

Digital Logic design

Aims

Make students familiar with Electronic circuits, Digital system, Number system, Boolean algebra, logic design, and logic circuit through lecturing, home work, and lab experiments.

Make students gain some experience on modern computer-aided-design tools

Text Book

-Digital Design (4th Edition), by M. Morris Mano,
Michael D. Ciletti

-Digital Fundamentals (10 th Edition) Floyd

-Fundamentals of Logic Design (5th Edition), by
C.R.Roth

Topics Covered:

1. Introduction to digital concepts
2. Number systems, operations, and codes
3. Logic gates
4. Boolean algebra and logic simplification
5. Functions of combinational logic
6. Decoder, encoder, MUX, DMUX
7. Flip-Flops and related devices
8. Flip-Flops applications
9. Counters
10. Shift registers
11. Sequential logic

Chapter 1

Summary

Digital and Analog Quantities

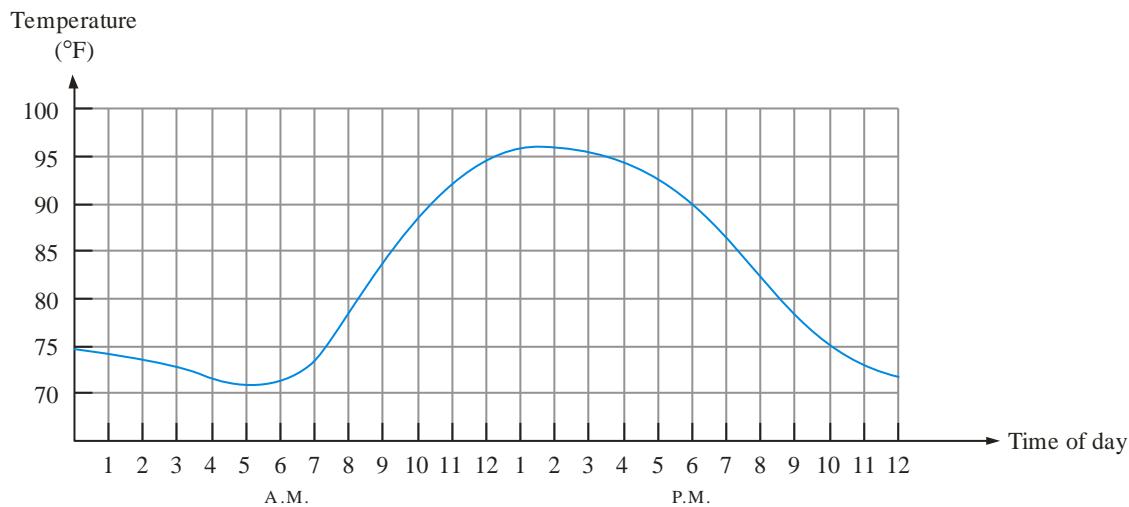
Electronic circuits can be divided into two broad categories digital and analog.

Digital electronics involves quantities with discrete values.

Analog electronics involves quantities with continuous values.

► FIGURE 1-1

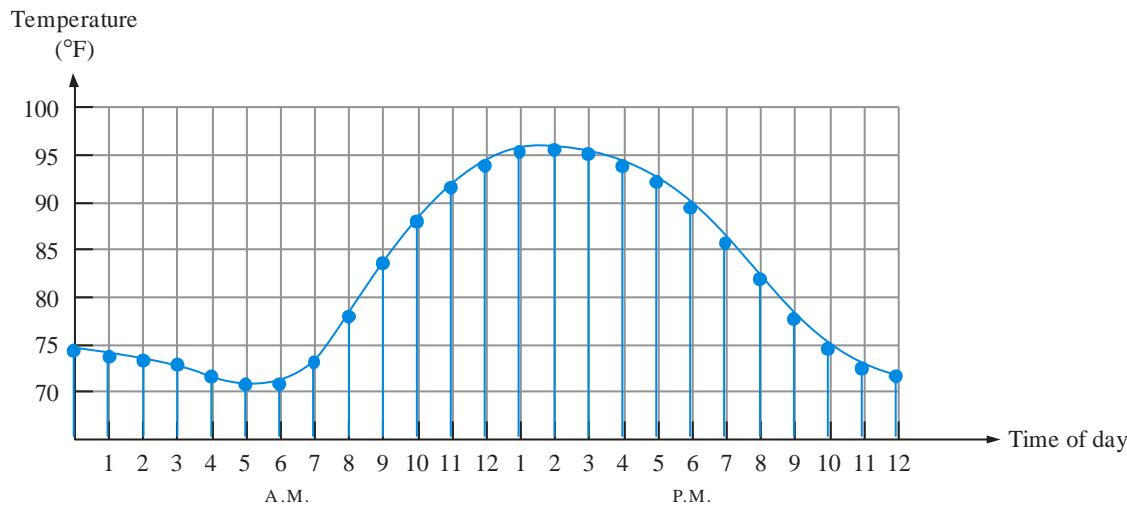
Graph of an analog quantity (temperature versus time).



Summary

Analog Quantities

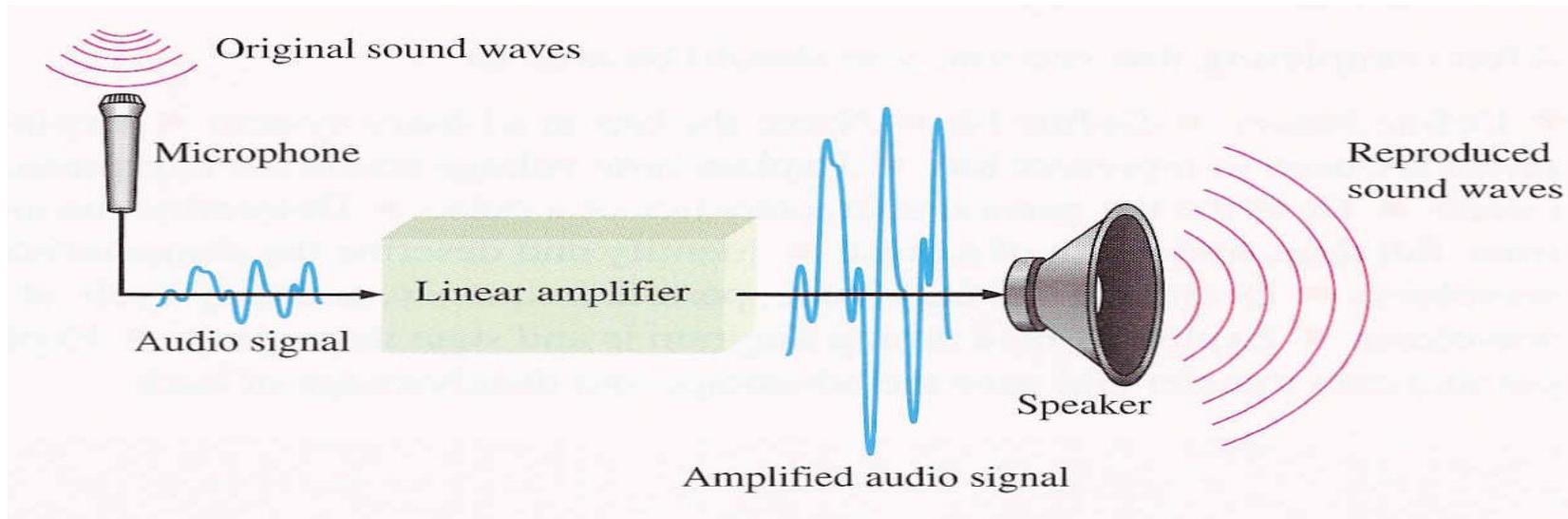
Most natural quantities that we see are **analog** and vary continuously. Analog systems can generally handle higher power than digital systems.



Digital systems can process, store, and transmit data more efficiently but can only assign discrete values to each point.

Summary

An Analog Electronic System



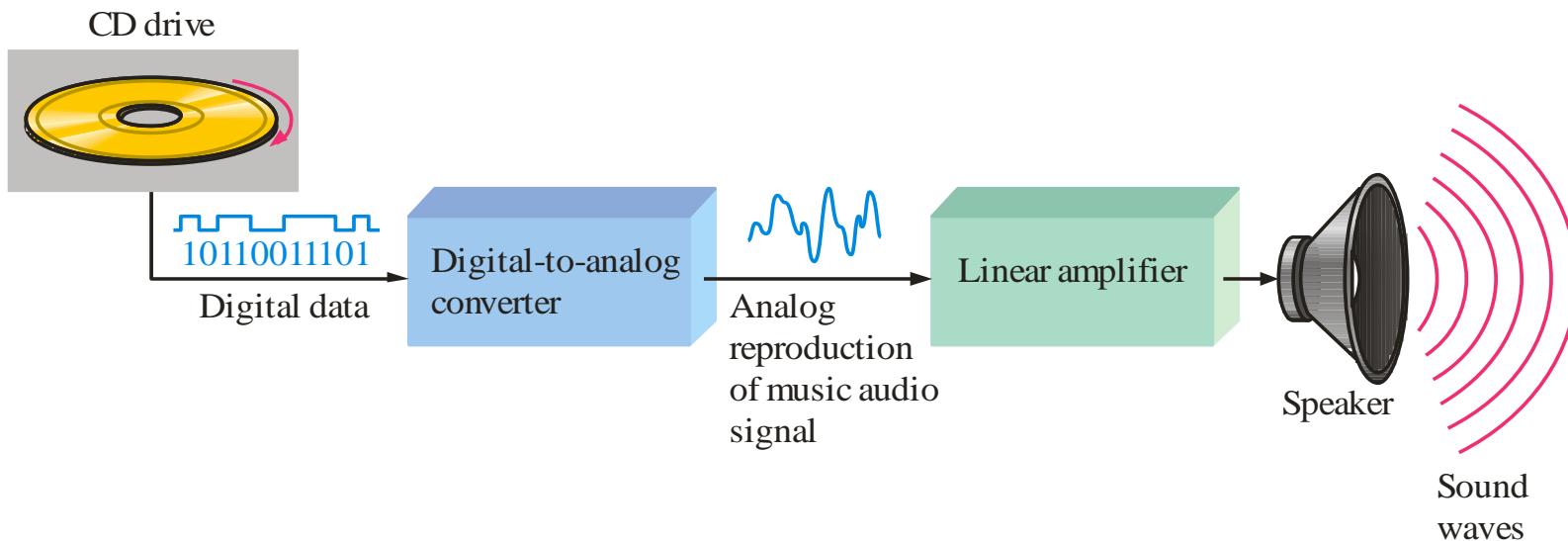
◀ FIGURE 1–3

A basic audio public address system.

Summary

Analog and Digital Systems

Many systems use a mix of analog and digital electronics to take advantage of each technology. A typical CD player accepts digital data from the CD drive and converts it to an analog signal for amplification.

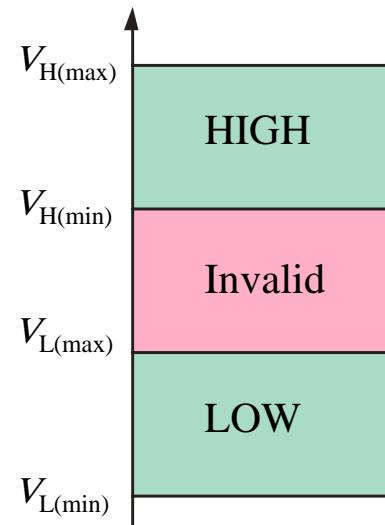


Summary

Binary Digits and Logic Levels

Digital electronics uses circuits that have two states, which are represented by two different voltage levels called HIGH and LOW. The voltages represent numbers in the binary system.

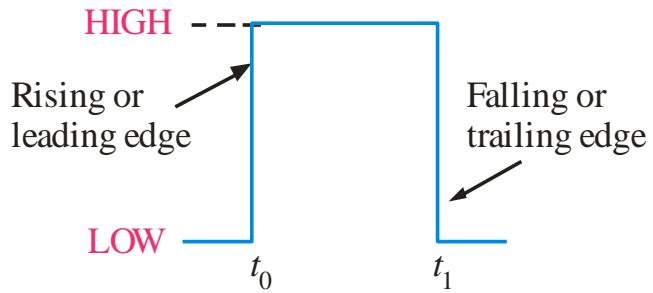
In binary, a single number is called a *bit* (for *binary digit*). A bit can have the value of either a 0 or a 1, depending on if the voltage is HIGH or LOW.



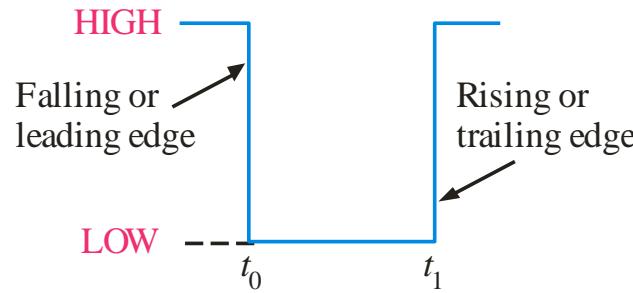
Summary

Digital Waveforms

Digital waveforms change between the LOW and HIGH levels. A positive going pulse is one that goes from a normally LOW logic level to a HIGH level and then back again. Digital waveforms are made up of a series of pulses.



(a) Positive-going pulse

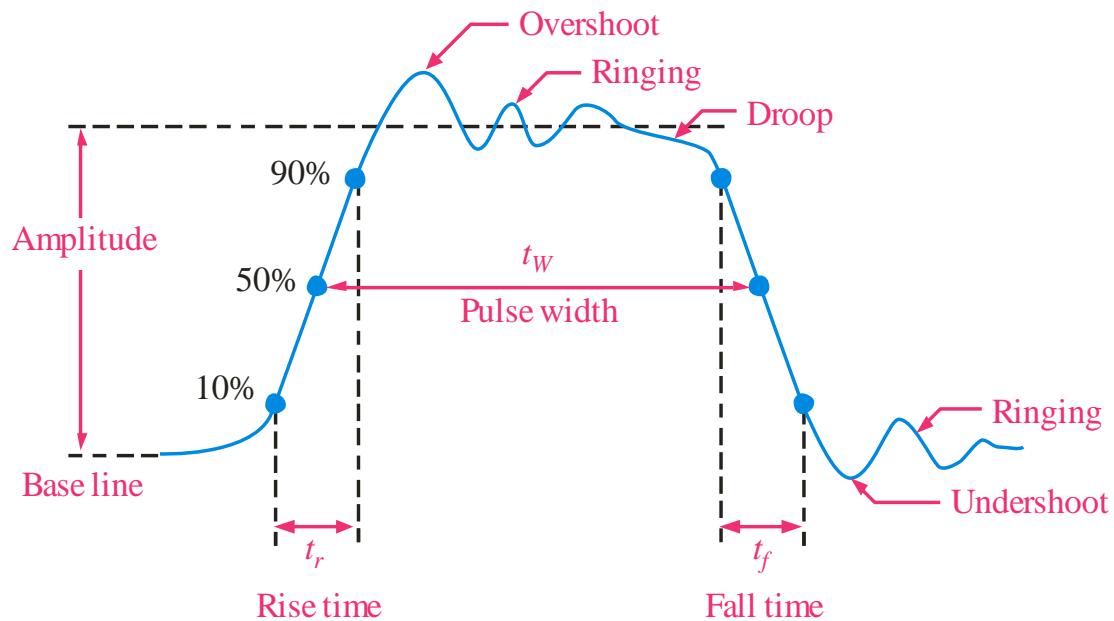


(b) Negative-going pulse

Summary

Pulse Definitions

Actual pulses are not ideal but are described by the rise time, fall time, amplitude, and other characteristics.





