Digital Techniques

Lecture 1

1st Class

Digital Techniques

Digital Computer and Digital System:

Digital computer is a part of digital system, it based on binary system. A block diagram of digital computer is shown in figure (1):

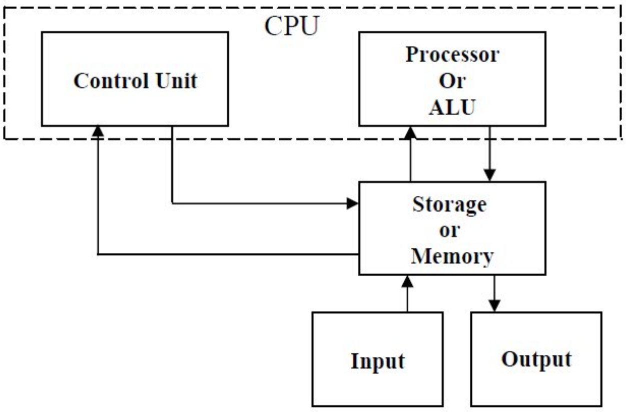


Figure (1) Digital Computer

Where
CPU is the Central Processing Unit.
CU is the Control Unit.
ALU is the Arithmetic Logic Unit.

- The processor when combined with the control unit form a component referred to as CPU.
- Storage unit stores programs as well as input, output and intermediate data.

- The processor unit performs arithmetic and other data processing tasks as specified by the program.
- The control unit supervised the flow of information between various units.

- The program and data prepared by the user are transformed into the memory unit by means of input devices such as: punch-card reader, keyboard and scanner …etc.
- The output unit presents the results of the computation to the user in a form that the user understands it (compatible with the user) such as: card punching, printer and magnetic tape …etc.

Binary System:

Since most of the electronic circuit used to contract digital computer are in herently binary in operation. Number system based on ones and zeroes is called the binary system (because there are only two possible digits). The binary system is the language of digital computer.

In general, quantities are represented as:

N= a0 2^0 + a1 2^1 + a2 2^2 + ….+ an 2^n

Where each coefficient can take only two value either 0 or 1.

EX:

Decimal (base 10)	Binary (base 2)
0 0	0 0 0 0 0
0 1	0 0 0 0 1
1 0	0 1 0 1 0
2 1	1 0 1 0 1

- For N bits, the number can take an 2^N values varying from 0 — 2^N-1.

EX: N=2 bits, can take $2^2 = 4$ values, varying from 0 — 3.

Decimal	Binary
0	0 0
1	0 1
2	1 0
3	1 1

Number System:

A number system is nothing more than a code representing quantity. The most familiar number system is the decimal system that uses 10 basic symbols. Beside decimal system there are binary, octal and hexadecimal systems, binary use only 2 digits while octal system use 8 digits and hexadecimal use 16 digits.

Radix	Number System	Basic digits
2	Binary	0, 1
8	Octal	0, 1,2,3,4,5,6,7
10	Decimal	0,1,2,3,4,5,6,7,8,9
16	Hexadecimal	0,1,2,3,4,5,6,7,8,9,10,A,B,C,D,E,F

Number representation:

- Binary Number: the decimal number can be present in binary by arranging the 1 and 0 under weight of the binary system to get the decimal number.

EX:

	32	16	8	4	2	1
	….. 2^5	2^4	2^3	2^2	2^1	2^0
Decimal						
4	0	0	0	1	0	0
12	0	0	1	1	0	0
22	0	1	0	1	1	0

- Octal Number: the decimal number can be present in Octal by arranging basic digits according to the octal system to get the decimal number.

EX:

	512	64	8	1
	…. 8^3	8^2	8^1	8^0
Decimal				
4	0	0	0	4
54	0	0	6	6
90	0	1	3	2

- Hexadecimal: the decimal number can be present in hexadecimal by arranging basic digits according to the weight of the hexadecimal system to get the decimal number.

EX:

	4096	256	16	1
	…. 16^3	16^2	16^1	16^0
Decimal				
4	0	0	0	4
25	0	0	1	9
45	0	0	2	D
284	0	1	1	C

Digital Techniques

Lecture 2

1st Class

Digital Techniques

Number Conversions:

The general number system can be written as shown in this equation:
$An\ x^{n} + \ldots + A1\ x^{1} + A0\ x^{0}\ \mathbf{.}\ A\text{-}1\ x^{-1} + \ldots + A\text{-}n\ x^{-n}$

X referred to as the base of the system.

- Binary to Decimal conversion:

<u>EX:</u>

- $(110)_2$ $(?)_{10}$

$$1 \times 2^{2} + 1 \times 2^{1} + 0 \times 2^{0} = 6$$

The answer is $(6)_{10}$

- $(1011.1100)_2\ (?)_{10}$

$$1 \times 2^{3} + 0 \times 2^{2} + 1 \times 2^{1} + 1 \times 2^{0}\ .\ 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} = 11.75$$

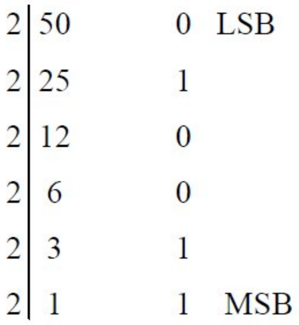
The answer is $(11.75)_{10}$

- Decimal to Binary conversions:

<u>EX</u>:

-

$(50)_{10}$ $(?)_2$



2	50	0	LSB
2	25	1	
2	12	0	
2	6	0	
2	3	1	
2	1	1	MSB

The answer is $(110010)_2$

- $(\ 0.6875\ )_{10}\ (\ ?\ )_2$

$$\begin{array}{rl}
0.6875 \times 2 = 1.375 & 1 \\
0.375 \times 2 = 0.75 & 0 \\
0.75 \times 2 = 1.5 & 1 \\
0.5 \times 2 = 1.0 & 1
\end{array}\Bigg\downarrow$$

The answer is $(\ 0.1011)_2$

- Binary to Octal conversions: each three bits from right to left represent a number.

<u>EX</u>:

- $(111010011)_2 \longrightarrow (723)_8$
- $(\ 110100001)_2 \longrightarrow (641)_8$

- Octal to binary conversions: each number is representing in binary using three bits.

- $(\ 752\ )_8 \longrightarrow (\ 111101010\ )_2$
- $(\ 631\ )_8 \longrightarrow (\ 110011001\ )_2$

- Binary to Hexadecimal conversions: each four bits from right to left represent a number.

- $(\ 1101110001010001)_2 \longrightarrow (\ DC51)_{16}$

- Hexadecimal to Binary conversions: number is representing in binary using four bits.

<u>EX:</u>

- $(\ A35C\ )_{16} \longrightarrow (\ 1010001101011100\ )_2$

- Octal to Decimal conversions:

- $(\ 371\ )_8 \longrightarrow (\ ?\ )_{10}$

$1 \times 8^0 + 7 \times 8^1 + 3 \times 8^2 = 249$

The answer is $(249)_{10}$

- Decimal to Octal conversions:

- $(30.5)_{10} \longrightarrow (?)_8$

8	30	6	LSB	$0.5 \times 8 = 4.0$
8	3	3	MSB	

The answer is $(36.4)_8$

- Hexadecimal to decimal conversions:

EX:

- $(1BF)_{16} \longrightarrow (?)_{10}$

$F \times \_16^{0} + B \times 16^{1} + 1 \times 16^{3} = 447$

The answer is $(447)_{10}$

- Decimal to Hexadecimal conversions:

EX:

- $(28.3)_{10} \longrightarrow (?)_{16}$

16	28	12	LSB	$0.3 \times 16 = 4.8$	4
16	1	1	MSB	$0.8 \times 16 = 12.8$	12
				$0.8 \times 16 = 12.8$	12 ↓

The answer is $(1C.4CC)_{16}$

Digital Techniques

Lecture 3

1st Class

Digital Techniques

Complement:

Complement is used in digital computer to simplify the subtraction operation and for logical manipulation.

There are two types of complement for each base (r)

1. The r's complement.
2. The (r-1)'s complement.

The r's Complement:

$$r^n - N$$

where n : number of bit

N: number

EX: The 10's complement for $(52520)_{10}$ is:

$$10^5 - 52520 = 47480$$

EX: The 10's complement of $(0.3267)_{10}$ is:

$$1 - 0.3267 = 0.6733$$

EX: The 2's complement of $(101100)_2$ is:

$$2^6 - 101100 = 1000000 - 101100 = 010100$$

The (r – 1) Complement:

The equation for (r–1) is:

$r^n - r^{-m} - N$

EX: The 9's complement of $(25.639)_{10}$ is:

$10^2 - 10^{-3} - 25.639 = 74.360$

EX: The 1's complement of $(101100)_2$ is :

$2^6 - 2^0 - 101100 = 1000000 - 1 - 101100 = 10011$

Subtraction Using R complement:

Take the r complement to the subtracted N and then add it to M, if an end carry occur, discard it. If an end carry does not occur, take r complement of the number obtained in step 1.

EX: Use 10 complement to subtract:

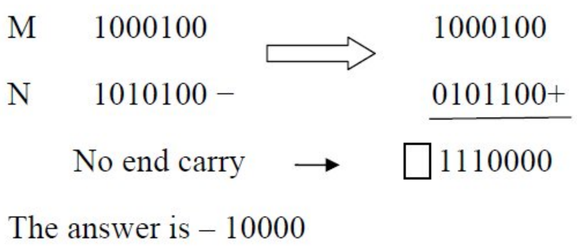
M 72532 72532

N 96750 - $\underline{03250 +}$

End Carry $\underline{\ 0\ }$ 75782

The answer is - 24218

EX: Use 2's complements subtract:



M 1000100 ⇒ 1000100

N 1010100 − $\underline{0101100+}$

No end carry → □1110000

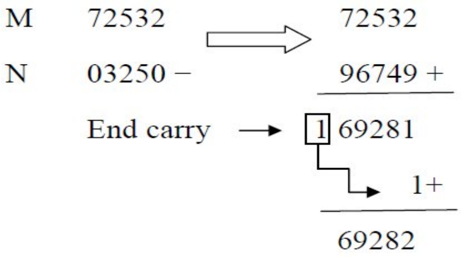
The answer is – 10000

Subtraction Using r – 1 Complement:

Add M to (r – 1) of N (subtracted) and check the carry:

If an end carry occur, add 1 to the least significant digit. And if an end carry does not occur, take the (r – 1)'s of the number obtained in step 1 and place a negative sign in front.

