$

Implicants, Prime Implicants and Essential Prime Implicants

- **Implicant:** implicant is a product term on the given function for that combination the function output must be 1.
- **Prime Implicant:** Prime implicant is a smallest possible product term of the given function, removing any one of the literal from which is not possible.
- **Essential Prime Implicant:** Essential prime implicant is a prime implicant it must cover atleast one minterm, which is not covered by any other prime implicant.

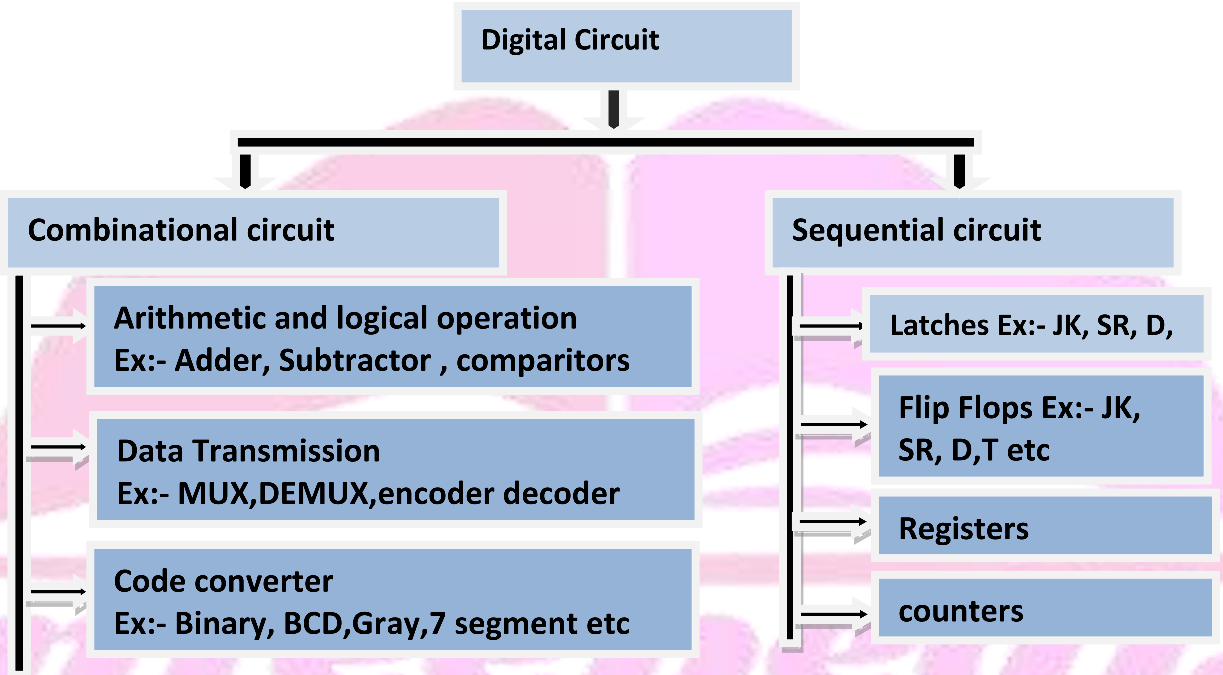
For the given K-map, find implicant, Prime implicant and Essential Prime Implicant can be represented as:-

A\BC	B'C'	B'C	BC	BC'
A'	1	1		1
A			1	1

Implicant:-5
Prime Implicants:- 4
Essential Prime Implicants:- 2

Combinational Circuit

- Digital circuit is made by using logic gates.


Digital Circuit
Combinational circuit
Sequential circuit
Arithmetic and logical operation
Ex:- Adder, Subtractor , comparitors
Data Transmission
Ex:- MUX,DEMUX,encoder decoder
Code converter
Ex:- Binary, BCD,Gray,7 segment etc
Latches Ex:- JK, SR, D,
Flip Flops Ex:- JK, SR, D,T etc
Registers
counters

There are two types of logic circuits

1. Combinational Circuits

- Formed by combination of different logic circuits
- Outputs depend on the current inputs
- Ex- Adder, Multiplexer, De multiplexer, Encoder, Decoder
- No memory allocation required

2. Sequential Circuits

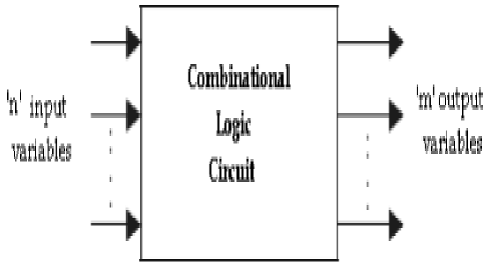
- Outputs depend on current and previous inputs
- Requires separating previous, current and future states or tokens.
- **Example:** Finite State Machines (FSMs), Pipelines, Latch,Flip flop etc.

Difference between Combinational and sequential circuits:

SNO	Combinational circuits	Sequential circuits
1	Output of any instance of time depends only upon the present input variables.	Output is generated dependent upon the present input variables and also on the basis of past history of these inputs.
2	Memory unit is not required. i.e. it doesn't allocate any memory to the elements.	Memory unit is required. i.e. it allocates any memory to the elements.
3	Faster	Slower
4	Easy to design	Difficult
5	Implemented using: logic gates	Implemented using flip flops
6	Used to perform arithmetic as well as boolean operations.	used to store data.
7	don't have capability to store any state.	capability to store any state or to retain earlier state.
8	don't have clock, they don't require triggering.	clock dependent they need triggering.

Combinational circuit

- Combination Logic Circuits are made up from basic gates (AND, OR, NOT) or universal gates (NAND, NOR) gates that are "combined" or connected together to produce more complicated switching circuits.
- These logic gates are the building blocks of combinational logic circuits.
- Examples for combinational digital circuits are Half adder, Full adder, Half subtractor, Full subtractor, Code converter, Decoder, Multiplexer, Demultiplexer, Encoder, ROM, etc.
- For n number of input variables to a combinational circuit, 2^n possible combinations of binary input states are possible.
- For each possible combination, there is one and only one possible output combination.
- A combinational logic circuit can be described by m Boolean functions and each output can be expressed in terms of n input variables.



Block diagram of a combinational logic circuit

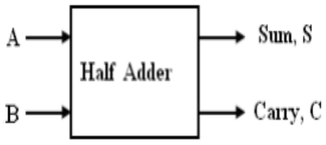
Adders

There are some following types of adder:-

1. Half adder
2. Full adder
3. n-bit parallel adder
4. Look ahead carry adder

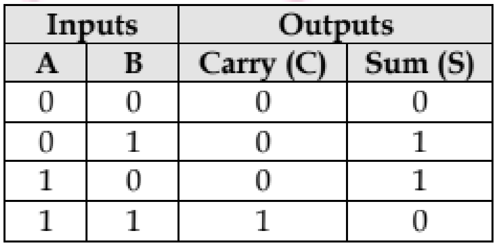
Half-Adder

- A half-adder is a combinational circuit that can be used to add two binary bits.
- It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.



Block schematic of half-adder

The truth table of a half-adder:



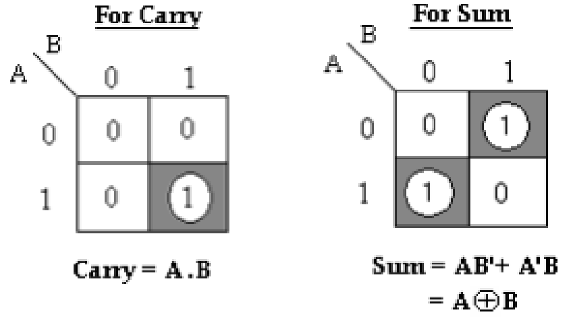
Inputs		Outputs	
A	B	Carry (C)	Sum (S)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table of half-adder

- K-map simplification for carry and sum:
- The Boolean expressions for the SUM and CARRY

S = A'B+ AB'= A ⊕ B

C = A . B



For Carry

A\B	0	1
0	0	0
1	0	1

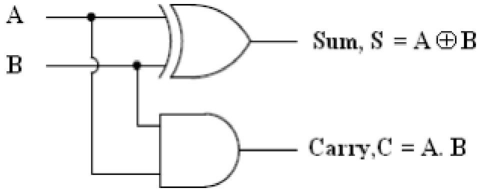
Carry = A.B

For Sum

A\B	0	1
0	0	1
1	1	0

Sum = AB'+ A'B
= A⊕B

The logic diagram of the half adder is:



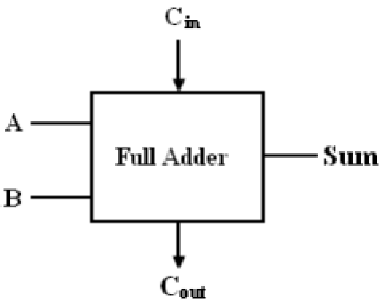
Drawback of half adder

Half adder does not handle and do processing on carries.

Full-Adder

- The full adder circuit overcomes the limitation of the half-adder.
- Full Adder is a combinational circuit that performs the addition of three bits (two significant bits and previous carry).
- It consists of three inputs and two outputs, two inputs are the bits to be added, the third input represents the carry form the previous position.
- Full adder circuit adds three bit binary numbers (X,Y,Z) & outputs two numbers of one bit binary numbers, Sum & Carry.

