Second Year of Computer Engineering
(2019 Course)
210245 : Digital Electronics & Logic Design

Presented By
Mrs. Smita Khairnar

Pimpri Chinchwad College of Engineering, Pune

Digital Electronics & Logic Design :210245
Unit Details

Unit No	Title of Unit	Total Hrs.
Unit I	Minimization Technique	(07 Hours)
Unit II	Combinational Logic Design	(07 Hours)
Unit III	Sequential Logic Design	(07 Hours)
Unit IV	Algorithmic State Machines and Programmable Logic Devices	(07 Hours)
Unit V	Logic Families	(07 Hours)
Unit VI	Introduction to Computer Architecture	(07 Hours)

Unit-I Minimization Technique [7Hrs]

Logic Design Minimization Technique -:

Minimization of Boolean function using K-map(up to 4 variables) and Quine Mc-Clusky Method

Representation of signed number- sign magnitude representation,1's complement and 2's complement form , Sum of product and Product of sum form, Minimization of SOP and POS using K-map.

Unit 1:Minimization Technique [7Hrs]

Sr.No.	Topic Content	Reference Book
1	Minimization of Boolean function using K-map (up to 4 variables) Minimization of Boolean function using K-map (up to 4 variables)	T1:138-163,R2:42-72
2		
3	Quine Mc-Clusky Method	T1:178-185, R3,T3:185-195
4		
5	Representation of signed number- sign magnitude representation ,1's complement and 2's complement form	T1(2),T3 :52-75
6	Sum of product and Product of sum form, Minimization of SOP and POS using K-map.	T1:155-163, T3: 141-184
7		
	Case Study: Digital locks using logic gates	
	Mapping of Course Outcomes:CO1	

T1: Modern Digital Electronics by R.P.Jain, 4th Edition, Tata McGraw Hill
R3:Anil Maini, —Digital Electronics: Principles and Integrated Circuits‖, Wiley India Ltd
T3:G.K. Kharate," Digital Electronics". Oxford Press,ISBN -10:0198061838
E-Material: Video Lecture :NPTEL, Swayam
Certification :https://www.coursera.org/learn/digital-systems

Introduction: Digital System supports only binary data, HIGH is represented as logic 1 and LOW as logic 0 in positive logic; and in negative logic, HIGH is represented as logic 0 and LOW is represented as logic 1

Digital system is defined by Boolean expression or truth table or state diagram

The requirements and inputs of the gates can be reduced by simplifying the Boolean expressions. Following are the three simplification methods

1. Boolean method

2.K-map method

3. Quine-McCluskey method

Prerequisite Course Content : Number system , Basic Gates, Boolean theorems

There are six types of Boolean Laws.

1. **Commutative law**

Any binary operation which satisfies the following expression is referred to as commutative operation.

$$(i)\ A.B = B.A \qquad (ii)\ A + B = B + A$$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

2. **Associative law**

$$(i)\ (A.B).C = A.(B.C) \qquad (ii)\ (A+B)+C = A+(B+C)$$

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

3. **Distributive law**

Distributive law states the following condition.

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

4. **AND law**

These laws use the AND operation. Therefore they are called as **AND** laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

5. **OR law**

These laws use the OR operation. Therefore they are called as **OR** laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

6. **INVERSION law**

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$$\overline{\overline{A}} = A$$

Boolean algebra can be used to analyse a logic circuit and express its operations mathematically.

It plays an important role in digital system design.

Boolean theorems helps us minimize the Boolean equations.

They are also known as single variable Boolean theorem or multivariable theorems depending upon number of variables involved.

DeMorgan's Theorems: These theorems play important role in the simplification of Boolean expressions

First theorem states that the complement of sum of digital signals is equal to the product of its complement.

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

NOR = Bubbled AND

Second theorem states that complement of a product of digital signal is equal to the sum of its complement

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

NAND = Bubbled OR

Minimization of Boolean function using K-Map (Upto 4 Variable)

Limitation of algebraic simplification there is no easy way to tell whether the simplified expression is in its simplest form or it can be simplified further

This problem is overcome in the Karnaugh map (K-map) method and the Quine-McCluskey procedure by providing systematic methods for simplifying switching functions

K-map is a graphical technique to simplify Boolean expressions

The number of cells in a K-map depends upon the number of variables in the Boolean expression .

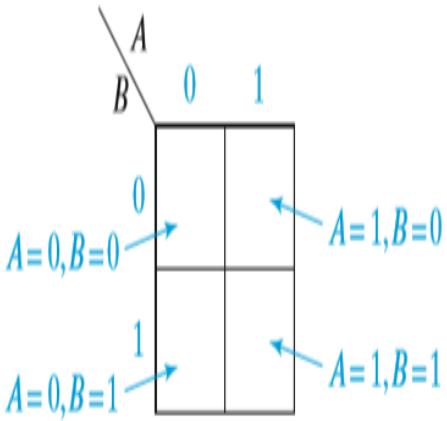
K-map can be used from two variable to six variables (2 variable -4 cells,3 variable -8 cells and 4 variables-16 cells)

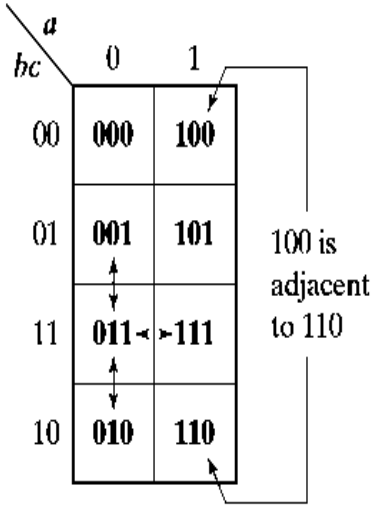
Karnaugh Map

- When a function is realized using AND and OR gates, the cost of realizing the function is directly related to the number of gates and gate inputs used.
- The Karnaugh map techniques developed that lead directly to minimum cost two-level circuits composed of AND and OR gates.
- An expression consisting of a sum of product terms corresponds directly to a two-level circuit composed of a group of AND gates feeding a single OR gate.
- Similarly, a product-of-sums expression corresponds to a two-level circuit composed of OR gates feeding a single AND gate.
- Therefore, to find minimum cost two-level AND-OR gate circuits, we must find minimum expressions in sum-of-products or product-of-sums form.

Representation of Karnaugh Maps

Two, three and four Variable Karnaugh Maps





(a) Binary notation

a / bc	0	1
00	0	4
01	1	5
11	3	7
10	2	6

(b) Decimal notation

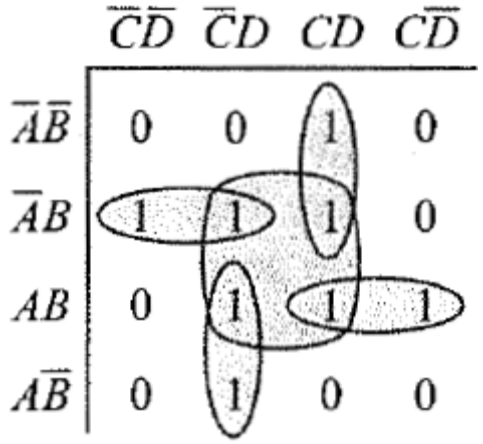
AB / CD	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

Summary of the Karnaugh-map method for simplifying Boolean equations

1. Enter a 1 on the Karnaugh map for each fundamental product that produces a 1 output in the truth table. Enter 0s elsewhere.
2. Encircle the octets, quads, and pairs. Remember to roll and overlap to get the largest groups possible.
3. If any isolated 1s remain, encircle each.
4. Eliminate any **redundant** group.
5. Write the Boolean equation by ORing the products corresponding to the encircled groups.

Eliminating Redundant Groups

- A group whose 1s are already used by other groups.



- All the 1 s of the quad are used by the pairs. Because of this, the quad is redundant and can be eliminated to get

Quine-McCluskey Method

- Simplification of a Boolean equation using K-map method is simple and convenient as long as the variables are up to six.
- As number of variables becomes more then six it becomes difficult to form groups and simplify the Boolean expression .
- W.V.Quine and E.T. McCluskey developed an exact method to simplify the Boolean expression.
- This method is known as Quine-McCluskey method .Also known as tabular method.
- Basic Principle of this method is that the minterms, whose binary equivalent differ only in one place, can be combined to reduce the minterms.(For example, ABC and ABC : can be reduced because only C variable differs)
- The QM method is based on the reduction principle, which says that

  AB+ ABbar = A

Steps used to simplify the Boolean function using Quine-McCluskey Method

1. List all the given minterms in their binary equivalent.
2. Arrange the minterms according to the number of 1's.
3. Compare each binary number with every term in the adjacent next higher category. If they differ by one position ,put a check mark and write in the next column.
4. Repeat step 3 for the resultant column and continue these cycles until no further elimination of variables takes place.
5. List the primary implicants. (Groups of minterms without check marks)
6. Select the minimum number of primes that must cover all the minterms.

Quine-McCluskey Method

Ex. Simplify the logic function using Quine-McCluskey minimization technique.

$Y(A,B,C,D) = \sum m(0,1,2,3,10,11,13,14,15)$

Stage 1		Stage 2	
ABCD		*ABCD*	
0 0 0 0	(0)√	0 0 0 -	(0,1)√
		0 0 - 0	(0,2)√
0 0 0 1	(1)√	0 0 - 1	(1,3)√
0 0 1 0	(2)√	0 0 1 -	(2,3)√
		- 0 1 0	(2,10)√
0 0 1 1	(3)√	- 0 1 1	(3,11)√
1 0 1 0	(10)√	1 0 1 -	(10,11)√
1 1 0 0	(12)√	1 - 1 0	(10,14)√
		1 1 0 -	(12,13)√
1 0 1 1	(11)√	1 1 - 0	(12,14)√
1 1 0 1	(13)√		
1 1 1 0	(14)√	1 - 1 1	(11,15)√
		1 1 - 1	(13,15)√
1 1 1 1	(15)√	1 1 1 -	(14,15)√

Quine-McCluskey Method

- In Stage 3, we combine members of different groups of Stage 2 in a similar way. Now it will have two '–‘ elements in each combination. This means each combination requires two literals to represent it.
- For example (0,1,2,3) is represented by $A'B'$ (0 0 – –).