EX: subtract using 9's complement:



M 72532 ⇒ 72532

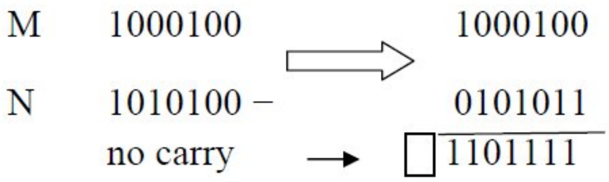
N 03250 − $\underline{96749 +}$

End carry → $\boxed{1}$ 69281

1+

69282

EX: Subtract using 1's complement:



$$\begin{array}{llcl} M & 1000100 & & 1000100 \\ & & \Rightarrow & \\ N & 1010100\ - & & 0101011 \\ & \text{no carry} & \rightarrow & \boxed{}\,\overline{1101111} \end{array}$$

Take 1's complement to the result and the answer is - 0010000.

Digital Techniques

Lecture 4

1st Class

Digital Techniques

Codes:

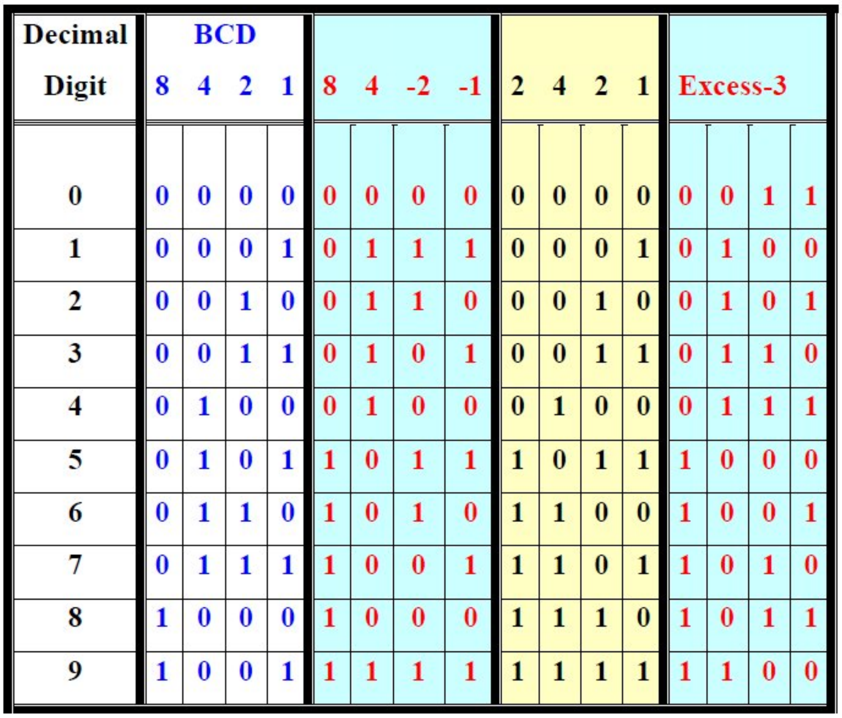
Three types of code will be considered:

1. BCD (Binary code decimal).
2. EX – 3 codes.
3. Gray code.

Binary Codes for Decimal Digits

- Internally, digital computers operate on binary numbers.
- When interfacing to humans, digital processors, e.g. pocket calculators, communication is decimal-based.
- Input is done in decimal then converted to binary for internal processing.
- For output, the result has to be converted from its internal binary representation to a decimal form.
- To be handled by digital processors, the decimal input (output) must be coded in binary in a digit by digit manner.
- For example, to input the decimal number **957**, each *digit* of the number is individually *coded* and the number is stored as **1001_0101_0111.**
- Thus, we need a specific code for each of the 10 decimal digits. There is a variety of such decimal binary codes.
- The shown table gives several common such codes.
- One commonly used code is the *Binary Coded Decimal* (**BCD**) code which corresponds to the first 10 binary representations of the decimal digits 0-9.
- The BCD code requires 4 bits to represent the 10 decimal digits.
- Since 4 bits may have up to 16 different binary combinations, a total of 6 combinations will be unused.
- The position weights of the BCD code are 8, 4, 2, 1.

- ➢ Other codes (shown in the table) use position weights of 8, 4, -2, -1 and 2, 4, 2, 1.
- ➢ An example of a non-weighted code is the *excess-3 code* where digit codes are obtained from their binary equivalent after adding 3. Thus the code of a decimal 0 is 0011, that of 6 is 1001, etc.



Decimal	BCD												Excess-3			
Digit	8	4	2	1	8	4	-2	-1	2	4	2	1				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	0	1	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	1	1	1	0	1	1	1	0	0	0
6	0	1	1	0	1	0	1	0	1	1	0	0	1	0	0	1
7	0	1	1	1	1	0	0	1	1	1	0	1	1	0	1	0
8	1	0	0	0	1	0	0	0	1	1	1	0	1	0	1	1
9	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0

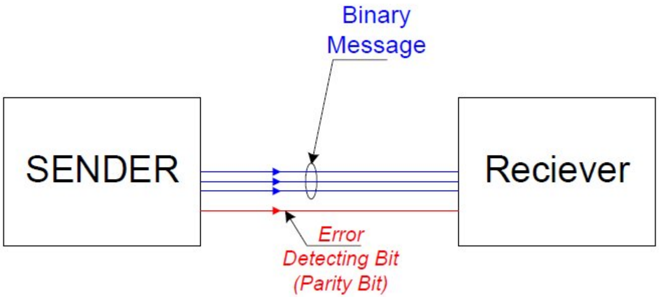
Example Converting **$(13)_{10}$** into binary, we get **1101**, coding the same number into BCD, we obtain **00010011**.

Exercise: Convert **$(95)_{10}$** into its binary equivalent value and give its BCD code as well.

Answer {**(1011111)2**, and **10010101**}

Error-Detection Codes

- Binary information may be transmitted through some communication medium, e.g. using wires or wireless media.
- A corrupted bit will have its value changed from 0 to 1 or vice versa.
- To be able to detect errors at the receiver end, the sender sends an extra bit (parity bit) with the original binary message.



- A *parity bit* is an extra bit included with the *n-bit binary message* to make the total number of 1's in this message (*including the parity bit*) either odd or even.
- If the parity bit makes the total number of 1's an *odd* (*even*) number, it is called *odd* (*even*) parity.
- The table shows the required *odd* (*even*) parity for a 3-bit message.

Three-Bit Message			**Odd Parity Bit**	**Even Parity Bit**
X	**Y**	**Z**	**P**	**P**
0	0	0	1	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	0	1

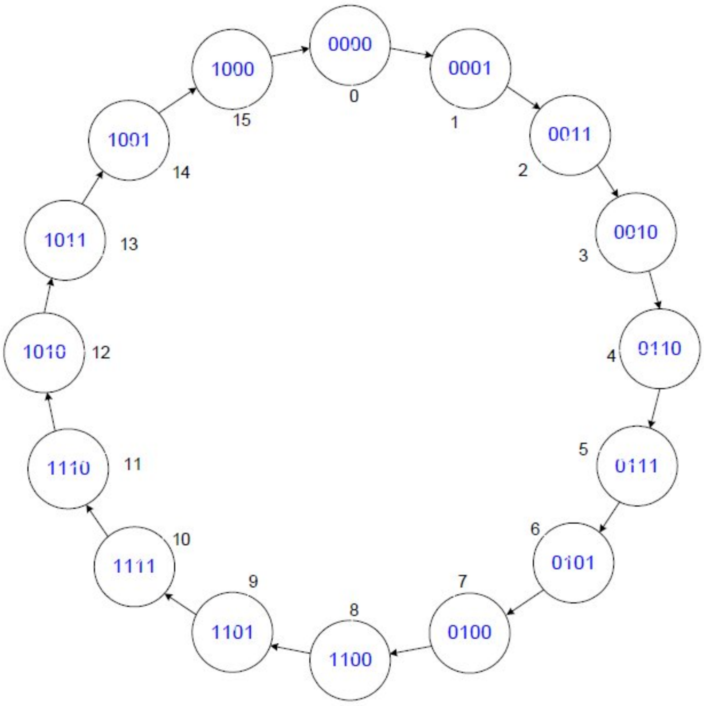
- At the receiver end, an error is detected if the message does not match have the proper parity (odd/even).
- Parity bits can detect the occurrence 1, 3, 5 or any odd number of errors in the transmitted message.
- No error is detectable if the transmitted message has 2 bits in error since the total number of
- 1's will remain even (or odd) as in the original message.
- In general, a transmitted message with even number of errors cannot be detected by the parity bit.