The block diagram of full adder:



Block schematic of full-adder

There are three input variables, eight different input combinations are possible. The truth table is shown below:

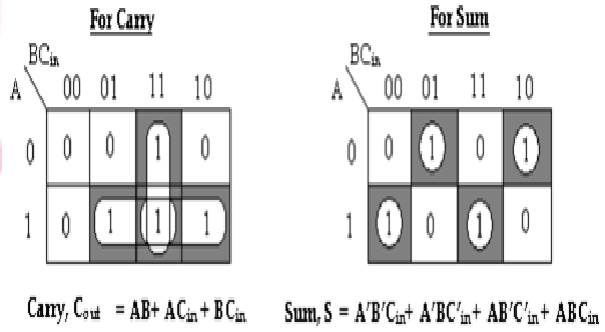
Inputs			Outputs	
A	B	C_{in}	Sum (S)	Carry (C_{out})
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The Karnaugh map:-

The Boolean expressions for the SUM and CARRY outputs Sum,

S = $A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in} = C_{in} \oplus (A \oplus B)$

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



For Carry

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

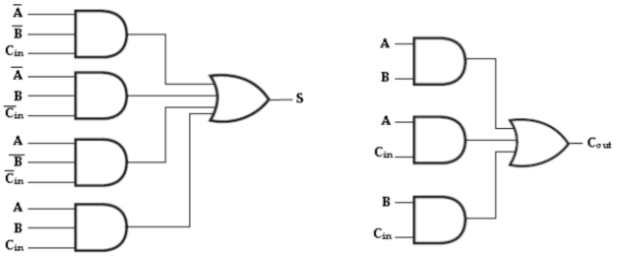
Carry, $C_{out} = AB + AC_{in} + BC_{in}$

For Sum

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

Sum, $S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$

The logic diagram for the Full adder sum and carry:-



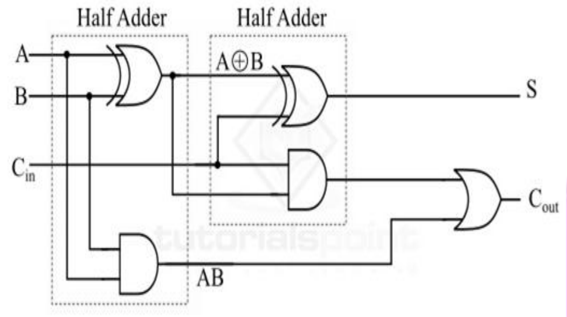
Implementation of full-adder in Sum of Products

Note:- The logic diagram of the full adder can also be implemented with two half adders and one OR gate.

Full Adder using Half Adder

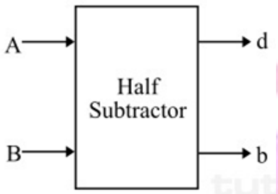
$S = C_{in} \oplus (A \oplus B)$

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



Half Subtractor

- It is used for the purpose of subtracting two single bit numbers.
- It contains 2 inputs and 2 outputs (difference and borrow).
- It produces the difference between the two binary bits at the input and also produces a borrow output (if any).
- In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.



Block Diagram

Truth Table of Half Subtractor:-

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

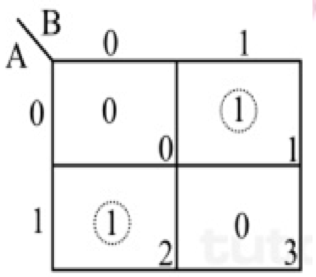
K-Map for Half Subtractor:-

Boolean expression for difference and borrow:-

d = A'B+ AB'= A ⊕ B

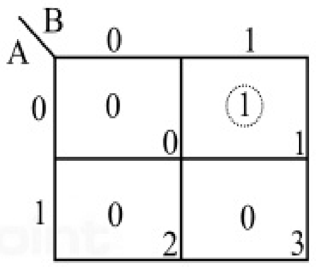
b = A' . B

For Difference (d)



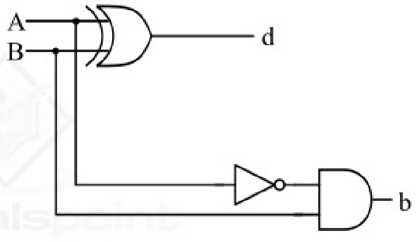
d = A'B + AB'

For Borrow (b)



b = A'B

Logic circuit diagram of the half subtractor:-



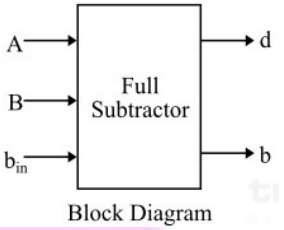
Circuit Diagram

Drawback of half Subtractor

Half subtractors do not take into account “Borrow-in” from the previous circuit. and are not suitable for more complex arithmetic operations.

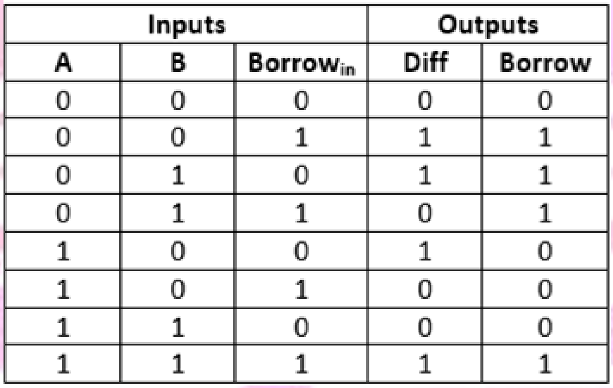
Full Subtractor

A full-subtractor is a combinational circuit that has three inputs A, B, b_{in} and two outputs d and b.
Where, A is the minuend, B is subtrahend, b_{in} is borrow produced by the previous stage, d is the difference output and b is the borrow output.



Block Diagram

Truth Table for full subtractor:-



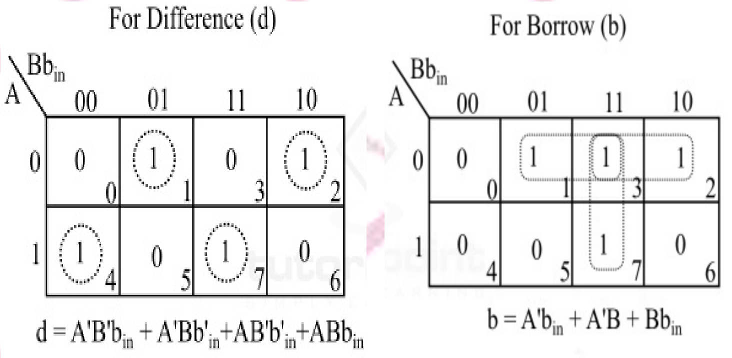
Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for full Subtractor:-

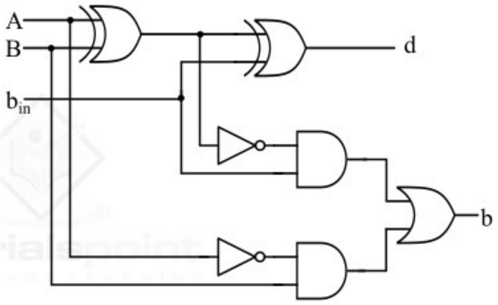
Boolean expression for difference and borrow:-

Difference d=$A \oplus B \oplus b_{in} = A'B'b_{in} + AB'b'_{in} + A'Bb'_{in} + ABb_{in}$

Borrow b=$A'B + (A \oplus B)'b_{in}$

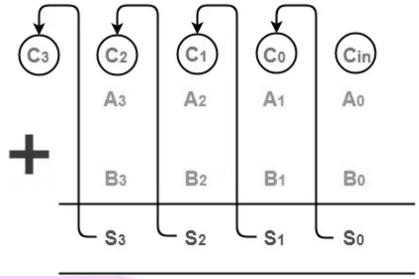

For Difference (d)
For Borrow (b)
Bb_{in}
A
00
01
11
10
0
1
$d = A'B'b_{in} + A'Bb'_{in} + AB'b'_{in} + ABb_{in}$
$b = A'b_{in} + A'B + Bb_{in}$

Logic circuit diagram of the full subtractor:-



Note:- Full Subtractor can be implemented by using 2 Half Subtractors and an OR gate.

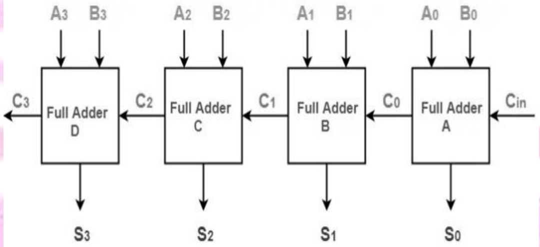
n-bit parallel adder



- It is also called ripple carry adder.
- It is used for the purpose of adding two n-bit binary numbers.
- It requires n full adders in its circuit for adding two n-bit binary numbers.
- Here we will discuss a 4 bit parallel adder to add 4 bit binary number.
- A 4-bit parallel adder consists of four full adders connected in parallel. Each full adder takes in two input bits, along with a carry input from the previous full adder.
- It produces a sum output bit and a carry output bit.