Summary

Periodic Pulse Waveforms

Periodic pulse waveforms are composed of pulses that repeats in a fixed interval called the **period**. The **frequency** is the rate it repeats and is measured in hertz.

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

The **clock** is a basic timing signal that is an example of a periodic wave.

Example

What is the period of a repetitive wave if $f = 3.2 \text{ GHz}$?

Solution

$$T = \frac{1}{f} = \frac{1}{3.2 \text{ MHz}} = 312.5 \text{ Ps}$$



Summary

International System of Units (SI multiples)

SI multiples of hertz (Hz)

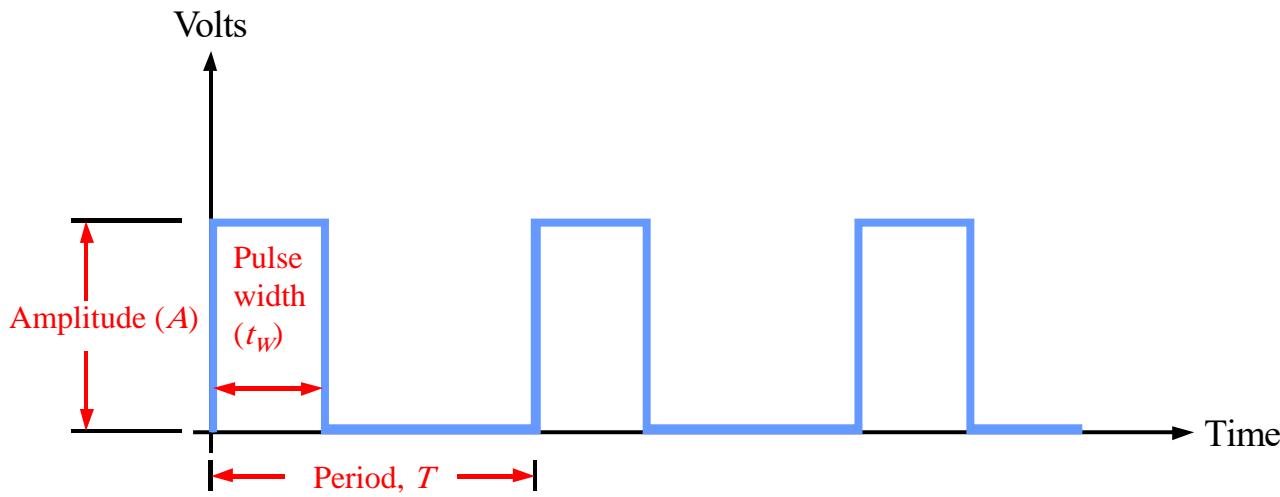
Submultiples			Multiples		
Value	SI symbol	Name	Value	SI symbol	Name
10^{-1} Hz	dHz	decihertz	10^1 Hz	daHz	decahertz
10^{-2} Hz	cHz	centihertz	10^2 Hz	hHz	hectohertz
10^{-3} Hz	mHz	millihertz	10^3 Hz	kHz	kilohertz
10^{-6} Hz	μ Hz	microhertz	10^6 Hz	MHz	megahertz
10^{-9} Hz	nHz	nanohertz	10^9 Hz	GHz	gigahertz
10^{-12} Hz	pHz	picohertz	10^{12} Hz	THz	terahertz
10^{-15} Hz	fHz	femtohertz	10^{15} Hz	PHz	petahertz
10^{-18} Hz	aHz	attohertz	10^{18} Hz	EHz	exahertz
10^{-21} Hz	zHz	zeptohertz	10^{21} Hz	ZHz	zettahertz
10^{-24} Hz	yHz	yoctohertz	10^{24} Hz	YHz	yottahertz

Common prefixed units are in bold face.

Summary

Pulse Definitions

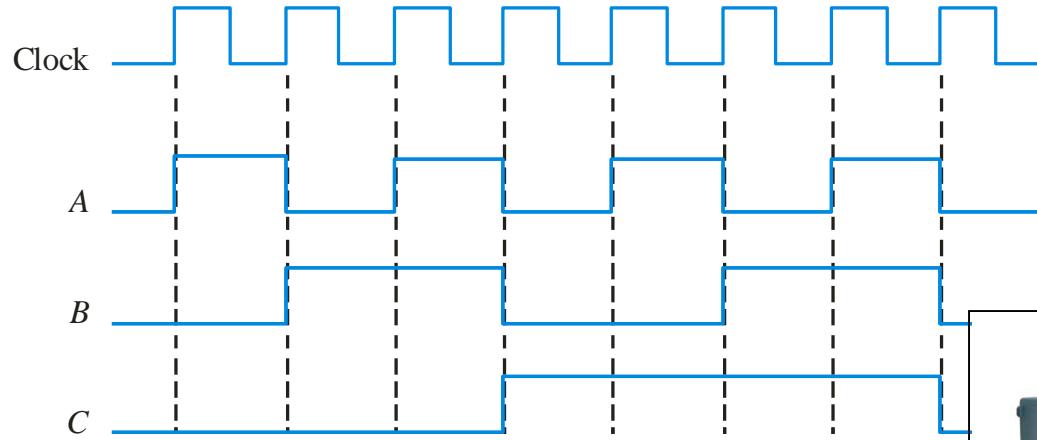
In addition to frequency and period, repetitive pulse waveforms are described by the amplitude (A), pulse width (t_W) and duty cycle. Duty cycle is the ratio of t_W to T .



Summary

Timing Diagrams

A timing diagram is used to show the relationship between two or more digital waveforms,



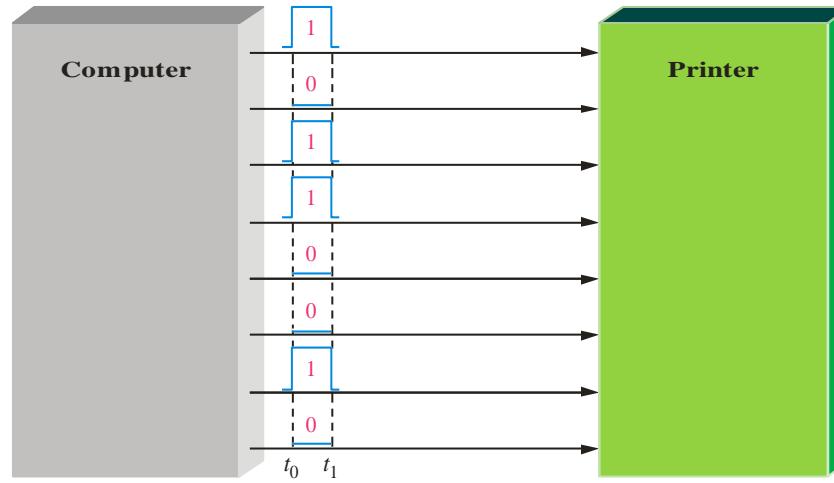
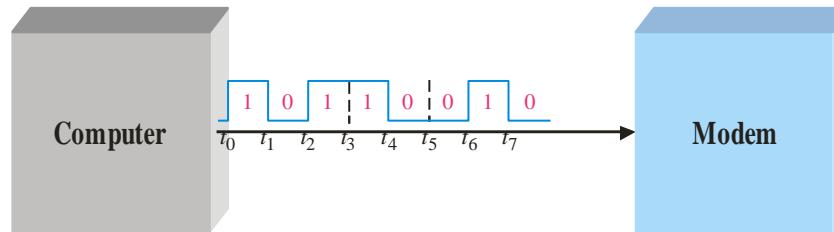
A diagram like this can be observed directly on a logic analyzer.



Summary

Serial and Parallel Data

Data can be transmitted by either serial transfer or parallel transfer.



Summary

Basic Logic Functions

AND

True only if *all* input conditions are true.



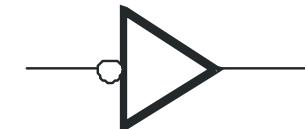
OR

True only if *one or more* input conditions are true.



NOT

Indicates the *opposite* condition.

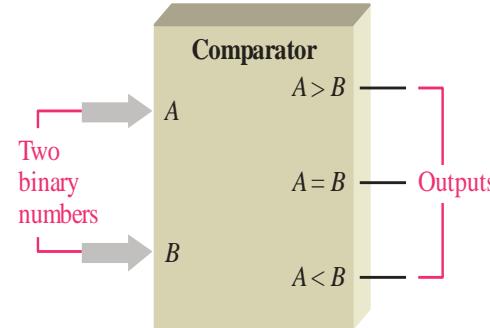


Summary

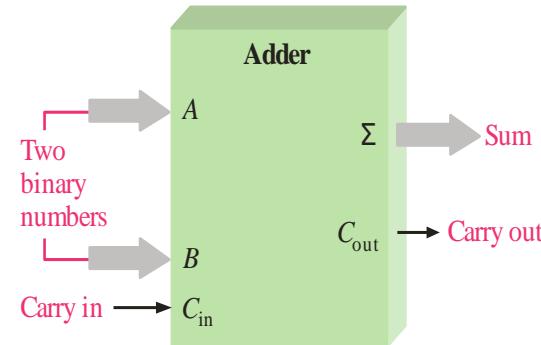
Basic System Functions

And, or, and not elements can be combined to form various logic functions. A few examples are:

