EE201

تصميم منطقي

Logic Design

إعداد وتجميع

ا.م.د/ هانى احمد عطاالله

كلية الهندسة بقنا

قسم الهندسة الكهربية

العام الجامعى

2025/2024

بيانات الكتاب

الكلية: الحاسبات والمعلومات

الفرقة: المستوى الثانى

التخصص: عام

تاريخ النشر: 2024

عدد الصفحات: 145

اعداد وتجميع : ا.م. د/ هانى احمد عطاالله

الرموز المستخدمة

📄	نص للقراءة والدراسة
📝	أنشطة ومهام
❓	أسئلة للتفكير والتقييم الذاتي
🎞	فيديو للمشاهدة
🔗	رابط خارجي
👥	تواصل عبر مؤتمر الفيديو

المحتوى

الصور والاشكال

الفيديو

Introduction

Digital Design is about designing in digital space so that the created contents can be displayed and seen on a digital device. With the availability of high computing power, designers are able to quickly create designs in digital space prior to actual deployment.

The course is an introduction to digital design technology. It allows you to understand the basics of digital design and helps you develop skills.

In this course the student will be learn the following topics: Binary systems and Boolean algebra- logic gates – simplifying the Boolean circuit-combinational circuits- encryption and decryption - asynchronous sequential circuits and their applications- synchronous sequential circuits and their applications- displacement recorders- counters – memories and types of memories.

The following is a brief summary of the topics that are covered in each chapter.

Chapter 1 presents the various binary systems suitable for representing information in digital systems. The binary number system is explained and binary codes are illustrated. Examples are given for addition and subtraction of signed binary numbers and decimal numbers in binary-coded decimal (BCD) format.

Chapter 2 introduces the basic postulates of Boolean algebra and shows the correlation between Boolean expressions and their corresponding logic diagrams. All Possible logic operations for two variables are investigated, and the most useful logic gates used in the design of digital systems are identified. This chapter also introduces basic CMOS logic gates.

Chapter 3 covers the map method for simplifying Boolean expressions. The map method is also used to simplify digital circuits constructed with AND-OR, NAND, or NOR gates. All other possible two-level gate circuits are considered, and their method of implementation is explained. Verilog HDL is introduced together with simple examples of gate-level models.

Chapter 4 outlines the formal procedures for the analysis and design of combinational circuits. Some basic components used in the design of digital systems, such as adders and code converters, are introduced as design examples. Frequently used digital logic functions such as parallel adders and subtractors, decoders, encoders, and multiplexers are explained, and their use in the design of combinational circuits is illustrated.

Chapter 5 outlines the formal procedures for analyzing and designing clocked (synchronous) sequential circuits. The gate structure of several types of flip-flops is presented together with a discussion on the difference between level and edge triggering.

Chapter 6 deals with various sequential circuit components such as registers, shift registers, and counters. These digital components are the basic building blocks from which more complex digital systems.

Logic Design

Table of Contents

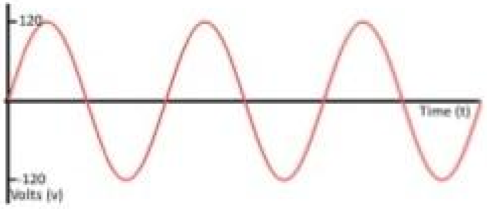
Ch1: Introduction to Digital Design and Binary Numbers

Digital vs. Analog

- **Analog**: Continuous function V of continuous variable t (time, space etc.) : V(t)

An analog* quantity is one having continuous values.

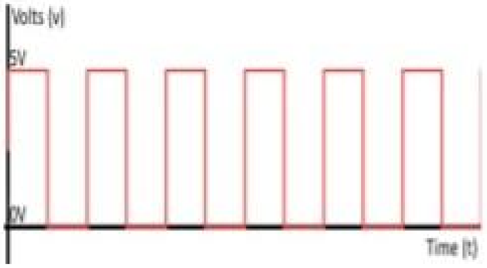
Example: Human voice in air, analog devices.

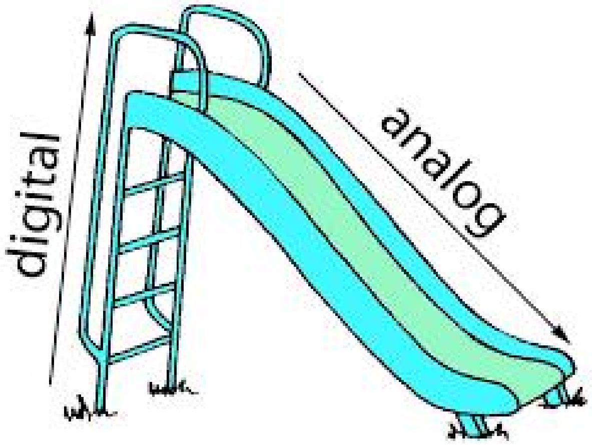


- **Digital**: Discrete function V_k of discrete sampling variable t_k, with k = integer: $V_k = V(t_k)$

A digital quantity is one having a discrete set of values.

Example: Computer, CDs, DVDs, digital devices.

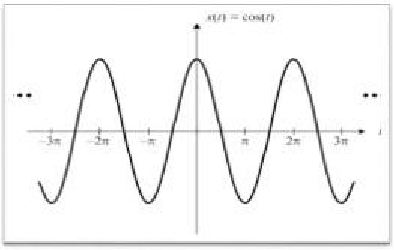




Continuous vs. Discrete

Continuous:

- defined for every instant of time
- denoted by x(t)



Discrete:

- defined at the discrete-instant of time
- denoted by x(n)

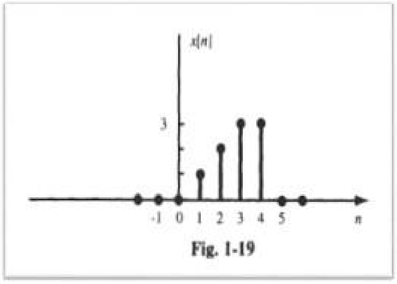
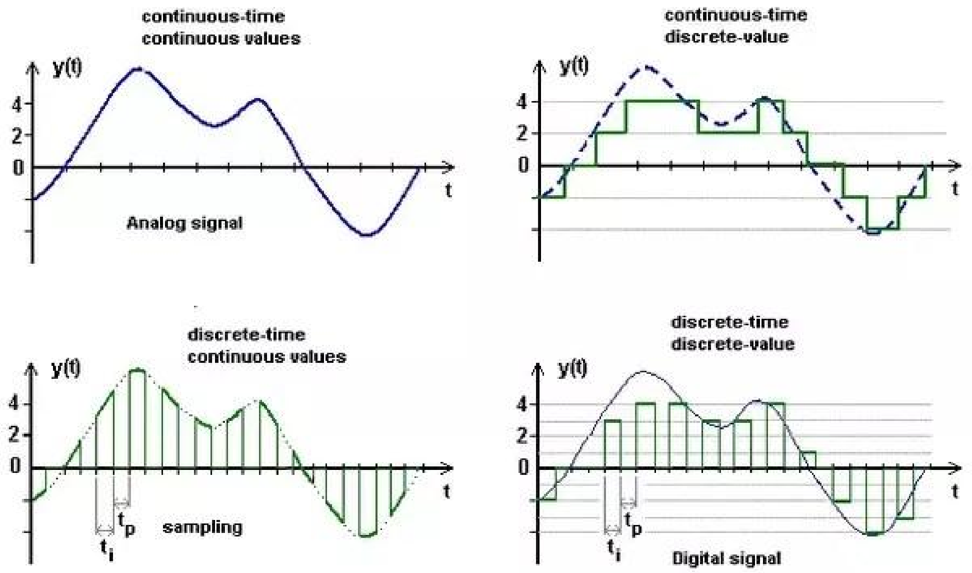


Fig. 1-19



Digital vs. Analog

- **An analog system has continuous range of values**
  - **A mercury thermometer**
  - **Vinyl records**
  - **Human eye**

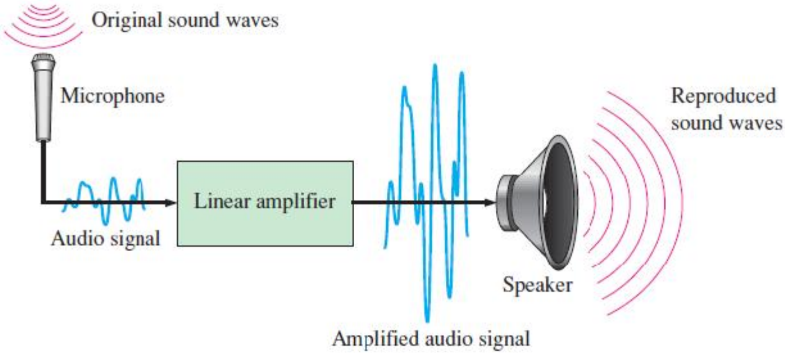
- A digital system has set of discrete values

 Digital Thermometer

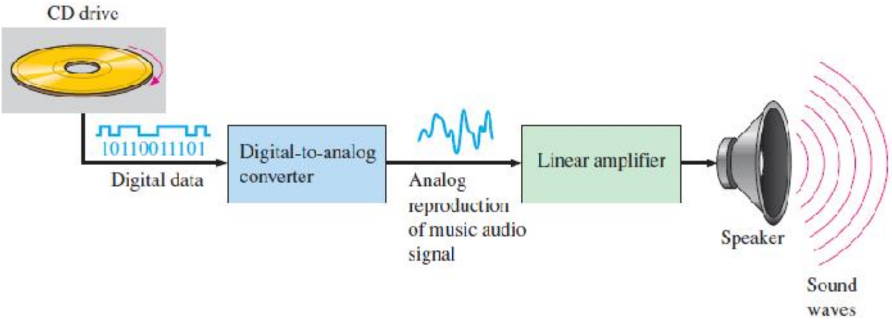
 Compact Disc (CD)

 Digital camera

Analog System: Audio System



System using Digital and Analog Methods: The Compact Disk (CD) Player



CD drive
10110011101
Digital data
Digital-to-analog converter
Analog reproduction of music audio signal
Linear amplifier
Speaker
Sound waves

Digital Systems and Units

CPU	Digital Watch	Digital Cameras

<u>Benefits of using digital</u>

- **Advantages of using Digital:**
- **Cheap electronic circuits**
- **Easier to calibrate and adjust**
- **Resistance to noise: Clearer picture and sound**

Data Types and Representation

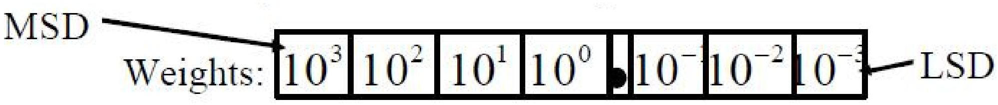
1- Integers: 35, 125, 5612*1e2

2-Real Numbers, 5.67, 0.05, 52*1e0.5

3-Strings,
a- Names (ali, maha, …)
b- Dates (12/9/2016)
c- Addresses (Qena, Egypt)
d- Telephone Numbers (0965214264)
e- string: $, &,%,@,…

Number Systems : 1- Decimal Code

Base (Radix) is 10 - symbols (0,1, . . 9) Digits (Base-1)



base 10

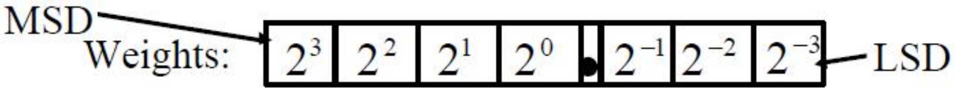
✓ If we were to write 1936.25 using a power series expansion and **base 10** arithmetic:

$$1\times10^3+9\times10^2+3\times10^1+6\times10^0+2\times10^{-1}+5\times10^{-2}$$

2- Binary Number System

Discrete elements of information are represented with bits called *binary codes*.

- **Base is 2 - symbols (0,1) - Binary Digits (Bits)**
- **Each position carries a weight (using decimal).**



✓ **If we write 10111.01 using a decimal power series we convert from binary to decimal:**

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

=1×16 + 0× 8 +1× 4 +1× 2 +1×1+ 0× 0.5 +1× 0.25 = 23.25

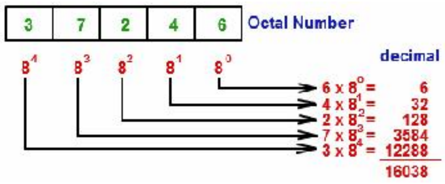
3- Octal Number system

The octal number system [from Greek: OKTΩ].

– Its base is 8 → eight digits 0, 1, 2, 3, 4, 5, 6, 7

✓ $(236.4)_8 = (?)_{10}$

$$2\times 8^2 + 3\times 8^1 + 6\times 8^0 + 4\times 8^{-1} = 158.5$$



4- Hexadecimal Number system

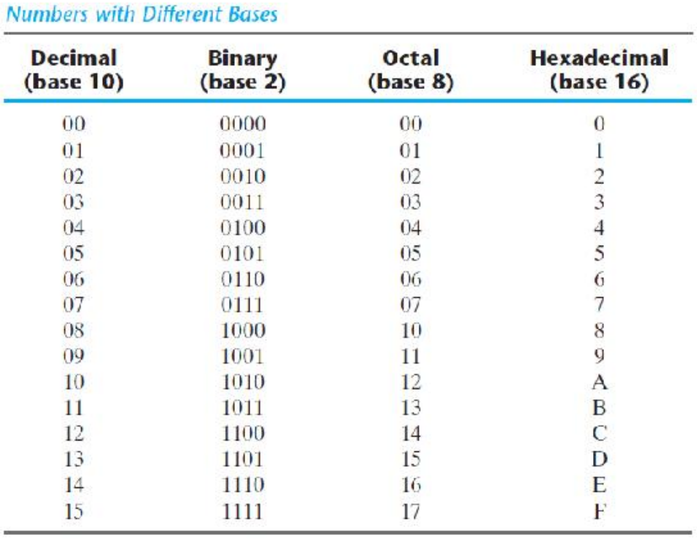
-The hexadecimal number system [from Greek: ΔΕΚΑΕΞΙ].

– Its base is 16 → first 10 digits are borrowed from the decimal system (0 -- 9) and the letters A, B, C, D, E, F are used for the digits 10, 11, 12, 13, 14, 15

- ✓ $(D63FA)_{16} = (\ ? \ )_{10}$

$$13\times16^4+6\times16^3+3\times16^2+15\times16^1+10\times16^0=877562$$

Different Systems



Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Conversion from Binary to Decimal

Let each bit of a binary number be represented by a variable whose subscript = bit positions, i.e.,

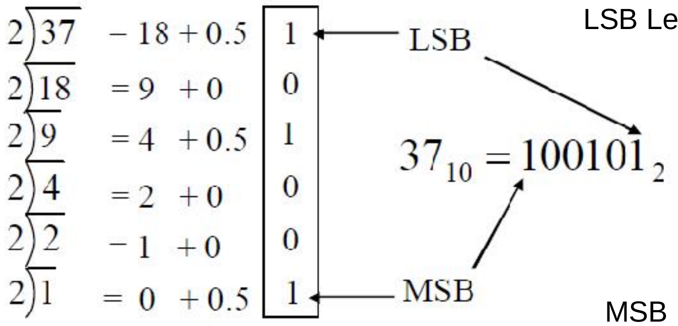
$$(110)_2 = (a_2 a_1 a_0)_2$$

Its decimal equivalent is:

$$(1\times 2^2 + 1\times 2^1 + 0\times 2^0)_{10} = (a_2\times 2^2 + a_1\times 2^1 + a_0\times 2^0)_{10}$$

It is necessary to separate the number into an integer part and a fraction: Repeatedly divide the decimal number by 2.

- ✓ Find the binary equivalent of 37.



$$\begin{array}{ll|c|}
2\overline{)37} & - 18 + 0.5 & 1 \\
2\overline{)18} & = 9 \ + 0 & 0 \\
2\overline{)9} & = 4 \ + 0.5 & 1 \\
2\overline{)4} & = 2 \ + 0 & 0 \\
2\overline{)2} & - 1 \ + 0 & 0 \\
2\overline{)1} & = 0 \ + 0.5 & 1
\end{array}$$

$$37_{10} = 100101_2$$

LSB Least Significant Bit

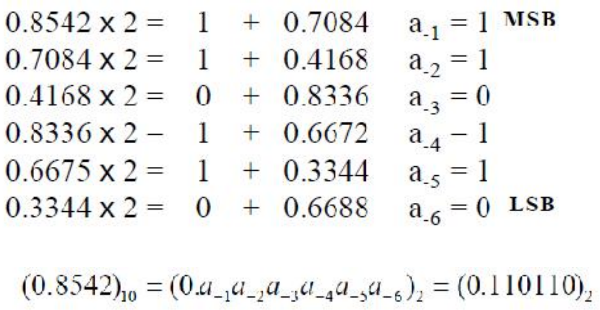
MSB Most Significant Bit

- ❑ $53_{10} = \underline{\ \ ?\ \ }_2$ ANS: $53_{10} = 110101_2$

Conversion from decimal fraction to binary:

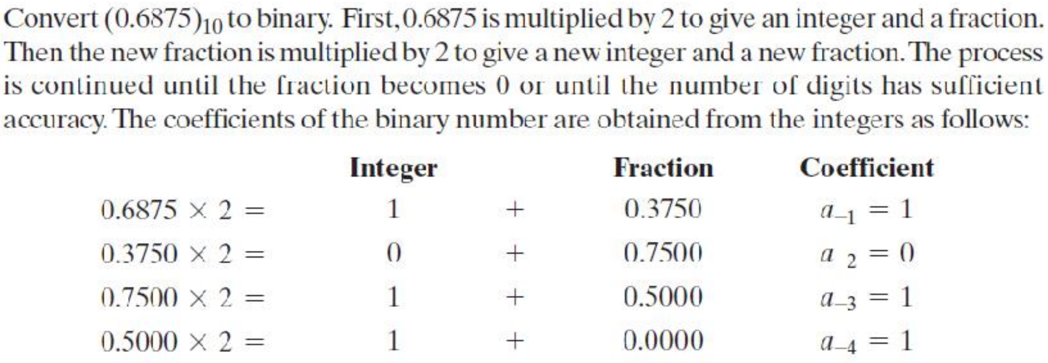
same method used for integers except multiplication is used instead of division.

✓ Convert $(0.8542)_{10}$ to binary (give answer to 6 digits).



0.8542 x 2 =	1	+	0.7084	$a_{-1} = 1$	MSB
0.7084 x 2 =	1	+	0.4168	$a_{-2} = 1$	
0.4168 x 2 =	0	+	0.8336	$a_{-3} = 0$	
0.8336 x 2 –	1	+	0.6672	$a_{-4} – 1$	
0.6675 x 2 =	1	+	0.3344	$a_{-5} = 1$	
0.3344 x 2 =	0	+	0.6688	$a_{-6} = 0$	LSB

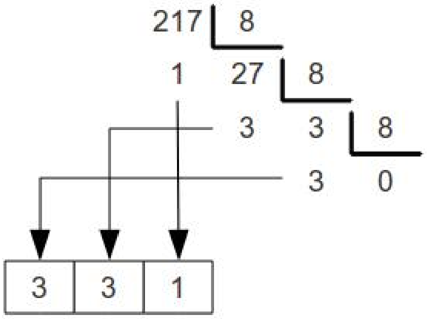
$$(0.8542)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6})_2 = (0.110110)_2$$

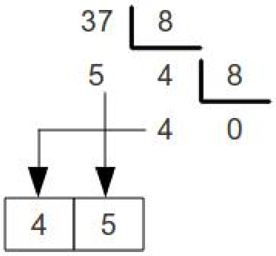


Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. Then the new fraction is multiplied by 2 to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficients of the binary number are obtained from the integers as follows:

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

• Conversion from Decimal to Octal





Decimal = 415

Division	Quotient	Remainder
415/8	51	7
51/8	6	3
6/8	0	6

Octal = 637

• Conversion from Decimal to Octal

Decimal to Octal Conversion

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

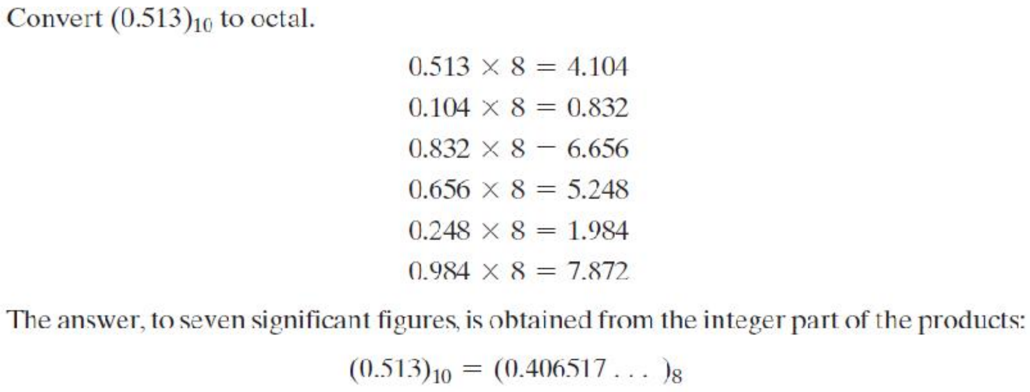
Answer: $(175)_{10} = (a_2\, a_1\, a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Conversion from decimal fraction to Octal:



Convert $(0.513)_{10}$ to octal.

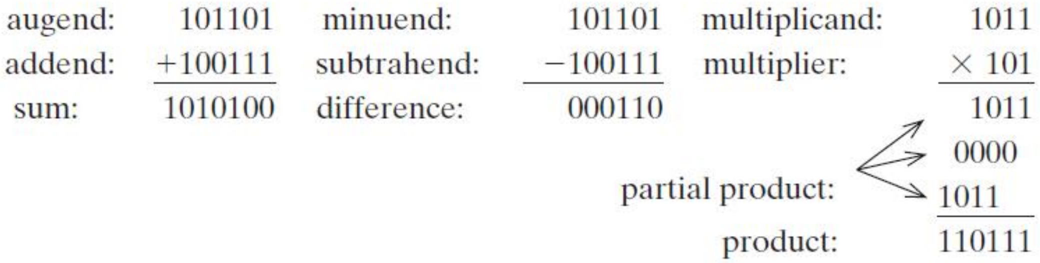
$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 - 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

The answer, to seven significant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517\ldots)_8$$

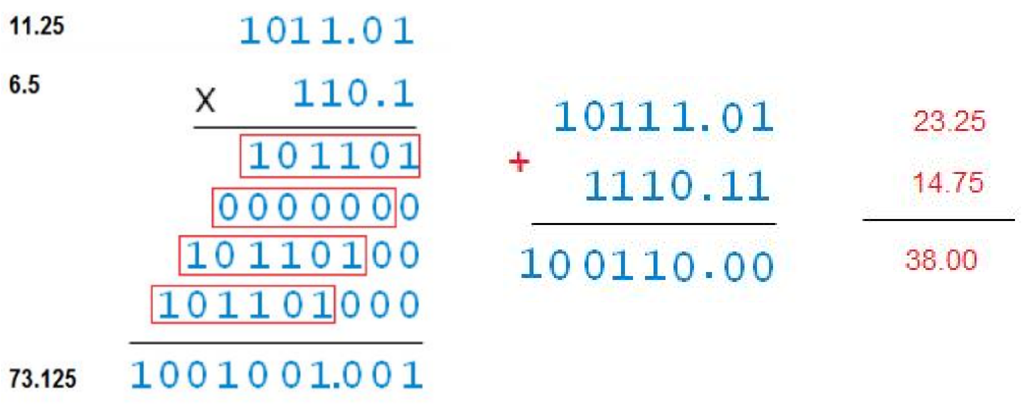
Arithmetic operations in Systems

Arithmetic operations in Binary Number System



augend:	101101	minuend:	101101	multiplicand:	1011
addend:	+100111	subtrahend:	−100111	multiplier:	× 101
sum:	1010100	difference:	000110		1011
					0000
				partial product:	1011
				product:	110111

Arithmetic operations in Binary Number System



Arithmetic operations in Octal Number System

Evaluate:

(i) $(162)_8 + (537)_8$

1 1 <---- carry

1 6 2

5 3 7

7 2 1

Therefore, sum = $(721)_8$

Arithmetic operations in Octal Number System

(iv) $(67.5)_8 + (45.6)_8$

Solution:

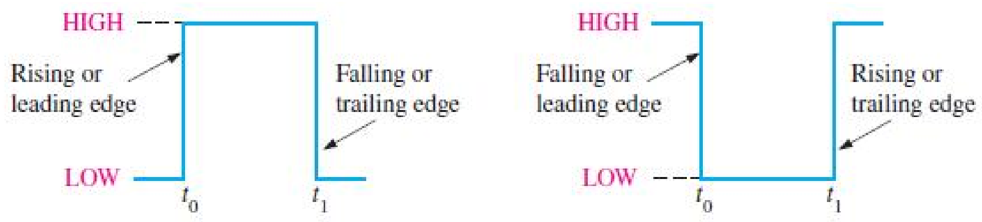
```
  1 1            <---- carry
  6 7 . 5
  4 5 . 6
1 3 5 . 3
```

Therefore, sum = $(135.3)_8$

Binary Digits

Each of the two digits in the **binary** system, 1 and 0, is called a **bit**, which is a contraction of the words *binary digit.* In digital circuits, two different voltage levels are used to represent the two bits. Generally, 1 is represented by the higher voltage, which we will refer to as a HIGH, and a 0 is represented by the lower voltage level, which we will refer to as a LOW. This is called **positive logic** and will be used throughout the book.

HIGH 1 and LOW 0



(a) Positive–going pulse

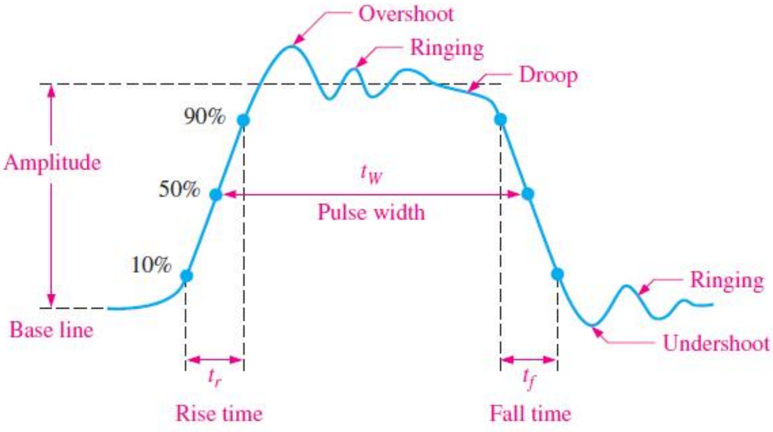
(b) Negative–going pulse

Non Ideal Pulse

rise time (tr),

fall time (t_f)

The **pulse width (t_w)** is a measure of the duration of the pulse and is often defined as the time interval between the 50% points on the rising and falling edges



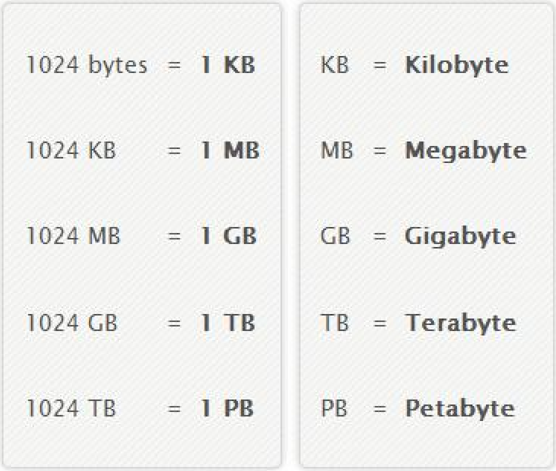
8 Nonideal pulse characteristics.

Binary Digits

1 Byte = 8 bits (Bit→ 0 or 1)

enough to represent one alphanumeric character

1 KB = 2^10 Bytes = 1024



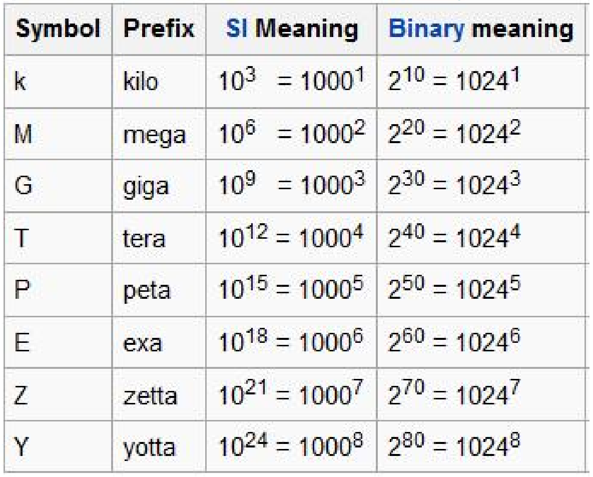
1024 bytes	=	**1 KB**	KB	=	**Kilobyte**
1024 KB	=	**1 MB**	MB	=	**Megabyte**
1024 MB	=	**1 GB**	GB	=	**Gigabyte**
1024 GB	=	**1 TB**	TB	=	**Terabyte**
1024 TB	=	**1 PB**	PB	=	**Petabyte**

Systematic multiples

the International System of Units (SI).

memory standards

JEDEC memory standards



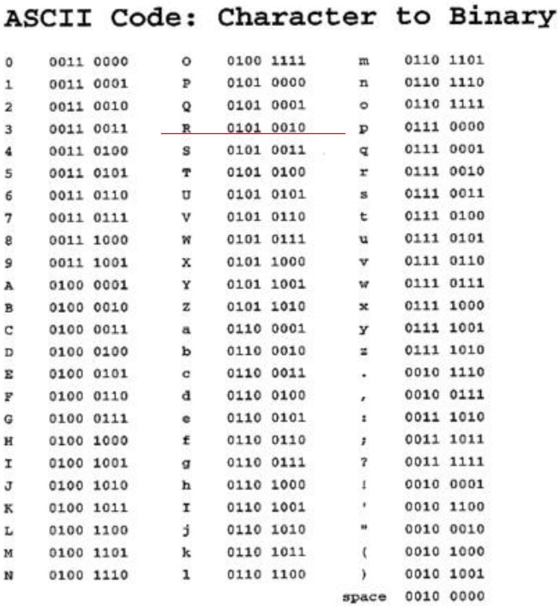
Symbol	Prefix	SI Meaning	Binary meaning
k	kilo	$10^3 = 1000^1$	$2^{10} = 1024^1$
M	mega	$10^6 = 1000^2$	$2^{20} = 1024^2$
G	giga	$10^9 = 1000^3$	$2^{30} = 1024^3$
T	tera	$10^{12} = 1000^4$	$2^{40} = 1024^4$
P	peta	$10^{15} = 1000^5$	$2^{50} = 1024^5$
E	exa	$10^{18} = 1000^6$	$2^{60} = 1024^6$
Z	zetta	$10^{21} = 1000^7$	$2^{70} = 1024^7$
Y	yotta	$10^{24} = 1000^8$	$2^{80} = 1024^8$

Joint Electron Device Engineering Council (JEDEC)

ASCII Code

An alphanumeric character (e.g. a letter or number such as 'A', 'B' or '7') is stored as 1 byte. For example, to store the letter 'R' uses 1 byte, which is stored by the computer as 8 bits, '01010010'.

ASCII (American Standard Code for Information Interchange) Code



ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(	0010 1000
N	0100 1110	l	0110 1100	)	0010 1001
				space	0010 0000

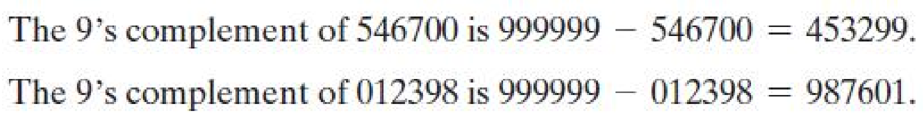
Complements of Numbers

Why? -→ to simplify the subtraction operation

There are two types of complements for each base¯*r* system: the radix complement and the diminished radix complement. The first is referred to as
the *r*’s complement and the second as the (*r* - 1)>s complement. When the value of the base *r* is substituted in the name,
the two types are referred to as **the 2’s complement** and **1’s complement** for binary numbers and the 10’s complement and 9’s complement for
decimal numbers.