Disadvantages of parallel Adder:-

- For n bit addition we need n full adder leads a complex circuit.
- N bit parallel adder does not allow to use all the full adders simultaneously.
- The carry propagation delay (delay associated with the travelling of carry bit) increases as circuit grows.
- After designing a complex circuit only addition is performed not subtraction.

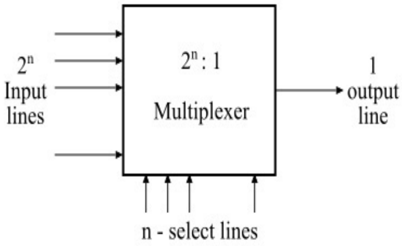


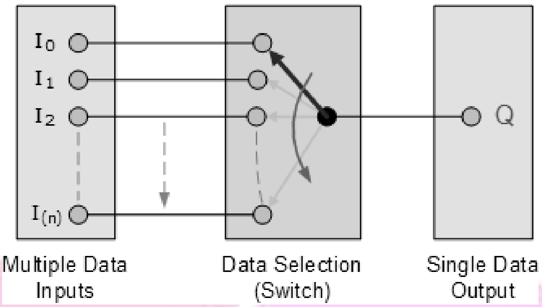
Carry lookahead adder

- In the n bit parallel adder each full adder has to wait for its carry-in from its previous stage full adder.
- Thus, nth full adder has to wait until all (n-1) full adders have completed their operations.
- This causes a delay and makes n-bit parallel adder extremely slow.
- To overcome this disadvantage, we can use Carry Look Ahead Adder.
- Carry Look Ahead Adder is an improved version of the n bit parallel adder.
- It generates the carry-in of each full adder simultaneously without causing any delay.
- The time complexity of carry look ahead adder = Θ (logn).
- Here a carry signal will be generated in two cases:
  - ✓ Input bits A and B are 1
  - ✓ When one of the two bits is 1 and the carry-in is 1.

Multiplexer or MUX

- It accept multiple data inputs and produce single output depending on control or select inputs.
- Here the input combination and selection lines are dependent to each other.
- If there are n selection lines then the input lines will be maximum 2^n.
- And if there are m input lines the $\log_2 m$ selection lines.
- Multiplexers are mainly used to increase amount of the data that can be sent over the network within certain amount of time and bandwidth.
- Multiplexers are data n selector, parallel to serial convertor, many to one circuit etc.





Types of Multiplexer

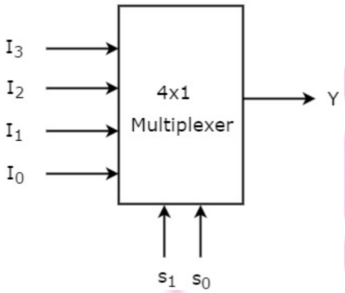
1. 2-to-1 MUX: Selects 1 out of 2 inputs.
2. 4-to-1 MUX: Selects 1 out of 4 inputs.
3. 8-to-1 MUX: Selects 1 out of 8 inputs.
4. 16-to-1 MUX: Selects 1 out of 16 inputs

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s0 and one output Y.

The block diagram of 4x1 Multiplexer:

One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines.



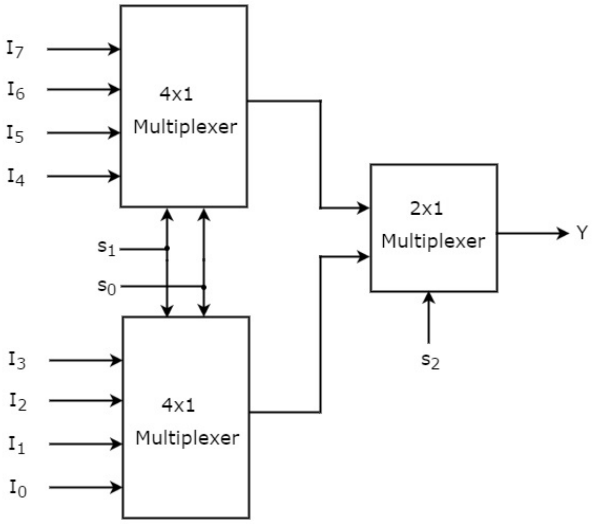
Truth table of 4x1 Multiplexer

The Boolean function for output

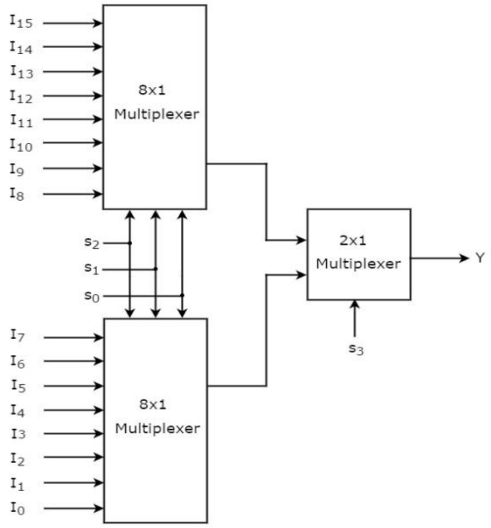
$Y=S_1'S_0'I_0+S_1'S_0I_1+S_1S_0'I_2+S_1S_0I_3$

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

The block diagram of 8x1 Multiplexer using 4 x 1 multiplexer



The block diagram of 16x1 Multiplexer using 8 x 1 multiplexer:

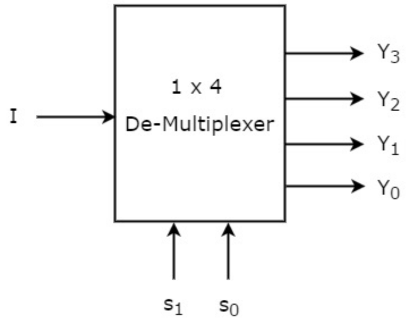


Applications of MUX

- Implementation of Digital Circuits
- Control Unit of CPU
- Parallel to Serial Data Conversion
- Computer Memory
- Data Routing
- Analog to Digital Conversion
- **Signal Selection**: In audio and video systems, MUX is used to select different input signals (e.g., different channels or inputs in a TV).

Demultiplexer or DEMUX

- A De-multiplexer is a combinational circuit that has only 1 input line and maximum 2^N output lines.
- The input will be connected to one of these outputs based on the values of selection lines.



Types of Demultiplexer:

1. 1-to-2 DEMUX: Routes 1 input to 1 of 2 outputs.
2. 1-to-4 DEMUX: Routes 1 input to 1 of 4 outputs.
3. 1-to-8 DEMUX: Routes 1 input to 1 of 8 outputs.
4. 1-to-16 DEMUX: Routes 1 input to 1 of 16 outputs.

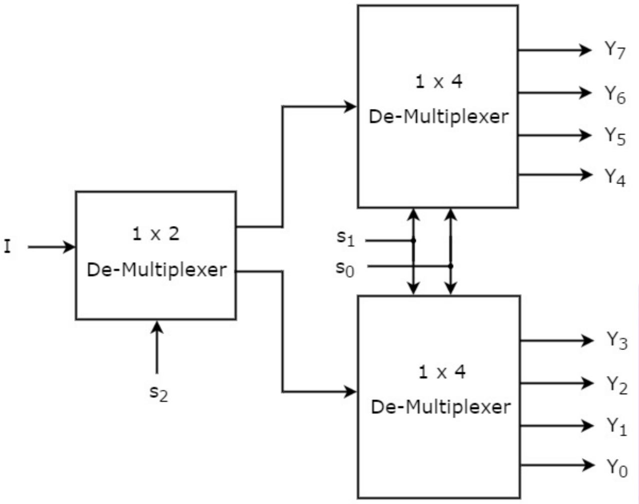
The Truth table of 1x4 De-Multiplexer

The Boolean functions for each output as

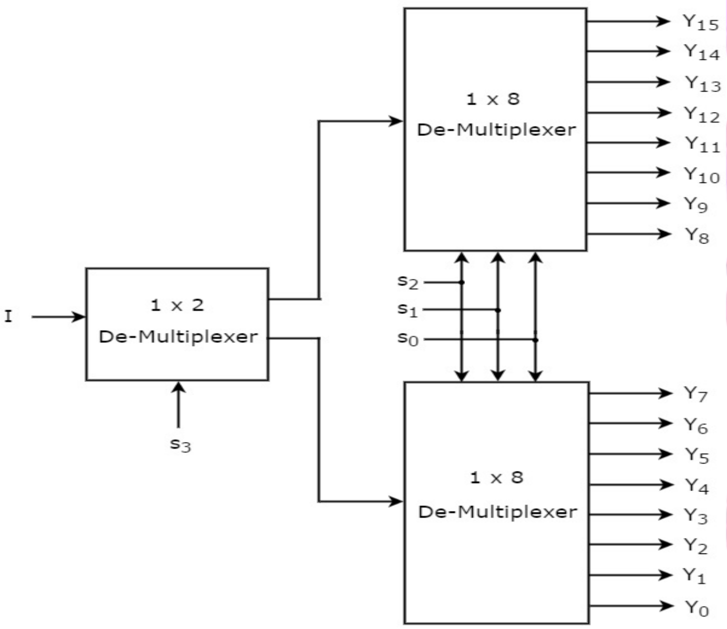
$Y3=S_1S_0I$
$Y2= S_1S_0' I$
$Y1= S_1' S_0 I$
$Y0= S_1' S_0' I$

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

1 X 8 Demultiplexer:



1x16 DeMultiplexer:



Applications of Demultiplexer

- Used in several input and output devices for data routing.
- Used in digital control systems to select one signal from a mutual stream of signals.
- Data transmission in synchronous systems.
- Serial to parallel converters.
- Design automatic test equipment.