The comparison function



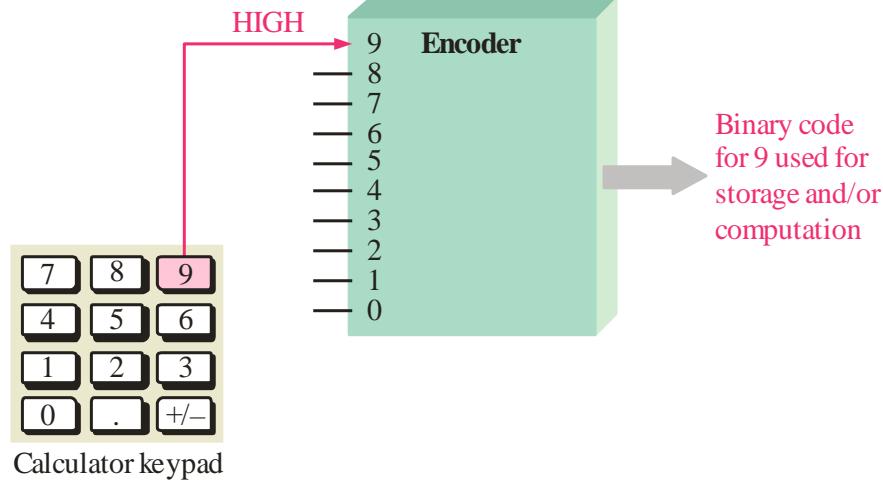
Basic arithmetic functions



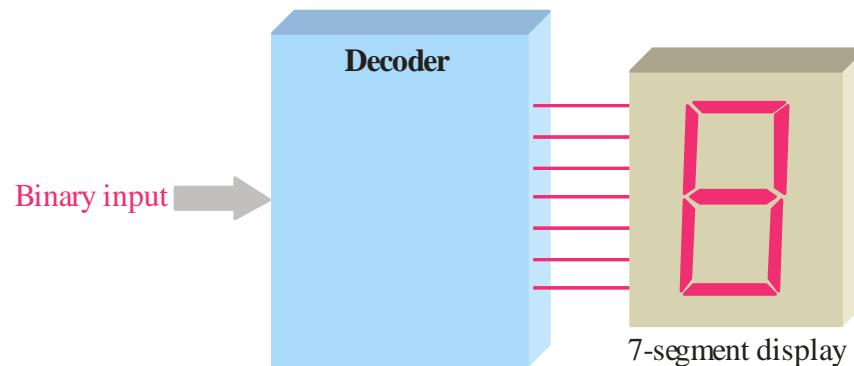
Summary

Basic System Functions

The encoding function



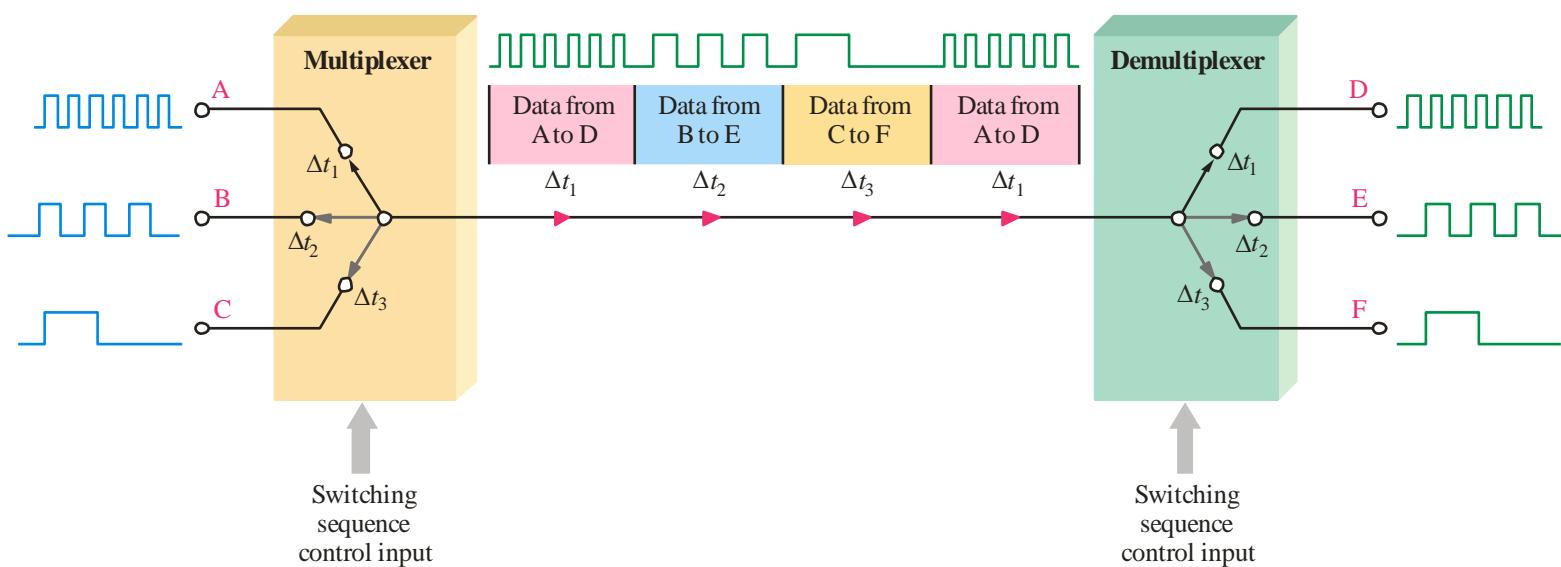
The decoding function



Summary

Basic System Functions

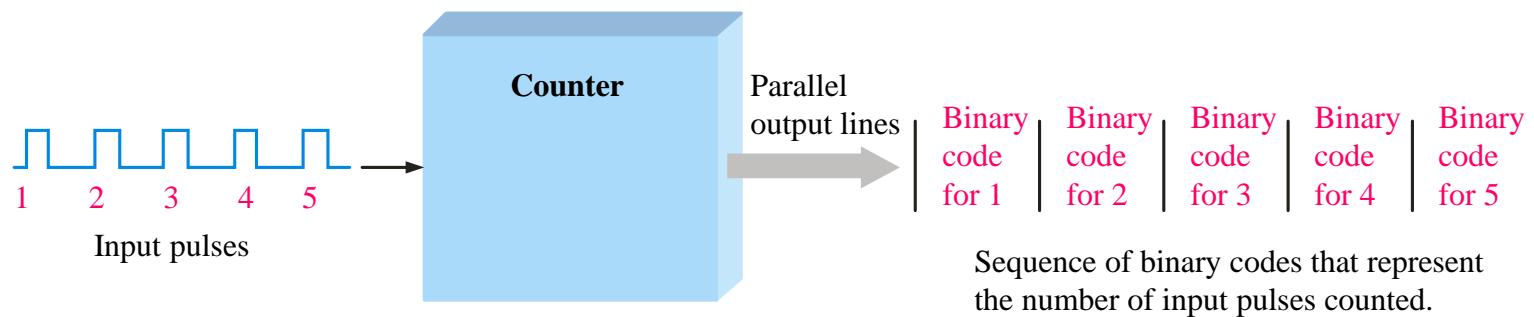
The data selection function



Summary

Basic System Functions

The counting function

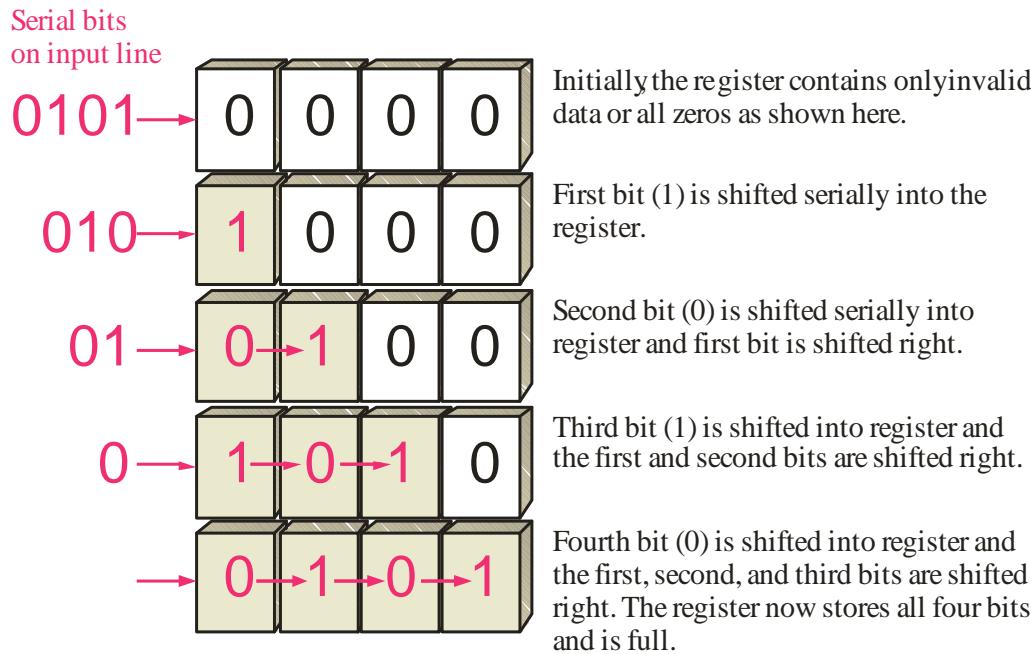


...and other functions such as code conversion and storage.

Summary

Basic System Functions

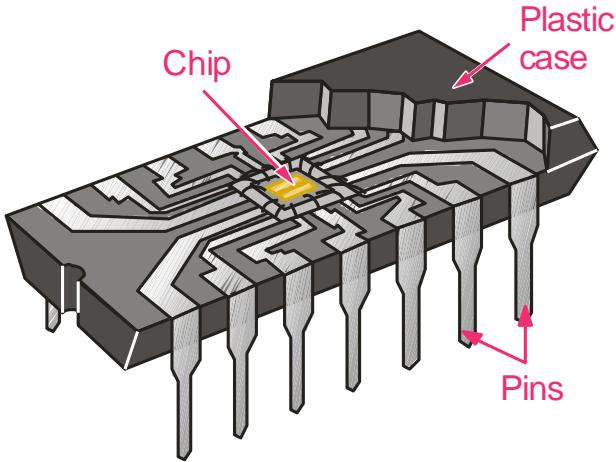
One type of storage function is the shift register, that moves and stores data each time it is clocked.



Summary

Integrated Circuits

Cutaway view of DIP (Dual-In-line Pins) chip:



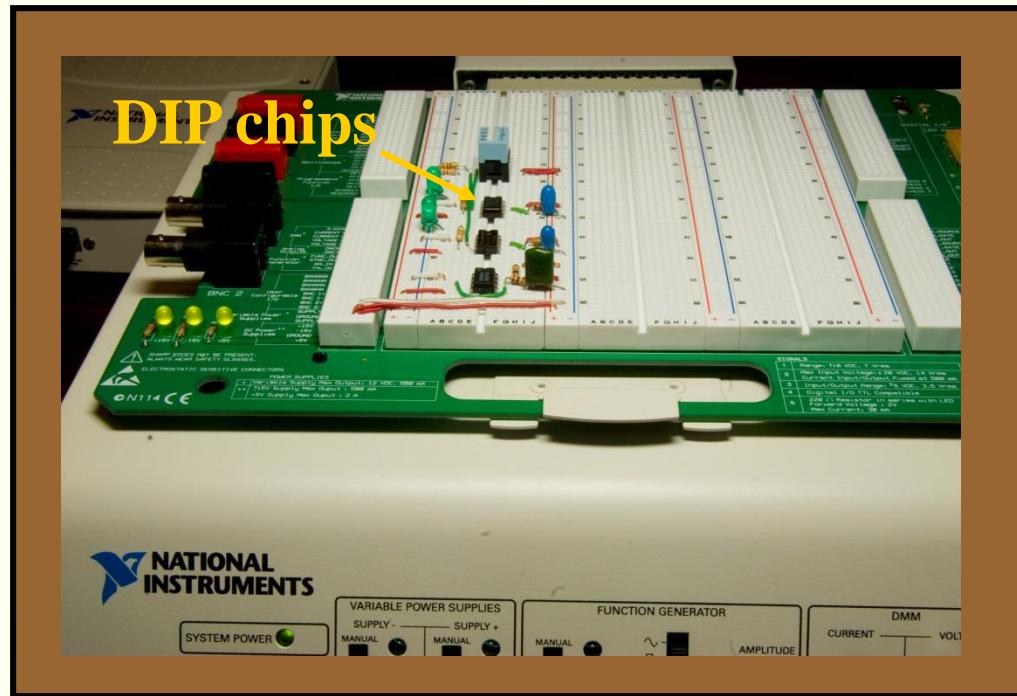
The Transistor-transistor logic (TTL) series, available as DIPs are popular for laboratory experiments with logic.

Summary

Integrated Circuits

An example of laboratory prototyping is shown. The circuit is wired using DIP chips and tested.

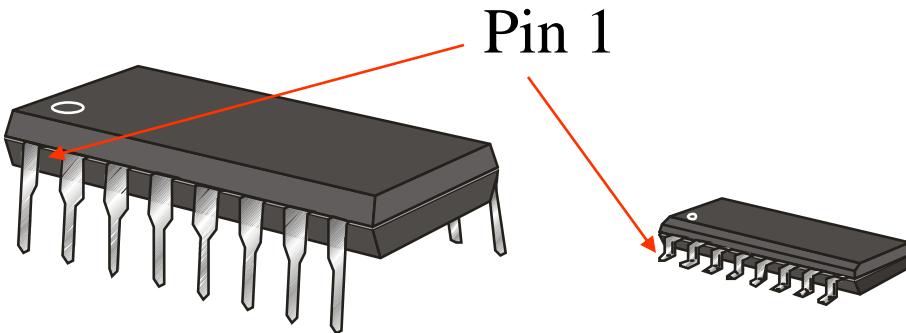
In this case, testing can be done by a computer connected to the system.



Summary

Integrated Circuits

DIP chips and surface mount chips



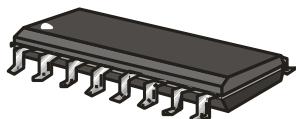
Dual in-line package

Small outline IC (SOIC)

Summary

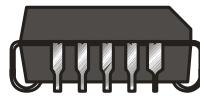
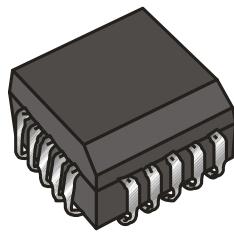
Integrated Circuits

Other surface mount packages:



End view

SOIC



End view

PLCC



End view

LCCC

Quiz

1. Compared to analog systems, digital systems
 - a. are less prone to noise
 - b. can represent an infinite number of values
 - c. can handle much higher power
 - d. all of the above

Quiz

2. The number of values that can be assigned to a bit are
 - a. one
 - b. two
 - c. three
 - d. ten

Quiz

3. The time measurement between the 50% point on the leading edge of a pulse to the 50% point on the trailing edge of the pulse is called the
- a. rise time
 - b. fall time
 - c. period
 - d. pulse width

Quiz

4. The time measurement between the 90% point on the trailing edge of a pulse to the 10% point on the trailing edge of the pulse is called the
- a. rise time
 - b. fall time
 - c. period
 - d. pulse width

Quiz

5. The reciprocal of the frequency of a clock signal is the
 - a. rise time
 - b. fall time
 - c. period
 - d. pulse width

Quiz

6. If the period of a clock signal is 500 ps, the frequency is
 - a. 20 MHz
 - b. 200 MHz
 - c. 2 GHz
 - d. 20 GHz

Quiz

7. AND, OR, and NOT gates can be used to form
- a. storage devices
 - b. comparators
 - c. data selectors
 - d. all of the above

Quiz

8. A shift register is an example of a
- a. storage device
 - b. comparator
 - c. data selector
 - d. counter

Quiz

9. A device that is used to switch one of several input lines to a single output line is called a
- a. comparator
 - b. decoder
 - c. counter
 - d. multiplexer

Quiz

10. For most digital work, an oscilloscope should be coupled to the signal using
- a. ac coupling
 - b. dc coupling
 - c. GND coupling
 - d. none of the above

Chapter 2



Summary

Decimal Numbers

The position of each digit in a weighted number system is assigned a weight based on the **base** or **radix** of the system. The radix of decimal numbers is ten, because only ten symbols (0 through 9) are used to represent any number.

The column weights of decimal numbers are powers of ten that increase from right to left beginning with $10^0 = 1$:

$\dots 10^5 \ 10^4 \ 10^3 \ 10^2 \ 10^1 \ 10^0.$

For fractional decimal numbers, the column weights are negative powers of ten that decrease from left to right:

$10^2 \ 10^1 \ 10^0. \ 10^{-1} \ 10^{-2} \ 10^{-3} \ 10^{-4} \ \dots$



Summary

Decimal Numbers

Decimal numbers can be expressed as the sum of the products of each digit times the column value for that digit. Thus, the number 9240 can be expressed as

$$(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$$

or

$$9 \times 1,000 + 2 \times 100 + 4 \times 10 + 0 \times 1$$

Example

Express the number 480.52 as the sum of values of each digit.

Solution

$$480.52 = (4 \times 10^2) + (8 \times 10^1) + (0 \times 10^0) + (5 \times 10^{-1}) + (2 \times 10^{-2})$$



Summary

Binary Numbers

For digital systems, the binary number system is used. Binary has a radix of two and uses the digits 0 and 1 to represent quantities.

The column weights of binary numbers are powers of two that increase from right to left beginning with $2^0 = 1$:

$$\dots 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0.$$

For fractional binary numbers, the column weights are negative powers of two that decrease from left to right:

$$2^2 \ 2^1 \ 2^0. \ 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} \ \dots$$

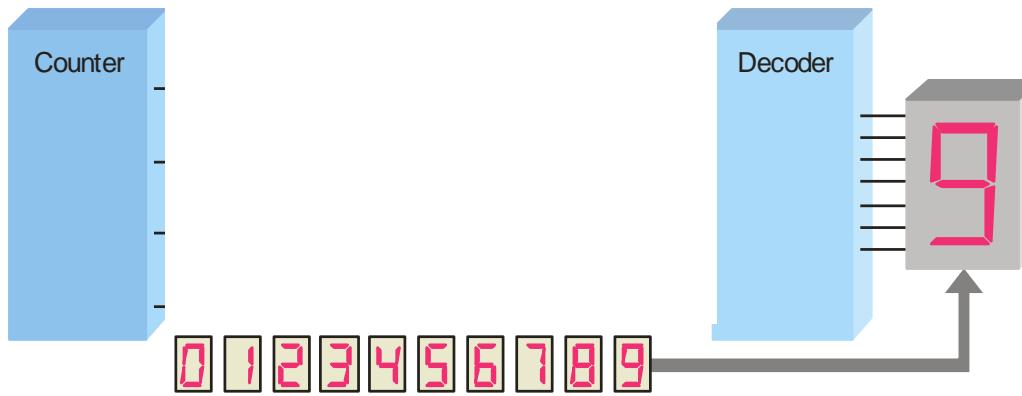
Summary

Binary Numbers

A binary counting sequence for numbers from zero to fifteen is shown.

Notice the pattern of zeros and ones in each column.

Digital counters frequently have this same pattern of digits:



Decimal Number	Binary Number
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
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Decimal Number	Binary Number
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
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1



Summary

Binary Conversions

The decimal equivalent of a binary number can be determined by adding the column values of all of the bits that are 1 and discarding all of the bits that are 0.

Example Solution

Convert the binary number 100101.01 to decimal.

Start by writing the column weights; then add the weights that correspond to each 1 in the number.

$$\begin{array}{ccccccccc} 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} \\ 32 & 16 & 8 & 4 & 2 & 1 & \frac{1}{2} & \frac{1}{4} \\ 1 & 0 & 0 & 1 & 0 & 1. & 0 & 1 \\ 32 & & +4 & & +1 & & +\frac{1}{4} = & 37\frac{1}{4} \end{array}$$



Summary

Binary Conversions

You can convert a decimal whole number to binary by reversing the procedure. Write the decimal weight of each column and place 1's in the columns that sum to the decimal number.

Example

Convert the decimal number 49 to binary.

Solution

The column weights double in each position to the right. Write down column weights until the last number is larger than the one you want to convert.

$$2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0.$$

$$64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1.$$

$$0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1.$$

Summary

Binary Conversions

You can convert a decimal fraction to binary by repeatedly multiplying the fractional results of successive multiplications by 2. The carries form the binary number.

Example

Convert the decimal fraction 0.188 to binary by repeatedly multiplying the fractional results by 2.