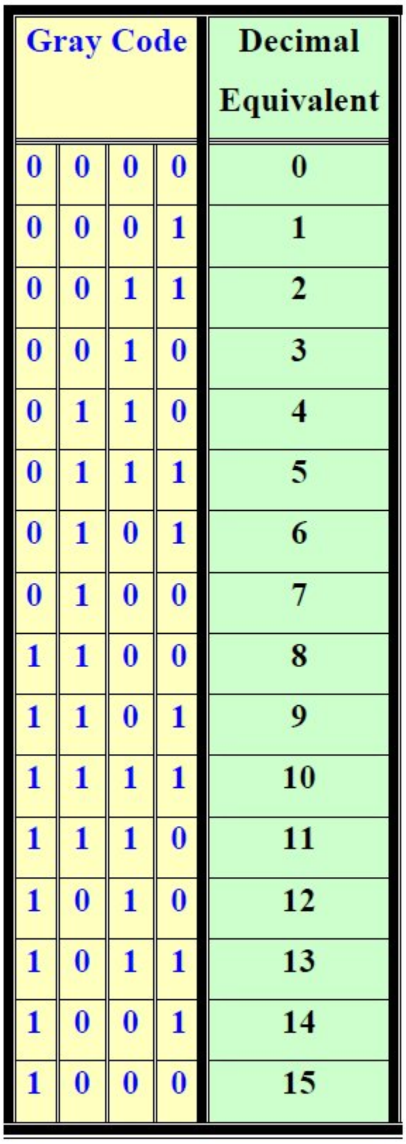
Error-Detection Codes

- Binary information may be transmitted through some communication medium, e.g. using wires or wireless media.
- Noise in the transmission medium may cause the transmitted binary message to be corrupted by changing a bit from 0 to 1 or vice versa.
- To be able to detect errors at the receiver end, the sender sends an extra bit (parity bit).

Gray Code

- The Gray code consists of 16 4-bit code words to represent the decimal Numbers 0 to 15.
- For Gray code, successive code words differ by only one bit from one to the next as shown in the table and further illustrated in the Figure.





Gray Code				Decimal Equivalent
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	0	1	0	3
0	1	1	0	4
0	1	1	1	5
0	1	0	1	6
0	1	0	0	7
1	1	0	0	8
1	1	0	1	9
1	1	1	1	10
1	1	1	0	11
1	0	1	0	12
1	0	1	1	13
1	0	0	1	14
1	0	0	0	15

Binary Number to Gray Code Conversion:

The conversion from binary to gray is shown in the examples below:

EX: Convert from Binary to Gray code

110101110	Binary
101111001	Gray

EX: Convert from Binary to Gray code

1100	Binary
1010	Gray

Gray Code to Binary Number Conversion:

EX: Convert from Gray code to Binary

101110101	Gray
110100110	Binary

EX: Convert from Gray code to Binary

1110010001	Gray
10111000010	Binary

Digital Techniques

Lecture 5

1st Class

Digital Techniques

Logic Gate

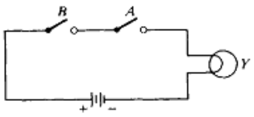
5.1 INTRODUCTION

The logic gate is the basic building block in digital systems. Logic gates operate with binary numbers. Gates are therefore referred to as binary logic gates. All voltages used with logic gates will be either HIGH or LOW. In this lecture, a HIGH voltage will mean a binary 1. A LOW voltage will mean a binary 0. Remember that logic gates are electronic circuits. These circuits will respond only to HIGH voltages (called 1s) or LOW (ground) voltages (called 0s).

All digital systems are constructed by using only three basic logic gates. These basic gates are called the AND gate, the OR gate, and the NOT gate. This chapter deals with these very important basic logic gates, or functions.

5.2 THE AND GATE

The AND gate is called the “all or nothing” gate. The schematic in Fig. 5.la shows the idea of the AND gate. The lamp (Y) will light only when both input switches (A and B) are closed. All the possible combinations for switches A and B are shown in Fig. 5.lb. The table in this figure is called a truth table. The truth table shows that the output (Y) is enabled (lit) only when both inputs are closed.



(a) AND circuit using switches

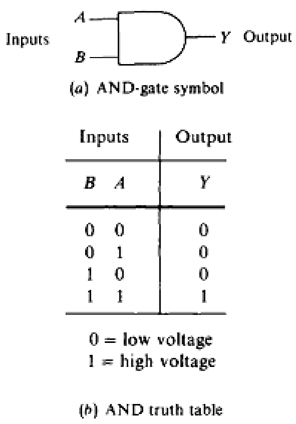
Input switches		Output light
B	A	Y
open	open	no
open	closed	no
closed	open	no
closed	closed	yes

(b) Truth table

Fig. 5.1

The standard logic symbol for the AND gate is drawn in Fig. 5.2a. This symbol shows the inputs as A and B. The output is shown as Y. This is the symbol for a 2-input AND gate. The truth table for the 2-input AND gate is shown in Fig. 5.2b. The inputs are shown as binary digits (bits). Note that only when both input A and input B are 1 will the output be 1. Binary 0 is defined as a LOW, or ground, voltage. Binary 1 is defined as a HIGH voltage. In this book, a HIGH voltage will mean about +5 volts (V) if the integrated circuits (ICs) being used are from the TTL family.

Boolean algebra is a form of symbolic logic that shows how logic gates operate. A Boolean expression is a “shorthand” method of showing what is happening in a logic circuit. The Boolean expression for the circuit in Fig. 5.2 is



(a) AND-gate symbol

Inputs		Output
B	A	Y
0	0	0
0	1	0
1	0	0
1	1	1

0 = low voltage
1 = high voltage

(b) AND truth table

Fig. 5.2

A-B=Y

The Boolean expression is read as A AND (. means AND) B equals the output Y. The dot (a) means the logic function AND in Boolean algebra, not multiply as in regular algebra, Sometimes the dot (-) is left out of the Boolean expression. The Boolean expression for the 2-input AND gate is then:

AB=Y

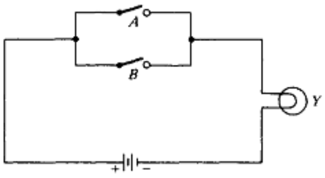
The Boolean expression reads A AND B equals the output Y.

The laws of Boolean algebra govern how AND gates operate. The formal laws for the AND function are:

$$A \cdot 0 = 0$$
$$A \cdot 1 = A$$
$$A \cdot A = A$$
$$A \cdot \overline{A} = 0$$

5.3 THE OR GATE

The OR gate is called the “any or all” gate. The schematic in Fig. 5.3a shows the idea of the OR gate. The lamp (Y) will glow when either switch A or switch B is closed. The lamp will also glow when both switches A and B are closed. The lamp (Y) will no2 glow when both switches (A and B) are open. All the possible switch combinations are shown in Fig. 5.3b. The truth table details the OR function of the switch and lamp circuit. The output of the OR circuit will be enabled (lamp lit) when any or all input switches are closed



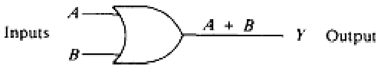
(a) OR circuit using switches

Input switches		Output light
B	A	Y
open	open	no
open	closed	yes
closed	open	yes
closed	closed	yes

(b) Truth table

Fig. 5.3

The standard logic symbol for an OR gate is drawn in Fig. 5.4a. Note the different shape of the OR gate. The OR gate has two inputs labeled A and B. The output is labeled Y. The shorthand Boolean expression for this OR function is given as A + B = Y. Note that the plus (+) symbol means OR in Boolean algebra. The expression (A + B = Y) is read as A OR (+ means OR) B equals output Y. You will note that the plus sign does not mean to add as it does in regular algebra.



(a) OR-gate symbol

Inputs		Output
B	A	Y
0	0	0
0	1	1
1	0	1
1	1	1

0 = low voltage
1 = high voltage

(b) OR truth table

Fig 5.4

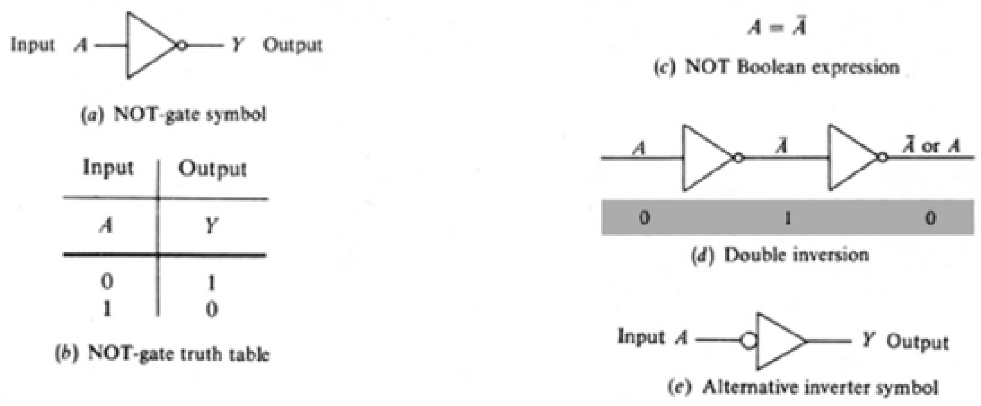
The truth table for the 2-input OR gate is drawn in Fig. 5.4b. The input variables (A and B) are given on the left. The resulting output (Y) is shown in the right column of the table. The OR gate is enabled (output is 1) anytime a 1 appears at any or all of the inputs. As before, a 0 is defined as a LOW (ground) voltage. A 1 in the truth table represents a HIGH (+5 V) voltage.

The laws of Boolean algebra govern how an OR gate will operate. The formal laws for the OR function are:

$$A + 0 = A$$
$$A + 1 = 1$$
$$A + A = A$$
$$A + \overline{A} = 1$$

5.4 THE NOT GATE

A NOT gate is also called an inverter. A NOT gate, or inverter, is an unusual gate. The NOT gate has only one input and one output. Figure 5.5a illustrates the logic symbol for the inverter, or NOT gate.



(a) NOT-gate symbol

Input	Output
A	Y
0	1
1	0

(b) NOT-gate truth table

(c) NOT Boolean expression

(d) Double inversion

(e) Alternative inverter symbol

Fig. 5.5

The process of inverting is simple. Figure 5.5b is the truth table for the NOT gate. The input is always changed to its opposite. If the input is 0, the NOT gate will give its complement, or opposite, which is 1. If the input to the NOT gate is a 1, the circuit will complement it to give a 0. This inverting is also called complementing or negating. The terms negating, complementing, and inverting all mean the same thing.