Comparison between multiplexer and Demultiplexer

S.No.	Multiplexer	Demultiplexer
1	Selects one of many inputs to pass through to a single output.	Takes a single input and routes it to one of many outputs.
2	Uses control lines to select which input is connected to the output.	Uses control lines to select which output receives the input signal
3	Often used in situations where data needs to be selected or merged from multiple sources.	Used when a single data source needs to be distributed to multiple destinations.

Encoder

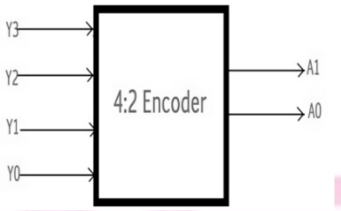
- It converts a set of binary inputs into a unique binary code.
- It has a maximum of 2^n input lines and 'n' output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High.

Types of Encoders

- **Binary Encoder**: Converts a set of 2^n input lines into an n-bit output code.
- **Priority Encoder**: Similar to a binary encoder but assigns priority to inputs if multiple inputs are active simultaneously.
- **Decimal to BCD Encoder**: Converts decimal input (0-9) into binary-coded decimal (BCD).

4 to 2 Encoder

The 4 to 2 Encoder consists of four inputs Y3, Y2, Y1 & Y0, and two outputs A1 & A0. At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output.



The Truth table of 4 to 2 encoders

Logical expression for A1 and A0:

A1 = Y3 + Y2

A0 = Y3 + Y1

INPUTS				OUTPUTS	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Priority Encoder

- Encoder generate some errors:-
  - ✓ There is an ambiguity, when all outputs of the encoder are equal to zero.
  - ✓ If more than one input is active High, then the encoder produces an output, which may not be the correct code.
- So, to overcome these errors, we should assign priorities to each input of the encoder. Then, the output of the encoder will be the code corresponding to the active high inputs, which have higher priority.
- A 4 to 2 priority encoder has 4 inputs: Y3, Y2, Y1 & Y0, and 2 outputs: A1 & A0.
- Here, the input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority.
- In this case, even if more than one input is '1' at the same time, the output will be the (binary) code corresponding to the input, which is having higher priority.

Application of Encoders

1. **Keyboards:** Encoders convert the keypresses into binary codes for the computer to process.
2. **Multiplexing:** Used in multiplexers to select inputs for processing.
3. **Robotics:** For encoding signals in robotic systems.
4. **Digital Signal Processing (DSP):** In DSP systems, encoders are used to convert analog signals into digital codes for further processing, filtering, and analysis.

Decoder

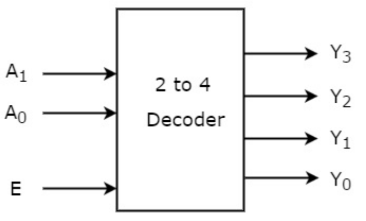
- Decoder is a combinational circuit that has ‘n’ input lines and maximum of 2^n output lines.
- One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. Decoder detects a particular code.
- The outputs of the decoder are nothing but the min terms of ‘n’ input variables lines, when it is enabled.

Types of decoder

- **Binary Decoder:** Converts n-bit binary input into a 2^n output.
- **BCD to 7-Segment Decoder:** Converts BCD input into a format suitable for driving a 7-segment display.
- **Demultiplexer (Demux):** Acts as a decoder with a single input and multiple outputs.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0.



‘One of these four outputs will be ‘1’ for each combination of inputs when enable, E is ‘1’.
The Truth table of 2 to 4 decoder:

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the Boolean functions for each output as:
Y3=E.A1.A0
Y2=E.A1.A0’
Y1=E.A1’.A0
Y0=E.A1’.A0’
Each output is having one product term. So, there are four product terms in total.

Applications of decoders

- **Memory Addressing:** Used in memory chips to select specific memory locations.
- **Display Systems:** BCD to 7-segment decoders are used in digital displays to convert binary input into readable numeric output.
- **Demultiplexing:** Used in communication systems to route signals to multiple destinations.

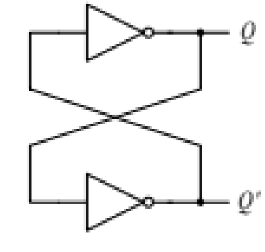
Comparison between Encoder and Decoder

S.No.	Encoder	Decoder
1.	Converts multiple inputs into a smaller number of outputs (compression).	Expands a smaller number of inputs into multiple outputs (decompression)
2.	Encodes data for transmission or storage.	Decodes data for further processing or display.
3.	Often used in input devices, communication systems, and data compression.	Used in output devices, data retrieval systems, and signal routing.

Sequential Circuit

Latches

- It is also known as a bistable-multivibrator. Because it has two stable states namely active high as well as active low.
- The simplest sequential circuit or storage element is a bistable element, which is constructed with two inverters connected sequentially in a loop.
- It works like a storage device by holding the data through a feedback lane.
- Latch is a basic building block of memory which store 1 bit of memory.
- Latch is not a synchronous system.
- A latch is level-triggered (outputs can change as soon as the inputs change).
- A bistable element has memory in the sense that it can remember the content (or state) of the circuit indefinitely.

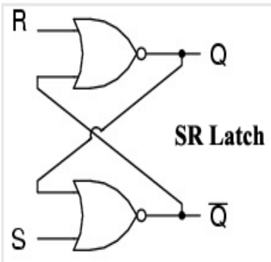


Types of Latches

- S-R Latch
- D latch
- JK Latch
- T Latch

SR Latch

- The bistable element is able to remember or store one bit of information.
- However, because it does not have any inputs, we cannot change the information bit that is stored in it.
- In order to change the information bit, we need to add inputs to the circuit.
- We replace the two inverters with two NOR or NAND gates. This circuit is called a SR latch.
- In addition to the two outputs Q and Q', there are two inputs S and R for set and reset respectively.



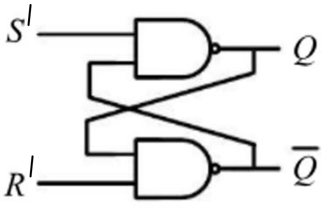
Truth table of SR Latch:-

NOR table		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

SR table			
S	R	Q	Q’
0	0	Hold(memory)	
0	1	0	1
1	0	1	0
1	1	Invalid(not used)	

SR truth table by using NAND gate:

We can also design SR Latch by using NAND Gate but it reverse the value of set and reset therefore we input inverted S (S’) and inverted R (R’) to get result.



SR Latch using NAND Gate

SR table			
S	R	Q	Q’
0	0	Hold(memory)	
0	1	0	1
1	0	1	0
1	1	Invalid(not used)	

Disadvantage of SR latch

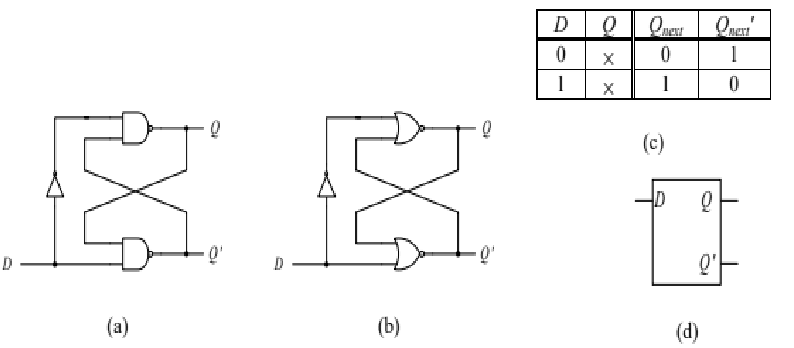
- In the SR latch we need to ensure that the two inputs, S and R, are never de-asserted at the same time.
- We prevent this situation in the D latch by adding an inverter between the original S and R inputs and replacing them with just one input D (for data).

D Latch

D latch ensures that there is no invalid condition in the circuit occurs.

Truth table of D latch:

D Latch		
D	**Q**	**Q’**
0	**0**	**1**
1	**1**	**0**

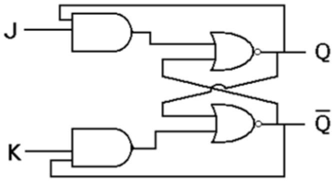


D	Q	Q_{next}	Q_{next}'
0	×	0	1
1	×	1	0

. D latch: (a) circuit using NAND gates; (b) circuit using NOR gates; (c) truth table; (d) logic symbol

JK Latch

- It is similar to SR Latch.
- When the JK inputs are high, the unclear states are eliminated in a JK latch, and the output is toggled.
- The main difference between SR latches and JK latches is that JK latches have the output feedback toward the inputs but SR latches does not have.