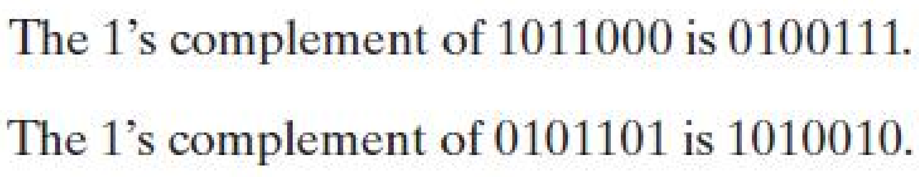
Complements of Numbers

The (*r* - 1)>s complement.



The 9’s complement of 546700 is 999999 − 546700 = 453299.

The 9’s complement of 012398 is 999999 − 012398 = 987601.



The 1’s complement of 1011000 is 0100111.

The 1’s complement of 0101101 is 1010010.

Invert the binary digits

Complements of Numbers

The *r*’s complement.

the 10’s complement of 012398 is 987602

and

the 10’s complement of 246700 is 753300

The 9’s complement + 1

Or the 10’s complement of the first digit then the 9’s for the reaming digits.

the 2’s complement of 1101100 is 0010100

and

the 2’s complement of 0110111 is 1001001

The 1’s complement + 1

Or keep the first One (not the first digit) from left then Invert all digits (after).

Subtraction with Complements

The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r’s complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r’s complement of $(N - M)$. To obtain the answer in a familiar form, take the r’s complement of the sum and place a negative sign in front.

The following examples illustrate the procedure:

Subtraction with Complements

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction **(a)** $X - Y$ and **(b)** $Y - X$ by using 2's complements.

$$\begin{array}{llrrr}
\text{(a)} & & X = & 1010100 & 84 \\
 & \text{2's complement of } Y = & + & \underline{0111101} & 67 \\
 & & \text{Sum} = & 10010001 & \\
 & \text{Discard end carry } 2^7 = & - & \underline{10000000} & \\
 & & \textit{Answer}: X - Y = & 0010001 & 17
\end{array}$$

Subtraction with Complements

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction **(a)** $X - Y$ and **(b)** $Y - X$ by using 2's complements.

$$\begin{array}{llrr}
\text{(b)} & & Y = & 1000011 \\
 & \text{2's complement of } X = & + & \underline{0101100} \\
 & & \text{Sum} = & 1101111
\end{array}$$

There is no end carry. Therefore, the answer is *Y* - *X* = -(2>s complement of 1101111) = -0010001

Signed Binary Numbers

1- the signed-magnitude

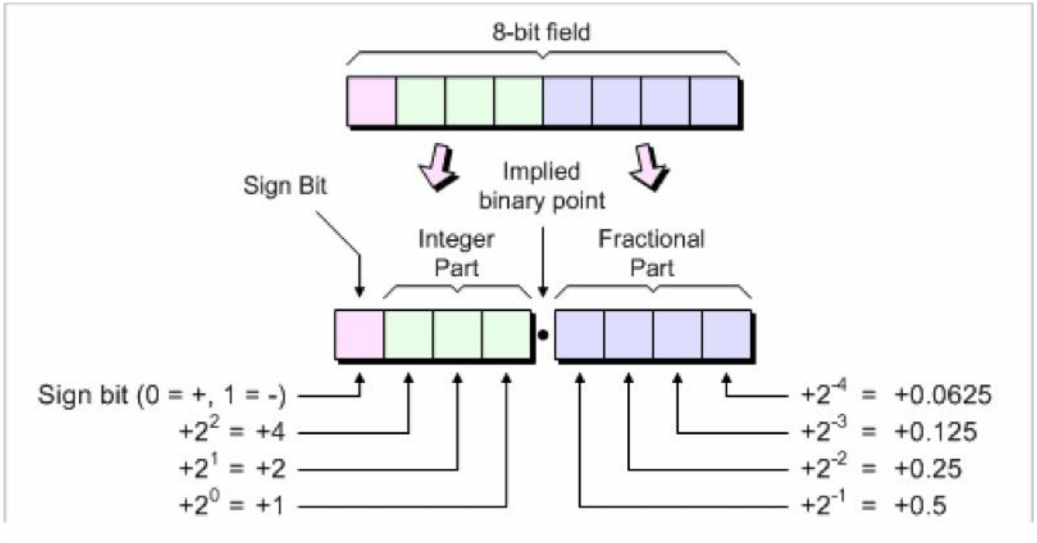
The representation: the sign with a bit placed in the leftmost position of the number. The convention is to make the sign **bit 0** for **positive** and **1** for **negative**.

For example,
the string of bits 01001 can be considered as 9 (unsigned binary) or as +9 (signed binary)

The string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned number and the binary equivalent of -9 when considered as a signed number

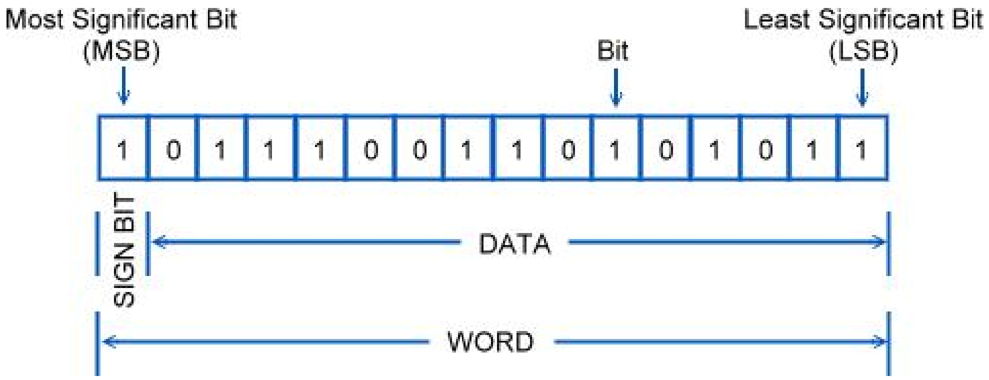
referred to as the *signed-magnitude*

Signed Binary Numbers



8-bit sign-magnitude binary

Signed Binary Numbers

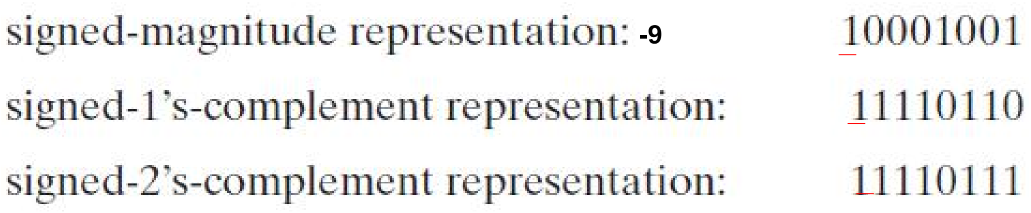


16-bit Sign-magnitude Binary

2- the signed-Complement system

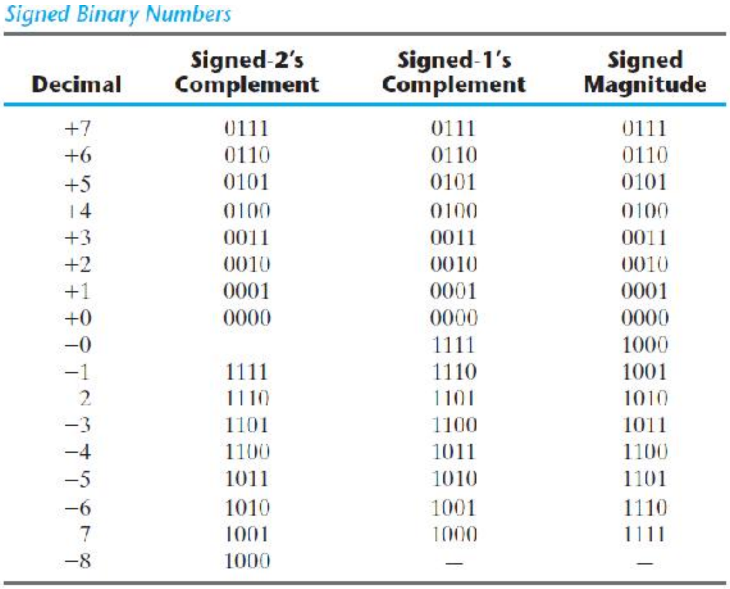
The complement will always start with a 1, indicating a negative number. The signed-complement system can use either the 1's or the 2's complement, but the 2's complement is the most common.

there are three different ways to represent -9 with eight bits



signed-magnitude representation: -9	10001001
signed-1's-complement representation:	11110110
signed-2's-complement representation:	11110111

Signed Binary Numbers



Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0		1111	1000
−1	1111	1110	1001
2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
7	1001	1000	1111
−8	1000	—	—

Arithmetic Addition

Discard Carry

$$\begin{array}{rr} +\ 6 & 00000110 \\ +13 & 00001101 \\ \hline +19 & 00010011 \end{array} \qquad \begin{array}{rr} -\ 6 & 11111010 \\ +13 & 00001101 \\ \hline +\ 7 & 00000111 \end{array}$$

$$\begin{array}{rr} +\ 6 & 00000110 \\ -13 & 11110011 \\ \hline -\ 7 & 11111001 \end{array} \qquad \begin{array}{rr} -\ 6 & 11111010 \\ -13 & 11110011 \\ \hline -19 & 11101101 \end{array}$$

Arithmetic Subtraction

Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign-bit position is discarded.

$$(\pm A) - (+B) = (\pm A) + (-B);$$
$$(\pm A) - (-B) = (\pm A) + (+B).$$

consider the subtraction (-6) - (-13) = +7.

written as (11111010 - 11110011).

The subtraction is changed to addition by taking the 2's complement of the subtrahend (-13), giving (+13).

11111010 + 00001101 = 100000111. Removing the end carry, we obtain the correct answer: 00000111 (+7).

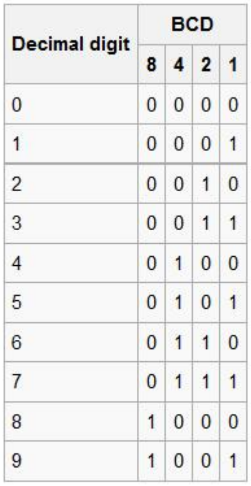
Why Signed Binary Numbers

It is worth noting that binary numbers in the signed¯complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers.

Therefore, **computers need only one common hardware circuit to handle both types of arithmetic.** This consideration has resulted in the signed¯complement system being used in virtually all arithmetic units of computer systems.

Binary Coded Decimal-BCD

each decimal digit is represented by its corresponding four-bit binary value



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

the binary combinations 1010 through 1111 are not used and have no meaning in BCD.

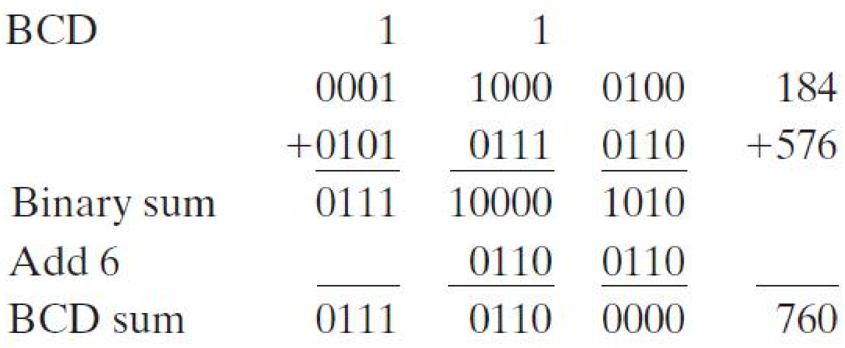
$$(185)_{10} = (0001\ 1000\ 0101)_{\text{BCD}} = (10111001)_2$$

an advantage in the use of decimal numbers, because computer input and output data are generated by people who use the decimal system.

BCD Addition

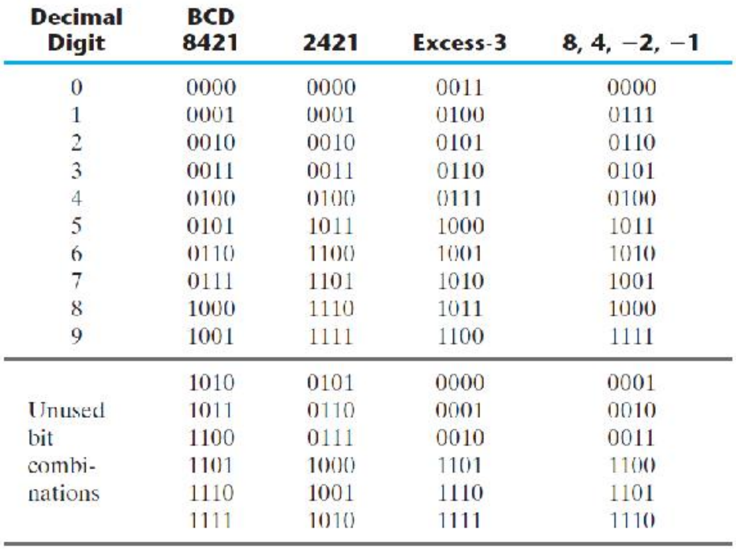
$$\begin{array}{rrrrrr}
4 & 0100 & 4 & 0100 & 8 & 1000 \\
+5 & +0101 & +8 & +1000 & +9 & 1001 \\
\hline
9 & 1001 & 12 & 1100 & 17 & 10001 \\
 & & & +0110 & & +0110 \\
\cline{4-4}\cline{6-6}
 & & & 10010 & & 10111
\end{array}$$

BCD Addition



BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

Different Binary Codes

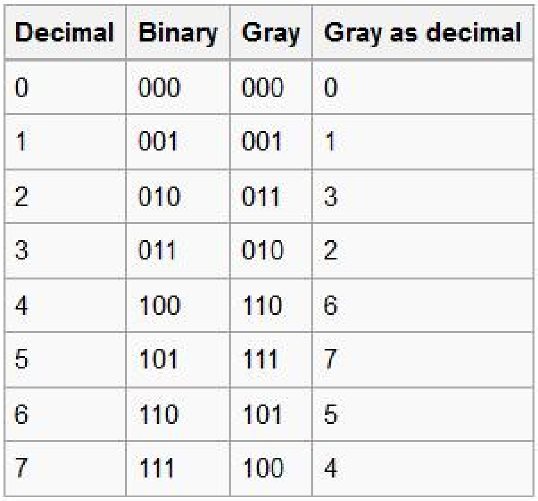


Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010 1011 1100 1101 1110 1111	0101 0110 0111 1000 1001 1010	0000 0001 0010 1101 1110 1111	0001 0010 0011 1100 1101 1110

Gray Code

The **reflected binary code** (**RBC**)

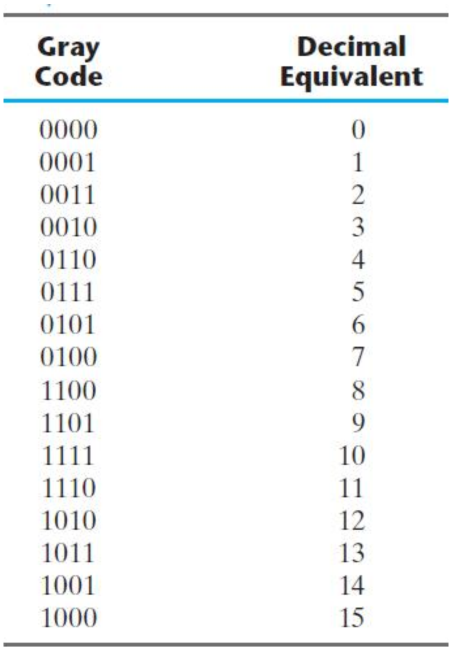
advantage of the Gray code over the straight binary number sequence is that only one bit in the code group changes in going from one number to the next.



Decimal	Binary	Gray	Gray as decimal
0	000	000	0
1	001	001	1
2	010	011	3
3	011	010	2
4	100	110	6
5	101	111	7
6	110	101	5
7	111	100	4

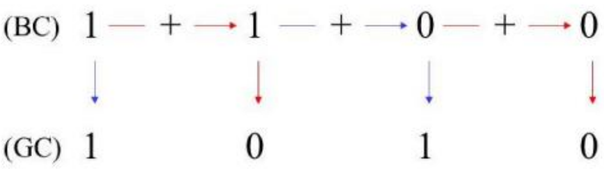
Gray Code

The **reflected binary code** (**RBC**)



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

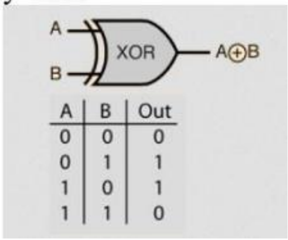
Binary to Gray Code Conversion



(BC) 1 — + → 1 — + → 0 — + → 0

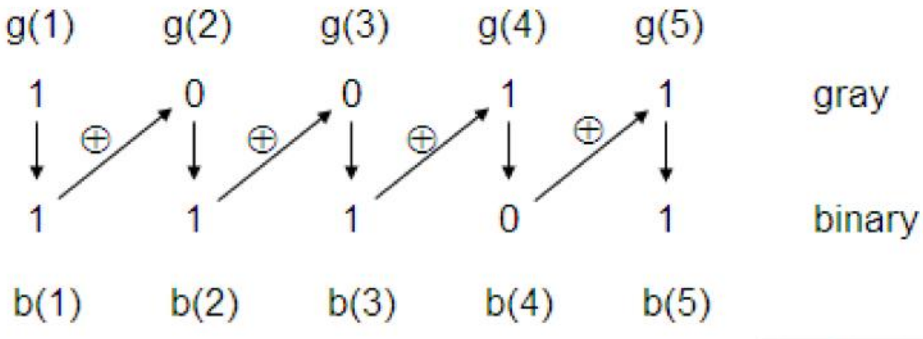
(GC) 1 0 1 0

- MSB does not change as a result of conversion
- Start with MSB of binary number and add it to neighboring binary bit to get the next Gray code bit
- Repeat for subsequent Gray coded bits



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

Gray Code to Binary Conversion



g(1) g(2) g(3) g(4) g(5)

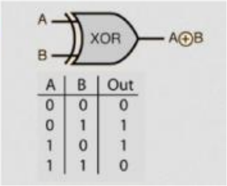
1 0 0 1 1 gray

⊕ ⊕ ⊕ ⊕

1 1 1 0 1 binary

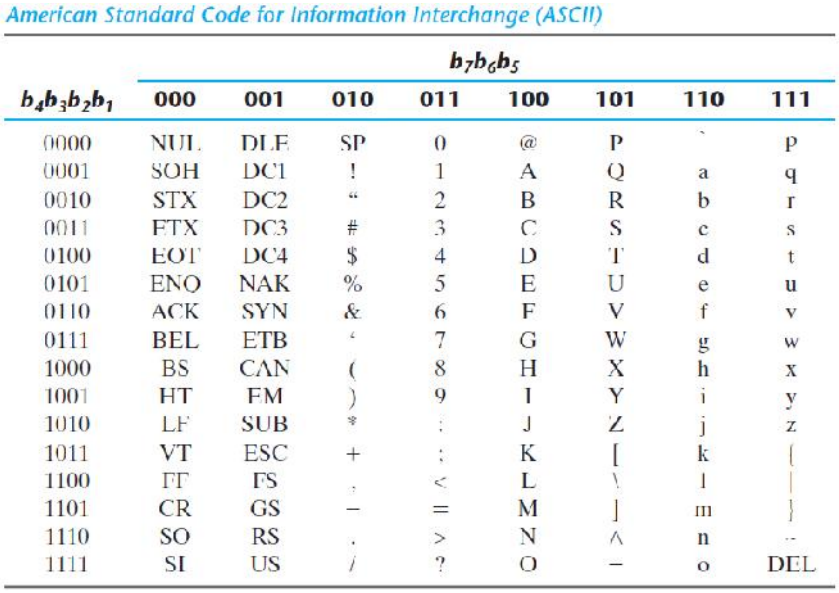
b(1) b(2) b(3) b(4) b(5)

g(1) b(1) xor g(2) b(2) xor g(3) b(3) xor g(4) b(4) xor g(5)



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

ASCII Character Code



American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	–	o	DEL

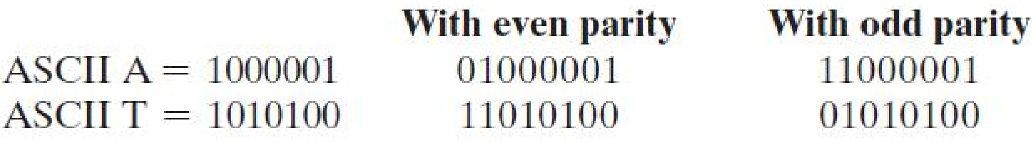
Error-Detecting Code

A *parity bit* is an extra bit included with a message to make the total number of 1's either even or odd.

Even Parity Check →

1- No. of One's Even → 0

2- No. of One's Odd→ 1



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

Odd Parity Check →

1- No. of One's Even → 1

2- No. of One's Odd→ 0

The parity bit is helpful in detecting errors during the transmission of information

BINARY STORAGE AND REGISTERS

Registers

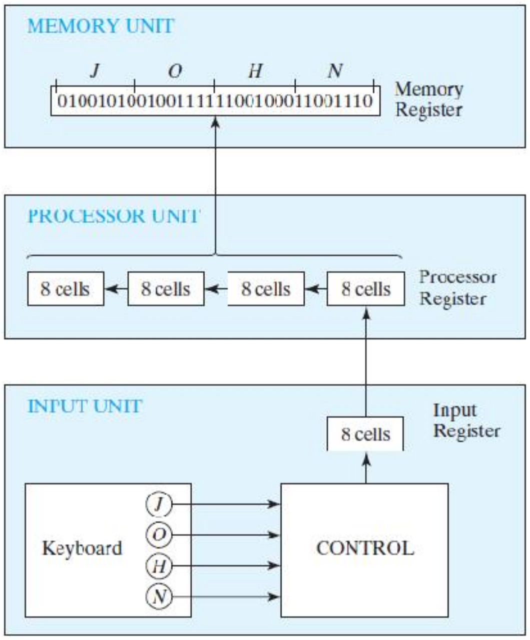
A *register* is a group of binary cells. A register with *n* cells can store any discrete quantity of information that contains *n* bits.

A register with 16 cells can be in one of 2^{16} possible states

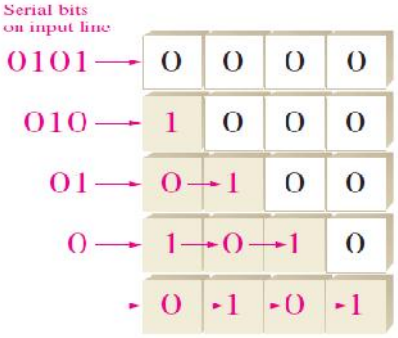
Register Transfer

is a basic operation that consists of a transfer of binary information from one set of registers into another set of registers.

Register Transfer



Register Transfer: Serial I/O



Initially, the register contains only *invalid* data or all zeros as shown here.

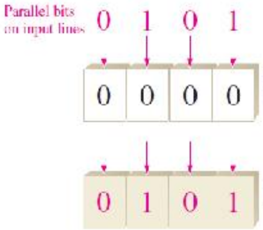
First bit (1) is shifted serially into the register.

Second bit (0) is shifted serially into register and first bit is shifted right.

Third bit (1) is shifted into register and the first and second bits are shifted right.

Fourth bit (0) is shifted into register and the first, second, and third bits are shifted right. The register now stores all four bits and is full.

Register Transfer: Parallel I/O



Initially, the register is empty, containing only nondata zeros.

All bits are shifted in and stored simultaneously.

BINARY LOGIC

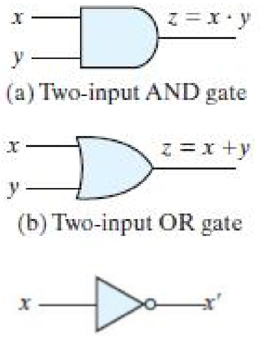
Binary logic consists of binary variables and a set of logical operations. The variables are designated by letters of the alphabet, such as *A*, *B*, *C*, *x*, *y*, *z*, etc., with each variable having two and only two distinct possible values: 1 and 0. There are three basic logical operations:

AND, OR, and NOT.

Truth Tables of Logical Operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

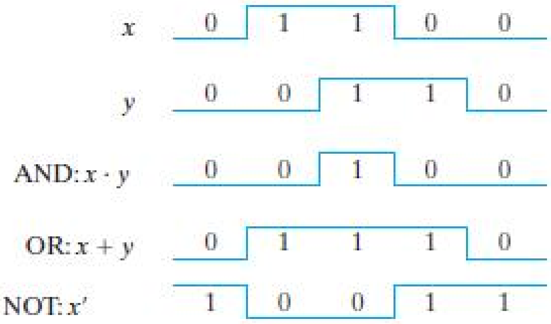
Logic Gates

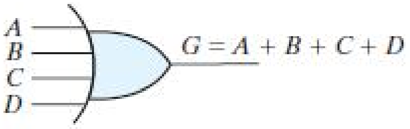


(a) Two-input AND gate

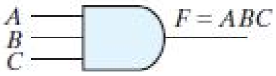
(b) Two-input OR gate

(c) NOT gate or inverter



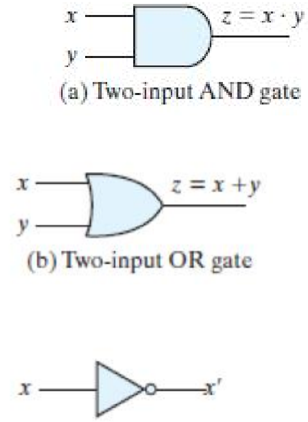


(b) Four-input OR gate



(a) Three-input AND gate

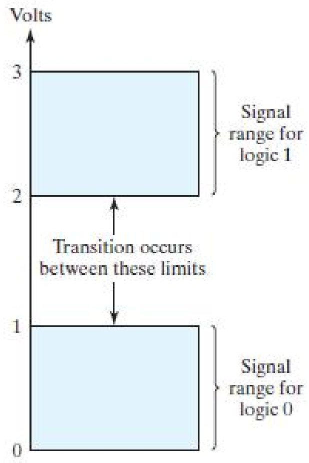
Logic Gates

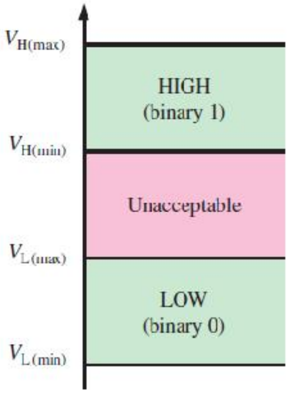


(a) Two-input AND gate

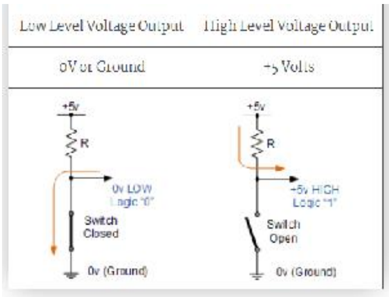
(b) Two-input OR gate

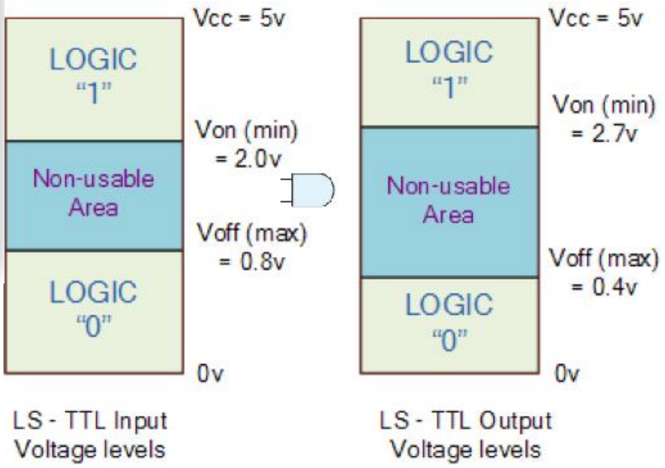
(c) NOT gate or inverter

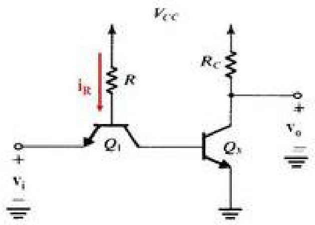




Transistor Transistor Logic (TTL)

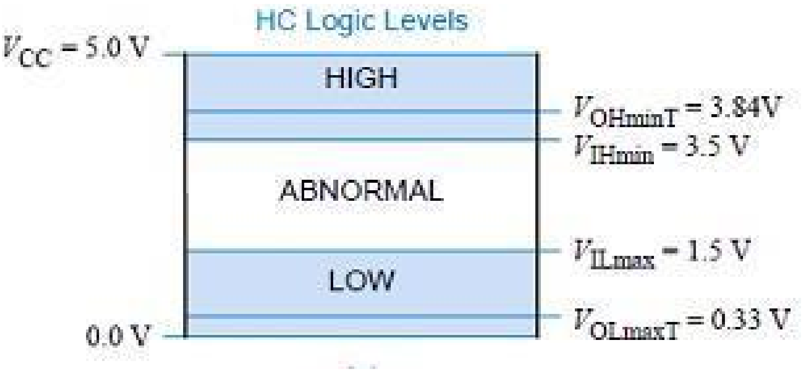


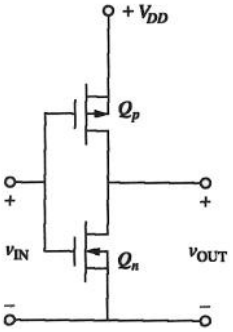




Low-power Schottky TTL

Complementary metal–oxide–semiconductor (CMOS)





CMOS Inverter

Boolean Algebra

1. (a) The structure is closed with respect to the operator $+$.
(b) The structure is closed with respect to the operator $\cdot$.

2. (a) The element 0 is an identity element with respect to $+$; that is, $x + 0 = 0 + x = x$.
(b) The element 1 is an identity element with respect to $\cdot$; that is, $x \cdot 1 = 1 \cdot x = x$.

3. (a) The structure is commutative with respect to $+$; that is, $x + y = y + x$.
(b) The structure is commutative with respect to $\cdot$; that is, $x \cdot y = y \cdot x$.

4. (a) The operator $\cdot$ is distributive over $+$; that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
(b) The operator $+$ is distributive over $\cdot$; that is, $x + (y \cdot z) = (x + y) \cdot (x - z)$.

5. For every element $x \in B$, there exists an element $x' \in B$ (called the *complement* of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.