The Boolean expression for inverting is shown in Fig. 5.5e. The expression A =Ā reads as A equals the output not A. The bar over the A means to complement A. Figure 5.5d illustrates what would happen if two inverters were used. The Boolean expressions are written above the lines between the inverters. The input A is inverted to Ā (not A). The Ā is then inverted again to form $\overline{\overline{A}}$ (not not A). The double inverted A (2) is equal to the original A, as shown in Fig. 5.5d. In the shaded section below the inverters, a 0 bit is the input. The 0 bit is complemented to a 1. The 1 bit is complemented again back to a 0. After a digital signal goes through two inverters, it is restored to its original form.

An alternative logic symbol for the NOT gate or inverter is shown in Fig. 5.5e. The invert bubble may be on either the input or the output side of the triangular symbol. When the invert bubble appears on the input side of the NOT symbol (as in Fig. 5.5e), the designer is usually trying to suggest that this is an active LOW input. An active LOW input requires a LOW to activate some function in the logic circuit. The alternative NOT gate symbol is commonly used in manufacturer’s logic diagrams.

The laws of Boolean algebra govern the action of the inverter, or NOT gate. The formal Boolean algebra laws for the NOT gate are as follows:

$$\bar{0} = 1 \qquad \bar{1} = 0$$

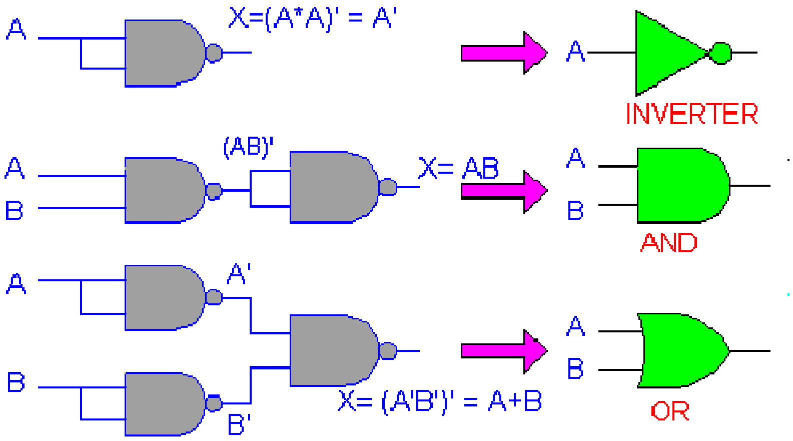
$$\text{If } A = 1\text{, then } \bar{A} = 0$$

$$\text{If } A = 0\text{, then } \bar{A} = 1$$

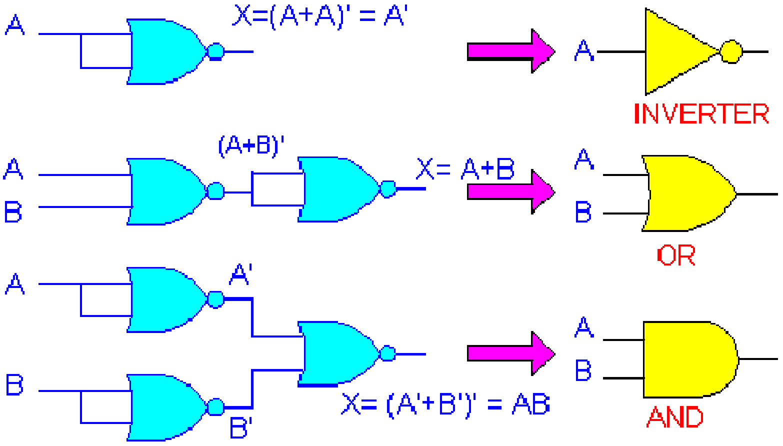
$$\bar{\bar{A}} = A$$

Universality of NAND & NOR Gates

It is possible to implement *any* logic expression using only NAND gates and no other type of gate. This is because NAND gates, in the proper combination, can be used to perform each of the Boolean operations OR, AND, and INVERT.



In a similar manner, it can be shown that NOR gates can be arranged to implement any of the Boolean operations.



Digital Techniques

Lecture 6

1st Class

Digital Techniques

USING PRACTICAL LOGIC GATES

Logic functions can be implemented in several ways. In the past, vacuum-tube and relay circuits performed logic functions. Presently tiny integrated circuits (ICs) perform as logic gates. These ICs contain the equivalent of miniature resistors, diodes, and transistors.

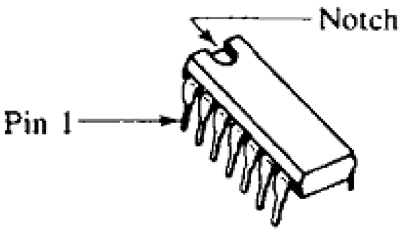


Figure 6.1: 14-pin DIP integrated circuit

A popular type of IC is illustrated in Fig. 6.1. This case style is referred to as a dual-in-line package (DIP) by IC manufacturers, This particular IC is called a 14-pin DIP integrated circuit. Note that immediately counterclockwise from the notch on the IC shown in Fig. 6.1 is pin number 1. The pins are numbered counterclockwise from 1 to 14 when viewed from the top of the IC. Manufacturers of ICs provide pin diagrams similar to the one in Fig. 6.2 for a 7408 IC. Note that this IC contains four 2-input AND gates; thus it is called a quadruple 2-input AND gate. Figure 6.2 shows the IC pins numbered from 1 through 14 in a counterclockwise direction from the notch. The power connections to the IC are the GND (pin 7) and Vcc (pin 14) pins. All other pins are the inputs and outputs to the four AND gates. The 7408 IC is part of a family of logic devices. It is one of many

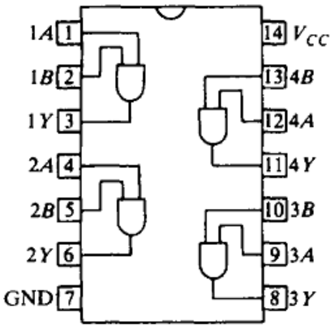
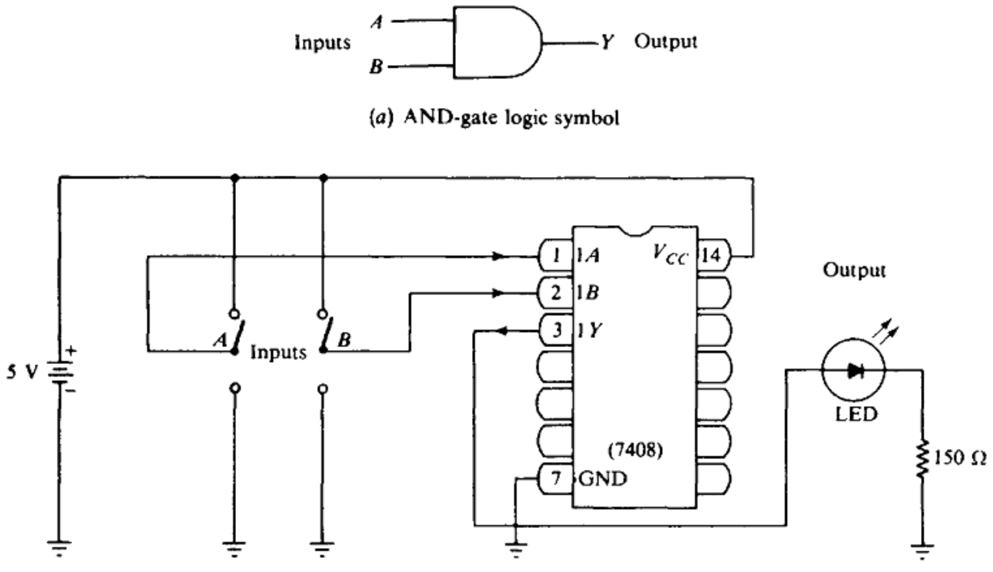


Fig. 6.2: Pin diagram for a 7408 IC



(a) AND-gate logic symbol

(b) Wiring an AND gate using a 7408 IC

Fig. 6.3

devices in the transistor-transistor logic (TTL) family of logic circuits. TTL devices are currently among the most popular logic devices.

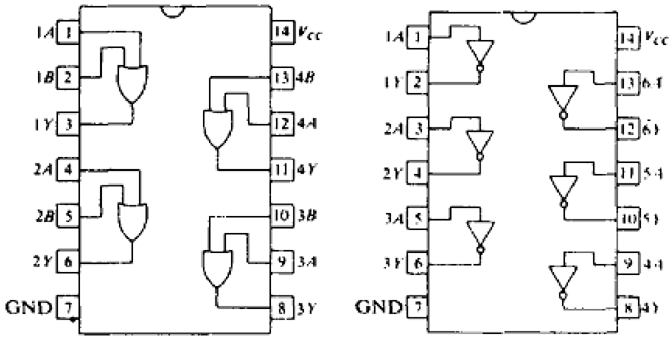
Given the logic diagram in Fig. 6.3a, wire a circuit using a 7408 IC. A wiring diagram for the circuit is shown in Fig. 6.3b. A 5-V power supply is used with all TTL devices. The positive (V',) and negative (GND) power connections are made to pins 14 and 7. Input switches (A and B) are wired to pins 1 and 2 of the 7408 IC. Note that, if a switch is in the up position, a logical 3 (+ 5 V) is applied to the input of the AND gate. At the right, a light-emitting diode (LED) and 150-ohm (a) limiting resistor are connected to ground. If the output at pin 3 is HIG'H (+5V), current will flow through the LED. Lighting the LED indicates a HIGH, or a binary 1, at the output of the AND gate.

The truth table in Fig. 6.4 shows the results of operating the 2-input AND circuit. The LED in Fig. 6.3b lights only when both input switches (A and B) are at + 5 V.