Truth table of JK latch

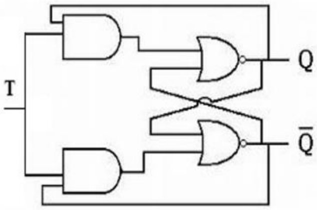
JK Latch			
J	**K**	**Q**	**Q’**
0	**0**	**Hold(memory)**	
0	**1**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**toggle**	

T Latch

The JK latch inputs are shorted to create the T latch. The T latch’s output switches on and off when the input is set to 1 or high.When the input of T is 0 then the output will retain the same (no change).

Truth table of T latch:-

T	**Q**
0	**No change**
1	**Toggle**



Flip Flops

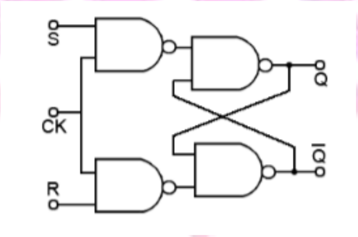
- A flip-flop circuit can store and recall a single bit of information.
- Latches and flip flops are the basic storage elements but different in working.
- As name implies it has ability to “flip” or “flop” between two stable states.
- By latching a value and changing it when triggered by a clock signal, flip-flops can store data over time.
- Flip flop is edge triggered
- Flip-flops change their content only either at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal.
- It is also known as a Bistable Multivibrator**.**

Types of flip-flops

- SR Flip Flop
- JK Flip Flop
- D Flip Flop
- T Flip Flop

SR Flip flop

SR flip-flop operates with positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal.



Truth table of SR flip flop:-

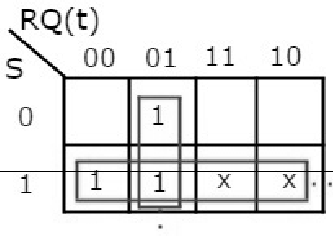
clk	S	R	Q	Q’
0	x	x	Hold(memory)	
1	0	0	Hold(memory)	
1	0	1	0	1
1	1	0	1	0
1	1	1	Invalid(not used)	

Characteristic table and excitation table for SR flip flop:

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Kmap for SR flip flop Q_{n+1}:-

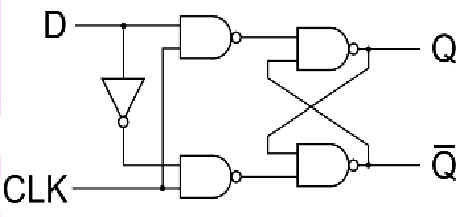


S \ RQ(t)	00	01	11	10
0		1		
1	1	1	x	x

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

clk	D	Q_{n+1}
0	x	0
1	0	0
1	1	1

D flip flop



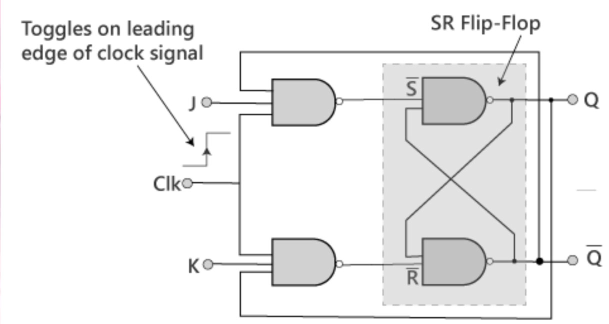
Characteristic table and excitation table for D flip flop:

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

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

$Q_{n+1} = D$

JK Flip Flop



Circuit

JK Flip-Flop Truth Table

J	K	$Q_{(n+1)}$	State
0	0	Q_n	No Change
0	1	0	RESET
1	0	1	SET
1	1	Q_n'	TOGGLE

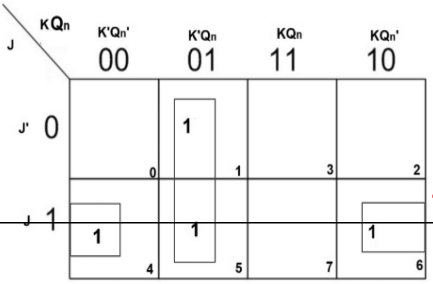
Charecterstic table

J	K	Q_n	$Q_{(n+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Excitation table

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

K-map to derive the characteristic equation.



Q(n+1) = JQn’ + K’Qn

Race Around Condition in JK Flip-Flop

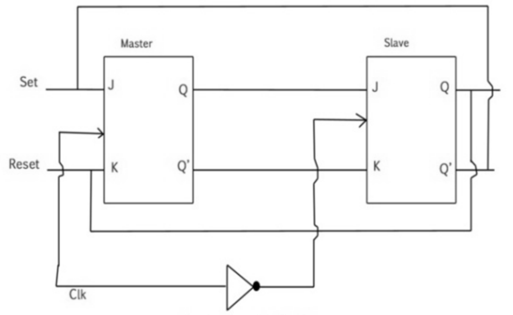
The race condition occurs:-

when level triggered J = K = 1 and clk pulse is high for long time than circuit propagation delay.

- For J-K flip-flop, if J=K=1, and if clk=1 for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain.
- This problem is called race around condition in J-K flip-flop.
- This problem (Race Around Condition) can be avoided by ensuring that the clock input is at logic “1” only for a very short time. This introduced the concept of Master Slave JK flip flop.

Master slave JK flip flop

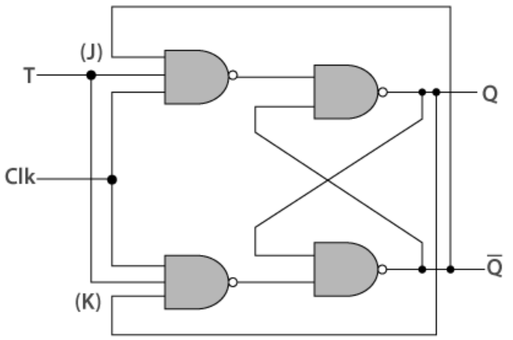
- The Master-Slave Flip-Flop is basically a combination of two JK flip-flops connected together in a series configuration.
- Out of these, one acts as the “master” and the other as a “slave”.
- The output from the master flip flop is connected to the two inputs of the slave flip flop whose output is fed back to inputs of the master flip flop.



Master-slave JK Flip-Flop

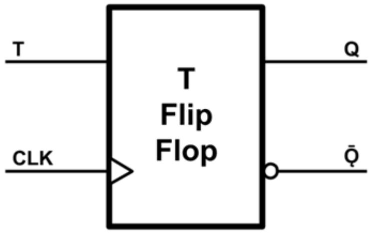
T Flip Flop

- T flip flop or to be precise is known as Toggle Flip Flop because it can able to toggle its output depending upon on the input.
- T here stands for Toggle.
- Toggle basically indicates that the bit will be flipped i.e., either from 1 to 0 or from 0 to 1.
- Here, a clock pulse is supplied to operate this flop, hence it is a clocked flip-flop.

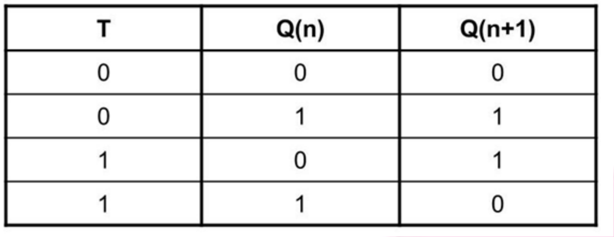


Below you can find the truth table for T Flip-Flop:

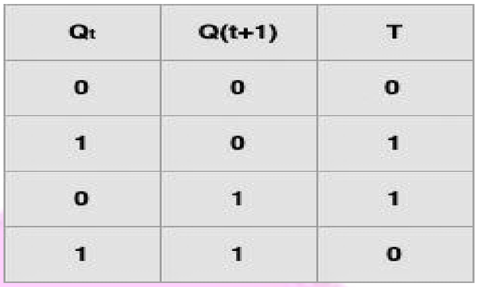
Clk	T	Q(n)	Q’(n)
0	x	0	0
1	0	0	1
0	1	Toggle	



Characteristic table and Excitation table for T Flip Flop



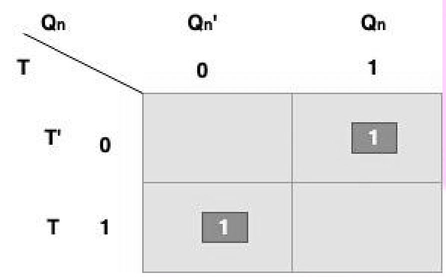
T	Q(n)	Q(n+1)
0	0	0
0	1	1
1	0	1
1	1	0



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

Characterstic equation of T Flip Flop:

Q(n+1) = TQn’ + T’Qn = T ⊕Qn



T \ Qn	Qn' 0	Qn 1
T' 0		1
T 1	1	

We can find T Flip-Flop in many applications, mainly in frequency dividers, binary counters, and parallel load registers.

Counters

- A sequential circuit used to count the pulse is called a counter,
- It is collection of flip flops where the clock signal is applied
- The number of the pulse can be counted using the output of the counter.
- The main properties of a counter are timing , sequencing , and counting.

