Ninevah University

College of Electronics Communication Dept.

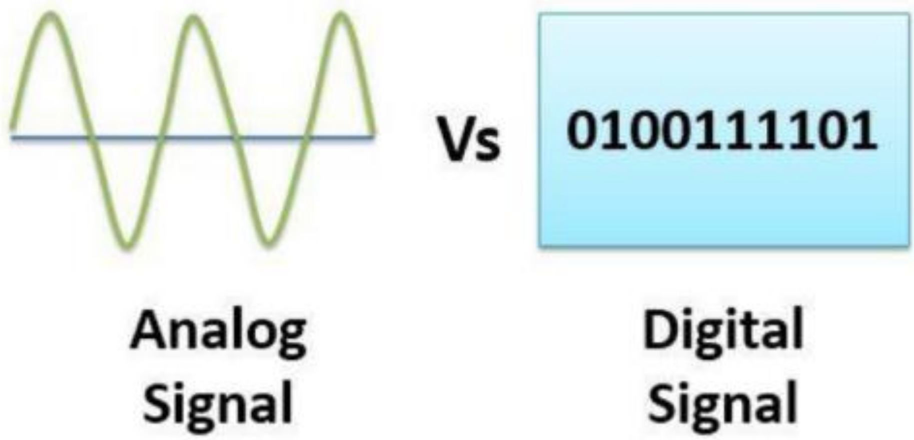
Digital Technique

Digital Technique is a subject that helps you to understand binary numbers and give the students good information about the most famous digital circuits that were used in every digital system

2021-2022

Digital Technique

Analog and digital signals : They used to transmit information, usually through electric signals. In both these technologies, the information, such as any audio or video, is transformed into electric signals. The difference between analog and digital technologies is that in analog technology, information is translated into electric pulses of varying amplitude. In digital technology, translation of information is into binary format (zero or one) where each bit is representative of two distinct amplitudes.



The analog signal is a varied its amplitude with the time such as 0 volt to 40 volt linearly, while the digital signal has two values of amplitude 0 volt and 5 volt.

As this subject is concerned with digital so we need first of all to know what the binary number system is.

How could you understand how binary number system works?
Let us start with decimal which consists of 10 symbols 0, 1, 2…..9
So when we need to write number ten there is no symbol for it. We put the zero symbol in the first digit and put the smallest symbol that is (1) in the second digit like this

1 0

After that we will increase the zero to 1, 2, up to 19 after that again we will put zero in this digit but increase the second digit up to 1 and so on…

Now look at the table 1

Decimal النظام العشري	Binaryالنظام الثنائي
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001

In binary number system we have only two digit 0 and 1

Let us count as before

Zero yes we have this symbol 0

One yes we have this symbol 1

Two the symbol are finished! What we will do as I do with decimal number I will put a zero symbol in the first digit and put 1 in the second digit

Two yes we will use two symbols 10

Three increase the 1st digit by one 11

$$\begin{array}{r} {}^{1\,1}\\ 1\;1 \\ +\quad 1 \\ \hline 1\;0\;0 \end{array}$$

Four 100

As you can see 1 + 1 equal to 2 and two in binary (10) zero will be in the first digit and one is added to the one of the second digit. The same process is again happen so

Four 100

Five 101

Six 110

And so on up to millions.]

Some of you will ask me what is this number 100 is it one hundred or four? What is the answer?

Yes you will confuse between the two systems. Accordingly we will use the following:

100_{10} The sub number $_{10}$ mean this number is decimal, it is one hundred.
100_2 The sub number $_2$ mean this number is binary, it is four in binary.

10^3	10^2	10^1	10^0	digit المرتبة	power الاس
1000	100	10	1	digit values قيمة المرتبة	

	2^4	2^3	2^2	2^1	2^0
16	8	4	2	1	

735_{10} 10101_2

$7*100+3*10+5*1=735_{10}$

$1*16+0*8+1*4+0*2+1*1=21_{10}$

Example1: Convert 9_{10} to binary

المتبقي من القسمة على ٢

/2	Result	Remain
9	4	1
4	2	0
2	1	0
1	0	1

1001

Example 2: Convert 21_{10} to binary

المتبقي من القسمة على ٢

/2	Result	Remain
21	10	1
10	5	0
5	2	1
2	1	0
1	0	1

10101 $=21_{10}$

10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	digit المرتبة power الاس
100	10	1	0.1	0. 01	0.001	

2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
4	2	1	½	¼	1/8	1/16

Example 3: 21.36_{10} and 101.101_2

21.36_{10}

$2*10+1*1+3*0.1+6*0.01$

$20 + 1 + 0.3 + 0.06$

101.101_2

$1*4+0*2+1*1+1*1/2+0*1/4+1*1/8$

$4 + 0 + 1 + ½ + 0 + 1/8$

$4 + 0 + 1 + 0.5 + 0 + 0.125$

So $101.101_2 = 5.625_{10}$

Example 4: Convert 5.625_{10} to binary

/2	Result	Remain
5	2	1
2	1	0
1	0	1

$\xrightarrow{101}$

$0.625 * 2 = 1.250$	1	0.25
$0.25 * 2 = 0.5$	0	0.5
$0.5 * 2 = 1$	1	0

$\xrightarrow{101.101}$

$0.625 * 2 = 1.250$	1	0.25
$0.25 * 2 = 0.5$	0	0.5
$0.5 * 2 = 1$	1	0

$\xrightarrow{101.101}$

Addition and Subtraction in binary number system:

Example 5: Add 2 to 3 in binary system.

Decimal	Binary	Addition
2	10_2	1 0
+ 3	+ 11_2	+ 1 1
5		1 0 1

10

Example 6: Add 7 to 9 in binary system.

10

Decimal	Binary	Addition
		1 1 1
7	111_2	1 1 1
+ 9	+ 1001_2	+ 1 0 0 1
16		1 0 0 0 0

10

Example 7: Subtact 2 from 3 in binary system.

Decimal	Binary	Addition
3	11_2	1 1
- 2	- 10_2	- 1 0
1		0 1

Example 8: Subtact 4 to 3 in binary system.

Decimal	Binary	Addition
		$\cancel{1}$ 10
		0 $\cancel{10}$
4	100_2	$\cancel{1}$ 0 0
- 3	- 11_2	- 1 1
1		0 0 1

Multiplying and Division in binary system:

Example 9: Multiply 2 by 3 in binary system.

Decimal	Binary	Multiplication
2	10_2	1 0
* 3	* 11_2	* 1 1
		1 0
		+ 1 0 .
		1 1 0

Example 10: Multiply 9 by 5 in binary system

Decimal	Binary	Multiplication
9	1001_2	1 0 0 1
* 5	* 101_2	* 1 0 1
45		1 0 0 1
		0 0 0 0
		+ 1 0 0 1 .
		1 0 1 1 0 1 = 45

32*1+0*16+1*8+1*4+0*2+1*1=45

Example 11: Divide 9 by 3 in binary system

```
          0 0 1 1
   11  |  1 0 0 1
       - 1 1
           0 1 1
       -     1 1
           0   0
```

Example 12: Divide 17_{10} by 4_{10} in binary system

```
          0 0 1 0 0 . 0 1
   100 |  1 0 0 0 1
       - 1 0 0
         0 0 0 0
   -          0 0 1
               0  0 1 0
                   1 0 0
                   0 0 0
```

Answer 17 / 4 = 100.01_2= 4.25_{10}

Ninevah University

College of Electronics

.Communication Dept

Digital Technique

Lec -2

Digital Technique is a subject that helps you to understand binary numbers and give the students good information about the most famous digital circuits that were used in every digital system.

2023-2022

Lecure 2

Octal and Hexadecimal System Numbers

Octal and hexadecimal systems are used in software and in printing data.

Octal number system consists of 8 symbols
They are 0, 1, 2, 3, 4, 5, 6, 7

Hexadecimal number system consists of 16 symbols
They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Where A=10, B=11, C=12, ……F=15.
Any number in hexadecimal number may be written in this form
169_{16} or 169H where **H** mean a symbol the same of sub $_{16.}$

For any number used in any system there are some information that must be known:

In Decimal number system 954 :

9	5	4	
$9*10^2$	$5*10^1$	$4*1^0$	So the **base** of this number is 10 or we called it as **radix.**

In binary number system 101:

1	0	1	
$1*2^2$	$0*2^1$	$1*2^0$	So the **base or the radix** of this number is 2.

What are the base for these system? Hexadecimal and Octal

For Hexadecimal radix=16
Octal radix=8

Q/ What is the radix and its value of the hexadecimal number 15_{16}?

1 5
$1*16^1 + 5*16^0$
$16 + 5 = 21_{10}$

Q/ Convert the following number 1F6AH to decimal

1	F	6	A
$1*16^3$	$15*16^2$	$6*16^1$	$10*16^0=$
1*4096	+ 15*256	+6*16	+ 10*1 =
4096	+ 3840	+ 96	+ 10 = 8042_{10}

Q/ What are the values of (1001) in Decimal, Octal and Hexadecimal?

Dec. $1*10^3+0*10^2+0*10^1+1*10^0 = 1001_{10}$

Oct. $1*8^3+0*8^2+0*8^1+1*8^0= 1*512+0+0+1 = 513_{10}$

Hex. $1*16^3+0*16^2+0*16^1+1*16^0= 4096+0+0+1 = 4097_{10}$

As you can note that any symbol of Octal number can be represented by 3 digit of binary number. Why? Because the largest number in octal is 7 and 7 in binary is 111

$7_8 = 111_2$
$3_8 = 011_2$

Q/ Convert 517_8 to binary number

5	1	7
101	001	111

$517_8=101001111_2$

This rule is also used for hexadecimal. Any digit in hexadecimal can be represented by 4 digits in binary number.

Q/ Convert F5E2H to binary number

F	5	E	2
1111	0101	1110	0010

$F5E2_{16} =1111010111100010_2$

Table 1: Many system are written in this table

Decimal	Binay	Oct.	Hex.	BCD	Excess-3	Gray
0	0	0	0	0000 0000	0011 0011	0
1	1	1	1	0000 0001	0011 0100	1
2	10	2	2	0000 0010	0011 0101	11
3	11	3	3	0000 0011	0011 0110	10
4	100	4	4	0000 0100	0011 0111	110
5	101	5	5	0000 0101	0011 1000	111
6	110	6	6	0000 0110	0011 1001	101
7	111	7	7	0000 0111	0011 1010	100
8	1000	10	8	0000 1000	0011 1011	1100
9	1001	11	9	0000 1001	0011 1100	1101
10	1010	12	A	0001 0000	0100 0011	1111
11	1011	13	B	0001 0001	0100 0100	1110
12	1100	14	C	0001 0010	0100 0101	1010
13	1101	15	D	0001 0011	0100 0110	1011
14	1110	16	E	0001 0100	0100 0111	1001
15	1111	17	F	0001 0101	0100 1000	1000
16	10000	20	10	0001 0110	0100 1001	11000
17	10001	21	11	0001 0111	0100 1010	11001

Binary Coded Decimal (BCD or 8421 code)

This type of number system is a decimal number but it is written with binary symbols. Each digit of decimal must be written with 4 symbol of binary number. Why? That is because the largest symbol in decimal is 9 and this number in binary is 1001 so its four.

Example: 721_{10}

0111 0010 0001 the space between the digits must be exist. If you write 721= ~~011100100001~~ this is wrong because there are no space between digits.

Note:-

Any digit in binary is called (bit)
Each four bits in binary is called (nibble)
Each eight bits in binary is called (byte)

BCD Code

- A number with k decimal digits will require 4k bits in BCD.
- Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
- A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.
- The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example: **Consider decimal 185 and its corresponding value in BCD and binary:**

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

If the binary sum is greater than or equal to 1010, we add 0110 to obtain the correct BCD sum and a carry.

BCD addition

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

Excess-3 system number

- The Excess-3 code is also called as XS-3 code.
- The Excess-3 code words are derived from the 8421 BCD code words adding (0011)2.
- It is non-weighted code used to express decimal numbers.
- Return to table 2 to compare between Exess-3 and Decimal.

Decimal number $\longrightarrow$ BCD $\xrightarrow{\text{Add number 0011}}$ XS-3

Ex: 94_{10} 1001 0100   1100 0111

;;;

Each 16 bits in binary is called (word)

Excess-3 system is the same as BCD but each digit equal to BCD+3:

It mean the zero number in BCD=0000 while zero in Ex-3= 0011

5 in BCD	=0101	Ex-3= 1000
9	=1001	= 1100
12	=0001 0010	=0100 0101 and so on.

Refer to table 1 for other examples.

Binary Codes

- Gray Code
  - The advantage is that only bit in the code group changes in going from one number to the next.
    - Error detection.
    - Representation of analog data.
    - Low power design.

Example 1

Binary ⟶ Gray

0010 ----------- 0011

1011 ---------- 1110

Example 2

Gray ⟶ Binary

0101 --------- 0110

1100 --------- 1000

Gray Code

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

Good Luck

Dena Nameer

Communication Dept. Electronics College

Logic Technology

Logic Gates

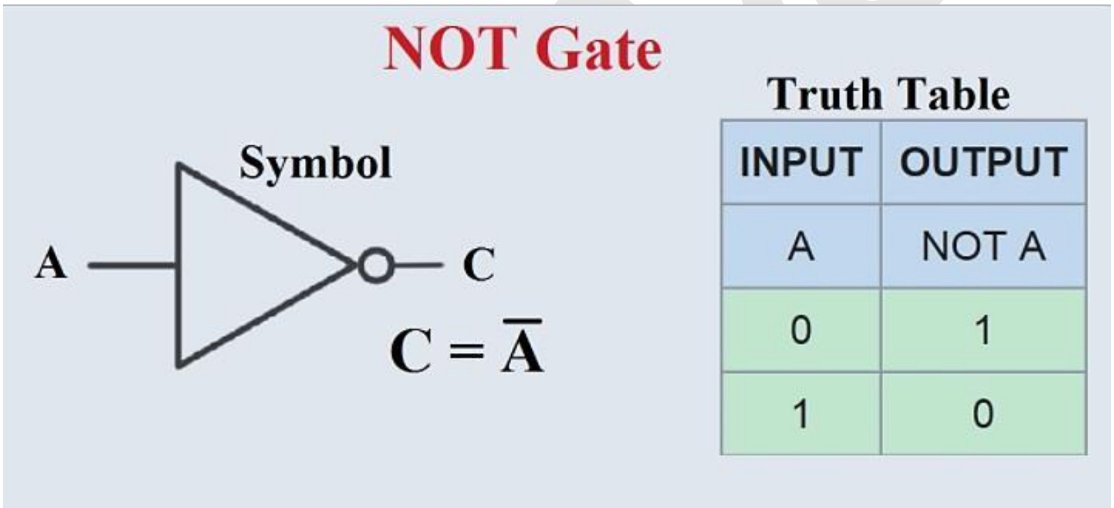
First Class

2022

By: Assistant Lecturer
Dena Nameer

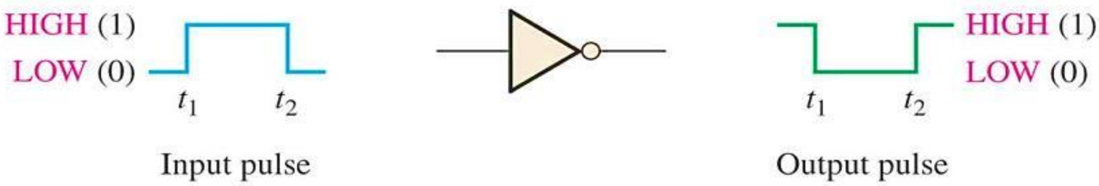
The Inverter (NOT Gate) •

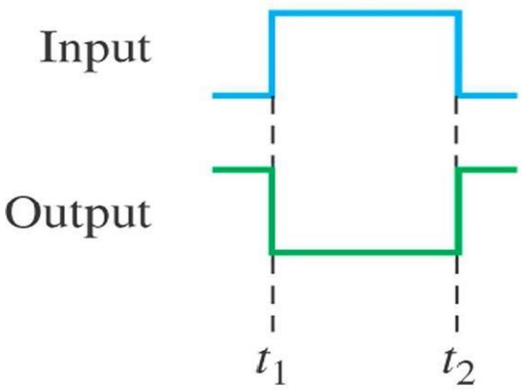
- The inverter (NOT gate) performs the operation called inversion or complementation.
- • The inverter changes one logic level to the opposite level. In terms of bits, it changes a 1 to a 0 and a 0 to a 1.
- • Standard logic *symbol* for the inverter as well as *Inverter Truth Table* are shown in Figure (1) below. •
- • When a **HIGH** level is applied to an inverter input, a **LOW** level will appear on its output. •
- When a **LOW** level is applied to its input, a **HIGH** will appear on its output.



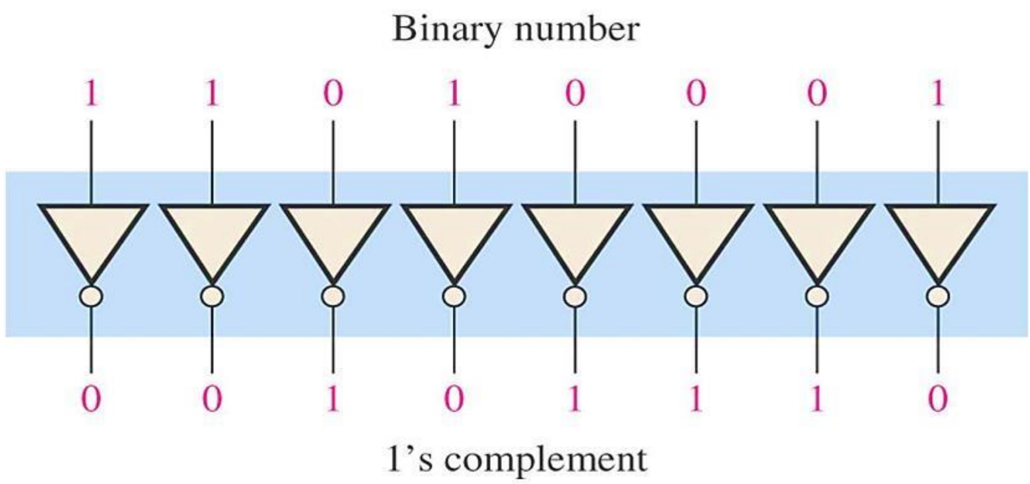
- A table such as this is called a **truth table**.
- The operation of an inverter (**NOT gate**) can be expressed as follows **:**
  - If the input variable is called A and the output variable is called $\overline{Y}$, then Y = $\overline{A}$.
  - This expression states that the output is the complement of the input, • So if A = 0, then Y =1, and if A = 1, then Y = 0. •

- The complemented variable A can be read as "A bar" or "not A".



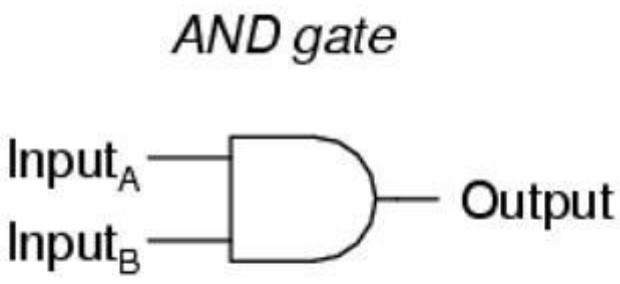


The inverter (NOT gate) is used in many applications. One of these application is to produce the 1's complement of an 8-bit(one byte) binary number. The bits of the binary number are applied to the inverter inputs and the 1's complement of the number appears on the output.



THE AND GATE

- The AND gate is one of the basic logic gates. An AND gate can have two or more inputs and performs what is known as " **logical multiplication** "
- Standard logic symbols for the AND gate are shown in Figure below as well as the truth table for a 2-input AND gate is shown in the table below.



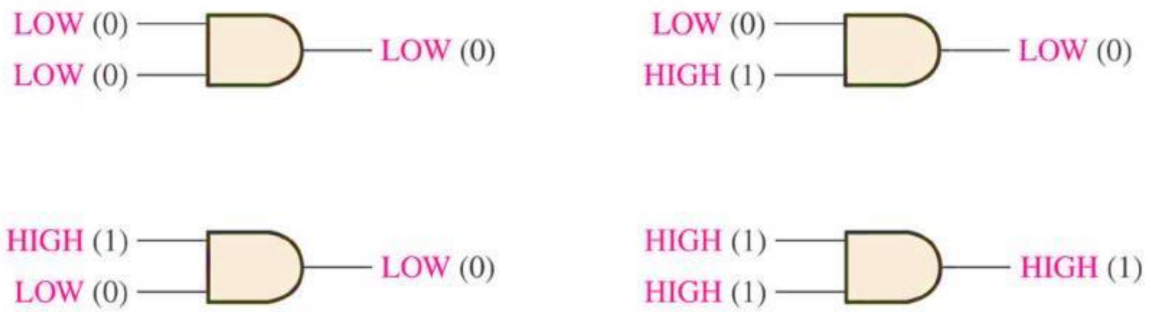
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

- An AND gate produces a HIGH output only when **all** of the inputs are HIGH. When any of the inputs is LOW, the output is LOW. Therefore, the basic purpose of an AND gate is to determine when certain conditions are simultaneously true, as indicated by HIGH levels on all of its inputs, and to produce a HIGH on its output to indicate that all these conditions are true.
- Assume the inputs of the 2-input AND gate are labeled A and B, and the output is labeled Y.
- The AND gate operation can be stated as follows:

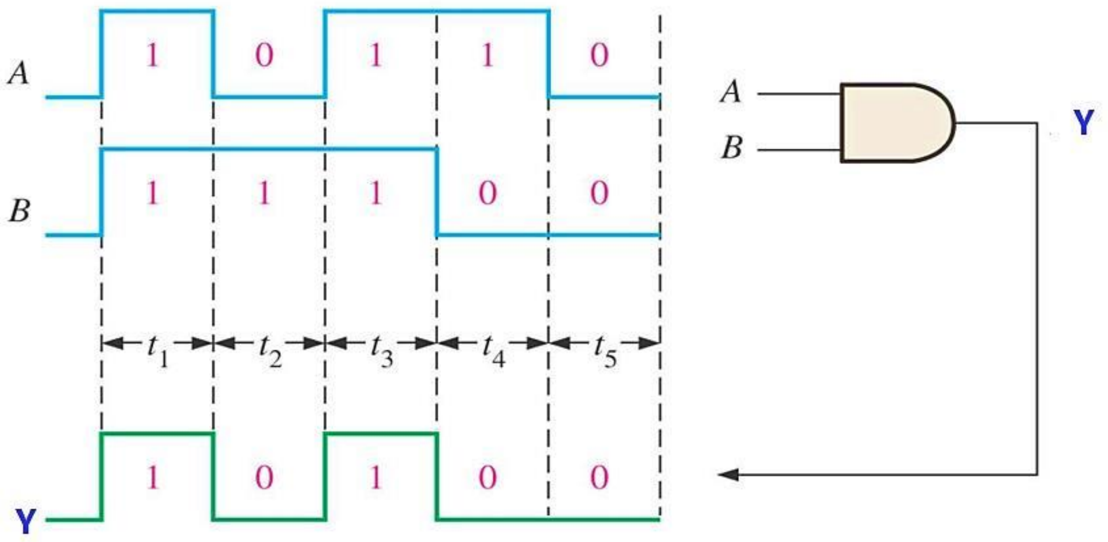
★ For a 2-input AND gate, output **Y** is **HIGH** **only** when both inputs A and B are **HIGH**.

Y is **LOW** when **either** A or B is **LOW**, or **when both** A and B are **LOW**

- Two input waveforms, A and B, are applied to an X-OR gate inputs as in Figure below, Y is the resulting output waveform.



- Figure below refers to the AND gate operation with a timing diagram showing input and output relationships.



H.W

Develop the truth table for a 3-input AND gate?

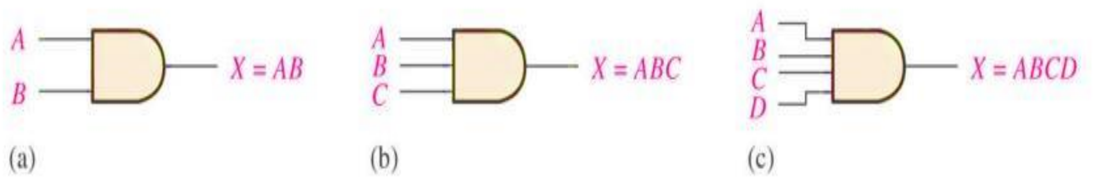
- The logical AND function of two variables is represented mathematically either by placing a dot between the two variables, as **A·B**, or by simply writing the adjacent letters without the dot, as **AB**. We will normally use the latter notation because it is easier to write.

- The operation of a 2-input AND gate can be expressed in equation form as follows:

  - If one input variable is A, the other input variable is B, and the output variable is Y, then the Boolean expression is:

$$Y = AB$$

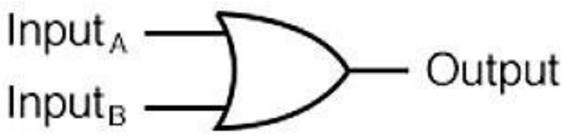
  - The figure below shows the AND gate logic symbol with two, three and four input variables and the equivalent output variable.



THE OR GATE

- The OR gate is another type of the basic gates from which all logic functions are constructed.
- An OR gate can have two or more inputs and performs what is known **as logical addition**.
- Standard logic symbol for the OR gate is shown in Figure below as well as the truth table of a 2-input OR gate is illustrated.

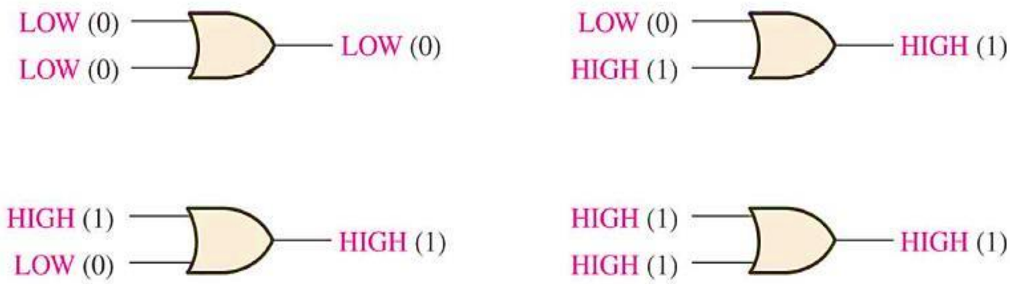
2 - input OR gate



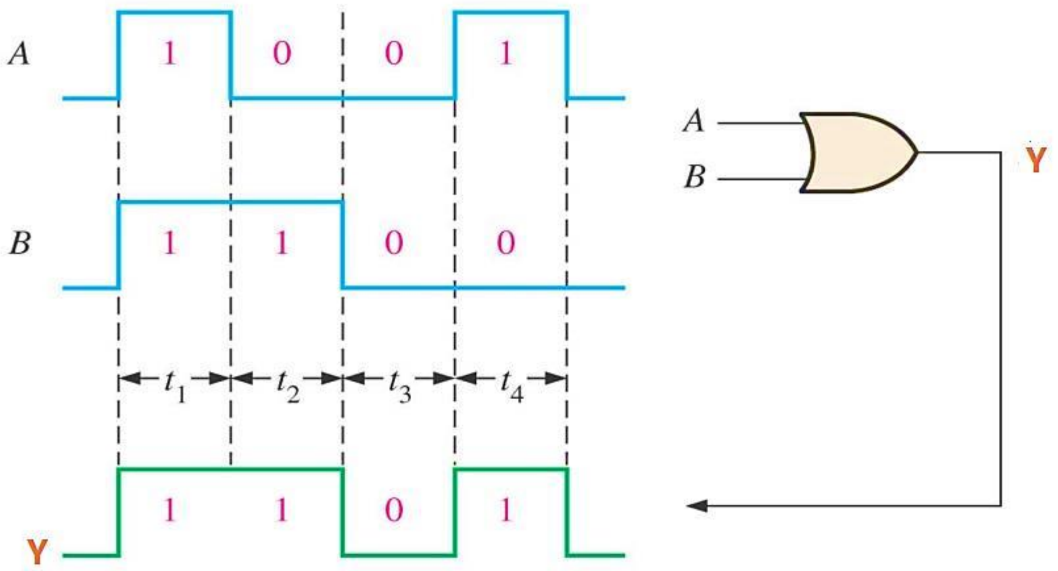
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

- This truth table can be expanded for any number of inputs; but regardless of the number of inputs, the output is HIGH when one or more inputs are HIGH.
- An OR gate can have any number of inputs greater than one.

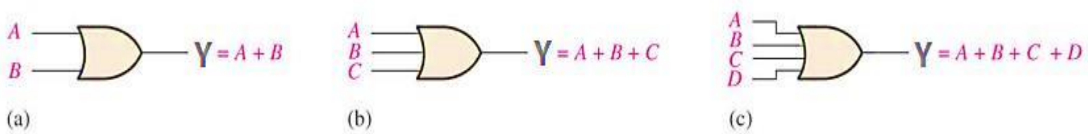
- An OR gate produces **a HIGH** on the output when **any of the inputs** is **HIGH** (when either input A or input B is HIGH, or when both A and B are HIGH).
- The output is **LOW only when <u>all</u> of the inputs** are **LOW**.



Two input waveforms, A and B, are applied to an OR gate inputs as in Figure below, Y is the resulting output waveform.

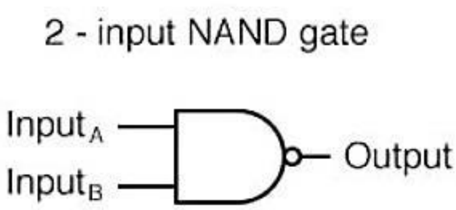


- The logical OR function of two variables is represented mathematically by a plus sign + between the two variables, for example, **A + B**.
- The operation of a 2-input OR gate can be expressed as follows:
  - If one input variable is **A**, the other input variable is **B**, and if the output variable is **Y**, then the Boolean expression is as follows; $\mathbf{Y} = \boldsymbol{A} + \boldsymbol{B}$
  - The Figure below shows the OR gate logic symbol with two, three and four input variables and the output variable indicated.