Inputs		Output	
A	*B*		
Voltage	Voltage	Voltage	LED lights?
GND	GND	GND	no
GND	+5 V	GND	no
+5 V	GND	GND	no
+5 V	+5 V	near +5 V	yes

Fig. 6.4 Truth table for a TTL-type AND gate

Manufacturers of integrated circuits also produce other logic functions. Figure 6.5 illustrates pin diagrams for two basic TTL ICs. Figure 6.5a is the pin diagram for a quadruple 2-input OR gate. In other words, the 7432 IC contains four 2-input OR gates. It could be wired and tested in a manner similar to the testing of the AND gate shown in Fig. 6.3b.

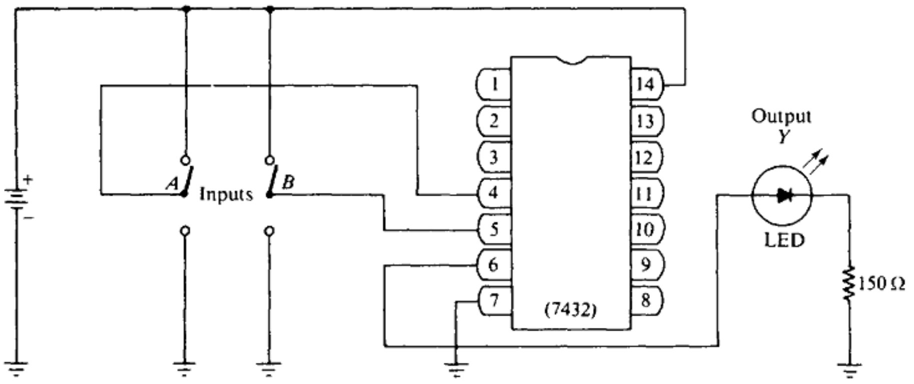


(a) Pin diagram for a 7432 IC (b) Pin diagram for a 7404 IC

Fig. 6.5

The 7404 IC shown in Fig. 6.5b is also a TTL device. The 7404 IC contains six NOT gates, or inverters. The 7404 is described by the manufacturer as a hex inverter IC. Note that each IC has its power connections (V_{CC} and GND). A 5-V dc power supply is always used with TTL logic circuits.

Q/ what logic function is performed by the circuit illustrated in the following Figure?



Digital Techniques

Lecture 7

1st Class

Digital Techniques

Boolean Algebra

Boolean algebra is a form of symbolic logic that shows how logic gates operate. A Boolean expression is a "shorthand" method of showing what is happening in a logic circuit.

Rules of Boolean algebra:

The following propositions are correct in and basic to Boolean algebra:

$A+1 = 1$	$A+\bar{A} = 1$	$A+0 = A$	$A+A = A$
$A . 0 = 0$	$A . \bar{A} = 0$	$A . A = A$	$A . 1 = A$
$\bar{\bar{A}} = A$	$A+AB=A$	$A+\bar{A}B=A$	$(A+B) . (A+C) = A + BC$

Laws of Boolean algebra:

- The Associative law:

  $(a + b) + c = a + (b + c)$

  $(a.b) . c = a. (b.c)$

- The Commutative law:

  $a + b = b + a$

  $a . b = b . a$

- The Distributive law:

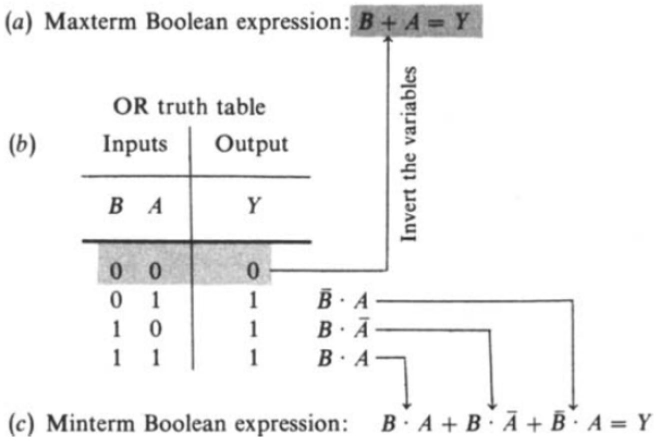
  $a (b + c) = ab + ac$

  $a + bc = (a + b)(a + c)$

SUM-OF-PRODUCTS BOOLEAN EXPRESSIONS

Consider the OR truth table in Fig. *7-1b.* The Boolean expression for this truth table can be written in two forms, as was observed in the introductory section. The minterm Boolean expression is developed from the output 1s in the truth table. Each 1 in the output column becomes a term to be ORed in the minterm expression. The minterm expression for this truth table is given in Fig. *7-1c* as

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



(*a*) Maxterm Boolean expression: $B + A = Y$

OR truth table

(*b*)

Inputs		Output	
B	*A*	*Y*	
0	0	0	
0	1	1	$\bar{B} \cdot A$
1	0	1	$B \cdot \bar{A}$
1	1	1	$B \cdot A$

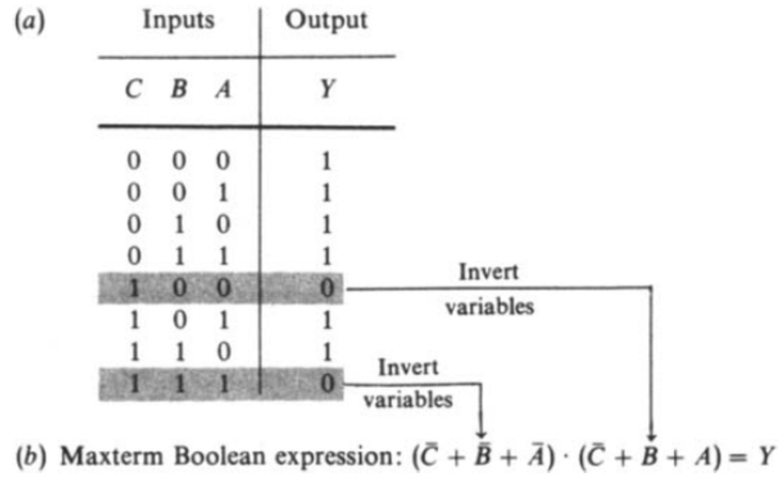
(*c*) Minterm Boolean expression: $B \cdot A + B \cdot \bar{A} + \bar{B} \cdot A = Y$

Fig. 7.1 Developing **a** minterm expression

PRODUCT-OF-SUMS BOOLEAN EXPRESSIONS

The truth table in Fig. *7-1* can also be described by using a maxterrn form of Boolean expression. This type of expression is developed from the 0s in the output column of the truth table. For each 0 in the output column, an ORed term is developed. Note that the input variables are inverted and then ORed. The maxterm Boolean expression for this truth table is given in Fig. *7-1*a. The maxterm expression for the OR truth table is shown as B + A = Y. This means the same thing as the familiar OR expression A + B = Y. For the truth table in Fig. *7-1,* the maxterm Boolean expression turns out to be the simplest. Both the minterm and maxterm expressions accurately describe the logic of the truth table in Fig. *7-1.*

Consider the truth table in Fig. *7-2*a. The minterm expression for this truth table would be rather long. The maxtenn Boolean expression is developed from the variables in lines 5 and 8. Each of these


(*a*)

Inputs			Output
C	*B*	*A*	*Y*
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Invert variables
Invert variables

(*b*) **Maxterm Boolean expression:** $(\bar{C}+\bar{B}+\bar{A})\cdot(\bar{C}+B+A)=Y$

Fig. 7-2 Developing **a** maxterm expression

lines has a 0 in the output column. The variables are inverted and ORed with parentheses around them. The terms are then ANDed. The complete maxterm Boolean expression is given in Fig. 7-2b. The maxterm expression is also called the product-of-sums form of a Boolean expression. The product-of-sums term comes from the arrangement of the sum (+) and product (**.**) symbols. A maxterm Boolean expression would be implemented by using an OR-AND pattern of logic gates illustrated in Fig. 7-3. Note that the outputs of the two OR gates are feeding into an AND gate. The maxterm expression (C + B' + A') . (C'+ B + A)= Y is implemented by using the OR-AND pattern of gates in Fig. 7-3.

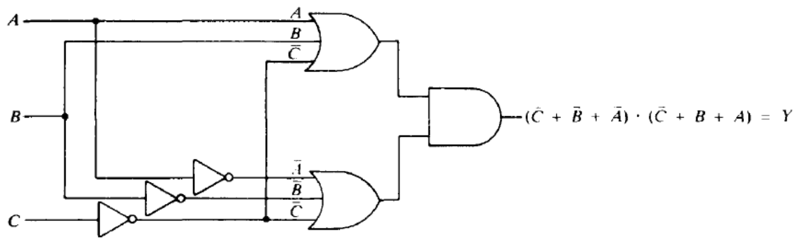


Fig. 7-3 Maxterm expression implemented with OR-AND circuit

Minimization of Combinational Circuits

When constructing digital circuits, in addition to obtaining a functionally correct circuit, we like to optimize them in terms of circuit size, speed, and power consumption. In this section, we will focus on the reduction of circuit size. Usually, by reducing the circuit size, we will also improve on speed and power consumption.

We saw how we can transform a Boolean function to another equivalent function by using the Boolean algebra theorems. If the resulting function is simpler than the original, then we want to implement the circuit based on the simpler function, since that will give us a smaller circuit size.

Using Boolean algebra to transform a function to one that is simpler is not an easy task, especially for the computer. There is no formula that says which is the next theorem to use. Luckily, there are easier methods for reducing Boolean functions. The ***Karnaugh map*** method is an easy way for reducing an equation manually and is discussed in the following section.

The ***Karnaugh map*** method.

This graphic method is based on **Boolean** theorems. It is only one of several methods used by logic designers to simplify logic circuits. Karnaugh maps are sometimes referred **to** as ***K- maps.***

- ❖ The first step in the Karnaugh mapping procedure is to develop a minterm Boolean expression from a truth table. Consider the familiar truth table in Fig. 7-4a. Each 1 in the Y column of the truth table produces two variables ANDed together. These ANDed groups are then ORed to form a sum-of-products (minterm) type of Boolean expression (Fig.7-4b). This expression will be referred to as the *unsimplified* Boolean expression.