Quine-McCluskey Method

Stage 1		Stage 2		Stage 3	
ABCD		*ABCD*		*ABCD*	
0 0 0 0	(0)√	0 0 0 -	(0,1)√	0 0 - -	(0,1,2,3)
		0 0 - 0	(0,2)√	0 0 - -	(0,2,1,3)
0 0 0 1	(1)√				
0 0 1 0	(2)√	0 0 - 1	(1,3)√	- 0 1 -	(2,10,3,11)
		0 0 1 -	(2,3)√		
0 0 1 1	(3)√	- 0 1 0	(2,10)√	1 - 1 -	(10,11,14,15)
1 0 1 0	(10)√			1 - 1 -	(10,14,11,15)
1 1 0 0	(12)√	- 0 1 1	(3,11)√	1 1 - -	(12,13,14,15)
		1 0 1 -	(10,11)√	1 1 - -	(12,14,13,15)
		1 - 1 0	(10,14)√		
1 0 1 1	(11)√	1 1 0 -	(12,13)√		
1 1 0 1	(13)√	1 1 - 0	(12,14)√		
1 1 1 0	(14)√				
		1 - 1 1	(11,15)√		
1 1 1 1	(15)√	1 1 - 1	(13,15)√		
		1 1 1 -	(14,15)√		

Quine-McCluskey Method

- If any duplicate terms are formed in each case by combining the same set of minterms in a different order.
- **These duplicate term must be deleting,**
- Now compare terms from the two groups.
- If no further combination is possible, the process terminates.
- There is no Stage 4 for this problem as no two members of Stage 3 has only one digit changing among them. This completes the process of determination of prime implicants.
- **The rule is all the terms that are not ticked (checked off) at any stage is treated as prime implicants.**

Quine-McCluskey Method

- A **prime implicant** of a function F is a product term implicant which is no longer an implicant if any literal is deleted from it.
- . To find a minimum expression we construct a *prime implicant table* in which there is a row for each prime implicant and a column for each minterm that must be covered.
- Then we place check marks **(√) or (X)** to indicate the minterms covered by each prime implicant.

	0	1	2	3	10	11	12	13	14	15
$A'B'$ (0,1,2,3)	√	√	√	√						
$B'C$ (2,3,10,11)			√	√	√	√				
AC (10,11,14,15)					√	√			√	√
AB (12,13,14,15)							√	√	√	√

$$Y = A'B' + B'C + AB \text{ or } Y = A'B' + AC + AB$$

Representation of signed number

1. Number Presentation system, effectiveness, unsigned and signed number operations
2. sign magnitude representation,1's complement and 2's complement form
3. In case of signed binary system, the most significant bit represents the sign of the number. When the MSB is 1,the number is negative ; and when it is 0,the number is positive.
4. There are three methods used to represent sign binary numbers.

a) Sign –magnitude representation

Unsigned 8 bit binary representation (255) ; sign 8 bit binary representation max.positive no (+127) and negative no. (-127)

b) 1's complement representation

The 1's complement of binary a binary is obtained when each bit of a binary number is subtracted from 1.(1's complement of 0 is 1-0 =1 ; 1's complement of 1 is 1-1 =0)

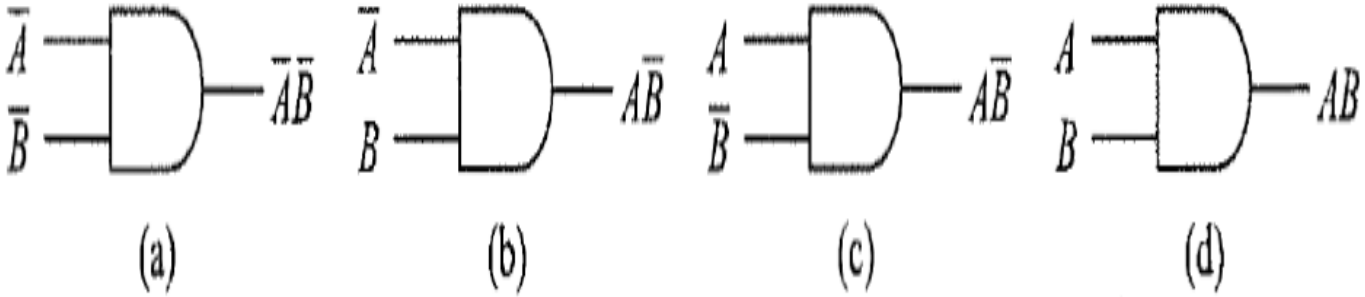
c) 2's complement representation

The 2's complement of a binary number is obtained by adding 1 to 1's complement of a binary number. The 2's complement of a positive number is negative and vice-versa.

Minimization of SOP and POS using K-map.

SUM-Of-PRODUCTS (SOP)

- logic statement, truth-table, SOP form, POS form; Simplification of logical functions using K-Maps up to 4 variables
- Possible ways to AND two or more input signals that are in complement and uncomplement form
- A SOP expression is two or more AND functions ORed together.
- ANDing two variables and their complements



A	B	*Fundamental Product*
0	0	$\bar{A}\bar{B}$
0	1	$\bar{A}B$
1	0	$A\bar{B}$
1	1	AB

SUM-Of-PRODUCTS (SOP)

- The fundamental products are also called **minterms**.
- Products $\overline{A}\overline{B}, \overline{A}B, A\overline{B}, AB$ are represented by m0, m1, m2 and m3 respectively. The suffix i of mi comes from decimal equivalent of binary values that makes corresponding product term high.

SUM-Of-PRODUCTS (SOP)- Example

- The fundamental products by listing each one next to the input condition that results in a high output

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	$1 \rightarrow \bar{A}BC$
1	0	0	0
1	0	1	$1 \rightarrow A\bar{B}C$
1	1	0	$1 \rightarrow AB\bar{C}$
1	1	1	$1 \rightarrow ABC$

- To get the sum-of-products equation, all you have to do is OR the fundamental products

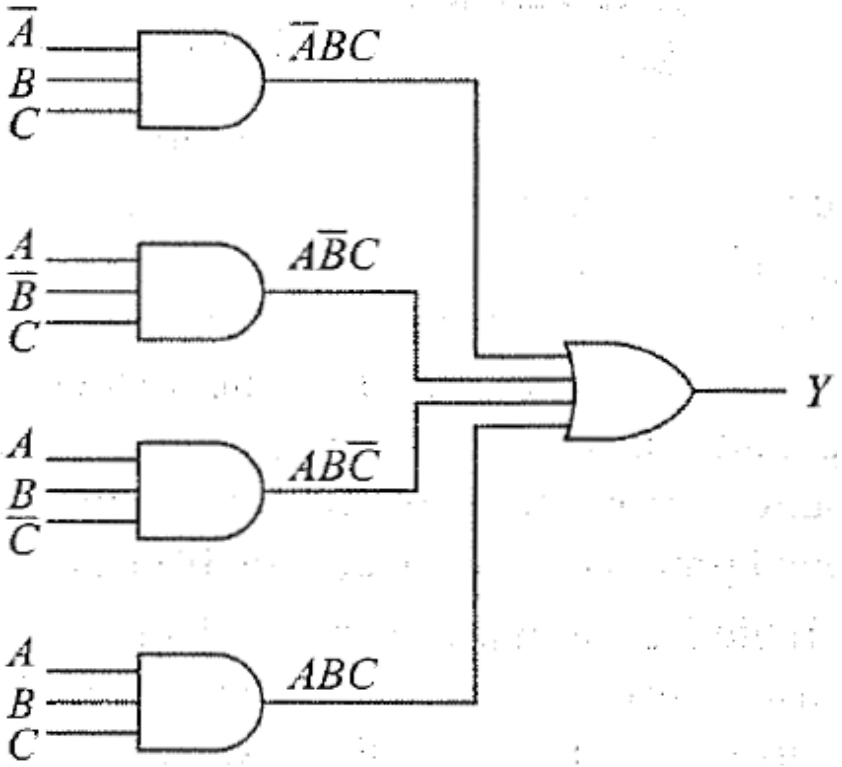
- Alternate representation $Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$$Y = F(A, B, C) = \Sigma m\,(3, 5, 6, 7)$$

SUM-Of-PRODUCTS (SOP)- Example

- where '∑:' symbolizes summation or logical OR operation that is performed on corresponding minterm's and Y = F (A, B, C) means Y is a function of three Boolean variables A, B and C. This kind of representation of a truth table is also known as **canonical sum form**.

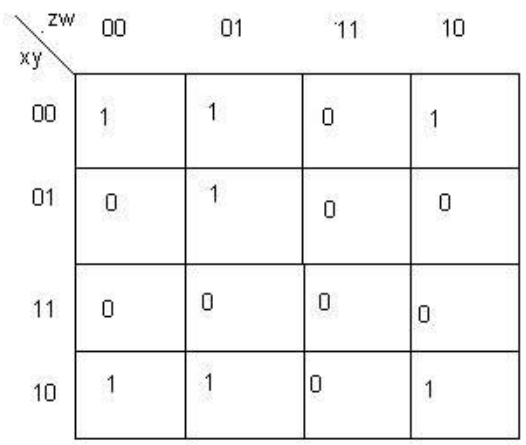
Logic Circuit Diagram

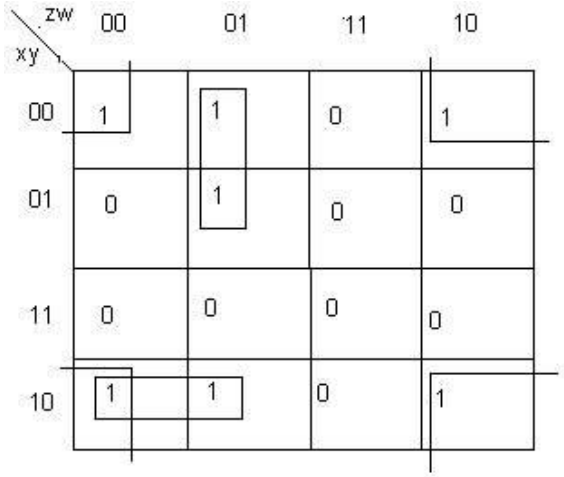


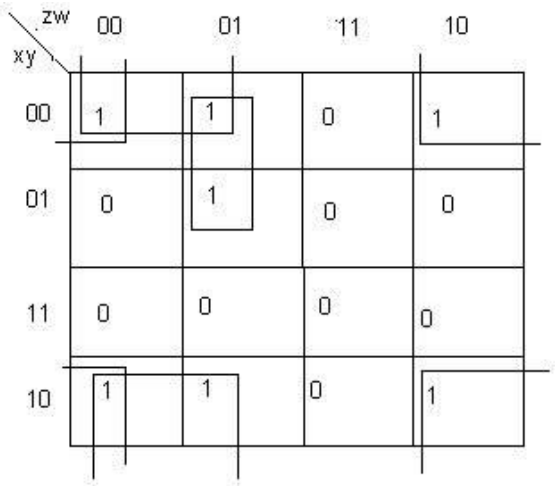
Sum of Product Example

- **Simplify the following Boolean function in sum of products form (SOP)**

F(x, y, z, w) = ∑m(0, 1, 2, 5, 8, 9, 10)







$$F = \bar{y}\,\bar{w} + \bar{y}\bar{z} + \bar{x}\,\bar{z}\,w$$

Don't-care Conditions

- Ideas about don't-care conditions:

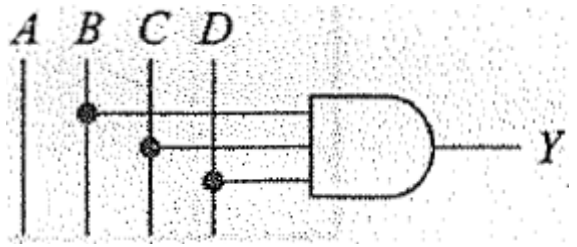
1. Given the truth table, draw a Karnaugh map with 0s, 1s, and don't-cares (X).
2. Encircle the actual 1s on the Karnaugh map in the largest groups you can find by treating the don't cares as 1s.
3. After the actual 1s have been included in groups, disregard the remaining don't cares by visualizing them as 0s.

Don't-care Conditions

Give the simplest logic circuit for following logic equation where *d* represents don't-care condition for following locations.

$F(A, B, C, D) = \sum m(7) + d(10, 11, 12, 13, 14, 15)$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	1	0
AB	×	×	×	×
$A\bar{B}$	0	0	×	×



PRODUCT-Of-SUM (POS)

- Given a truth table, identify the fundamental sums needed for a logic design. Then by ANDing these sums, will get the product-of-sums equation corresponding to the truth table.
- But, in the sum-of-products method, the fundamental product produces an output l for the corresponding input condition.
- But with the product of- sums method, the fundamental sum produces an output 0 for the corresponding input condition.

PRODUCT-Of-SUMS METHOD

- **Converting** a **Truth Table to an Equation**

A	B	C	Y	*Maxterm*
0	0	0	$0 \rightarrow A + B + C$	M_0
0	0	1	1	M_1
0	1	0	1	M_2
0	1	1	$0 \rightarrow A + \overline{B} + \overline{C}$	M_3
1	0	0	1	M_4
1	0	1	1	M_5
1	1	0	$0 \rightarrow \overline{A} + \overline{B} + C$	M_6
1	1	1	1	M_7

$$Y = A + B + C = 0 + 0 + 0 = 0$$

$$Y = A + \overline{B} + \overline{C} = 0 + \overline{1} + \overline{1} = 0 + 0 + 0 = 0$$

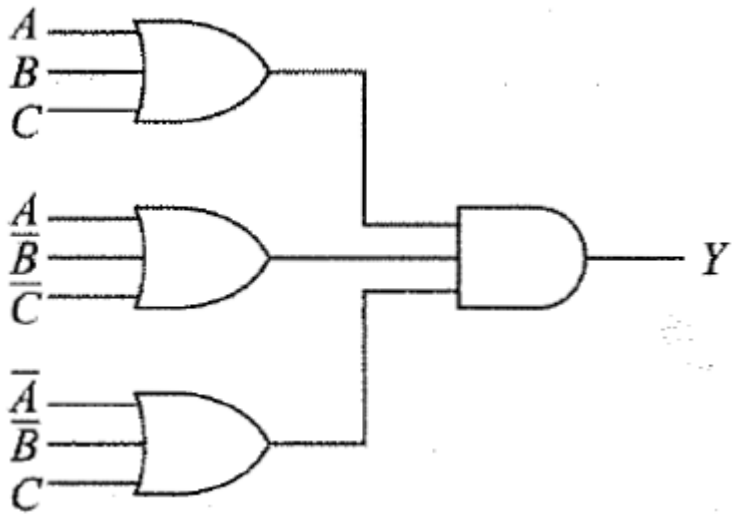
$$Y = \overline{A} + \overline{B} + C = \overline{1} + \overline{1} + 0 = 0 + 0 + 0 = 0$$

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

$$Y = F(A, B, C) = \Pi M(0, 3, 6)$$

PRODUCT-Of-SUMS METHOD

- **Logic Circuit**



ICT Tools

- **NPTEL**
- **ERP**
- **Kahoot:** For creating Quize: https://youtu.be/V4FQ-j91waA
- Quizizz
- Zoom
- Coursera
- Udemy
- Cisco Webex
- Microsoft Teams
- Moodle

Activities in Academic Planning

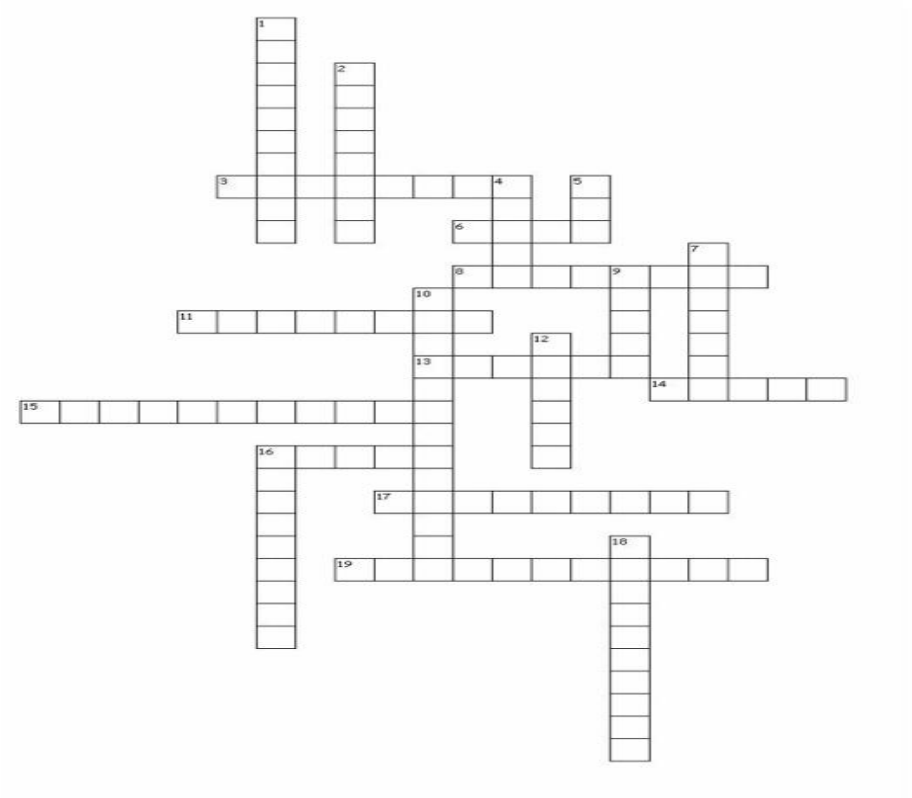
1. Virtual Group Activity
2. Mini Projects
3. Quiz Competition
4. Poster Presentation
5. Group discussion arranged for students
6. Learning through Gaming technique.(snake and ladder)
7. Puzzels (Cross word)
8. Use of Animation for better understanding of concepts
9. YouTube link : https://www.youtube.com/watch?v=sasnEZBBX64 (code convertor)

Cross word Puzzle (Example)

6. I am a type of gate that can make a complete logic set all by myself : Universal Gate

15.The operation that is performed by taking the 2's complement of the second term and then adding them together

Name:



Unit 2: Combinational Logic Design[7Hrs]

Sr.No	Topic Content	Reference Book
1	**Code Converter** : BCD, Excess-3, Gray code, Binary Code.	T1:173-175 ,T3:264-276
2		
3	Half- Adder, Full Adder, Half Subtractor, Full Subtractor,	T1:163-177 ,T3(4) , R3:166-180
4	Binary Adder (IC 7483), BCD adder, Look ahead carry generator	T1:201-209 ,T3: 198-219,R3:180-186
5	**Multiplexers (MUX):** MUX (IC 74153, 74151), Cascading multiplexers, Demultiplexers (DEMUX)- Decoder (IC 74138, IC 74154),	T1:192-200 , T3:241-264,R3:205-229
6	Implementation of SOP and POS using MUX, DMUX, Comparators (2 bit), Parity generators and Checker.	T1:215-216 , T3:229-232,R3:229-241
7		
	Case Study: Combinational Logic Design of BCD to 7-segment display Controller	T3:276-287, T1:228-233
	Mapping of Course Outcomes: CO2	

T1: Modern Digital Electronics by R.P.Jain, 4th Edition, Tata McGraw Hill
T3:G.K. Kharate," Digital Electronics". Oxford Press,ISBN -10:0198061838
R3:Anil Maini, —Digital Electronics: Principles and Integrated Circuits‖, Wiley India Ltd
E-Material: Video Lecture :NPTEL, Swayam
Certification :https://www.coursera.org/learn/digital-systems

Unit-II Combinational Logic Design [7Hrs]

- **Code Converter** -: BCD, Excess-3, Gray code, Binary Code. Half- Adder, Full Adder, Half Subtractor, Full Subtractor, Binary Adder (IC 7483), BCD adder, Look ahead carry generator
- **Multiplexers (MUX):** MUX (IC 74153, 74151), Cascading multiplexers, Demultiplexers (DEMUX)- Decoder (IC 74138, IC 74154), Implementation of SOP and POS using MUX, DMUX, Comparators (2 bit), Parity generators and Checker.

Code Converter

- **Classification of Codes**
- **Properties of Code**
- **Applications of Code**
- **Conversion of Code and its importance**
- **Examples of each below code**

  BCD, Excess-3, Gray code, Binary Code

Code Converter

- Codes are the representation of information in a particular format. The information may include numbers, alphabets and symbols.
- Different codes are used to store and transmit the data efficiently.
- The commonly used binary codes are classified as
  - Weighted codes
  - Self –Complementary codes
  - Unit distance codes
  - Cyclic codes
  - Alphanumeric codes
  - Error detecting and correcting codes.

.