Solution

$$\begin{array}{ll} 0.188 \times 2 = 0.376 & \text{carry} = 0 \\ 0.376 \times 2 = 0.752 & \text{carry} = 0 \\ 0.752 \times 2 = 1.504 & \text{carry} = 1 \\ 0.504 \times 2 = 1.008 & \text{carry} = 1 \\ 0.008 \times 2 = 0.016 & \text{carry} = 0 \end{array}$$

↓ MSB

Answer = .00110 (for five significant digits)

Summary

Binary Conversions

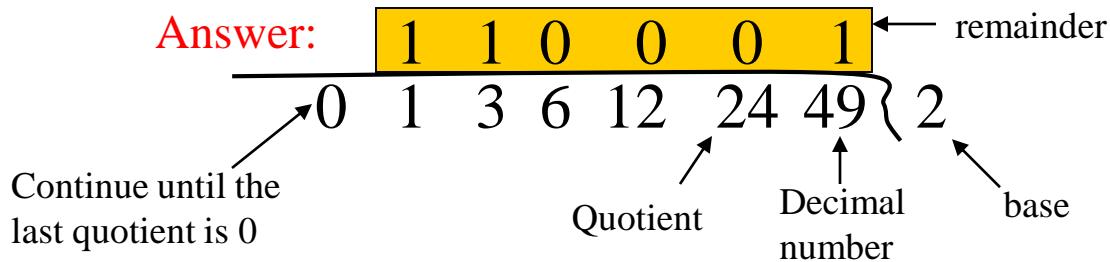
You can convert decimal to any other base by repeatedly dividing by the base. For binary, repeatedly divide by 2:

Example

Convert the decimal number 49 to binary by repeatedly dividing by 2.

Solution

You can do this by “reverse division” and the answer will read from left to right. Put quotients to the left and remainders on top.



Summary

Octal Numbers

Octal uses eight characters the numbers 0 through 7 to represent numbers.

There is no 8 or 9 character in octal.

Binary number can easily be converted to octal by grouping bits 3 at a time and writing the equivalent octal character for each group.

Example

Express $1\ 001\ 011\ 000\ 001\ 110_2$ in octal:

Solution

Group the binary number by 3-bits starting from the right. Thus, 113016_8

Decimal	Octal	Binary
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

Summary

Octal Numbers

Octal is also a weighted number system. The column weights are powers of 8, which increase from right to left.

$$\text{Column weights } \{ \begin{matrix} 8^3 & 8^2 & 8^1 & 8^0 \\ 512 & 64 & 8 & 1 \end{matrix} .$$

Example Express 3702_8 in decimal.

Solution Start by writing the column weights:

$$\begin{array}{cccc} 512 & 64 & 8 & 1 \\ 3 & 7 & 0 & 2_8 \end{array}$$

$$3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}$$

Decimal	Octal	Binary
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111



Octal Conversions: binary to octal

- group the binary positions in groups of three
- Convert the following binary numbers into octal:
 - a) 10110111
 - b) 01101100
- Solution
 - $10110111 = 010\ 110\ 111 = 267$
 - $01101100 = 001\ 101\ 100 = 154$



Octal Conversions: octal to binary

- replace each octal number with three equivalent binary numbers even if the number can be represented by less than three bits
- Convert the following octal number into binary:
 - a) 327 b) 601
- Solution
- a) $327 = 011\ 010\ 111 = 11010111$
- b) $601 = 110\ 000\ 001 = 110000001$



Summary

Octal Conversions: octal to decimal

- To convert from octal to decimal, (multiply by weighting factors).
- Convert $(713)_8$ to decimal.
- Solution
 - $713 = 7 \times 8^2 + 1 \times 8^1 + 3 \times 8^0 = 459$



Summary

Octal Conversions: decimal to octal

- To convert from decimal to octal, the successive-division procedure or the sum of weights procedure can be used



Summary

Octal Conversions (contd.)

- Convert the following decimal numbers to octal:

a) $(596)_{10}$

b) $(1000)_{10}$

- Solution

a) $596 \div 8 = 74$ remainder 4

$74 \div 8 = 9$ remainder 2

$9 \div 8 = 1$ remainder 1

$1 \div 8 = 0$ remainder 1

	8^3	8^2	8^1	8^0
	512	64	8	1
596 =	1	1	2	4
1000 =	1	7	5	0

b) $1000 \div 8 = 125$ remainder 0

1124

$125 \div 8 = 15$ remainder 5

1750

$15 \div 8 = 1$ remainder 7

$1 \div 8 = 0$ remainder 1

Summary

Hexadecimal Numbers

Hexadecimal uses sixteen characters to represent numbers: the numbers 0 through 9 and the alphabetic characters A through F.

Large binary number can easily be converted to hexadecimal by grouping bits 4 at a time and writing the equivalent hexadecimal character.

Example

Express $1001\ 0110\ 0000\ 1110_2$ in hexadecimal:

Solution

Group the binary number by 4-bits starting from the right. Thus, **960E**

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Summary

Hexadecimal Numbers

Hexadecimal is a weighted number system. The column weights are powers of 16, which increase from right to left.

$$\text{Column weights } \{ \begin{array}{cccc} 16^3 & 16^2 & 16^1 & 16^0 \\ 4096 & 256 & 16 & 1 \end{array} .$$

Example Express $1A2F_{16}$ in decimal.

Solution Start by writing the column weights:
4096 256 16 1

1 A 2 F₁₆

$$1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10}$$

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Summary

Hexadecimal Numbers

- The 16 allowable digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F
- The weights are powers of 16.

Decimal	Binary	Hexadecimal
0	0000 0000	0 0
1	0000 0001	0 1
2	0000 0010	0 2
3	0000 0011	0 3
4	0000 0100	0 4
5	0000 0101	0 5
6	0000 0110	0 6
7	0000 0111	0 7
8	0000 1000	0 8
9	0000 1001	0 9
10	0000 1010	0 A
11	0000 1011	0 B
12	0000 1100	0 C
13	0000 1101	0 D
14	0000 1110	0 E
15	0000 1111	0 F



Summary

Hexadecimal Conversion: binary to hexadecimal

- grouping the binary positions in groups of four
- Convert the following binary numbers into hexadecimal: a) 10101111 b) 01101100
- **Solution:**
 - $10110111 = 1011 \ 0111 = (B \ 7)_{16}$
 - $01101100 = 0110 \ 1100 = (6 \ C)_{16}$



Summary

Hexadecimal Conversion: Hexadecimal to binary

- replace each hexadecimal number with four equivalent binary numbers even if the number can be represented by less than four bits
- Convert the following hexadecimal number into binary:
 - a) A2E
 - b) 60F
- **Solution:**
 - a) $(A2E)_{16} = 1010\ 0010\ 1110$
 $= (101000101110)_2$
 - b) $(60F)_{16} = 0110\ 0000\ 1111$
 $= (011000001111)_2$



Summary

Hexadecimal Conversion: Hexadecimal to decimal

- To convert from hexadecimal to decimal, (multiply by weighting factors).
- Convert (7AD)₁₆ to decimal.
- **Solution:**
 - $$(7AD)_{16} = 7 \times 16^2 + 10 \times 16^1 + 13 \times 16^0$$
$$= (1965)_{10}$$



Summary

Hexadecimal Conversion: decimal to Hexadecimal

- To convert from *decimal to hexadecimal*, the successive-division procedure or the sum of weights procedure can be used.
- Convert the following decimal numbers to hexadecimal:
- a) $(596)_{10}$ b) $(1000)_{10}$

Solution:

- $596 \div 16 = 37$ remainder 4
- $37 \div 16 = 2$ remainder 5 254
- $2 \div 16 = 0$ remainder 2

- $1000 \div 16 = 62$ remainder 8
- $62 \div 16 = 3$ remainder 14 3E8
- $3 \div 16 = 0$ remainder 3



Number with Different Base

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



Summary

Binary Addition

The rules for binary addition are

$0 + 0 = 0$	Sum = 0, carry = 0
$0 + 1 = 1$	Sum = 1, carry = 0
$1 + 0 = 1$	Sum = 1, carry = 0
$1 + 1 = 10$	Sum = 0, carry = 1

When an input carry = 1 due to a previous result, the rules are

$1 + 0 + 0 = 01$	Sum = 1, carry = 0
$1 + 0 + 1 = 10$	Sum = 0, carry = 1
$1 + 1 + 0 = 10$	Sum = 0, carry = 1
$1 + 1 + 1 = 11$	Sum = 1, carry = 1



Summary

Binary Addition

Example Add the binary numbers 00111 and 10101 and show the equivalent decimal addition.

Solution

$$\begin{array}{r} \text{0 } \text{1 } \text{1 } \text{1 } \\ 00111 \quad \quad 7 \\ 10101 \quad \quad 21 \\ \hline 11100 = 28 \end{array}$$



Summary

Binary Subtraction

The rules for binary subtraction are

$$0 - 0 = 0$$

$$1 - 1 = 0$$

$$1 - 0 = 1$$

$$10 - 1 = 1 \quad \text{0 - 1 with a borrow of 1}$$

Example

Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

Solution

$$\begin{array}{r} 111 \\ 10101 \\ - 00111 \\ \hline 01110 \end{array} \quad \begin{array}{r} 21 \\ - 7 \\ \hline 14 \end{array}$$

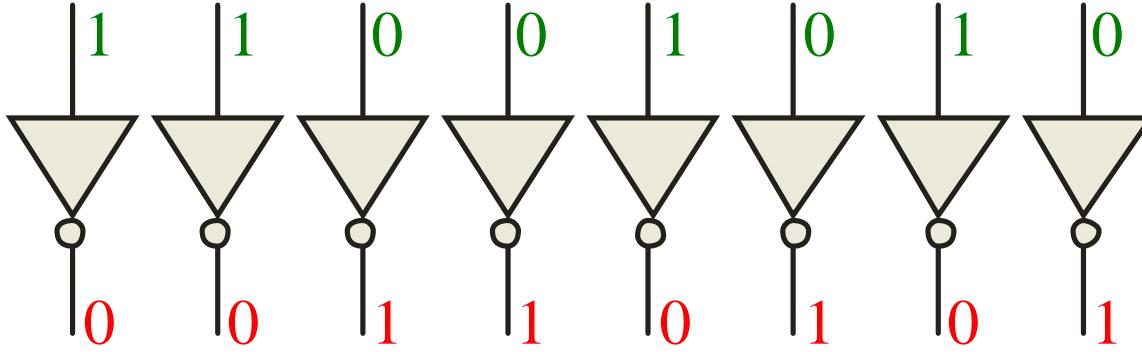
Summary

1's Complement

The 1's complement of a binary number is just the inverse of the digits. To form the 1's complement, change all 0's to 1's and all 1's to 0's.

For example, the 1's complement of 11001010 is
00110101

In digital circuits, the 1's complement is formed by using inverters:



Summary

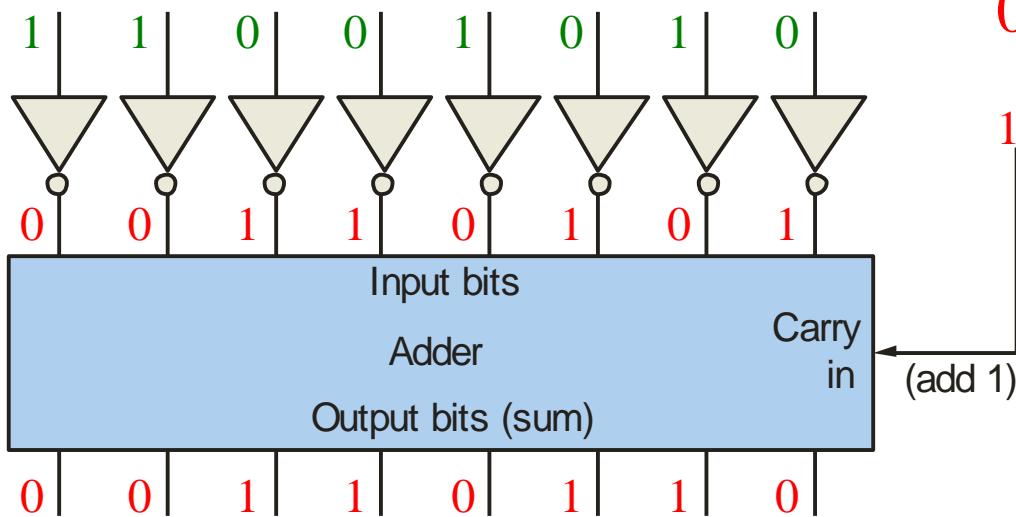
2's Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

Recall that the 1's complement of 11001010 is

00110101 (1's complement)

To form the 2's complement, add 1:



$$\begin{array}{r} & & & & & & +1 \\ & & & & & & \\ 00110101 & & & & & & \text{(1's complement)} \\ \hline 00110110 & & & & & & \text{(2's complement)} \end{array}$$

Summary

Signed Binary Numbers

There are several ways to represent signed binary numbers. In all cases, the MSB in a signed number is the sign bit, that tells you if the number is positive or negative.

Computers use a modified 2's complement for signed numbers. Positive numbers are stored in *true* form (with a 0 for the sign bit) and negative numbers are stored in *complement* form (with a 1 for the sign bit).

For example, the positive number 58 is written using 8-bits as
00111010 (true form).

Sign bit

Magnitude bits

Summary

Signed Binary Numbers

Negative numbers are written as the 2's complement of the corresponding positive number.

The negative number -58 is written as:

$$-58 = \begin{matrix} 1 \\ 1000110 \end{matrix} \text{ (complement form)}$$

Sign bit Magnitude bits

An easy way to read a signed number that uses this notation is to assign the sign bit a column weight of -128 (for an 8-bit number). Then add the column weights for the 1's.

Example Solution

Assuming that the sign bit $= -128$, show that $11000110 = -58$ as a 2's complement signed number:

Column weights: $-128 \ 64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1$.

$$\begin{array}{r} 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \\ -128 +64 \qquad \qquad \qquad +4 +2 \qquad = -58 \end{array}$$

Summary

Arithmetic Operations with Signed Numbers

- Using the signed number notation with negative numbers in 2's complement form simplifies addition and subtraction of signed numbers.
- This section will cover only on the 2's complement arithmetic, because, it widely used in computers and microprocessor-based system .

$$00011110 = +30$$

$$00001111 = +15$$

$$00101101 = +45$$

$$000001110 = +14$$

$$11101111 = -17$$

$$11111101 = -3$$

$$11111111 = -1$$

$$11111000 = -8$$

$$111110111 = -9$$

Discard carry

Summary

Arithmetic Operations with Signed Numbers

Addition

Rules for **addition**: Add the two signed numbers. Discard any final carries. The result is in signed form.

Both Number Positive:

$$\begin{array}{r} 00000111 \\ +00000100 \end{array}$$

$7 + 4$

0001011

The Sum is Positive and is therefore in true binary

Positive Number with
Magnitude Larger than
Negative Number:

Discard
Carry

$$\begin{array}{r} 00001111 \\ +11111010 \end{array}$$

$15 + (-6)$

1

0001001

The Final Carry is Discarded.
The Sum is Positive and is therefore in true binary



Arithmetic Operations with Signed Numbers

Addition

**Negative Number with
Magnitude Larger than
Positive Number:**

$$\begin{array}{r} 0001000 \\ +1110100 \\ \hline 11111000 \end{array}$$

$$16 + (-24)$$

The Sum is Negative and is therefore in 2's complement form

Both Number Negative:

Discard
Carry

$$\begin{array}{r} 11111011 \\ +11110111 \\ \hline 11110010 \end{array}$$

$$-5 + (-9)$$

The Final Carry is Discarded.
The Sum is Negative and is therefore in 2's complement form

Summary

Arithmetic Operations with Signed Numbers

Addition

Note that if the number of bits required for the answer is exceeded, overflow will occur. This occurs only if both numbers have the same sign. The overflow will be indicated by an incorrect sign bit.

Two examples are:

$$\begin{array}{r} 01000000 = +128 \\ 01000001 = +129 \\ \hline 10000001 = -126 \end{array}$$

Discard carry →

$$\begin{array}{r} 10000001 = -127 \\ 10000001 = -127 \\ \hline 100000010 = +2 \end{array}$$

Wrong! The answer is incorrect
and the sign bit has changed.