6. There exist at least two elements $x, y \in B$ such that $x \neq y$.

Boolean Algebra

Identity

$$0 + x = x \qquad 1 \cdot x = x$$

Commutation

$$x \cdot y = y \cdot x \qquad x + y = y + x$$

Distribution

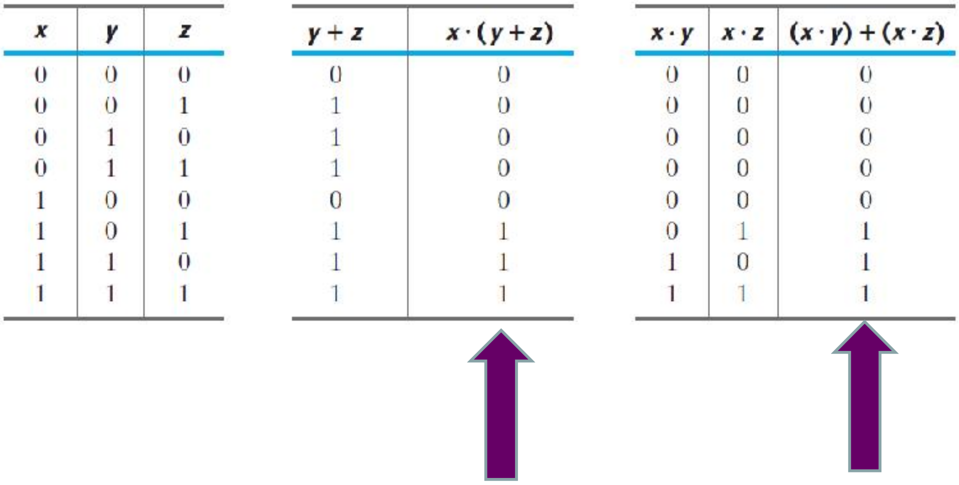
$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

Complement

$$x + x' = 1 \qquad x \cdot x' = 0$$

Boolean Algebra



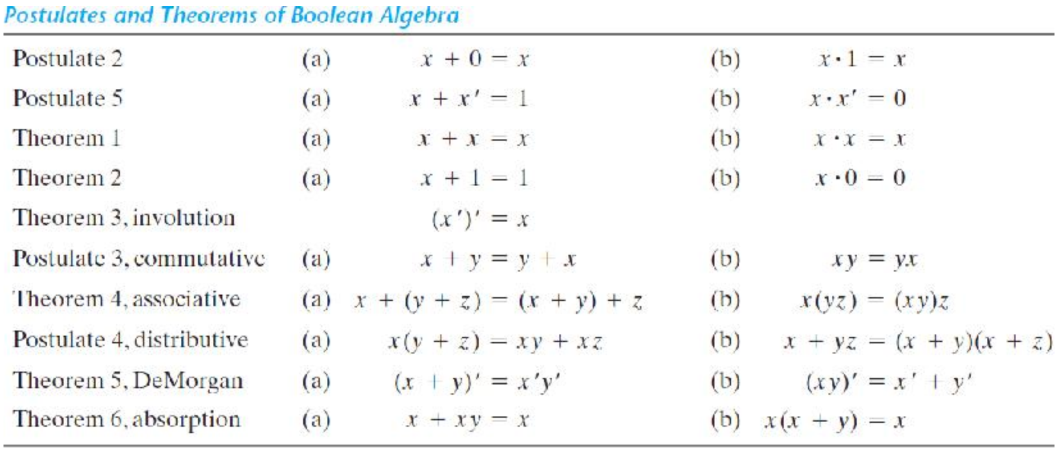
x	y	z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$y+z$	$x\cdot(y+z)$
0	0
1	0
1	0
1	0
0	0
1	1
1	1
1	1

$x\cdot y$	$x\cdot z$	$(x\cdot y)+(x\cdot z)$
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	1	1
1	0	1
1	1	1

Basic Theorems

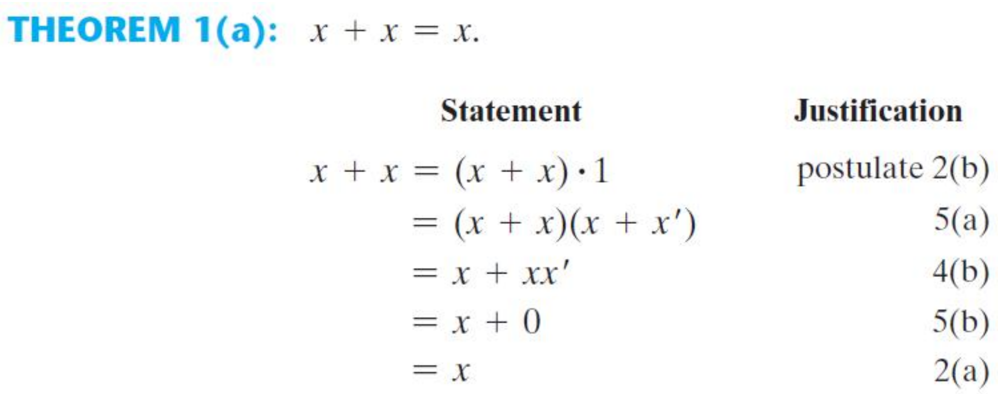
Duality



Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

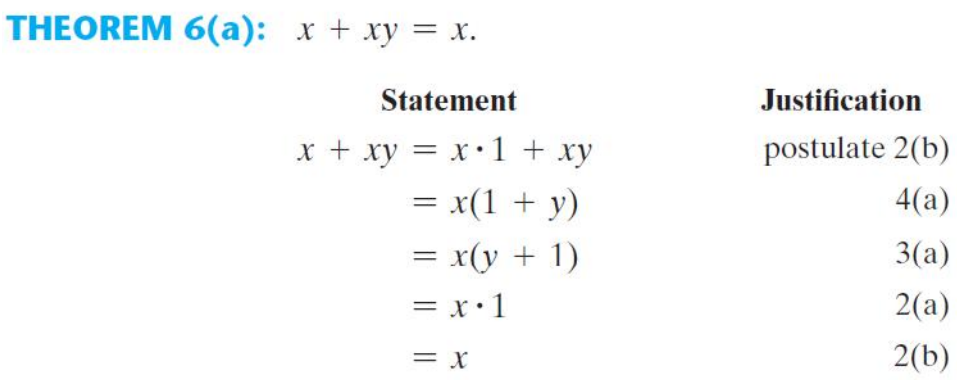
Basic Theorems



THEOREM 1(a): $x + x = x.$

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

Basic Theorems



THEOREM 6(a): $x + xy = x.$

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

Basic Theorems

DeMorgan's **theorem**

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

Using Duality

De Morgan's Theorem

$$\overline{x\cdot y} = \bar{x} + \bar{y} \quad \Rightarrow \quad x\cdot y = \overline{\bar{x} + \bar{y}}$$

$$\overline{x+y} = \bar{x}\cdot\bar{y} \quad \Rightarrow \quad x+y = \overline{\bar{x}\cdot\bar{y}}$$

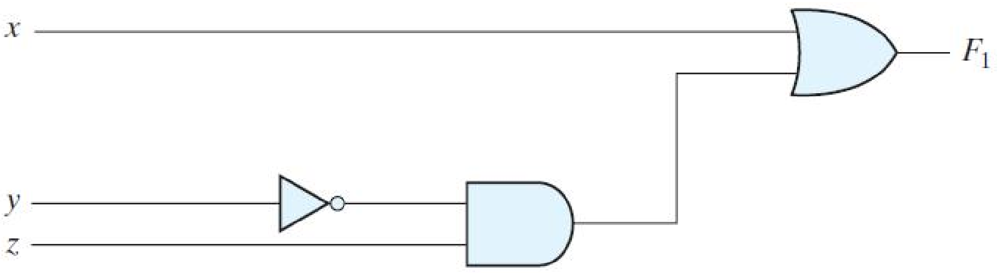
x	y	$x + y$	$(x + y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x'	y'	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

Boolean Function

$$F_1 = x + y'z$$

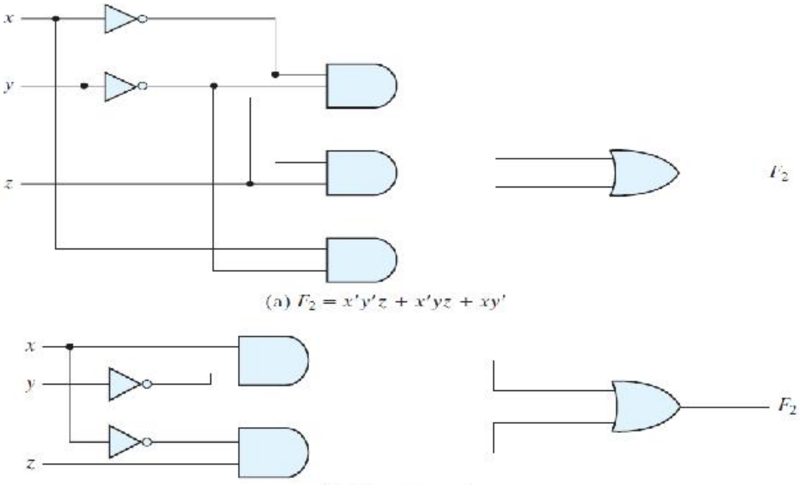
x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Simplifications of Function

$$F_2 = x'y'z + x'yz + xy'$$

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



(a) $F_2 = x'y'z + x'yz + xy'$

(b) $F_2 = xy' + x'z$

Simplifications of Function

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$

2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$

3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$

4. $xy + x'z + yz = xy + x'z + yz(x + x')$

$$= xy + x'z + xyz + x'yz$$
$$= xy(1 + z) + x'z(1 + y)$$
$$= xy + x'z.$$

5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

Complement of a Function

The complement of a function *F* is *F’*

$$\begin{aligned}
(A + B + C)' &= (A + x)' && \text{let } B + C = x \\
&= A'x' && \text{by theorem 5(a) (DeMorgan)} \\
&= A'(B + C)' && \text{substitute } B + C = x \\
&= A'(B'C') && \text{by theorem 5(a) (DeMorgan)} \\
&= A'B'C' && \text{by theorem 4(b) (associative)}
\end{aligned}$$

Complement of a Function

These theorems can be generalized as follows:

$$(A + B + C + D + \cdots + F)' = A'B'C'D' \ldots F'$$

$$(ABCD \ldots F)' = A' + B' + C' + D' + \cdots + F'$$

Find the complement of the functions $F_1 = x'yz' + x'y'z$

$$\begin{aligned}
F_1' = (x'yz' + x'y'z)' &= (x'yz')'(x'y'z)' \\
&= (x + y' + z)(x + y + z')
\end{aligned}$$

Canonical and Standard Forms

Minterms and Maxterms

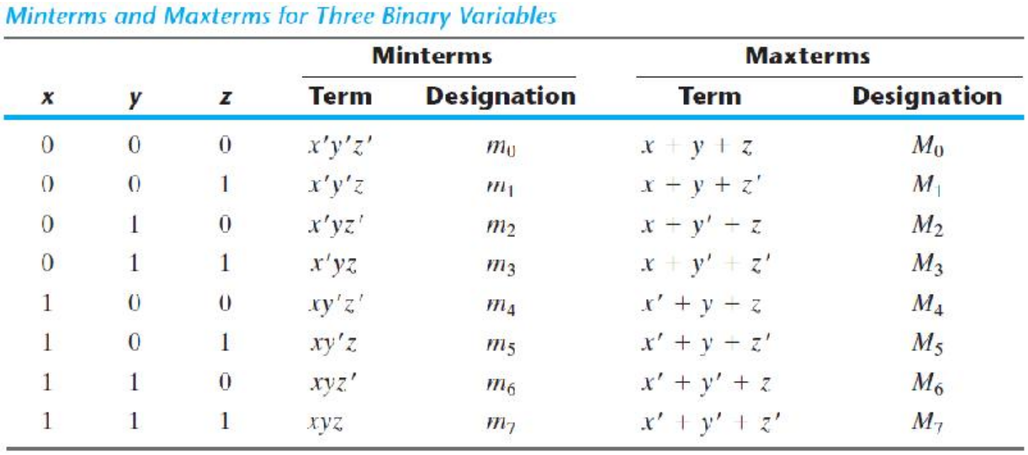
A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

Minterms or Standard Products
Function → (Sum of Product)

Maxterms or Standard Sums
Function → (Product of Sums)

Canonical and Standard Forms

Minterms and Maxterms



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

Hint: Minterms are the complements of the Maxterms or vice versa

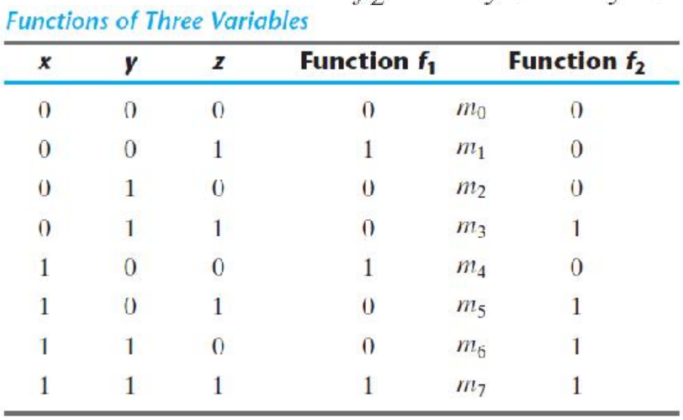
Canonical and Standard Forms

Minterms or Standard Products

For example

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

$$f_2 = x'yz + xy'z + xyz' + xyz$$



Functions of Three Variables

x	y	z	Function f_1		Function f_2
0	0	0	0	m_0	0
0	0	1	1	m_1	0
0	1	0	0	m_2	0
0	1	1	0	m_3	1
1	0	0	1	m_4	0
1	0	1	0	m_5	1
1	1	0	0	m_6	1
1	1	1	1	m_7	1

$$f_1 = m_1 + m_4 + m_7 \qquad f_2 = m_3 + m_5 + m_6 + m_7$$

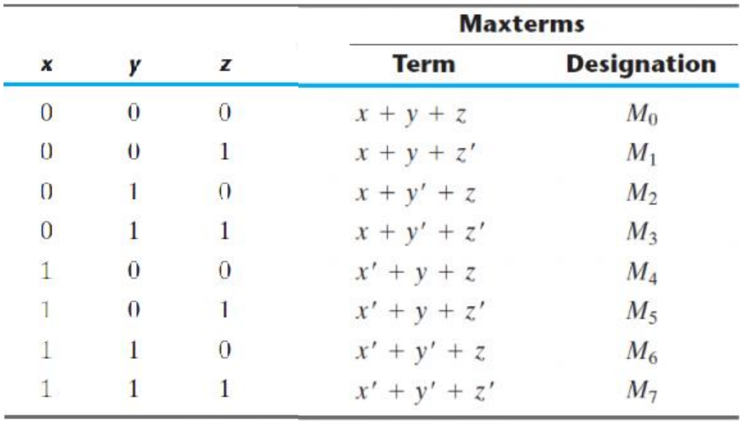
Canonical and Standard Forms

Maxterms or Standard Sums

For example $f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$

The complement of f_1 is read as

$$f_1 = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z)$$



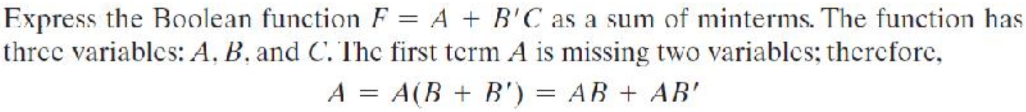
			Maxterms	
x	y	z	Term	Designation
0	0	0	$x + y + z$	M_0
0	0	1	$x + y + z'$	M_1
0	1	0	$x + y' + z$	M_2
0	1	1	$x + y' + z'$	M_3
1	0	0	$x' + y + z$	M_4
1	0	1	$x' + y + z'$	M_5
1	1	0	$x' + y' + z$	M_6
1	1	1	$x' + y' + z'$	M_7

$$f_1 = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

Canonical and Standard Forms

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

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table



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,

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

Example

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 *BC* is missing one variable; hence,

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

Combining all terms, we have But $AB'C$ appears twice $(x + x = x)$

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

$$F = A'B'C + AB'C + AB'C + ABC' + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7$$

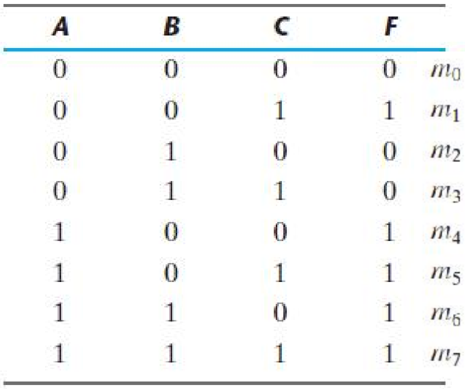
$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

The summation symbol stands for the ORing of terms

Another solution

An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

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



A	B	C	F	
0	0	0	0	m_0
0	0	1	1	m_1
0	1	0	0	m_2
0	1	1	0	m_3
1	0	0	1	m_4
1	0	1	1	m_5
1	1	0	1	m_6
1	1	1	1	m_7

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

Example : Maxterms

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

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

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

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

Example : Maxterms

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

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0M_2M_4M_5 \end{aligned}$$

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

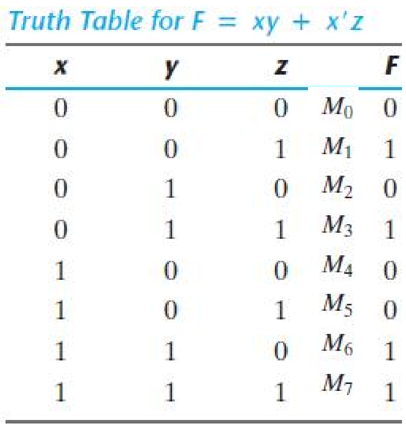
The product symbol denotes the ANDing of maxterms

The maxterms whose product defines the Boolean function are those which give the 0's of the function in a truth table

Example : Maxterms

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

Hint. An alternative procedure for deriving the maxterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the maxterms from the truth table.



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

x	y	z		F
0	0	0	M_0	0
0	0	1	M_1	1
0	1	0	M_2	0
0	1	1	M_3	1
1	0	0	M_4	0
1	0	1	M_5	0
1	1	0	M_6	1
1	1	1	M_7	1

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

Conversion between Canonical Forms

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

This function has a complement that can be expressed as

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of *F* by DeMorgan's theorem, we obtain *F* in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

$$m_j' = M_j$$

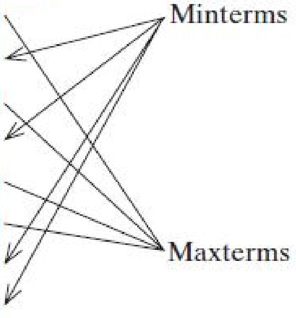
the maxterm with subscript *j* is a complement of the minterm with the same subscript *j* and vice versa.

Minterms and Maxterms of Function F

$$F = xy + x'z$$

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

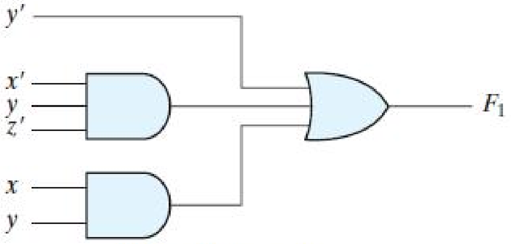
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$$F(x, y, z) = \Pi(0, 2, 4, 5) \qquad F(x, y, z) = \Sigma(1, 3, 6, 7)$$

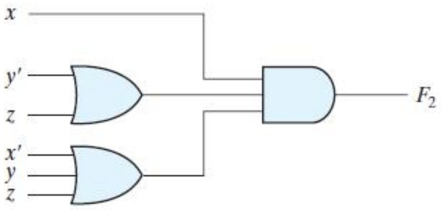
Minterms and Maxterms of Functions

$$F_1 = y' + xy + x'yz'$$



(a) Sum of Products

$$F_2 = x(y' + z)(x' + y + z')$$

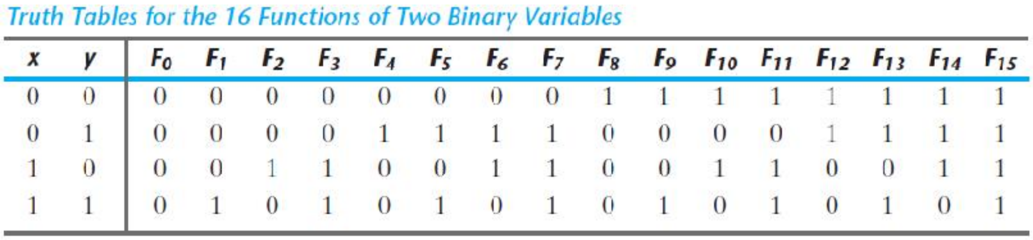


(b) Product of Sums

CH 2

Boolean Algebra and Logic Gates

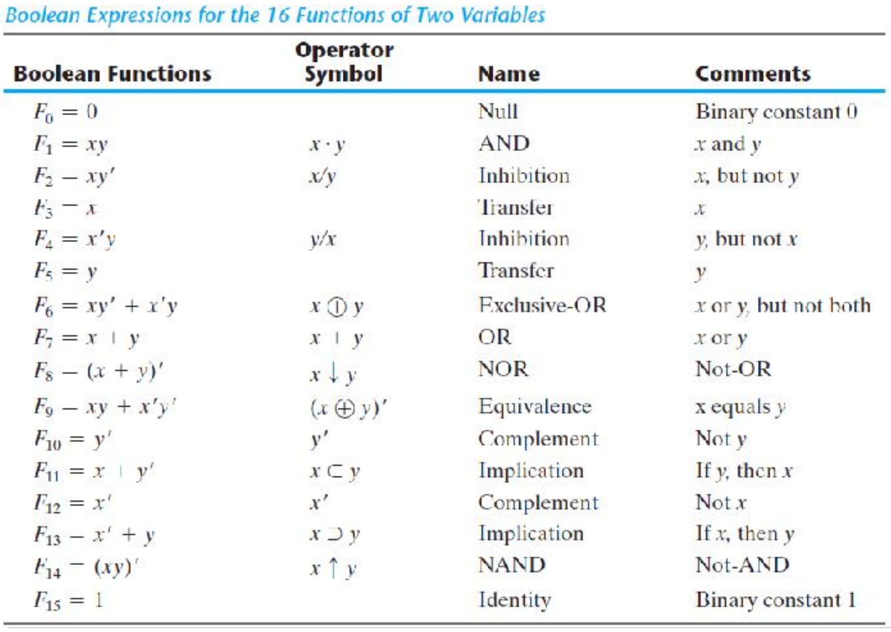
2.7 Other Logic Operations



Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

2.7 Other Logic Operations

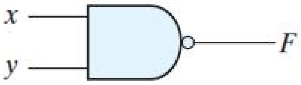


Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Logic Gates

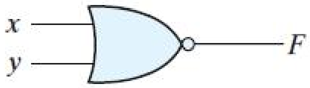
NAND



$F = (xy)'$

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

NOR

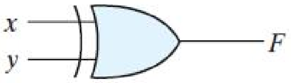


$F = (x + y)'$

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

Logic Gates

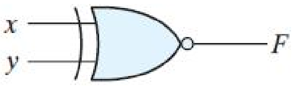
Exclusive-OR
(XOR)



$$F = xy' + x'y$$
$$= x \oplus y$$

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

Exclusive-NOR
or
equivalence



$$F = xy + x'y'$$
$$= (x \oplus y)'$$

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

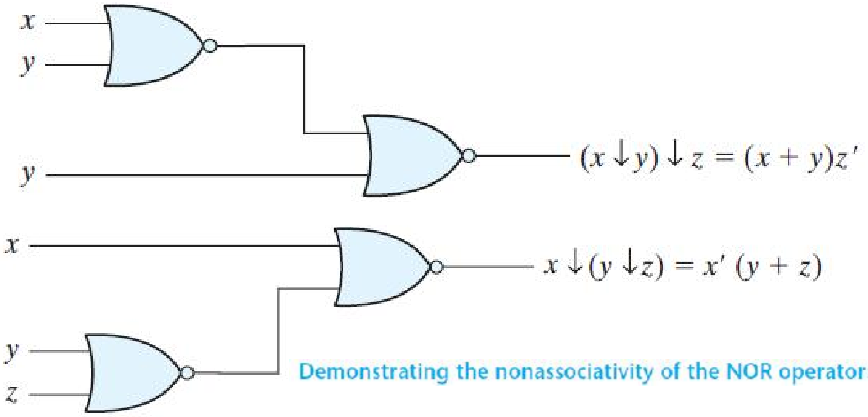
Logic Gates

The NAND and NOR operators are not associative

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

Logic Gates

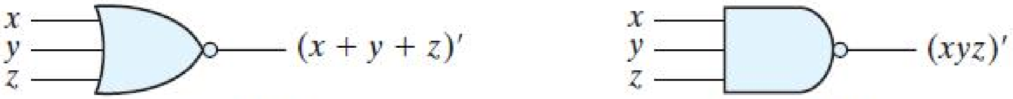

x
y
y
$(x \downarrow y) \downarrow z = (x + y)z'$
x
$x \downarrow (y \downarrow z) = x'(y + z)$
y
z

Demonstrating the nonassociativity of the NOR operator:

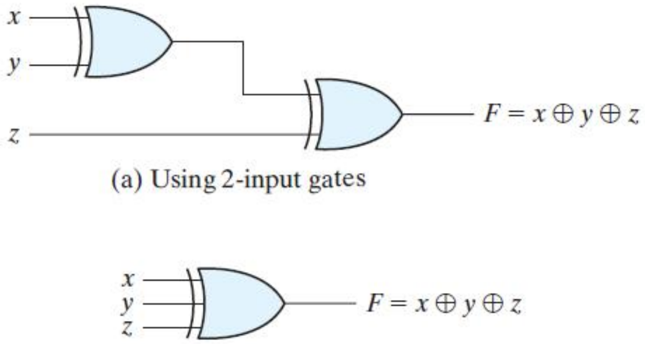
$$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$$

The exclusive-OR and equivalence gates are both commutative and associative and can be extended to more than two inputs.

Logic Gates



(a) 3-input NOR gate (b) 3-input NAND gate



(a) Using 2-input gates

(b) 3-input gate

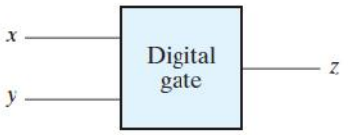
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(c) Truth table

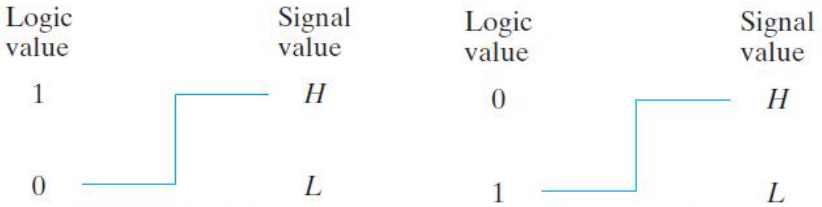
Signal assignment and logic polarity

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

(a) Truth table with H and L



(b) Gate block diagram



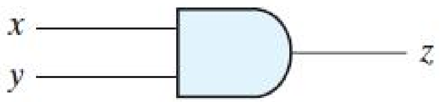
(a) Positive logic (b) Negative logic

a positive logic system **a negative logic system.**

Signal assignment and logic polarity

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

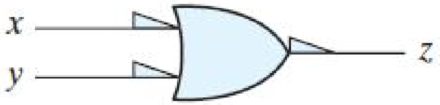
(c) Truth table for positive logic



(d) Positive logic AND gate

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

(e) Truth table for negative logic



(f) Negative logic OR gate

An integrated circuit (IC)

Small-scale integration (SSI) devices → Up to 10 gates in a single package

Medium-scale integration (MSI) → complexity of approximately 10 to 1,000 gates in a single package

Large-scale integration (LSI) devices → contain thousands of gates in a single package

Very large-scale integration (VLSI) devices → contain millions of gates within a single package.

Logic Families

Many different logic families of digital integrated circuits

TTL transistor–transistor logic;

ECL emitter¯coupled logic; **high-speed operation**

MOS metal¯oxide semiconductor; **high component density**

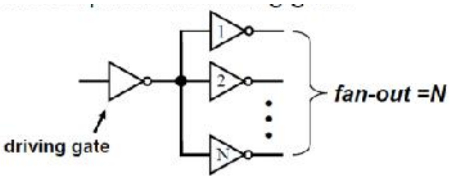
CMOS complementary metal¯oxide semiconductor.
low power consumption

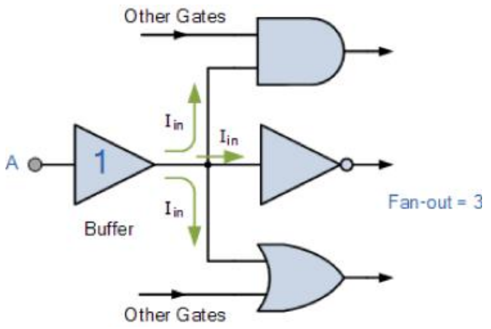
Low power consumption is essential for VLSI design

Basic Definitions

Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation.

Fan-in is the number of inputs available in a gate.

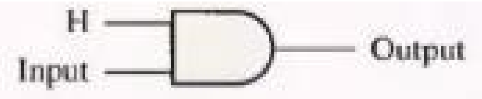


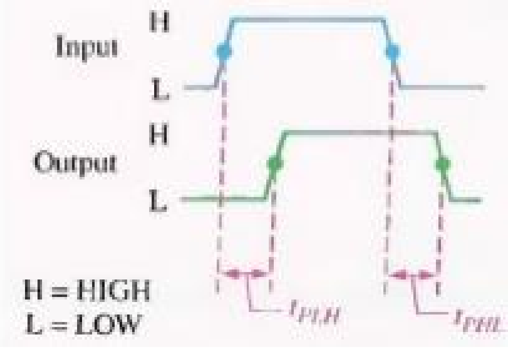


Basic Definitions

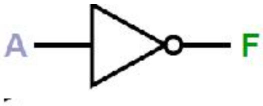
Power dissipation is the power consumed by the gate that must be available from the power supply.

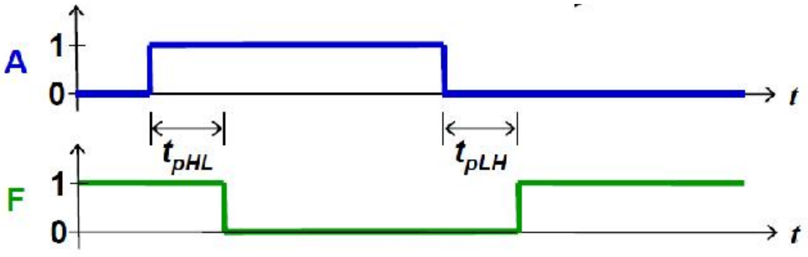
Propagation delay is the average transition delay time for a signal to propagate from input to output.





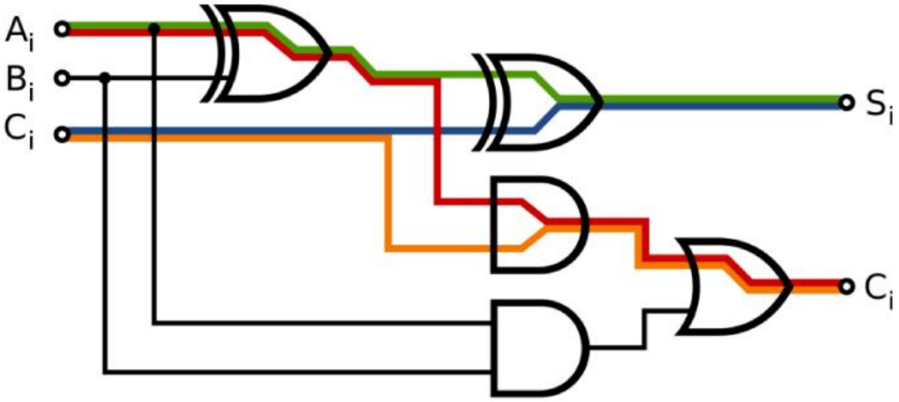
Basic Definitions





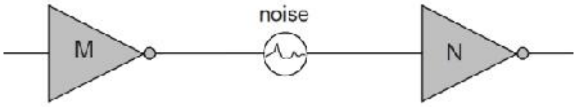
propagation delay $t_p = \left(t_{pHL} + t_{pLH}\right)/2$

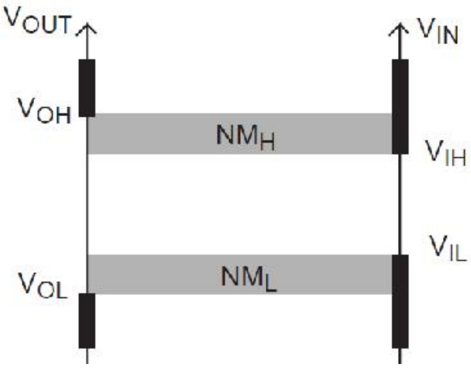
Propagation delay



Noise margin

Noise margin is the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.





$NM_{II} \equiv V_{OII} - V_{III}$ noise margin high

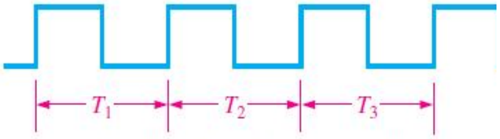
$NM_{L} \equiv V_{IL} - V_{OL}$ noise margin low

Standard Load and Waveform

A standard load is usually defined as the amount of current needed by an input of another similar gate in the same family.

Waveform Characteristics

the pulse width

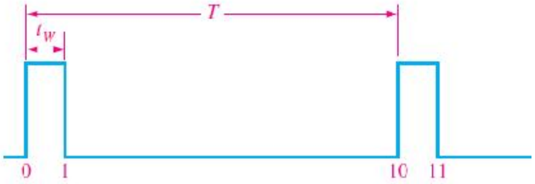


Period = $T_1 = T_2 = T_3 = \ldots = T_n$

Frequency = $\frac{1}{T}$

(a) Periodic (square wave)

$$f = \frac{1}{T}$$



duty cycle, which is the ratio of the pulse width (t_W) to the period (T).

$$\text{Duty cycle} = \left(\frac{t_W}{T}\right)100\%$$

computer-aided design (CAD)

The designer can choose between
1- application-specific integrated circuit (ASIC),
2- field-programmable gate array (FPGA),
With each of these devices comes a set of CAD tools that provide the necessary software to facilitate the hardware fabrication of the unit.