Code Converter

Weighted codes: In weighted code, the weight of a digit or bit depends on its position binary, BCD,

(8-4-2-1) are example of weighted code .

Self –Complementary Codes: Excess-3 code is the example of self complementary code**.**

In this code ,the 1's complement of the excess-3 code is the excess -3 code for the 9's complement of the corresponding decimal number.

For example excess-3 code for 2 is 0101, (2+3=5)

1's complement of 0101 is 1010.It is excess-3 code of 7 and 7 is 9's complement of 2.

Unit Distance Codes: The name itself indicates there is unit distance between two consecutive codes. That is bit patterns for two consecutive numbers differ in only one bit position.

Gray code is an example of unit distance code.

Code Converter

- **Gray code** :It is also called as unit distance code and reflected binary code .The reason for calling this code as reflected binary code is the first N/2 values compared with those of the last N/2 values in reverse order.
- It is not arithmetic code that there are no specific weights assigned to each bit positions.
- The successive code wards differ only one bit this nature of the code is called unit distance code
- Switching activity is reduced because of one digit change in consequence numbers, hence low power consumption and fast response.
- Gray code cannot be used for arithmetic operations
- Example (1000)2 --- (1100) Gray

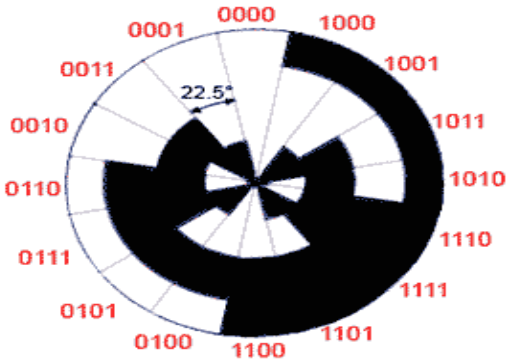


Table 1.6.4

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

Gray Code

Applications:

1. Important feature of Gray code is it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications such as Shaft encoders where error susceptibility increases with number of bit changes between adjacent numbers in sequence.

2. It is sometimes convenient to use the Gray code to represent the digital data converted from the analog data (Outputs of ADC).

3. Gray codes are used in angle-measuring devices in preference to straight forward binary encoding.

4. Gray codes are widely used in K-map

The disadvantage of Gray code is that it is not good for arithmetic operation

Code Conversion

Binary to Gray Code Conversion

- Record the most significant bit as it is.
- EX-OR this bit to the next position bit, record the resultant bit.
- Record successive EX-ORed bits until completed.
- Convert 0011 binary to Gray : Ans: 0010

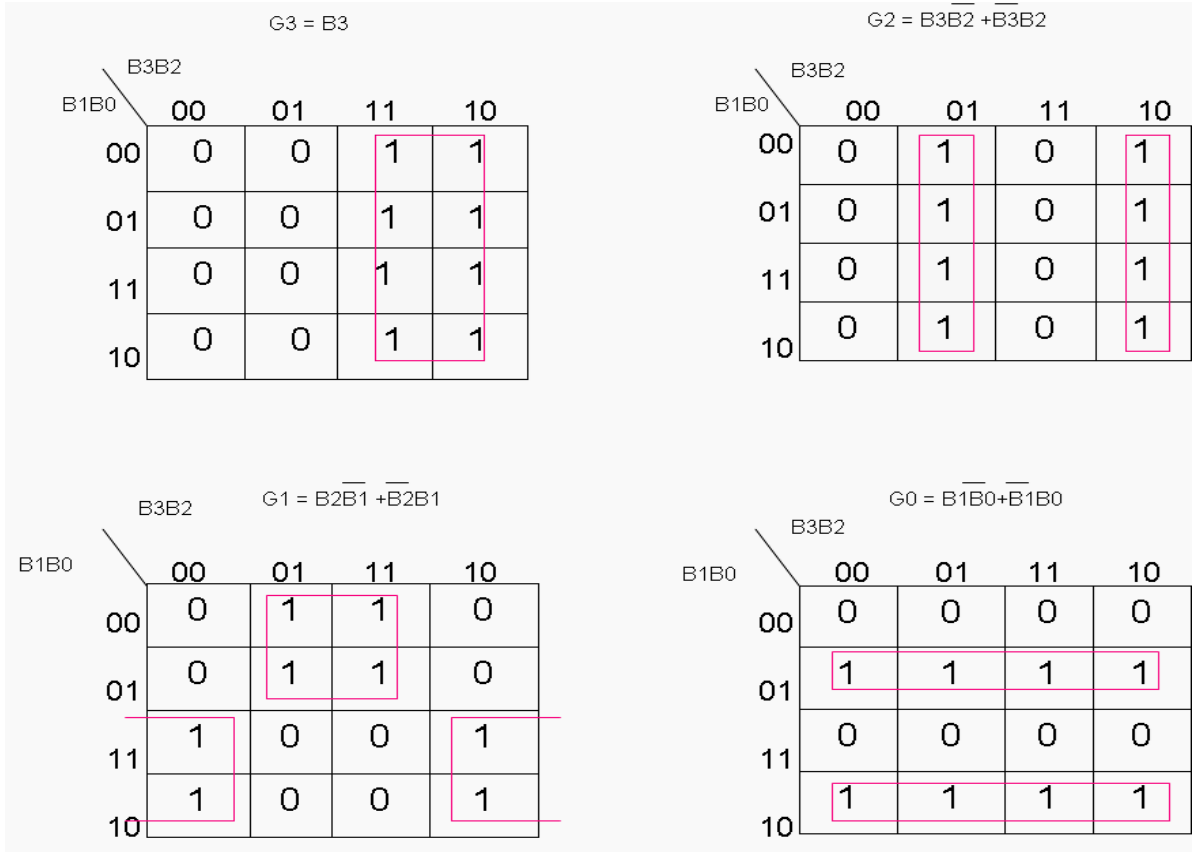
Gray To Binary Conversion

- The Gray code can be converted to binary by a reverse process.
- Record the most significant bit as it is.
- EX-OR binary MSB to the next bit of Gray code and record the resultant bit.
- Continue the process until the LSB is recorded.
- Convert 1011 Gray to Binary code. : Ans : 1101

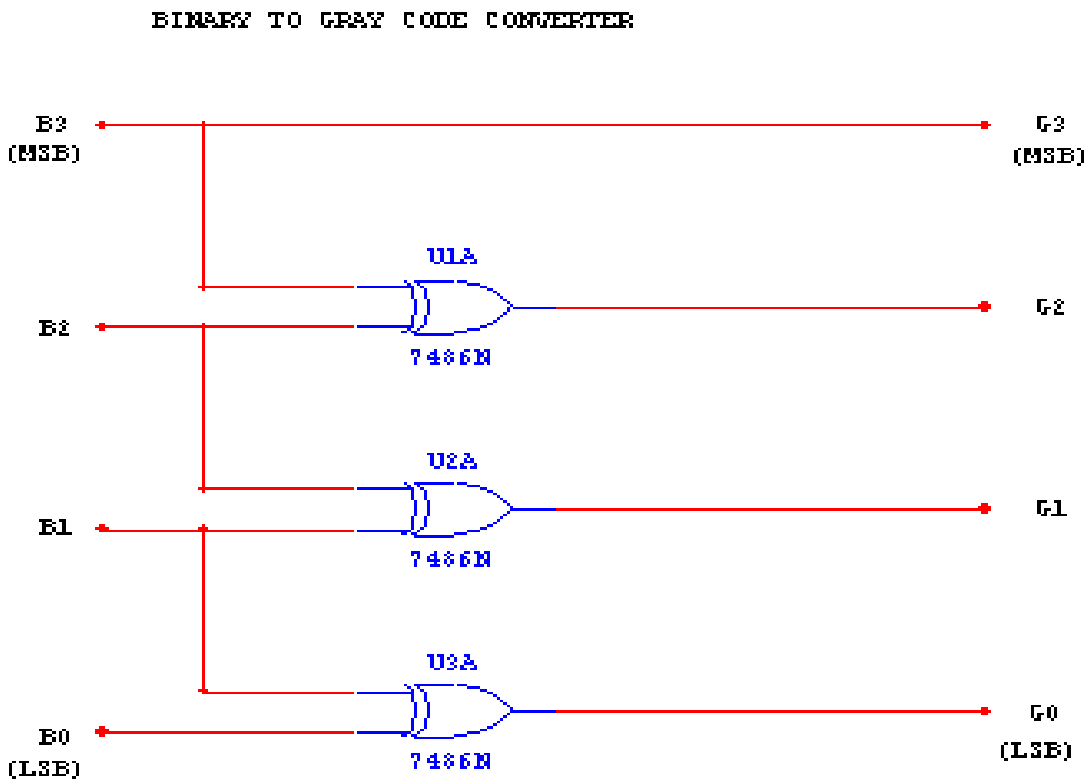
Code Conversion

INPUT (BINARY CODE)				OUTPUT (GRAY CODE)			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Binary to Gray Code Conversion



Code Conversion



Binary to Gray Code Conversion logic Diagram

Binary Coded Decimal :BCD Code

- Binary Coded Decimal (BCD) is used to represent each of decimal digits (0 to 9) with a 4-bit binary code.
- For example $(23)_{10}$ is represented by 0010 0011 using BCD code rather than$(10111)_2$
- This code is also known as 8-4-2-1 code as 8421 indicates the binary weights of four bits(2^3, 2^2, 2^1, 2^0).
- It is easy to convert between BCD code numbers and the familiar decimal numbers. It is the main advantage of this code. With four bits, sixteen numbers (0000 to 1111) can be represented
- In BCD code only 10 of these are used. The six code combinations (1010 to 1111) are not used and are invalid Ex. 247 : 0010 0100 0111

Applications: Some early computers processed BCD numbers. Arithmetic operations can be performed using this code. Input to a digital system may be in natural BCD and output may be 7-segment LEDs.

Code Converter

- **Excess-3** :It is also known as self complementary code .
- As the name implies it is excess of 3 for the regular BCD code i.e. if u add 3(0011) to the BCD Addition u can get Excess-3 code.
- . The excess-3 code has no limitation, so that it considerably simplifies arithmetic operations
- These codes are usually unweighted binary decimal codes

Applications:

The codes 0000 and 1111 can cause a fault in the transmission line. The excess-3 code doesn't use these codes and gives an advantage for memory organization

This code has a vital role in arithmetic operations when sum of two decimal digits exceeds 9, they can not be added using BCD code.