- ❖ The second step in the mapping procedure is to plot 1s in the Karnaugh map in Fig. 7-4c. Each ANDed set of variables from the minterm expression is placed in

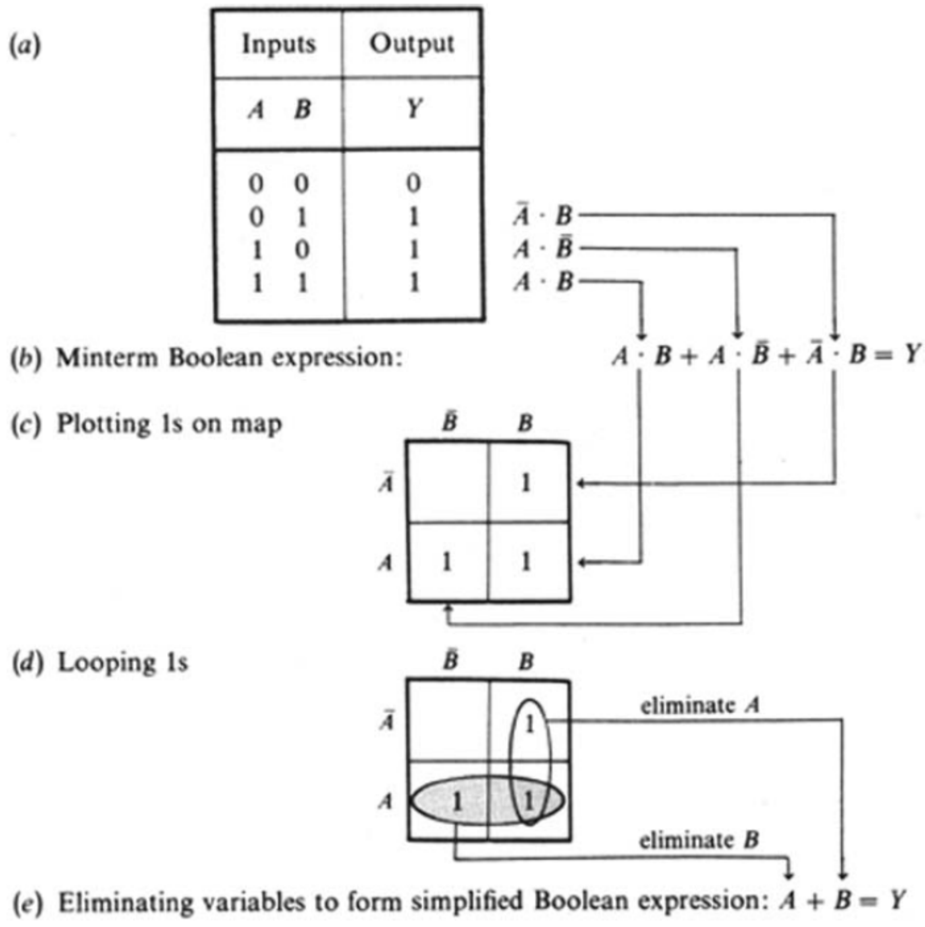


Fig. 7- 4 Using a map

the appropriate square of the map. The map is just a very special output column of the truth table.

❖ The third step is to loop adjacent groups of two, four, or eight 1s together. Figure 7-4d shows two loops drawn on the map. Each loop contains two 1s.

❖ The fourth step is to eliminate variables. Consider first the shaded loop in Fig. 7-4d.

❖ Note that a B and a B’ (not B) are contained within the shaded loop. When a variable and its complement are within a loop, that variable is eliminated. From the shaded loop, thc B and B’ terms arc eliminated, leaving the A variable (Fig. 7-4e).Next consider the unshaded loop in Fig. 7-4d. It contains an A and a A’ (not A). The A and A’ tcrms are eliminated, leaving only the B variable (Fig. 7-4e).

❖ The fifth step is to OR the remaining variables. The final simplified Boolean expression is A + B = Y (Fig. 7-4e). The simplified expression is that of a 2-input OR gate.

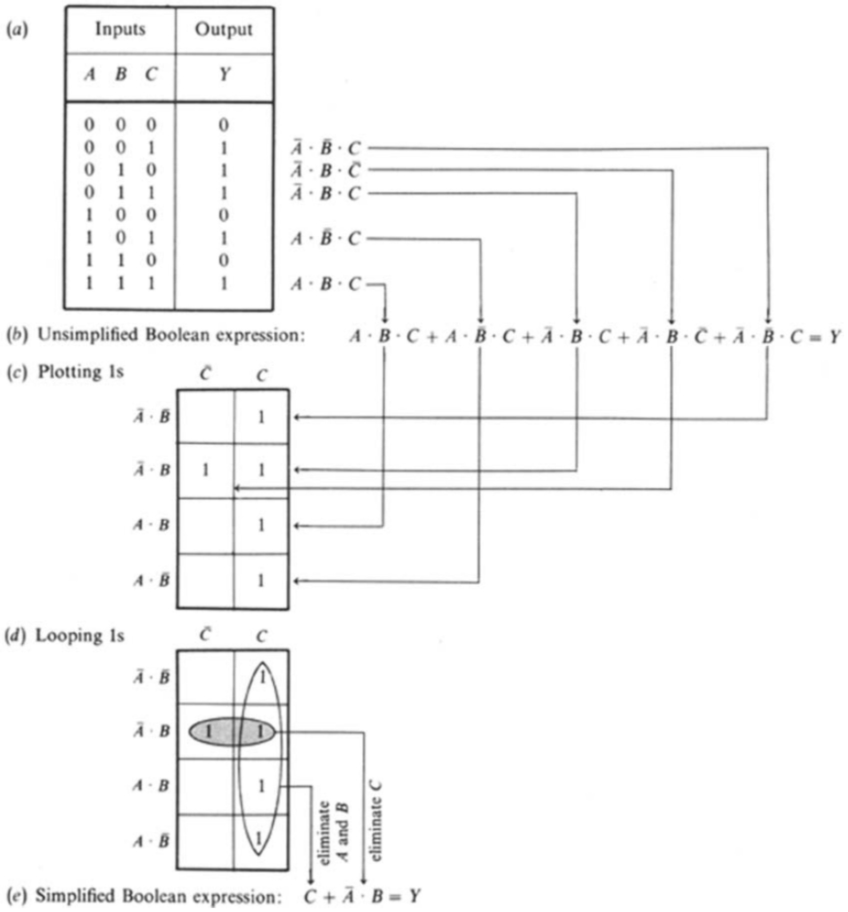

(a)
Inputs
Output
A B C
Y
0 0 0 0
0 0 1 1
0 1 0 1
0 1 1 1
1 0 0 0
1 0 1 1
1 1 0 0
1 1 1 1
$\bar{A} \cdot \bar{B} \cdot C$
$\bar{A} \cdot B \cdot \bar{C}$
$\bar{A} \cdot B \cdot C$
$A \cdot \bar{B} \cdot C$
$A \cdot B \cdot C$
(b) Unsimplified Boolean expression: $A \cdot B \cdot C + A \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot C + \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C = Y$
(c) Plotting 1s
$\bar{C}$
C
$\bar{A} \cdot \bar{B}$
$\bar{A} \cdot B$
$A \cdot B$
$A \cdot \bar{B}$
1
1 1
1
1
(d) Looping 1s
$\bar{C}$
C
$\bar{A} \cdot \bar{B}$
$\bar{A} \cdot B$
$A \cdot B$
$A \cdot \bar{B}$
eliminate A and B
eliminate C
(e) Simplified Boolean expression: $C + \bar{A} \cdot B = Y$

Fig.7-5 using a three-variable map

In summary, the steps in simplifying a logic expression using a Karnaugh map are as follows:

1. Write a minterm Boolean expression from the truth table.
2. Plot a 1 on the map for each ANDed group of variables. (The number of 1s in the Y column of the truth table will equal the number of Is on the map.)
3. Draw loops around adjacent groups of two, four, or eight 1s on the map. (The loops may overlap.)
4. Eliminate the variable(s) that appear(s) with its (their) complement(s) within a loop, and save the variable(s) that is (are) left.
5. Logically OR the groups that remain to form the simplified minterm expression.

KARNAUGH MAPS WITH THREE VARIABLES

Consider the truth table in Fig. 7-*5a.* The *first step* in using the Karnaugh map is to write the minterm Boolean expression for the truth table. Figure *7-5b* illustrates the *unsimplified* minterm expression for the truth table. The *second step* is plotting 1s on the map. Five 1s are plotted on the map in Fig. *7-5c.* Each 1 corresponds to an ANDed group of variables (such as ***A*** . *B* . *C).* The *third step* is to loop adjacent groups of 1s on the map. Loops are placed around groups of ***eight***, ***four***, or ***two*** 1s. Two loops are drawn on the map in Fig. 7-*5d.* The shaded loop contains two 1s. The larger loop contains four 1s. The *fourth step* is to eliminate variables. The shaded loop in Fig. 7-*5d* contains both the *C* and C' terms. The *C* variable can thus be eliminated, leaving the *A'* . *B* term. The large loop contains the ***A*** and *A'* as well as the *B* and *B'* terms. These can be eliminated, leaving only the *C* variable. The *fifth step* is to OR the remaining terms. The *C* and *A'*. *B* terms are ORed in Fig. *5-28e.* The final simplified Boolean expression is then *C* + *A'.B* = ***Y.***T his is much easier to implement with ICs than the *unsimplified* version of Fig. 7-5b. The simplified expression will generate the truth table in Fig. 7-5a.