An important development in the design of digital systems is the use of a hardware description language (HDL).

1- HDLs—Verilog
2- VHDL (VHSIC Hardware Description Language)
Very High Speed Integrated Circuit
have been approved as standards by the Institute of Electronics and Electrical Engineers (IEEE) and are in use by design teams worldwide

Minimization of Logic Functions

We have chips with millions of gates

–Why care about minimizing a function?

–What do a few gates matter?

Basic logic functions replicated thousands of times

–Saving one gate for a memory cell pays off

What is the criterion for "minimization"

Should we minimize

- •Number of product terms?
- •Number of logic operations?
- •Number of variables (literals)?
- •Number of wires?
- •…?

systematic approach to minimize expression

Karnaugh maps (K-maps)

For implementation: minimize number of gates

Karnaugh Maps

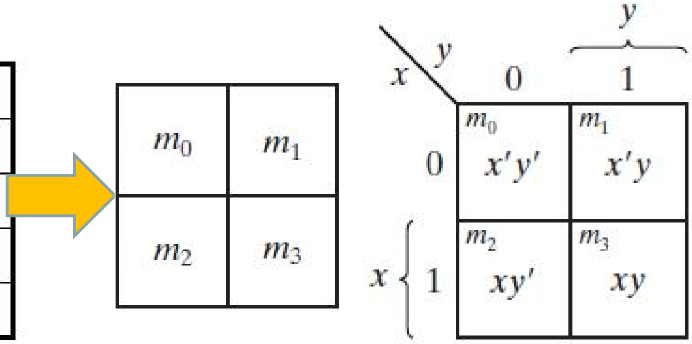
• Karnaugh maps (K-maps) are *graphical* representations of Boolean functions.

• One *map cell* corresponds to a row in the truth table.

• Also, one map cell corresponds to a minterm or a maxterm in the boolean expression

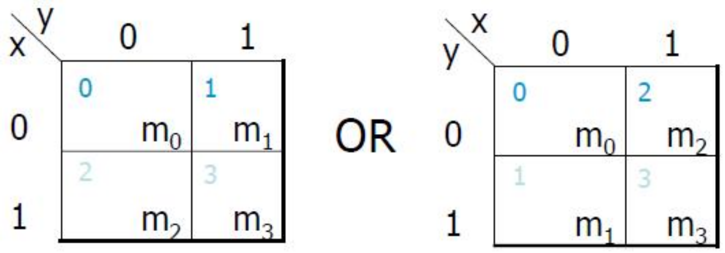
• Multiple-cell areas of the map correspond to standard terms.

x	y	minterm
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3



Karnaugh Maps

NOTE: ordering of variables is IMPORTANT for f(x,y), x is the row, y is the column. Cell 0 represents x'y'; Cell 1 represents x'y; etc.
If a minterm is present in the function, then a 1 is placed in the corresponding cell.



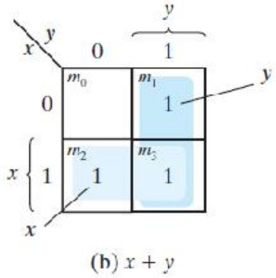
Boolean Function in Karnaugh Map

1s and 0s represent function in Karnaugh map
–1 represent *On-set* (*F=1*), 0 represents *Off-set* (*F=0*)
–Similar to truth table
–*0*s are typically not shown

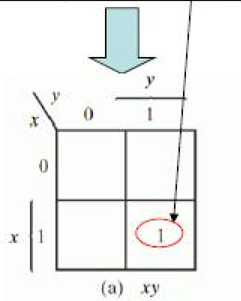
Example: And & OR Gates

x	y	f
0	0	0
0	1	0
1	0	0
1	1	1

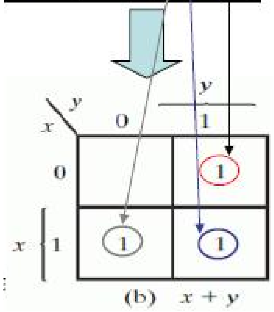
x	y	f
0	0	0
0	1	1
1	0	1
1	1	1



(b) $x + y$



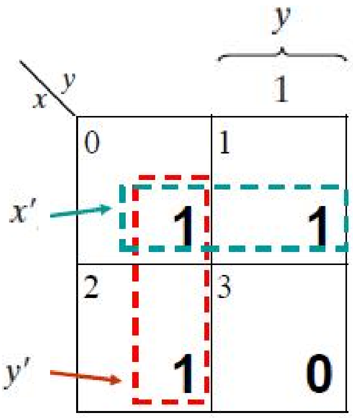
(a) xy



(b) $x + y$

Boolean Function in Karnaugh Map

F = x' + y' ;



Any two adjacent cells in the map differ by ONLY one variable, which appears complemented in one cell and uncomplemented in the other.

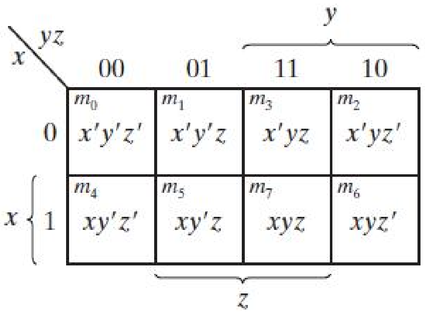
•**Example: m_0** (=x'y') **is adjacent to m_1** (=x'y) **this means that**
x'y' +x'y= x'(y'+y)=x'

•**Also m_0** (=x'y') **is adjacent to m_2** (=xy')
x'y' +xy'= y'(x'+x)=y

Three-Variable K-Map

• Karnaugh map with 3 variables:
– Two variables on one side, one on the other
– Note Gray code sequence (single variable change) facilitates grouping of 1-entries into logic blocks
• Note that the minterms are not arranged in a binary sequence, but similar to the Gray code.
• For simplifying Boolean functions, we must recognize the basic property possessed by adjacent squares.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



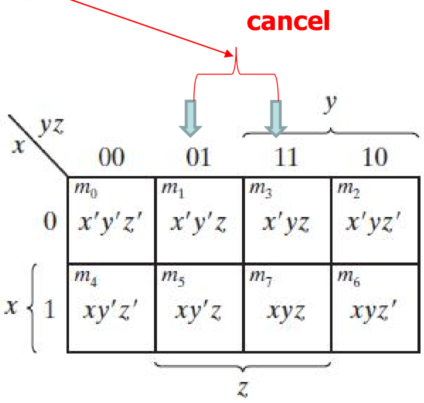
x \ yz	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

Three-Variable K-Map

To clarify this concept, consider the sum of two adjacent squares such as m_5 and m_7:

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Here, the two squares differ by the variable *y* , which can be removed when the sum of the two minterms is formed. Thus, any two minterms in adjacent squares (vertically or horizontally, but not diagonally, adjacent) that are ORed together will cause a removal of the dissimilar variable.

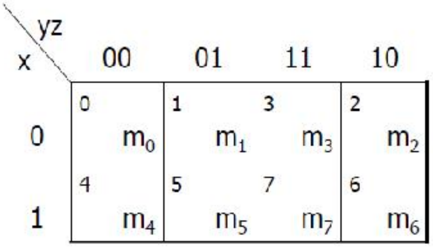


x \ yz	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

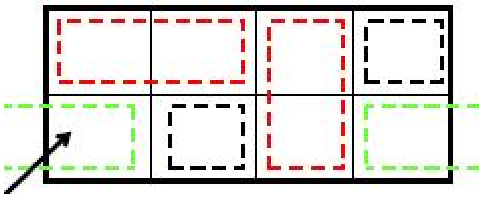
Neighboring minterms can be combined:

Three-Variable K-Map

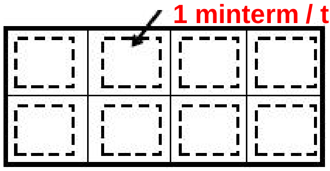
-Note: variable ordering is (x,y,z); yz specifies column, x specifies row.
-Each cell is adjacent to *three* other cells (left or right or top or bottom or edge wrap)



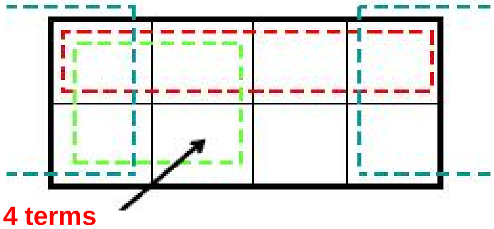
x \ yz	00	01	11	10
0	0 m_0	1 m_1	3 m_3	2 m_2
1	4 m_4	5 m_5	7 m_7	6 m_6



Group of 2 terms



1 minterm / term



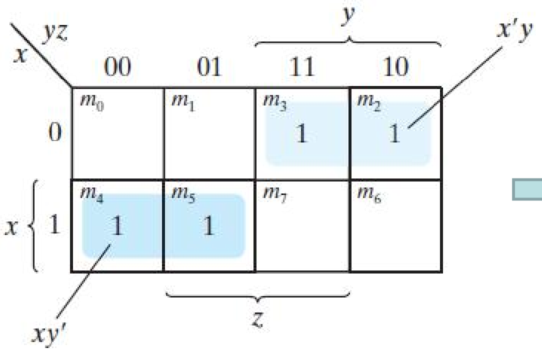
Group of 4 terms

Three-Variable K-Map

Simplify the Boolean function

$$F(x, y, z) = \Sigma(2, 3, 4, 5)$$

$$F(x,y,z) = \Sigma(2,3,4,5) = x'yz' + x'yz + xy'z' + xy'z$$



$$F = x'y + xy'$$

Three-Variable K-Map

For the Boolean function

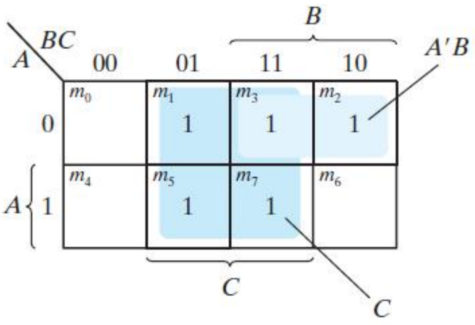
$$F = A'C + A'B + AB'C + BC$$

(a) Express this function as a sum of minterms.

(b) Find the minimal sum-of-products expression.

$$F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$$

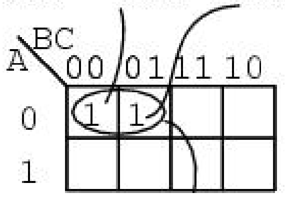
	$\bar{B}\,\bar{C}$	$\bar{B}\,C$	BC	$B\bar{C}$
$\bar{A}$				
A				



$$F = C + A'B$$

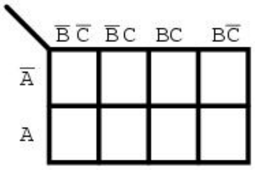
Three-Variable K-Map

Out= $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$



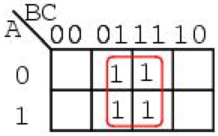
A\BC	00	01	11	10
0	1	1		
1				

Out= $\bar{A}\bar{B}$



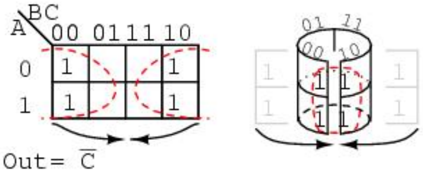
	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$				
A				

Out= $\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC$



A\BC	00	01	11	10
0		1	1	
1		1	1	

Out= C



A\BC	00	01	11	10
0	1			1
1	1			1

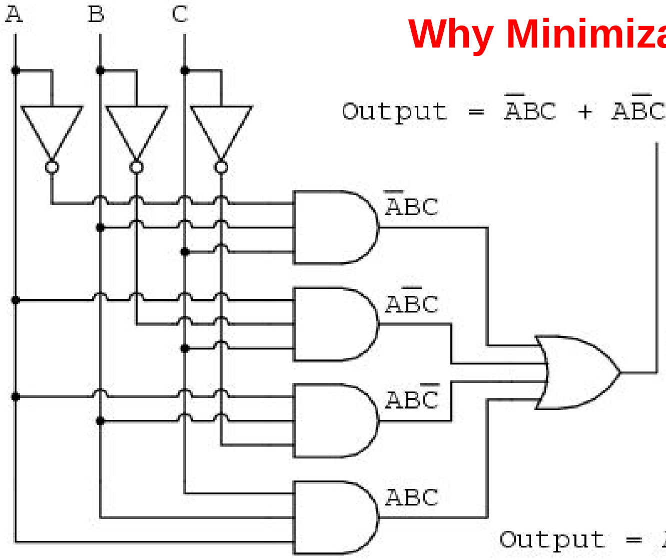
Out= $\bar{C}$

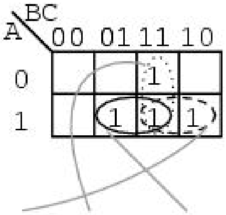
Out= $\bar{C}$

Three-Variable K-Map

Why Minimization?

Output = $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$



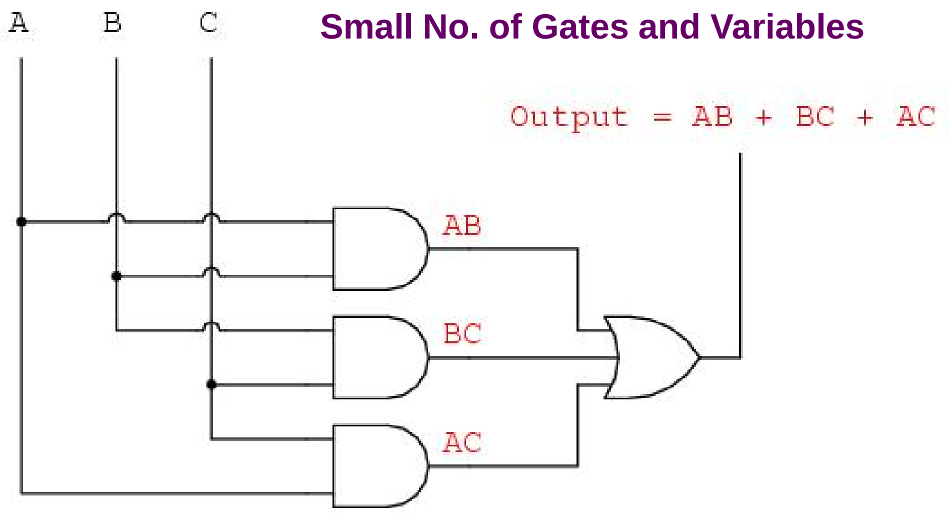


A\BC	00	01	11	10
0			1	
1		1	1	1

Output = AB + BC + AC

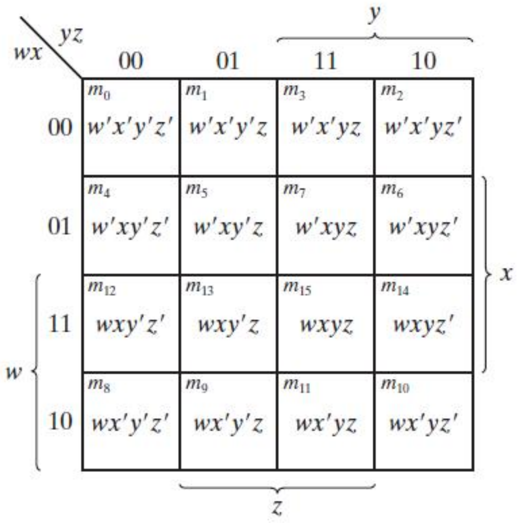
Three-Variable K-Map

Small No. of Gates and Variables



Four-Variable K-Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

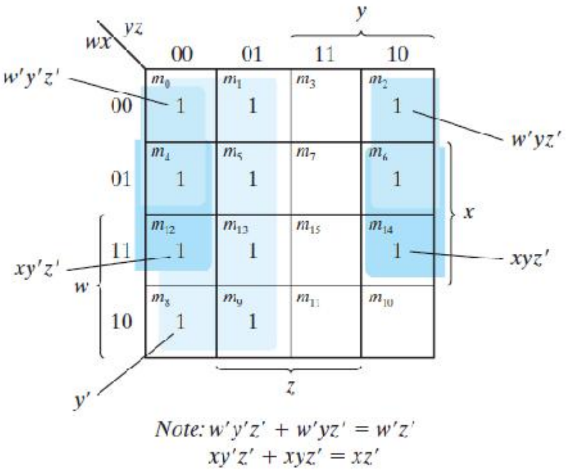


$wx \backslash yz$	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

Four-Variable K-Map

Simplify the Boolean function

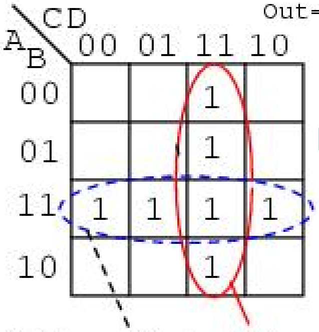
$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$



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

Four-Variable K-Map

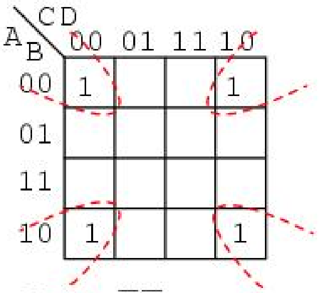
$$Out= \overline{A}\overline{B}CD+\overline{A}BCD+ABCD+A\overline{B}CD+AB\overline{C}\overline{D}+AB\overline{C}D+ABC\overline{D}$$

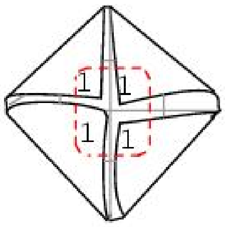


Out= AB + CD

$$Out= AB + CD$$

$$Out= \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D}$$

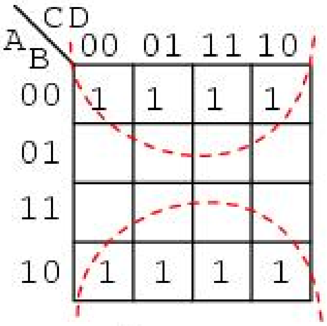




$$Out= \overline{B}\overline{D}$$

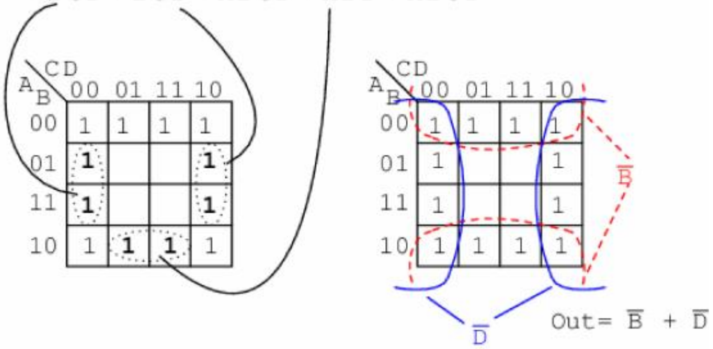
Four-Variable K-Map

$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D}$$



$\text{Out} = \overline{B}$

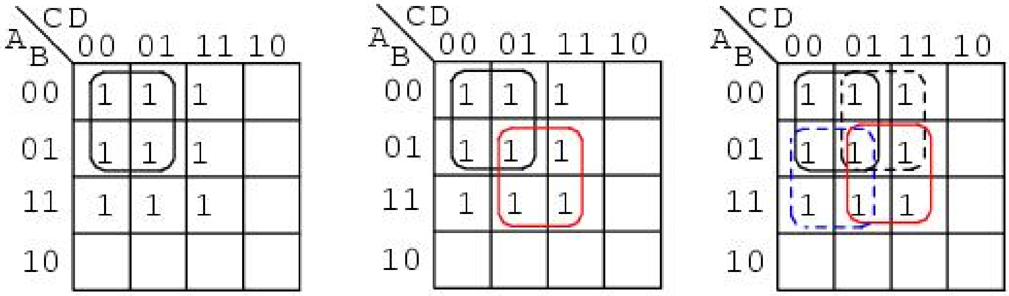
$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + B\overline{C}\,\overline{D} + BC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}D + A\overline{B}C\overline{D}$$



$\text{Out} = \overline{B} + \overline{D}$

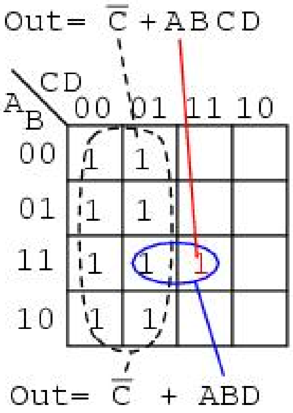
Four-Variable K-Map

$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD$$



$\text{Out} = \overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$

Four-Variable K-Map



Simplification by Boolean Algebra

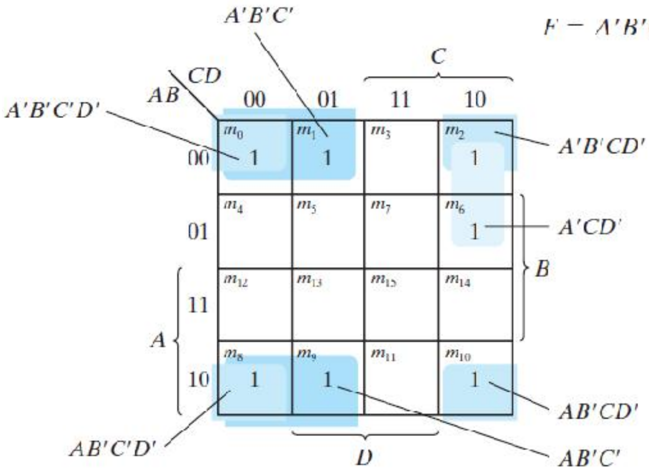
Out= $\overline{C}$ + ABCD

Applying rule $A + \overline{A}B = A + B$ to the $\overline{C}$ + ABCD term

Out= $\overline{C}$ + ABD

Four-Variable K-Map

$F - A'B'C' + B'CD' + A'BCD' - AB'C'$



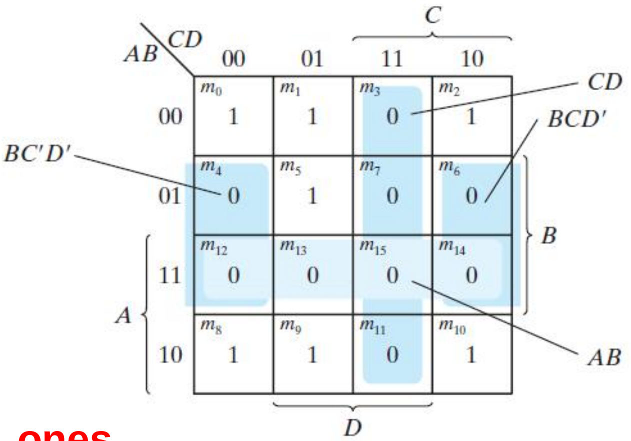
$$F = B'D' + B'C' + A'CD'$$

Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' - B'D'$
$A'B'C' + AB'C' = B'C'$

Digital Design

Example 3.7

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$



$AB \backslash CD$	00	01	11	10
00	m_0 1	m_1 1	m_3 0	m_2 1
01	m_4 0	m_5 1	m_7 0	m_6 0
11	m_{12} 0	m_{13} 0	m_{15} 0	m_{14} 0
10	m_8 1	m_9 1	m_{11} 0	m_{10} 1

If the squares marked with 1's

$$F = B'D' + B'C' + A'C'D$$

sum-of-products

ones

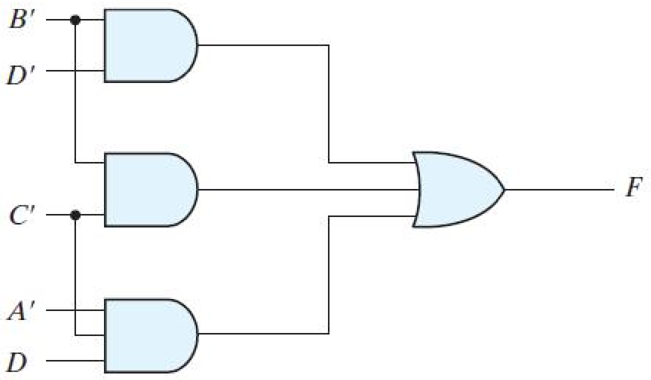
Note: $BC'D' + BCD' = BD'$

Dr. Hany Ahmed Date: 2016

Digital Design

Example 3.7

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:



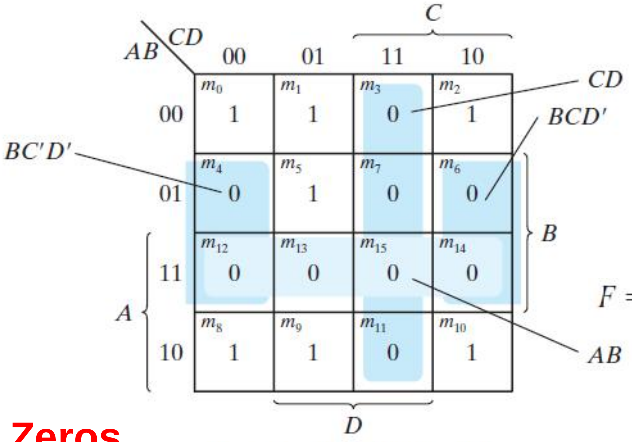
(a) $F = B'D' + B'C' + A'C'D$

sum-of-products

Dr. Hany Ahmed Date: 2016

Example 3.7

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$



Note: $BC'D' + BCD' = BD'$

Zeros

If the squares marked with 0's

$$F' = AB + CD + BD'$$

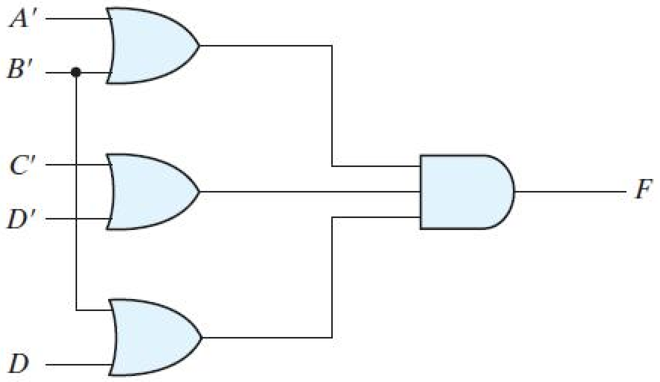
Applying DeMorgan's theorem

$$F = (A' + B')(C' + D')(B' + D)$$

productof-sums form

Example 3.7

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:



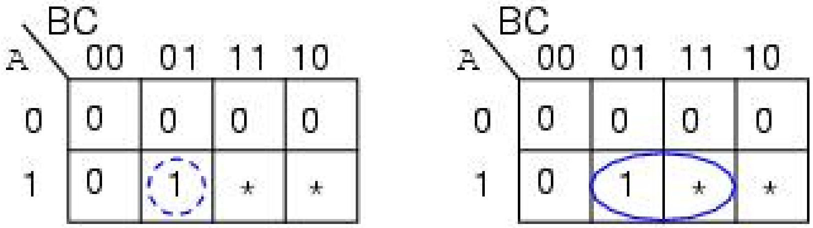
(b) $F = (A' + B')(C' + D')(B' + D)$

product-of-sums

Don't-Care Conditions

Don't Care Cells in the Karnaugh Map >>>>>> **x or** *****

Don't cares in a Karnaugh map, or truth table, may be either 1s or 0s,

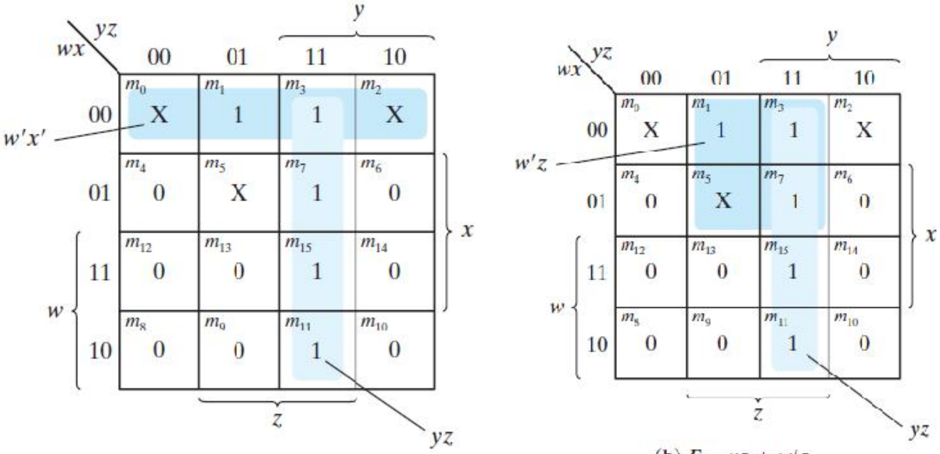


$Out = A\overline{B}C$ $Out = AC$

Don't-Care Conditions

Simplify the Boolean function $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$

which has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$



(a) $F = yz + w'x'$

(b) $F = yz + w'z$

Don't-Care Conditions

$$F(w,x,y,z) = yz + w'x' = \Sigma(0,1,2,3,7,11,15)$$
$$F(w,x,y,z) = yz + w'z = \Sigma(1,3,5,7,11,15)$$

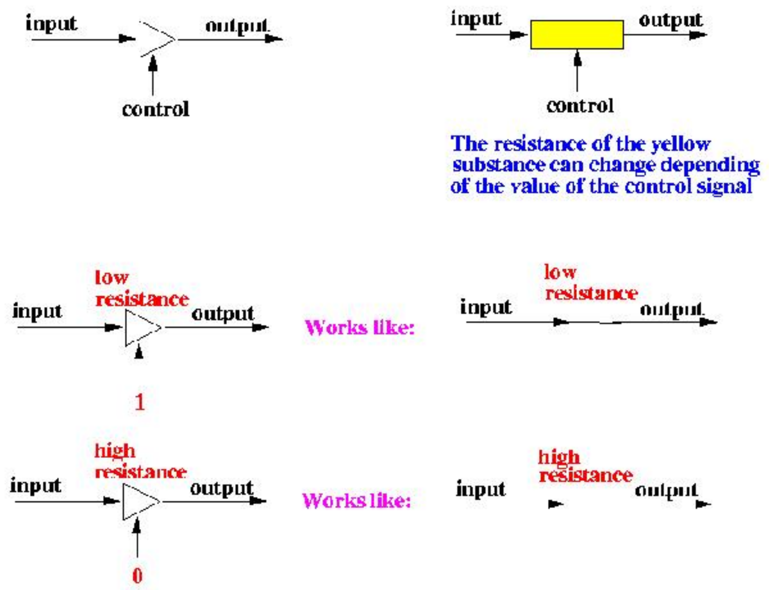
It is also possible to obtain a simplified product-of-sums expression for the function. In this case, the only way to **combine the 0's** is to include don't-care minterms 0 and 2 with the 0's to give a simplified complemented function:

$$F' = z' + wy'$$

$$F(w,x,y,z) = z(w' + y) = \Sigma(1,3,5,7,11,15)$$

Tri-State-Buffers

Tri-state-buffer is actually a "variable resistor"



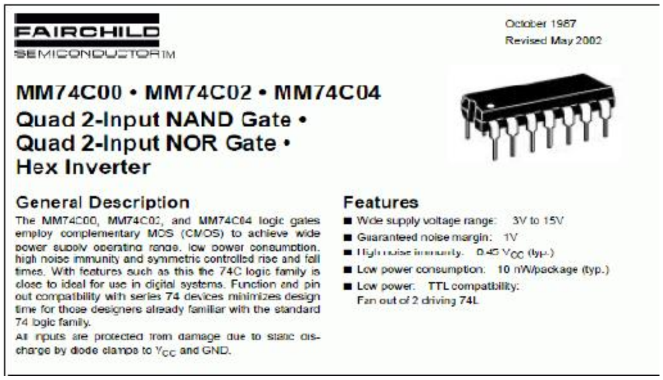
The resistance of the yellow substance can change depending of the value of the control signal

CH 3 Nand and Nor Implementation

Nand and Nor Implementation

- Digital circuit are frequently constructed with NAND or NOR gates rather than AND and OR gates.
- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

NAND Circuits



FAIRCHILD SEMICONDUCTOR™

October 1987
Revised May 2002

MM74C00 • MM74C02 • MM74C04
Quad 2-Input NAND Gate •
Quad 2-Input NOR Gate •
Hex Inverter

General Description

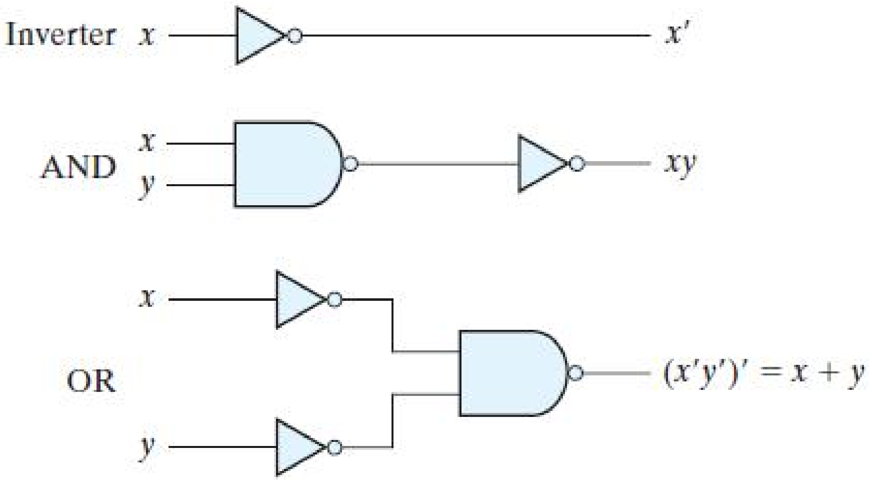
The MM74C00, MM74C02, and MM74C04 logic gates employ complementary MOS (CMOS) to achieve wide power supply operating range, low power consumption, high noise immunity and symmetric controlled rise and fall times. With features such as this the 74C logic family is close to ideal for use in digital systems. Function and pin out compatibility with series 74 devices minimizes design time for those designers already familiar with the standard 74 logic family.