THE NAND GATE

- The NAND gate is a popular logic element because it can be used as a universal gate; that is, NAND gates can be used in combination to perform the AND, OR, and inverter gates.
- The term NAND is a contraction of NOT-AND and implies an AND function with a complemented (inverted) output.
- The standard logic symbol for a 2-input NAND gate and its equivalency to an **AND gate followed by an inverter** are shown in Figure below, also the truth table of the logical operation of the 2-input NAND gate is shown in table below:



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

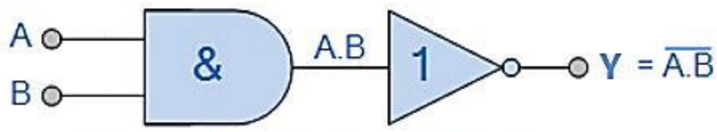
- For the specific case of a 2-input NAND gate, as shown in Figure above with the inputs labeled A and B and the output (Y), the operation can be stated as follows:
  - For a 2-input NAND gate, output **Y** is LOW only when both inputs A and B are HIGH; **Y** output is HIGH when either A or B is LOW, or when both A and B are LOW.

Note

That this operation is **opposite** to the <u>AND gate</u> in terms of the output level. In other words, The NAND is the same as AND except the output is **inverted**.

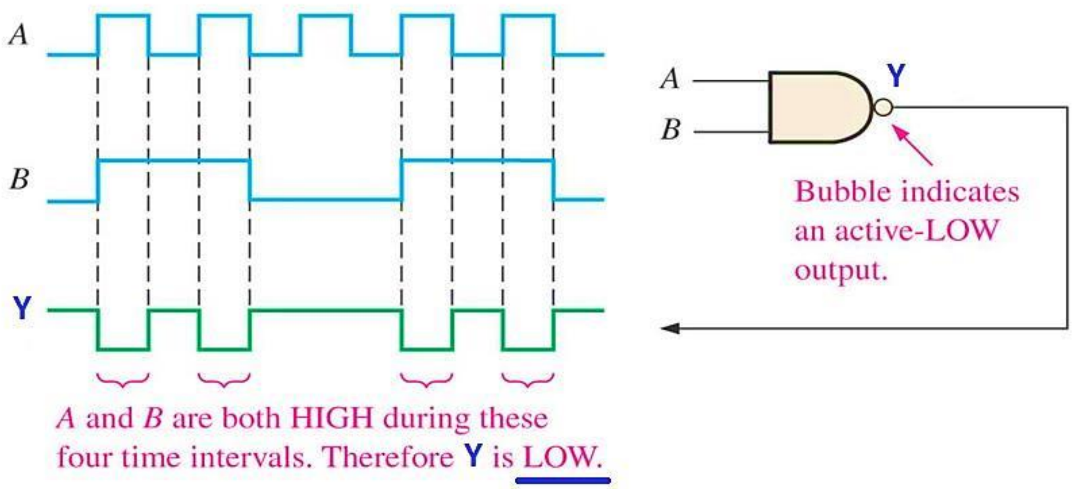
The NAND gate is equivalent to an AND gate followed by NOT (Invertor) as shown before. So the expression will be as follows, if the inputs are A and B , and the output is Y:

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



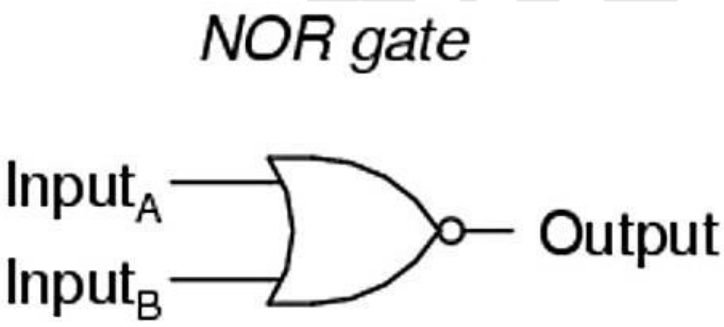
2-input "AND" gate plus a "NOT" gate

Two input waveforms, A and B, are applied to a NAND gate inputs as in Figure below, Y is the resulting output waveform.



THE NOR GATE

- The NOR gate, like the NAND gate, is a useful logic element because it can also be used as a universal gate; that is, NOR gates can be used in combination to perform the AND, OR, and inverter operations.
- The term NOR is a contraction of **NOT-OR** and implies an OR function **with an inverted** (complemented) output.
- The standard logic symbol for a 2-input NOR gate and its equivalent OR gate *followed by* an inverter are shown in Figure below. Also the truth table for a 2-input NOR gate is shown in the table below.



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

As a result, the NOR is the same as the OR except the output is inverted.

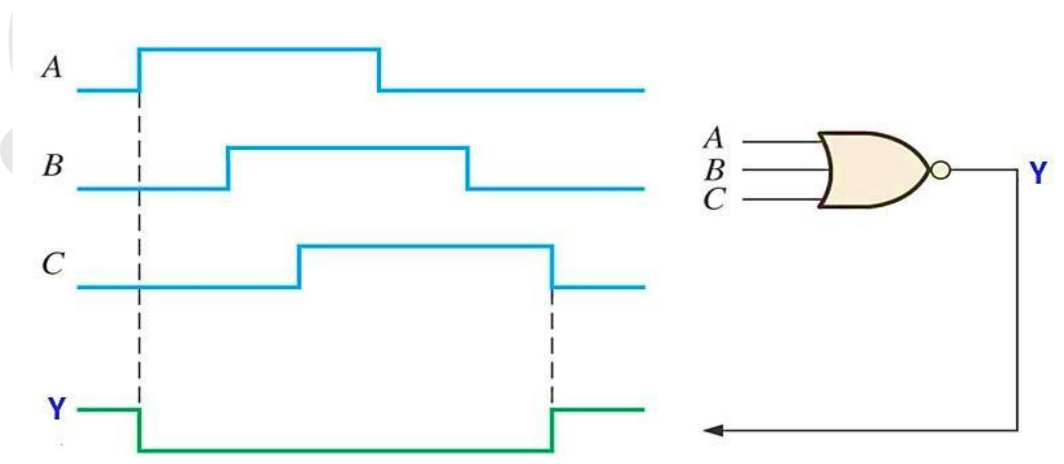
- A NOR gate produces a LOW output when any of its inputs is HIGH. Only when all of its inputs are LOW is the output HIGH.

- For a 2-input NOR gate, output X is **LOW** when either input A or input B is **HIGH**, or when **both** A and B are **HIGH**.
- **Y (output)** is **HIGH only** when both A and B are **LOW**.

If three input waveforms, **A**, **B** and **C**, are applied to the NOR gate inputs as in Figure below, what is the resulting output waveform?

Solution:

The output **Y** is LOW when any input is HIGH as shown by the output waveform Y in the timing diagram

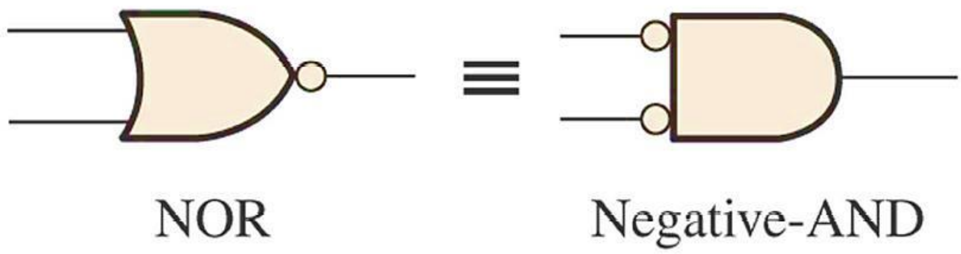


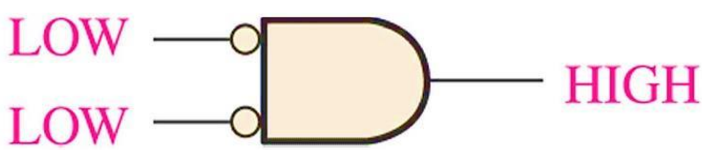
NEGATIVE-AND EQUIVALENT OPERATION OF THE NOR GATE

- A NOR gate can be used for an AND operation that requires all LOW inputs to produce a HIGH output. This aspect of NOR operation is called **Negative-AND**.

- For a 2-input NOR gate performing a negative-AND operation, output **Y** is **HIGH** only when both inputs **A** and **B** are LOW.

- When a NOR gate is used to detect all LOWs on its inputs rather than one or more HIGHs, it is performing the negative-AND operation and is represented by the standard symbol in Figure below



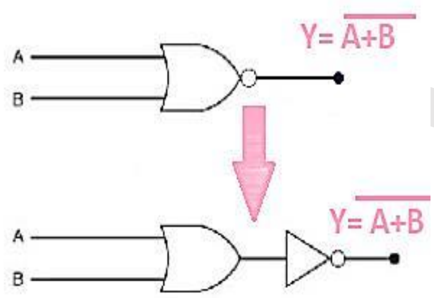


LOGIC EXPRESSION FOR A NOR GATE

The Boolean expression for the output of a 2-input NOR gate can be written as:

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

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



Home Work

1. When is the output of a NOR gate HIGH?
2. When is the output of a NOR gate LOW?
3. Describe the functional difference between a NOR gate and a negative-AND gate?
4. Do they both have the same truth table?
5. Write the output expression for a 3-input NOR with input variables A, B, and C?

Good luck

Dena.N

Logic Gates

XOR gate & XNOR- gate

LEC-2/Part2

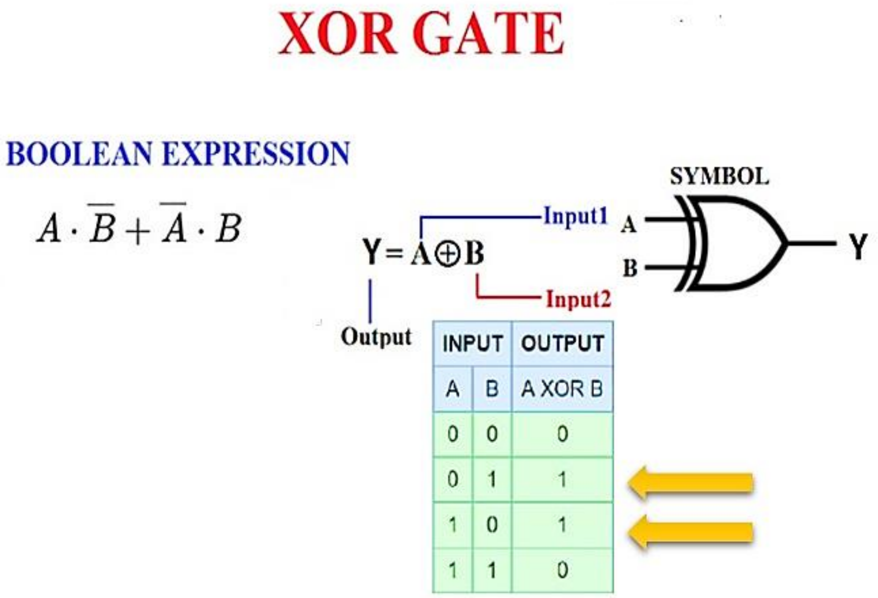
First Class

2022

By: Assistant Lecturer
Dena Nameer

Exclusive-OR (XOR) gate

- The XOR gate has **only two inputs.**
- The output of an **exclusive-OR** gate is HIGH only when the two inputs are at opposite logic levels.
- This operation can be stated as follows with reference to inputs A and B and output Y:
- For an exclusive-OR gate, output **Y** is **HIGH** when input **A** is LOW and input B is • HIGH, or when input A is HIGH and input B is LOW; **Y** is **LOW** when both A and B are HIGH or LOW.
- Standard logic symbols for the **exclusive OR (XOR) gate** are shown in Figure below as well as the truth table for **XOR** gate is shown in table below:

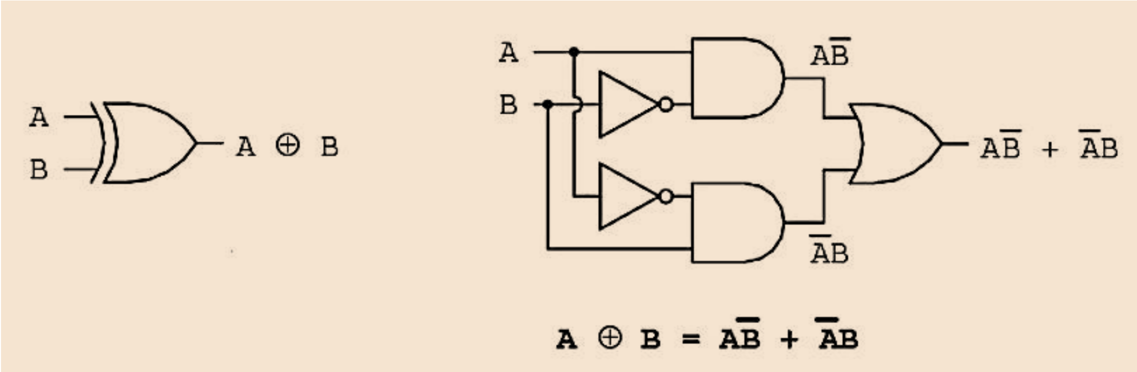


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

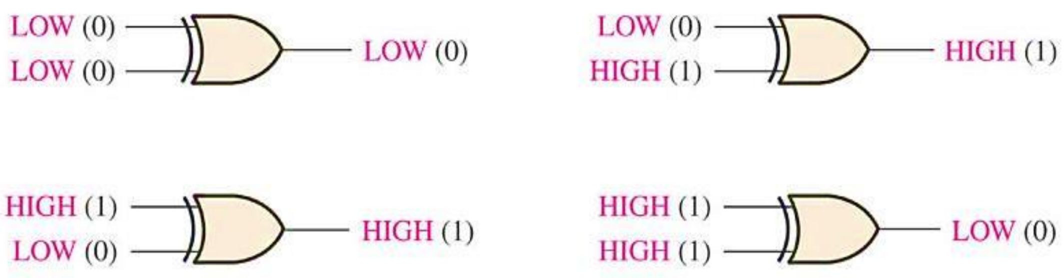
- The Boolean expression for the output of a 2-input XOR gate can be written as:

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

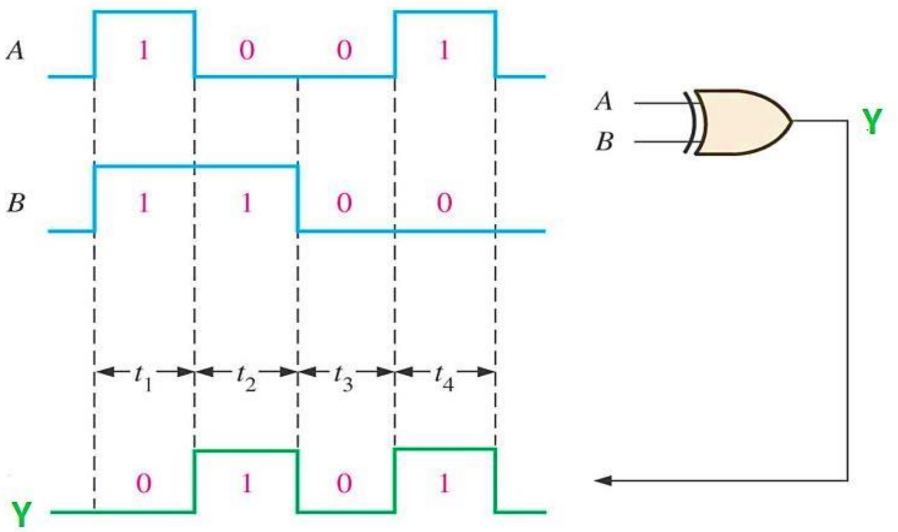
- As a result, we can make XOR gate by using NOT, AND, and OR gates as Figure below:



The four possible input combination and the resulting outputs for an XOR gate are illustrated in figure below:

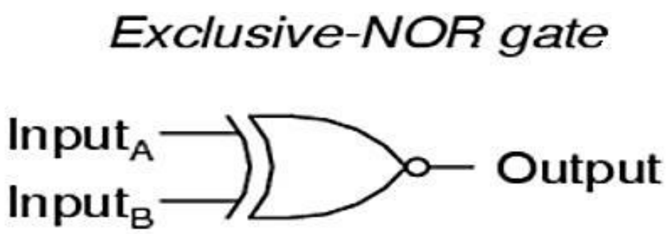


Example of exclusive-OR gate operation with pulse waveform inputs.



THE EXCLUSIVE-NOR GATES

- The standard logic symbol for an exclusive-NOR (XNOR) gate and its equivalent exclusive-OR (XOR) gate **followed by an inverter** are shown in Figure below as well as the truth table for exclusive NOR (XNOR) gate is shown in table below:



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

2-input “Ex-OR” Gate plus a “ NOT” Gate

- Like the XOR gate, an **XNOR** has only two inputs.
- The bubble on the output of the XNOR symbol indicates that its output is opposite that of the **XOR** gate.
- When the two input logic levels are opposite, the output of the exclusive-NOR gate is **LOW**.
- For an exclusive-NOR gate, output **Y** is LOW when input A is LOW and input B is HIGH, or when A is HIGH and B is LOW; **Y** is HIGH when both A and B are HIGH or LOW.

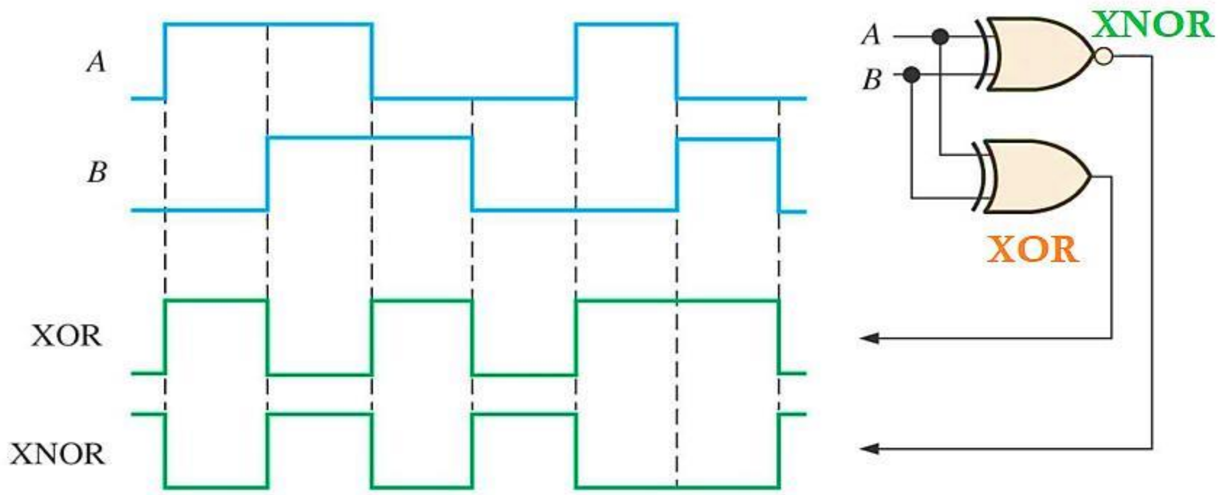
- The Boolean expression for the output of a 2-input XNOR gate can be written as:

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

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

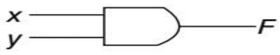
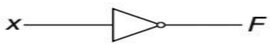
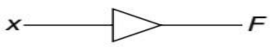
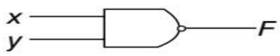
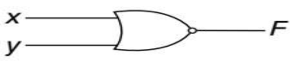
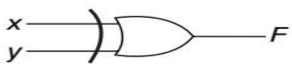
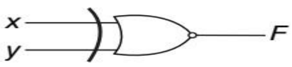
Example

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, A and B, in Figure below?



Solution: the XOR output is HIGH only when both inputs are at opposite levels and the XNOR output is HIGH only when both inputs are the same levels.

Symbols and truth tables for logic gates (limited to 2 inputs)

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

Digital logic gates

Home Work

1. When is the output of an XOR gate HIGH?
2. When is the output of an XNOR gate HIGH?
3. How can you use an XOR gate to detect when two bits are different?

IC Digital Logic Families

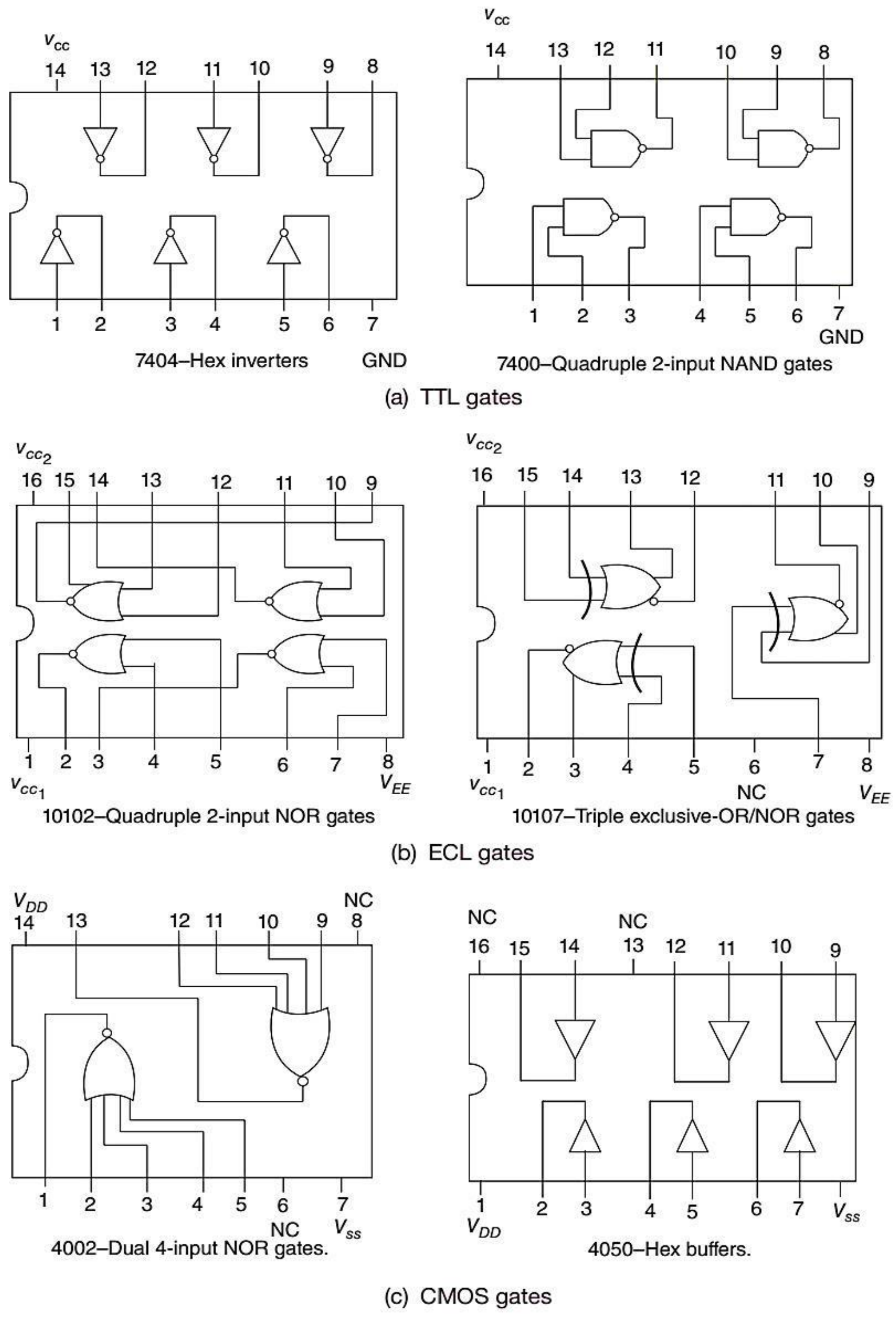
Digital IC gates are classified not only by their logic operation, but also by the specific logic-circuit family to which they belong. Each logic family has its own basic electronic circuit upon which more complex digital circuits and functions are developed.

The basic circuit in each family is either a NAND or a NOR gate. The electronic components employed in the construction of the basic circuit are usually used to name the logic family.

Many different logic families of digital ICs have been introduced commercially. The ones that have achieved widespread popularity are listed below:

TTL **T**ransistor-**T**ransistor **L**ogic
ECL **E**mitter-**C**oupled **L**ogic
MOS **M**etal-**O**xide **S**emiconductor
CMOS **C**omplementary **M**etal-**O**xide **S**emiconductor
I^2L **I**ntegrated-injection **L**ogic

- ***TTL*** has an extensive list of digital functions and is currently the most popular logic family.
- ***ECL*** is used in systems requiring **high-speed operations**.
- ***MOS and I^2L*** are used in circuits requiring high component density.
- ***CMOS*** is used in systems requiring **low power consumption**.



Some typical integrated-circuit gates

Good luck

Dena.N

Boolean Algebra

LEC-3/Part2

First Class

2022

Maher

Introduction

George Boole, a nineteenth-century English Mathematician, developed a system of logical algebra by which reasoning can be expressed mathematically. In 1854, Boole published a classic book, "An Investigation of the Laws of thought" on which he founded the Mathematical theories of Logic and Probabilities, Boole‟s system of logical algebra, now called Boolean algebra, was investigated as a tool for analyzing and designing relay switching circuits by Claude E. Shannon at the Massachusetts institute of Technology in 1938. Shannon, a research assistant in the Electrical Engineering Department, wrote a thesis entitled "A" symbolic Analysis of Relay and Switching Circuits. As a result of his work, Boolean algebra is now, used extensively in the analysis and design of logical circuits. Today Boolean algebra is the backbone of computer circuit analysis.

Two Valued Logical Symbol:

The electronics circuits and signals

- A **logic 1** will represent **closed switch**, a **high voltage**, or an "**on" lamp.**
- A **logic 0** will represent an **open switch**, **low voltage**, or an "**off" lamp**.

These describe the only two states that exist in digital logic systems and will be used to represent the in and out conditions of logic gates.

Fundamental Concepts of Boolean Algebra:

Boolean algebra is a logical algebra in which symbols are used to represent logic levels. Any symbol can be used, however, letters of the alphabet are generally used. Since the logic levels are generally associated with the symbols 1 and 0, whatever letters are used as variables that can take the values of 1 or 0.

Boolean algebra has only two mathematical operations, ***addition*** and ***multiplication***. These operations are associated with the **OR** gate and the **AND** gate, respectively.

Logical Addition:

When the **+** (the logical addition) symbol is placed between two variables, say X and Y, since both X and Y can take only the role **0 and 1**, we can define the **+** Symbol by listing, all possible combinations for X and Y and the resulting value of **X + Y**.

The possible input and output combinations may arranged as follows:

0 + 0 = 0
0 + 1 =1
1 + 0 = 1
1 + 1 = 1

This table represents a standard binary addition, except for the last entry. When both X and Y represents 1‟s, the value of X + Y is 1. The symbol + therefore does not has the "**Normal" meaning**, but is a Logical addition symbol. The plus symbol (+) read as "**OR**", therefore **X +Y** is read as **X or Y**.

Logical Multiplication:

We can define the "**.**" (logical multiplication) symbol or **AND** operator by listing all possible combinations for (input) variables X and Y and the resulting (output) value of X **.** Y as,

0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1. 1 = 1

Note

Three of the basic laws of Boolean algebra are the same as in ordinary algebra; the commutative law, the associative law and the distributive law.

- **The commutative law:** for addition and multiplication of two variables is written as,

$$A + B = B + A$$

And $$A \cdot B = B \cdot A$$

- **The associative law:** for addition and multiplication of three variables is written as,

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

And $$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

- **The distributive law**: for three variables involves, both addition and multiplication and is written as,

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

Note that while either '+' and „.‟ s can be used freely. The two cannot be mixed without ambiguity in the absence of further rules. For example does A . B + C means (A . B) + C or A . (B+ C)?

These two form different values for A = 0, B = 1 and C = 1, because we have

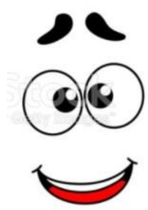
$$(A \cdot B) + C = (0.1) + 1 = 1$$
$$\text{and } A \cdot (B + C) = 0 \cdot (1 + 1) = 0$$

which are different. The rule which is used is that „.‟ is always performed before '+'. Thus X . Y + Z is (X.Y) + Z.

Basic Duality in Boolean Algebra:

We state the duality theorem without proof. Starting with a Boolean relation, we can derive another Boolean relation by

1) **Changing each OR (+) sign to an AND (.) sign.**
2) **Changing each AND (.) sign to an OR (+) sign.**
3) **Complementary each 0 and 1 For instance.**



$A + 0 = A$

The dual relation is $A \cdot 1 = A$

Also since $A(B + C) = AB + AC$ by distributive law.

Its dual relation is $A + BC = (A + B)(A + C)$

Fundamental Laws and Theorems of Boolean Algebra:

1. $X + 0 = X$
2. $X + 1 = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$

OR operations

5. $X \cdot 0 = 0$
6. $X \cdot 1 = X$
7. $X \cdot X = X$
8. $X \cdot \overline{X} = 0$

AND operations

9. $X = \overline{\overline{X}}$ Double Complement

10. $X + Y = Y + X$
11. $XY = YX$

Commutative laws

12. $(X + Y) + Z = X + (Y + Z)$

13. $(X . Y).Z = X.(Y . Z)$

Associative laws

14. $X (Y + Z) = XY + XZ$ Distribution Law

15. $X + Y.Z = (X + Y) . (X + Z)$ Dual of Distributive Law

16. $X + XZ = X$

17. $X (X + Z) = X$

Laws of absorption

18. $X + \bar{X} Y = X + Y$

19. $X (\bar{X} + Y) = X.Y$

Identity Theorems

20. $\overline{X+Y} = \bar{X} . \bar{Y}$

21. 21. $\overline{X.Y} = \bar{X} + \bar{Y}$

De Morgan's Theorems

Good luck

Dena.N

Boolean Algebra

LEC-3/Part2

First Class

2022

By: Assistant Lecturer
Dena Nameer

Introduction

George Boole, a nineteenth-century English Mathematician, developed a system of logical algebra by which reasoning can be expressed mathematically. In 1854, Boole published a classic book, "An Investigation of the Laws of thought" on which he founded the Mathematical theories of Logic and Probabilities, Boole‟s system of logical algebra, now called Boolean algebra, was investigated as a tool for analyzing and designing relay switching circuits by Claude E. Shannon at the Massachusetts institute of Technology in 1938. Shannon, a research assistant in the Electrical Engineering Department, wrote a thesis entitled "A" symbolic Analysis of Relay and Switching Circuits. As a result of his work, Boolean algebra is now, used extensively in the analysis and design of logical circuits. Today Boolean algebra is the backbone of computer circuit analysis.