Counter works in two modes: Bidirectional counters are capable of counting in either the up direction or the down direction through any given count sequence.

- Up counter
- Down counter

Counter modes		
	Q(+ve)	**Q’(-ve)**
positive	down	Up
negative	Up	down

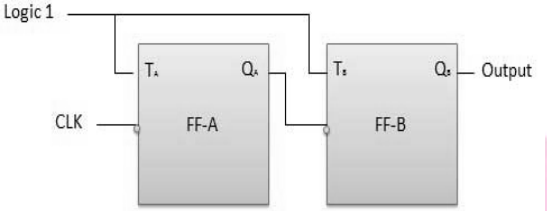
Types of counters

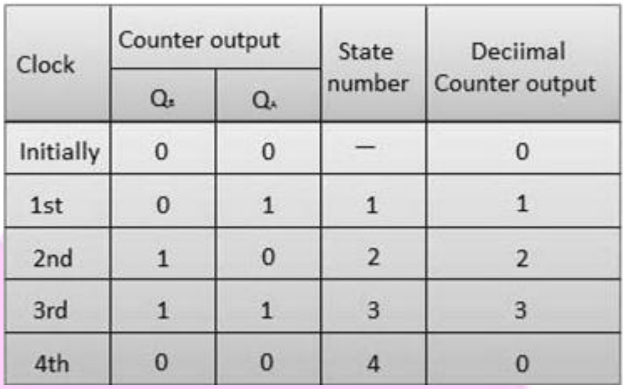
- **Asynchronous Counters or ripple counter**
- **Synchronous Counters**
  1. **Ring counter**
  2. **Johnson counter/twisted ring counter**

Asynchronous counter or ripple counter

- In asynchronous/ripple counter output of the first flip-flop is provided as the clock to the second flip-flop i.e flip-flop(FF) are not clocked simultaneously.
- Circuit is simpler, but speed is slow.

The logic diagram and truth table of a 2-bit ripple up counter





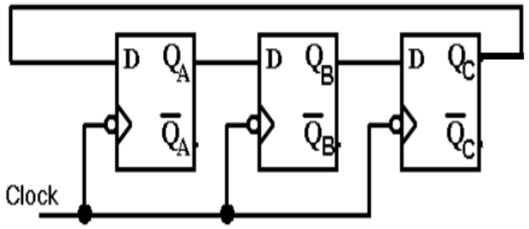
Clock	Counter output		State number	Deciimal Counter output
	Q_B	Q_A		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

Synchronous Counter

- If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

Ring counter

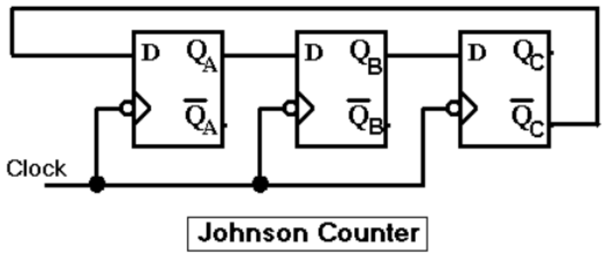
- It is also called One hot Counter.
- In this counter, the output of the last flip-flop is connected to the input of the first flip-flop.
- The main point of this Counter is that it circulates a single one (or zero) bit around the ring.
- No. of states in Ring counter = No. of flip-flop used
- A ring counter is a special type of application of the Serial IN Serial OUT Shift register.



Johnson counter

It is also known as a switch-tail ring counter, walking ring counter, or twisted ring counter.

- It connects the complement of the output of the last shift register to the input of the first register and circulates a stream of ones followed by zeros around the ring.
- In the Twisted Ring Counter, the number of states = 2 X the number of flip-flops.



Difference between synchronous and asynchronous

Key	Synchronous Counter	Asynchronous Counter
Trigger	In case of Synchronous Counters, all the constituent flip-flops are triggered with same clock simultaneously.	In case of Asynchronous Counters, there is triggering of different flip-flops with different clock.
Operation Speed	Operation speed of a synchronous counter is faster as compared to that of an asynchronous counter.	The operation speed of an asynchronous counter is comparatively slower than a synchronous counter.
Error Prone	Synchronous Counters are less error-prone; they hardly produce any decoding errors because each flip-flop is individually clocked.	Asynchronous Counters are more error-prone and produce decoding errors in the system.
Complexity	All the flip-flops in a synchronous counter coordinate with the clock, hence its design and implementation is complex as compared to that of an asynchronous counter.	In an asynchronous counter, the output of one flip-flop acts as the input of the next flip-flop, hence its design and implementation is quite simple.
Sequence	A Synchronous counter can be operated in any desired count sequence, as it could get manipulated by changing the clock sequence.	An Asynchronous counter can operate only in a fixed count sequence, i.e., UP and DOWN.
Delay	There is no propagation delay observed in case of Synchronous Counters.	In case of asynchronous counters, there is a subsequent propagation delay from one flip-flop to another.

Some important counters:

Binary counter: It is a type of counter that counts in binary, That is, it can only count up to a certain maximum number of bits before winding around and begin over from zero.

Ripple counter: A ripple counter is a type of binary counter that uses a series of flip-flops to count each bit in the binary sequence.

decade counter: A decade counter is a type of counter that counts in decimal, meaning it can count up to ten before wrapping around and starting over.

Johnson counter: A Johnson counter is a type of ring counter that uses feedback to create a sequence that can count signals.

digital timer:A digital timer is a device that uses a counter to measure the duration of an event or time interval.

pulse counter: A pulse counter is a device that counts the number of pulses or events that occur over a specific time period.

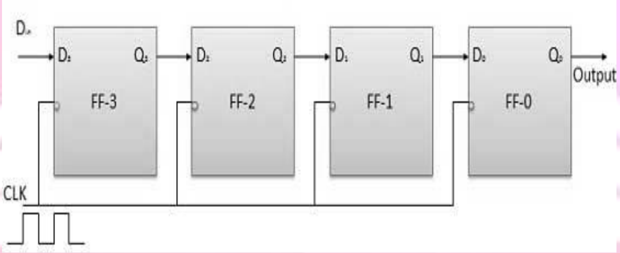
Modulus Counter (MOD-N Counter): The modulus of a counter is the number of states in its count sequence. The maximum possible modulus is determined by the number of flip-flops.

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter.
Where, MOD number = 2^n.

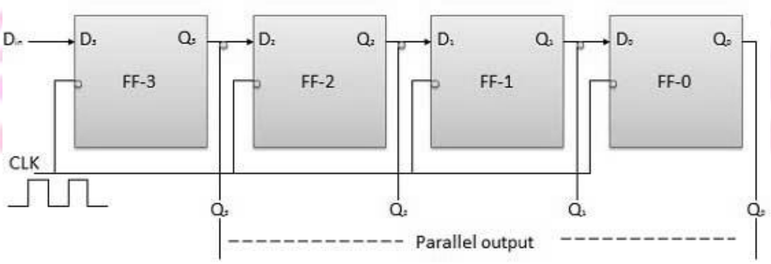
Shift Register

- A Register is a collection of flip flops.
- A flip flop is used to store single bit digital data.
- If we want to store an n-bit word, we have to use an n-bit register containing n number of flip flops.
- There are four mode of operations of a shift register.

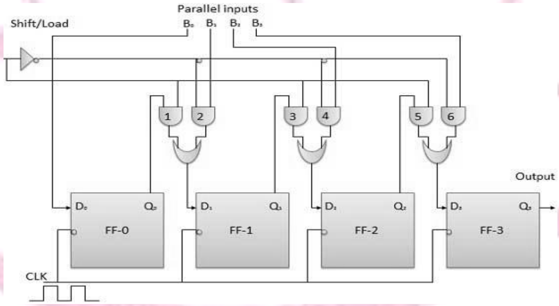
Serial Input Serial Output



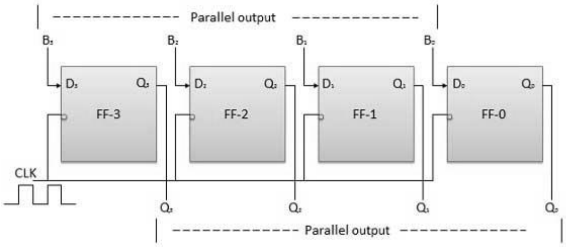
Serial Input Parallel Output



Parallel Input Serial Output



Parallel Input Parallel Output



Bidirectional Shift Register

- For performing the multiplication and division operation using the shift register, it is required that the data should be moved in both the direction, i.e., left or right in the register. Such registers are called the "Bidirectional" shift register.
- The binary number after shifting each bit of the number to the left by one position will be equivalent to the number produced by multiplying the original number by 2.
- In the same way, the binary number after shifting each bit of the number to the right by one position will be equivalent to the number produced by dividing the original number by 2.