All inputs are protected from damage due to static discharge by diode clamps to V_{CC} and GND.

Features

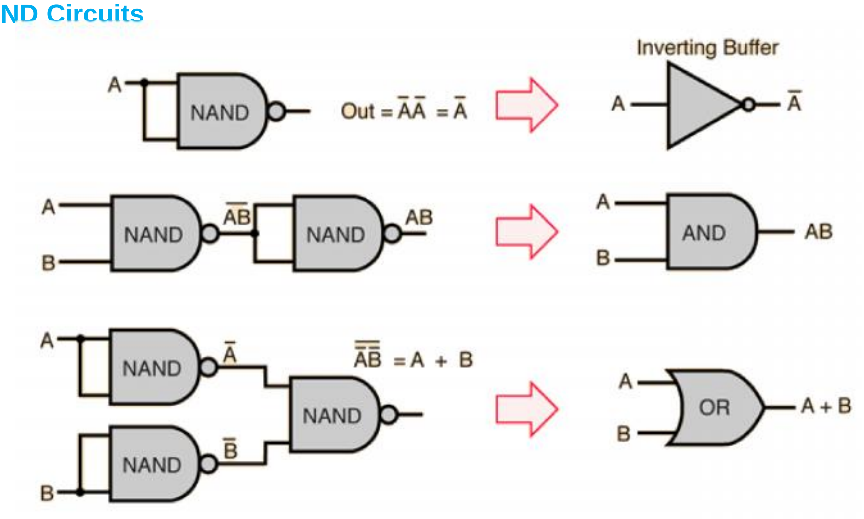
- Wide supply voltage range: 3V to 15V
- Guaranteed noise margin: 1V
- High noise immunity: 0.45 V_{CC} (typ.)
- Low power consumption: 10 nW/package (typ.)
- Low power: TTL compatibility: Fan out of 2 driving 74L

Logic operations with NAND gates



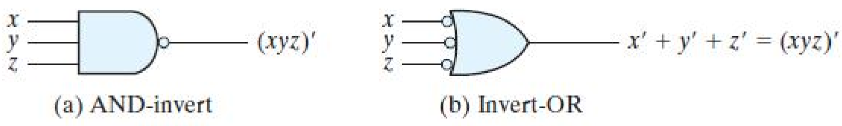
Logic operations with NAND gates

NAND Circuits



Logic operations with NAND gates

Two graphic symbols for a three-input NAND gate



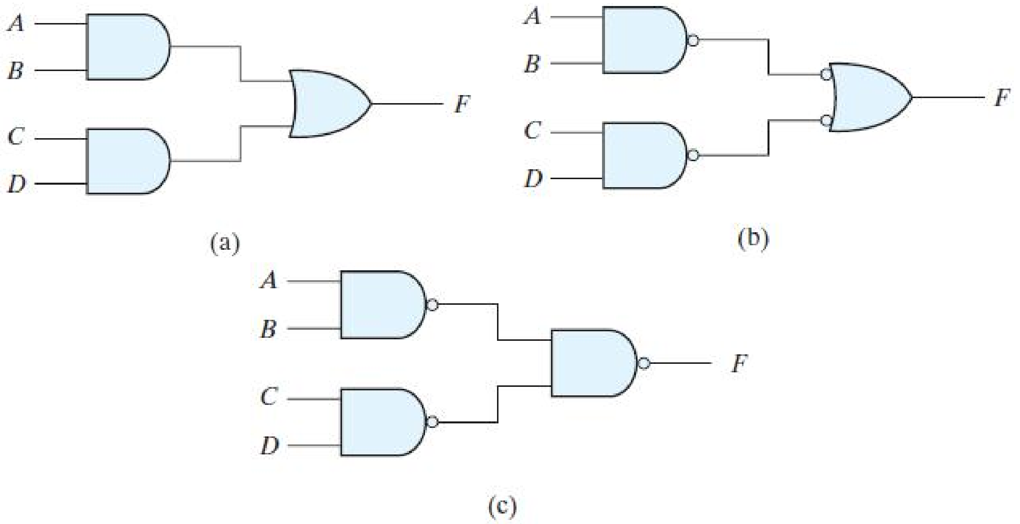
(a) AND-invert

(b) Invert-OR

The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form.

Two-Level Implementation

Example Implement $F = AB + CD$



(a)

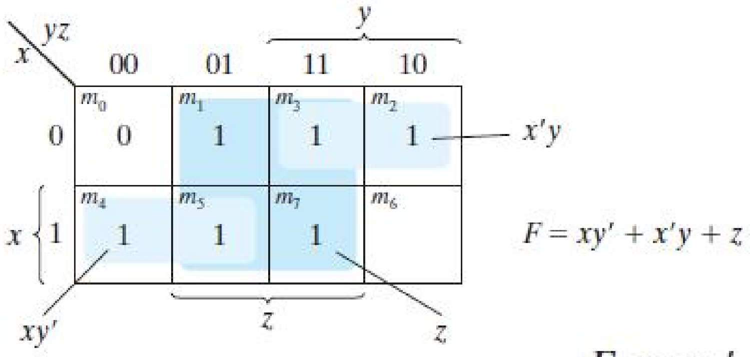
(b)

(c)

Logic operations with NAND gates

Implement the following Boolean function with NAND gates:

$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$



x \ yz	00	01	11	10
0	m_0 0	m_1 1	m_3 1	m_2 1
1	m_4 1	m_5 1	m_7 1	m_6

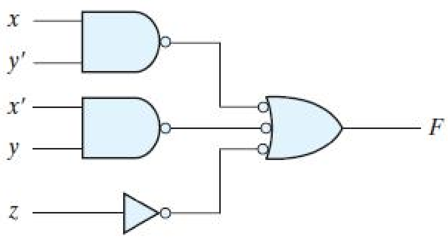
(a)

$$F = xy' + x'y + z$$

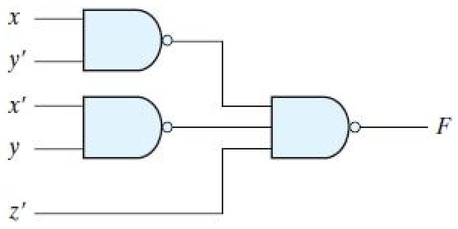
Logic operations with NAND gates

Implement the following Boolean function with NAND gates:

$$F = xy' + x'y + z$$



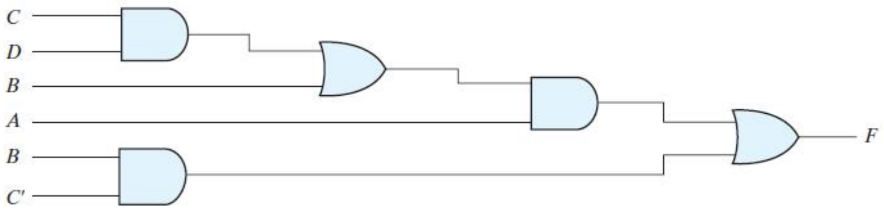
(b)



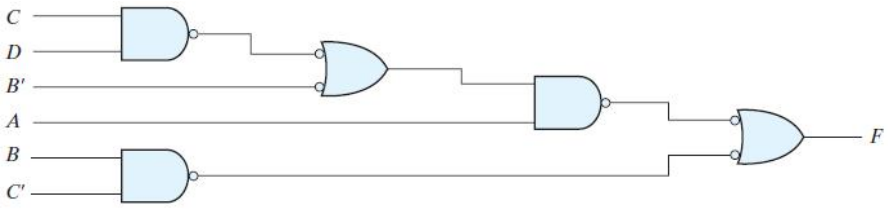
(c)

Multilevel NAND Circuits

$$F = A(CD + B) + BC'$$

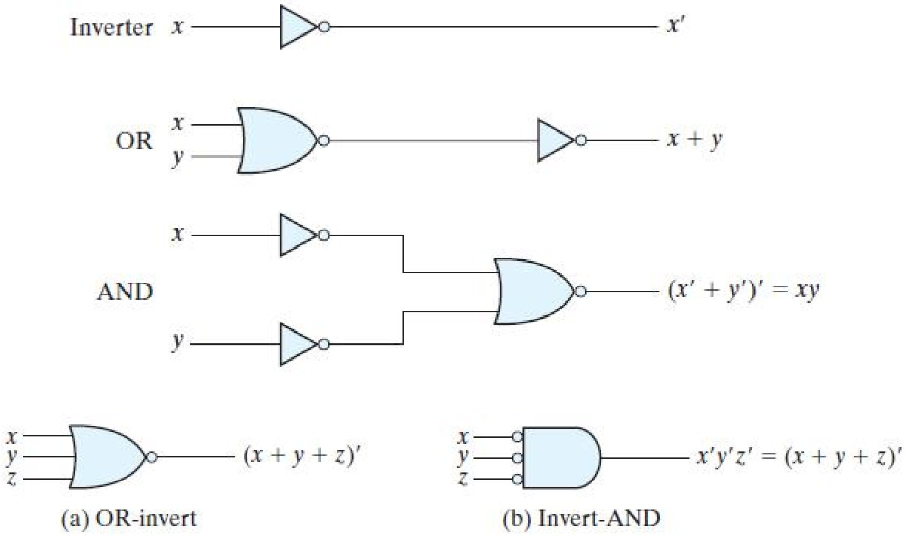


(a) AND–OR gates



(b) NAND gates

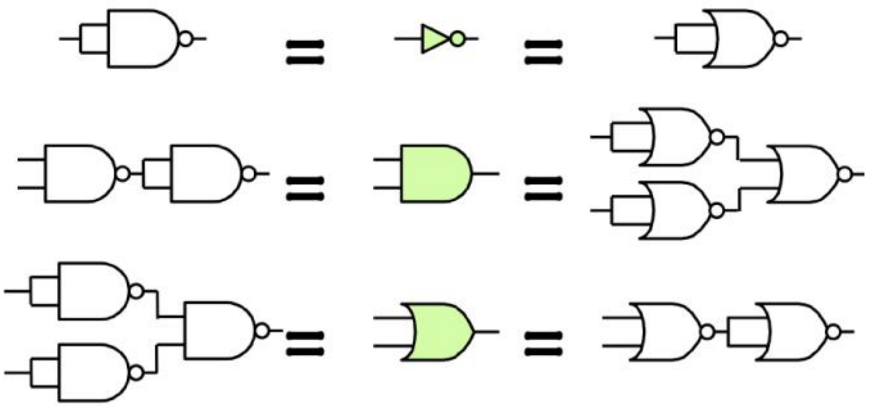
Nor Implementation



(a) OR-invert

(b) Invert-AND

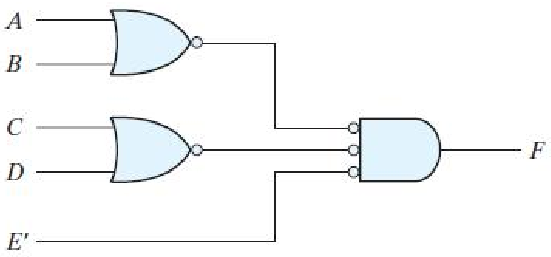
Nand and Nor Implementation



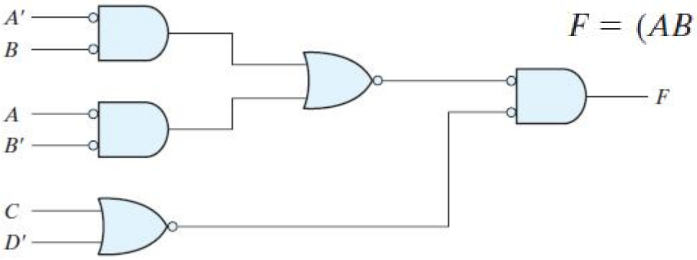
Nand and Nor Implementation

NOR Circuits

Implementing $F = (A + B)(C + D)E$

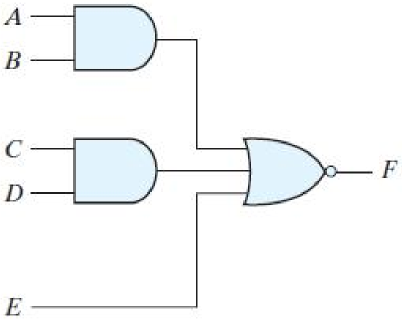


$F = (AB' + A'B)(C + D')$



Other Two Level Implementation

AND–OR	OR–AND
NAND–NAND	NOR–NOR
NOR–OR	NAND–AND
OR–NAND	AND–NOR

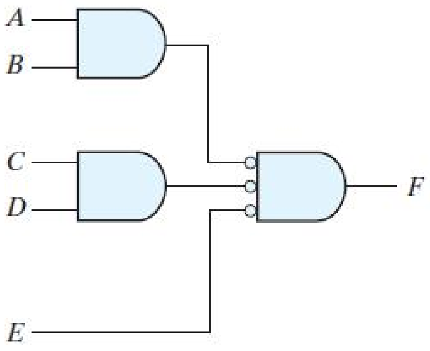


$F = (AB + CD + E)'$

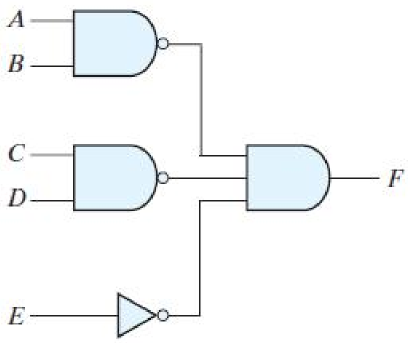
(a) AND–NOR

Other Two Level Implementation

$F = (AB + CD + E)'$



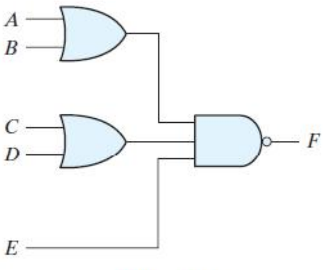
(b) AND–NOR



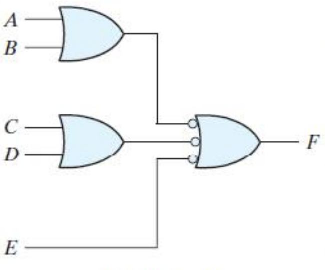
(c) NAND–AND

Other Two Level Implementation

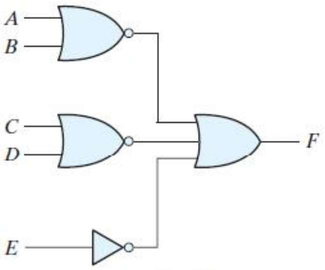
$F = [(A + B)(C + D)E]'$



(a) OR–NAND



(b) OR–NAND



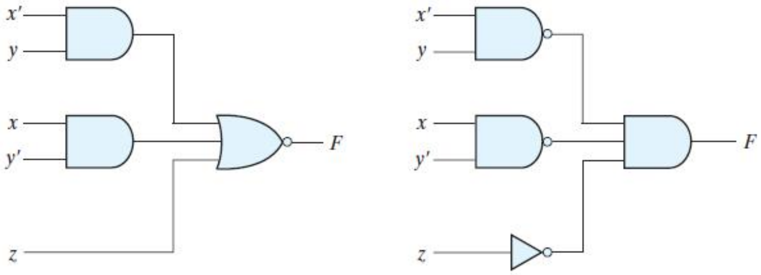
(c) NOR–OR

Nand and Nor Implementation

Example

$$F' = x'y + xy' + z$$

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



AND–NOR

NAND–AND

(b) $F = (x'y + xy' + z)'$

The exclusive-OR (XOR)

$$x \oplus y = xy' + x'y$$

The following identities apply to the exclusive-OR operation:

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

$$x \oplus x = 0$$

$$x \oplus x' = 1$$

$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

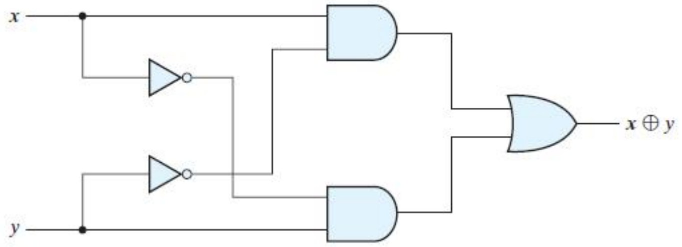
the exclusive-OR operations both commutative and associative;

$$A \oplus B = B \oplus A$$

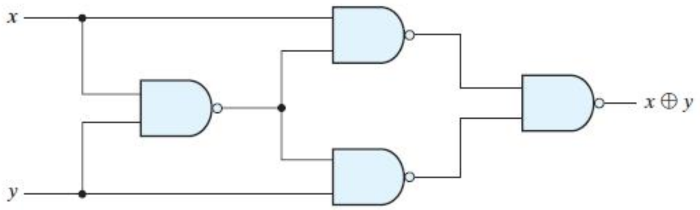
and

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

The exclusive-OR (XOR)



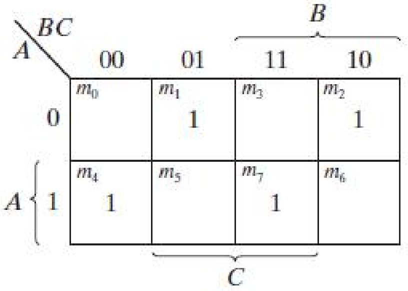
(a) Exclusive-OR with AND–OR–NOT gates



(b) Exclusive-OR with NAND gates

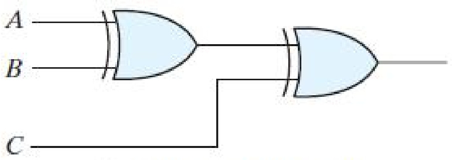
Odd Function

$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \Sigma(1, 2, 4, 7) \end{aligned}$$



A \ BC	00	01	11	10
0	m_0	m_1 1	m_3	m_2 1
1	m_4 1	m_5	m_7 1	m_6

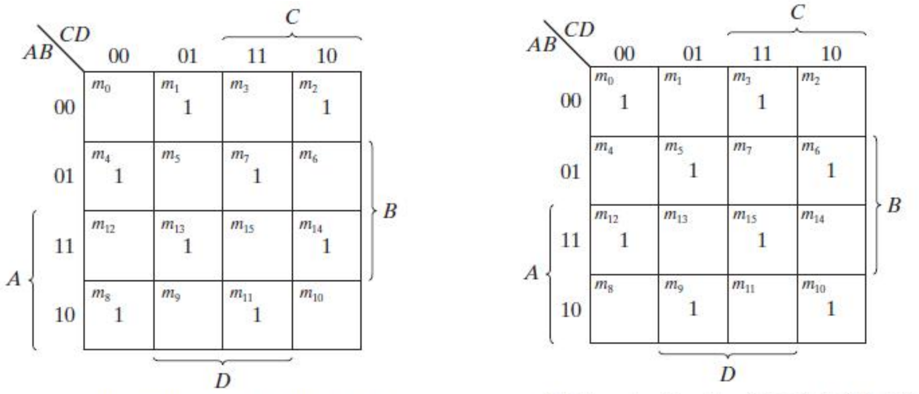
(a) Odd function $F = A \oplus B \oplus C$



(a) 3-input odd function

Odd Function

$$
\begin{aligned}
A \oplus B \oplus C \oplus D &= (AB' + A'B) \oplus (CD' + C'D) \\
&= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D) \\
&= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)
\end{aligned}
$$



(a) Odd function $F = A \oplus B \oplus C \oplus D$

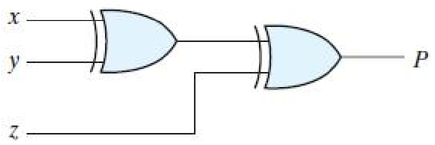
(b) Even function $F = (A \oplus B \oplus C \oplus D)'$

Parity Generation and Checking

Even-Parity-Generator Truth Table

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

$$P = x \oplus y \oplus z$$



(a) 3-bit even parity generator

Even Parity Check →

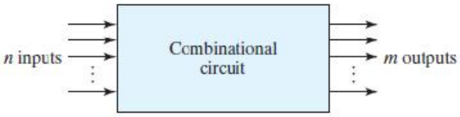
1- No. of One's **Even → 0**

2- No. of One's **Odd→ 1**

CH 4 combinational circuit:

Logic circuits can be classified as either

1- Combinational Logic Circuits — the output(s) depend only on the input(s) at any specific time and not on any previous input(s).

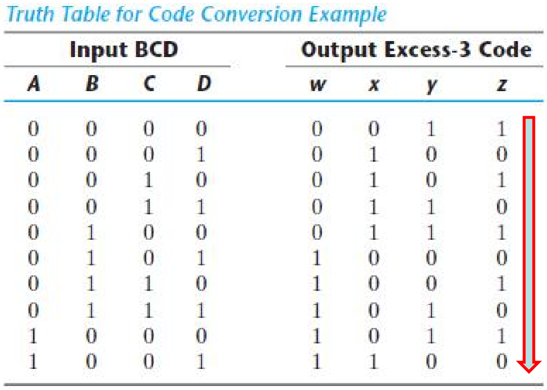


An example of the two concepts is a television remote control. You can enter a number and the output (a particular television channel) depends only on the number entered. It does not matter what channels been viewed previously. So the relationship between the input (a number) and the output is combinational.

2- Sequential Logic Circuits — the output(s) depend both on previous and current input(s).

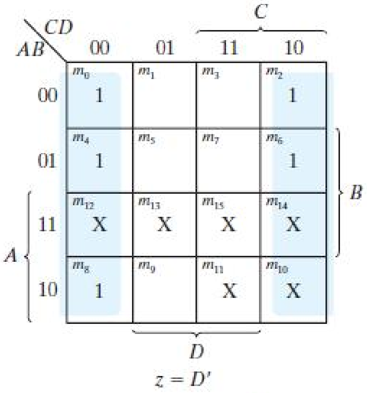
The remote control also has inputs for stepping either up or down one channel. When using this input method, the channel selected depends on what channel has been previously selected and the sequence of up/down button pushes. The channel up/down buttons illustrate a sequential input/output relationship.

CH 4 combinational circuit:
Code Conversion BCD to Excess-3



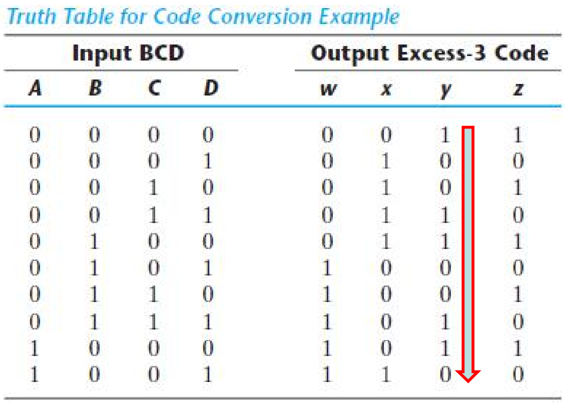
Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



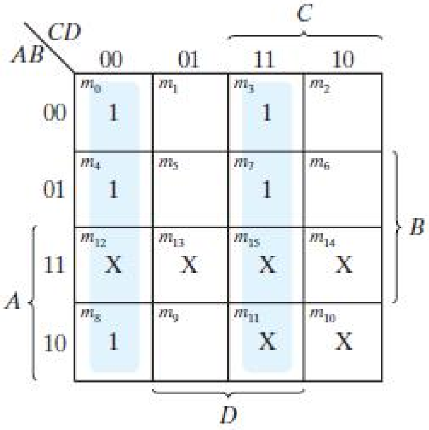
$z = D'$

CH 4 combinational circuit:
Code Conversion BCD to Excess-3



Truth Table for Code Conversion Example

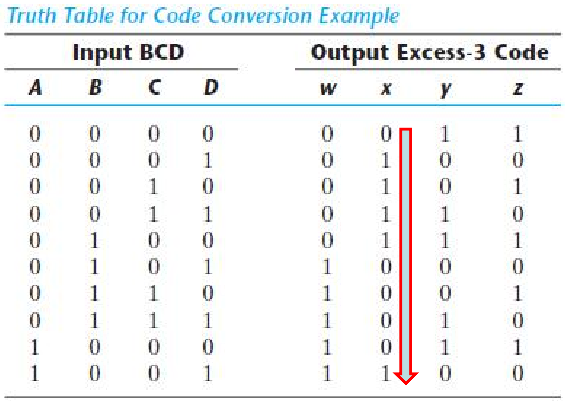
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



AB \ CD	00	01	11	10
00	m_0 1	m_1	m_3 1	m_2
01	m_4 1	m_5	m_7 1	m_6
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9	m_{11} X	m_{10} X

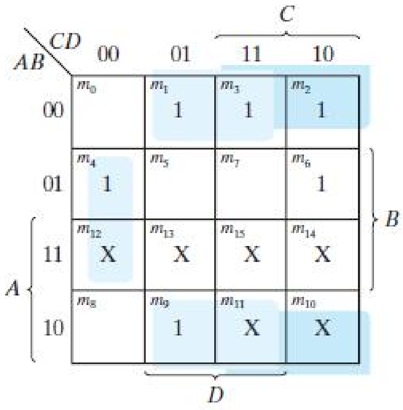
$y = CD + C'D'$

CH 4 combinational circuit:
Code Conversion BCD to Excess-3



Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



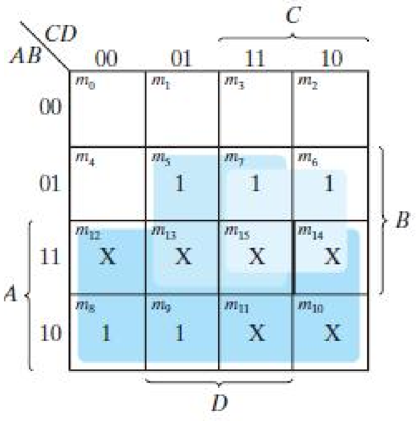
AB \ CD	00	01	11	10
00	m_0	m_1 1	m_3 1	m_2 1
01	m_4 1	m_5	m_7	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8	m_9 1	m_{11} X	m_{10} X

$x = B'C + B'D + BC'D'$

CH 4 combinational circuit:
Code Conversion BCD to Excess-3

Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5 1	m_7 1	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9 1	m_{11} X	m_{10} X

$w = A + BC + BD$

Code Conversion BCD to Excess-3

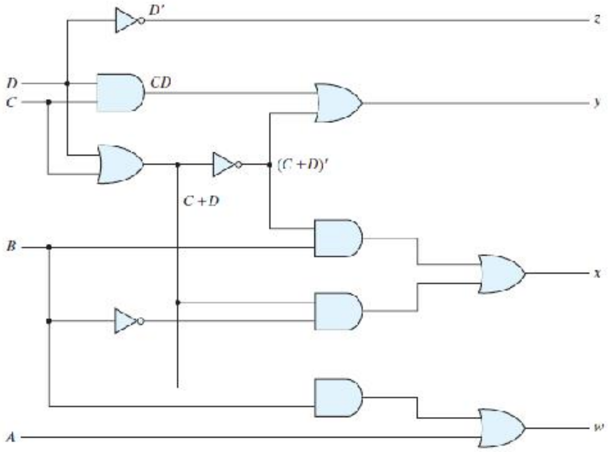
$z = D'$

$y = CD + C'D' = CD + (C + D)'$

$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$

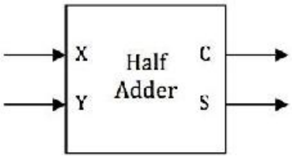
$= B'(C + D) + B(C + D)'$

$w = A + BC + BD = A + B(C + D)$



CH 4 combinational circuit:

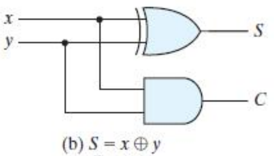
Half Adder (2 bit)



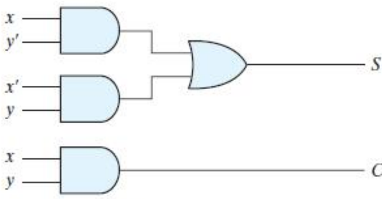
x_i	y_i	$Carry_{i+1}$	Sum_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder

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



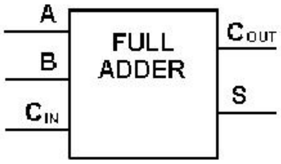
(b) $S = x \oplus y$
$C = xy$



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

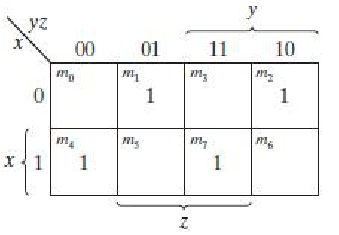
CH 4 combinational circuit:

Full Adder (3 bit)

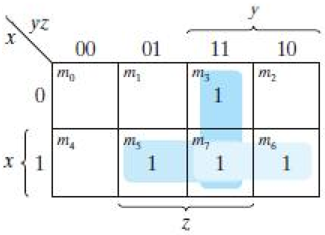


Full Adder

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



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

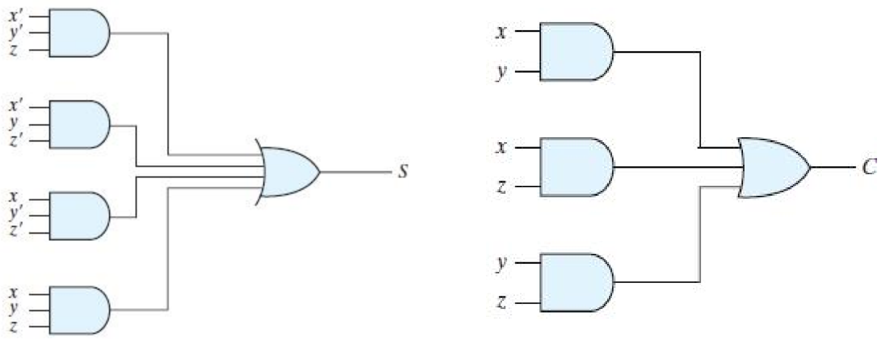


(b) $C = xy + xz + yz$

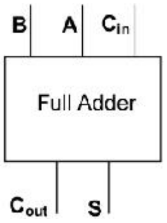
$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

CH 4 combinational circuit: Full Adder (3 bit)

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$



CH 4 combinational circuit: Full Adder (3 bit)



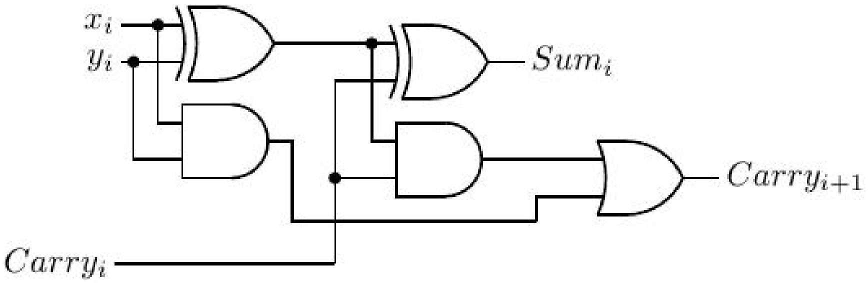
$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

$$\begin{aligned} Carry_{i+1} &= x_i \cdot y_i + Carry_i \cdot x_i' \cdot y_i + Carry_i \cdot x_i \cdot y_i' \\ &= x_i \cdot y_i + Carry_i \cdot (x_i' \cdot y_i + x_i \cdot y_i') \\ &= x_i \cdot y_i + Carry_i \cdot (x_i \oplus y_i) \end{aligned}$$

CH 4 combinational circuit:
Full Adder (3 bit)

$$S = z \oplus (x \oplus y)$$



TASKs

1- Learning MULTISIM 14 Software.