KARNAUGH *MAPS* WITH FOUR VARIABLES

Consider the truth table with four variables in Fig. 7-6a. The ***first step*** in simplification by using a Karnaugh map is to write the minterm Boolean expression. The lengthy unsimplified minterm expression appears in Fig. 7-6b. ***An*** ANDed group for four variables is written for each 1 in the ***Y*** column of the truth table. The ***second step*** is to plot 1s on the Karnaugh map. Nine 1s are plotted on the map in Fig. 7-6c. Each 1 on the map represents an ANDed group of terms from the unsimplified expression. The ***third step*** is to loop adjacent groups of 1s. Adjacent groups of eight, four, or two 1s are looped. Larger loops provide more simplification. Two loops have been drawn in Fig. 7-6c. The larger loop contains eight 1s. The ***fourth step*** is to eliminate variables. The large loop in Fig. 7-6c eliminates the ***A*** **,** *B* , and ***C*** variables. This leaves the *D* term. The small loop contains two 1s and

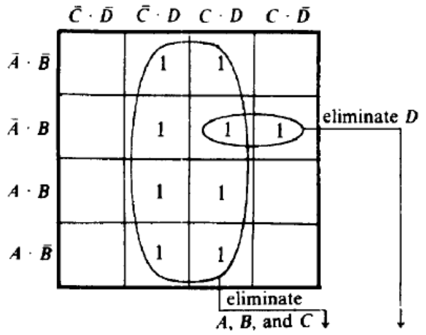
(*a*)

Inputs				Output
A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

(*b*) Unsimplified minterm expression

$$\bar{A}\cdot\bar{B}\cdot\bar{C}\cdot D+\bar{A}\cdot\bar{B}\cdot C\cdot D+\bar{A}\cdot B\cdot\bar{C}\cdot D+\bar{A}\cdot B\cdot C\cdot\bar{D}$$
$$+\bar{A}\cdot B\cdot C\cdot D+A\cdot\bar{B}\cdot\bar{C}\cdot D+A\cdot\bar{B}\cdot C\cdot D$$
$$+A\cdot B\cdot\bar{C}\cdot D+A\cdot B\cdot C\cdot D=Y$$

(*c*) Plotting and looping 1s on map



(*d*) Simplified Boolean expression: $D+\bar{A}\cdot B\cdot C=Y$

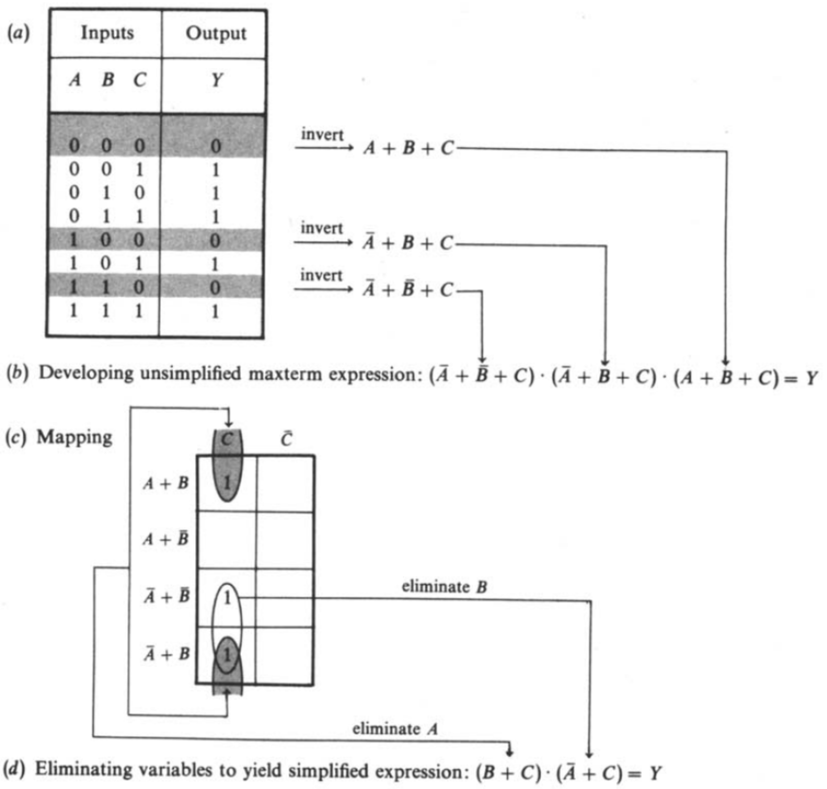
Fig. 7-6 Using a four-variable map

eliminates the D variable. That leaves the A’B C term. The fifth step is to logically OR the remaining terms. Figure 7-6d shows the remaining groups ORed to form the simplified minterm expression D + A’ B C = Y. The amount of simplification in this example is obvious when the two Boolean expressions in Fig. 7-6 are compared.

USING MAPS WITH MAXTERM EXPRESSIONS

A different form of the Karnaugh map is used with maxterm Boolean expressions. The steps for

1. Write a maxterm Boolean expression from the truth table. (Note the inverted form in Fig.7-7a.)
2. Plot a 1 on the map for each ORed group of variables. The number of OS in the Y column of the truth table will equal the number of 1s on the map.
3. Draw loops around adjacent groups of two, four, or eight 1s on the map.
4. Eliminate the variable(s) that appear(s) with its (their) complement(s) within a loop, and save the variable(s) that is (are) left.
5. Logically AND the groups that remain to form the simplified maxterm expression.



(a)

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(b) Developing unsimplified maxterm expression: $(\bar{A}+\bar{B}+C)\cdot(\bar{A}+B+C)\cdot(A+B+C)=Y$

(d) Eliminating variables to yield simplified expression: $(B+C)\cdot(\bar{A}+C)=Y$

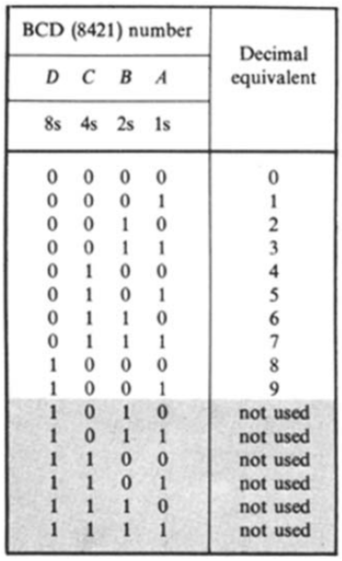
Fig. 7-7 Mapping with Maxterm expressions.

The maxterm mapping procedure and Karnaugh map are different from those used for minterm expressions. Both techniques should be tried on a truth table to find the less costly logic circuit.

DON’T CARES ON KARNAUGH MAPS

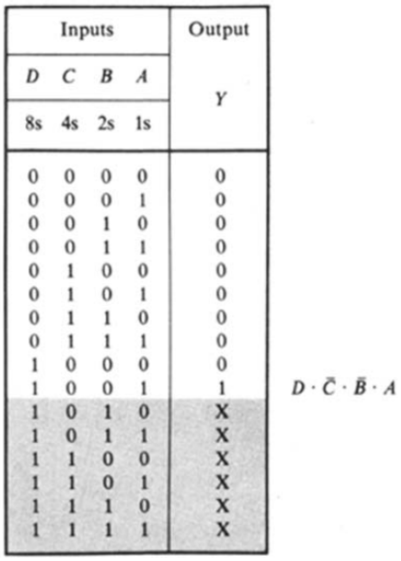
Consider the table for BCD (8421) numbers given in Fig.7-8. Note that the binary numbers 0000 to 1001 on the table are used to specify decimal numbers from 0 to 9. For convenience, the table is completed in the shaded section, which shows other possible combinations of the variables D, C, B, and A. These six combinations (1010, 1011, 1100, 1101, 1110, and 1111) are not used by the BCD code. These combinations are called don’t cares when plotted on a Karnaugh map. The don’t cares may have some effect on simplifying any logic diagram that might be constructed.

Suppose a problem specifying that a warning light would come ON when the BCD count reached 1001 (decimal 9); see the truth table in Fig.7-9. See 1 is placed in the output column (Y) of the truth table after the input 1001. The Boolean expression for this table (above the shaded section) is D.C’.B’.A = Y. This is shown to the right of the table. The “not used” combinations in the shaded section of the truth table might have some effect on this problem. A Karnaugh map is drawn in Fig. 7-10b. The 1 for the D.C’.B’.A term is plotted on the map. The six *don’t cares* (X’s from the truth table) are plotted as X’s on the map. An X on the map means that square can be either a 1 or a 0. A loop is drawn around adjacent 1s. The X’s on the map can be considered 1s, so the single loop is drawn around the 1 and three X’s. Remember that only groups of two, four, or eight adjacent 1s and X’s are looped together. The loop contains four squares, which will eliminate two variables. The B and C variables are eliminated, leaving the simplified Boolean expression D.A = Y in Fig. 7-10c. As was said earlier, unused combinations from a truth table are called don’t cares. They are shown as X’s on a Karnaugh map. Including don’t cares (X’s) in loops on a map helps to further simplify Boolean expressions.



BCD (8421) number				Decimal equivalent
D	*C*	*B*	*A*	
8s	4s	2s	1s	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	not used
1	0	1	1	not used
1	1	0	0	not used
1	1	0	1	not used
1	1	1	0	not used
1	1	1	1	not used

Fig 7-8 Table of BCD numbers



Inputs				Output
D	*C*	*B*	*A*	*Y*
8s	4s	2s	1s	
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

$D \cdot \bar{C} \cdot \bar{B} \cdot A$

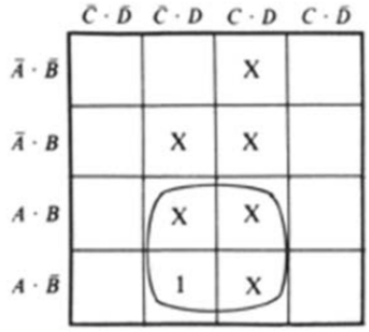
Fig. 7.9

$$D \cdot \bar{C} \cdot \bar{B} \cdot A = Y$$

(*a*) Unsimplified Boolean expression

$$D \cdot A = Y$$

(*c*) Simplified Boolean expression



	$\bar{C} \cdot \bar{D}$	$\bar{C} \cdot D$	$C \cdot D$	$C \cdot \bar{D}$
$\bar{A} \cdot \bar{B}$			X	
$\bar{A} \cdot B$		X	X	
$A \cdot B$		X	X	
$A \cdot \bar{B}$		1	X	

(*b*) Map

Fig. 7.10 Using a map

Subject: Logic Techniques

Tutorial Sheet No.1 1

Computer and System Eng. Dept.