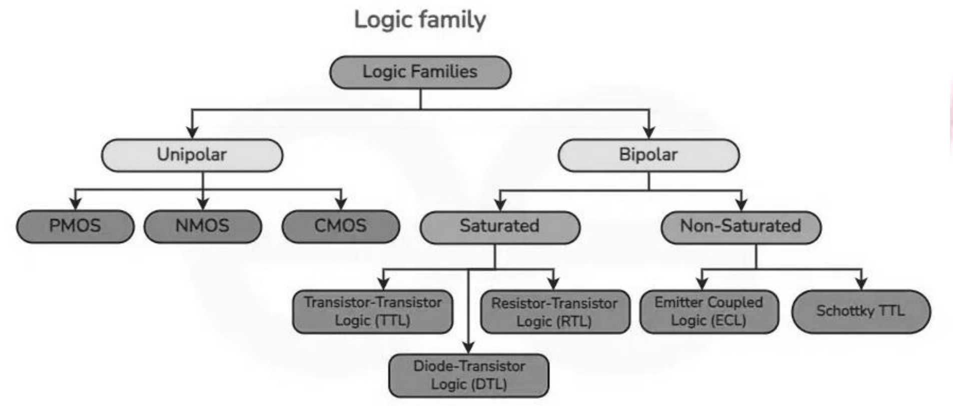
Universal Shift Register

A "Universal" shift register is a special type of register that can load the data in a parallel way and shift that data in both directions, i.e., right and left.

Logic Families

- Collection of integrated circuits (ICs) that share common electrical characteristics, logic levels, and power supply requirements.
- The main purpose of logic families is to perform various digital functions such as AND, OR, NOT, NAND, NOR, etc.
- They determine the speed, power consumption, and compatibility of digital circuits.
- Each logic family has its unique characteristics based on its internal circuitry and manufacturing technology.

Classification of Logic Families


Logic family
Logic Families
Unipolar
Bipolar
PMOS
NMOS
CMOS
Saturated
Non-Saturated
Transistor-Transistor Logic (TTL)
Resistor-Transistor Logic (RTL)
Emitter Coupled Logic (ECL)
Schottky TTL
Diode-Transistor Logic (DTL)

- **Bipolar Logic Families:-** These use Bipolar Junction Transistors (BJTs) as the main switching devices.
  Example: TTL, DTL (Diode-Transistor Logic), RTL (Resistor-Transistor Logic), ECL.
- **Unipolar Logic Families:-** These use Field-Effect Transistors (FETs) as the main switching devices.
  Example: CMOS, NMOS, PMOS.

Key Parameters of Logic Families

- **Propagation Delay:** Time taken by a logic gate to change its state. Lower is better for high-speed circuits.
- **Power Dissipation:** Power consumed by the circuit. Less power dissipation is desirable for energy efficiency.
- **Fan-In:** Number of inputs a gate can handle.
- **Fan-Out:** Number of standard loads that the output of a gate can drive.
- **Noise Margin:** The amount of noise that a circuit can tolerate without error.
- **Voltage Levels:** The logic levels (HIGH and LOW) for which the family is designed.

Difference Between Bipolar and Unipolar

Feature	Bipolar	Unipolar
Definition	Uses Bipolar Junction Transistors (BJTs) where both electrons and holes are charge carriers.	Uses Field-Effect Transistors (FETs), such as MOSFETs, where only one type of charge carrier (either electrons or holes) is involved.
Charge Carriers	Both electrons (negative) and holes (positive) are involved in current conduction	Only one type of charge carrier is involved: electrons (in NMOS) or holes (in PMOS).
Device Types	Examples include BJT-based families like TTL, ECL, DTL, RTL.	Examples include MOSFET-based families like CMOS.
Switching Device	Bipolar Junction Transistor (BJT).	Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET).
Current Flow	Current flows due to both majority and minority carriers (electron-hole recombination).	Current flows due to majority carriers only (either electrons or holes).
Power Consumption	Generally higher power consumption due to continuous current flow through the base of BJTs.	Lower power consumption as MOSFETs do not require continuous current flow (only requires voltage to control gate).
Switching Speed	Moderate to high switching speed.	Can achieve very high switching speeds with lower power dissipation.
Input Impedance	Low input impedance; more power is required to drive the circuit.	High input impedance; requires very little input current, making it more efficient.
Output Drive Capability	High drive capability due to the current-driven nature of BJTs.	Lower output drive capability compared to BJTs, but can be enhanced with advanced CMOS technology.
Noise Immunity	Generally moderate noise immunity.	High noise immunity due to high input impedance and low power consumption.
Applications	Used in older, high-speed applications (e.g., ECL) and general-purpose logic circuits (e.g., TTL).	Widely used in modern digital circuits, microprocessors, and low-power devices due to low power consumption (e.g., CMOS).

Comparison of Bipolar Logic Families

Parameter	RTL (Resistor-Transistor Logic)	DTL (Diode-Transistor Logic)	TTL (Transistor-Transistor Logic)	ECL (Emitter-Coupled Logic)
Basic Components	Resistors and Transistors	Diodes, Resistors, and Transistors	Transistors	Transistors (Differential Amplifier Configuration)
Circuit Complexity	Very Simple	Simple	Moderate	Complex
Speed (Switching Time)	Slow (propagation delay: 30-40 ns)	Faster than RTL but still slow (20-30 ns)	High Speed (10-20 ns)	Very High Speed (0.5-2 ns)
Power Consumption	High due to resistive elements	Lower than RTL	Moderate to High (depending on type)	Very High
Fan-Out	Low (≤ 5)	Moderate (≤ 10)	High (10)	High (≥ 25)
Noise Margin	Low	Moderate	Moderate	Low
Input Impedance	Low	Low to Moderate	Moderate	Low
Output Drive Capability	Low	Moderate	High	High
Voltage Levels	Not standardized	More defined but variable	Standardized (e.g., 0V for LOW, 5V for HIGH)	Negative Logic (e.g., -0.8V for LOW, -1.7V for HIGH)
Immunity to Noise	Poor	Better than RTL	Moderate	Poor due to differential operation
Temperature Stability	Poor	Moderate	Good	Very Good
Advantages	Simple design, easy to understand	Improved speed and noise margin over RTL	Widely used, good balance between speed and power	Extremely fast switching speeds
Disadvantages	Very high power, slow speed	Complex circuit design with diodes	Higher power consumption than CMOS	Very high power consumption, complex design
Typical Applications	Early digital circuits (obsolete now)	Early digital circuits (obsolete now)	General-purpose logic circuits, microcontrollers, microprocessors	High-frequency applications like radar, supercomputers, communication systems

1. **RTL (Resistor-Transistor Logic)**
   - The first type of digital logic family, using resistors and transistors.
   - Very simple and inexpensive to design but has high power consumption and slow switching speeds.
   - Rarely used today due to its inefficiency. Used in very early digital systems.
2. **DTL (Diode-Transistor Logic)**
   - Improved over RTL by using diodes to create logic functions before the transistor stages.
   - Offers better noise margins and faster speeds than RTL, but with increased circuit complexity.
   - Largely replaced by TTL and CMOS technologies. Used in early digital circuits.
3. **TTL (Transistor-Transistor Logic)**
   - Most popular and widely used bipolar logic family. Uses BJTs for both input and output stages.
   - Good balance between speed and power consumption. Available in various sub-families like Standard TTL, Low-Power TTL, High-Speed TTL, etc.
   - General-purpose logic circuits, microprocessors, microcontrollers, and digital systems.
4. **ECL (Emitter-Coupled Logic)**
   - The fastest bipolar logic family. Uses a differential amplifier configuration with transistors.
   - Provides extremely high speed and low propagation delay but consumes a lot of power. Operates with negative power supplies.
   - High-speed computing, radar systems, supercomputers, and high-frequency communication systems.

Important Points

- RTL and DTL are mostly obsolete today due to their slow speed and high power consumption.
- TTL is still popular for many digital applications that need a balance between speed, power, and cost.
- ECL is used in specialized applications where high speed is crucial, despite its high power consumption and complexity.

Unipolar logic families

- It use unipolar devices like MOSFETs (Metal-Oxide-Semiconductor Field-Effect Transistors) as switching elements.
- These families primarily include CMOS (Complementary Metal-Oxide-Semiconductor), NMOS (N-type Metal-Oxide-Semiconductor), and PMOS (P-type Metal-Oxide-Semiconductor).
- They are characterized by their high input impedance, low power consumption, and high noise immunity.

Comparison of Unipolar Logic Families

Parameter	NMOS (N-type MOS)	PMOS (P-type MOS)	CMOS (Complementary MOS)
Basic Components	N-channel MOSFETs	P-channel MOSFETs	Both N-channel and P-channel MOSFETs
Circuit Complexity	Moderate	Moderate	Complex (requires complementary pairs)
Speed (Switching Time)	High speed (faster than PMOS)	Lower speed compared to NMOS	Very high speed (depends on technology)

Power Consumption	Moderate (higher than CMOS, lower than PMOS)	High (high static power dissipation)	Very Low (almost negligible static power)
Fan-Out	High (depends on the technology)	Moderate	Very High (50 or more)
Noise Margin	Moderate to High	Low to Moderate	Very High
Input Impedance	Very High	Very High	Very High
Output Drive Capability	Moderate (better than PMOS)	Low (poor drive capability)	High (due to both N and P MOSFETs)
Voltage Levels	Generally 0V to V_DD (e.g., 0V to 5V)	Generally 0V to V_DD	Wide range (1.8V, 3.3V, 5V, etc.)
Immunity to Noise	Moderate	Low to Moderate	Very High (due to high noise margins)
Temperature Stability	Moderate to Good	Poor (affected by temperature variations)	Excellent (stable across a wide range)
Advantages	Faster than PMOS, easier to fabricate	Simple design, easy to understand	Low power consumption, high speed, high noise immunity
Disadvantages	Consumes more power than CMOS	Very high power consumption, low speed	More complex to design and fabricate
Typical Applications	Used in early microprocessors and digital circuits	Used in early, low-speed circuits (now obsolete)	Modern digital circuits, microprocessors, microcontrollers, battery-operated devices

1. NMOS (N-type Metal-Oxide-Semiconductor)

- Uses N-channel MOSFETs for digital logic. Electrons (negative charge carriers) are used for conduction, providing faster operation than PMOS.
- Higher speed and lower power consumption than PMOS but consumes more power than CMOS. Better drive capability compared to PMOS.
- Early microprocessors and digital circuits, before the rise of CMOS. Rarely used today for new designs.