NI Multisim™

NATIONAL INSTRUMENTS
ELECTRONICS WORKBENCH GROUP

2- Implementing the Half and Full Adders using Multisim.

CH 4 : Combinational Circuit

Adders, Multipliers, Comparators, and 7 Segment Display

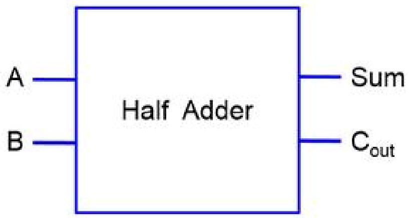
1- Adders

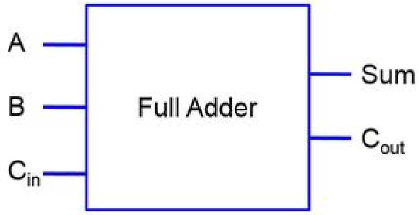
Half Adder

- 2 Inputs (A & B)
- 2 Outputs (Sum & C_{out})
- Used for LSB only

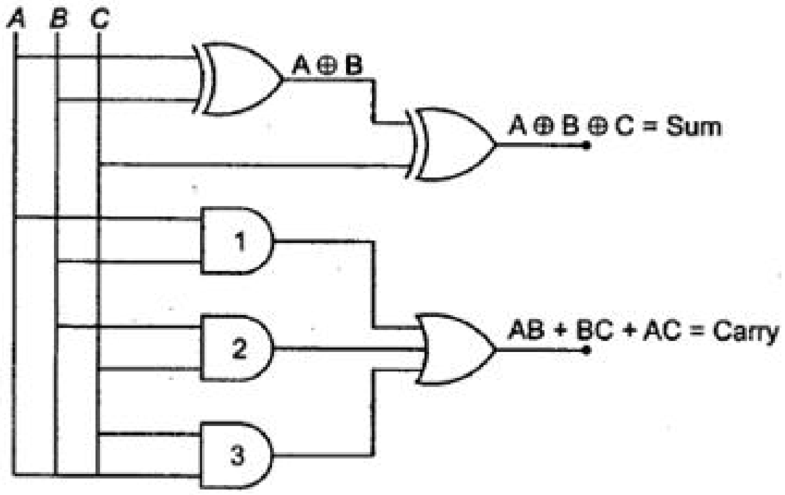
Full Adder

- 3 Inputs (A, B, C_{in})
- 2 Outputs (Sum & C_{out})
- Used for all other bits





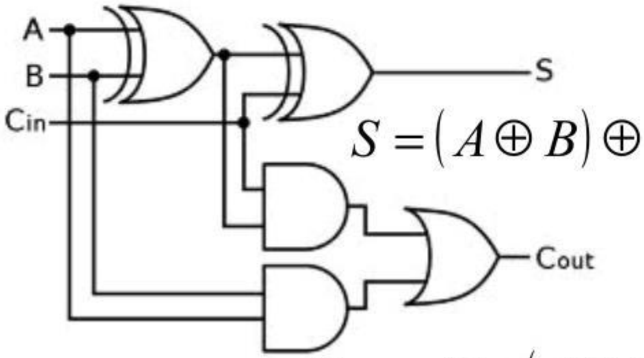
Full Adder Circuit



Full adder logic diagram

Full Adder Circuit

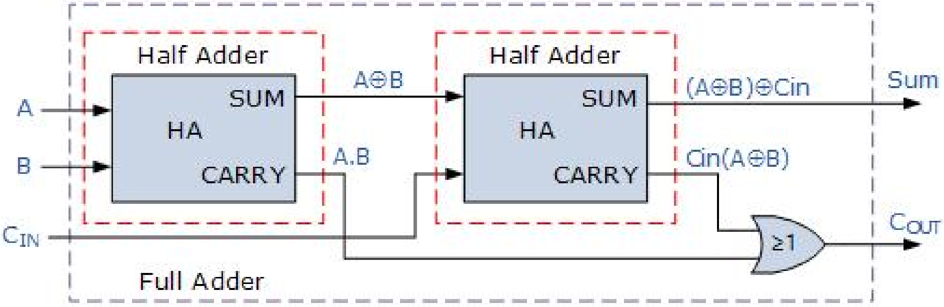
- Full adder logic diagram



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

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

Implementation of Full Adder with Two Half Adders

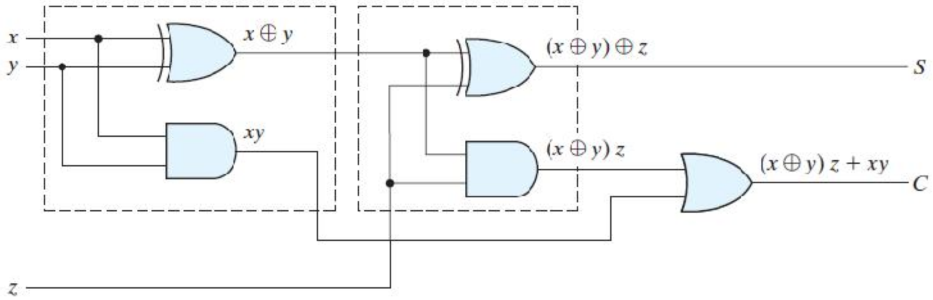


Full Adder

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

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

Implementation of Full Adder with Two Half Adders



Four-bit Adder

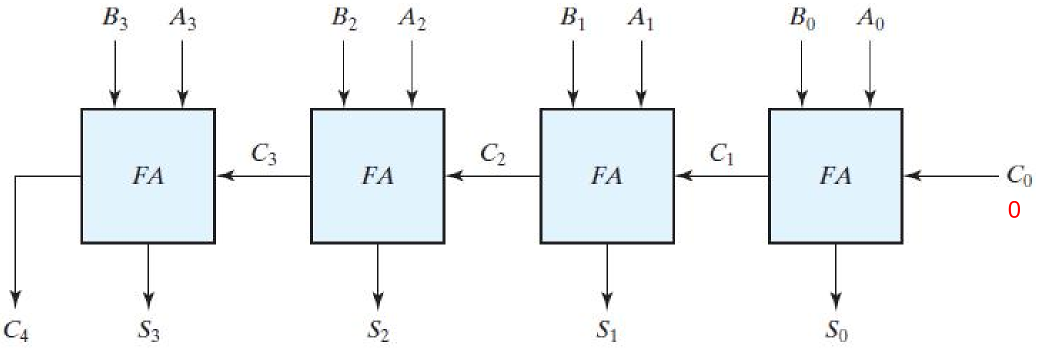
To demonstrate with a specific example, consider the two binary numbers $A = 1011$ and $B = 0011$. Their sum $S = 1110$ is formed with the four-bit adder as follows:

0	1	1	0	C_i
1	0	1	1	A_i
0	0	1	1	B_i
1	1	1	0	S_i
0	0	1	1	C_{i+1}

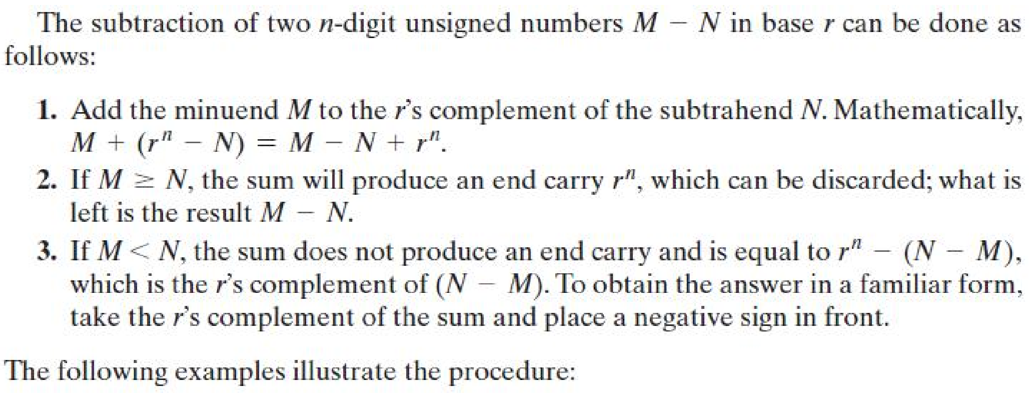
$A = 1011$

$B = 0011$

C3 C2 C1 Co
A3 A2 A1 Ao
B3 B2 B1 Bo +
C4 S3 S2 S1 So



Subtraction with Complements

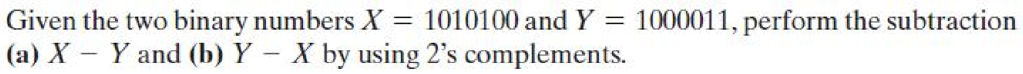


The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

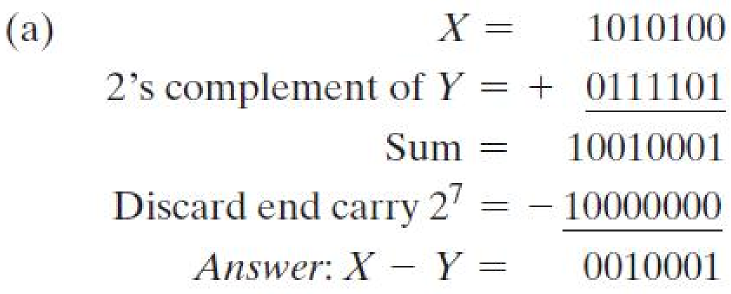
1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

The following examples illustrate the procedure:

Subtraction with Complements



Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction **(a)** $X - Y$ and **(b)** $Y - X$ by using 2's complements.



(a)

$$\begin{aligned} X &= \quad 1010100 \\ \text{2's complement of } Y &= +\ \underline{0111101} \\ \text{Sum} &= \quad 10010001 \\ \text{Discard end carry } 2^7 &= -\ \underline{10000000} \\ \textit{Answer}\text{: } X - Y &= \quad 0010001 \end{aligned}$$

84

67

17

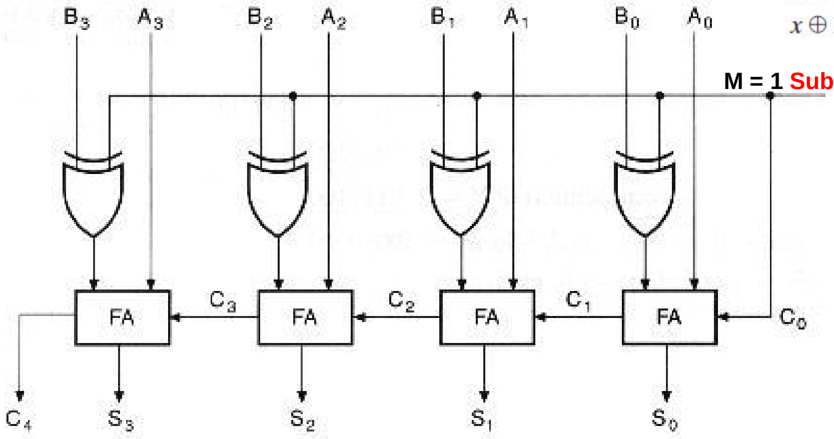
Subtraction with Complements

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction **(a)** $X - Y$ and **(b)** $Y - X$ by using 2's complements.

$$\begin{array}{llr} \text{(b)} & Y = & 1000011 \\ & \text{2's complement of } X = + & \underline{0101100} \\ & \text{Sum} = & 1101111 \end{array}$$

There is no end carry. Therefore, the answer is *Y* - *X* = -(2>s complement of 1101111) = -0010001

Four-bit Adder–Subtractor



$x \oplus 1 = x'$

M = 1 Subtraction

$x \oplus 0 = x$

M = 0 Addition
M = Co

Unsigned Numbers

C4 End Carry	Case Addition	Case Subtraction
0	Correct (N) bits	The answer is negative and the 2's complement (N) bits
1	Overflow (N+1) bits	Discard the End Carry (N) bits

Signed Binary Numbers

1- the signed-magnitude

The representation: the sign with a bit placed in the leftmost position of the number. The convention is to make the sign **bit 0** for **positive** and **1** for **negative**.

For example,
the string of bits 01001 can be considered as 9 (unsigned binary) or as +9 (signed binary)

The string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned number and the binary equivalent of -9 when considered as a signed number

referred to as the *signed-magnitude*

2- the signed-Complement system

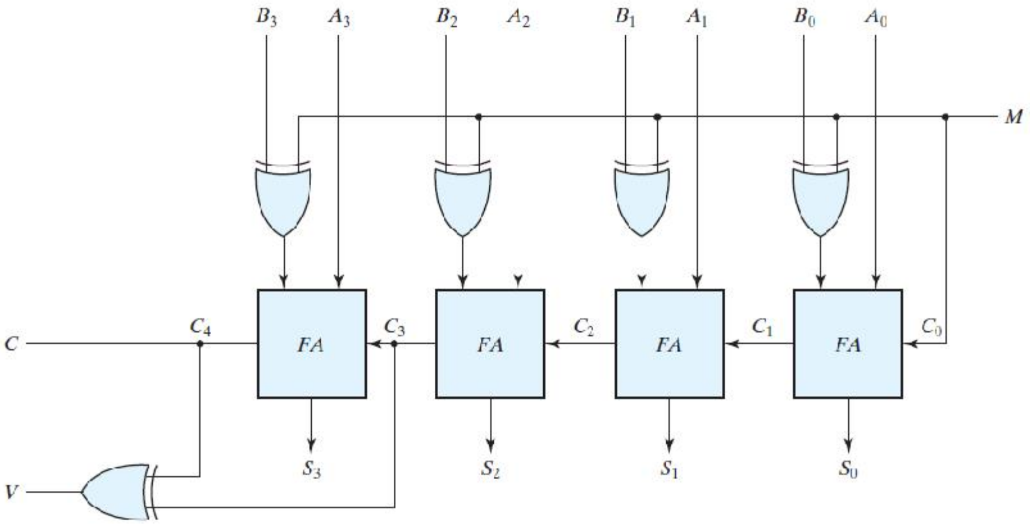
The complement will always start with a 1, indicating a negative number. The signed-complement system can use either the 1's or the 2's complement, but the 2's complement is the most common.

there are three different ways to represent -9 with eight bits

signed-magnitude representation: **-9**	10001001
signed-1's-complement representation:	11110110
signed-2's-complement representation:	11110111
signed-magnitude representation: **+9**	00001001

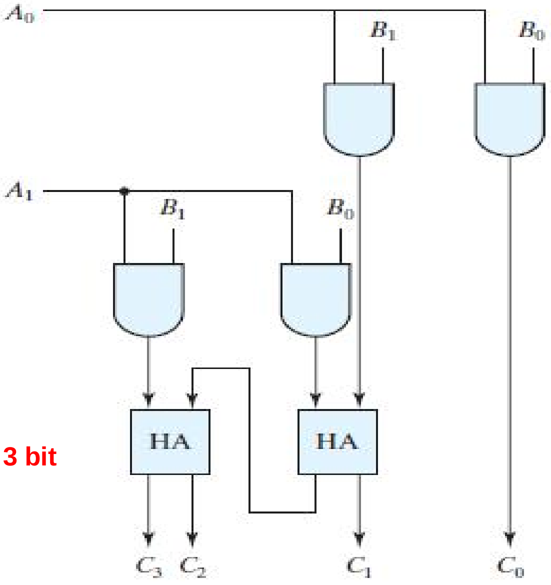
The representation: the sign with a bit placed in the leftmost position of the number. The convention is to make the sign **bit 0** for **positive** and **1** for **negative**.

Four-bit adder–subtractor with overflow detection



Two-bit by two-bit binary multiplier

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0

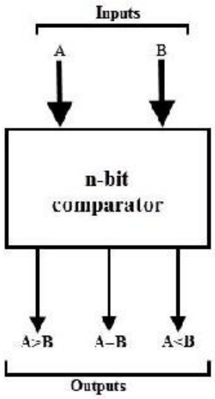


Task 1: Design the circuit of 3 * 3 bit Multiplier

Exact Deadline

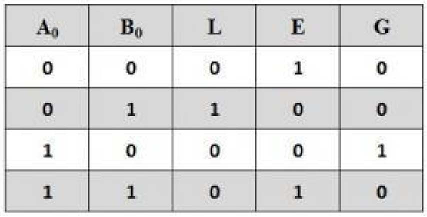
Magnitude Comparator

magnitude comparator **is a combinational circuit that compares two numbers *A* and *B* and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether *A* > *B*, *A* = *B*, or *A* < *B*.**

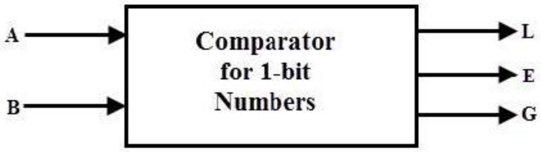


Single Bit Magnitude Comparator

A comparator used to compare two bits, i.e., two numbers each of single bit is called a single bit comparator. It consists of two inputs for allowing two single bit numbers and three outputs to generate less than, equal and greater than comparison outputs.

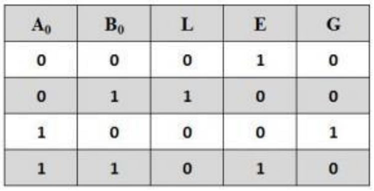


A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



The truth table for the single bit comparator is given below. When A0 B0 = 00 & 11, both inputs are equal, therefore A=B output will be high. When A0 B0 = 01, B is more than A and hence AB is active.

Single Bit Magnitude Comparator



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

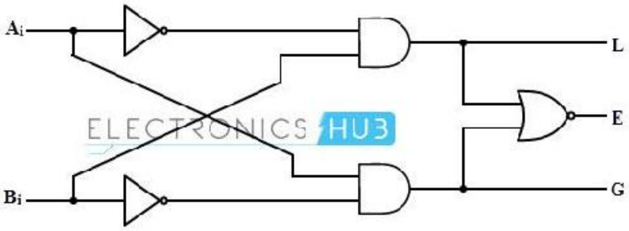
$$A0 < B0: L = \overline{A0}\ B0$$

$$A0 = B0: E = \overline{A0}\ \overline{B0} + A0\ B0$$

$$A0 > B0: G = A0\ \overline{B0}$$

It is to be noted that E can be realized as $\overline{(L + G)}$.

By using these Boolean expressions, we can implement a logic circuit for this comparator using two AND gates, one NOT gate and one **Ex-NOR gate** as shown



Magnitude 4 Bit Comparator

magnitude comparator **is a combinational circuit that compares two numbers *A* and *B* and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether *A* > *B*, *A* = *B*, or *A* < *B*.**

$$A = A_3\ A_2\ A_1\ A_0$$
$$B = B_3\, B_2\, B_1\, B_0$$

can be expressed logically with an exclusive-NOR function as

$$x_i = A_iB_i + A_i'B_i' \quad \text{for } i = 0, 1, 2, 3$$

where $x_i = 1$ only if the pair of bits in position i are equal (i.e., if both are 1 or both are 0).

Magnitude 4 Bit Comparator

E: Equal

$$(A = B) = x_3x_2x_1x_0$$

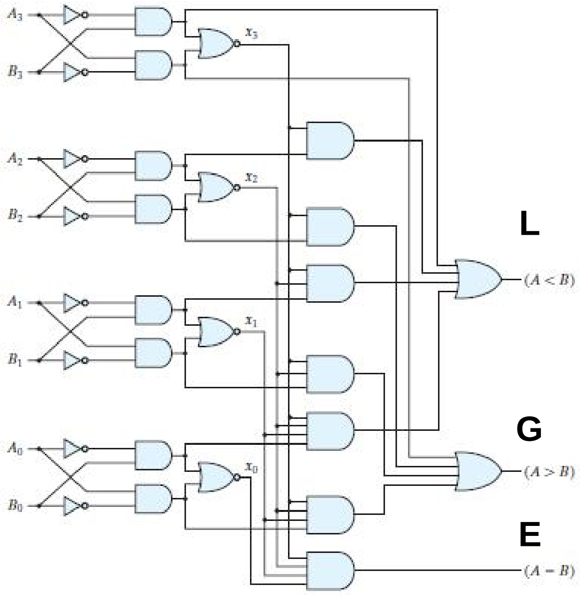
G: Greater

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

L: Less Than

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1' + x_3x_2x_1A'n_0B_0'$$

Magnitude 4 Bit Comparator



TASK 2

Design Magnitude comparator: 2-Bit Comparator

TASK 3

Design BCD to 7 segment

TASKs

1- Learning MULTISIM 14 Software.

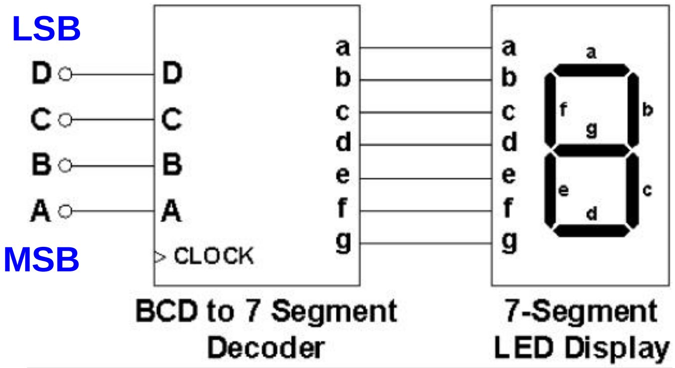
NI Multisim™

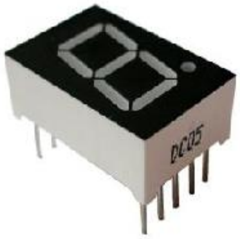
NATIONAL INSTRUMENTS
ELECTRONICS WORKBENCH GROUP

2- Implementing the BCD to 7 segment and multipliers.

7 Segment Display and Decoders

BCD to Seven Segment





BCD to Seven Segment

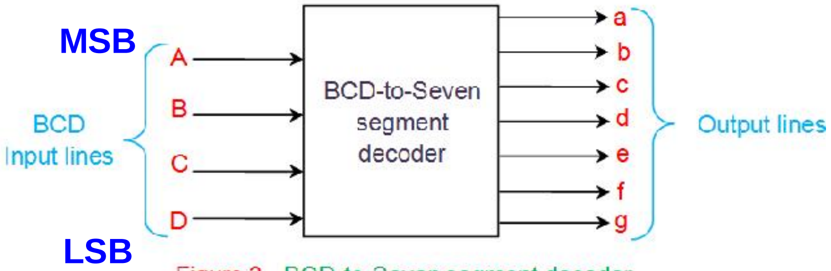
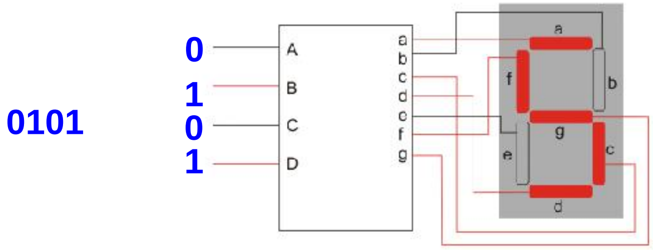


Figure 3 BCD-to-Seven segment decoder



BCD to Seven Segment

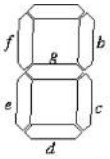


FIGURE 3: Seven segment display format showing arrangement of segments.

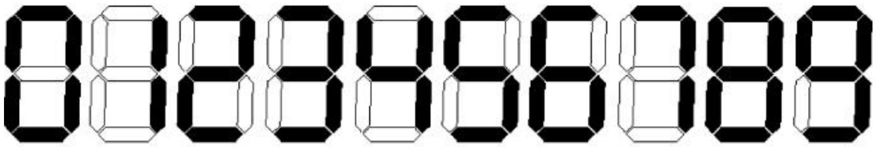
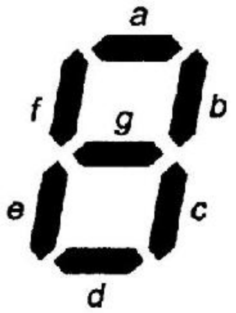


FIGURE 4 : Display of decimal digits with a 7-segment device.

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9



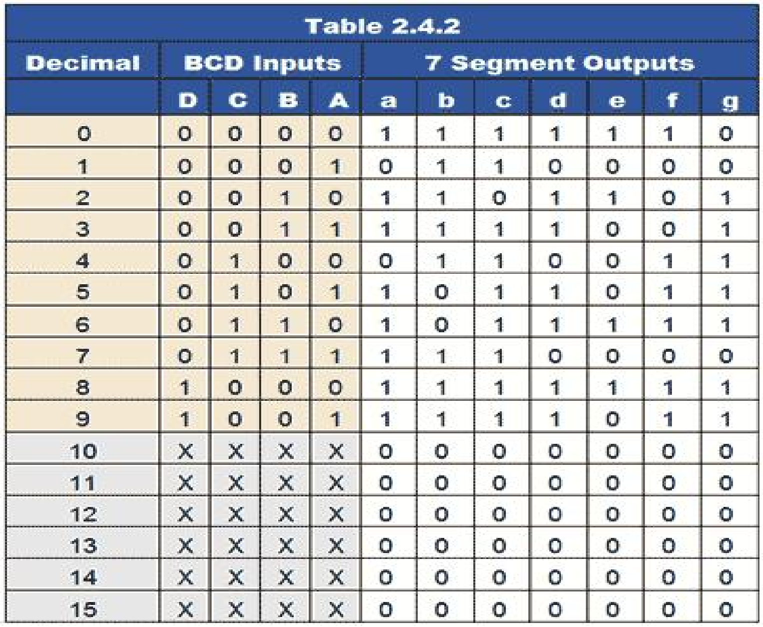
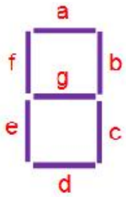
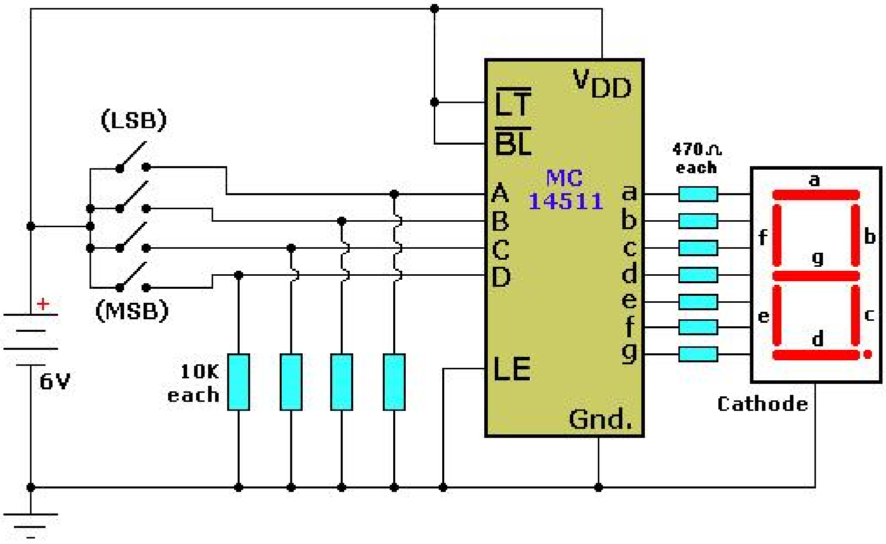


Table 2.4.2

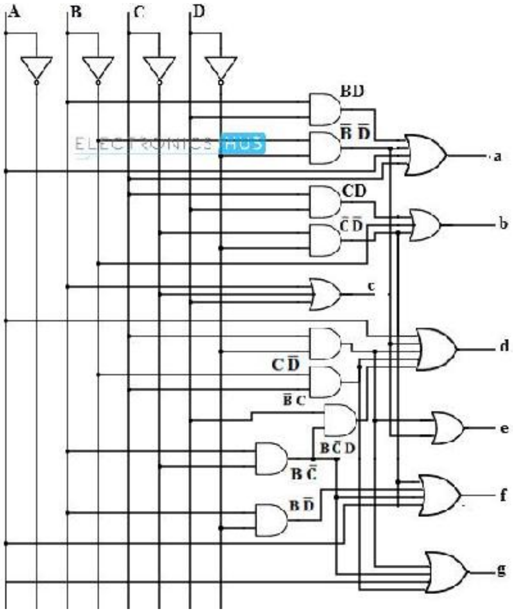
Decimal	BCD Inputs				7 Segment Outputs						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	X	X	X	X	0	0	0	0	0	0	0
11	X	X	X	X	0	0	0	0	0	0	0
12	X	X	X	X	0	0	0	0	0	0	0
13	X	X	X	X	0	0	0	0	0	0	0
14	X	X	X	X	0	0	0	0	0	0	0
15	X	X	X	X	0	0	0	0	0	0	0



BCD to Seven Segment



BCD to Seven Segment



TASK 3

Design BCD to 7 segment

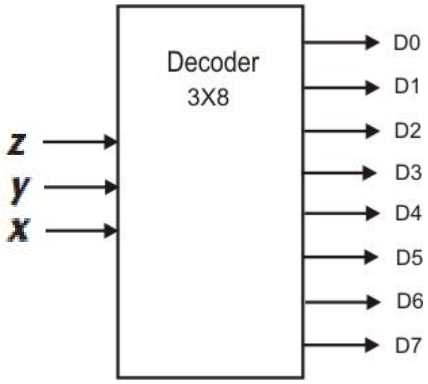
Exact Deadline

Decoders

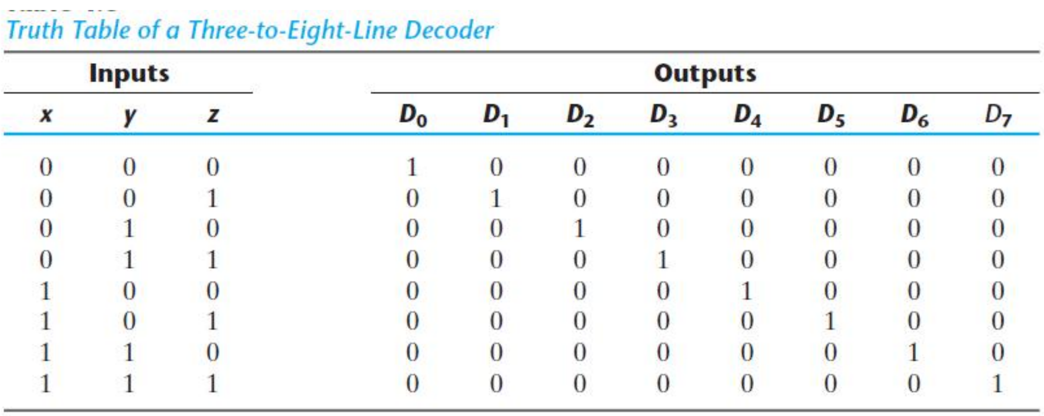
A *decoder* is a combinational circuit that converts binary information *from n input lines to a maximum of 2^n unique output lines.*