Two Valued Logical Symbol:

The electronics circuits and signals

- A **logic 1** will represent **closed switch**, a **high voltage**, or an "**on" lamp.**
- A **logic 0** will represent an **open switch**, **low voltage**, or an "**off" lamp**.

These describe the only two states that exist in digital logic systems and will be used to represent the in and out conditions of logic gates.

Fundamental Concepts of Boolean Algebra:

Boolean algebra is a logical algebra in which symbols are used to represent logic levels. Any symbol can be used, however, letters of the alphabet are generally used. Since the logic levels are generally associated with the symbols 1 and 0, whatever letters are used as variables that can take the values of 1 or 0.

Boolean algebra has only two mathematical operations, ***addition*** and ***multiplication***. These operations are associated with the **OR** gate and the **AND** gate, respectively.

Logical Addition:

When the **+** (the logical addition) symbol is placed between two variables, say X and Y, since both X and Y can take only the role **0 and 1**, we can define the **+** Symbol by listing, all possible combinations for X and Y and the resulting value of **X + Y**.

The possible input and output combinations may arranged as follows:

0 + 0 = 0
0 + 1 =1
1 + 0 = 1
1 + 1 = 1

This table represents a standard binary addition, except for the last entry. When both X and Y represents 1‟s, the value of X + Y is 1. The symbol + therefore does not has the "**Normal" meaning**, but is a Logical addition symbol. The plus symbol (+) read as "**OR**", therefore **X +Y** is read as **X or Y**.

Logical Multiplication:

We can define the "**.**" (logical multiplication) symbol or **AND** operator by listing all possible combinations for (input) variables X and Y and the resulting (output) value of X **.** Y as,

0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1. 1 = 1

Note

Three of the basic laws of Boolean algebra are the same as in ordinary algebra; the commutative law, the associative law and the distributive law.

- **The commutative law:** for addition and multiplication of two variables is written as,

$$A + B = B + A$$

And $$A . B = B . A$$

- **The associative law:** for addition and multiplication of three variables is written as,

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

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

- **The distributive law**: for three variables involves, both addition and multiplication and is written as,

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

Note that while either '+' and „.‟ s can be used freely. The two cannot be mixed without ambiguity in the absence of further rules. For example does A . B + C means (A . B) + C or A . (B+ C)?

These two form different values for A = 0, B = 1 and C = 1, because we have

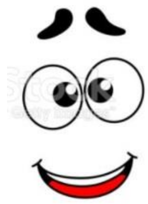
$$(A . B) + C = (0.1) + 1 = 1$$
$$\text{and } A . (B + C) = 0 . (1 + 1) = 0$$

which are different. The rule which is used is that „.‟ is always performed before '+'. Thus X . Y + Z is (X.Y) + Z.

Basic Duality in Boolean Algebra:

We state the duality theorem without proof. Starting with a Boolean relation, we can derive another Boolean relation by

1) **Changing each OR (+) sign to an AND (.) sign.**
2) **Changing each AND (.) sign to an OR (+) sign.**
3) **Complementary each 0 and 1 For instance.**



$$\mathbf{A + 0 = A}$$

The dual relation is $\mathbf{A \cdot 1 = A}$

Also since **A (B + C) = AB + AC** by distributive law.

Its dual relation is **A + B C = (A + B) (A + C)**

Fundamental Laws and Theorems of Boolean Algebra:

1. $X + 0 = X$
2. $X + 1 = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$

OR operations

5. $X \cdot 0 = 0$
6. $X \cdot 1 = X$
7. $X \cdot X = X$
8. $X \cdot \overline{X} = 0$

AND operations

9. $\overline{\overline{X}} = X$ **Double Complement**

10. $X + Y = Y + X$

11. $X.Y = Y.X$

Commutative laws

12. $(X + Y) + Z = X + (Y + Z)$

13. $(X.Y).Z = X.(Y.Z)$

Associative laws

14. $X.(Y + Z) = XY + XZ$ **Distribution Law**

15. $X + (Y.Z) = (X + Y).(X + Z)$ **Dual of Distributive Law**

16. $X + XZ = X$

17. $X(X + Z) = X$

Laws of absorption

18. $X + \bar{X}Y = X + Y$

19. $X(\bar{X} + Y) = X.Y$

Identity Theorems

20. $\overline{X+Y} = \bar{X}.\bar{Y}$

21. $\overline{X.Y} = \bar{X} + \bar{Y}$

De Morgan's Theorems

Laws of Boolean Algebra

- Commutative Laws

  $A + B = B + A$

  $A \cdot B = B \cdot A$

- Associative Laws

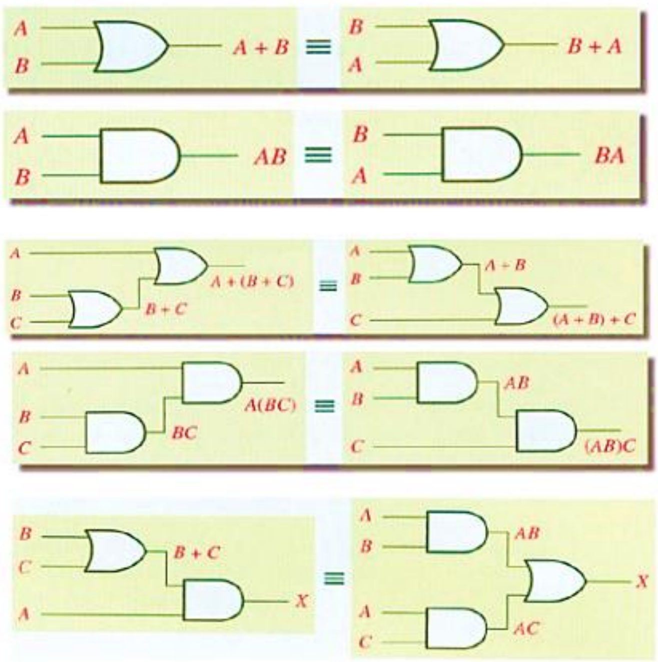
  $A + (B + C) = (A + B) + C$

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

- Distributive Law

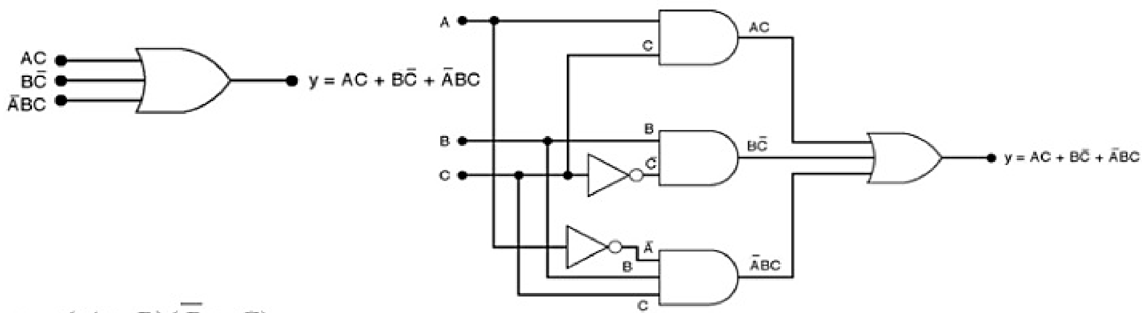
  $A \cdot (B + C) = A \cdot B + A \cdot C$

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

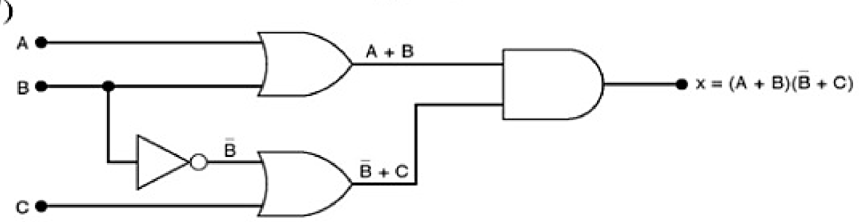


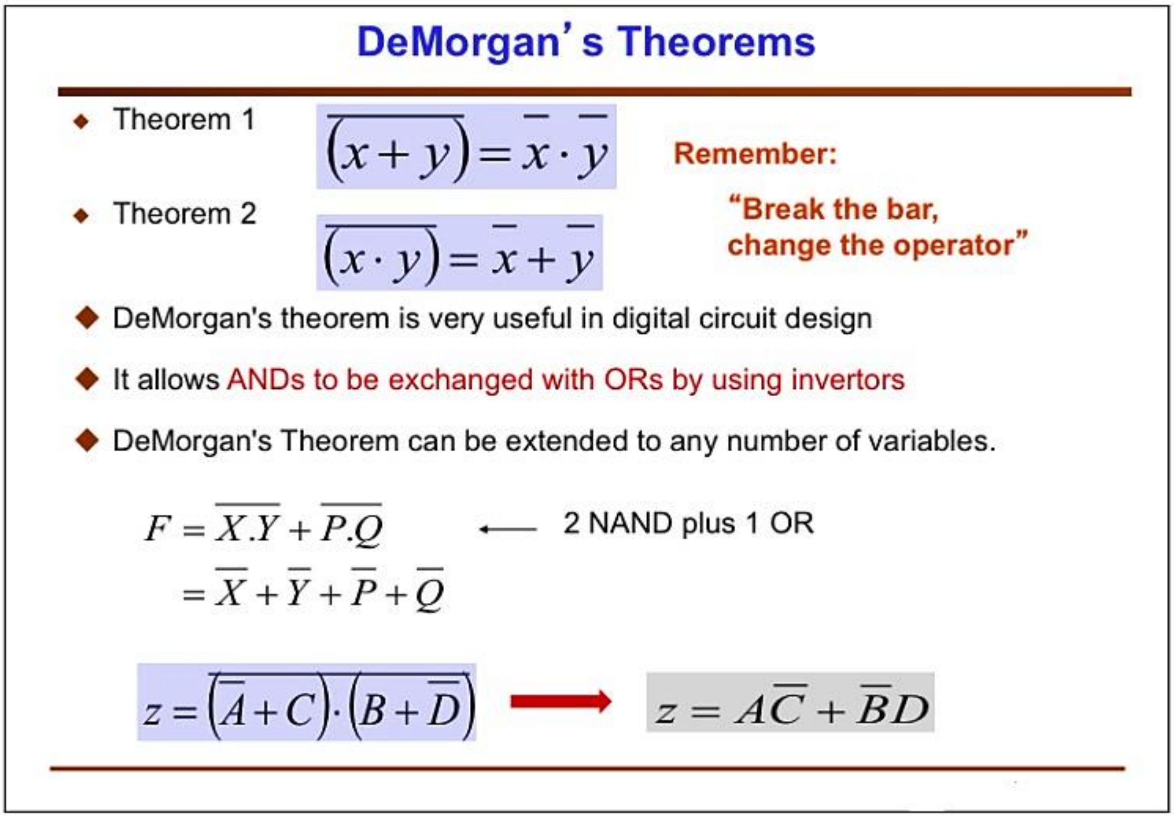
Implementing Circuits From Boolean Expressions

- When the operation of a circuit is defined by a Boolean expression, we can *draw a logic-circuit* diagram directly from that expression.



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





De'Morgan's Theorems is important to Boolean logic. They allow us to **exchange OR operation with AND operation and vice versa**. Applying De'Morgan, we can also simplify Boolean expression in many cases.

Good luck

Dena.N

Boolean Algebra

LEC-3/Part2

First Class

2022

By: Assistant Lecturer
Dena Nameer

Introduction

George Boole, a nineteenth-century English Mathematician, developed a system of logical algebra by which reasoning can be expressed mathematically. In 1854, Boole published a classic book, "An Investigation of the Laws of thought" on which he founded the Mathematical theories of Logic and Probabilities, Boole‟s system of logical algebra, now called Boolean algebra, was investigated as a tool for analyzing and designing relay switching circuits by Claude E. Shannon at the Massachusetts institute of Technology in 1938. Shannon, a research assistant in the Electrical Engineering Department, wrote a thesis entitled "A" symbolic Analysis of Relay and Switching Circuits. As a result of his work, Boolean algebra is now, used extensively in the analysis and design of logical circuits. Today Boolean algebra is the backbone of computer circuit analysis.

Two Valued Logical Symbol:

The electronics circuits and signals

- A **logic 1** will represent **closed switch**, a **high voltage**, or an "**on" lamp.**
- A **logic 0** will represent an **open switch**, **low voltage**, or an "**off" lamp**.

These describe the only two states that exist in digital logic systems and will be used to represent the in and out conditions of logic gates.

Fundamental Concepts of Boolean Algebra:

Boolean algebra is a logical algebra in which symbols are used to represent logic levels. Any symbol can be used, however, letters of the alphabet are generally used. Since the logic levels are generally associated with the symbols 1 and 0, whatever letters are used as variables that can take the values of 1 or 0.

Boolean algebra has only two mathematical operations, ***addition*** and ***multiplication***. These operations are associated with the **OR** gate and the **AND** gate, respectively.

Logical Addition:

When the **+** (the logical addition) symbol is placed between two variables, say X and Y, since both X and Y can take only the role **0 and 1**, we can define the **+** Symbol by listing, all possible combinations for X and Y and the resulting value of **X + Y**.

The possible input and output combinations may arranged as follows:

0 + 0 = 0
0 + 1 =1
1 + 0 = 1
1 + 1 = 1

This table represents a standard binary addition, except for the last entry. When both X and Y represents 1"s, the value of X + Y is 1. The symbol + therefore does not has the "**Normal" meaning**, but is a Logical addition symbol. The plus symbol (+) read as "**OR**", therefore **X +Y** is read as **X or Y**.

Logical Multiplication:

We can define the "**.**" (logical multiplication) symbol or **AND** operator by listing all possible combinations for (input) variables X and Y and the resulting (output) value of X **.** Y as,

0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1. 1 = 1

Note

Three of the basic laws of Boolean algebra are the same as in ordinary algebra; the commutative law, the associative law and the distributive law.

- **The commutative law:** for addition and multiplication of two variables is written as,

$$A + B = B + A$$

And $$A . B = B . A$$

- **The associative law:** for addition and multiplication of three variables is written as,

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

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

- **The distributive law**: for three variables involves, both addition and multiplication and is written as,

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

Note that while either '+' and „.‟ s can be used freely. The two cannot be mixed without ambiguity in the absence of further rules. For example does A . B + C means (A . B) + C or A . (B+ C)?

These two form different values for A = 0, B = 1 and C = 1, because we have

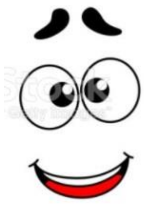
$$(A . B) + C = (0.1) + 1 = 1$$
$$\text{and } A . (B + C) = 0 . (1 + 1) = 0$$

which are different. The rule which is used is that „.‟ is always performed before '+'. Thus X . Y + Z is (X.Y) + Z.

Basic Duality in Boolean Algebra:

We state the duality theorem without proof. Starting with a Boolean relation, we can derive another Boolean relation by

1) **Changing each OR (+) sign to an AND (.) sign.**
2) **Changing each AND (.) sign to an OR (+) sign.**
3) **Complementary each 0 and 1 For instance.**



$$A + 0 = A$$

The dual relation is $A \cdot 1 = A$

Also since **$A (B + C) = AB + AC$** by distributive law.

Its dual relation is **$A + B C = (A + B) (A + C)$**

Fundamental Laws and Theorems of Boolean Algebra:

1. $X + 0 = X$
2. $X + 1 = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$

OR operations

5. $X \cdot 0 = 0$
6. $X \cdot 1 = X$
7. $X \cdot X = X$
8. $X \cdot \overline{X} = 0$

AND operations

9. $\overline{\overline{X}} = X$

Double Complement

10. $X + Y = Y + X$

11. $X.Y = Y.X$

Commutative laws

12. $(X + Y) + Z = X + (Y + Z)$

13. $(X.Y).Z = X.(Y.Z)$

Associative laws

14. $X.(Y + Z) = XY + XZ$ **Distribution Law**

15. $X + (Y.Z) = (X + Y).(X + Z)$ **Dual of Distributive Law**

16. $X + XZ = X$

17. $X(X + Z) = X$

Laws of absorption

18. $X + \bar{X}Y = X + Y$

19. $X(\bar{X} + Y) = X.Y$

Identity Theorems

20. $\overline{X+Y} = \bar{X}.\bar{Y}$

21. $\overline{X.Y} = \bar{X} + \bar{Y}$

De Morgan's Theorems

Laws of Boolean Algebra

- Commutative Laws

  A + B = B + A

  A • B = B • A

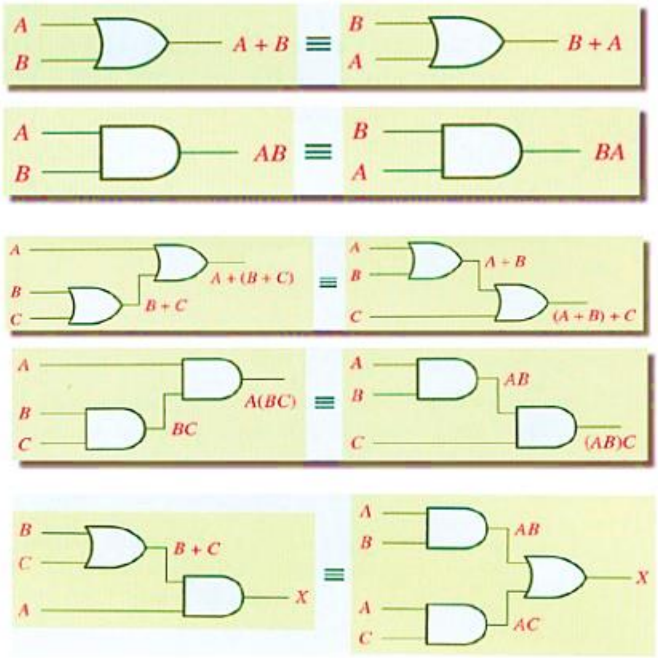
- Associative Laws

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

  A • (B • C) = (A • B) • C

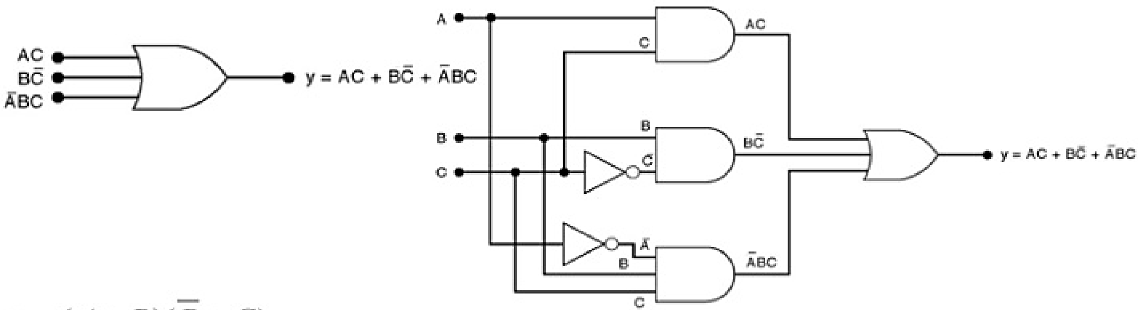
- Distributive Law

  A • (B + C) = A • B + A • C
  A (B + C) = A B + A C

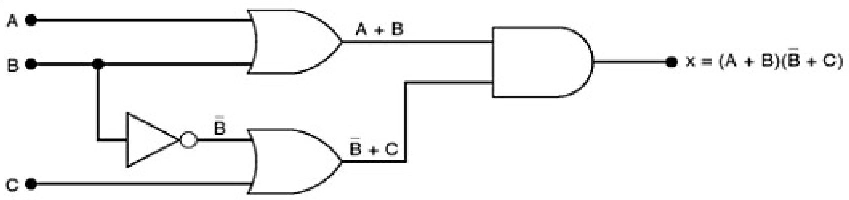


Implementing Circuits From Boolean Expressions

- When the operation of a circuit is defined by a Boolean expression, we can *draw a logic-circuit* diagram directly from that expression.



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



DeMorgan's Theorems

- Theorem 1 $\overline{(x+y)} = \bar{x}\cdot\bar{y}$
- Theorem 2 $\overline{(x\cdot y)} = \bar{x}+\bar{y}$

Remember:

"Break the bar, change the operator"

- DeMorgan's theorem is very useful in digital circuit design
- It allows ANDs to be exchanged with ORs by using invertors
- DeMorgan's Theorem can be extended to any number of variables.

$$F = \overline{X.Y} + \overline{P.Q} \quad \longleftarrow \text{2 NAND plus 1 OR}$$
$$= \bar{X} + \bar{Y} + \bar{P} + \bar{Q}$$

$$z = \overline{(\bar{A}+C)\cdot(B+\bar{D})} \longrightarrow z = A\bar{C} + \bar{B}D$$

De'Morgan's Theorems is important to Boolean logic. They allow us to **exchange OR operation with AND operation and vice versa**. Applying De'Morgan, we can also simplify Boolean expression in many cases.

Good luck

Dena.N

Kurnough map

LEC-1-2-3/Part3

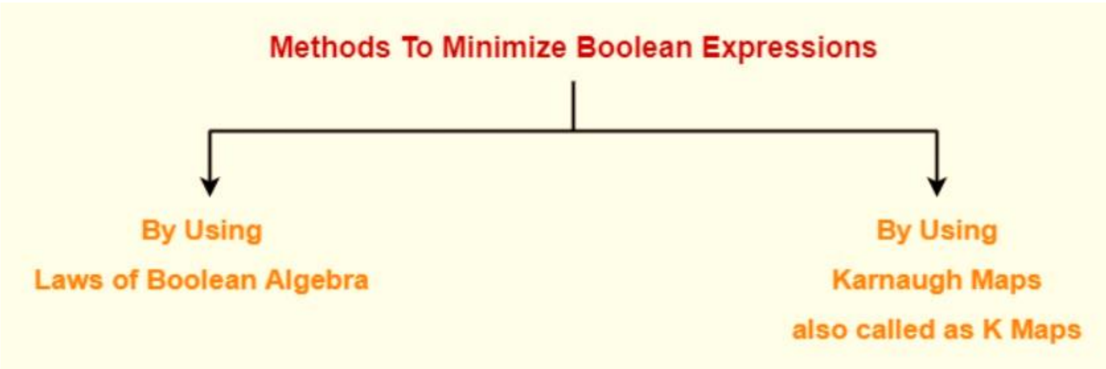
First Class

2022

By: Assistant Lecturer
Dena Nameer

Methods to minimize Boolean expression:

1. By using laws of Boolean Algebra.
2. By using Karnaugh Maps also called as K Maps.



Introduction for Karnaugh map

A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized. Since any Boolean function can be expressed as a sum of minterms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.

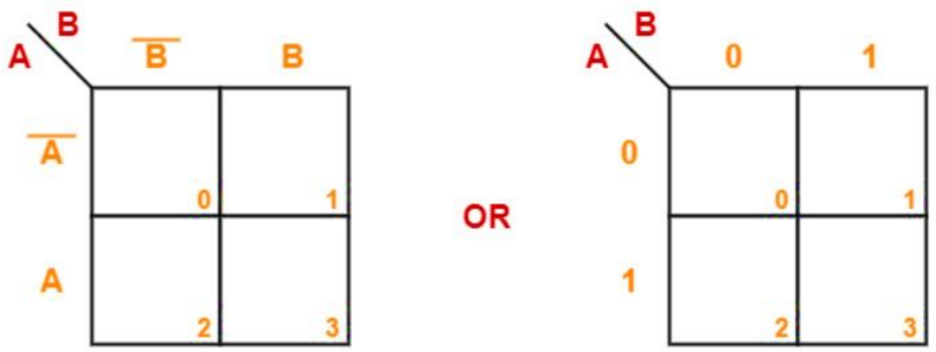
The Karnaugh Map also called as K Map is a graphical representation that provides a systematic method for simplifying the boolean expressions.

For a Boolean expression consisting of n-variables, number of cells required in K-Map = 2^n cells.

Two Variable K- Map

- Two variable K-Map is drawn for a Boolean expression consisting of two variables.
- The number of cells present in two variable K-Map = 2^2 = 4 cells.
- So, for a Boolean function consisting of two variables, we draw a (2 x 2) K-Map.

Two variable K-Map may be represented as-



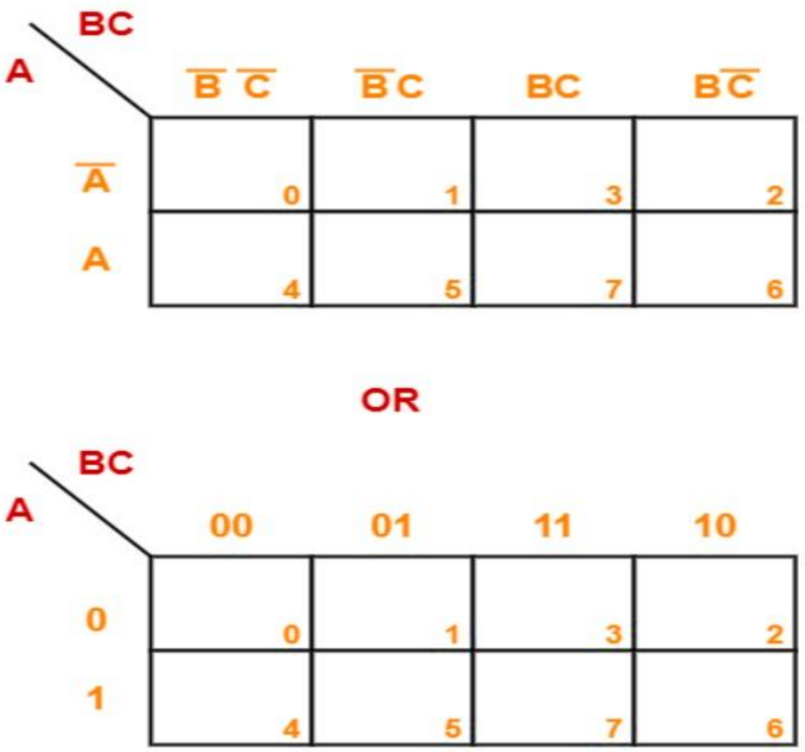
Two Variable K Map

Here, **A** and **B** are the *two variables* of the given Boolean function.

Three Variable K Map-

- Three variable K Map is drawn for a Boolean expression consisting of three variables.
- The number of cells present in three variable K-Map = 2^3 = 8 cells.
- So, for a Boolean function consisting of three variables, we draw a (2 x 4) K Map.

<u>**Three variable**</u> K-Map may be represented as-



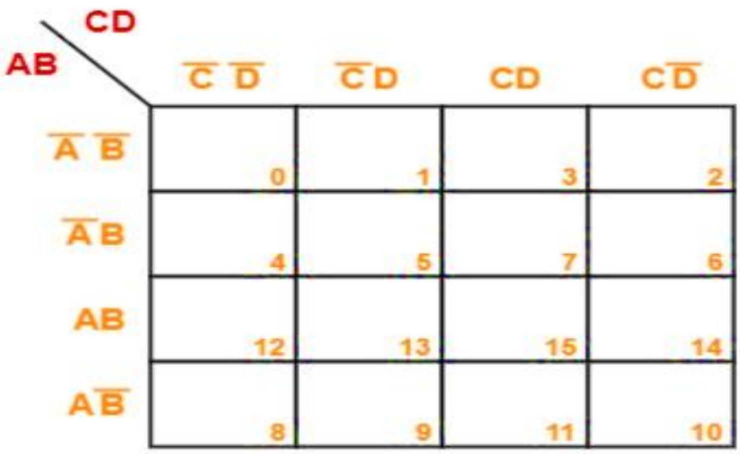
Three Variable K Map

Here, **A**, **B** and **C** are the three variables of the given Boolean function.

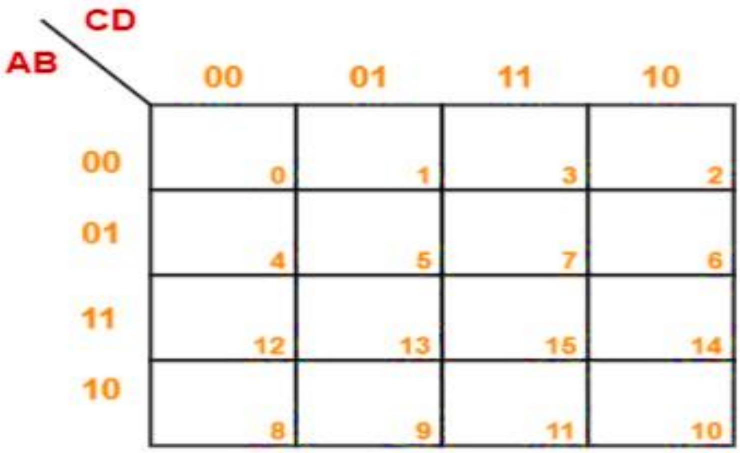
Four Variable K-Map-

- Four variable K-Map is drawn for a Boolean expression consisting of four variables.
- The number of cells present in four variable K-Map = 2^4 = 16 cells.
- So, for a boolean function consisting of four variables, we draw a (4 x 4) K-Map.

Four variable K-Map may be represented as-



OR



Four Variable K Map

Here, **A**, **B**, **C** and **D** are the four variables of the given Boolean function.