2. PMOS (P-type Metal-Oxide-Semiconductor)

- Uses P-channel MOSFETs for digital logic. Holes (positive charge carriers) are used for conduction, which makes them slower than NMOS devices.
- Consumes high power due to resistive elements and has a lower switching speed. Poor output drive capability.
- Early digital circuits. Mostly obsolete today due to high power consumption and lower speed.

3. CMOS (Complementary Metal-Oxide-Semiconductor)

- Uses both N-channel and P-channel MOSFETs in a complementary fashion to create logic gates. It is the most widely used unipolar logic family in modern digital electronics.

- Offers a great balance between speed, power consumption, noise immunity, and input/output characteristics. Very low static power consumption (almost zero) because only one transistor (either NMOS or PMOS) is on at any time during steady-state.
- Predominant in all modern digital circuits, including microprocessors, microcontrollers, memory devices, FPGAs, ASICs, battery-operated devices, and portable electronics.

Important Points

- NMOS and PMOS were widely used in early digital designs but have been largely replaced by CMOS due to its superior power efficiency and performance.
- CMOS is the most popular unipolar logic family today, known for its low power consumption, high speed, and high noise immunity, making it ideal for a wide range of applications, from simple logic gates to complex microprocessors.

Programmable Logic Devices (PLDs)

- PLDs are digital devices used to build reconfigurable logic circuits. They allow designers to implement custom logic without designing a circuit from scratch.
- PLDs are integrated circuits that can be programmed to perform specific logic functions. This makes them versatile for designing digital circuits like counters, state machines, combinational logic, etc.
- The key advantage of PLDs is flexibility and reduced time-to-market since they can be programmed and reprogrammed by users to implement any logic function.

Types of Programmable Logic Devices (PLDs)

1. **PLA (Programmable Logic Array)**
2. **PAL (Programmable Array Logic)**

1. Programmable Logic Array (PLA)

- Used to implement combinational logic circuits.
- It consists of two programmable gates:
- The AND gate and the OR Gate.
- Programmable AND Gate: Allows for the creation of any number of required product terms (AND operations).
- Programmable OR Gate: Combines the product terms into sum-of-products expressions (OR operations).

Advantages of PLA

- Highly flexible for implementing any logic circuit because both Gates (AND and OR) are fully programmable,.
- Can create complex combinational circuits with ease.
- Good for implementing circuits with a high number of logic terms.

Disadvantages of PLA

- More expensive and complex due to full programmability.
- Slower speed compared to simpler PLDs due to more programmable connections.
- Higher power consumption compared to simpler alternatives like PALs.

Applications of PLA

- Used in custom digital circuit designs.
- Can be used to implement complex state machines and combinational logic.
- Suitable for small-scale integration where maximum flexibility is needed.

2. Programmable Array Logic (PAL)

- Programmable Array Logic (PAL) is a simpler form of PLD that uses a fixed OR plane and a programmable AND plane.
- It is less flexibile than PLA but is faster and more cost-effective.
- **Programmable AND Gate:** Allows customization of the AND operations to generate the desired product terms.
- **Fixed OR Gate:** The OR gate is fixed, meaning that the connections in the OR gate are predefined by the manufacturer and cannot be changed.

Advantages of PAL

- Simpler and cheaper to manufacture compared to PLA.
- Faster operation due to the fixed OR plane.
- Easier to use for simpler combinational logic circuits.

Disadvantages of PAL

- Less flexible than PLA since the OR plane is fixed.
- Cannot implement highly complex logic functions that require more extensive OR combinations.
- Limited programmability and scalability.

Applications of PAL

- Used in simpler digital logic designs where speed is a priority.
- Commonly used in control logic, state machines, address decoders, and basic combinational circuits.
- Ideal for applications where cost and speed are more critical than flexibility.

Differences between PLA and PAL

Feature	**PLA (Programmable Logic Array)**	**PAL (Programmable Array Logic)**
AND Plane	Programmable	Programmable
OR Plane	Programmable	Fixed
Flexibility	High (fully programmable)	Limited (fixed OR plane)
Cost	Higher (more complex)	Lower (simpler design)
Speed	Slower (more programmable connections)	Faster (fixed OR plane reduces delay)
Applications	Complex combinational logic, state machines	Simple logic circuits, control logic, state machines
Power Consumption	Higher due to more programmable elements	Lower due to fixed elements

Analog-to-Digital Converter (ADC)

- It converts analog to digital signals.
- The quality of conversion is determined by resolution (measured in bits e.g. 8-bit, 10-bit, 12-bit ADC). Hogh resolution means better conversion.
- **Example:** An 8-bit ADC can represent an analog signal with 2^8 = 256 different levels.
- **Sampling Rate:** The number of times the ADC samples the analog signal per second.
- Measured in samples per second (SPS) or Hertz (Hz).
- A higher sampling rate captures more details of the analog signal.
- **Quantization:** The process of mapping a large set of input values to a smaller set.
- The difference between the actual analog value and the quantized digital value is called quantization error.

Types of ADCs

- **Successive Approximation Register (SAR) ADC:** Commonly used in microcontrollers. It uses a binary search algorithm to find the digital equivalent of the analog signal.
- **Flash ADC:** Fastest type, used in applications like digital oscilloscopes. It uses a bank of comparators for rapid conversion.
- **Sigma-Delta ADC:** High resolution and good noise performance, used in audio and precision measurement applications.
- **Dual-Slope ADC:** Used in digital multimeters for its accuracy and noise immunity.

Working Principle of ADC

- **Sampling:** The analog signal is sampled at regular intervals based on the sampling rate.
- **Quantization:** Each sampled value is approximated to the nearest level that can be represented digitally.
- **Encoding:** The quantized values are then converted into a binary code, forming the digital output.

Applications of ADC

- **Audio Processing**
- **Sensors:** Reading analog sensor signals (like temperature, light, and pressure) and converting them into digital form for microcontroller processing.
- **Medical Equipment:** ECG, EEG, and other diagnostic equipment convert analog biological signals into digital signals for analysis.
- **Communication Systems:** Digital communication systems, like mobile phones and modems, use ADCs to convert analog signals to digital for processing.

Digital-to-Analog Converter (DAC)

- A DAC takes a digital input (a binary number) and converts it back into a corresponding analog output (like a voltage or current).
- The resolution of a DAC is measured in bits (e.g., 8-bit, 10-bit, 12-bit DAC). Higher resolution provides a more accurate analog output.
- The speed at which a DAC can convert digital data to an analog signal is called conversion rate. It is measured in samples per second (SPS).

Types of DACs

- **Binary-Weighted DAC:** Uses weighted resistors to generate the analog output. Simple but not very precise for high resolutions.
- **R-2R Ladder DAC:** Uses a repeating ladder network of resistors for digital-to-analog conversion. Widely used due to simplicity and accuracy.
- **Sigma-Delta DAC:** Similar to Sigma-Delta ADCs, used for high-resolution applications such as audio outputs.
- **Current Steering DAC:** Used in high-speed applications like video processing.

Applications of DAC

- **Audio Systems:** Converting digital audio signals (from MP3 players, computers) back into analog signals to drive speakers and headphones.
- **Video Systems:** Used in televisions and monitors to convert digital signals (from HDMI, DisplayPort) into analog signals for display.
- **Signal Generators:** Used in function generators to produce various analog waveforms.
- **Motor Control:** In industrial systems, DACs are used to convert digital control signals into analog signals to control motor speeds.
- **Communication Systems:** In radio transmitters and other communication equipment to convert digital data back to analog form for transmission.

Key Differences Between ADC and DAC

Feature	ADC (Analog-to-Digital Converter)	DAC (Digital-to-Analog Converter)
Function	Converts analog signals to digital signals	Converts digital signals to analog signals
Input	Analog (e.g., voltage, current)	Digital (binary code)
Output	Digital (binary code)	Analog (e.g., voltage, current)
Resolution	Determined by the number of bits (e.g., 8-bit, 12-bit)	Determined by the number of bits (e.g., 8-bit, 12-bit)
Key Applications	Audio recording, sensor data processing, medical equipment	Audio playback, video systems, motor control
Types	SAR, Flash, Sigma-Delta, Dual-Slope	Binary-Weighted, R-2R Ladder, Sigma-Delta, Current Steering
Error Sources	Quantization error, sampling rate	Resolution error, output settling time