Example 3 * 8 Decoder

Examples
2*4
3*8
4*16
5 * 32
Decoder



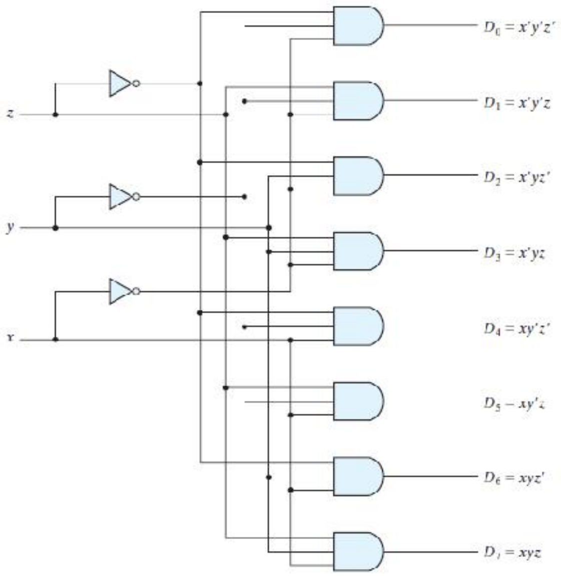
Decoders



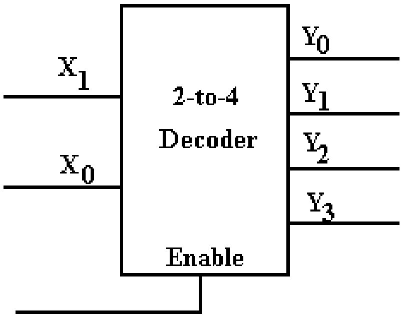
Truth Table of a Three-to-Eight-Line Decoder

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

Decoders

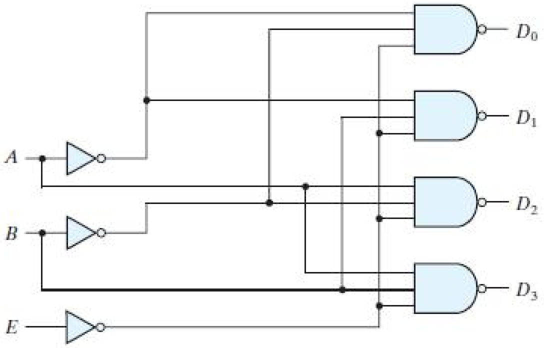


2* 4 Decoder with Enable Active High

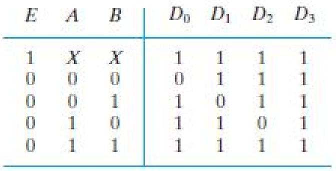


Enable	X_1	X_0	Y_0	Y_1	Y_2	Y_3
0	d	d	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

2* 4 Decoder with Enable Active Low



(a) Logic diagram

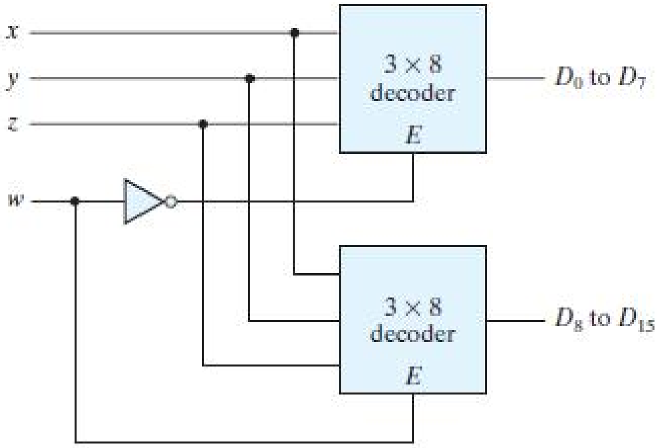


E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

4 × 16 decoder constructed with two 3 × 8 decoders

4*16 Decoders by using 3*8



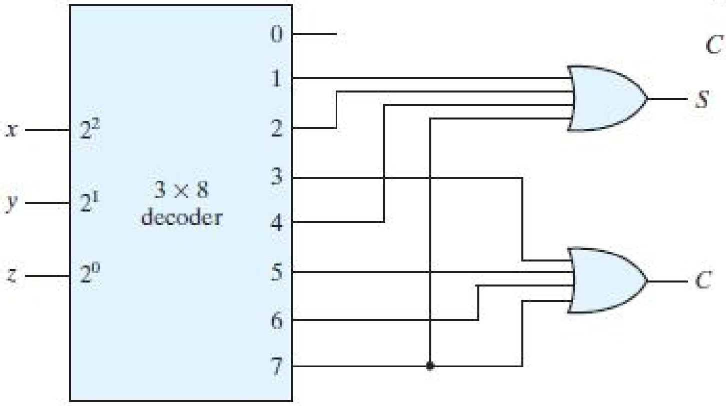
w	z	y	x
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

4 × 16 decoder constructed with two 3 × 8 decoders

Implementation of a full adder with a decoder

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$



From the truth table of the full adder (see Table 4.4), we obtain the functions for the combinational circuit in sum-of-minterms form:

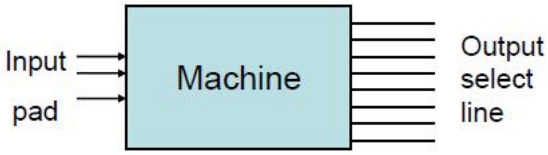
Implementation of a full adder with a decoder

CH 4 : Combinational Circuit

Encoders, Multiplexers, and Three State Gates.

Decoders Applications

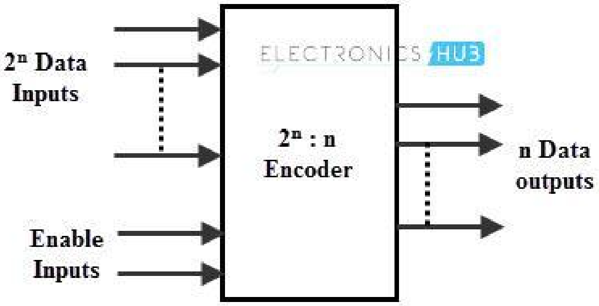
Need to activate only one product:
–1: activated (product released)
–0: not activated
•Only **one slot** can be activated at a time.
Machine It could be modeled as follows



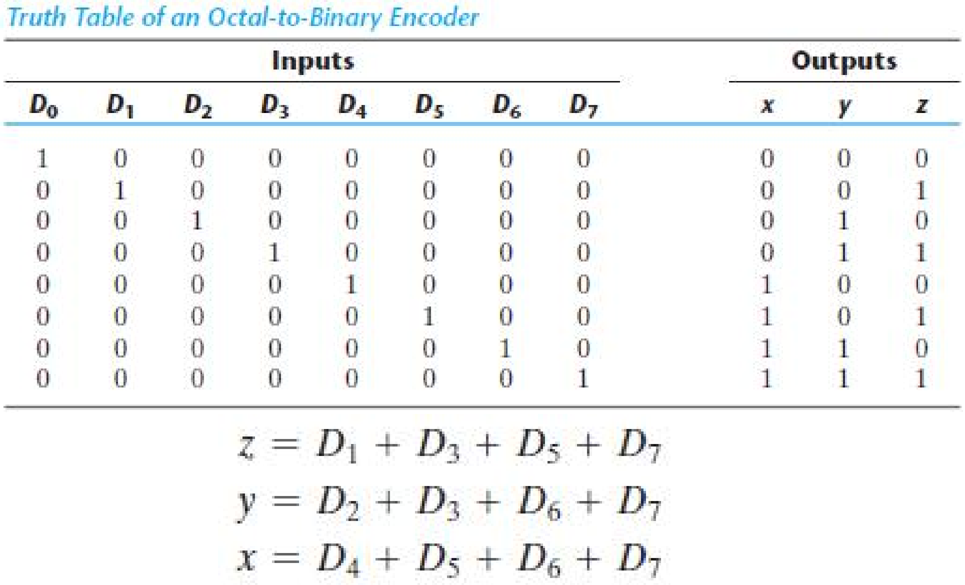
Encoders

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2*n* (or fewer) input lines and *n* output lines.

The output lines generate the binary equivalent of the input line whose value is 1.



8 * 3 Encoder

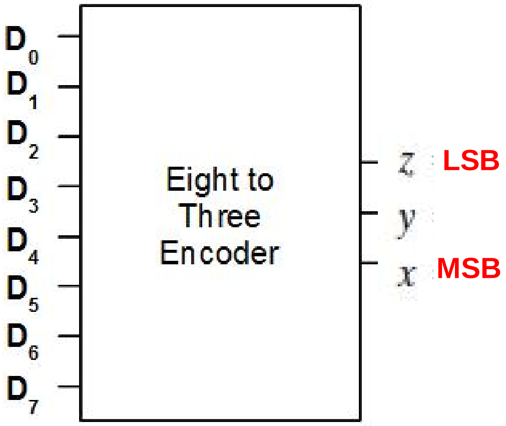


Truth Table of an Octal-to-Binary Encoder

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

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

8 * 3 Encoder

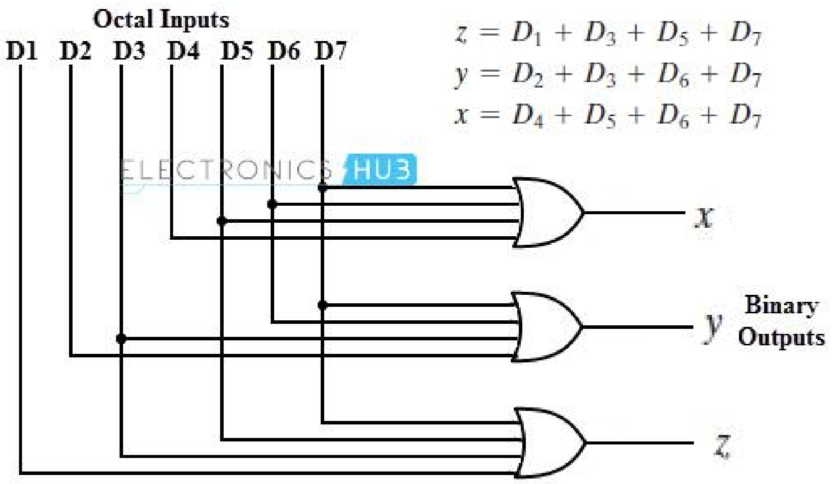


LSB

MSB

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

8 * 3 Encoder

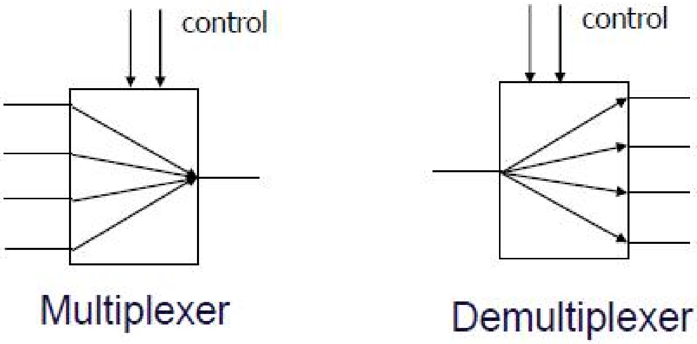


$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Multiplexers

Making Connections

- •Direct point-to-point connections between gates
- –Wires we've seen so far
- •Route one of many inputs to a single output --- multiplexer
- •Route a single input to one of many outputs --- demultiplexer



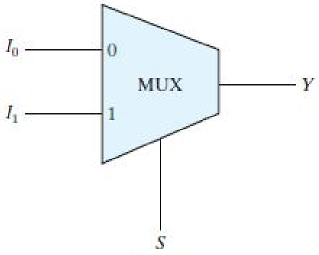
Multiplexers

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

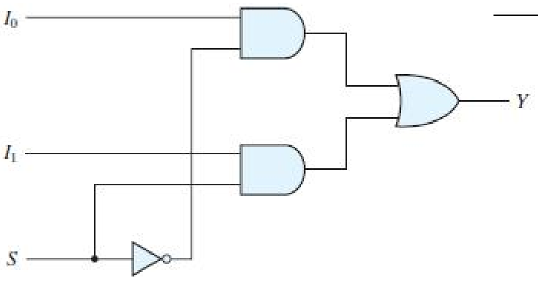
•Multiplexers: general concept

–2^n data inputs, n control inputs (called "selects"), 1 output
–Used to connect 2n points to a single point
–Control signal pattern forms binary index of input connected to output

2*1 Multiplexer

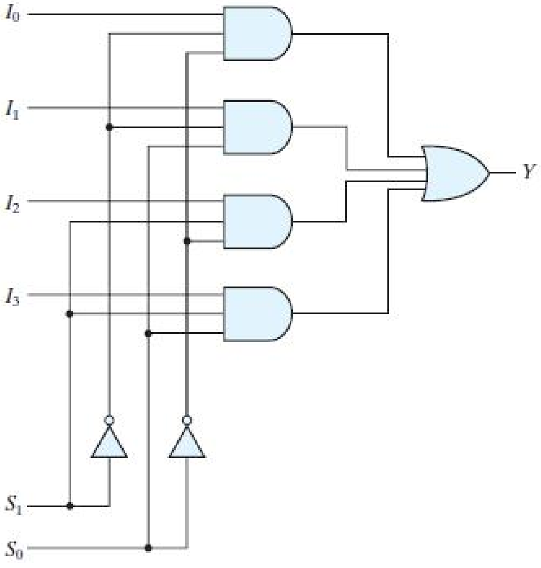


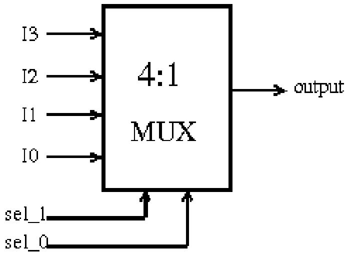
s	I_1	I_0	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$$Y = S' I_0 + S I_1$$

4*1 Multiplexer



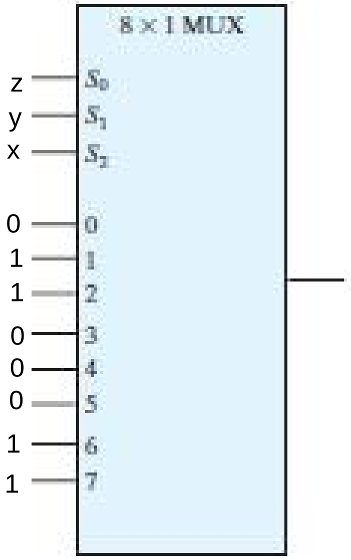


S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Boolean Function Implementation

$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$

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

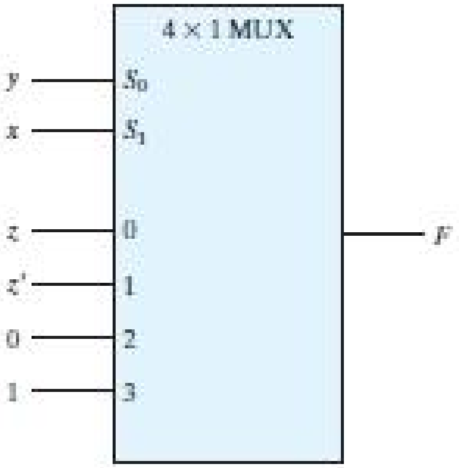


Boolean Function Implementation

$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

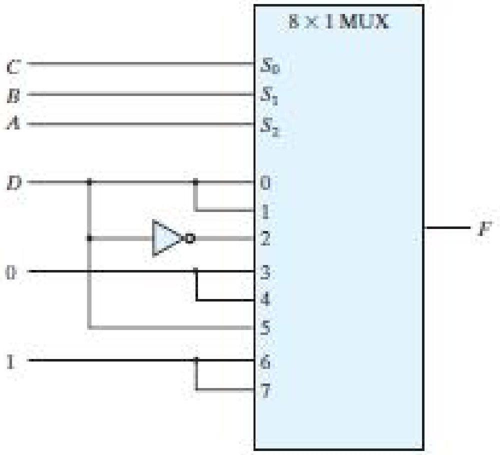


(b) Multiplexer implementation

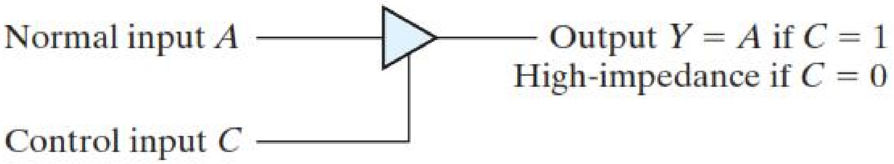
Boolean Function Implementation

$$F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$$

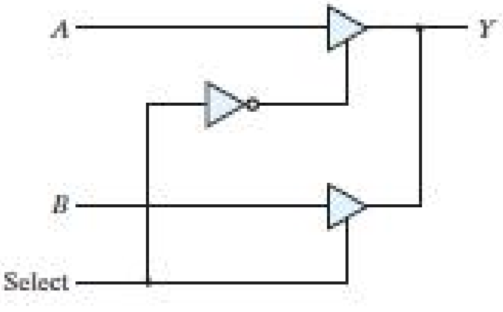
A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = D'
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



Three-State Gates



Multiplexers with three-state gates



(a) 2-to-1-line mux

Introduction

Sequential Logic Circuit

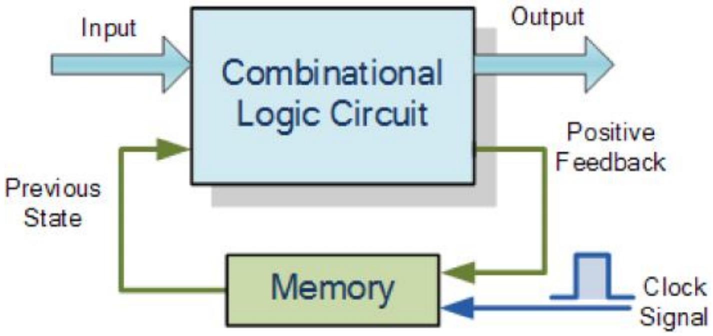
- **A circuit with memory, whose outputs depend on the current input and the sequence of past outputs, is called a sequential circuit.**

- **The behaviour of such a circuit may be described by a state table that specifies its output and next state as functions of its current state and input.**

It consists of a combinational circuit to which storage elements are connected to form a feedback path. The storage elements are devices capable of storing binary information.

Introduction

sequential circuit



Sequential Logic Circuit

Sequential Logic Circuit

Types of Sequential Circuits

•Two types of sequential circuits:

•**Synchronous**: The behavior of the circuit depends on the input signal at discrete instances of time *(also called clocked)*

•**Asynchronous**: The behavior of the circuit depends on the input signals at any instance of time and the order of the inputs change

•A combinational circuit with feedback

Storage elements

Storage elements

•**What's required from storage element?**

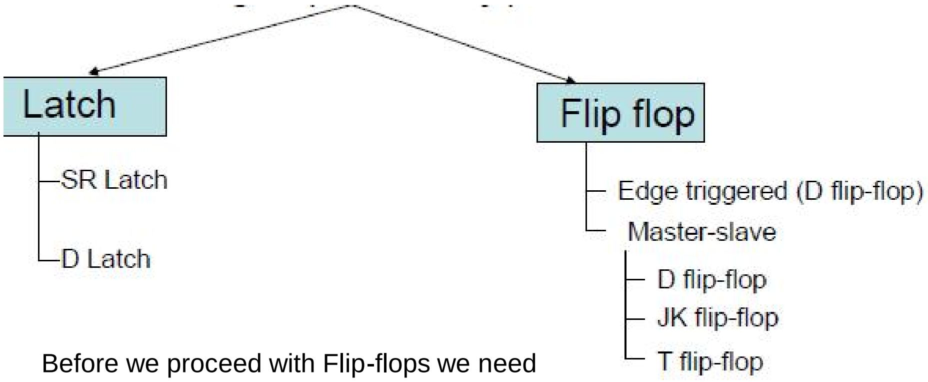
–Store data (hold)

–Accept writing a new data (write)

–Read the stored data

Storage elements

•Types of storage (memory) elements



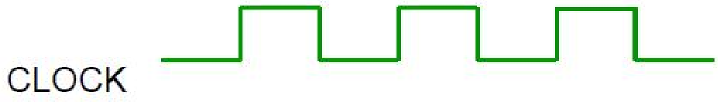
Before we proceed with Flip-flops we need to define, What’s the meaning of “Clock”?

Clock

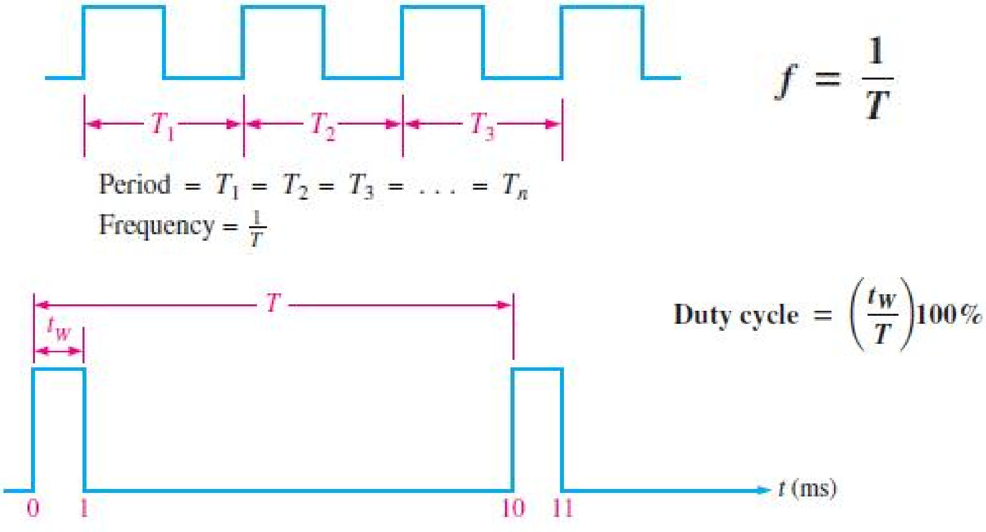
The Term Clock

•In electronics, a **clock signal** is a particular type of signal that oscillates between a high and a low state and is utilized to coordinate actions of circuits. A clock signal is produced by a clock generator.

•Although more complex arrangements are used, the most common clock signal is in the form of a square wave with a 50% duty cycle, usually with a fixed, constant frequency. Circuits using the clock signal for synchronization may become active at either the rising edge, falling edge of the clock Cycle.



Clock



$$f = \frac{1}{T}$$

$$\text{Period} = T_1 = T_2 = T_3 = \ldots = T_n$$

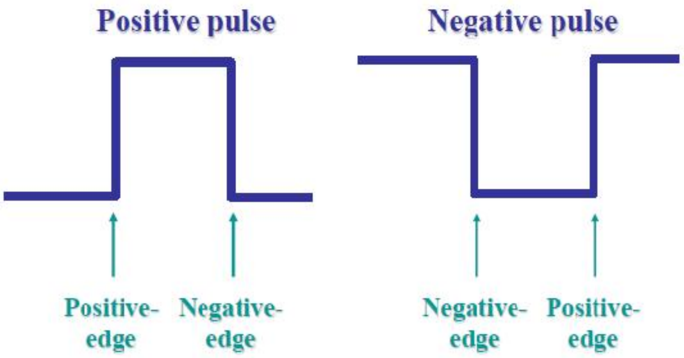
$$\text{Frequency} = \frac{1}{T}$$

$$\text{Duty cycle} = \left(\frac{t_W}{T}\right)100\%$$

clock pulses

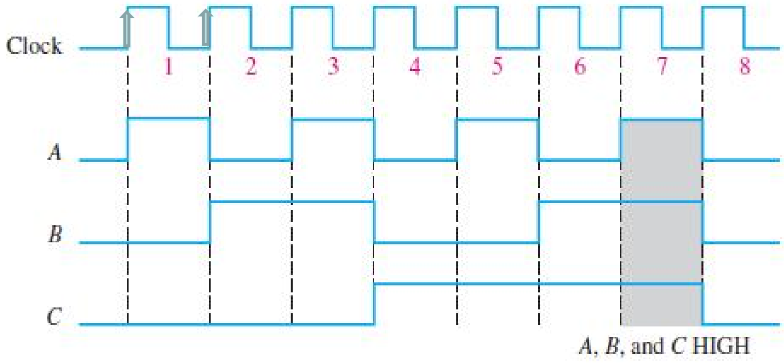
What's the clock pulses?

–**Definition of clock-pulse transition :**



Timing Diagrams

A **timing diagram** is a graph of digital waveforms showing the actual time relationship of two or more waveforms and how each waveform changes in relation to the others

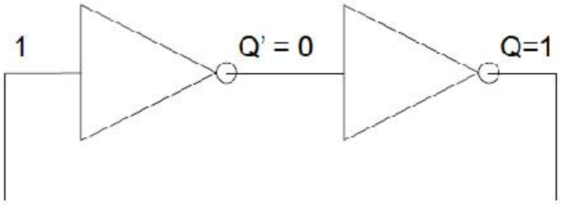


Basic memory element

Storage elements

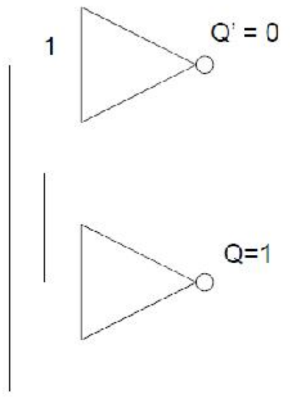
- What's the Basic idea for the memory cell?

  - Basic memory element consists of two cascaded inverters and the output of the last inverter is fed back to the input of the first inverter.
  - Q and Q' are the outputs of the memory element.
  - This memory element will always store one bit.
  - This cell called **LATCH**



Basic memory element

How to write in this cell a new value? We need a special technique to write a new value into the memory.



Storage Elements: SR latches

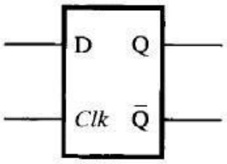
Storage
elements that operate with signal levels (rather than signal transitions) are referred to as latches ; *those controlled by a clock transition are flip-flops* .

D Latch

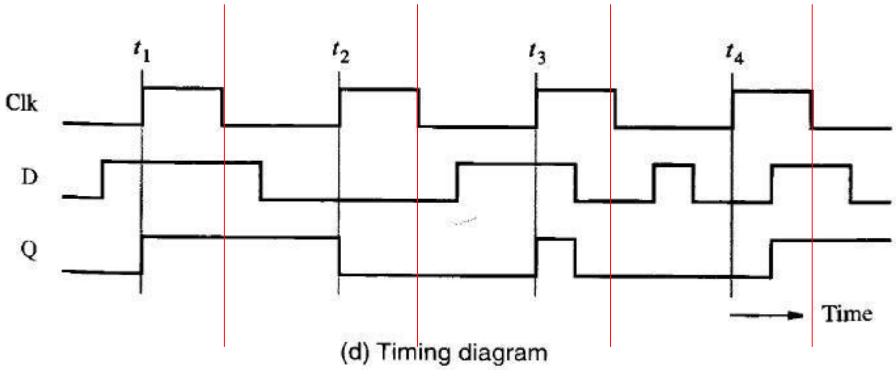
Storage Elements: D latches

Clk	D	Q(*t* + 1)
0	x	Q(*t*)
1	0	0
1	1	1

(b) Characteristic table

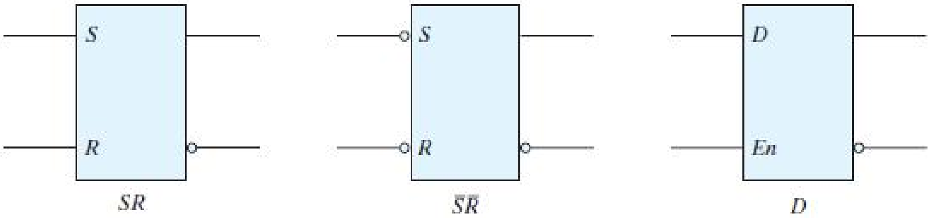


(c) Graphical symbol

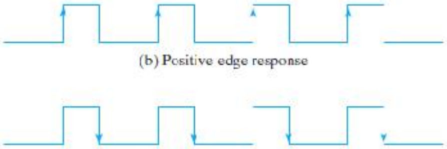


(d) Timing diagram

Storage Elements: latches

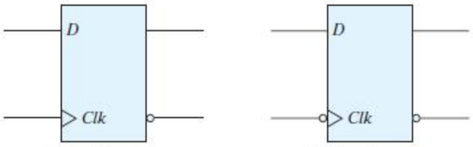


Graphic symbols for latches



(b) Positive edge response

(c) Negative-edge response

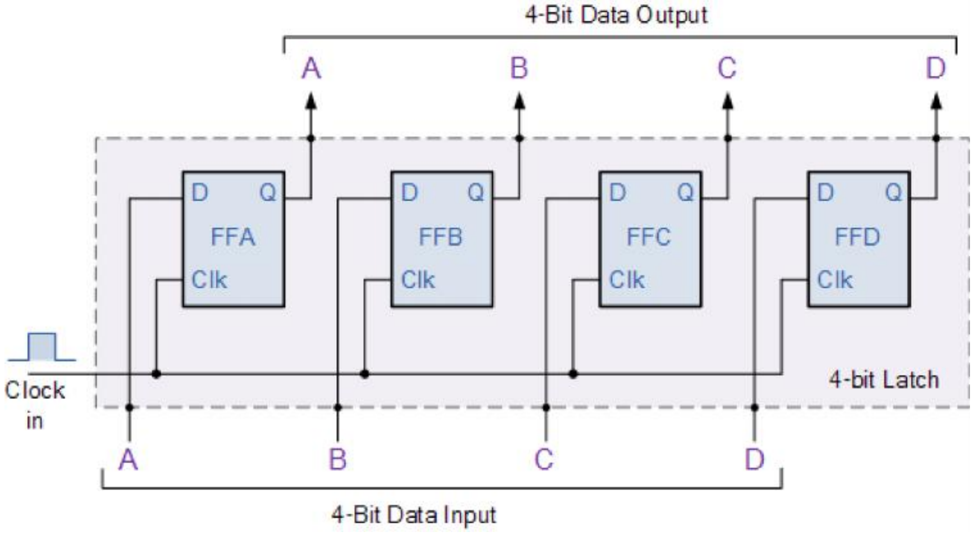


(a) Positive-edge

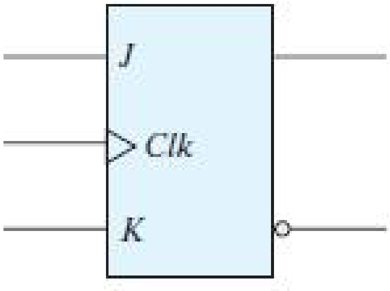
(a) Negative-edge

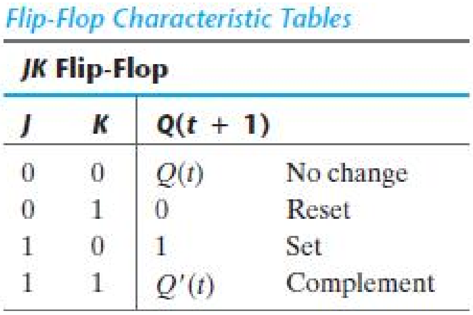
4-bit Data Latch

A data latch can be used as a device to hold or remember the data present on its data input, thereby acting a bit like a single bit memory device and IC’s



JK Flip-Flops





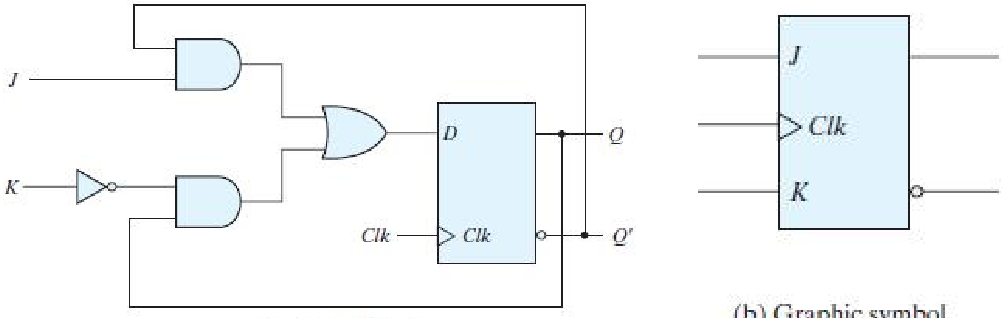
Flip-Flop Characteristic Tables

JK Flip-Flop

J	*K*	**Q(*t* + 1)**	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

JK Flip-Flops

Other Flip-Flops

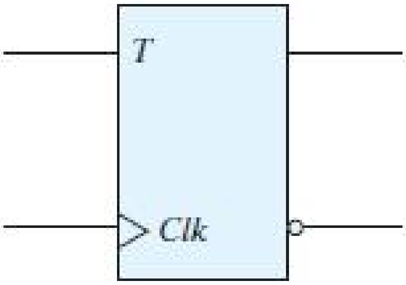


(a) Circuit diagram

(b) Graphic symbol

$$D = JQ' + K'Q$$

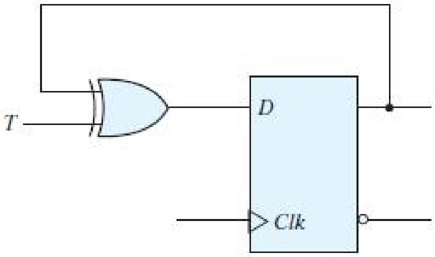
T Flip-Flops



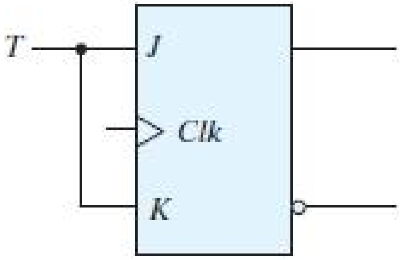
***T* Flip-Flop**

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

T Flip-Flops



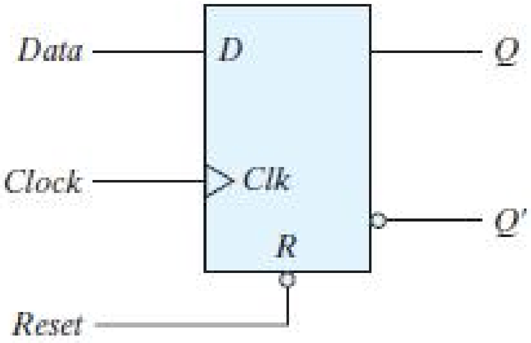
(b) From *D* flip-flop



(a) From *JK* flip-flop

$$D = T \oplus Q = TQ' + T'Q$$

D Flip-Flop with SYN Reset



(b) Graphic symbol

R	*Clk*	*D*	*Q*	*Q'*
0	X	X	0	1
0	↑	0	0	1
0	↑	1	1	0

(b) Function table

Digital Design

CH 5 : Synchronous Sequential Logic Circuits

Registers and Counters

How could you describe combinational circuit?