Code Converter

- **BCD To Excess – 3 Code Conversion:**
- Convert BCD 2 i. e. 0010 to Excess – 3 code
- For converting 4 bit BCD code to Excess – 3, add 0011 i. e. decimal 3 to the respective code using rules of binary addition.

0010 + 0011 = 0101 – Excess – 3 code for BCD 2

- **Excess – 3 Code To BCD Conversion:**
- The 4 bit Excess-3 coded digit can be converted into BCD code by subtracting decimal value 3 i.e. 0011 from 4 bit Excess-3 digit.
- e.g. Convert 4-bit Excess-3 value 0101 to equivalent BCD code.

  0101-0011= 0010- BCD for 2

Code Converter

BCD to Excess-3 code converter

INPUT (BCD CODE)				OUTPUT (EXCESS-3 CODE)			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

$E3 = B3 + B2 (B1 + B0)$

B1B0 \ B3B2	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

$E2 = \overline{B2}(B1 + B0) + B2\overline{B1}\overline{B0}$

B1B0 \ B3B2	00	01	11	10
00	0	1	X	0
01	1	0	X	1
11	1	0	X	X
10	1	0	X	X

$E1 = B1B0 + \overline{B1}\overline{B0}$

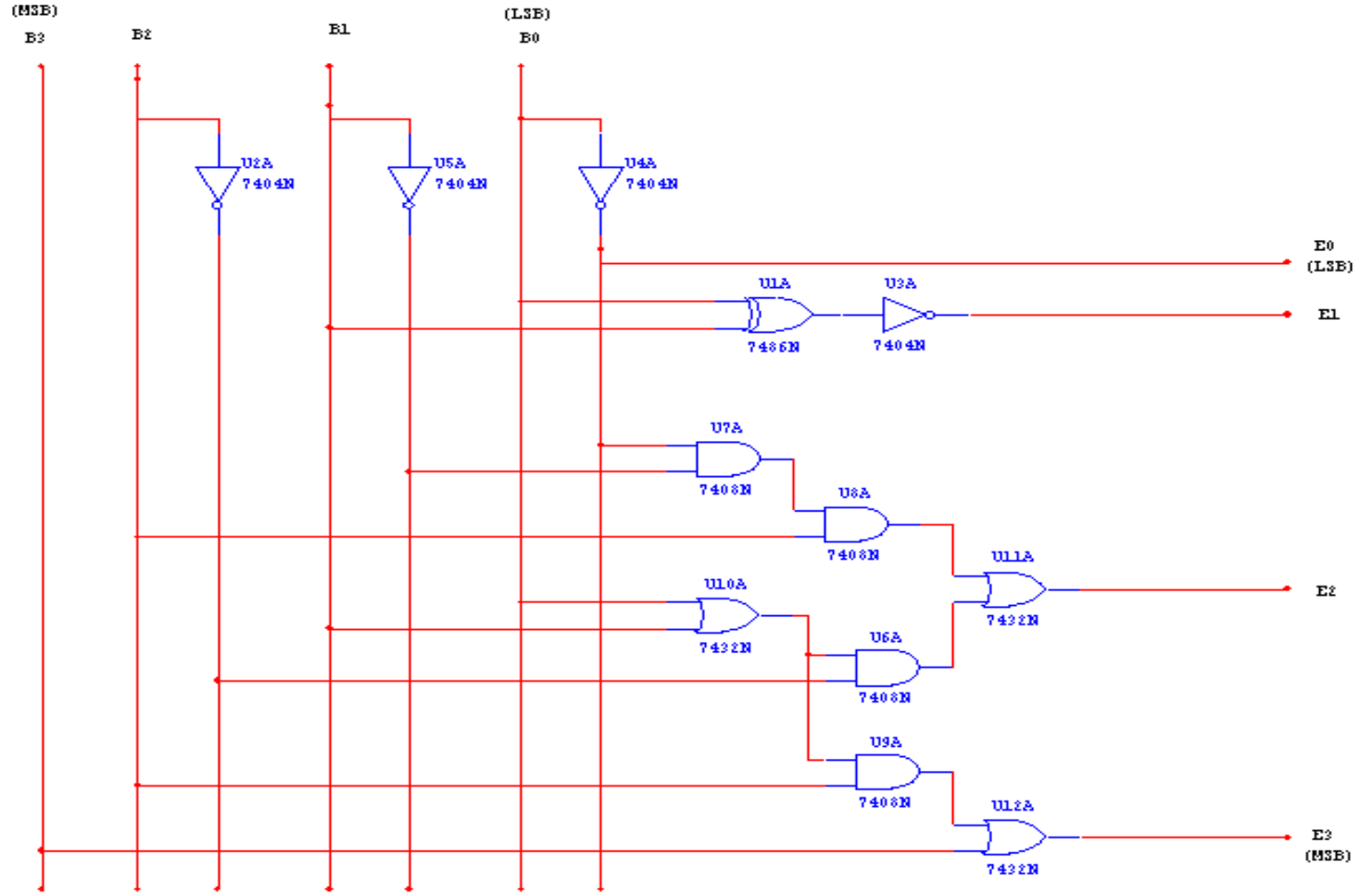
B1B0 \ B3B2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	1	1	X	X
10	0	0	X	X

$E0 = \overline{B0}$

B1B0 \ B3B2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

Code Converter

BCD to Excess-3 code converter

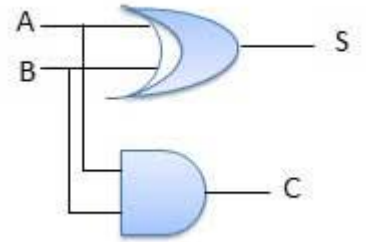


Half Adder

- **Half Adder**

Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

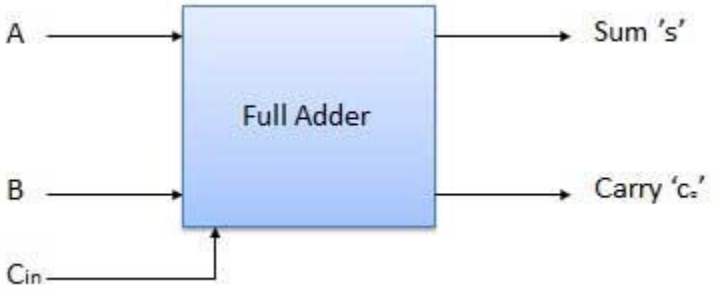
Inputs		Output
A	B	S C
0	0	0 0
0	1	1 0
1	0	1 0
1	1	0 1



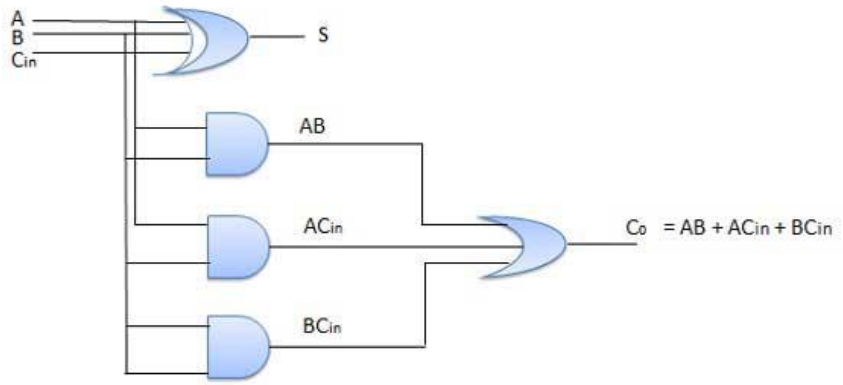
•.

Full Adder

- Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.



Inputs			Output	
A	B	Cin	S	Co
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



Adder and subtractor are basically used for performing arithmetical functions like addition, subtraction, multiplication and division in electronic calculators and digital instruments.

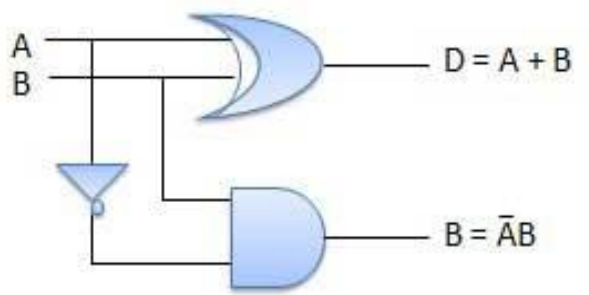
In Timers and Program Counters and Useful in Digital Signal Processing

Half Subtractors

- **Half Subtractors**

Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

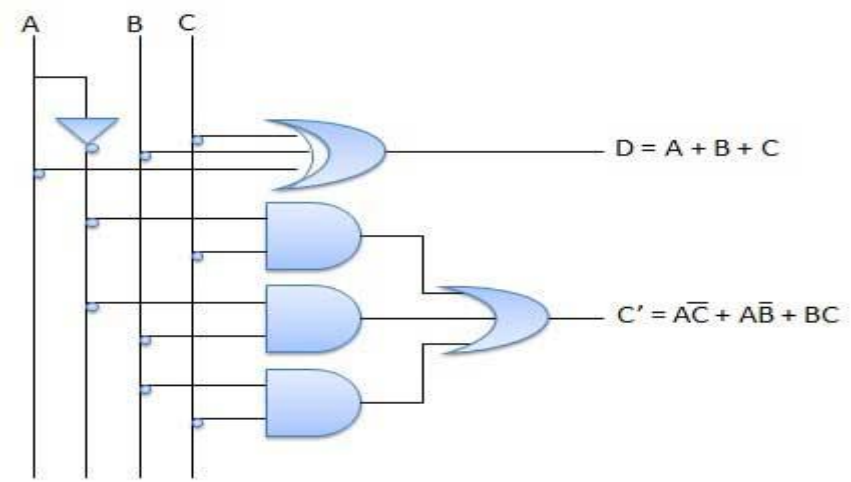


Full Subtractor

- **Full Subtractor**

The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Inputs			Output	
A	B	C	(A-B-C)	C'
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



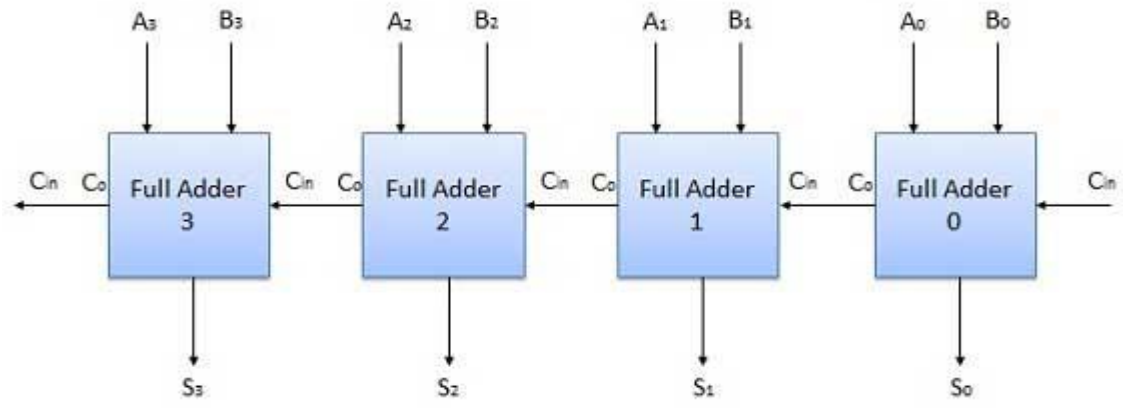
Adder and subtractor are also used in microcontrollers for arithmetic additions

N-Bit Parallel Adder

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

4 Bit Parallel Adder

In the block diagram, A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.