First class / 2009 - 2008

1. Convert the following decimal numbers to binary numbers:

$(572)_{10} \longrightarrow (\qquad\qquad)_2$ $(72)_{10} \longrightarrow (\qquad\qquad)_2$

$(127)_{10} \longrightarrow (\qquad\qquad)_2$ $(255)_{10} \longrightarrow (\qquad\qquad)_2$

$(17.325)_{10} \longrightarrow (\qquad\qquad)_2$ $(9.152)_{10} \longrightarrow (\qquad\qquad)_2$

$(0.572)_{10} \longrightarrow (\qquad\qquad)_2$

2. Convert the following binary numbers to decimal numbers:

$(110101)_2 \longrightarrow (\qquad\qquad)_{10}$ $(100011)_2 \longrightarrow (\qquad\qquad)_{10}$

$(11101)_2 \longrightarrow (\qquad\qquad)_{10}$ $(0.1010)_2 \longrightarrow (\qquad\qquad)_{10}$

$(0.11011)_2 \longrightarrow (\qquad\qquad)_{10}$ $(1011.011)_2 \longrightarrow (\qquad\qquad)_{10}$

3. Convert the following Hexadecimal numbers to decimal numbers:

   $(15)_{16} \longrightarrow (\qquad\qquad)_{10}$ $(B9)_{16} \longrightarrow (\qquad\qquad)_{10}$

   $(AA4)_{16} \longrightarrow (\qquad\qquad)_{10}$ $(D2763) \longrightarrow (\qquad\qquad)_{10}$

4. Convert the following decimal numbers to hexadecimal numbers:

   $(25)_{10} \longrightarrow (\qquad\qquad)_{16}$ $(396)_{10} \longrightarrow (\qquad\qquad)_{16}$

   $(1342)_{10} \longrightarrow (\qquad\qquad)_{16}$

5. Convert the following hexadecimal numbers to binary numbers:

   $(CD)_{16} \longrightarrow (\qquad\qquad\qquad\qquad)_2$

   $(649)_{16} \longrightarrow (\qquad\qquad\qquad\qquad)_2$

$(A15)_{16} \longrightarrow (\qquad\qquad\qquad\qquad)_2$

6. Convert the following Binary numbers to hexadecimal numbers:

$(0110010110) \longrightarrow (\qquad\qquad\qquad\qquad)_{16}$

$(01101111010110)_2 \longrightarrow (\qquad\qquad\qquad\qquad)_{16}$

7. Convert the following decimal numbers to octal numbers:

$(19)_{10} \longrightarrow (\qquad\qquad\qquad)_8$

$(63)_{10} \longrightarrow (\qquad\qquad\qquad)_8$

$(103)_{10} \longrightarrow (\qquad\qquad\qquad)_8$

8. Convert the following octal numbers to decimal numbers:

$(124)_8 \longrightarrow (\qquad\qquad\qquad)_{10}$

$(14)_8 \longrightarrow (\qquad\qquad\qquad)_{10}$

$(371)_8 \longrightarrow (\qquad\qquad\qquad)_{10}$

$(15.5)_8 \longrightarrow (\qquad\qquad\qquad)_{10}$

9. Convert the following octal numbers to binary numbers:

$(61)_8 \longrightarrow (\qquad\qquad\qquad\qquad)_2$

$(53.3)_8 \longrightarrow (\qquad\qquad\qquad\qquad)_2$

$(34)_8 \longrightarrow (\qquad\qquad\qquad\qquad)_2$

$(24.25) \longrightarrow (\qquad\qquad\qquad\qquad)_2$

10. Convert the following Binary numbers to Octal Numbers:

$(101101)_2 \longrightarrow (\qquad\qquad\qquad\qquad)_8$

$(011010.110100)_2 \longrightarrow (\qquad\qquad\qquad\qquad)_8$

$(011.110)_2 \longrightarrow (\qquad\qquad\qquad\qquad)_8$

Control and Systems *Engineering Dep.*	*Class / 1^{st}*

Sheet No. (   )

Q1: The input to an automobile safety system are supplied by four ON-OFF switching. One switch is indicate if all the doors are closed or opens, each front seat belt has a switch indicating if it is fastened and a pressure switch shows if the front passenger seat is occupied. The circuit output is a logical only witch allow the driver to operate the starter motor when it is safe to do so (e.g. doors closed and seat belt fastened). Design combinational logic circuit to implement the safety system.

Q2: Design an industrial process requires that a section of plant is kept between two temperatures, if temperature exceeded some upper limit a refrigerator is switch on, and if temperature falls below lower limit a heat is switch on , two thermostatic switches A,B provide an indication of the temperature to a control unit.

Q3: It is required to design a combinational logic circuit that could be controlled to work in two modes:

Mode1: Binary adder (add 3 bit number A to 3 bit number B)

Mode2: 2's complement subtracter. (Subtracts a 3 bit number A from a 3 bit number B using 2's complement).

Q4: Four chairs A, B, C, and D are placed in a row. Each chair may be occupied ("1") or empty ("0"). A Boolean function F is ("1") if and only if there are two or more adjacent chairs that are empty.

a) Give the truth table defining the Boolean F.
b) Express F as a minterm expansion (standard sum of products).
c) Express F as a maxterm expansion (standard product of sums).
d) Simplify the minterm expansion of F.

Q5: A buzzer is to sound when the following conditions apply:

- Switches A, B, C are on.
- Switches A and B are on but switch C is off.
- Switches A and C are on but switch B is off.
- Switches B and C are on but switch A is off.

a) Draw a truth table for this situation and obtain a Boolean expression for it.
b) Minimize this expression and draw a logic diagram for it.

1st year class [San 2-1-2010] Tutorial sheet 3

Subject: Digital Techniques *Computer and Control Department*

Q1-

Circuit 1 in Figure 1-a is intended to receive 4 bits as input and produce 7 outputs to drive a 7 segment display circuit which is intended to display eight characters which are shown in Figure 1-b.

a- Complete the truth table of circuit 1.

b- Design the logic part of circuit 1 which produces the output a.

Truth table for circuit 1

Inputs				Outputs							
K	**L**	**M**	**N**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**Displayed character**
0	0	0	0								A
0	0	0	1	1	0	0	1	1	1	0	C
0	0	1	0	1	0	0	1	1	1	1	E
0	0	1	1	1	0	0	0	1	1	1	F
0	1	0	0								H
0	1	0	1	1	1	0	0	1	1	1	P
0	1	1	0	0	1	1	1	1	1	0	U
0	1	1	1	0	1	1	1	0	1	1	y
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								

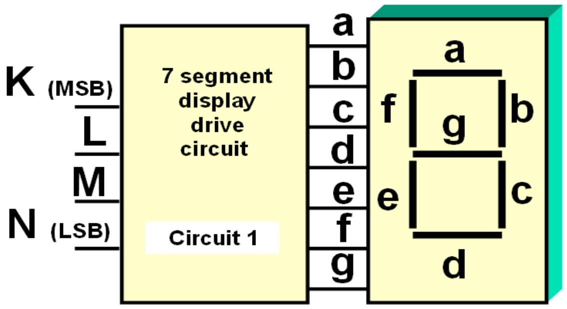
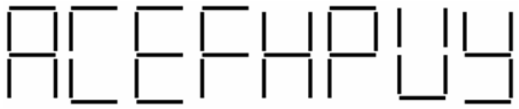


Figure 1-a



The characters to be displayed

Figure 1-b

Q2 A- Given the truth table below, write F as SOF and G as POS. *(6 marks)*
Then minimize F and G using Boolean algebra. *(4 marks)*

Inputs			Outputs	
A	**B**	**C**	**F**	**G**
0	0	0	1	1
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

F =

B- Write the T.T and Minimize the following logic function using Boolean algebra.

F(A,B,C,) = ∑ 0,2,4,6

G =

Q3-Complete the truth table for F where;   F(A,B,C) = AC

Inputs			Output
A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Q4- Given F(A,B,C,D) = $\sum$ 0,1,3,5,6,7,11,13:

i- Fill the Karnauph map below:

CD AB	00	01	11	10
00				
01				
11				
10				

(2 marks)

ii- Use the minterms, in the K-map you filled in a, to find F in its simplest possible form.

iii- Plot the logic circuit of F.

Q5- It is required to design a combinational logic circuit that could be controlled to work in two modes:

Mode 1: Binary adder. (Adds a 3 bit number A to a 3 bit number B)

Mode 2: 1's complement subtracter. (subtracts a 3 bit number B from a 3 bit number A using 1's complement)

Q6- It is required to design a combinational logic circuit that could be controlled to work in two modes:

Mode 1: Binary adder. (Adds a 3 bit number A to a 3 bit number B)

Mode 2: 2's complement subtracter. (subtracts a 3 bit number B from a 3 bit number A using 2's complement.

Q7- Design a logic circuit that converts a 3 bit gray code (A,B,C) into a 3 bit normal binary (D,E,F).

(1) Complete the truth table.

(2) Obtain the K-map for D, E, and F.

(3) Obtain the simplified Boolean expression for D, E, and F using K-Map

Q8- Complete the following T.T:

Binary sequence	**Gray code**
0000	
0001	

0010	
0011	
0100	
0101	
0110	
0111	
:	:

Q9- Plot the logic circuit for the following logic function using NOR gates Only:

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

Q10- A buzzer is to sound when the following conditions apply:

- Switches A, B, C are on.
- Switches A and B are on but switch C is off.
- Switches A and C are on but switch B is off.
- Switches B and C are on but switch A is off

Draw a truth table for this situation and obtain a Boolean expression for it. Minimize this expression and draw a logic diagram for it.