- Truth table
- Logic function between input and output
- F = A+B => if A= 1 and B= 0 F= 1

if A =0 and B=0 F=0 (not depends on the pervious value of F)

How could you describe a sequential circuit?

•The sequential circuit output is now function in the inputs and past outputs •So, we need a tool to help us describe the behavior of the circuit.

Finite State Machine (FSM)

Finite State Machine (FSM)

•***F***inite ***S***tate ***M***achine is a tool to model the desired behavior of a sequential system.

•The designer has to develop a finite state model of the system behavior and then designs a circuit that implements this model

•A FSM consists of several ***states***. ***Inputs*** into the machine are combined with the current state of the machine to determine the new state or ***next*** state of the machine.

•Depending on the state of the machine, outputs are generated based on either the state or the state and inputs of the machine

How to describe FSM?

–**State equation** (transition equation) input variables, present states, next states equation

–**State table** input variables, present states □ next states, truth table

–**State diagram**

Finite State Machine (FSM)

State tables

•Similar to the truth table

•Doesn’t contain the system clock when specifying its transitions (it is implicit that transitions occur only when allowed by clock)

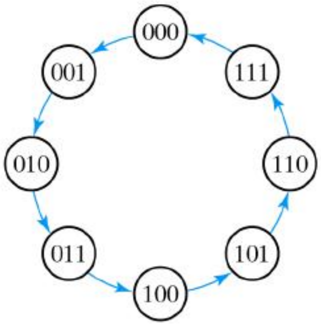
•Unless different stated, all the transitions are occurring on the positive edge of the clock

Present State	Inputs	Next State	Outputs

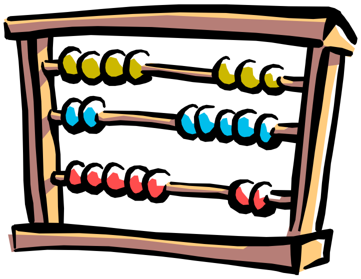
Finite State Machine (FSM)

State diagram

- Graphical representation of the state table
- Each state is represented by a circle **vertex**
- Each row of the state table is represented as a directed **arc** from present state vertex to the next state vertex
- In this diagram, the outputs are associated with the states



Counters



Counters are sequential circuits which "count" through a specific state sequence. They can count up, count down, or count through other fixed sequences.

Counters

A *counter* is a sequential circuit that goes through a predetermined sequence of states upon the application of clock pulses.

Counters are categorized as:

Synchronous Counter:

All FFs receive the common clock pulse, and the change of state is determined from the present state.

Ripple Counters: Asynchronous

The FF output transition serves as a source for triggering other FFs. No common clock.

Counters

Counters are a specific type of sequential circuit

The state serves as the “output” (Moore)

A counter that follows the binary number sequence is called a binary counter

n-bit binary counter: n flip-flops, count in binary from 0 to 2^n-1

Counters are available in two types:

Synchronous Counters

Ripple Counters

Synchronous Counters:

A common clock signal is connected to the C input of each flip-flop

Synchronous Binary Counters

Synchronous Counters

Clock is directly connected to the flip-flop clock inputs

Logic is used to implement the desired state sequencing

The design procedure for a binary counter is the same as any other synchronous sequential circuit.

Most efficient implementations usually use D- FFs or T-FFs or JK-FFs. We will examine T and D flip-flop designs.

Synchronous Binary Counters

Synchronous Binary Up Counter Using D flip flop

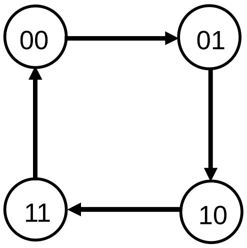
The output value increases by one on each clock cycle

After the largest value, the output “wraps around” back to 0

Using two bits, we’d get something like this:

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

State tables



State diagram

Synchronous Binary Counters

Synchronous Binary Up Counter Using D flip flop

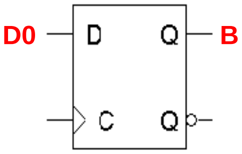
Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0
		DA	DB

A \ B	0	1
0	A'B' 0	1 A'B 1
1	1 AB' 2	AB 3

D1= A'B + AB'

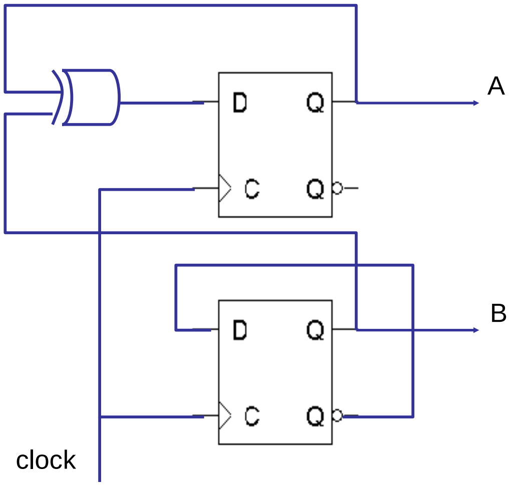
A \ B	0	1
0	1 A'B' 0	A'B 1
1	1 AB' 2	AB 3

D0= B'



Synchronous Binary Counters

Synchronous Binary Up Counter Using D flip flop



D1= A'B + AB'

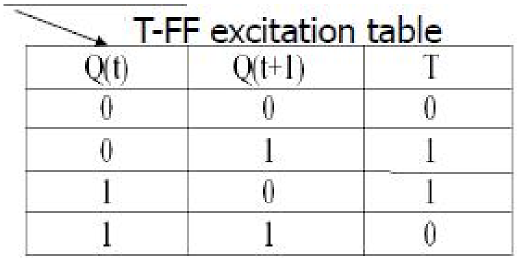
D0= B'

Design of Counters

Design Counter Using T flip-flop that counts from 0-1-2-3-4-5-6-7

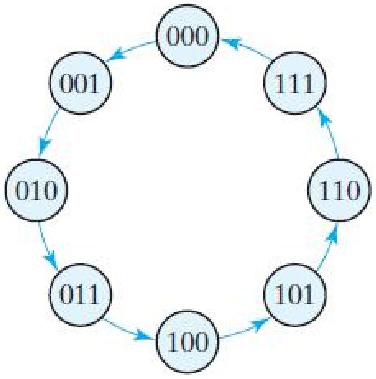
1- State diagram of three-bit binary counter

Using T-type FFs, design a 3-bits binary counter that can count in binary from 0 to 7 with step 1

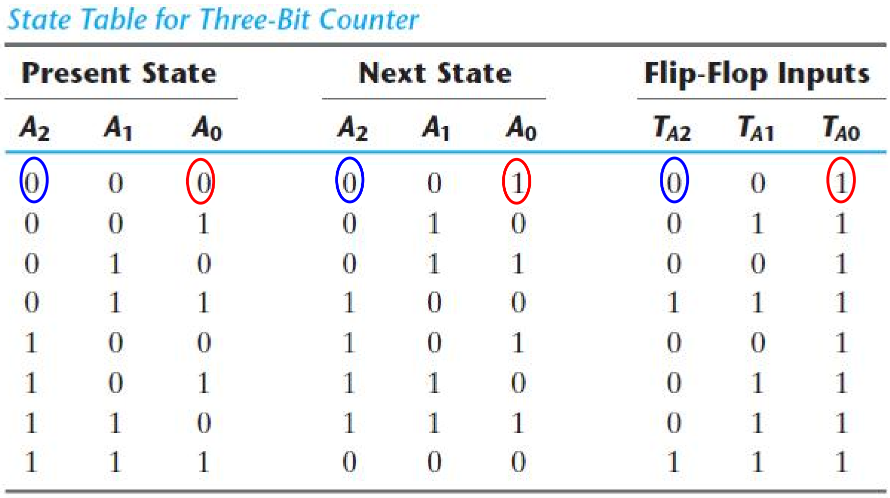


T-FF excitation table

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



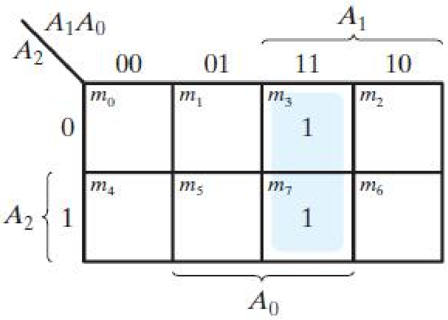
Synchronous Binary Counters



State Table for Three-Bit Counter

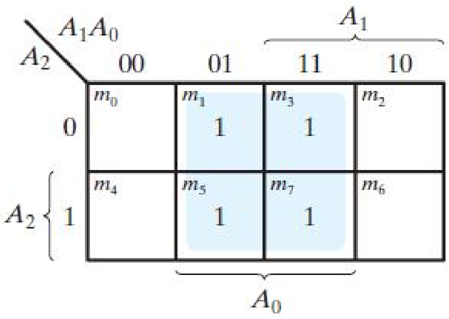
Present State			Next State			Flip-Flop Inputs		
A_2	A_1	A_0	A_2	A_1	A_0	T_{A2}	T_{A1}	T_{A0}
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	1	1
1	1	1	0	0	0	1	1	1

Synchronous Binary Counters



A_2 \ A_1A_0	00	01	11	10
0	m_0	m_1	m_3 1	m_2
1	m_4	m_5	m_7 1	m_6

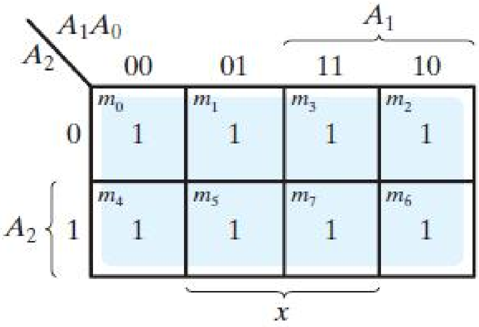
$T_{A2} = A_1A_0$



A_2 \ A_1A_0	00	01	11	10
0	m_0	m_1 1	m_3 1	m_2
1	m_4	m_5 1	m_7 1	m_6

$T_{A1} = A_0$

Synchronous Binary Counters



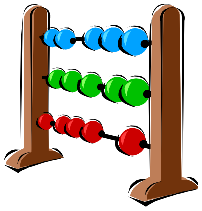
A_2 \ A_1A_0	00	01	11	10
0	m_0 1	m_1 1	m_3 1	m_2 1
1	m_4 1	m_5 1	m_7 1	m_6 1

$T_{A0} = 1$

Summary

Counters serve many purposes in sequential logic design

There are lots of variations on the basic counter



Some can increment or decrement

An enable signal can be added

The counter's value may be explicitly set

There are also several ways to make counters

You can follow the sequential design principles to build counters from scratch

You could also modify or combine existing counter devices

Exercise

Design Counter using T flip flops that counts from 0-3-7-4-5-1-6-2

Registers

A common sequential device: Registers

They're a good example of sequential analysis and design

They are also frequently used in building larger sequential circuits

Registers hold larger quantities of data than individual flip-flops

Registers are central to the design of modern **processors**

There are many different kinds of registers

We'll show some applications of these special registers

What good are registers?

Flip-flops are limited because they can store only one bit

- We had to use two flip-flops for our two-bit counter examples
- Most computers work with integers and single-precision floating-point numbers that are 32-bits long

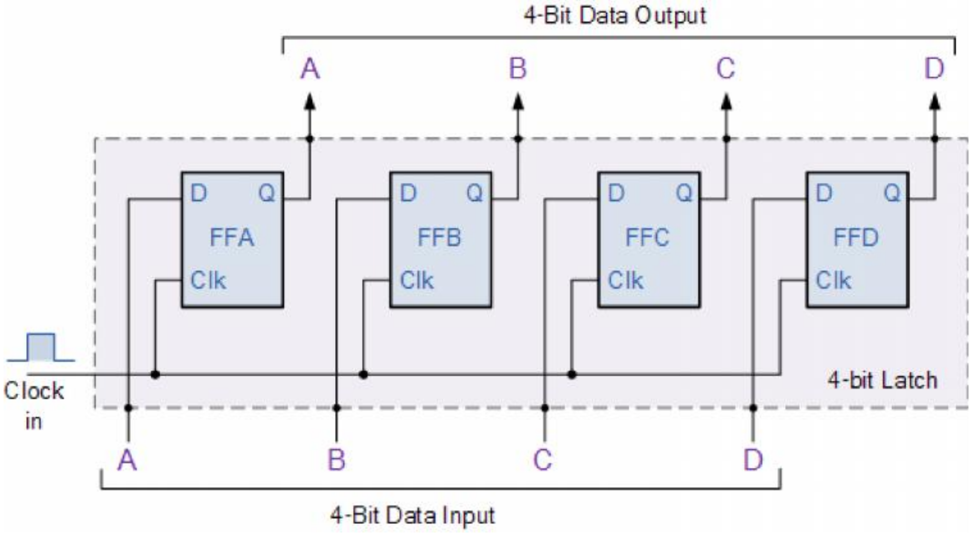
A register is an extension of a flip-flop that can store multiple bits

Registers are commonly used as temporary storage in a processor

- They are faster and more convenient than main memory
- More registers can help speed up complex calculations

4-bit Data Latch

A data latch can be used as a device to hold or remember the data present on its data input, thereby acting a bit like a single bit memory device and IC’s



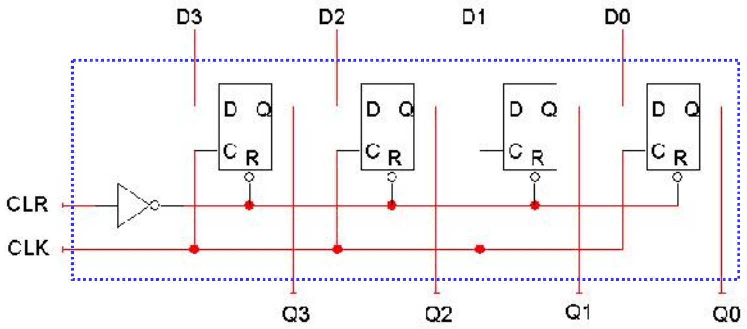
A Basic Register (Parallel in/out)

Basic registers are easy to build. We can store multiple bits just by putting a bunch of flip-flops together!

A 4-bit register is given below

This register uses D flip-flops, so it's easy to store data without worrying about flip-flop input equations

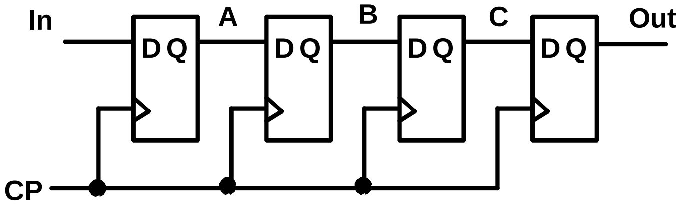
All the flip-flops share a common CLK and CLR signal



A Basic Register (Serial in/out)

Shift Registers move data laterally within the register toward its MSB or LSB position

In the simplest case, the shift register is simply a set of D flip-flops connected in a row like this:



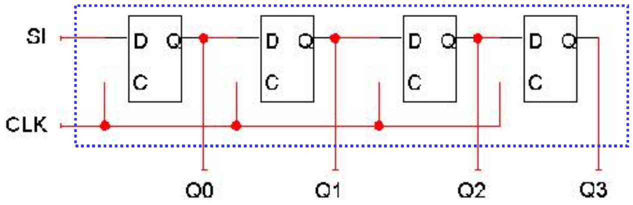
Data input, In, is called a *serial input* or the *shift right input*.

Data output, Out, is often called the *serial output*.

The vector (A, B, C, Out) is called the *parallel output*.

Shift Register

A shift register “shifts” its output once every clock cycle.



$Q_0(t+1) = SI$
$Q_1(t+1) = Q_0(t)$
$Q_2(t+1) = Q_1(t)$
$Q_3(t+1) = Q_2(t)$

SI is an input that supplies a new bit to shift “into” the register

For example, if on some positive clock edge we have:

SI = 1

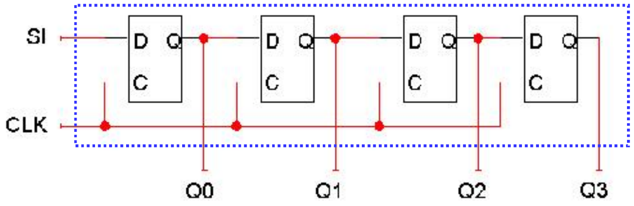
Q_0-Q_3 = 0110

then the next state will be:

Q_0-Q_3 = 1011

The current Q_3 (0 in this example) will be lost on the next cycle

Shift direction



$Q_0(t+1) = SI$
$Q_1(t+1) = Q_0(t)$
$Q_2(t+1) = Q_1(t)$
$Q_3(t+1) = Q_2(t)$

Present Q_0-Q_3	SI	Next Q_0-Q_3
ABCD	X	XABC

The circuit and example make it look like the register shifts “right.”

Present Q_3-Q_0	SI	Next Q_3-Q_0
DCBA	X	CBAX

But it really depends on your interpretation of the bits. If you consider Q3 to be the most significant bit instead, then the register is shifting in the *opposite* direction!

Serial data transfer

One application of shift registers is converting between “serial data” and “parallel data”

Computers typically work with multiple-bit quantities

- ASCII text characters are 8 bits long
- Integers, single-precision floating-point numbers, and screen pixels are up to 32 bits long

But sometimes it’s necessary to send or receive data serially, or one bit at a time. Some examples include:

- Input devices such as keyboards and mice
- Output devices like printers
- Any serial port, USB or Firewire device transfers data serially
- Recent switch from Parallel ATA to Serial ATA in hard drives

Receiving serial data

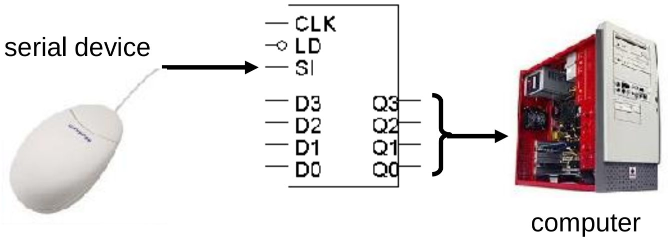
To *receive* serial data using a shift register:

- The serial device is connected to the register’s SI input
- The shift register outputs Q3-Q0 are connected to the computer

The serial device transmits one bit of data per clock cycle

- These bits go into the SI input of the shift register
- After four clock cycles, the shift register will hold a four-bit word

The computer then reads all four bits at once from the Q3-Q0 outputs.



Sending data serially

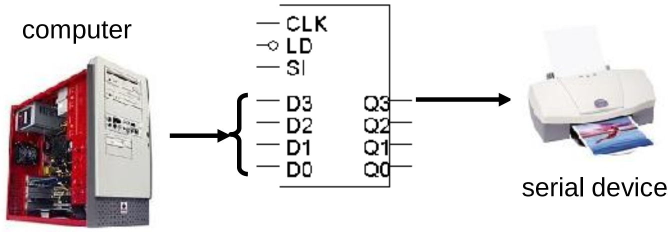
To *send* data serially with a shift register, you do the opposite:

- The CPU is connected to the register's D inputs
- The shift output (Q3 in this case) is connected to the serial device

The computer first stores a four-bit word in the register, in one cycle

The serial device can then read the shift output

- One bit appears on Q3 on each clock cycle
- After four cycles, the entire four-bit word will have been sent



Registers summary

A register is a special state machine that stores multiple bits of data

Several variations are possible:

- **Parallel loading to store data into the register**
- **Shifting the register contents either left or right**
- **Counters are considered a type of register too!**

One application of shift registers is converting between serial and parallel data

Most programs need more storage space than registers provide

- **We'll introduce RAM to address this problem**

Registers are a central part of modern processors

TASKs - Implementing the registers and Counters.

Best Wishes

Sheet (1)

1- Convert the following numbers with the indicated bases to decimal:
(a) $(4310)_5$ (b) $(198)_{12}$ (c) $(435)_8$ (d) $(345)_6$

2- Express the following numbers in decimal:
(a) $(10110.0101)_2$ (b) $(16.5)_{16}$ (c) $(26.24)_8$
(d) $(DADA.B)_{16}$ (e) $(1010.1101)_2$

3- What is the largest binary number that can be expressed with 16 bits? What are the equivalent decimal and hexadecimal numbers?

4- Determine the base of the numbers in each case for the following operations to be correct:
(a) 14/2 = 5 (b) 54/4 = 13 (c) 24 + 17 = 40.

5- The solutions to the quadratic equation x^2-11x+22=0 are x = 3 and x = 6. What is the base of the numbers?

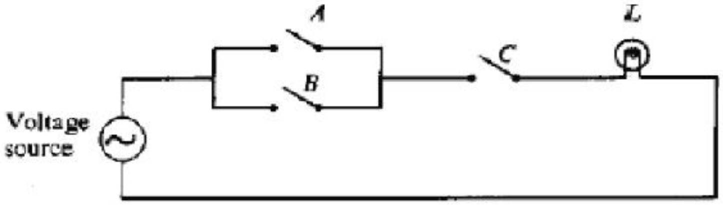
6- (a) Find the 16's complement of C3DF. (b) Convert C3DF to binary.
(c) Find the 2's complement of the result in (b).
(d) Convert the answer in(c) to hexadecimal and compare with the answer in (a).

7- Perform subtraction on the given unsigned binary numbers using the 2's complement of the subtrahend. Where the result should be negative, find its 2's complement and affix a minus sign.
(a) 10011 - 10010 (b) 100010 - 100110
(c) 1001 - 110101 (d) 101000 - 10101

8- Represent the unsigned decimal numbers 791 and 658 in BCD, and then show the steps necessary to form their sum.

9- The state of a 12-bit register is 100010010111. What is its content if it represents (a) Three decimal digits in BCD?
(b) Three decimal digits in the excess-3 code?
(c) Three decimal digits in the 84-2-1 code?
(d) A binary number?

10- Express the switching electrical circuit below in binary logic form?



Sheet (2)

1- Simplify the following Boolean expressions:

(a)* $ABC + A'B + ABC'$ (b)* $x'yz + xz$

(c)* $(x + y)'(x' + y')$ (d)* $xy + x(wz + wz')$

(e)* $(BC' + A'D)(AB' + CD')$ (f) $(a' + c')(a + b' + c')$

2- Find the complement of F = wx + yz; then show that $FF' = 0$ and $F + F' = 1$.

3- Find the complement of the following expressions:

(a)* $xy' + x'y$ (b) $(a + c)(a + b')(a' + b + c')$

(c) $z + z'(v'w + xy)$

4- Given the Boolean functions F1 and F 2 , show that (a) The Boolean function E = F1 + F2 contains the sum of the minterms of F1 and F2. (b) The Boolean function G = F1F2 contains only the minterms that are common to F1 and F2.

5- Implement the Boolean function $F = xy + x'y' + y'z$

(a) With AND, OR, and inverter gates. (b) With OR and inverter gates
(c) With AND and inverter gates. (d) With NAND and inverter gates.
(e) With NOR and inverter gates.

6- For the Boolean function $F = xy'z + x'y'z + w'xy + wx'y + wxy$

(a) Obtain the truth table of F. (b) Draw the logic diagram, using the original Boolean expression. (c) Use Boolean algebra to simplify the function to a minimum number of literals. (d) Obtain the truth table of the function from the simplified expression and show that it is the same as the one in part (a).
(e) Draw the logic diagram from the simplified expression, and compare the total number of gates with the diagram of part (b).

7- Obtain the truth table of the following functions, and express each function in sum-of-minterms and product-of-maxterms form:

(a)* $(b + cd)(c + bd)$ (b) $(cd + b'c + bd')(b + d)$

(c) $(c' + d)(b + c')$ (d) $bd' + acd' + ab'c + a'c'$

Sheet (2)

8- Express the following function as a sum of minterms and as a product of maxterms:

$$F(A, B, C, D) = B'D + A'D + BD$$

9- Express the complement of the following functions in sum-of-minterms form:

(a) $F(A,B,C,D) = \sum(2,4,7,10,12,14)$

(b) $F(x,y,z) = \prod(3,5,7)$

10- Determine whether the following Boolean equation is true or false.

$$x'y' + x'z + x'z' = x'z' + y'z' + x'z$$

11- Convert each of the following expressions into sum of products and product of sums:

(a) $(u + xw)(x + u'v)$

(b) $x' + x(x + y')(y + z')$

12- Write the following Boolean expressions in sum of products form:

$$(b + d)(a' + b' + c)$$

13- Write the following Boolean expression in product of sums form:

$$a'b + a'c' + abc$$

14- Show that the dual of the exclusive-OR is equal to its complement.

15- Show that the simplest form of

$$XYZ' + XY'Z + X'YZ + XYZ$$

is $XY + XZ + YZ$

Sheet (3)

1- Simplify the following Boolean expressions:

(a)* $F(x, y, z) = \Sigma(0, 1, 5, 7)$

(b)* $F(x, y, z) = \Sigma(1, 2, 3, 6, 7)$

(c) $F(x, y, z) = \Sigma(2, 3, 4, 5)$

(d) $F(x, y, z) = \Sigma(1, 2, 3, 5, 6, 7)$

(e) $F(x, y, z) = \Sigma(0, 2, 4, 6)$

(f) $F(x, y, z) = \Sigma(3, 4, 5, 6, 7)$

2- Simplify the following Boolean expressions, using three-variable maps:

(a)* $xy + x'y'z' + x'yz'$

(b)* $x'y' + yz + x'yz'$

(c)* $F(x, y, z) = x'y + yz' + y'z'$

(d) $F(x, y, z) = x'yz + xy'z' + xy'z$

3- Simplify the following Boolean functions, using Karnaugh maps:

(a)* $F(x, y, z) = \Sigma(2, 3, 6, 7)$

(b)* $F(A, B, C, D) = \Sigma(4, 6, 7, 15)$

(c)* $F(A, B, C, D) = \Sigma(3, 7, 11, 13, 14, 15)$

(d)* $F(w, x, y, z) = \Sigma(2, 3, 12, 13, 14, 15)$

(e) $F(w, x, y, z) = \Sigma(11, 12, 13, 14, 15)$

(f) $F(w, x, y, z) = \Sigma(8, 10, 12, 13, 14)$

4- Simplify the following Boolean expressions, using four-variable maps:

(a)* $A'B'C'D' + AC'D' + B'CD' + A'BCD + BC'D$

(b)* $x'z + w'xy' + w(x'y + xy')$

(c) $A'B'C'D + AB'D + A'BC' + ABCD + AB'C$

(d) $A'B'C'D' + BC'D + A'C'D + A'BCD + ACD'$

5- Find all the prime implicants for the following Boolean functions, and determine which are essential:

(a)* $F(w, x, y, z) = \Sigma(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

(b)* $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 10, 11, 14, 15)$

(c) $F(A, B, C, D) = \Sigma(2, 3, 4, 5, 6, 7, 9, 11, 12, 13)$

(d) $F(w, x, y, z) = \Sigma(1, 3, 6, 7, 8, 9, 12, 13, 14, 15)$

(e) $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$

(f) $F(w, x, y, z) = \Sigma(0, 1, 2, 5, 7, 8, 10, 15)$

Sheet (3)

6- Convert the following Boolean function from a sum-of-products form to a simplified product-of-sums form.

$$F(x, y, z) = \Sigma(0, 1, 2, 5, 8, 10, 13)$$

7- Simplify the following Boolean functions:

(a)* $F(A, B, C, D) = \Pi(1, 3, 5, 7, 13, 15)$

(b) $F(A, B, C, D) = \Pi(1, 3, 6, 9, 11, 12, 14)$

8- Simplify the following expressions to (1) sum-of-products and (2) products-of-sums:

(a)* $x'z' + y'z' + yz' + xy$

(b) $ACD' + C'D + AB' + ABCD$

(c) $(A' + B + D')(A' + B' + C')(A' + B' + C)(B' + C + D')$

(d) $BCD' + ABC' + ACD$

9- Simplify the following Boolean function F, together with the don't-care conditions d, and then express the simplified function in sum-of-minterms form:

(a) $F(x, y, z) = \Sigma(0, 1, 4, 5, 6)$
$d(x, y, z) = \Sigma(2, 3, 7)$

(b)* $F(A, B, C, D) = \Sigma(0, 6, 8, 13, 14)$
$d(A, B, C, D) = \Sigma(2, 4, 10)$

(c) $F(A, B, C, D) = \Sigma(5, 6, 7, 12, 14, 15,)$
$d(A, B, C, D) = \Sigma(3, 9, 11, 15)$

(d) $F(A, B, C, D) = \Sigma(4, 12, 7, 2, 10,)$
$d(A, B, C, D) = \Sigma(0, 6, 8)$

10- Simplify the following functions, and implement them with two-level NAND gate circuits:

(a) $F(A, B, C, D) = AC'D' + A'C + ABC + AB'C + A'C'D'$

(b) $F(A, B, C, D) = A'B'C'D + CD + AC'D$

(c) $F(A, B, C) = (A' + C' + D')(A' + C')(C' + D')$

(d) $F(A, B, C, D) = A' + B + D' + B'C$

Sheet (3)

11- Draw a logic diagram using only two-input NOR gates to implement the following function:

$$F(A, B, C, D) = (A \oplus B)'(C \oplus D)$$

12- Simplify the following functions, and implement them with two-level NOR gate circuits

(a)* $F = wx' + y'z' + w'yz'$

(b) $F(w, x, y, z) = \Sigma(0, 3, 12, 15)$

13- Draw the multiple-level NOR circuit for the following expression:

$$CD(B + C)A + (BC' + DE')$$

14- Draw the multiple-level NAND circuit for the following expression

$$w(x + y + z) + xyz$$

15- Implements the following Boolean function F, using the two-level forms of logic (a) NANDAND, (b) AND-NOR, (c) OR-NAND, and (d) NOR-OR:

$$F(A, B, C, D) = \Sigma(0, 4, 8, 9, 10, 11, 12, 14)$$

16- Derive the circuits for a three-bit parity generator and four-bit parity checker using an odd parity bit.

Best Wishes

References

Main Reference:

1. M. Morris Mano,” **Digital Design**,” 5th edition.
2. Thomas Floyd,” **Digital Fundamentals**,” 10th edition.
3. Hany Ahmed**, “Lecture Notes: Logic Design ”,** 2024