Summary

Arithmetic Operations with Signed Numbers

Subtraction

- Rules for **subtraction**: To Subtract two signed numbers, take the **2's Complement** of the subtrahend and **ADD** the numbers. The results is in signed form. **Discard** any final carry bit
- To subtract A-B, take two's comp of B and add to A

Example:

$$00001000 - 00000011$$

$$8 - 3 = 8 + (-3) = 5$$

Solution:

Discard Cary \rightarrow



$$00001000$$

$$+ \begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{matrix}$$

\longrightarrow 2's Complement

\longrightarrow Difference

Summary

Arithmetic Operations with Signed Numbers

Rules for **subtraction**: 2's complement the subtrahend and add the numbers. Discard any final carries. The result is in signed form.

Repeat the examples done previously, but subtract:

$$\begin{array}{r} 00011110 \quad (+30) \\ - 00001111 \quad -(+15) \\ \hline \end{array} \quad \begin{array}{r} 00001110 \quad (+14) \\ - 11101111 \quad -(-17) \\ \hline \end{array} \quad \begin{array}{r} 11111111 \quad (-1) \\ - 1111000 \quad -(-8) \\ \hline \end{array}$$

2's complement subtrahend and add:

$$\begin{array}{r} 00011110 = +30 \\ 11110001 = -15 \\ \hline 100001111 = +15 \end{array} \quad \begin{array}{r} 00001110 = +14 \\ 00010001 = +17 \\ \hline 00011111 = +31 \end{array} \quad \begin{array}{r} 11111111 = -1 \\ 00001000 = +8 \\ \hline 100000111 = +7 \end{array}$$

↑
Discard carry

Summary

Binary Multiplication

The binary multiplication table is simple:

$$0 * 0 = 0 \mid 1 * 0 = 0 \mid 0 * 1 = 0 \mid 1 * 1 = 1$$

Extending multiplication to multiple digits:

Multiplicand

1011

Multiplier

$\times \underline{101}$

Partial Products

1011

Product

0000 -

$\underline{1011 - -}$

110111

Summary

Binary Multiplication

- a) $11100 \times 101 = 10001100$
 - $(16+8+4) \times (4+1) = (128+8+4)$
 - $28 \times 5 = 140$
- b) $11011 \times 1101 = 101011111$
 - $(16+8+2+1) \times (8+4+1) = (256+64+16+8+4+2+1)$
 - $27 \times 13 = 351$

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \\ \times \quad \quad 1 \ 0 \ 1 \\ \hline 1 \ 1 \ 1 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \\ \hline 1 \ 1 \ 1 \ 0 \ 0 \\ \hline 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \ 1 \\ \times \quad \quad 1 \ 1 \ 0 \ 1 \ 1 \\ \hline 1 \ 1 \ 0 \ 1 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 0 \\ \hline 1 \ 1 \ 0 \ 1 \ 1 \\ 1 \ 1 \ 0 \ 1 \ 1 \\ \hline 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \end{array}$$



Summary

Binary Division

Use the same procedure as decimal division

Perform the following binary divisions:

(a) $110 \div 11$ (b) $110 \div 10$

Solution

(a)	$\overline{11} \overline{110}$	(b)	$\overline{10} \overline{110}$
	$\begin{array}{r} 10 \\ \hline 11 \\ -0 \\ \hline 11 \\ -0 \\ \hline 00 \end{array}$		$\begin{array}{r} 11 \\ \hline 10 \\ -0 \\ \hline 10 \\ -0 \\ \hline 00 \end{array}$
	$3\overline{)6}$		$2\overline{)6}$



Summary

Binary Division

- $11001 \div 101 = 101$
- $(16+8+1) \div (4+1) = (4+1)$
- $25 \div 5 = 5$

$$\begin{array}{r} 1\ 0\ 1 \\ \hline 1\ 1\ 0\ 0\ 1 \\ -1\ 0\ 1 \\ \hline 1\ 0\ 1 \\ -1\ 0\ 1 \\ \hline 0\ 0\ 0 \end{array}$$

Summary

Binary coded decimal (BCD)

Binary coded decimal (BCD) is a weighted code that is commonly used in digital systems when it is necessary to show decimal numbers such as in clock displays.

The table illustrates the difference between straight binary and BCD. BCD represents each decimal digit with a 4-bit code. Notice that the codes 1010 through 1111 are not used in BCD.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101



Summary

Binary coded decimal (BCD)

- is used to represent each of the ten decimal digits as a 4-bit binary code
- $(125)_{10} = (0001\ 0010\ 0101)_{BCD}$
- $(1476)_{10} = (0001\ 0100\ 0111\ 0110)_{BCD}$
- $(1000\ 0010\ 1001)_{BCD} = (829)_{10}$
- $(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$



Summary

Binary coded decimal (BCD)

$$\begin{array}{r} 4 \quad 0100 \\ +5 \quad +0101 \\ \hline 9 \quad 1001 \end{array} \qquad \begin{array}{r} 4 \quad 0100 \\ +8 \quad +1000 \\ \hline 12 \quad 1100 \end{array} \qquad \begin{array}{r} 8 \quad 1000 \\ +9 \quad +1001 \\ \hline 17 \quad 10001 \end{array}$$
$$\begin{array}{r} \\ \\ +0110 \\ \hline 10010 \end{array} \qquad \begin{array}{r} \\ \\ +0110 \\ \hline 10111 \end{array}$$

Summary

Binary coded decimal (BCD)

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

Summary

Gray code

Gray code is an unweighted code that has a single bit change between one code word and the next in a sequence. Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

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



ASCII

ASCII is a code for alphanumeric characters and control characters. In its original form, ASCII encoded 128 characters and symbols using 7-bits. The first 34 characters are control characters, that are based on obsolete teletype requirements, so these characters are generally assigned to other functions in modern usage.

In 1981, IBM introduced extended ASCII, which is an 8-bit code and increased the character set to 256. Other extended sets (such as Unicode) have been introduced to handle characters in languages other than English.

ASCII



American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	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



ASCII

Control Characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete



Summary

Parity Method

The parity method is a method of error detection for simple transmission errors involving one bit (or an odd number of bits). A parity bit is an “extra” bit attached to a group of bits to force the number of 1’s to be either even (even parity) or odd (odd parity).

EVEN PARITY		ODD PARITY	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

The BCD code with parity bits



Summary

Parity Method

Example

The ASCII character for “a” is 1100001 and for “A” is 1000001. What is the correct bit to append to make both of these have odd parity?

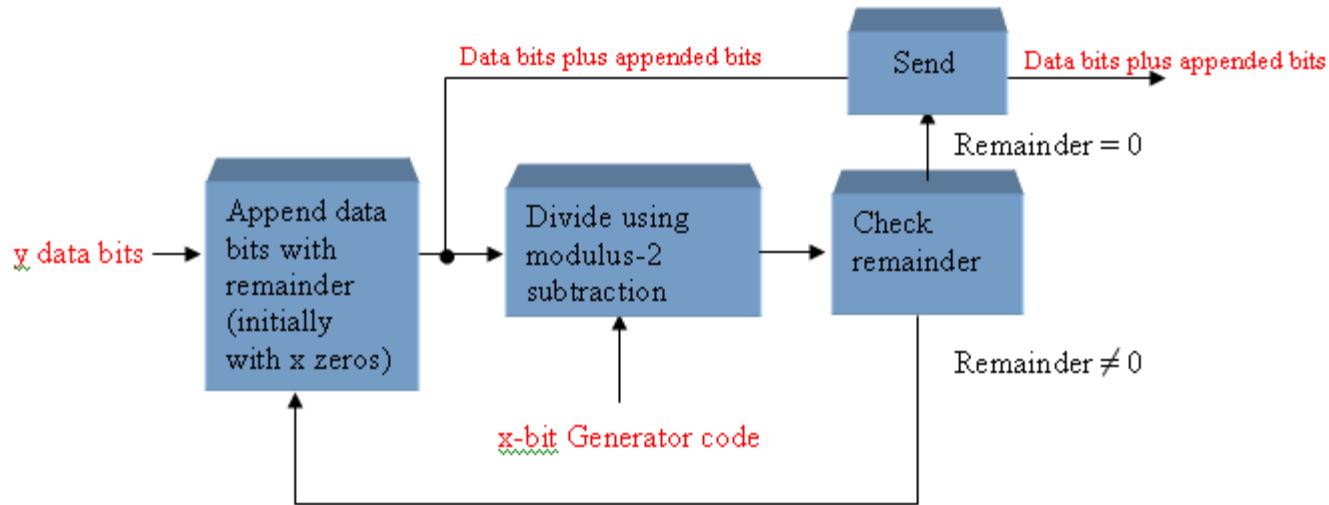
Solution

The ASCII “a” has an odd number of bits that are equal to 1; therefore the parity bit is **0**. The ASCII “A” has an even number of bits that are equal to 1; therefore the parity bit is **1**.

Summary

Cyclic Redundancy Check

The cyclic redundancy check (CRC) is an error detection method that can detect multiple errors in larger blocks of data. At the sending end, a checksum is appended to a block of data. At the receiving end, the check sum is generated and compared to the sent checksum. If the check sums are the same, no error is detected.



Selected Key Terms

Byte A group of eight bits

Floating-point number A number representation based on scientific notation in which the number consists of an exponent and a mantissa.

Hexadecimal A number system with a base of 16.

Octal A number system with a base of 8.

BCD Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

Selected Key Terms

Alphanumeric

Consisting of numerals, letters, and other characters

ASCII

American Standard Code for Information Interchange; the most widely used alphanumeric code.

Parity

In relation to binary codes, the condition of evenness or oddness in the number of 1s in a code group.

Cyclic

A type of error detection code.

redundancy check (CRC)

Quiz

1. For the binary number 1000, the weight of the column with the 1 is
- a. 4
 - b. 6
 - c. 8
 - d. 10

Quiz

2. The 2's complement of 1000 is
- a. 0111
 - b. 1000
 - c. 1001
 - d. 1010

Quiz

3. The fractional binary number 0.11 has a decimal value of
- a. $\frac{1}{4}$
 - b. $\frac{1}{2}$
 - c. $\frac{3}{4}$
 - d. none of the above

Quiz

4. The hexadecimal number 2C has a decimal equivalent value of
- a. 14
 - b. 44
 - c. 64
 - d. none of the above

Quiz

5. Assume that a floating point number is represented in binary. If the sign bit is 1, the
- a. number is negative
 - b. number is positive
 - c. exponent is negative
 - d. exponent is positive

Quiz

6. When two positive signed numbers are added, the result may be larger than the size of the original numbers, creating overflow. This condition is indicated by
- a. a change in the sign bit
 - b. a carry out of the sign position
 - c. a zero result
 - d. smoke

Quiz

7. The number 1010 in BCD is
- a. equal to decimal eight
 - b. equal to decimal ten
 - c. equal to decimal twelve
 - d. invalid

Quiz

8. An example of an unweighted code is
- a. binary
 - b. decimal
 - c. BCD
 - d. Gray code

Quiz

9. An example of an alphanumeric code is

- a. hexadecimal
- b. ASCII
- c. BCD
- d. CRC

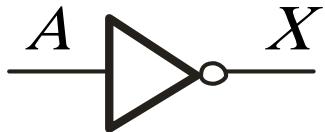
Quiz

10. An example of an error detection method for transmitted data is the
- a. parity check
 - b. CRC
 - c. both of the above
 - d. none of the above

Chapter 3



The Inverter



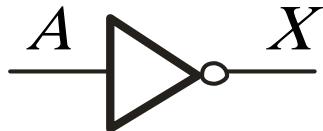
The inverter performs the Boolean **NOT** operation. When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW.

Input	Output
A	X
LOW (0)	HIGH (1)
HIGH (1)	LOW(0)

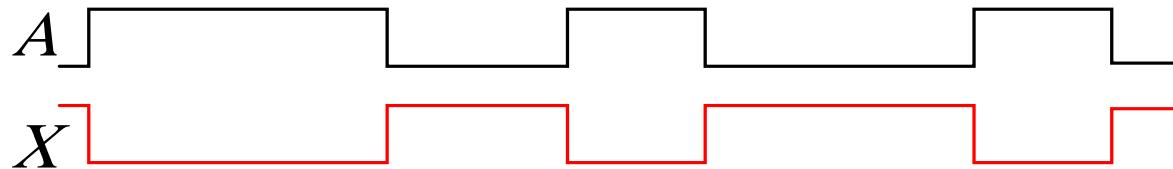
The **NOT** operation (complement) is shown with an overbar. Thus, the Boolean expression for an inverter is $X = \overline{A}$.

Summary

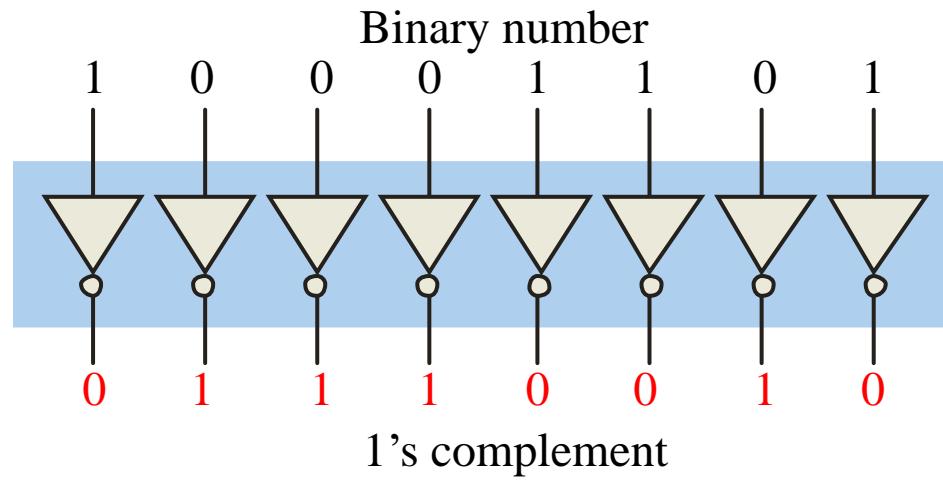
The Inverter



Example waveforms:

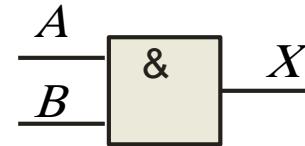
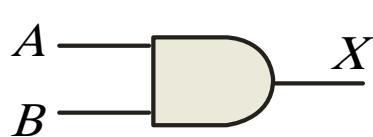


A group of inverters can be used to form the 1's complement of a binary number:



Summary

The AND Gate



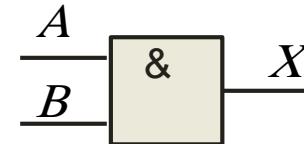
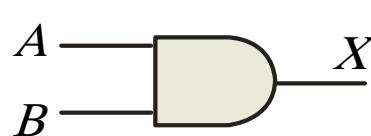
The **AND gate** produces a HIGH output when all inputs are HIGH; otherwise, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

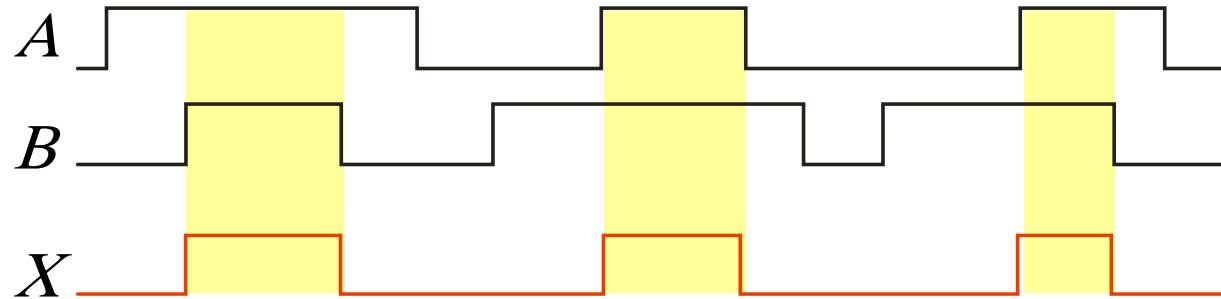
The **AND** operation is usually shown with a dot between the variables but it may be implied (no dot). Thus, the AND operation is written as $X = A \cdot B$ or $X = AB$.