Karnaugh Map Simplification Rules-

To minimize the given Boolean function

- We draw a K Map according to the number of variables it contains.
- We fill the K Map with 0's and 1's according to its function.
- Then, we minimize the function in accordance with the following steps to find the minterm solution or K-map:

➢ **Step-1:**

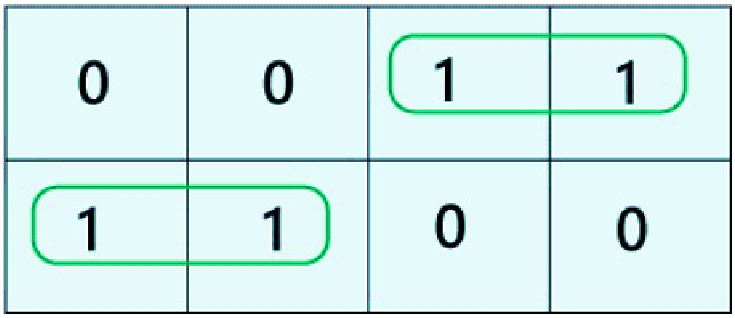
Firstly, we define the given expression in its canonical form.

➢ **Step-2:**

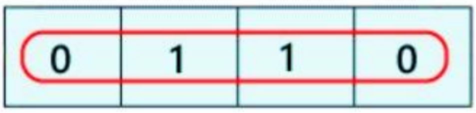
Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

➢ **Step-3:**

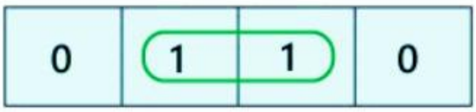
Next, we form the groups by considering each one in the K-map.



Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.



Incorrect



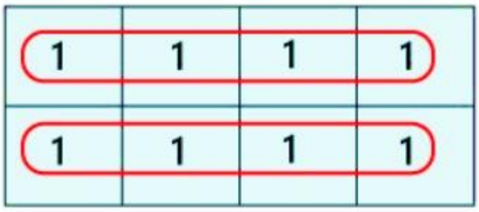
Correct

- We can only create a group whose number of cells can be represented in the power of 2.
- In other words, a group can only contain 2n i.e. 1, 2, 4, 8, 16 and so on number of cells.

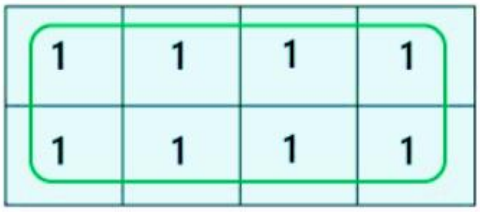
Incorrect

Correct

- We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.

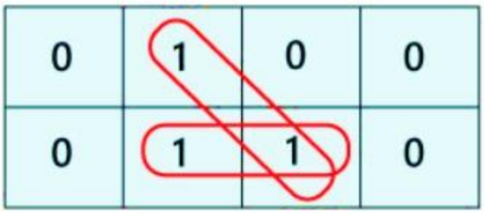


Incorrect

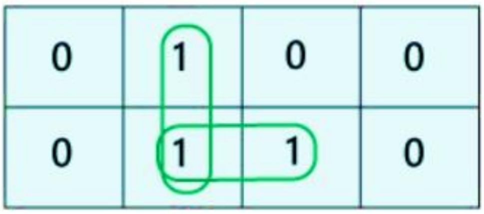


Correct

- Groups can be only either horizontal or vertical.
- We cannot create groups of diagonal or any other shape.

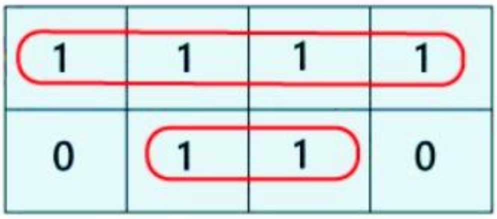


Incorrect

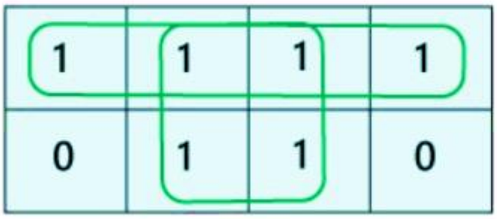


Correct

- The elements in one group can also be used in different groups only when the size of the group is increased. In other words, each group should be as large as possible.

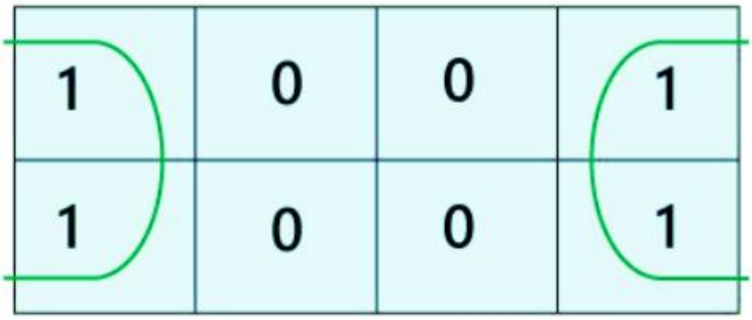


Incorrect

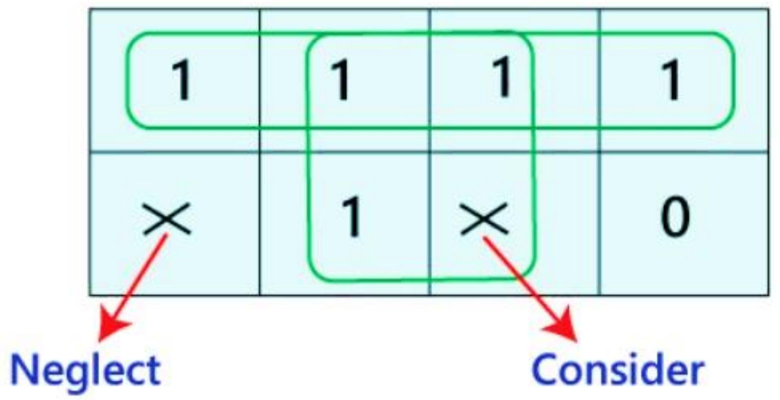


Correct

- The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.



- We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.

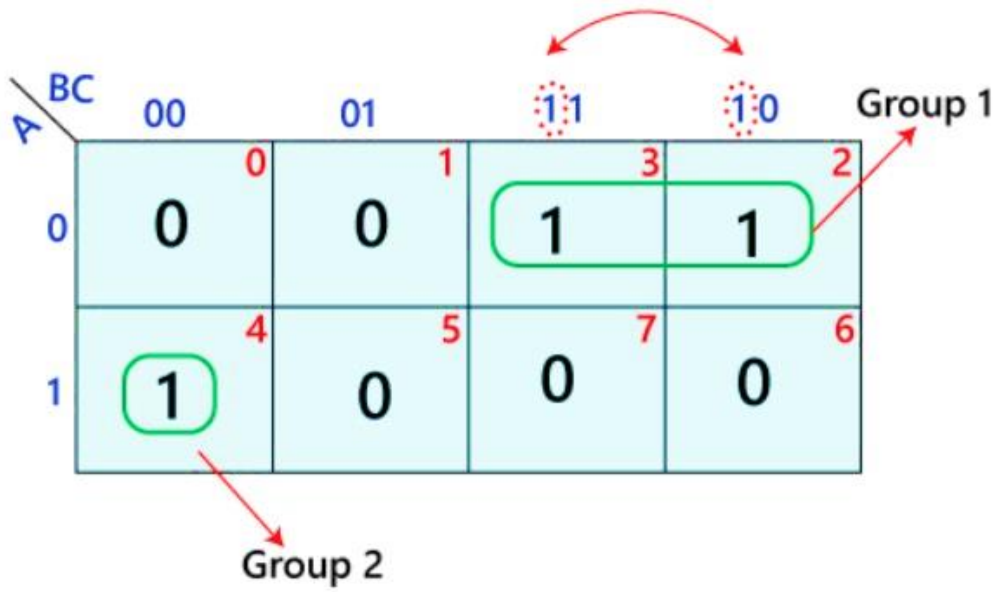


Step 4:

In the next step, we find the Boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of **two groups**, i.e., group 1 and group 2, with two and one number of 'ones'.

In the first group, the **ones** are present in **the row** for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only **B** term in common is the product term corresponding to the group as **(A'B)**.

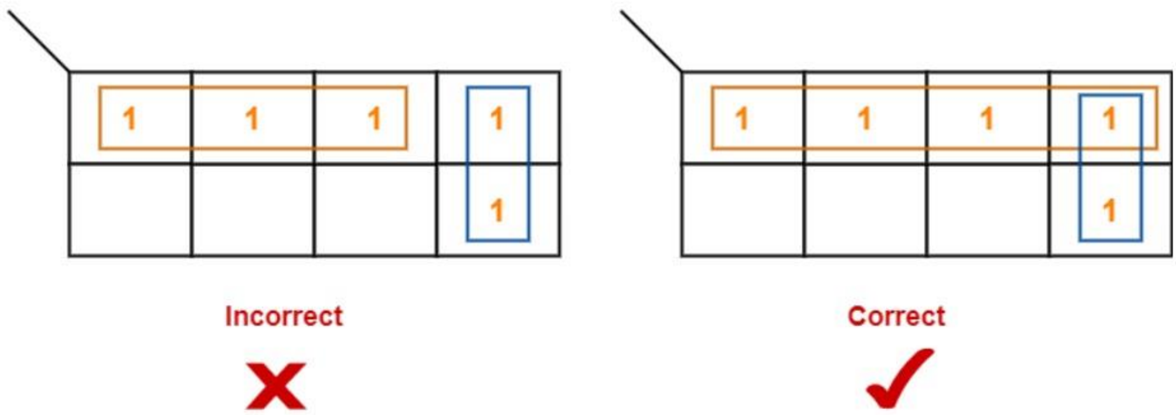
Just like group 1, **in group 2**, the one's are present in **a row** for which the value of A is 1. So, the corresponding variables of this column are **B'C'**. The overall product term of this group is(**AB'C'**).

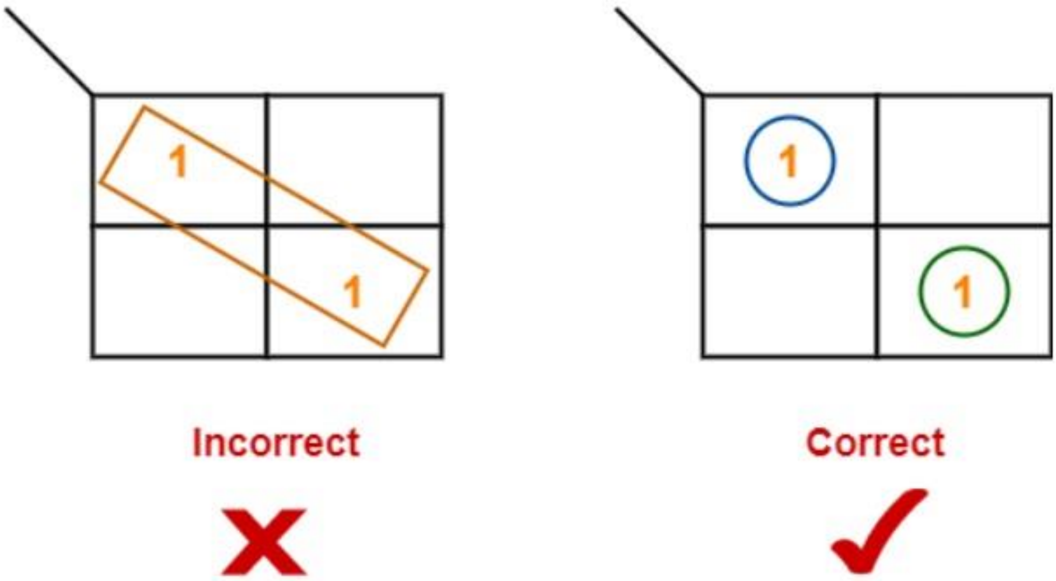


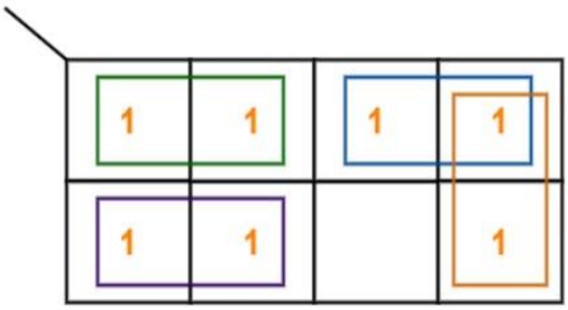
Step 5:

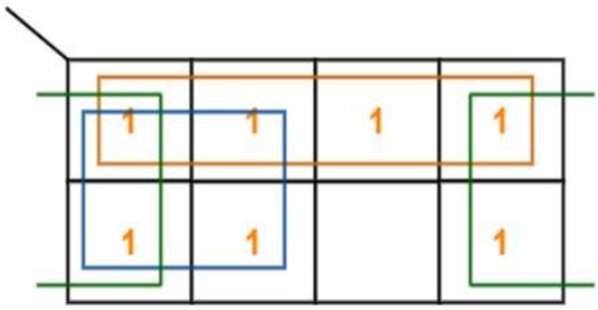
Lastly, we find the Boolean expression for the Output. To find the simplified Boolean expression in the SOP form, we combine the product-terms of **all** individual groups. So the simplified expression of the above k-map is as follows:

A'B+AB'C'



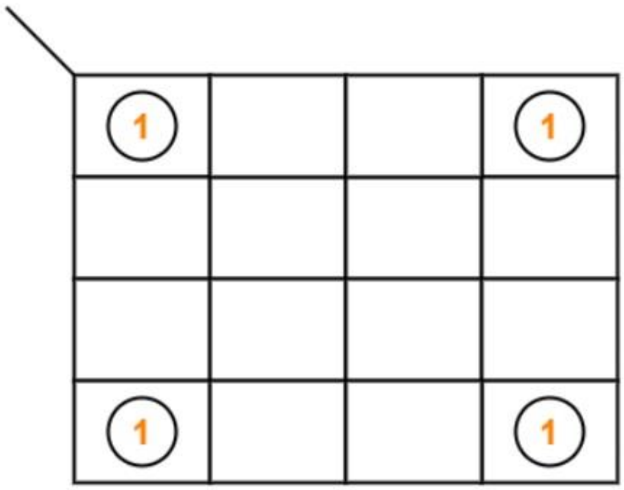


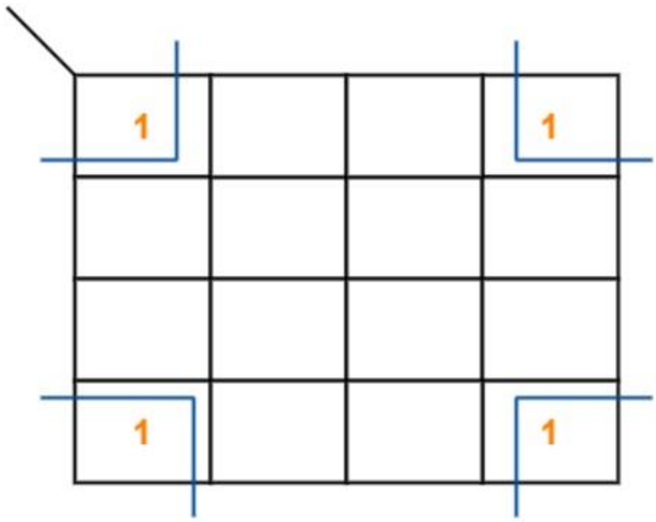




Incorrect

Correct





Incorrect

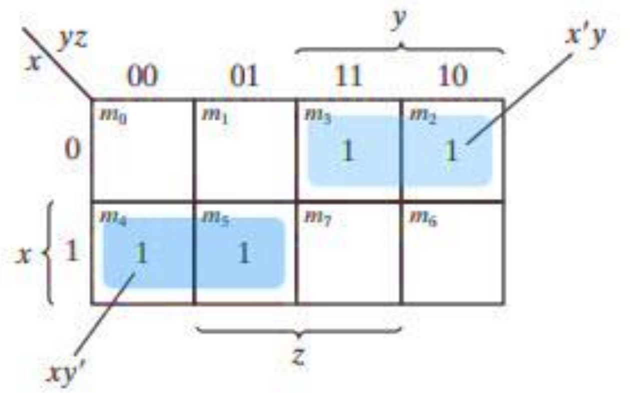
Correct

Let's take some examples.

Example 1:

Simplify the Boolean function:

F (x, y, z) = ∑(2, 3, 4, 5)

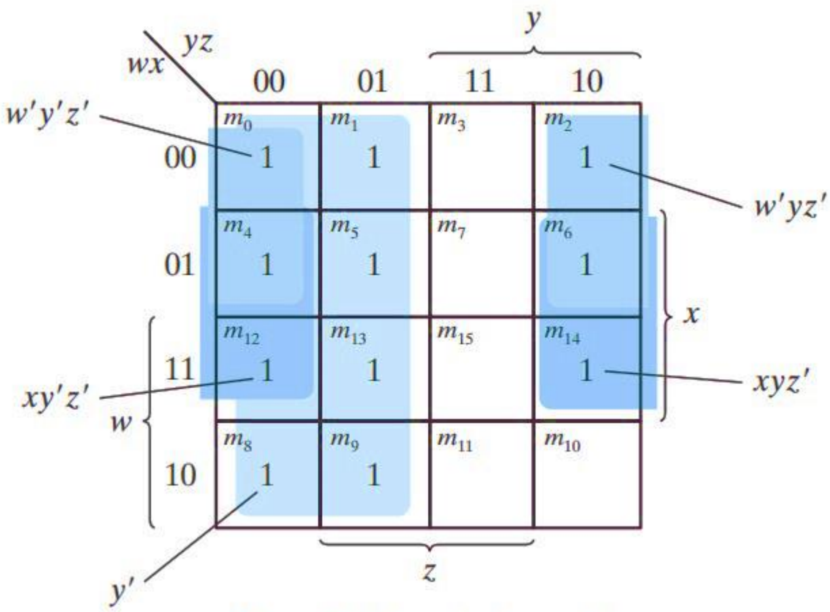


F (x, y, z) = ∑ (2, 3, 4, 5) = xy' + x'y

Example 2:

Simplify the Boolean function:

F (w, x, y, z) = ∑ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)



Note: $w'y'z' + w'yz' = w'z'$

$xy'z' + xyz' = xz'$

The simplified function is:

$F = y' + w'z' + xz'$

Map for Example 2, $F(w, x, y, z) = \sum(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

$= y' + w'z' + xz'$

Good luck

Dena.N

Adder

LEC-1/Course-2

First Class

2022

By: Assistant Lecturer
Dena Nameer

Digital Adder

In digital electronics an adder is a logic circuit that implements addition of numbers. In many computers and other types of processors, adders are used to calculate addresses, similar operations and table indices in the arithmetic logic unit (ALU) and also in other parts of the processors. These can be built for many numerical representations like binary coded decimal or excess-3.

- Adders are classified into two types:
  1. Half adder
  2. Full adder

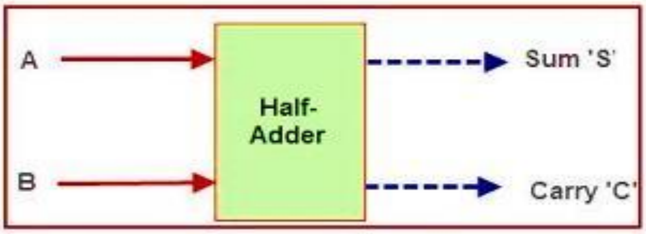
Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations. The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations:

- ✓ 0 + 0 = 0
- ✓ 0 + 1 = 1
- ✓ 1 + 0 = 1
- ✓ 1 + 1 = 10

The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a **carry** . When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits. *A combinational circuit that performs the addition of two bits is called a half adder* . *One that performs the addition of three bits (two significant bits and a previous carry) is a full adder* . The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

➢ Half Adder

From the verbal explanation of a half adder, we find that this circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The truth table for the half adder is listed below. The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum. The simplified Boolean functions for the two outputs can be obtained directly from the truth table.



Half Adder

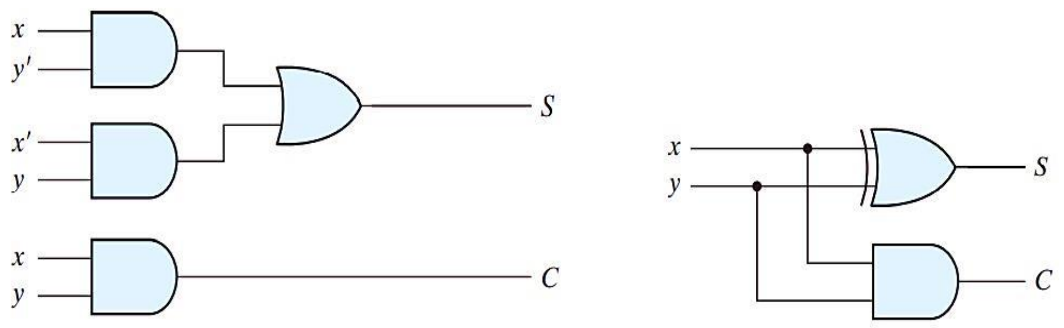
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The simplified sum-of-products expressions are

S = X'Y + XY'

C= XY

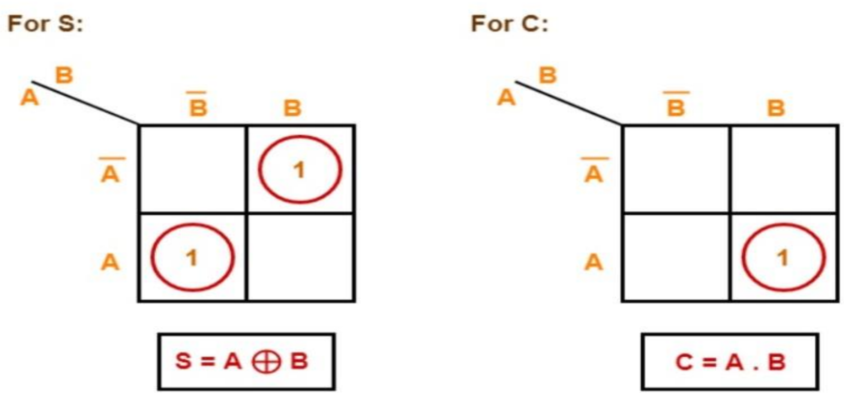
The logic diagram of the half adder implemented in sum of products is shown in Fig. 1-(a). It can be also implemented with an exclusive-OR and an AND gate as shown in Fig. 1-(b). This form is used to show that two half adders can be used to construct a full adder.



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

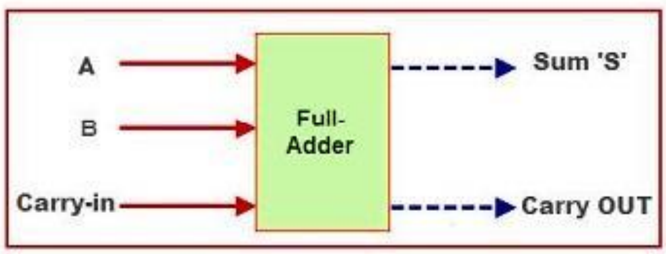
Figure-1



K Maps

➢ Full Adder

Addition of n-bit binary numbers requires the use of a full adder, and the process of addition proceeds on a bit-by-bit basis, right to left, beginning with the least significant bit. After the least significant bit, addition at each position adds not only the respective bits of the words, but must also consider a possible carry bit from addition at the previous position.



Full adder is difficult to implement than a half adder as it has three inputs. The first two inputs are A and B and the third input is an input carry as C-in. When full adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next. The output carry is designated as C OUT and the normal output is designated as S. The truth table of the full adder is listed in Table below:

Full Adder

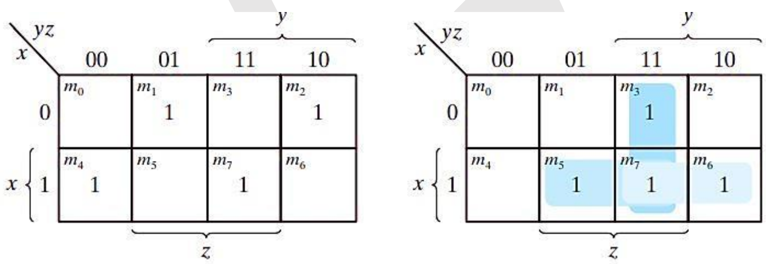
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

When all input bits are 0, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1. The input and output bits of the combinational circuit have different interpretations at various stages of the problem. On the one hand, physically, the binary signals of the inputs are considered binary digits to be added arithmetically to form a two-digit sum at the output. On the other hand, the same binary values are considered as variables of Boolean functions when expressed in the truth table or when the circuit is implemented with logic gates. The maps for the outputs of the full adder are shown in Fig.2. As well as the logic diagram for the full adder implemented in sum-of-products form is shown in Fig. 2

While the simplified expressions are

S= X'Y'Z + X'YZ' + XY'Z' + XYZ

C = XY + XZ + YZ



(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

K-Maps for full adder

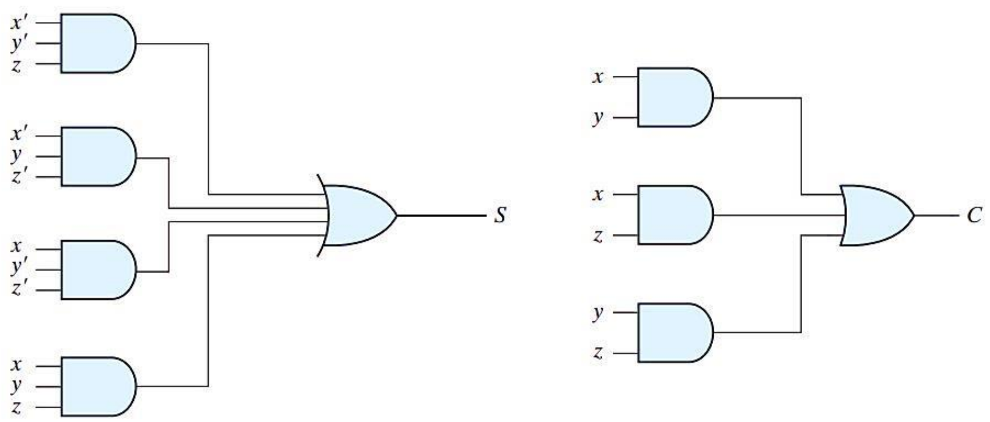
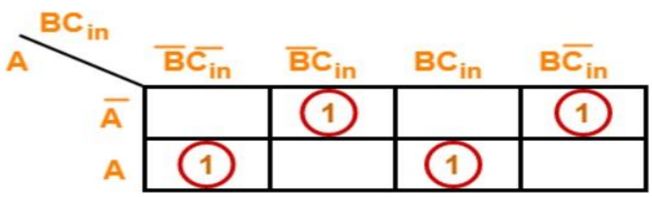


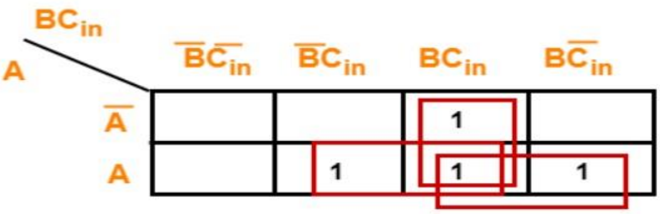
Figure - 2

For S:

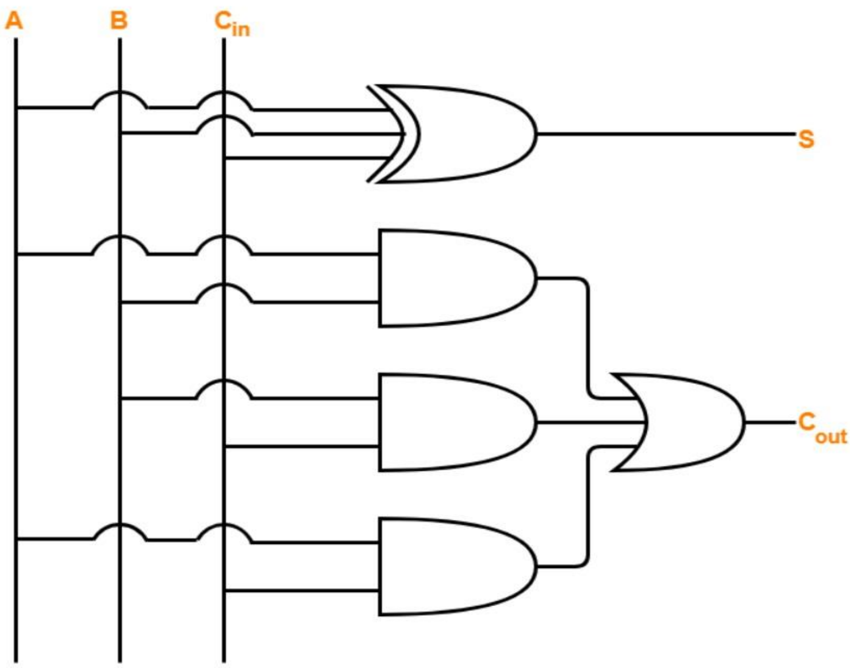


$$\boxed{S = A \oplus B \oplus C_{in}}$$

For C_{in}:



$$\boxed{C_{out} = AB + BC_{in} + C_{in}A}$$



Full Adder Logic Diagram

Good luck

Dena.N. ObedAghaZ

Ninevah University College of Electronics

Communication Dept.

Half Adder & Full Adder

LEC-1/Course-2

First Class

2022

By: Assistant Lecturer
Dena Nameer

Digital Adder

In digital electronics an adder is a logic circuit that implements addition of numbers. In many computers and other types of processors, adders are used to calculate addresses, similar operations and table indices in the arithmetic logic unit (ALU) and also in other parts of the processors. These can be built for many numerical representations like binary coded decimal or excess-3.

- Adders are classified into two types:
  1. Half adder
  2. Full adder

Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations. The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations:

- ✓ 0 + 0 = 0
- ✓ 0 + 1 = 1
- ✓ 1 + 0 = 1
- ✓ 1 + 1 = 10

The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a **carry** . When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits. *A combinational circuit that performs the addition of two bits is called a half adder* . *One that performs the addition of three bits (two significant bits and a previous carry) is a full adder* . The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

Half Adder

From the verbal explanation of a half adder, we find that this circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The truth table for the half adder is listed below. The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum. The simplified Boolean functions for the two outputs can be obtained directly from the truth table.