BCD Adder

BCD Adder:

- BCD adder is a circuit that adds two BCD digits & produces a sum of digits also in BCD.

Rules for BCD addition:

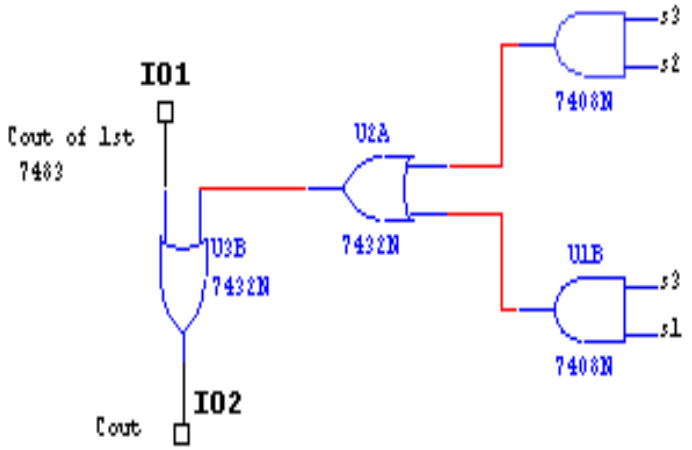
- Add two numbers using rules of Binary addition.
- If the 4 bit sum is greater than 9 or if carry is generated then the sum is invalid. To correct the sum add 0110 i. e. (6)10 to sum. If carry is generated from this addition add it to next higher order BCD digit.
- If the 4 bit sum is less than 9 or equal to 9 then sum is in proper form.

BCD Adder

Truth Table for design of combinational circuit for BCD adder to check invalid

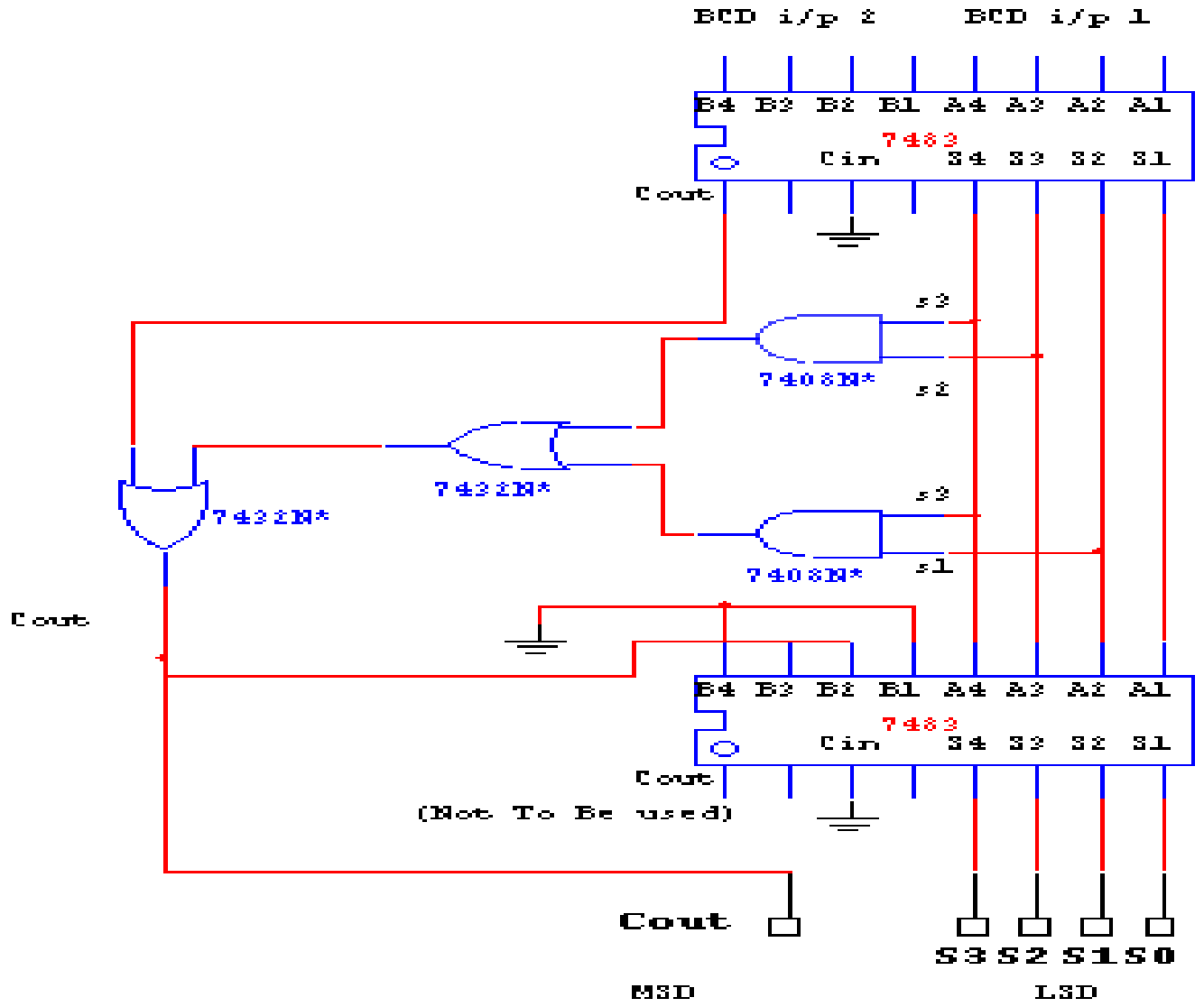
INPUT				OUTPUT
S3	S2	S1	S0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

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



Y = S3S2 + S3S1

BCD Adder Circuit Diagram



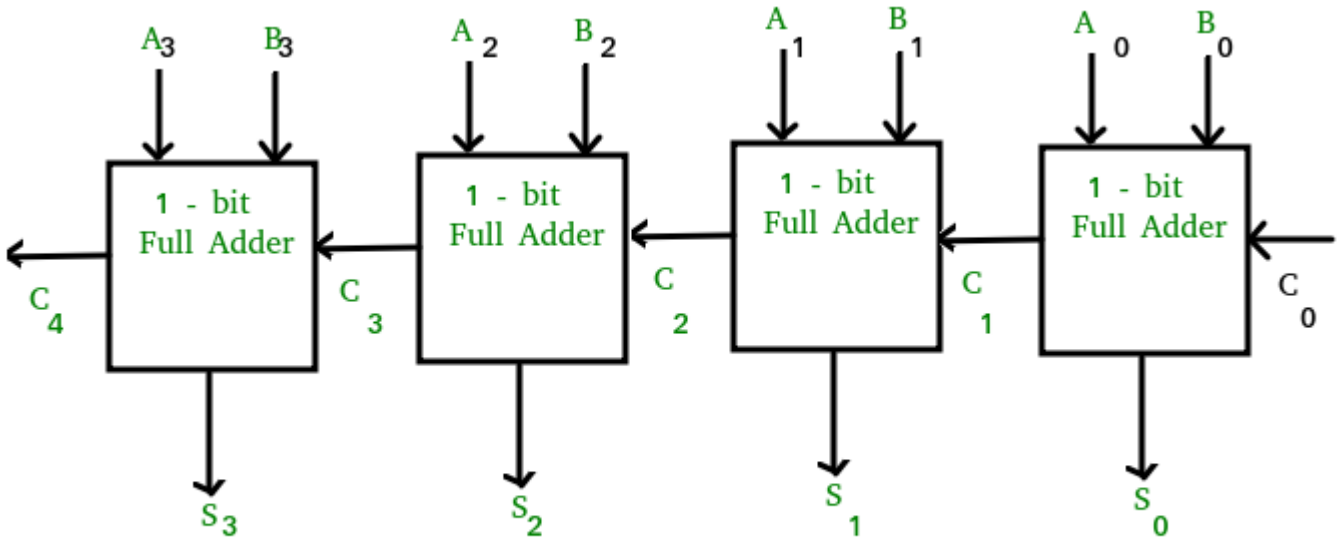
Look ahead carry generator

In adder circuit Sum and carry outputs will be delayed due to propagation delays of the gates involved in the full adder.

Final stage Carry has to ripple through all the stages, therefore reducing the speed of the adder as a number of stages are increased.

Limitation of ripple carry adder are overcome in look ahead carry generator.

It requires additional circuitry but the speed of the adder becomes independent of the number of bits.