Summary

The AND Gate



Example waveforms:



The AND operation is used in computer programming as a selective mask. If you want to retain certain bits of a binary number but reset the other bits to 0, you could set a mask with 1's in the position of the retained bits.

Example

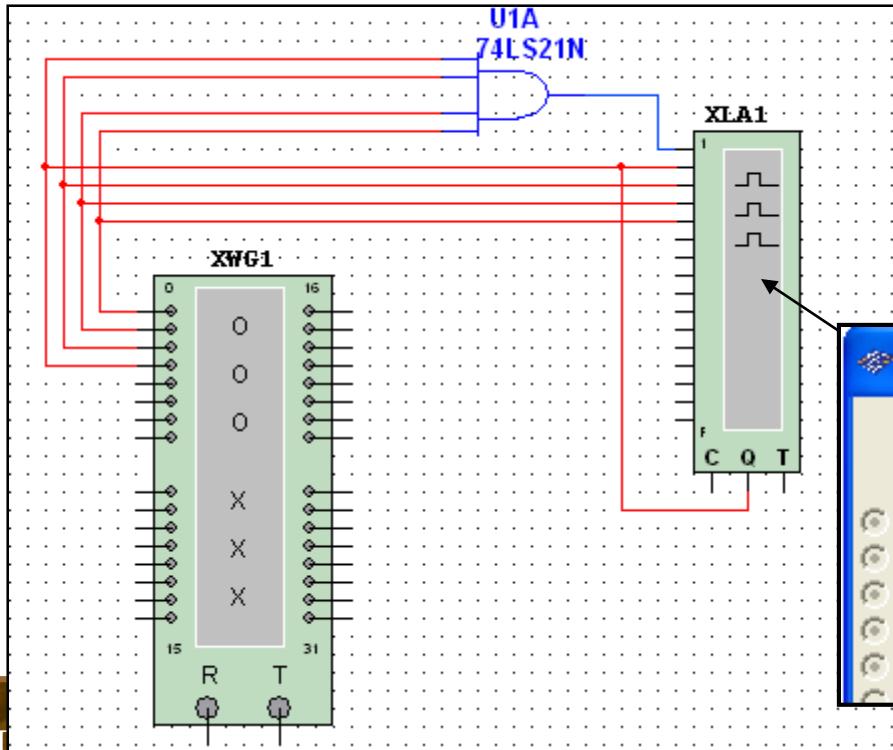
If the binary number 10100011 is ANDed with the mask 00001111, what is the result? **00000011**

Summary

The AND Gate

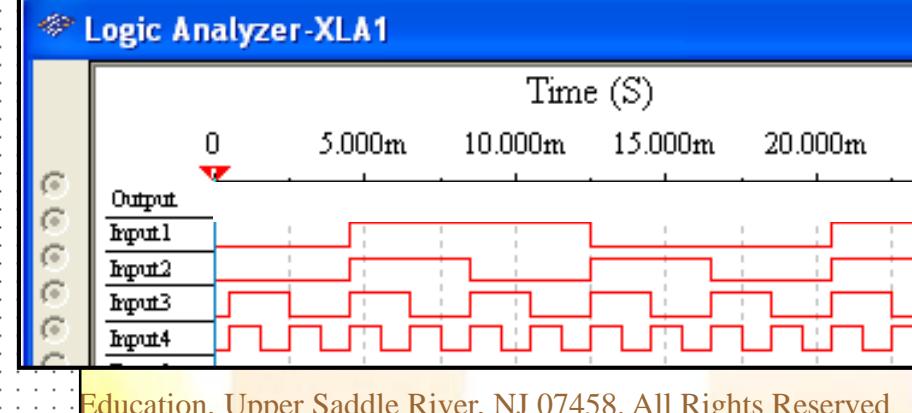
Example

A Multisim circuit is shown. XWG1 is a word generator set in the count down mode. XLA1 is a logic analyzer with the output of the AND gate connected to first (upper) line of the analyzer. What signal do you expect to see on this line?



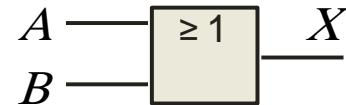
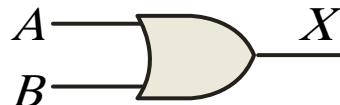
Solution

The output (line 1) will be HIGH only when all of the inputs are HIGH.



Summary

The OR Gate



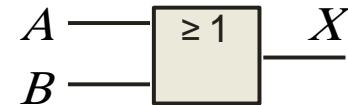
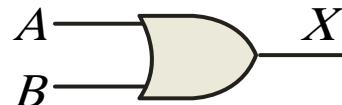
The **OR gate** produces a HIGH output if any input is HIGH; if all inputs are LOW, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

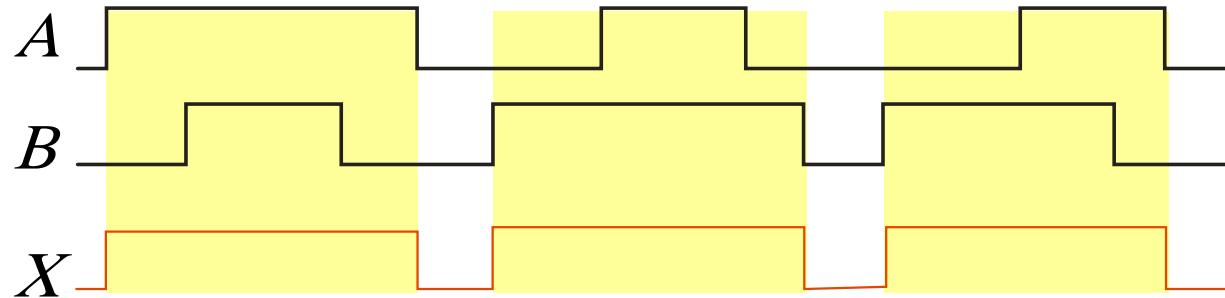
The **OR** operation is shown with a plus sign (+) between the variables. Thus, the OR operation is written as $X = A + B$.



The OR Gate



Example waveforms:



The OR operation can be used in computer programming to set certain bits of a binary number to 1.

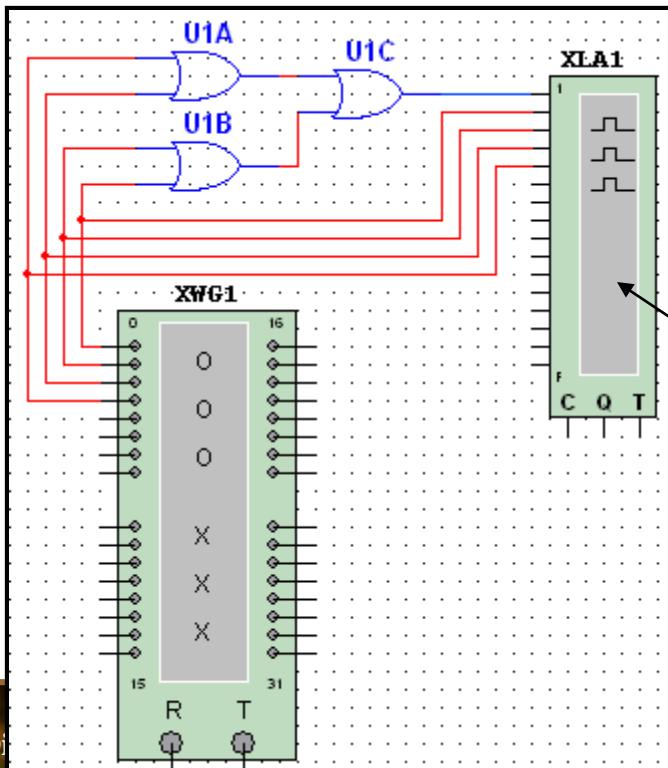
Example ASCII letters have a 1 in the bit 5 position for lower case letters and a 0 in this position for capitals. (Bit positions are numbered from right to left starting with 0.) What will be the result if you OR an ASCII letter with the 8-bit mask 00100000?

Solution The resulting letter will be lower case.

Summary

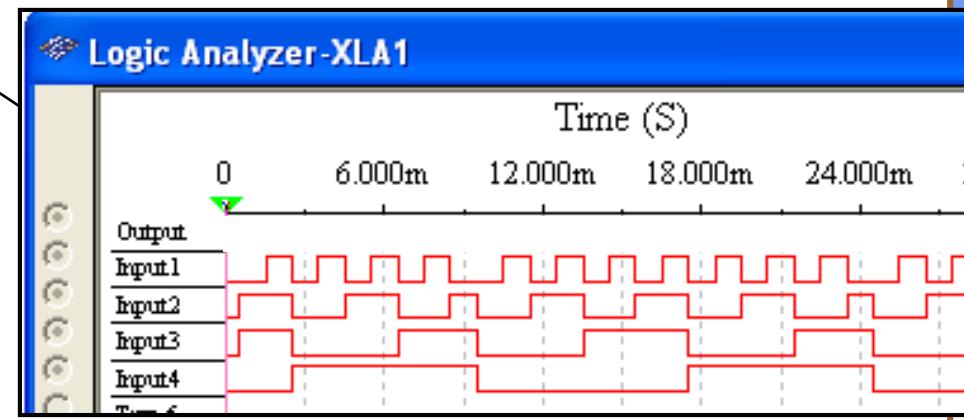
The OR Gate

Example A Multisim circuit is shown. XWG1 is a word generator set to count down. XLA1 is a logic analyzer with the output connected to first (top) line of the analyzer. The three 2-input OR gates act as a single 4-input gate. What signal do you expect on the output line?



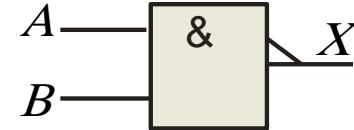
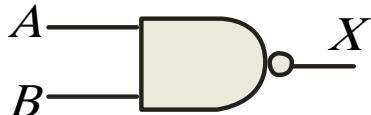
Solution

The output (line 1) will be HIGH if any input is HIGH; otherwise it will be LOW.



Summary

The NAND Gate



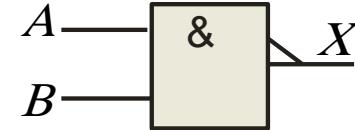
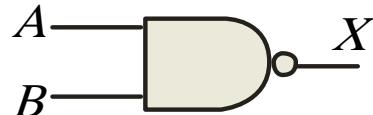
The **NAND gate** produces a LOW output when all inputs are HIGH; otherwise, the output is HIGH. For a 2-input gate, the truth table is

Inputs		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

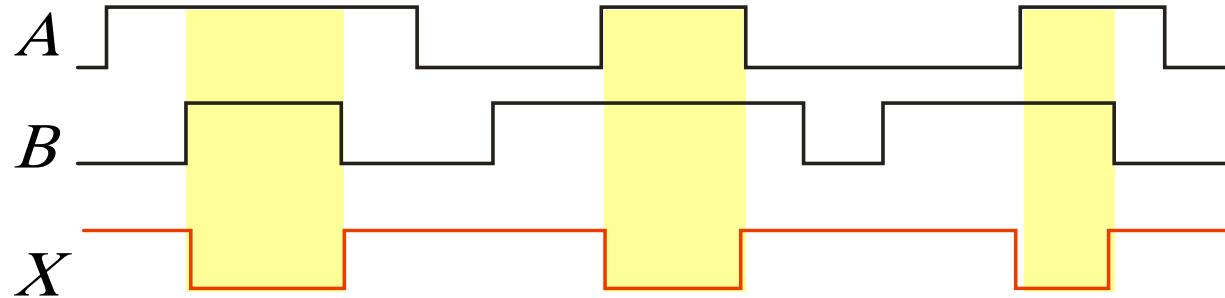
The **NAND** operation is shown with a dot between the variables and an overbar covering them. Thus, the **NAND** operation is written as $X = \overline{A \cdot B}$ (Alternatively, $X = \overline{AB}$.)

Summary

The NAND Gate

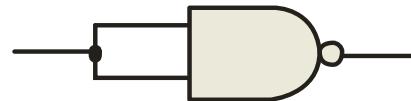


Example waveforms:



The NAND gate is particularly useful because it is a “universal” gate – all other basic gates can be constructed from NAND gates.

Question How would you connect a 2-input NAND gate to form a basic inverter?



Summary

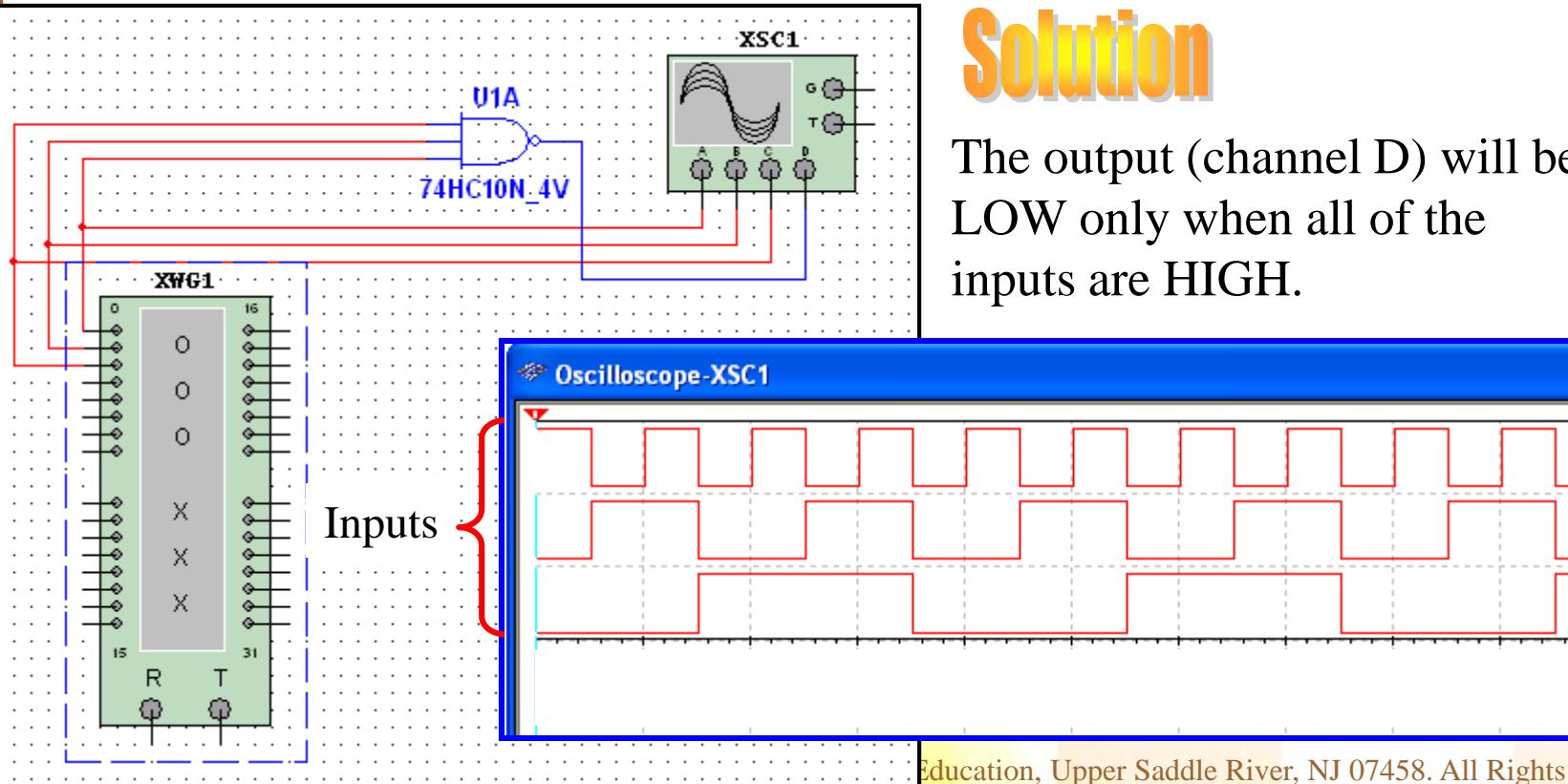
The NAND Gate

Example

A Multisim circuit is shown. XWG1 is a word generator set in the count up mode. A four-channel oscilloscope monitors the inputs and output. What output signal do you expect to see?