Half Adder

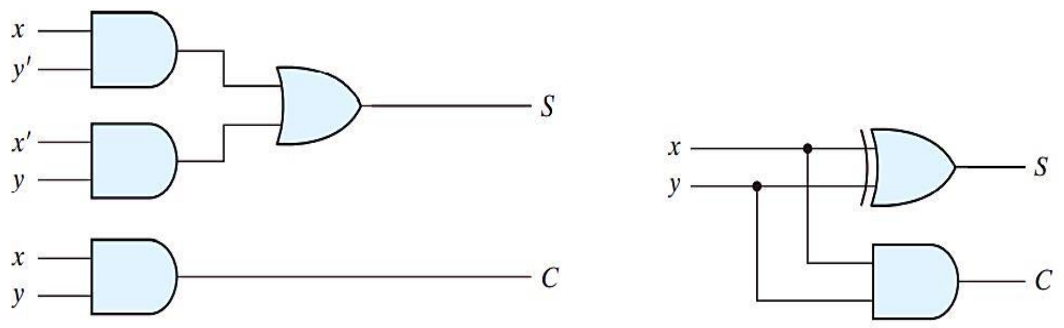
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The simplified sum-of-products expressions are

$$S = X'Y + XY'$$

$$C = XY$$

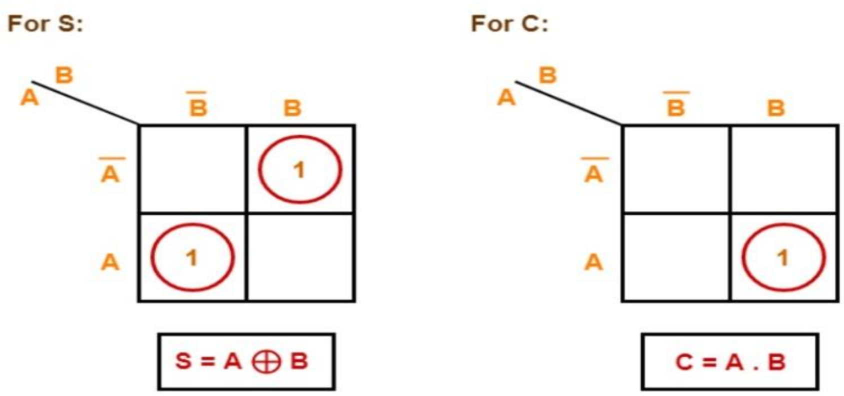
The logic diagram of the half adder implemented in sum of products is shown in Fig. 1-(a). It can be also implemented with an exclusive-OR and an AND gate as shown in Fig. 1-(b). This form is used to show that two half adders can be used to construct a full adder.



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

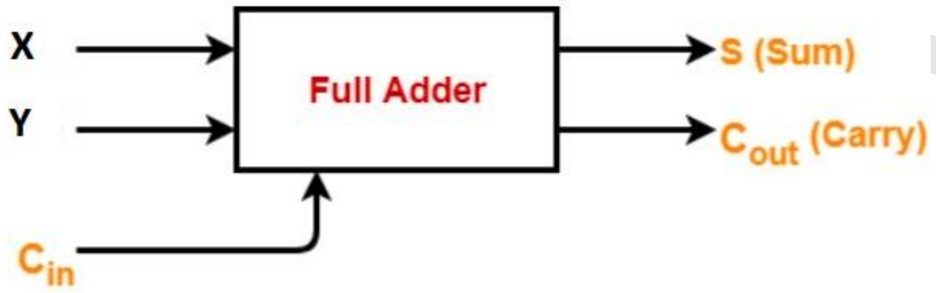
Figure-1



K Maps

➢ Full Adder

Addition of n-bit binary numbers requires the use of a full adder, and the process of addition proceeds on a bit-by-bit basis, right to left, beginning with the least significant bit. After the least significant bit, addition at each position adds not only the respective bits of the words, but must also consider a possible carry bit from addition at the previous position.



A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be added. The third input, z or (C_{in}), represents the carry from the previous lower significant position. *The output* carry is designated as **C_{OUT}** and the normal output is designated as **S**. The truth table of the full adder is listed in Table below:

Full Adder

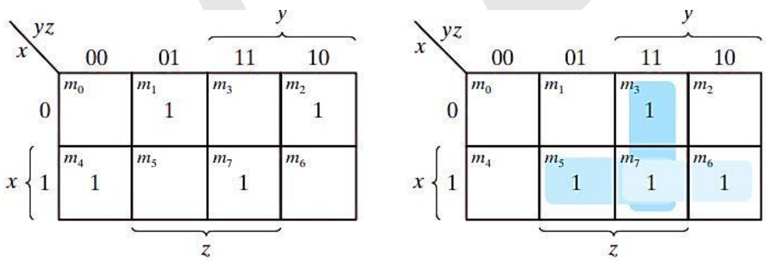
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

When all input bits are 0, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1. The input and output bits of the combinational circuit have different interpretations at various stages of the problem. On the one hand, physically, the binary signals of the inputs are considered binary digits to be added arithmetically to form a two-digit sum at the output. On the other hand, the same binary values are considered as variables of Boolean functions when expressed in the truth table or when the circuit is implemented with logic gates. The maps for the outputs of the full adder are shown in Fig.2. As well as the logic diagram for the full adder implemented in sum-of-products form is shown in Fig. 2

While the simplified expressions are

S= X'Y'Z + X'YZ' + XY'Z' + XYZ

C = XY + XZ + YZ



(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

K-Maps for full adder

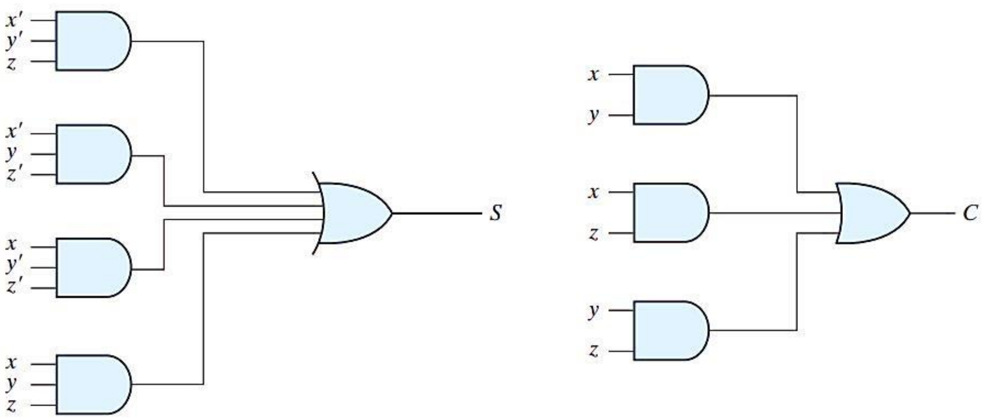


Figure - 2

If you used A,B and C_{in} as inputs instead of X,Y and Z.

For S:

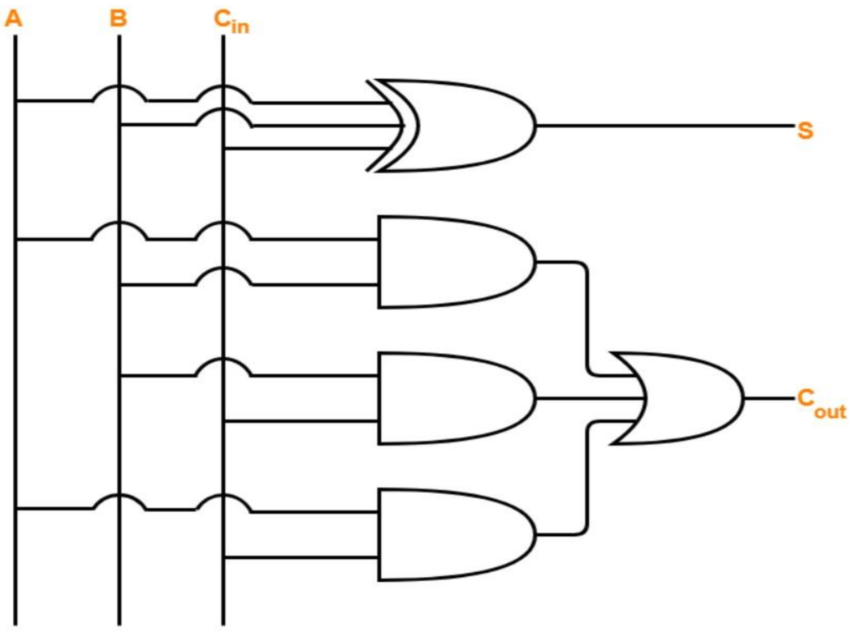
A \ BC_{in}	$\overline{B}\overline{C}_{in}$	$\overline{B}C_{in}$	BC_{in}	$B\overline{C}_{in}$
$\overline{A}$		1		1
A	1		1	

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

For C_{in}:

A \ BC_{in}	$\overline{B}\overline{C}_{in}$	$\overline{B}C_{in}$	BC_{in}	$B\overline{C}_{in}$
$\overline{A}$			1	
A		1	1	1

$$\boxed{C_{out} = AB + BC_{in} + C_{in}A}$$



Full Adder Logic Diagram

Good luck

Dena.N.ObedAghaZ

Encoder & Decoder

First Class

2022

By: Assistant Lecturer
Dena Nameer

Encoder and Decoder

Encoder:

An encoder is a digital function that performs the inverse of a decoder . It has (2^n) inputs and (n) outputs. The outputs lines generate the binary codes corresponding to the input values.

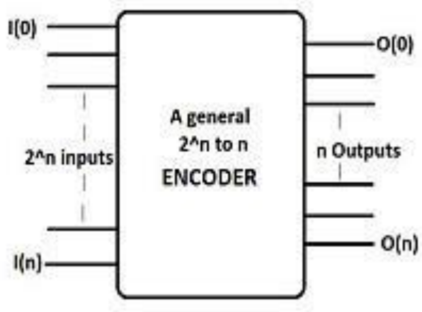
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

For this design it is assumed that only one input has a value of one at any given time.

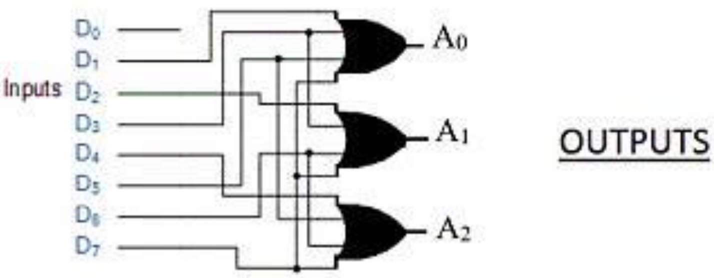
So $A_0=D_1+D_3+D_5+D_7$

$A_1= D_2+D_3+D_6+D_7$

$A_2= D_4+D_5+D_6+D_7$



A General encoder's block diagram.



Priority Encoder:

The operation of the priority encoder is such that if two or more inputs are equal to one at the same time, the input having the highest priority will take precedence.

D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Thus: $A_0 = D_3 + D_1 D_2'$

$D_3 D_2$ \ $D_1 D_0$	0 0	0 1	1 1	1 0
0 0	d	0	1	1
0 1	0	0	0	0
1 1	1	1	1	1
1 0	1	1	1	1

And the same for A_1 and V:

$A_1 = D_2 + D_3$

$V = D_0 + D_1 + D_2 + D_3$

DIGITAL TECHNIQUE

Decoder:

The decoder is a combinational circuit that converts binary information from (n) coded inputs to a maximum of (2^n) unique output.

3 to 8 Decoder

A_2 A_1 A_0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 0 0	0 0 0 0 0 0 0 1
0 0 1	0 0 0 0 0 0 1 0
0 1 0	0 0 0 0 0 1 0 0
0 1 1	0 0 0 0 1 0 0 0
1 0 0	0 0 0 1 0 0 0 0
1 0 1	0 0 1 0 0 0 0 0
1 1 0	0 1 0 0 0 0 0 0
1 1 1	1 0 0 0 0 0 0 0

$D_0=A_2'A_1'A_0'$

$D_1=A_2'A_1'A_0$

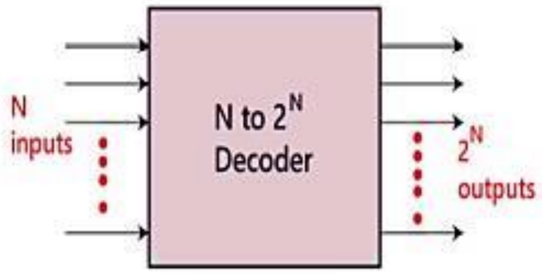
$D_3=A_2'A_1\,A_0$

$D_4=A_2\,A_1'A_0'$

$D_5=A_2\,A_1'A_0$

$D_6=A_2\,A_1\,A_0'$

$D_7=A_2\,A_1\,A_0$



A General decoder's block diagram

DIGITAL TECHNIQUE

Decoder with Enable:

3 to 8 Decoder with Enable

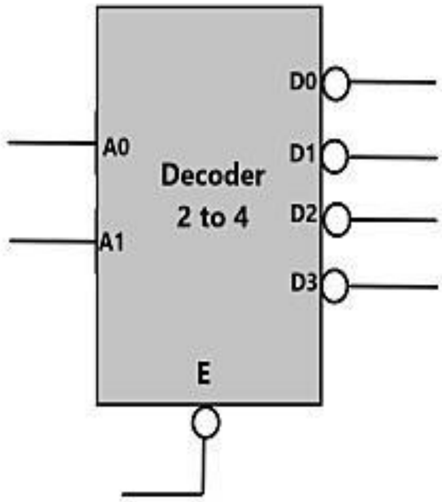
E	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	X	X	1	1	1	1

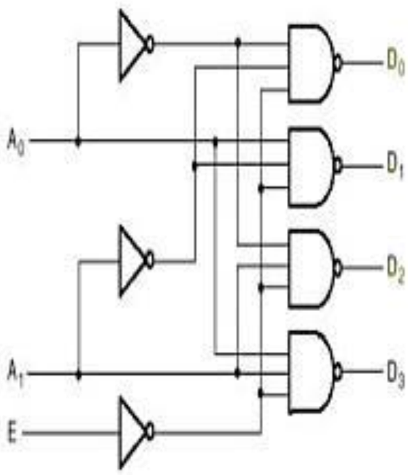
$D_0' = E'A_1'A_0'$

$D_1' = E'A_1'A_0$

$D_2' = E'A_1 A_0'$

$D_3' = E'A_1 A_0$





2 to 4 Decoder block diagram and internal logic circuit

Decoder applications

Example:

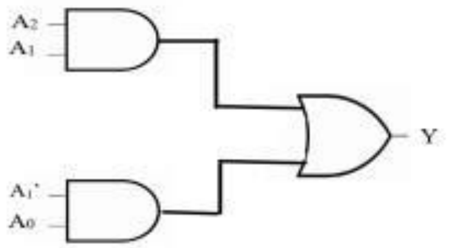
Find the output Y of the following canonical form:

$Y=\sum m\,(1,5,6,7)$

2^2 2^1 2^0 A_2 A_1 A_0	y
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

A_2 \ A_1A_0	0 0	0 1	1 1	1 0
0	0	1	0	0
1	0	1	1	1

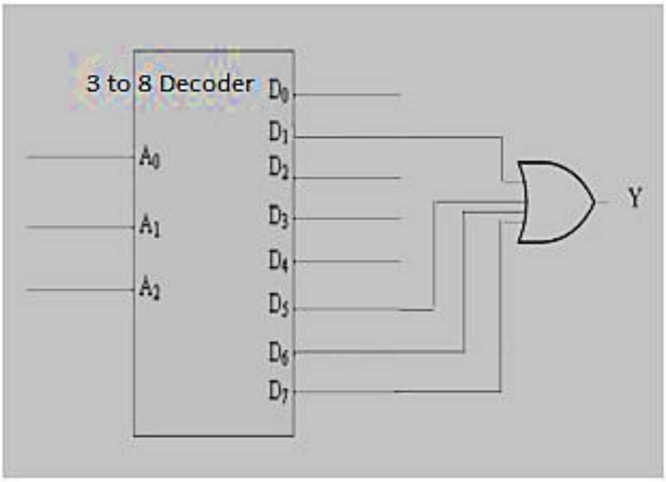
$Y=A_2A_1+A_1'A_0$



In order to solve the previous example by using a Decoder we need a 3 to 8 Decoder:

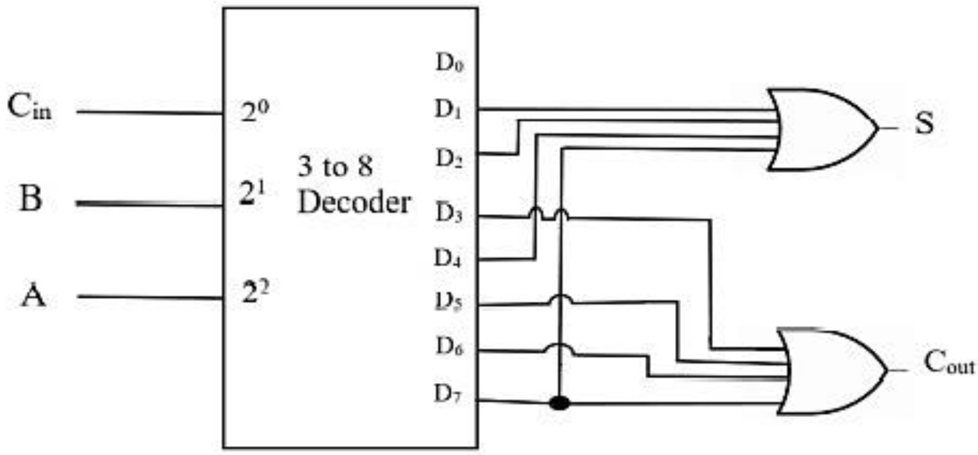
$Y=\sum m\,(1,5,6,7)$

DIGITAL TECHNIQUE



Full adder circuit using a Decoder:

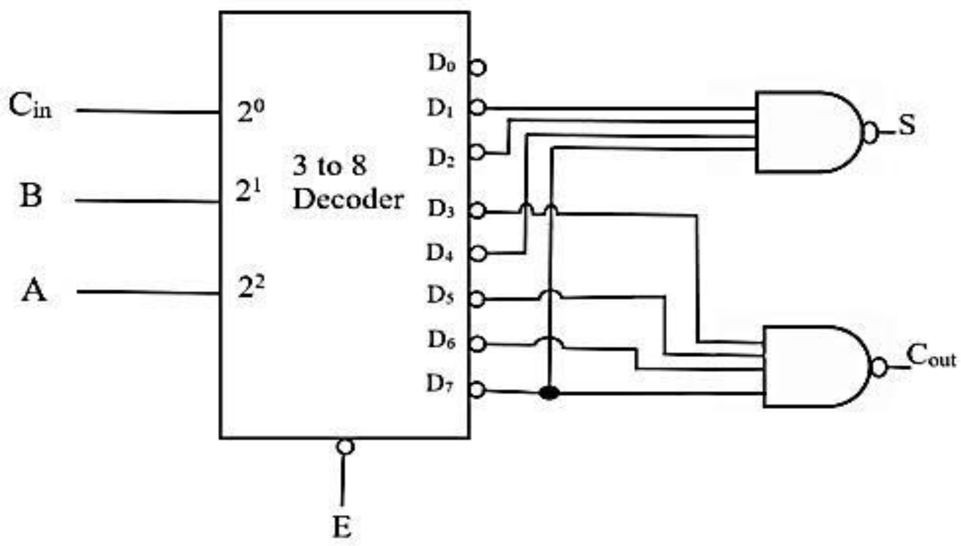
2^2 A	2^1 B	2^0 C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



DIGITAL TECHNIQUE

Design a F.A circuit using a Decoder with Enable:

A B C_{in}	C_{out} S
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1



Good luck

Dena.N

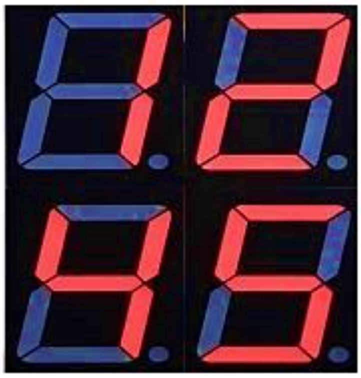
BCD to 7-Segment Decoder

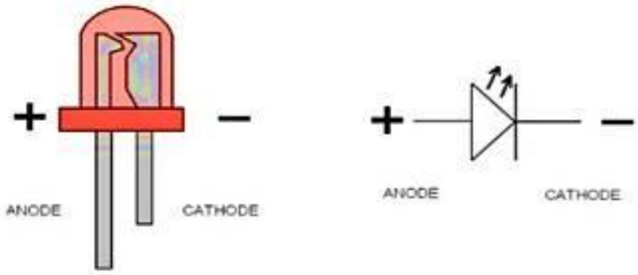
First Class

2022

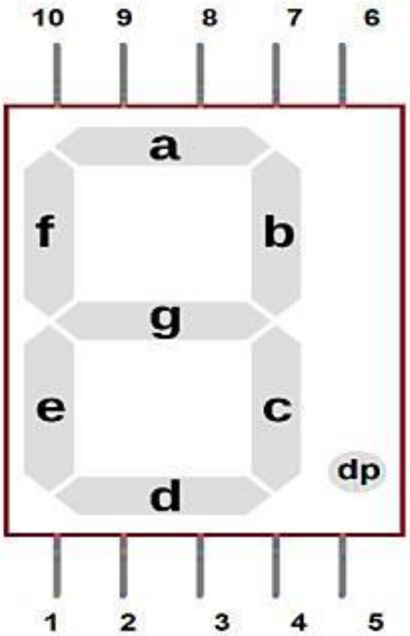
By: Assistant Lecturer
Dena Nameer

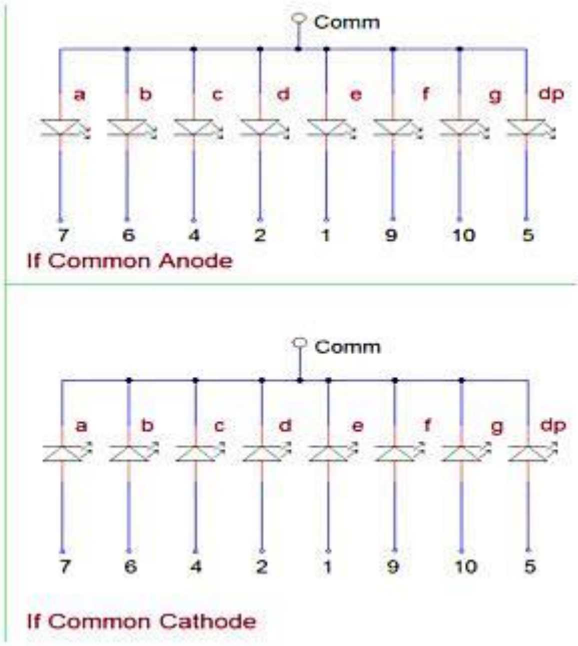
BCD to 7-Segment Decoder



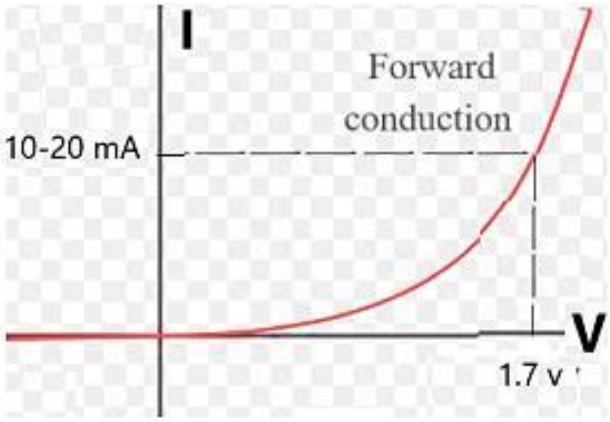


1.7 volt

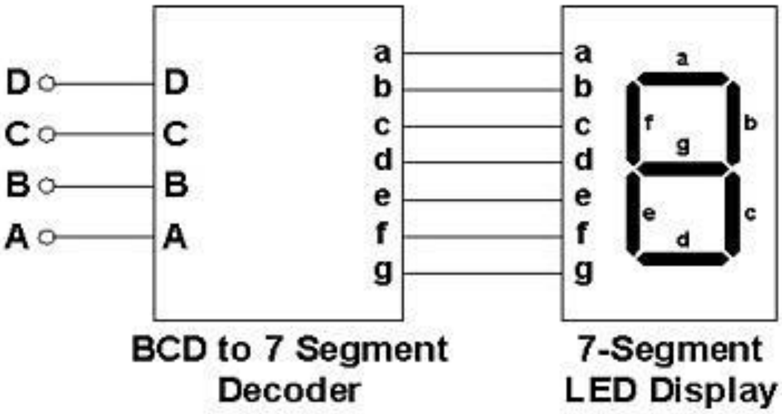


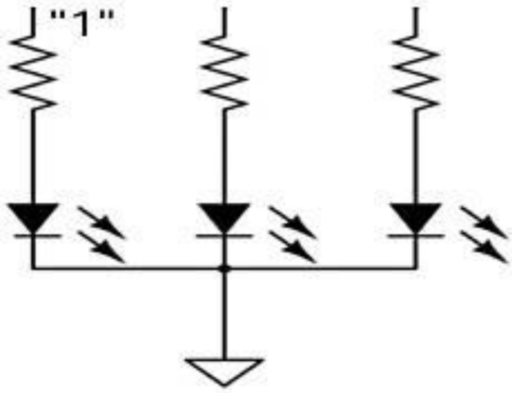


DIGITAL TECHNIQUE



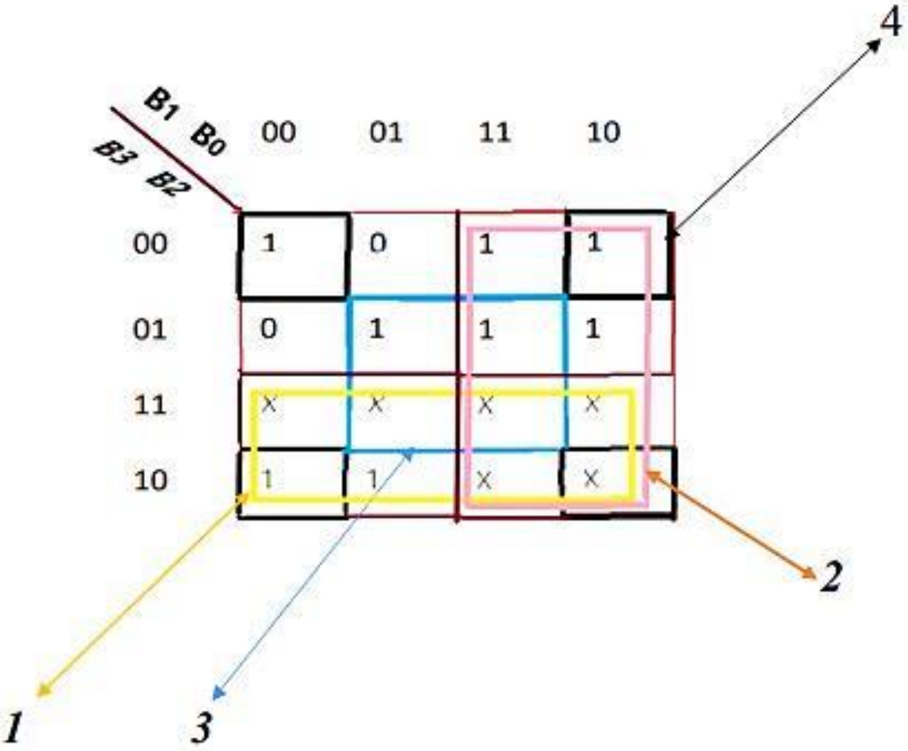
BCD to 7-Segment Decoder common cathode





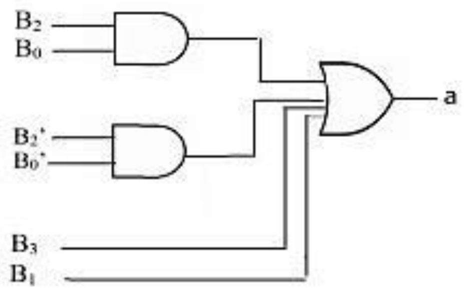
DIGITAL TECHNIQUE

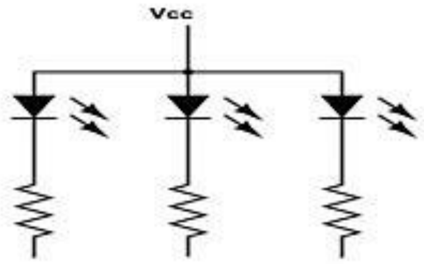
Decimal	BCD B_3	B_2	B_1	B_0	Decoded outputs a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
	1	0	1	0	X	X	X	X	X	X	X
	1	0	1	1	X	X	X	X	X	X	X
	1	1	0	0	X	X	X	X	X	X	X
	1	1	0	1	X	X	X	X	X	X	X
	1	1	1	0	X	X	X	X	X	X	X
	1	1	1	1	X	X	X	X	X	X	X



DIGITAL TECHNIQUE

$$a = B_3+B_1+B_2B_0+B_2'B_0'$$





Common Anode Circuit

Scale of Integration:

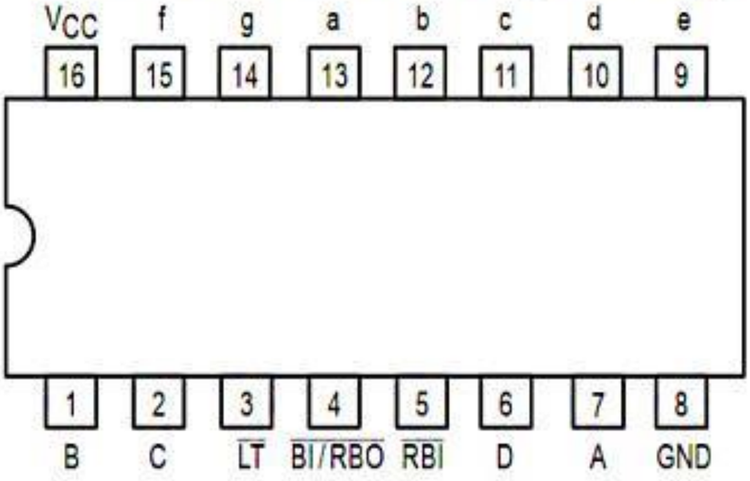
The number of components fitted into a standard size IC represents its integration scale, in other words it is a density of components. It is classified as follows:

1. SSI- Small Scale Integration (less than 10 gates)
2. MSI- Medium Scale Integration (10-100 gates)
3. LSI- Large Scale Integration (100-3000 gates)
4. VLSI- Very Large Scale Integration (≈ millions)

DIGITAL TECHNIQUE

BCD TO 7-SEGMENT DECODER

CONNECTION DIAGRAM DIP (TOP VIEW)



PIN NAMES

A, B, C, D	BCD Inputs
$\overline{RBI}$	Ripple-Blanking (Active Low) Input
$\overline{LT}$	Lamp-Test (Active Low) Input
$\overline{BI/RBO}$	Blanking Input or Ripple-Blanking Output (Active Low)
$\overline{BI}$	Blanking (Active Low) Input

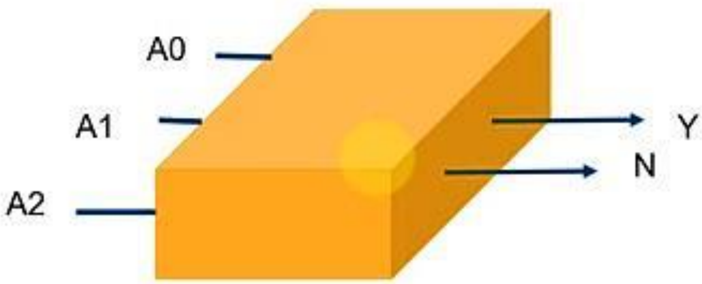
DIGITAL TECHNIQUE

Examples in digital circuit design

Design a logic circuit that check a binary number of 3 bit, if the binary number is more than five the output Y=1, while if it is less than 3 the output N=1. Otherwise both of them equal to 0.