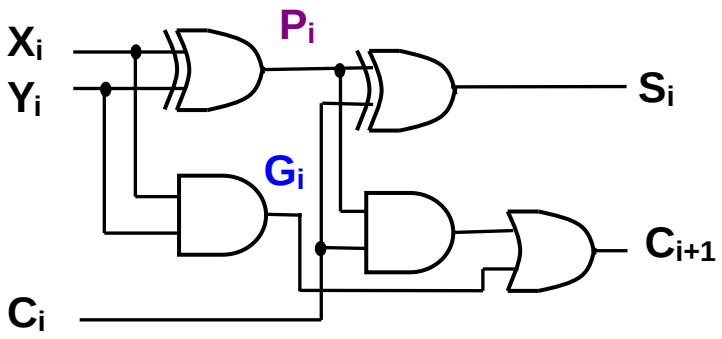


Look ahead carry generator

- Consider the FA, where intermediate signals are labelled as P_i and G_i:

  $P_i = X_i \oplus Y_i$

  $G_i = X_i \cdot Y_i$



- The outputs C_{i+1}, S_i, in terms of P_i, G_i, C_i are:

  $S_i = P_i \oplus C_i$ … (1)

  $C_{i+1} = G_i + P_i \cdot C_i$ … (2)

- Looking at equation (2):

  $G_i = X_i \cdot Y_i$ is a *carry generate* signal, and

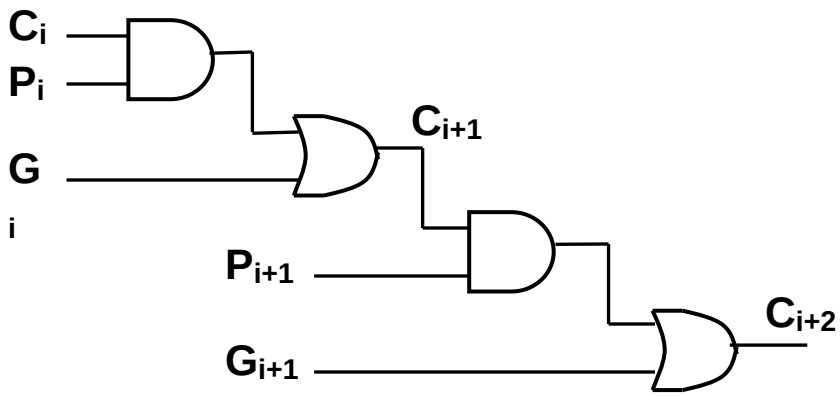
  $P_i = X_i \oplus Y_i$ is a *carry propagate* signal.

Look ahead carry generator

- For 4-bit ripple-carry adder, the equations for the four carry signals are:

$$C_{i+1} = G_i + P_i \cdot C_i$$
$$C_{i+2} = G_{i+1} + P_{i+1} \cdot C_{i+1}$$
$$C_{i+3} = G_{i+2} + P_{i+2} \cdot C_{i+2}$$
$$C_{i+4} = G_{i+3} + P_{i+3} \cdot C_{i+3}$$

These formulae are deeply nested, as shown here for C_{i+2}:



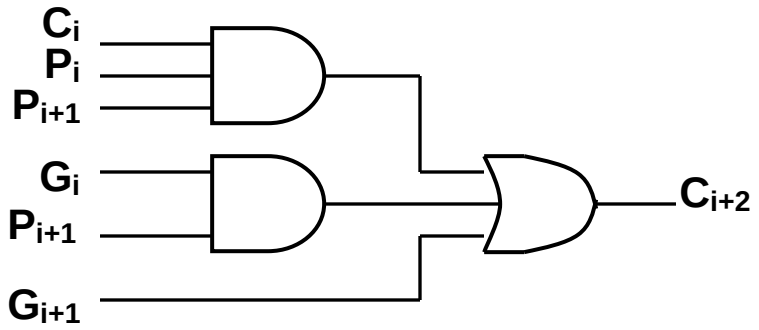
4-level circuit for $C_{i+2} = G_{i+1} + P_{i+1}.C_{i+1}$

Look ahead carry generator

- Nested formulae/gates cause more propagation delay.
- Reduce delay by expanding and flattening the formulae for carries. Example, for C_{i+2}:

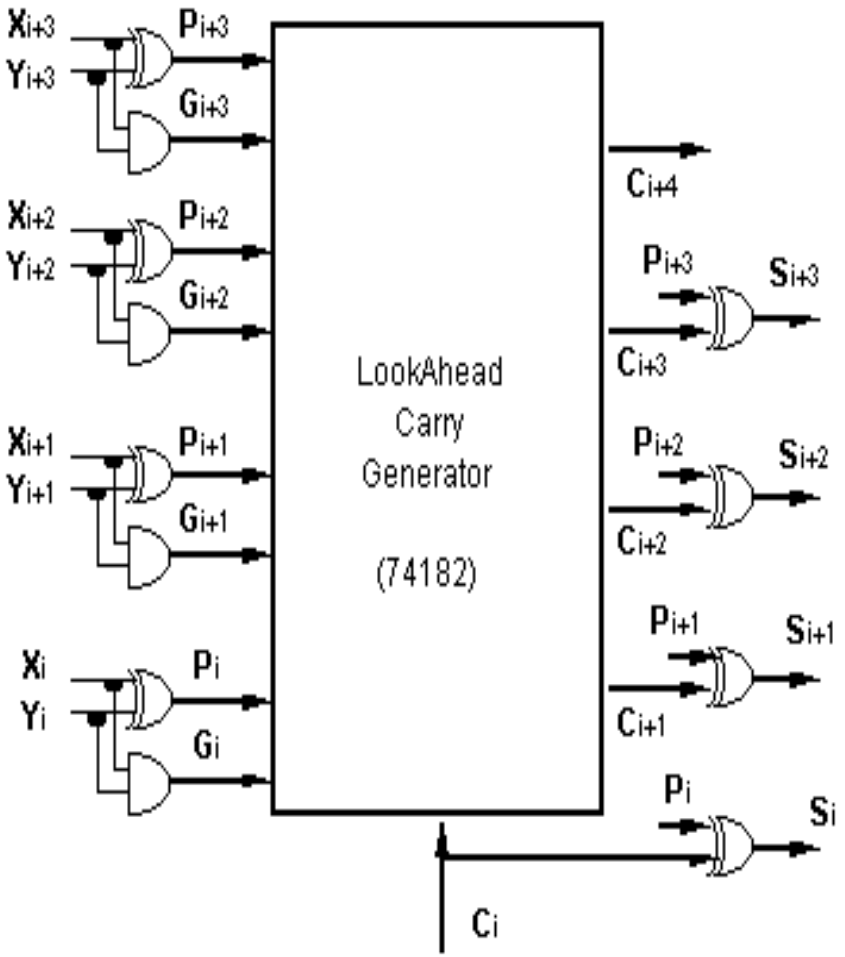
$$C_{i+2} = G_{i+1} + P_{i+1} \cdot C_{i+1}$$
$$= G_{i+1} + P_{i+1} \cdot (G_i + P_i \cdot C_i)$$
$$= G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i \cdot C_i$$

- New faster circuit for C_{i+2}:



Look ahead carry generator

- The 74182 IC chip allows faster lookahead adder to be built.
- Assuming gate delay is t, maximum propagation delay for circuit is hence 4t
  - t to get generate and propagate signals
  - 2t to get the carries
  - t for the sum signals



4-Bit Adder with Look-Ahead (74181)

Multiplexers (MUX):

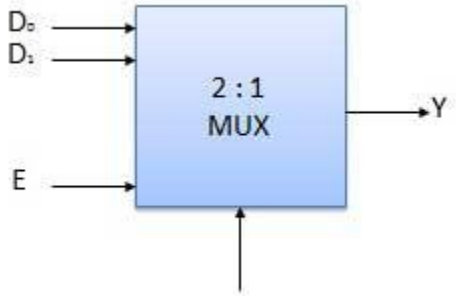
Multiplexer is a combinational logic circuit with multiple data inputs and single output depending on control or select inputs. For N input lines, log n (base2) selection lines, or we can say that for 2^n input lines, n selection lines are required. Multiplexers are also known as **"Data n selector, parallel to serial convertor, many to one circuit, universal logic circuit"**.

Advantages: Simplification of logical expression is not required.

It minimizes the IC count and logic design is simplified

Application : Multiplexers are mainly used to increase amount of the data that can be sent over the network within certain amount of time and bandwidth.

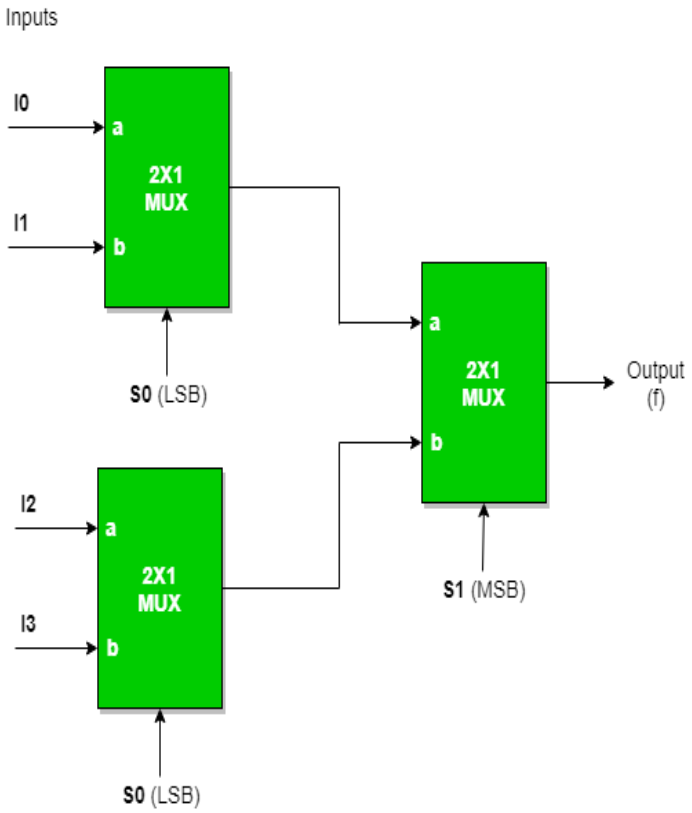
IC No	Description	Output
74153	Dual 4:1	Same as input
74151 A	8:1	Complementary output



Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

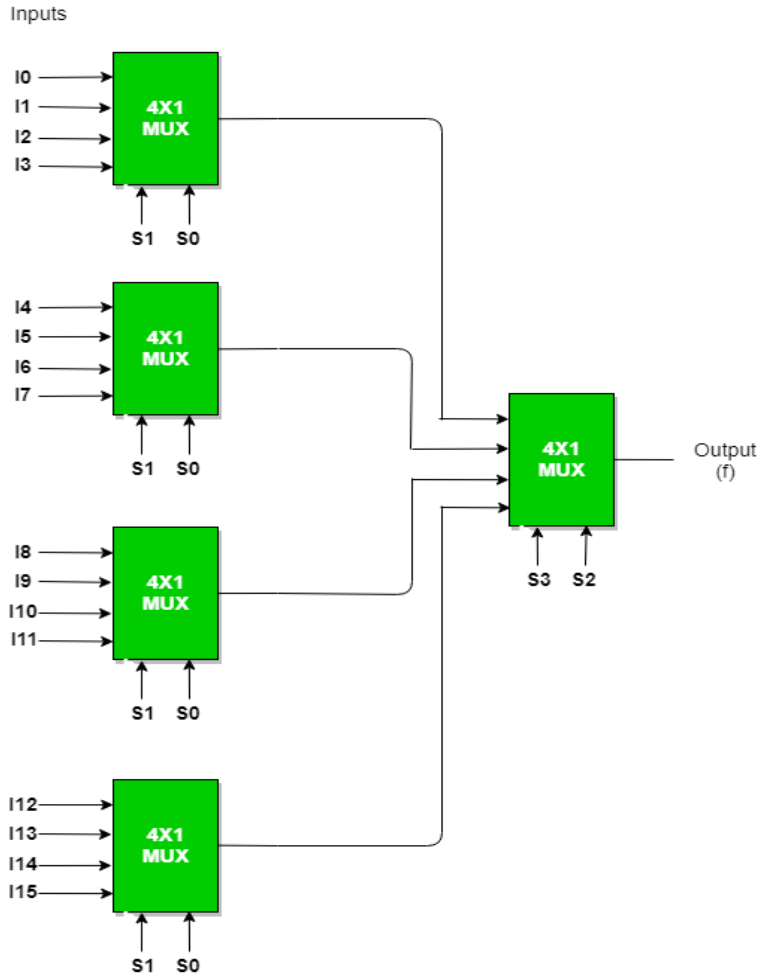
x = Don't care

Implementation of 4:1 Mux using 2:1 and 16:1 using 4:1(Cascading of Mux)



Truth Table

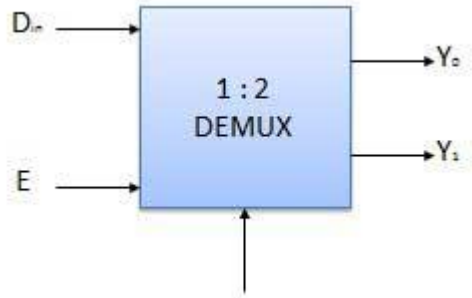
S0	S1	f
0	0	I0
0	1	I1
1	0	I2
1	1	I3



Demultiplexer (DEMUX):

A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in fig.

- Demultiplexers comes in multiple variations.
- 1 : 2 demultiplexer
- 1 : 4 demultiplexer
- 1 : 16 demultiplexer
- 1 : 32 demultiplexer



Enable	Select	Output
E	S	Y0 Y1
0	x	0 0
1	0	0 Din
1	1	Din 0

x = Don't care

IC no	Description	Output
74138	3-line to 8 line decoder (1:8 DEMUX)	Inverted Input
74154	4-line to 16 line decoder (1:16 DEMUX)	Same as Input

Advantages: Simplification of logical expression is not required.

It minimizes the IC count and logic design is simplified

Requires some logic gates in order to realize Boolean expression

Application : Multiple output combinational circuit is designed easily.

Difference between Multiplexer, Demultiplexer & Decoder

Point	Multiplexer	Demultiplexer	Decoder
Input	Many input lines	Single input line	**Many input line also Acts as select line**
Output	Single output line	Many output lines	**Many output line, Active low output**
Select line	**$2^m = n$**	**$n = 2^m$**	**Enable inputs used**

Implementation of Full Adder using Multiplexer

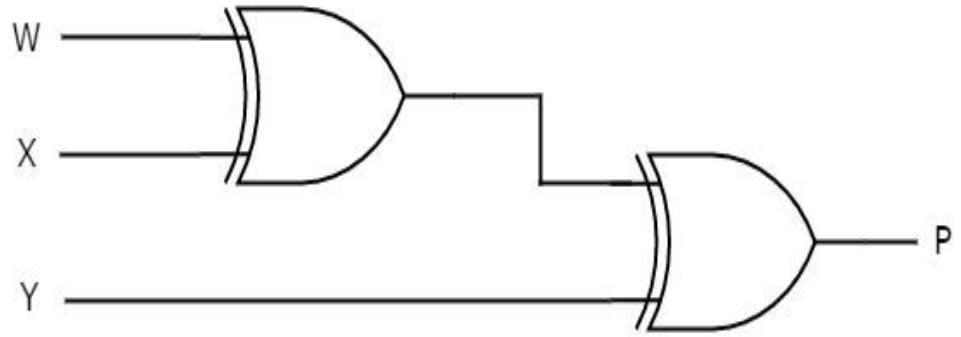
A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of these variables denoted by A and B represent the two significant bits to be added. The third input represents the carry from previous lower significant position

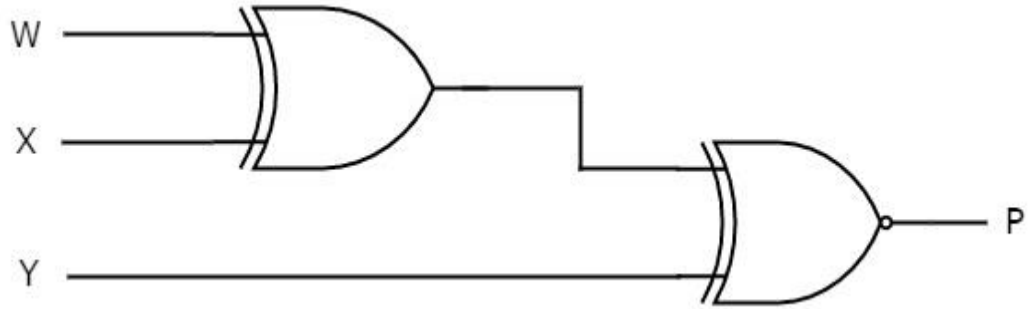
Input			Output	
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum} = \sum m\,(1, 2, 4, 7),$$
$$\text{Carry} = \sum m\,(3, 5, 6, 7)$$

Parity generators and Checker

There are two types of parity bit generators based on the type of parity bit being generated. **Even parity generator** generates an even parity bit. Similarly, **odd parity generator** generates an odd parity bit.





It generates an even parity bit, P. If odd number of ones present in the input, then even parity bit, P should be ‘1’ so that the resultant word contains even number of ones.

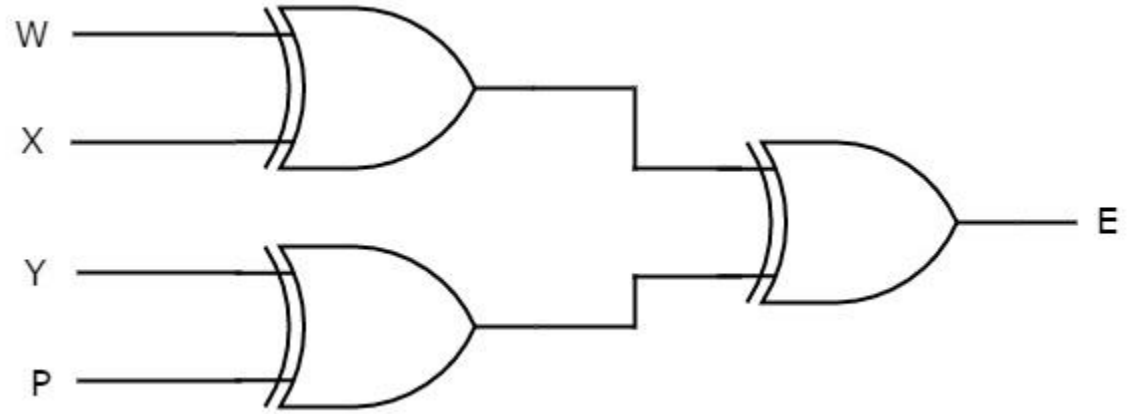
If even number of ones present in the input, then odd parity bit, P should be ‘1’ so that the resultant word contains odd number of ones. For other combinations of input, odd parity bit, P should be ‘0

Binary Input WXY	Even Parity bit P
000	0
001	1
010	1

Application: In digital Communication as error detecting code.

Parity Checker

There are two types of parity checkers based on the type of parity has to be checked. **Even parity checker** checks error in the transmitted data, which contains message bits along with even parity. Similarly, **odd parity checker** checks error in the transmitted data, which contains message bits along with odd parity.



It generates even parity check bit E:This bit will be zero if received data contains even number of ones .This means there is no error in the received data.

If received data contains odd number of ones .This means there is error in the received data.

4-bit Received Data WXYP	Even Parity Check bit E
0000	0
0001	1
0010	1

Comparator (2 –Bit)

A magnitude digital Comparator is a combinational circuit that **compares two digital or binary numbers** in order to find out whether one binary number is equal, less than or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and other for B and have three output terminals, one for A > B condition, one for A = B condition and one for A < B condition.

1-Bit Magnitude Comparator

- A comparator used to compare two bits is called a single bit comparator. It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers. Similarly two bit comparator can be designed (A1,B1: A2,B2) as input and three outputs

A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

References

Text Books:

T1: Modern Digital Electronics by R.P.Jain, 4th Edition, Tata McGraw Hill

T2: Moris Mano , “Digital Logic and Computer Design “, Pearson,ISBN 978-93-325-4252-5

T3:G.K. Kharate,” Digital Electronics". Oxford Press,ISBN -10:0198061838

Reference Books:

R1: John Yarbrough,” Digital Logic applications and Design”, Cengage Learning ,ISBN -12:978-81-315-0058-3

R2:D.Leach,Malvino,Saha,” Digital Principal and Applications “Tata McGraw Hill
R3:Anil Maini, —Digital Electronics: Principles and Integrated Circuits‖, Wiley India Ltd
R4:Norman B. and Bradely,” Digital Logic Design Principles” Willey

- **MOOC Courses:**
- Digital Circuits, by Prof. Santanu Chattopadhyay , https://swayam.gov.in/nd1_noc19_ee51/preview (Unit I, II, III, IV)
- Digital Circuits and Systems , Prof. S. Srinivasan https://nptel.ac.in/courses/117/106/117106086/ (Unit I, II, III, IV)
- Microprocessors and Interfacing By Prof. Shaik Rafi Ahamed | IIT Guwahati
- https://swayam.gov.in/nd1_noc20_ee11/preview (Unit VI)


PCET's Pimpri Chinchwad College of Engineering
PCCOE
"Knowledge Brings Freedom"
Progress Credibility Confidence
Optimism Excellence
Since 1999

Thank You