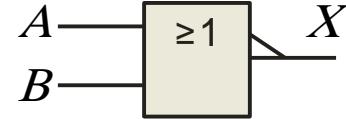
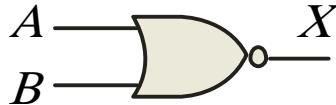
Solution

The output (channel D) will be LOW only when all of the inputs are HIGH.



Summary

The NOR Gate



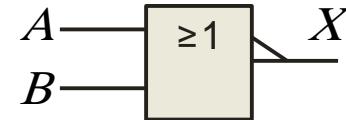
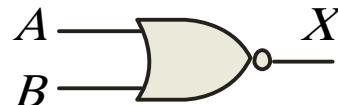
The **NOR gate** produces a LOW output if any input is HIGH; if all inputs are HIGH, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

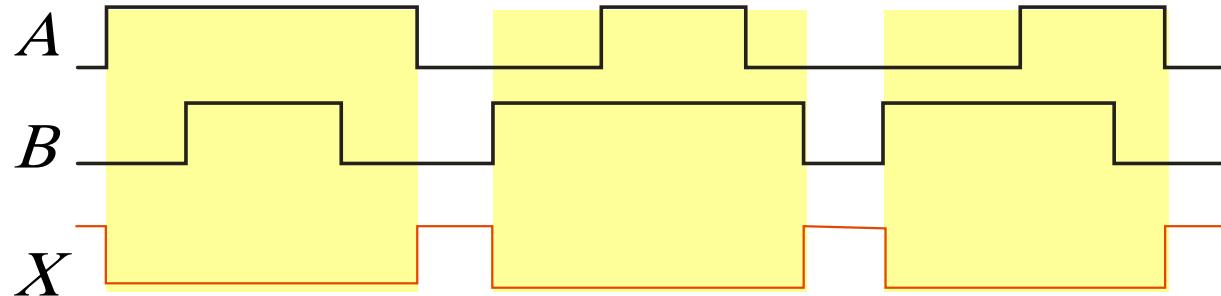
The **NOR** operation is shown with a plus sign (+) between the variables and an overbar covering them. Thus, the NOR operation is written as $X = \overline{A + B}$.

Summary

The NOR Gate



Example waveforms:



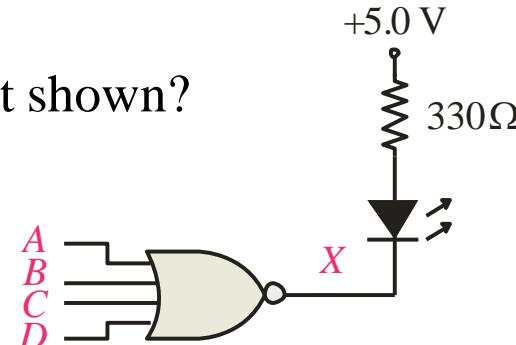
The NOR operation will produce a LOW if any input is HIGH.

Example

When is the LED is ON for the circuit shown?

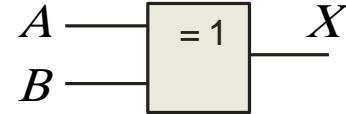
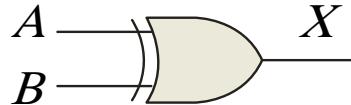
Solution

The LED will be on when any of the four inputs are HIGH.



Summary

The XOR Gate



The **XOR gate** produces a HIGH output only when both inputs are at opposite logic levels. The truth table is

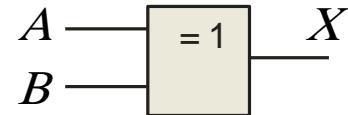
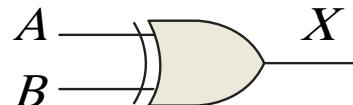
Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

The **XOR** operation is written as $X = \bar{A}B + A\bar{B}$.

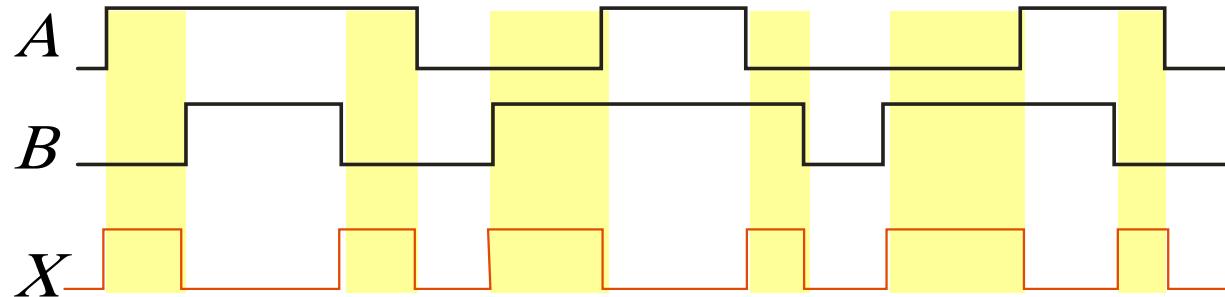
Alternatively, it can be written with a circled plus sign between the variables as $X = A \oplus B$.

Summary

The XOR Gate



Example waveforms:



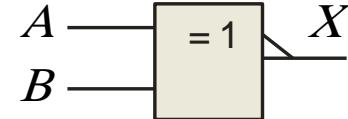
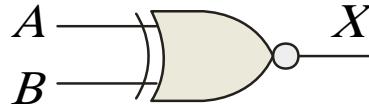
Notice that the XOR gate will produce a HIGH only when exactly one input is HIGH.

Question If the A and B waveforms are both inverted for the above waveforms, how is the output affected?

There is no change in the output.

Summary

The XNOR Gate



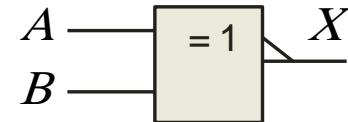
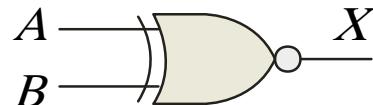
The **XNOR gate** produces a HIGH output only when both inputs are at the same logic level. The truth table is

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

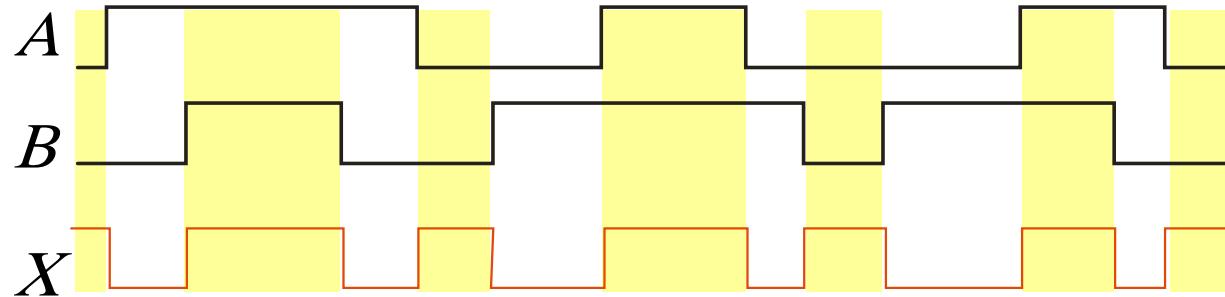
The **XNOR** operation shown as $X = \overline{A}\overline{B} + AB$. Alternatively, the XNOR operation can be shown with a circled dot between the variables. Thus, it can be shown as $X = A \odot B$.



The XNOR Gate



Example waveforms:



Notice that the XNOR gate will produce a HIGH when both inputs are the same. This makes it useful for comparison functions.

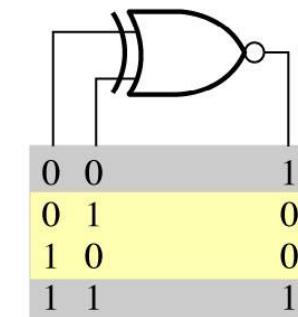
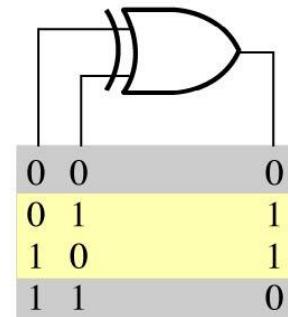
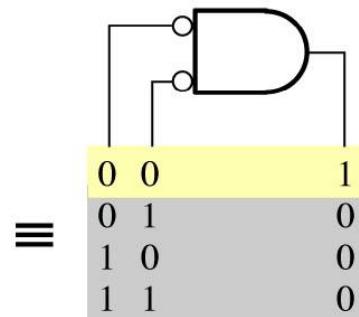
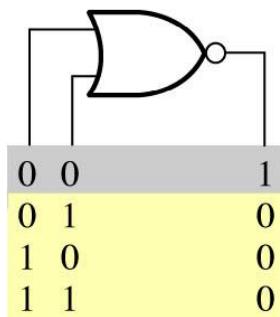
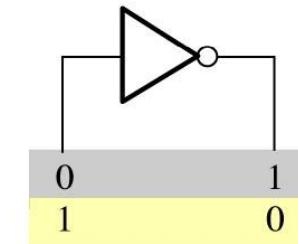
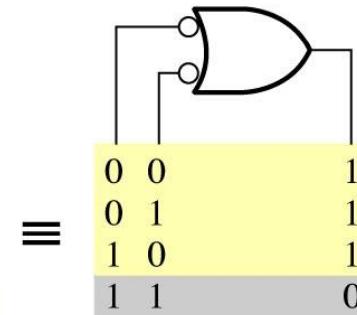
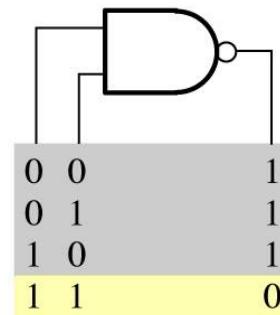
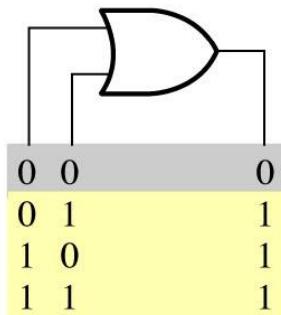
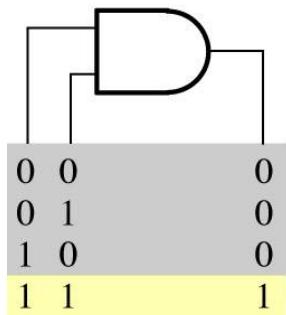
Question If the A waveform is inverted but B remains the same, how is the output affected?

The output will be inverted.



Summary

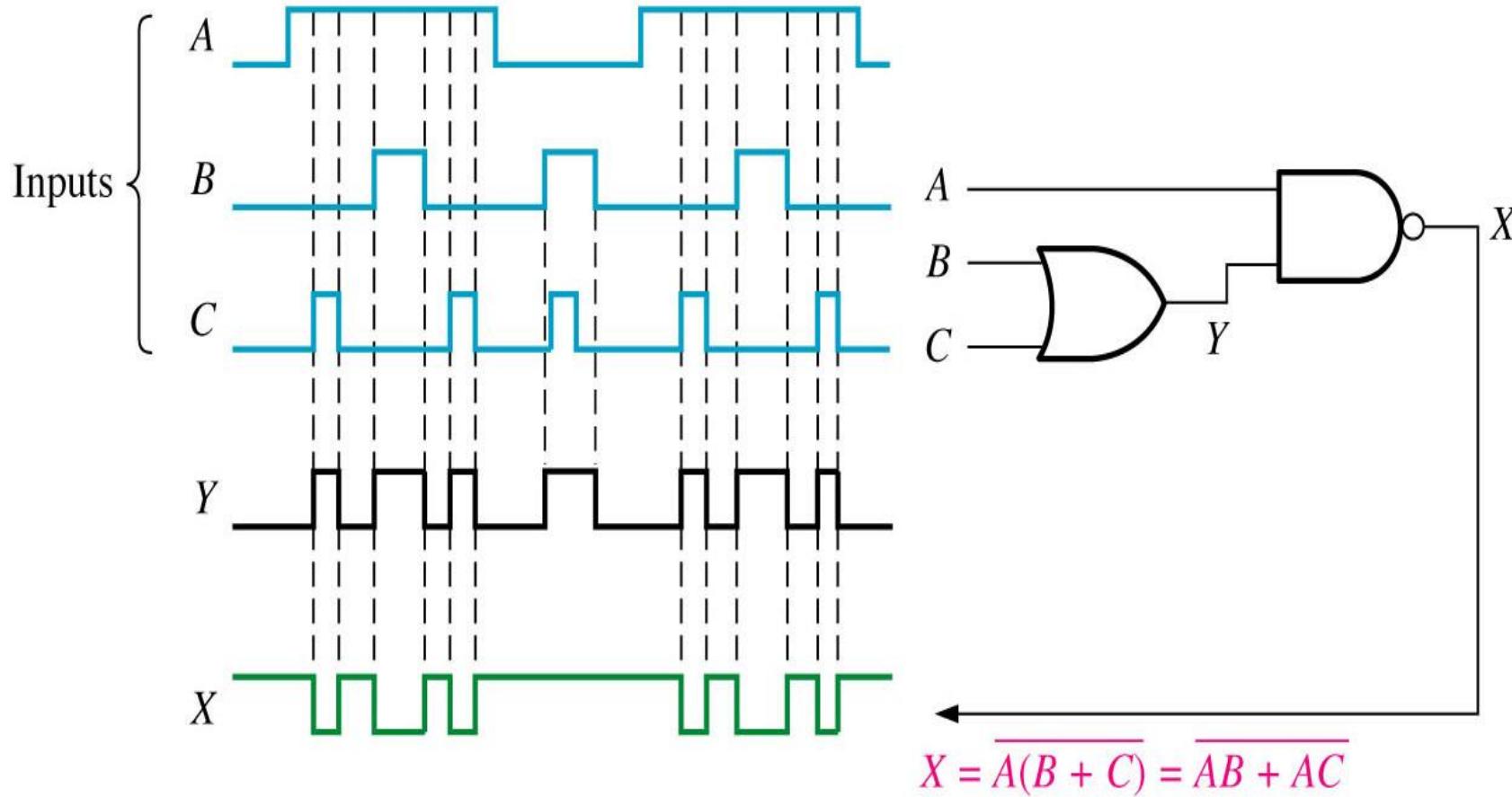
Logic Gate Summary



Note: Active states are shown in yellow.