How to design?

First of all you must imagine the system from outside



1st step is the truth table

A2	A1	A0	Y	N
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	0

You can write the canonical form for the system:

DIGITAL TECHNIQUE

Using SOP: $Y=\sum_m(6,7)$, $N=\sum_m(0,1,2)$

2^{nd} step is to simplify the circuit using K-maps:

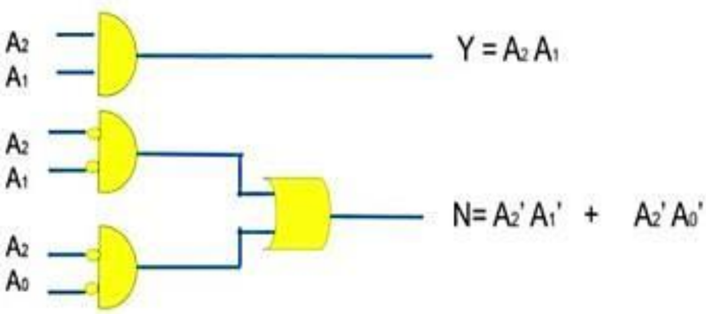
A_2 \ $A_1 A_0$	00	01	11	10
0				
1			1	1

$Y=A_2 A_1$

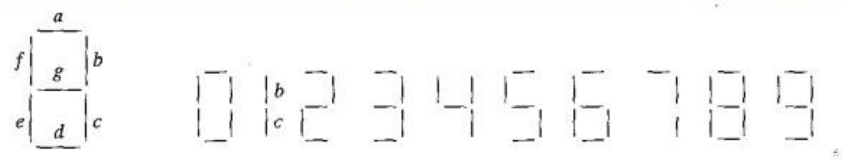
A_2 \ $A_1 A_0$	00	01	11	10
0	1	1		1
1				

$N= A_2' A_1' + A_2' A_0'$

Logic circuit:



Problem:
A BCD-to-seven-segment decoder is a combinational circuit that converts a decimal digit in BCD to an appropriate code for the selection of segments in a display indicator used for displaying the decimal digit in a familiar form. The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the display as shown in Fig. P4-9(a). The numeric display chosen to represent the decimal digit is shown Fig. P4-9(b). Design the BCD-to-seven-segment decoder using a minimum number of gates. The six invalid combinations should result in a blank display.



(a) Segment designation (b) Numerical designation for display

Solution:

Design procedure:

1. Derive the truth table that defines the required relationship between inputs and outputs.

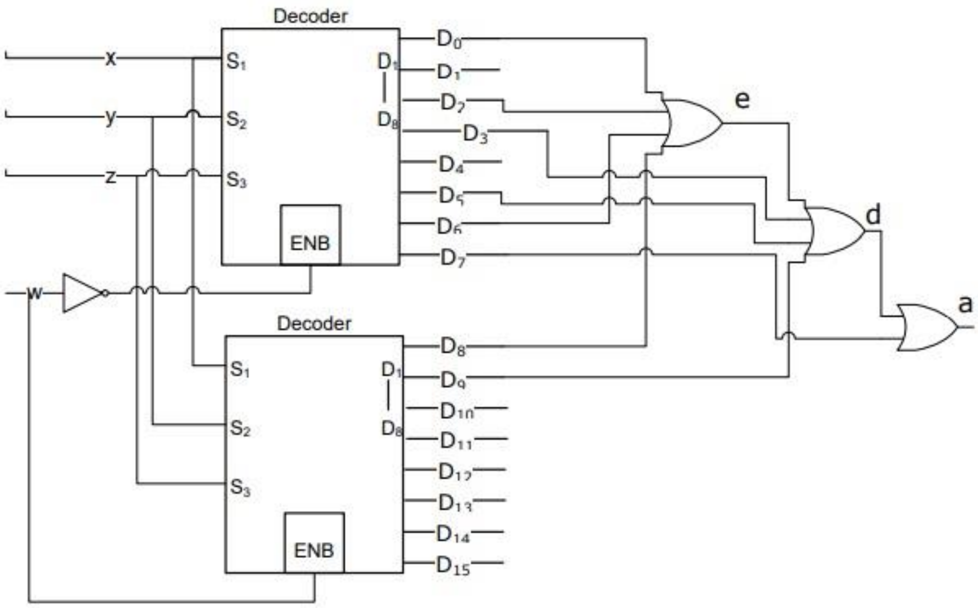
w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	X	0	0	0	0	0	0	0
1	1	X	X	0	0	0	0	0	0	0

2. Express the Boolean expressions for the outputs (a-g) in sum of minterms

$a(w,x,y,z)=\Sigma(0,2,3,5,6,7,8,9)$
$b(w,x,y,z)=\Sigma(0,1,2,3,4,7,8,9)$
$c(w,x,y,z)=\Sigma\ (0,1,3,4,5,6,7,8,9)$
$d(w,x,y,z)=\Sigma(0,2,3,5,6,8,9)$
$e(w,x,y,z)=\Sigma(0,2,6,8)$
$f(w,x,y,z)=\Sigma(0,4,5,6,8,9)$
$g(w,x,y,z)=\Sigma(2,3,4,5,6,7,8,9)$

3. Draw the logic circuit. Two 3-to-8-line decoders with enable inputs have been connected to form a 4-to-16-line decoder. Together they generate all the minterms of the input variables. OR gates are to be used to implement each of the functions a-g. The inputs to each OR gate are selected from the decoder outputs according to the list of minterm of each function.

The diagram below shows the circuit for output a, d and e. The same procedure should be followed to include the remaining functions and complete the logic circuit.



Good luck *Dena.N*

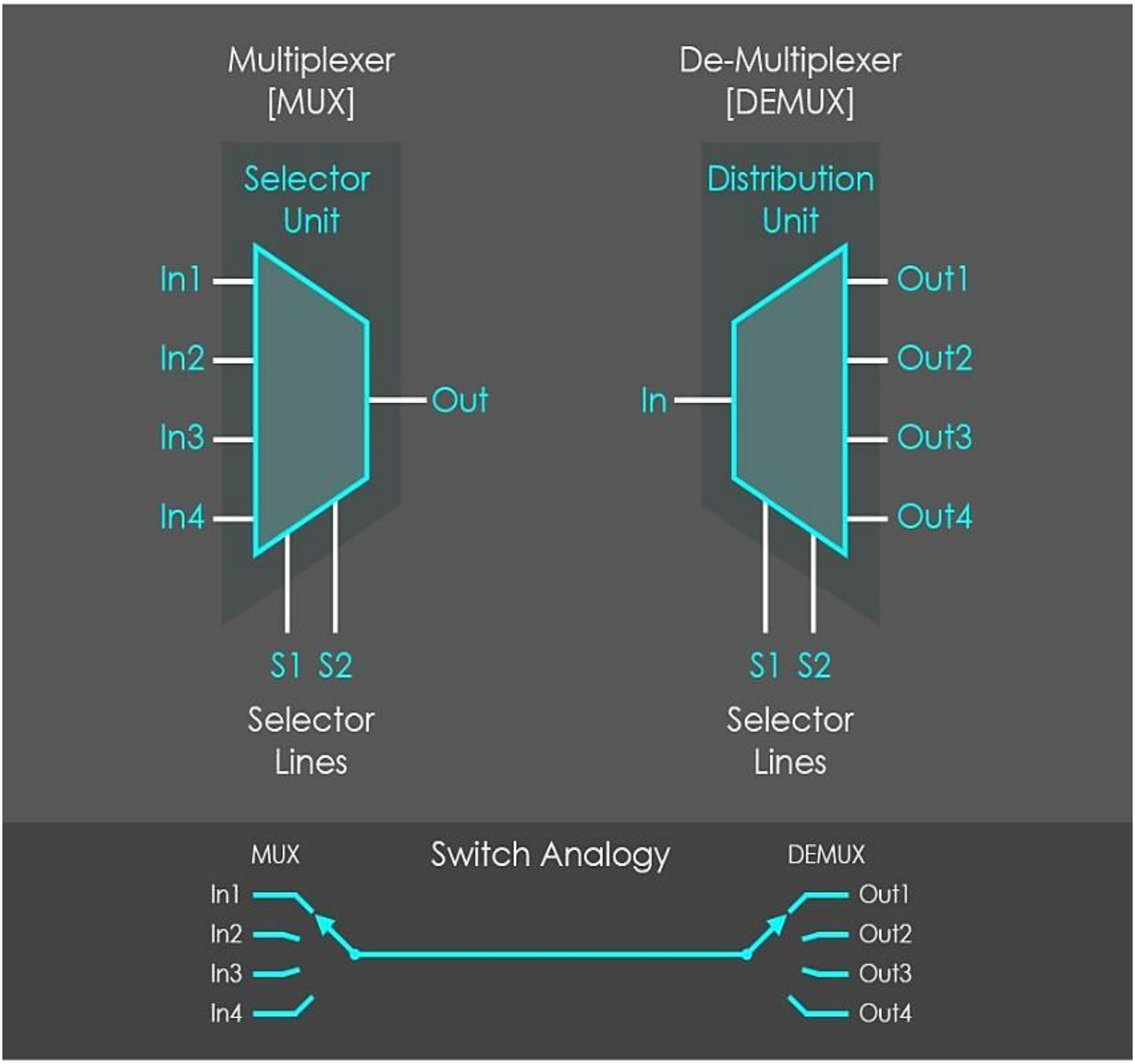
Multiplexer & Demultiplexer

First Class

2022

By: Assistant Lecturer
Dena Nameer

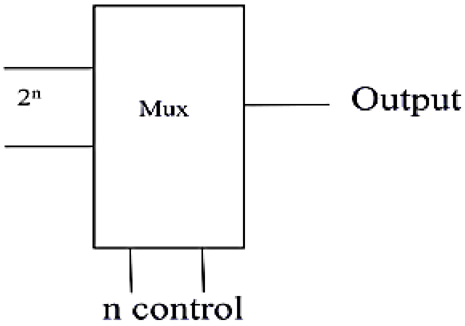
Multiplexer and De-Multiplexer

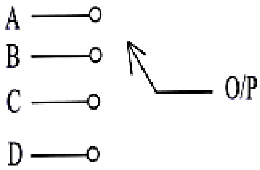


The Multiplexer

The multiplexer is a combinational logic circuit designed to switch one of several input lines to a single common output line.

Multiplexer (Mux):



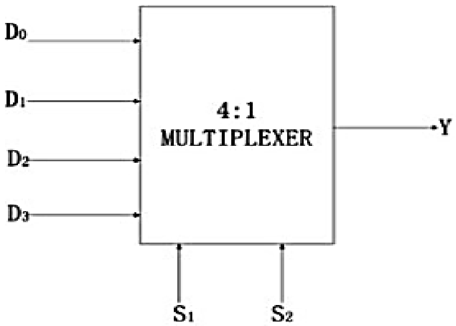


Mux 4 to 1:

The Mux has 2^n inputs , n control and one output

4 inputs = 2^2 , n=2 (S_0,S_1) select lines (control).

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



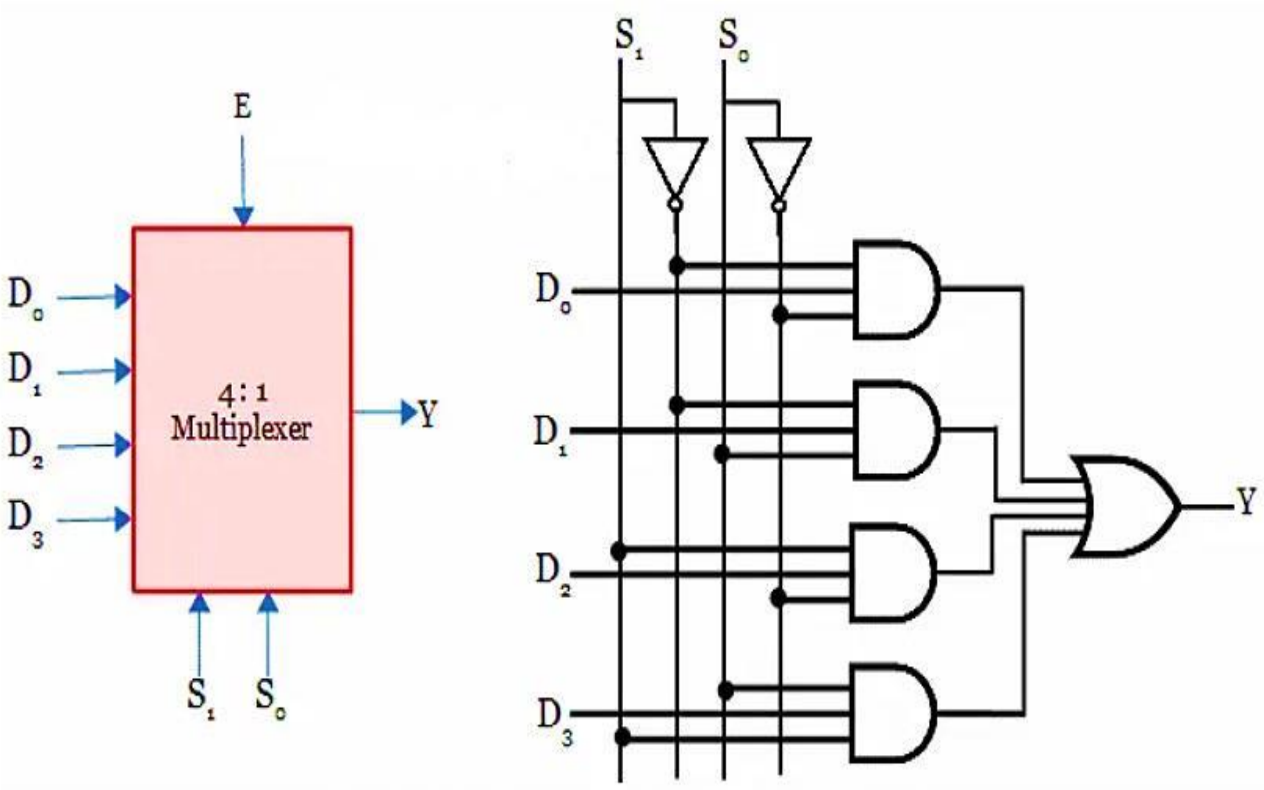
(a)

INPUT		OUTPUT
S_2	S_1	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

(b)

Figure-1

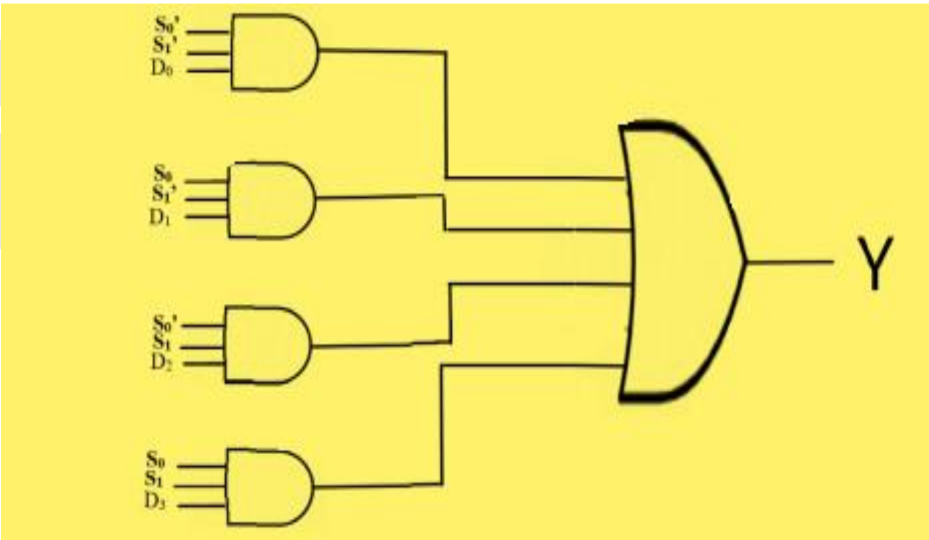
The 4:1 multiplexer block diagram and truth table



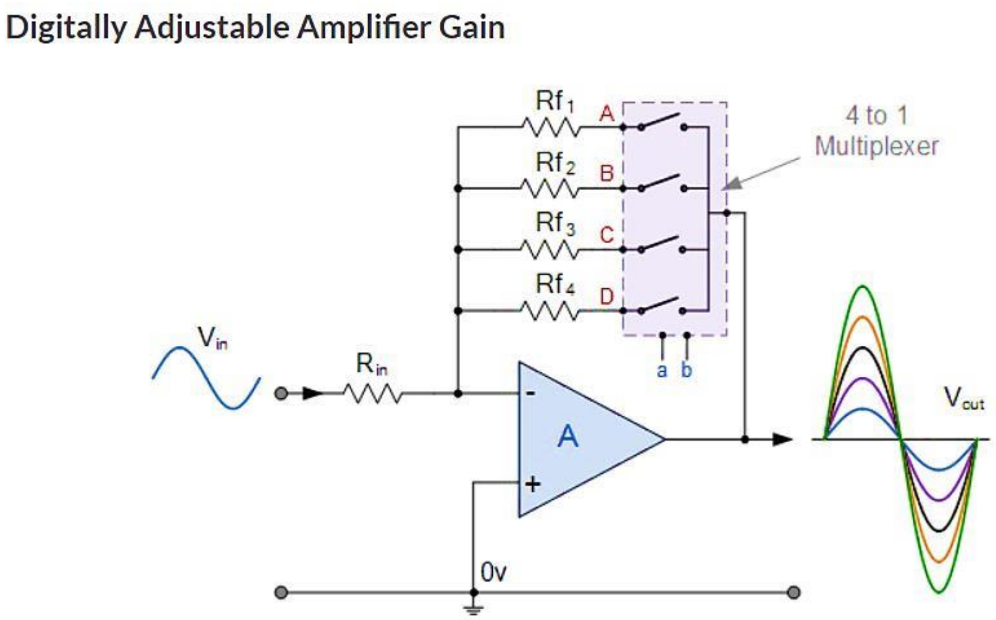
Block diagram and logic circuit of 4 : 1 mux

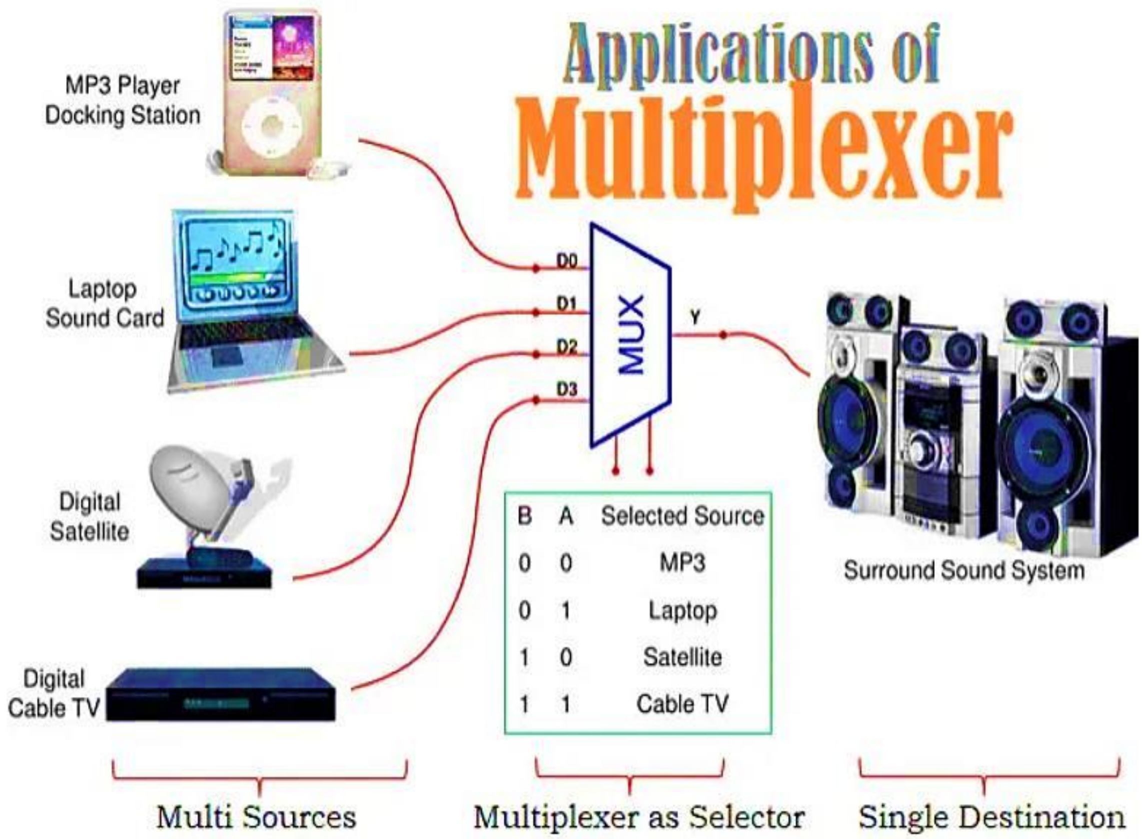
For selection inputs, S_1S_0 = 00, first AND gate alone is enabled and the output produced is

$$Y = D_0\overline{S_1}\,\overline{S_0}$$



For example:





Application of Multiplexer

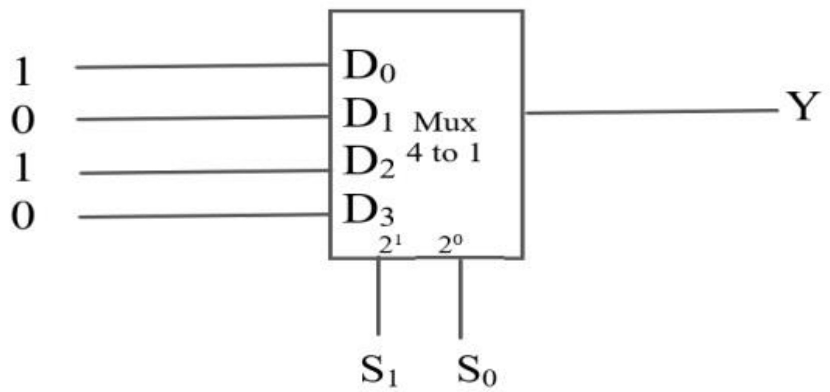
Example:

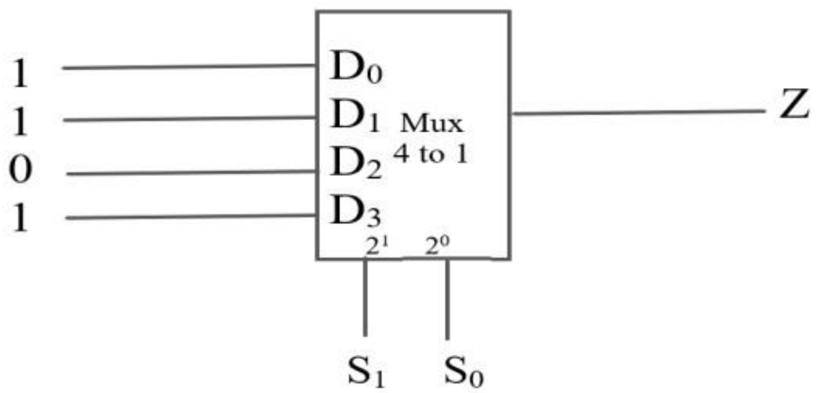
Design a logical circuit for the following canonical forms using multiplexers:

$Y=\sum_m (0,2)$

$Z=\sum_m (0,1,3)$

S_1	S_0	Z	Y
0	0	1	1
0	1	1	0
1	0	0	1
1	1	1	0

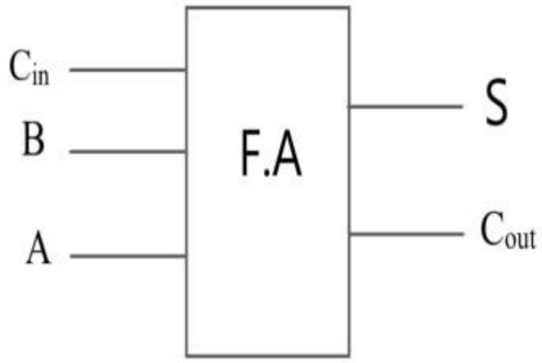


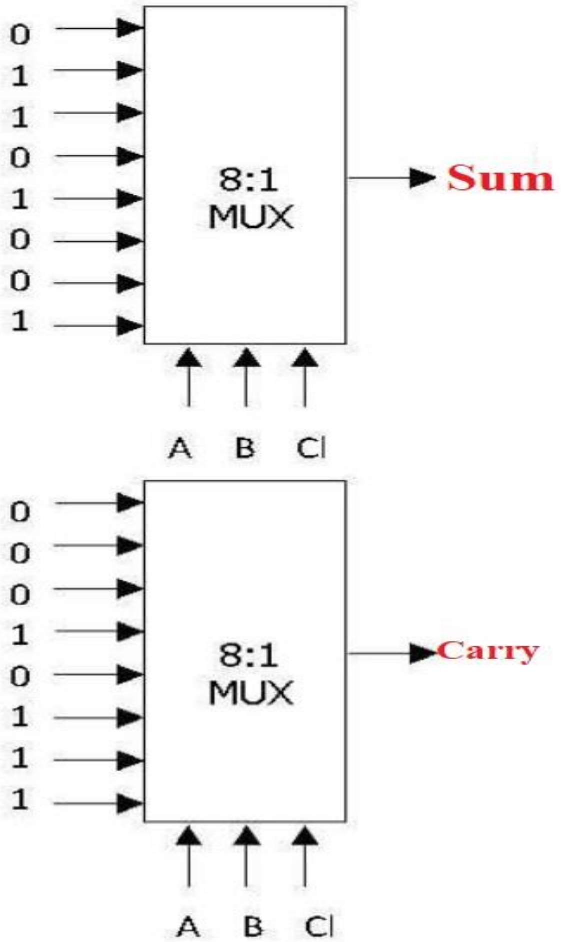


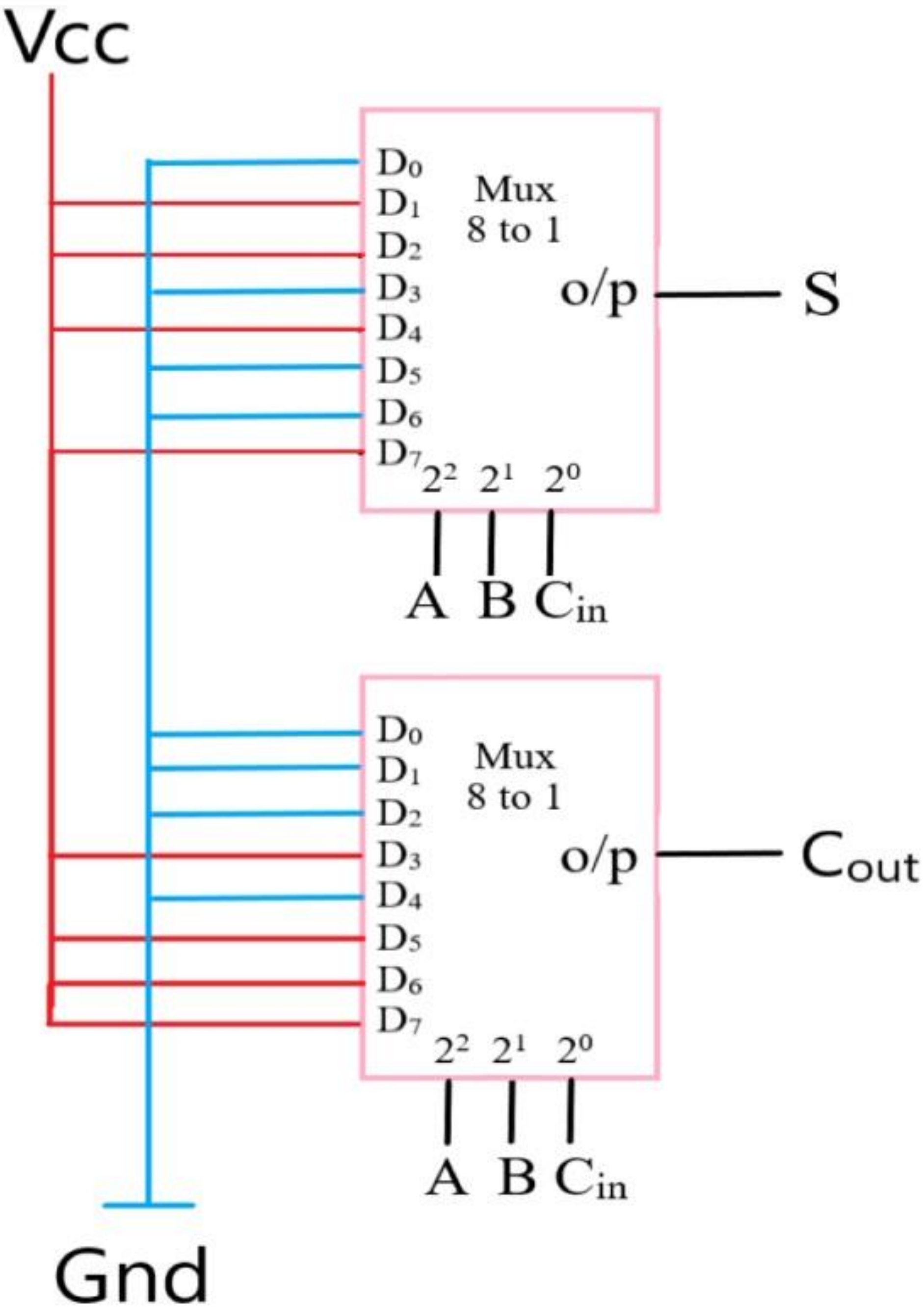
Example-2

Multiplexer can be used to design the Full Adder circuit:

A B C_{in}	C_{out} S
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1





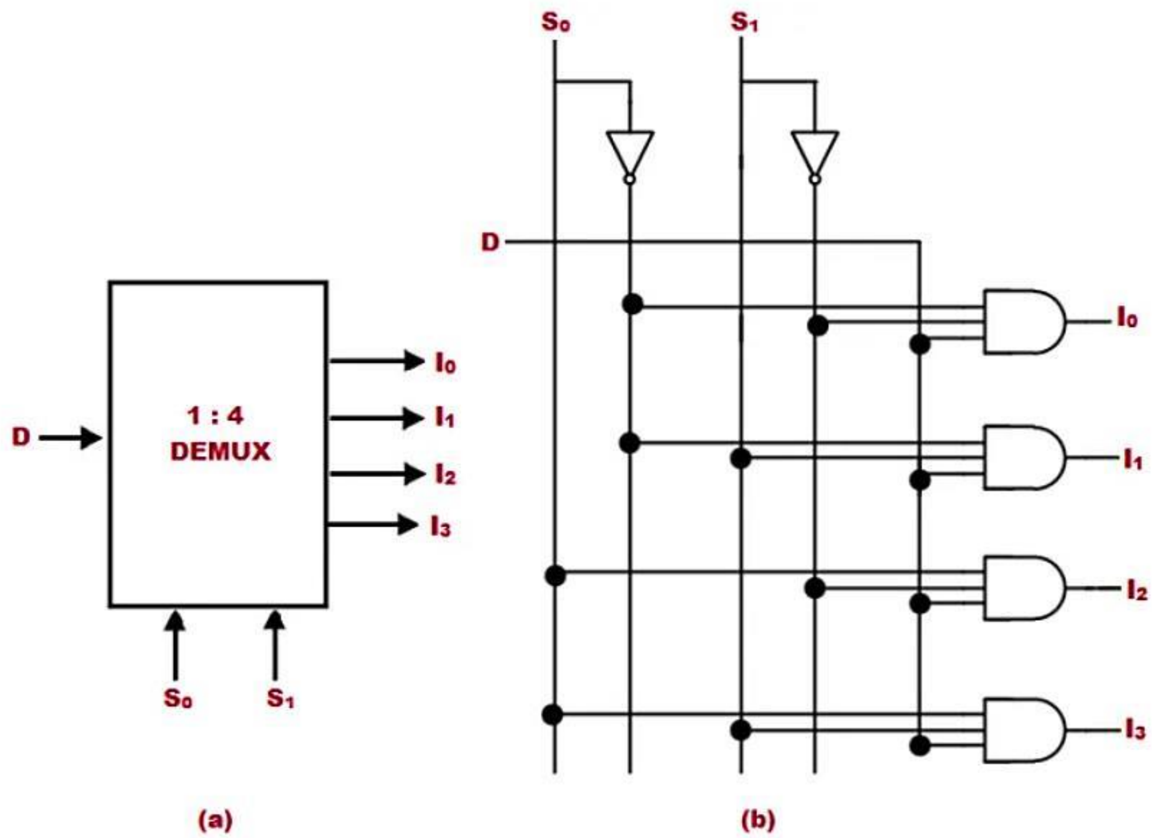

Vcc
D0
D1
D2
D3
D4
D5
D6
D7
Mux
8 to 1
o/p
S
2^2 2^1 2^0
A B Cin
D0
D1
D2
D3
D4
D5
D6
D7
Mux
8 to 1
o/p
Cout
2^2 2^1 2^0
A B Cin
Gnd

The Demultiplexer

Demultiplexer is a combinational circuit that accepts multiplexed data and distributes over multiple output lines. In other words, the function of Demultiplexer is the inverse of the multiplexing operation. Similar to Multiplexer, the output depends on the control input.

The 1:4 Demux consists of 1 data input bit, 2 control bits and 4 output bits. D is the input bit, I_0, I_1, I_2, I_3 are the four output bits and S_0 and S_1 are the control bits.

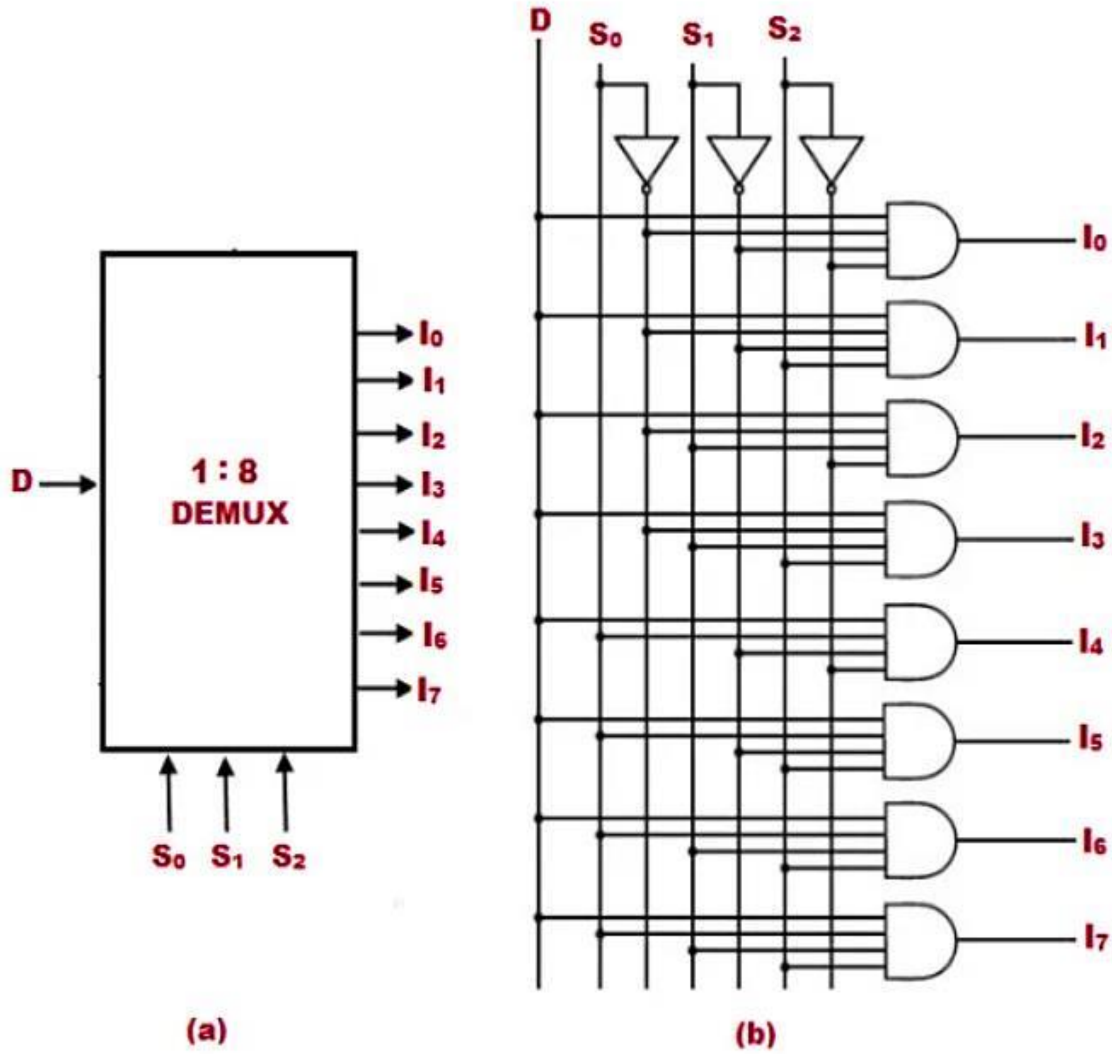
Figure below illustrates the block diagram and circuit diagram of 1:4 Demux.



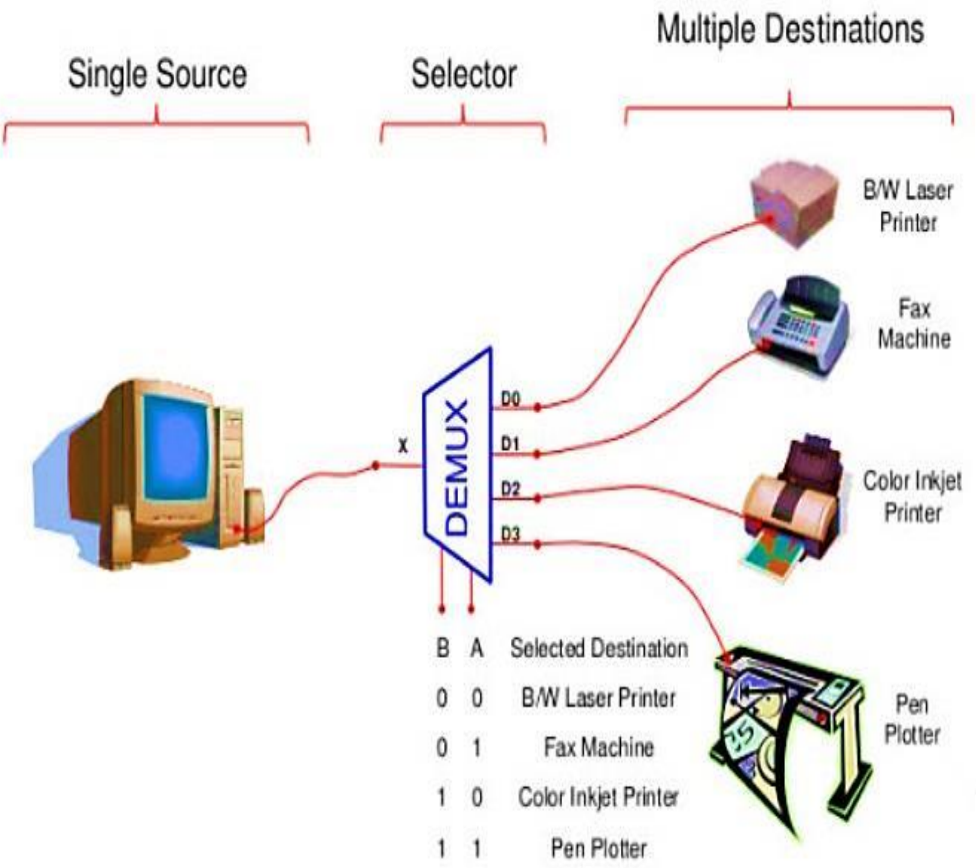
(a) Block Diagram of 1:4 Demux (b) Circuit Diagram of 1:4 Demux using Logic Gates

The 1:8 Demux consists of 1 data input bit, 3 control bits and 8 output bits. I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7 are the eight output bits, S_0, S_1 and S_2 are the control bits and input D.

Figure below illustrates the block diagram and circuit diagram of 1:8 Demux.



(a) Block Diagram of 1:8 Demux (b) Circuit Diagram of 1:8 Demux using Logic Gates



Applications of Demux

Good luck

Dena.N

Error-Detecting Code

- To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
- A parity bit is an extra bit included with a message to make the total number of 1's either even or odd.

- Example:
  - Consider the following two characters and their even and odd parity:

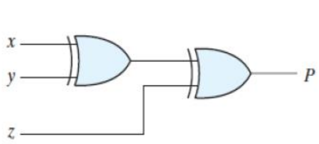
	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

Parity bit generation and checking

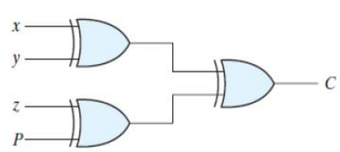
❑ Even parity generation and checking

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



(a) 3-bit even parity generator



(b) 4-bit even parity checker

The circuit that generates the parity bit in the transmitter is called a parity generator. The circuit that checks the parity in the receiver is called a parity checker

❑ H.W 1: Draw the 7-bit even parity generation and checking circuit.

Digital Circuits

- *Digital circuits are two types*

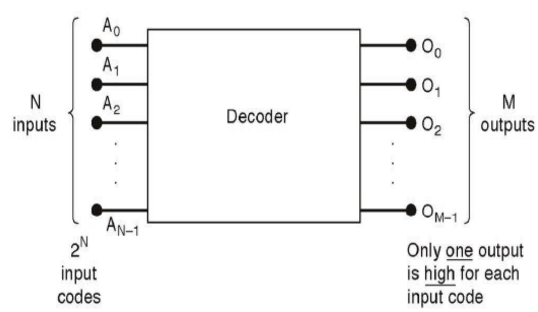
1. *Combinational circuit* **consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.**

2. *Sequential Circuits* **employ memory elements in addition to logic gates. Their outputs are a function of the inputs and the state of the memory elements.**

Decoder

Decoder: A decoder is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to the input number.

- A decoder has **N** inputs and **2^N** outputs.

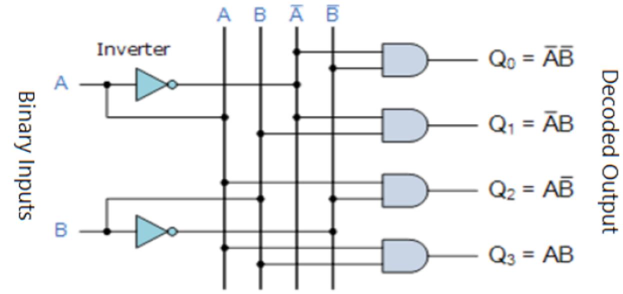


- Exactly one output will be active for each combination of the inputs.
- Each of these input combinations only one of the **M** outputs will be active ***high*** (1), all the other outputs are ***low*** (0).
- An **AND** gate can be used as the basic decoding element because it produces a ***high*** output only when all inputs are ***high***.

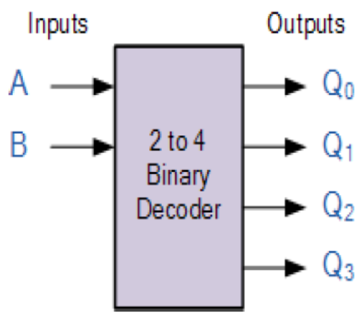
➢ 2 to 4 decoder

✓ 2 Inputs

✓ 4 Outputs



2-4 decoder with active high output



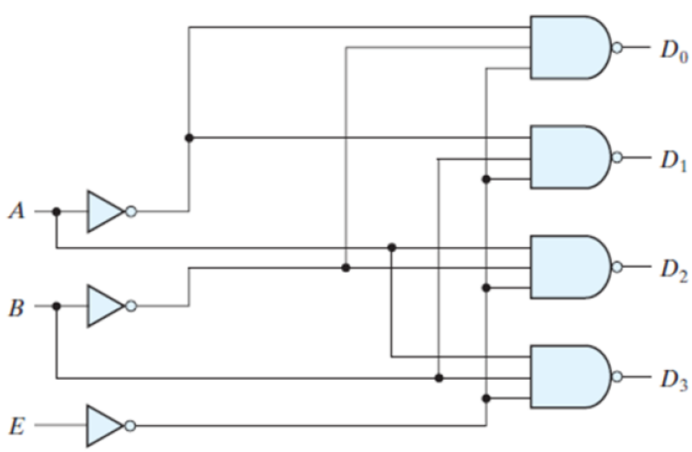
Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Logical symbol and truth table of 2-4 decoder

➢ 2 to 4 decoder with enable

✓ Active (*Low*)



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

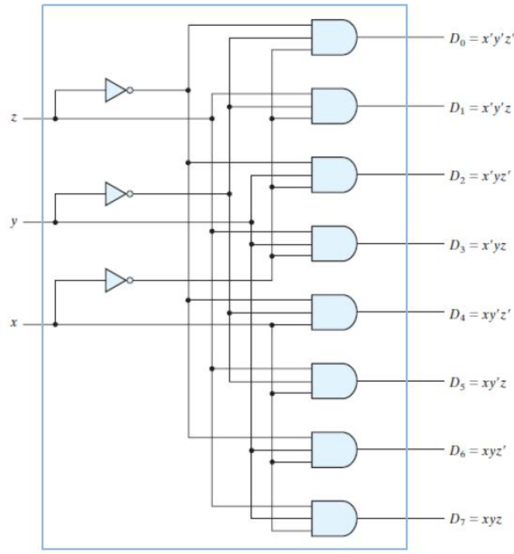
(b) Truth table

Two-to-four-line decoder with enable input

➢ 3 to 8 decoder (active-*high*)

✓ 3 Inputs

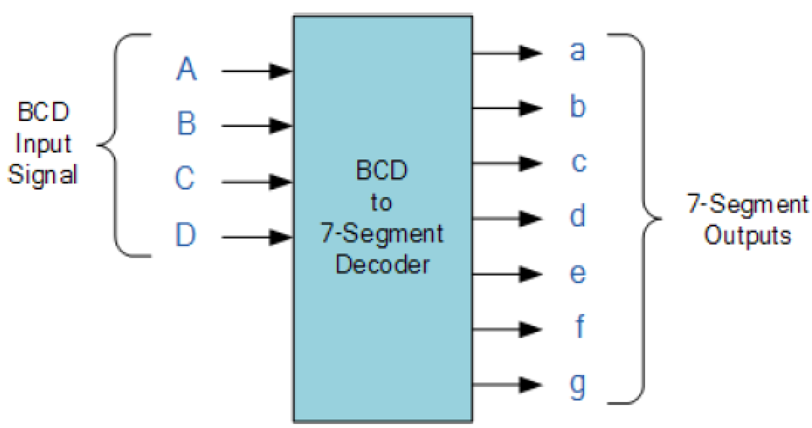
✓ 8 Outputs



x	y	z	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

➢ 4 to 16 decoder

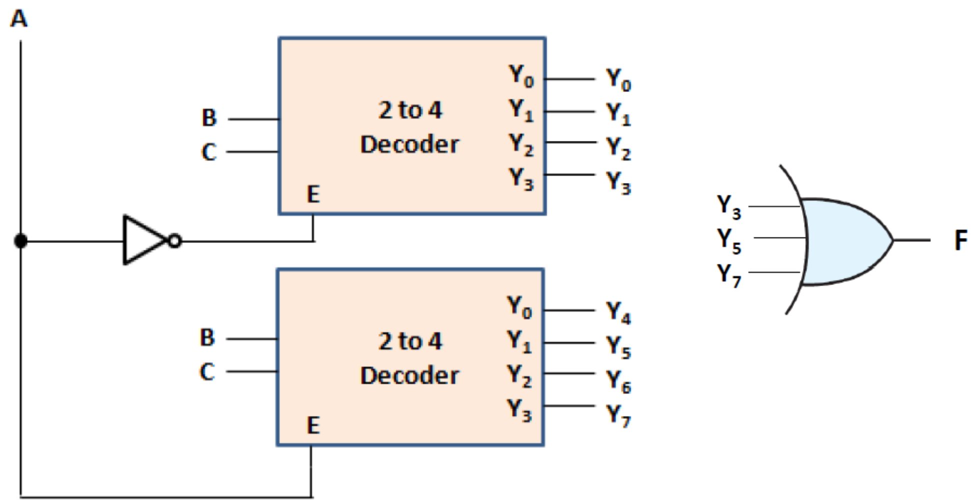
✓ BCD to 7-Segment Display Decoder



❑ Implement the following function using 2 to 4 decoder:

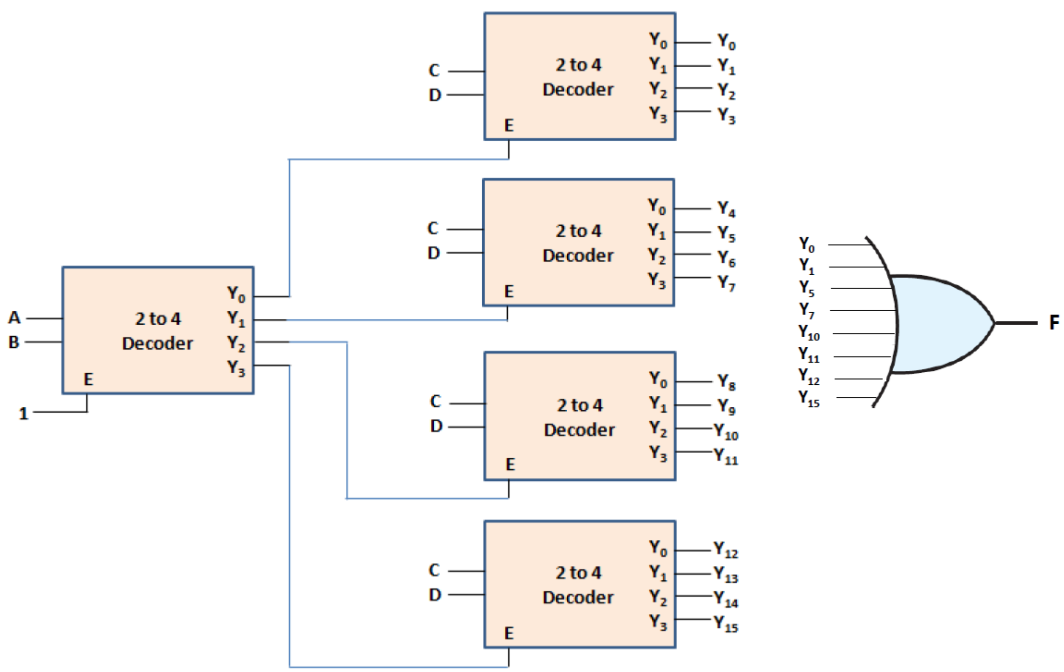
$$F(A,B,C) = \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,C$$

F(A,B,C) = Σ (3,5,7)



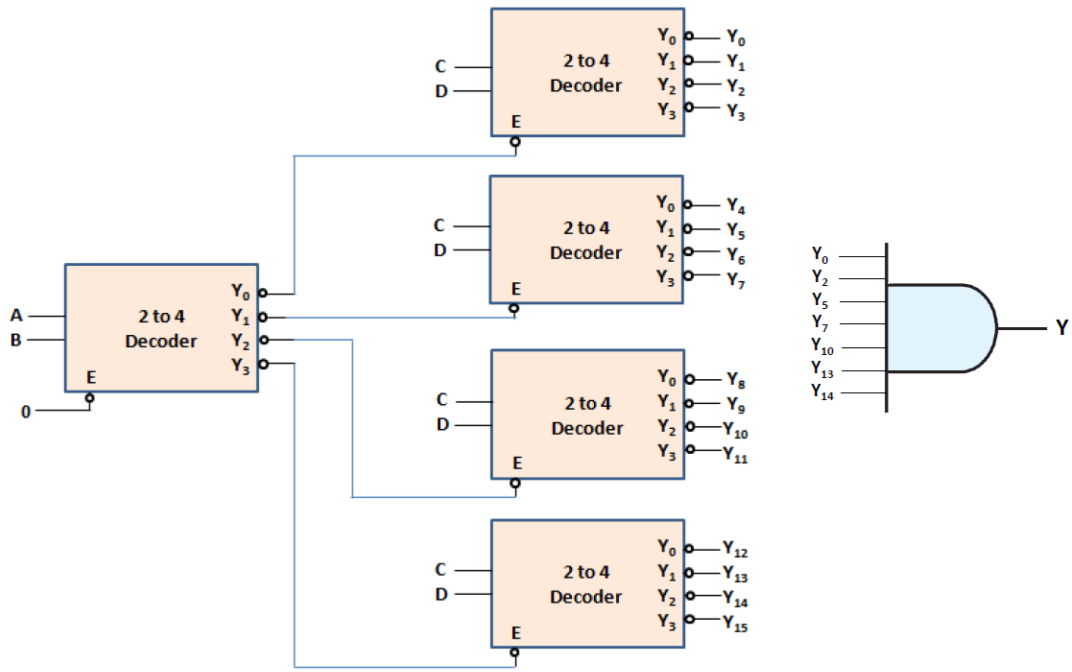
2: Implement the following function using 2 to 4 decoder:

F(A,B,C,D)= Σ (0,1,5,7,10,11,12,15) **+ d** (2,4,6,14) .



❑ Implement the following function using 2 to 4 decoder:

Y(A,B,C,D) = Π (0,2,5,7,10,13,14) .

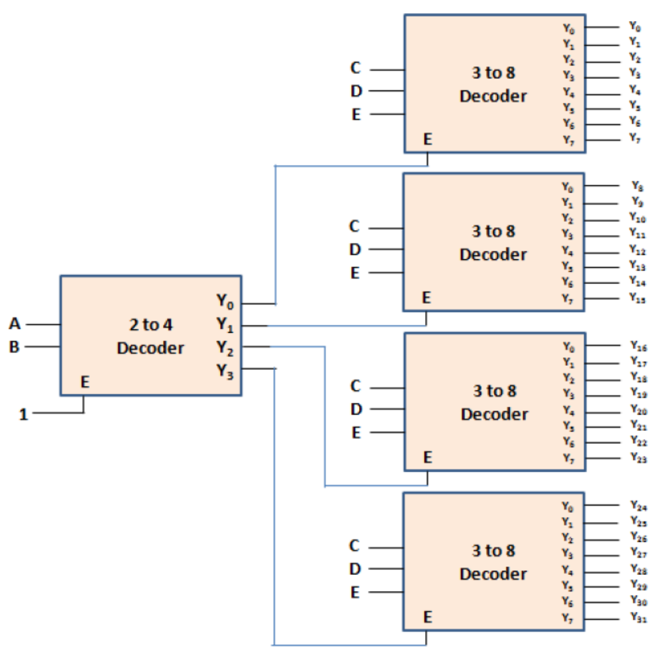


❑ Implement the following function using (3 to 8) and (2 to 4) decoder (active high):

Input binary number [0-30]

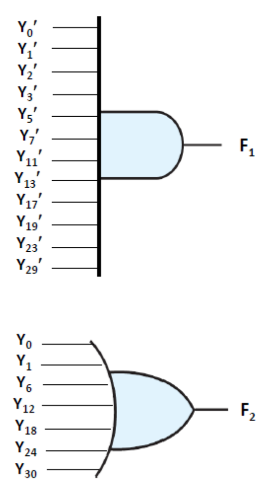
Output F_1 = 0 when input is 0,1,2 or prime number .

Output F_2 = 1 when input is 0,1 or multiples of number 6.



F_1 (A,B,C,D,E) = Π (0,1,2,3,5,7,11,13,17,19,23,29)

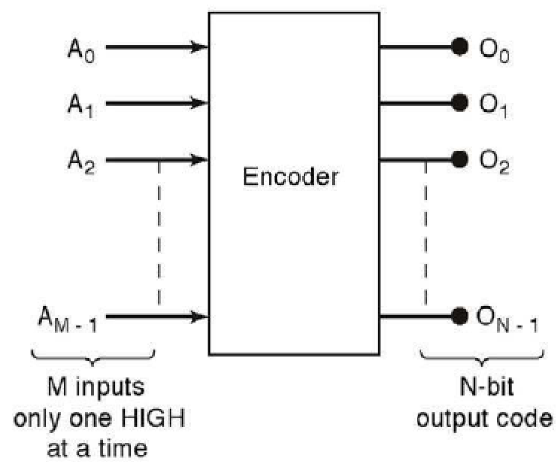
F_2 (A,B,C,D,E) = Σ (0,1,6,12,18,24,30)



Encoders

❑ <u>Encoder circuit</u> : An encoder is a combinational logic gates that accepts one or multiple inputs and generates a specific output code. Only one input is triggered at a time as shown in figure below.

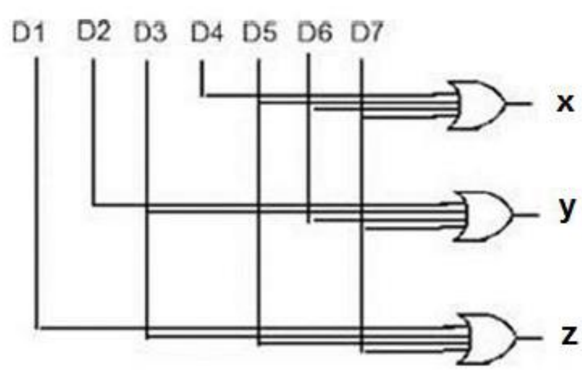
- $M = 2^N$ M: number of inputs and N: number of outputs



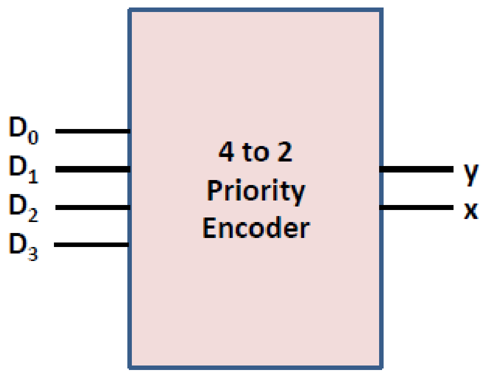
❑ 8 to 3 encoder Implementation

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- z = D1 + D3 + D5 + D7
- y = D2 + D3 + D6 + D7
- x = D4 + D5 + D6 + D7



❑ Priority encoder



Inputs				Outputs	
D_0	D_1	D_2	D_3	x	y
0	0	0	0	X	X
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

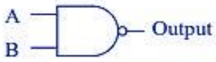
Name:

Class:

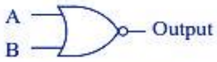
Questions

Question 1

Identify each of these logic gates by name, and complete their respective truth tables:



A	B	Output
0	0	
0	1	
1	0	
1	1	



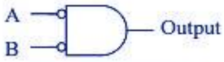
A	B	Output
0	0	
0	1	
1	0	
1	1	

A	B	Output
0	0	
0	1	
1	0	
1	1	

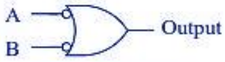
A	B	Output
0	0	
0	1	
1	0	
1	1	

A	B	Output
0	0	
0	1	
1	0	
1	1	

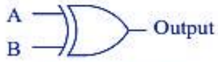
A	Output
0	
1	



A	B	Output
0	0	
0	1	
1	0	
1	1	



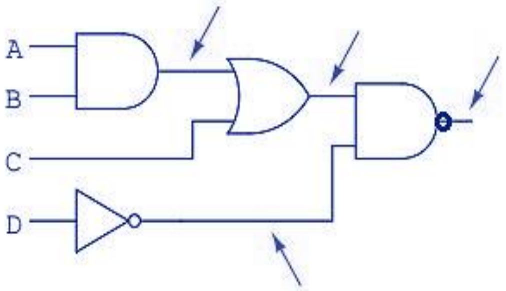
A	B	Output
0	0	
0	1	
1	0	
1	1	



A	B	Output
0	0	
0	1	
1	0	
1	1	

Question 2

Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expressions next to each gate output in the diagram:



Question 3

Apply DE Morgan's theorem to each of the following expressions

1) $\overline{(A+ B + C)D}$ 2) $\overline{A\overline{B} + \overline{C}D} + EF$

Question 4

Simplify the following expression: $Y = AB + A(B + C) + B(B + C)$

Question 5

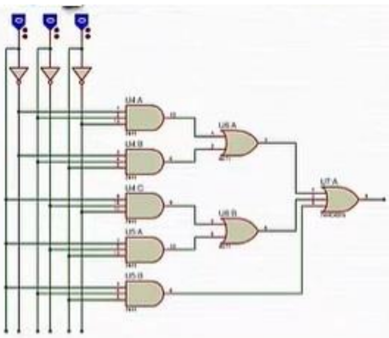
List the truth table of the function: Write canonical form and minetmer

* $F = xy + x\overline{y} + \overline{y}z$

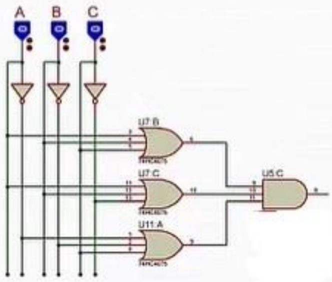
اضافى للتقويه

A	B	C	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Minterms → Sum of Products → Out = $\bar{A}.B.\bar{C} + \bar{A}.B.C + A.\bar{B}.\bar{C} + A.\bar{B}.C + A.B.C$



Maxterms → Product of Sum → Out = $(A + B + C).(A + B + \bar{C}).(\bar{A} + \bar{B} + C)$



خريطة كارنوف (Karnaugh Map)

هي عبارة عن اعادة رسم جدول الحقيقة على صورة خلايا في جدول يمكن استخدامها فيما بعد في عمليات تبسيط التعابير البولينية

جدول الحقيقة (Truth Table)

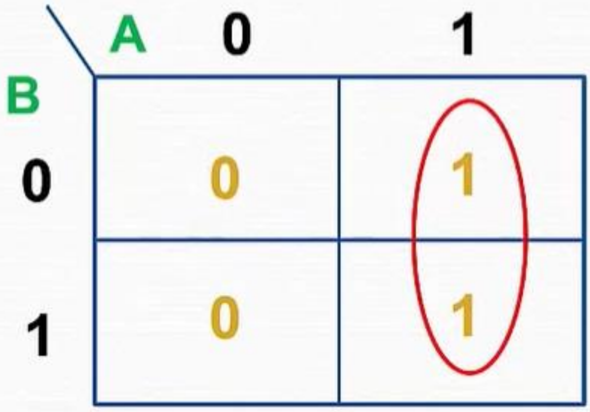
A	B	Out
0	0	0
0	1	0
1	0	1
1	1	1

خريطة كارنوف (Karnaugh Map)

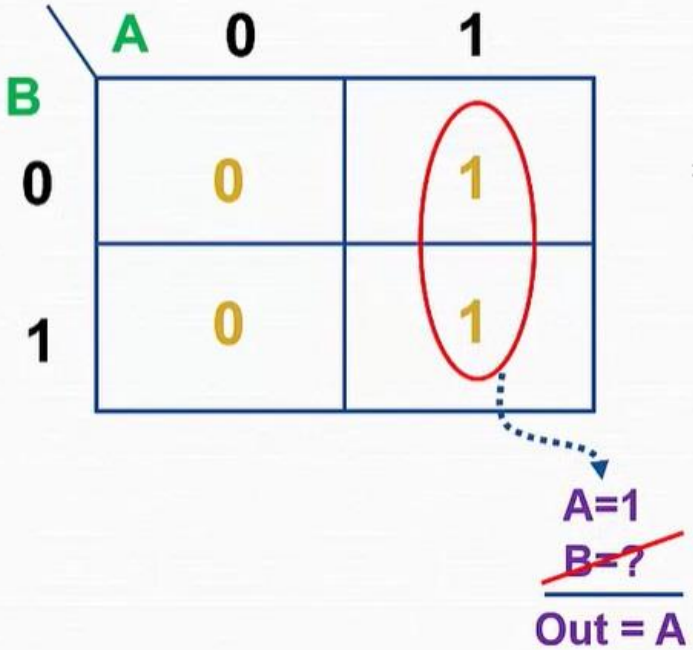
هي عبارة عن اعادة رسم جدول الحقيقة على صورة خلايا في جدول يمكن استخدامها فيما بعد في عمليات تبسيط التعابير البولينية

جدول الحقيقة (Truth Table)

A	B	Out
0	0	0
0	1	0
1	0	1
1	1	1



خطوات الاختصار (التبسيط)



1- نرسم حلقة/ حلقات لتجمع مجموعة من «1» ... ولكن بشرطين :

- يجب ان تكون «1» متجاورة في وضع أفقي أو عمودي فقط
- يجب أن يكون عددها : 1، 2، 4، 8، 16 ، ...

2- نعبر عن كل حلقة بتعبير بوليني جديد

قبل التبسيط

A	B	Out
0	0	0
0	1	0
1	0	1
1	1	1

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

بعد التبسيط

B \ A	0	1
0	0	1
1	0	1

$$Out = A$$

مدخلين

B \ A	0	1
0		
1		

ثلاث مداخل

الخلايا المتجاورة يجب ان تكون تختلف في بت واحد فقط (bit)

C \ AB	00	01	11	10
0				
1				

أربعة مداخل

CD \ AB	00	01	11	10
00				
01				
11				
10				

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

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

SOP & POS

LEC-4/Part2

First Class

2022

By: Assistant Lecturer
Dena Nameer

Introduction

All Boolean expressions, regardless of their form, can be converted into either of two **standard forms**:

- The sum-of-products (SOP) form
- The product-of-sums (POS) form

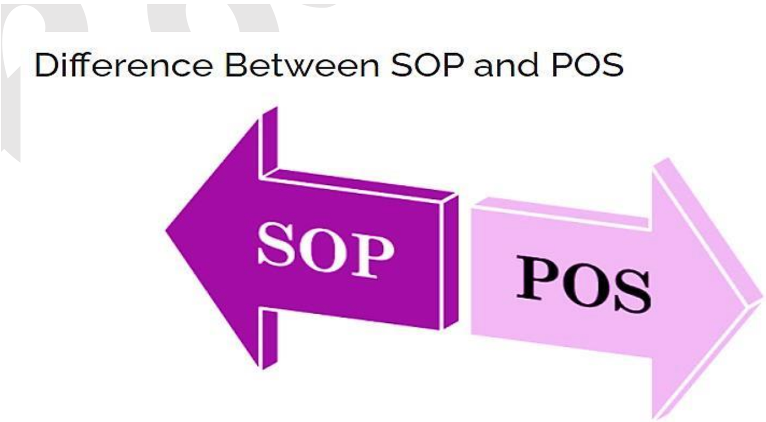
Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

The Sum-of-Products (SOP) Form

When we add two or multiple product terms by a Boolean addition, the output expression is a sum-of-products **(SOP)**. It is mainly implemented by an **AND-OR** logic where the product of the variables are first produced by **AND** gate and then added by the **OR** gates.

The product-of-Sums (POS) Form

POS (Product of Sums) is the representation of the Boolean function in which the variables are first summed, and then the Boolean product is applied in the sum terms. It just needs the variables to be inserted as the inputs to the OR gate. The **terms generated** by **the OR gates** are inserted in the **AND gate**. The sum term is formed by an OR operation, and product of two or multiple sum terms is created by an AND operation.



The prior difference between the SOP and POS is that the SOP contains the OR of the multiple product terms. Conversely, POS produces a logical expression comprised of the AND of the multiple OR terms.

Comparison Chart

BASIS FOR COMPARISON	SOP	POS
Expands to	Sum of Product	Product of Sum
Basic	Form of representation of a boolean expression incorporating minterms	Technique of generating a boolean expression involving maxterms.
Expression includes	Product terms are taken where the input set produces a value 1.	Only Sum terms which generate a value 0.
Method	1 represents the variable and 0 is the complement of it.	0 represents the variable and 1 complement of the variable.
Obtained through	Adding corresponding product terms.	Multiplying the relevant sum terms.
Order of implementation	OR gate is employed after the AND gate.	AND gate is used after the OR gate.

Minterms and Maxterms

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now consider two binary variables x and y combined with an **AND** operation. Since each variable may appear in either form, there are four possible combinations: x'y', x'y, xy', and xy. Each of these four AND terms is called a **minterm**, or a **standard product**.

A Boolean function can be expressed algebraically from a given truth table by form-ing a minterm for each combination of the variables that produces a **1** in the function and then taking the OR of all those terms.
hat mean

For example:

$$f_1 = x'y'z + xy'z' + xyz$$

x	y	z	Minterms Term	Minterms Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

Functions of Three Variables

x	y	z	F1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The minterm or standard product is:

$$F = m_1 + m_4 + m_7$$

Note:

The minterms whose sum defines the Boolean function are those which give the **1's** of the function in a truth table .*Also* **"sum" meaning the ORing of terms.**

Now consider the complement of a Boolean function. If we take the complement of F, we obtain :

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	***y***	***z***	**Term**	**Designation**	**Term**	**Designation**
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Functions of Three Variables

x	***y***	***z***	**F**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$F(\text{maxterms}) = M0 .M2 .M3 .M5 .M6$$

Note:
This example demonstrate a second property of Boolean algebra: Any Boolean function can be expressed as a product of maxterms (**with "product" meaning the ANDing of terms**). The procedure for obtaining the product of maxterms directly from the truth table is as follows: Form a **maxterm** for each combination of the variables that produces a (**0**) in the function, and then **form the AND of all those maxterms.**

Functions of Three Variables

x	*y*	*z*	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

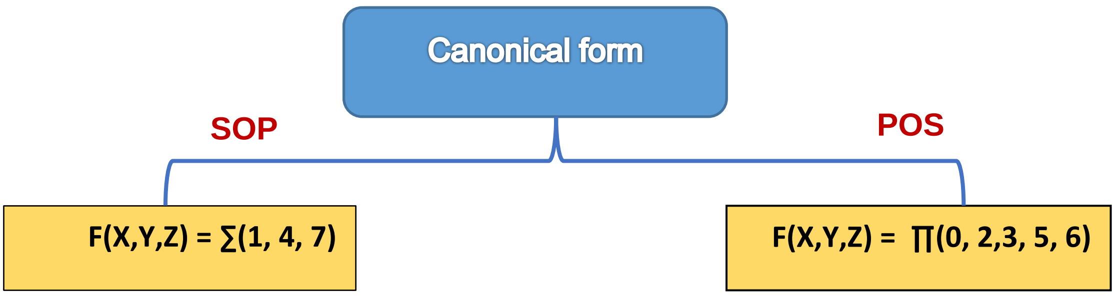
Minterm

Maxterm

Canonical form

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

For example the canonical form is:



Example 1:

Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

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

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

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

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

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

Truth Table for $F = A + B'C$

A	B	C	F
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	1

Example 2:

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$
\begin{aligned}
F = xy + x'z &= (xy + x')(xy + z) \\
&= (x + x')(y + x')(x + z)(y + z) \\
&= (x' + y)(x + z)(y + z)
\end{aligned}
$$

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

$$
\begin{aligned}
x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\
x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\
y + z &= y + z + xx' = (x + y + z)(x' + y + z)
\end{aligned}
$$

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

$$
\begin{aligned}
F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\
&= M_0 M_2 M_4 M_5
\end{aligned}
$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

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

Truth Table for $F = xy + x'z$

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

Minterms

Maxterms

Good luck

Dena.N

Subtractor

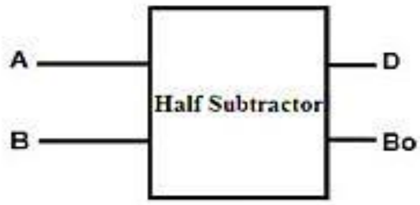
First Class

2022

Subtractor

By: Assistant Lecturer
Dena Nameer

The Half-Subtractor



Half-Subtractor Block Diagram

Half-Subtractor Truth Table

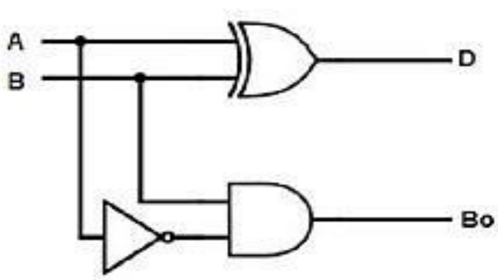
Input		Output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The difference (D) can be expressed as

$D = A \oplus B$

The borrow (B) can be expressed as

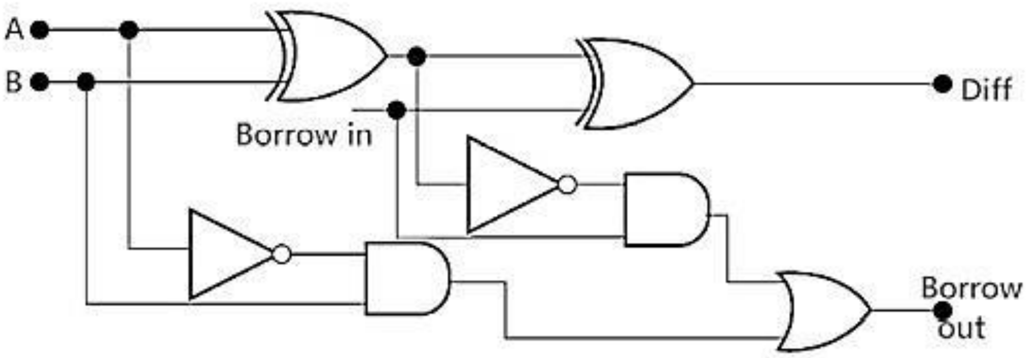
$B_o = \bar{A}B$



Half-Subtractor Circuit

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

Borrow=x' z+x' y+yz



Diff = (A ⊕ B) ⊕ $Borrow_{in}$

Borrow = A'.B + (A ⊕ B)'

Good luck

Dena.N

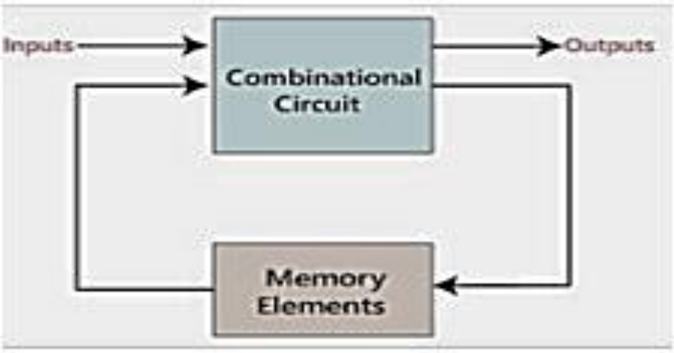
Sequential Logic Gate

First Class

2022

By: Assistant Lecturer
Dena Nameer

Sequential Logic Circuits



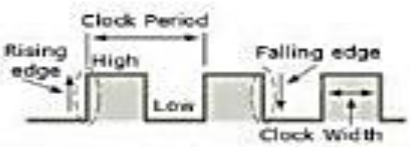
In our previous lectures we saw how combinational circuits produce a single output from a combination of input signals. A combinational circuit does not have memory as its output is determined only by the present input, and not by any previous input. The output cannot reflect previous input level conditions.

In contrast, sequential circuits—the subject of this session—do have memory. The output of a sequential circuit is determined both by the present input and previous input.

What specifically does a sequential circuit need in order to reflect past input into its present output? Clearly, it needs a memory element. Such a memory element is called a *flip-flop*.

Two Main Types of Sequential Circuits

There are two types of sequential circuit, Synchronous and Asynchronous. Synchronous types use a clock input to drive the circuit, while Asynchronous sequential circuits do not use a clock signal as synchronous circuits do.



Good luck

Dena.N

Code Conversion

First Class

2022

By: Assistant Lecturer
Dena Nameer

Code Conversion

➢ Three bit binary to gray code conversion

Truth Table:

Input				Output		
Dec.	B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

SOP

$G_0\,(B_2,B_1,B_0) = \Sigma\,(1,2,5,6)$

$G_1\,(B_2,B_1,B_0) = \Sigma\,(2,3,4,5)$

$G_2\,(B_2,B_1,B_0) = \Sigma\,(4,5,6,7)$

POS

$G_0\,(B_2,B_1,B_0) = \Pi\,(0,3,4,7)$

$G_1\,(B_2,B_1,B_0) = \Pi\,(0,1,6,7)$

$G_2\,(B_2,B_1,B_0) = \Pi\,(0,1,2,3)$

$$G_0\,(B_2,B_1,B_0) = \Sigma\,(1,2,5,6)$$

B_2 \ B_1B_0	$B_1'B_0'$	$B_1'B_0$	B_1B_0	B_1B_0'
B_2'		1		1
B_2		1		1

$$G_0\,(B_2,B_1,B_0) = B_1'B_0 + B_1B_0'$$

$G_1(B_2,B_1,B_0) = \Pi(0,1,6,7)$

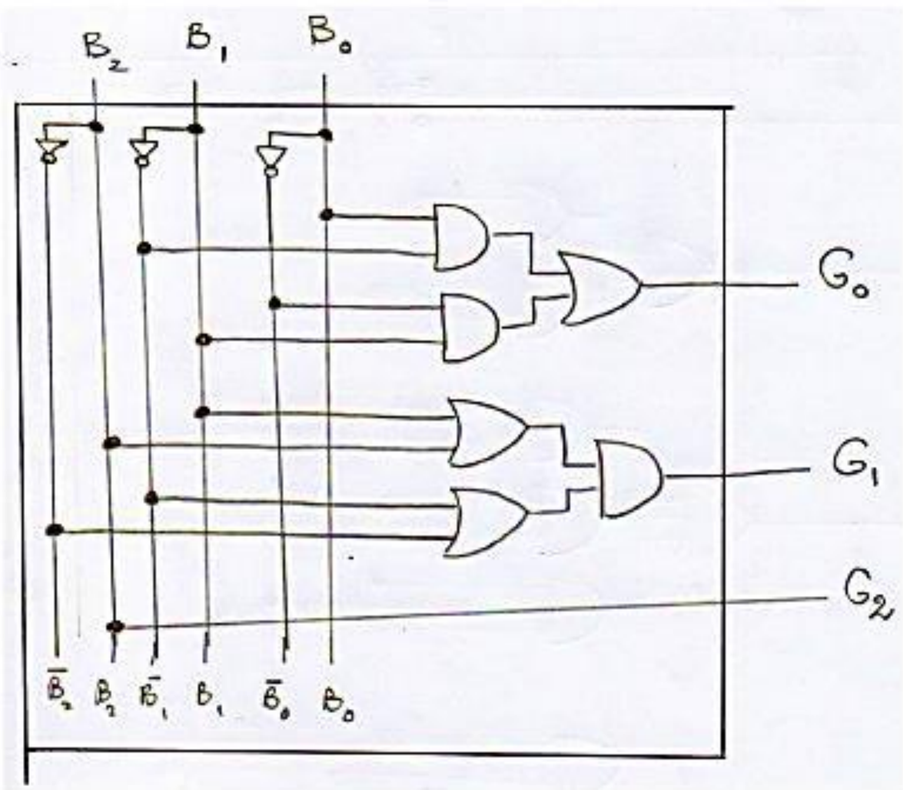
$B_2 \backslash B_1B_0$	B_1+B_0	B_1+B_0'	$B_1'+B_0'$	$B_1'+B_0$
B_2	0	0		
B_2'			0	0

$G_1(B_2,B_1,B_0) = (B_2+B_1)(B_2'+B_1')$

$G_2(B_2,B_1,B_0) = \Pi(0,1,2,3)$

$B_2 \backslash B_1B_0$	B_1+B_0	B_1+B_0'	$B_1'+B_0'$	$B_1'+B_0$
B_2	0	0	0	0
B_2'				

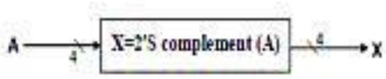
$G_2(B_2,B_1,B_0) = B_2$



Code Conversion

➢ 4-bit Binary to 2'S complement conversion

Input					Output			
Dec.	A_3	A_2	A_1	A_0	X_3	X_2	X_1	X_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1
2	0	0	1	0	1	1	1	0
3	0	0	1	1	1	1	0	1
4	0	1	0	0	1	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	0
7	0	1	1	1	1	0	0	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	0	1	1	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	0	1	0	1
12	1	1	0	0	0	1	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	0	1	0
15	1	1	1	1	0	0	0	1



SOP standard form

$X_0 (A_3,A_2,A_1,A_0) = \Sigma\,(1,3,5,7,9,11,13,15)$; $X_1 (A_3,A_2,A_1,A_0) = \Sigma\,(1,2,5,6,9,10,13,14)$

$X_2 (A_3,A_2,A_1,A_0) = \Sigma\,(1,2,3,4,9,10,11,12)$; $X_3 (A_3,A_2,A_1,A_0) = \Sigma\,(1,2,3,4,5,6,7,8)$

POS standard form

$X_0 (A_3,A_2,A_1,A_0) = \Pi\,(0,2,4,6,8,10,12,14)$; $X_1 (A_3,A_2,A_1,A_0) = \Pi\,(0,3,4,7,8,11,12,15)$

$X_2 (A_3,A_2,A_1,A_0) = \Pi\,(0,5,6,7,8,13,14,15)$; $X_3 (A_3,A_2,A_1,A_0) = \Pi\,(0,9,10,11,12,13,14,15)$

SOP standard form

✓ $X_0 (A_3,A_2,A_1,A_0) = \Sigma\,(1,3,5,7,9,11,13,15)$

A_3A_2 \ A_1A_0	$A_1'A_0'$	$A_1'A_0$	A_1A_0	A_1A_0'
$A_3'A_2'$		1	1	
$A_3'A_2$		1	1	
A_3A_2		1	1	
A_3A_2'		1	1	

$X_0 (A_3,A_2,A_1,A_0) = A_0$

✓ $X_2 (A_3,A_2,A_1,A_0) = \Sigma\,(1,2,3,4,9,10,11,12)$

A_3A_2 \ A_1A_0	$A_1'A_0'$	$A_1'A_0$	A_1A_0	A_1A_0'
$A_3'A_2'$		1	1	1
$A_3'A_2$	1			
A_3A_2	1			
A_3A_2'		1	1	1

$X_2 (A_3,A_2,A_1,A_0) = A_2'A_0 + A_2'A_1 + A_2A_1'A_0'$

POS standard form

✓ $X_1(A_3,A_2,A_1,A_0) = \Pi(0,3,4,7,8,11,12,15)$

$A_3A_2 \backslash A_1A_0$	A_1+A_0	A_1+A_0'	$A_1'+A_0'$	$A_1'+A_0$
A_3+A_2	0		0	
A_3+A_2'	0		0	
$A_3'+A_2'$	0		0	
$A_3'+A_2$	0		0	

$X_1(A_3,A_2,A_1,A_0) = (A_1+A_0)(A_1'+A_0')$

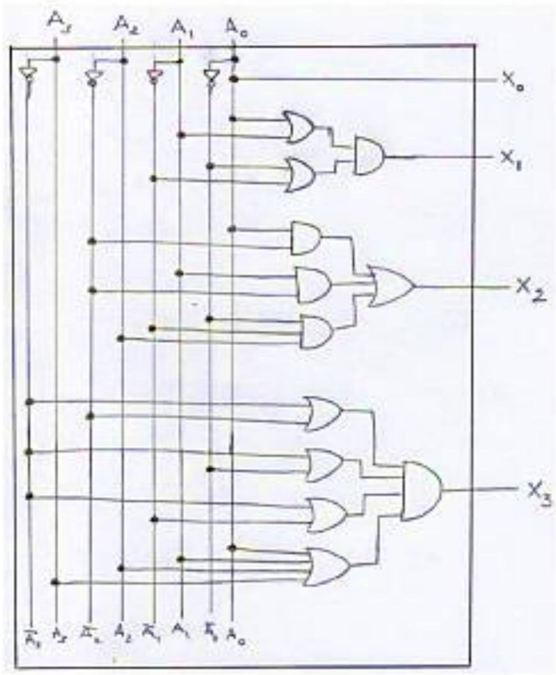
✓ $X_3(A_3,A_2,A_1,A_0) = \Pi(0,9,10,11,12,13,14,15)$

$A_3A_2 \backslash A_1A_0$	A_1+A_0	A_1+A_0'	$A_1'+A_0'$	$A_1'+A_0$
A_3+A_2	0			
A_3+A_2'				
$A_3'+A_2'$	0	0	0	0
$A_3'+A_2$		0	0	0

$X_3(A_3,A_2,A_1,A_0) = (A_3'+A_2')(A_3'+A_0')(A_3'+A_1')(A_3+A_2+A_1+A_0)$

$X_0(A_3,A_2,A_1,A_0) = A_0$; $X_1(A_3,A_2,A_1,A_0) = (A_1+A_0)(A_1'+A_0')$; $X_2(A_3,A_2,A_1,A_0) = A_2'A_0 + A_2'A_1 + A_2A_1'A_0'$

$X_3(A_3,A_2,A_1,A_0) = (A_3'+A_2')(A_3'+A_0')(A_3'+A_1')(A_3+A_2+A_1+A_0)$



Good luck

Dena.N

Quine-McClusckey Tabulation Method

The Quine-McClusckey algorithm is functionally identical to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a deterministic way to check that the minimal form of a Boolean function has been reached. It is sometimes referred to as the tabulation method.

The largest number of variables that can be used in Karnaugh map is 6 variables, while un limited numbers of variables can be used in the Quine-McClusckey algorithm.

Two steps process utilizing tabular listings to:

- Identify prime implicants
- Identify minimal PI set

All work is done in tabular form:

- Number of variables is not limited
- Basic for many computer implementations

Example (1):

Simplify the following canonical form using Quine-McClusckey method:

Y=∑m (0,2,8,5,10,7,13,15)

0 0000, 2 0010, 8 1000, 5 0101

10 1010, 7 0111, 13 1101, 15 1111

Dec.	Binary A B C D	Click
0	0 0 0 0	✓
2	0 0 1 0	✓
8	1 0 0 0	✓
5	0 1 0 1	✓
10	1 0 1 0	✓
7	0 1 1 1	✓
		✓

✓

Dec.	Binary A B C D	Click
0,2	0 0 - 0	✓
0,8	- 0 0 0	✓
2,10	- 0 1 0	✓
8,10	1 0 - 0	✓
5,7	0 1 - 1	✓
5,13	- 1 0 1	✓
7,15	- 1 1 1	✓
13,15	1 1 - 1	✓

Dec.	Binary A B C D	Click
0,2,8,10	- 0 - 0	
0,8,2,10	- 0 - 0	
5,13,7,15	- 1 - 1	
5,7,13,15	- 1 - 1	

Y=B’D’+BD

To confirm that we get the right solution we can use Karnaugh map method in order to ensure our result:

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

Y=B’D’+BD

Concept of Prime Implicant

Sometimes the Quine-McCluscкey method gives the simplified solution but not the optimum one as we will see in the next example:

Example (2):

Simplify the following canonical form using Quine-McCluscкey method:

$Y = \sum m\ (0,1,2,3,5,7,13,15)$

Dec.	Binary A B C D	Click
0	0 0 0 0	✓
1	0 0 0 1	✓
2	0 0 1 0	✓
3	0 0 1 1	✓
5	0 1 0 1	✓
7	0 1 1 1	✓
13	1 1 0 1	✓
15	1 1 1 1	✓

Dec.	Binary A B C D	Click
0,1	0 0 0 -	✓
0,2	0 0 - 0	✓
1,3	0 0 - 1	✓
1,5	0 - 0 1	✓
2,3	0 0 1 -	✓
3,7	0 - 1 1	✓
5,13	- 1 0 1	✓
5,7	0 1 - 1	✓
7,15	- 1 1 1	✓
13,15	1 1 - 1	✓

Dec.	Binary A B C D	Click
0,1,2,3	0 0 - -	
0,2,1,3	0 0 - -	
1,5,3,7	0 - - 1	
1,3,5,7	0 - - 1	
5,13,7,15	- 1 - 1	
5,7,13,15	- 1 - 1	

Y=A'B'+A'D+BD

Using Karnaugh map method we get:

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

Out of K map: Y=A'B'+BD

Out of Q.M: Y=A'B'+**A'D**+BD

Concept of prime implicant:

Y=A'B'+A'D+BD

	0	1	2	3	5	7	13	15
A'B'	X	X	X	X				
A'D		X		X	X	X		
BD					X	X	X	X

The P.I is: Y=A'B'+BD

Good luck

Dena.N