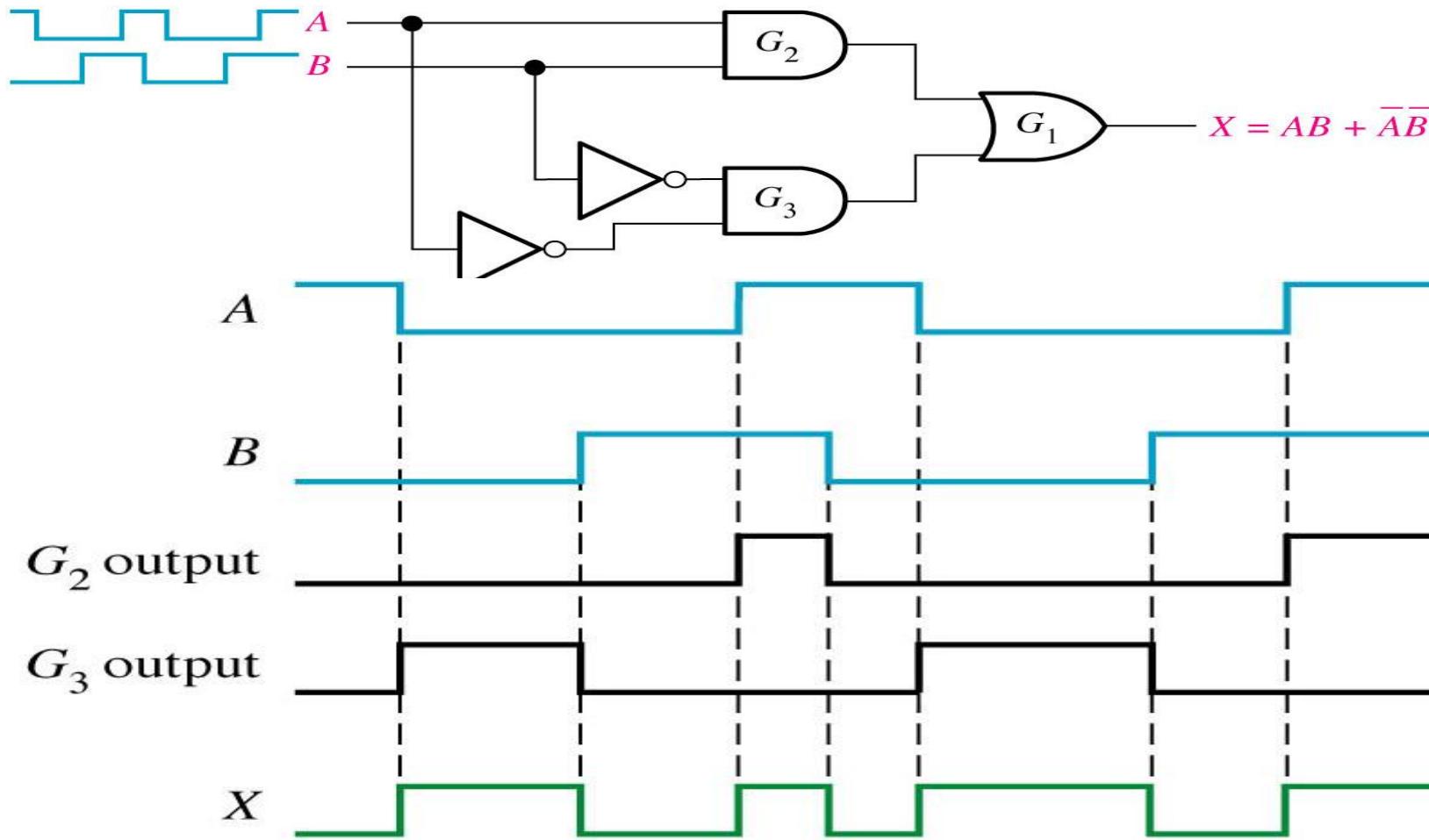
Summary

Logic Circuit Operation w/ Pulse Waveforms



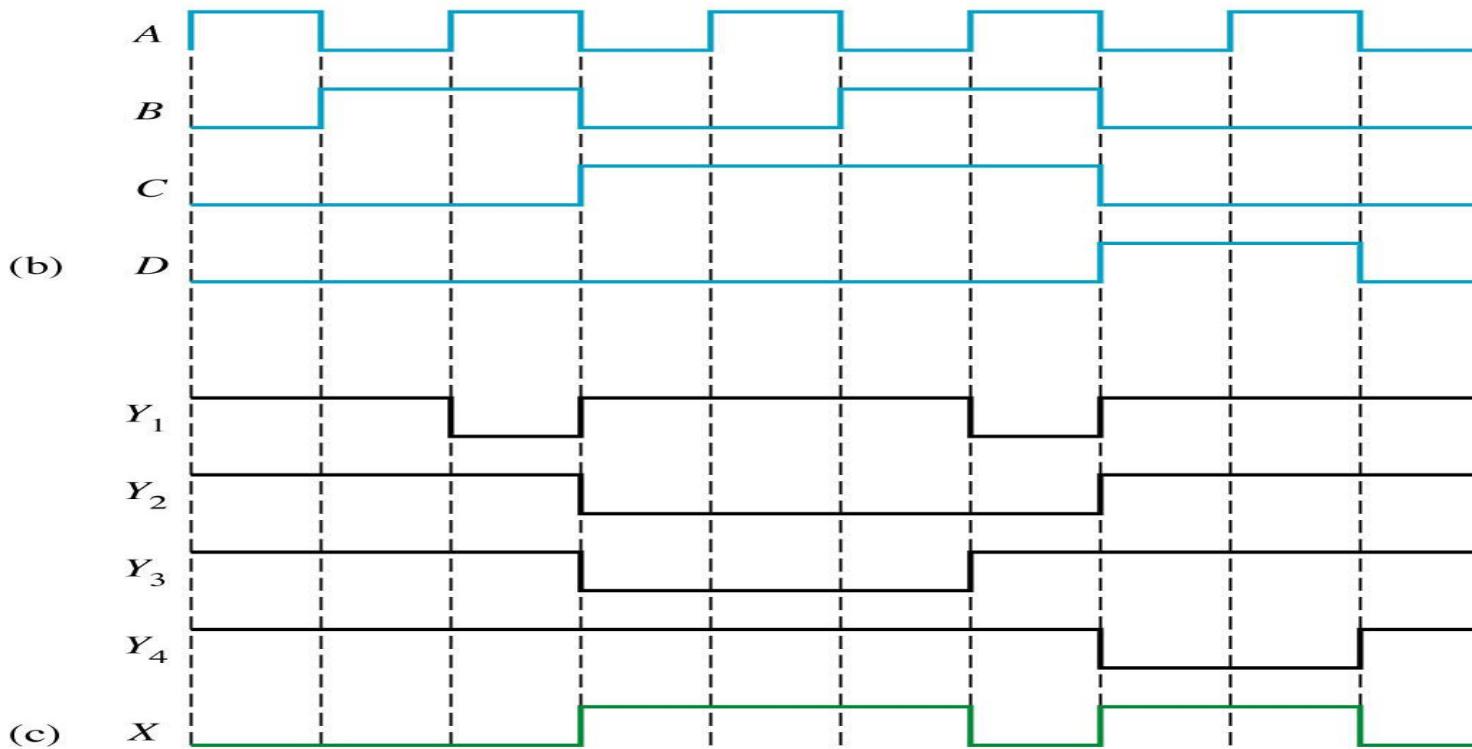
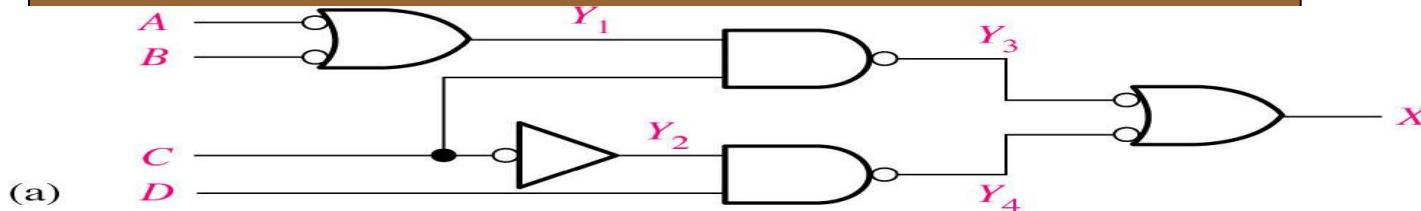
Summary

Example: Draw timing diagram for this circuit.



Summary

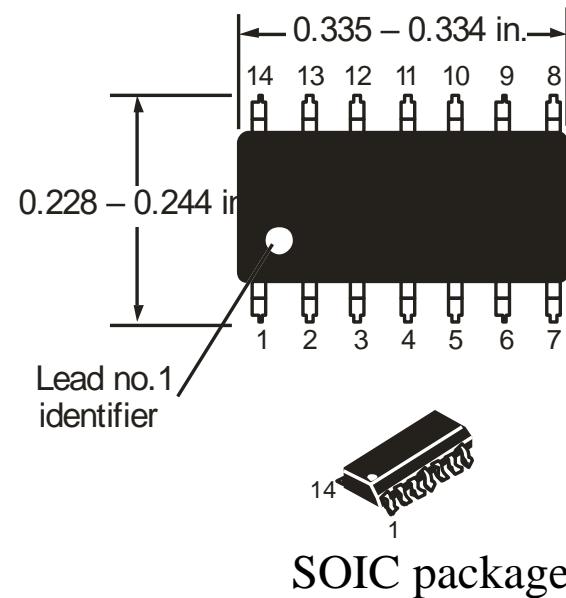
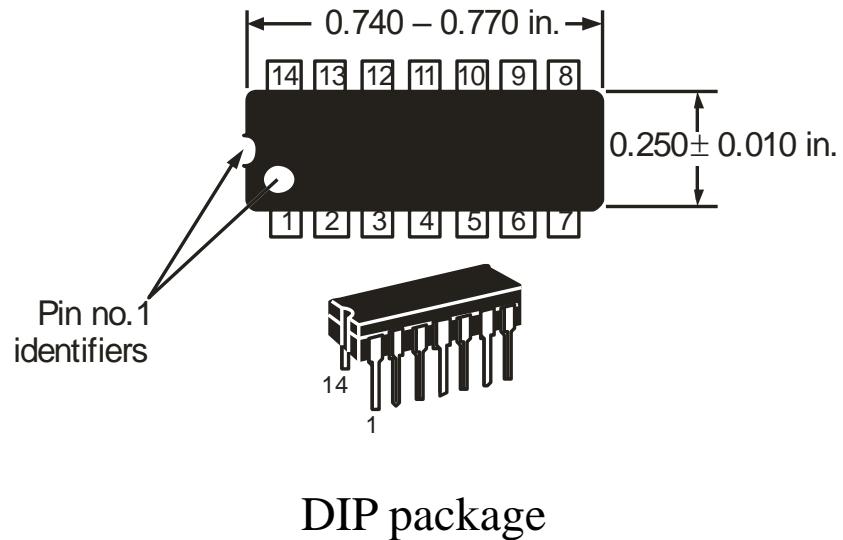
Example: Draw timing diagram for this circuit.



Summary

Fixed Function Logic

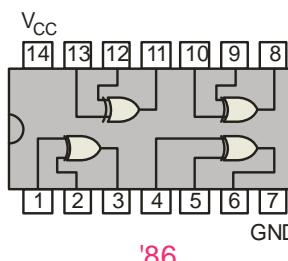
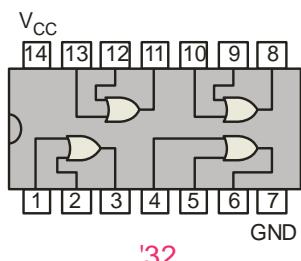
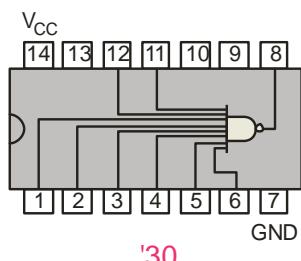
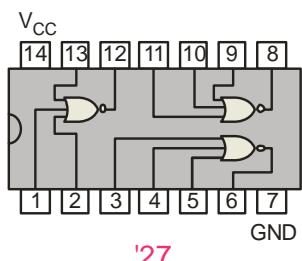
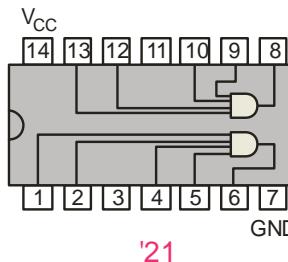
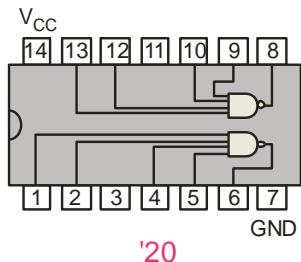
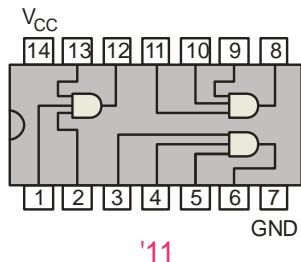
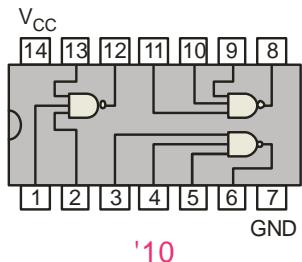
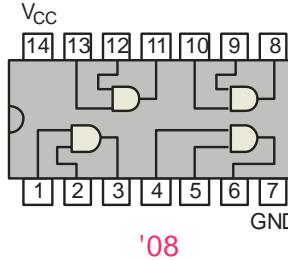
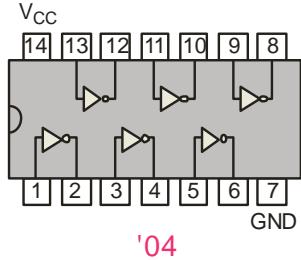
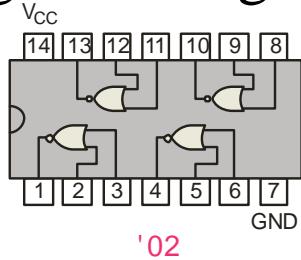
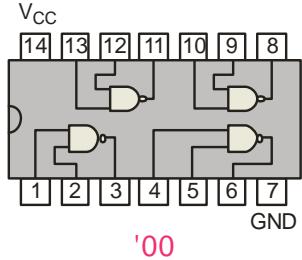
Two major fixed function logic families are TTL and CMOS. A third technology is BiCMOS, which combines the first two. Packaging for fixed function logic is shown.





Fixed Function Logic

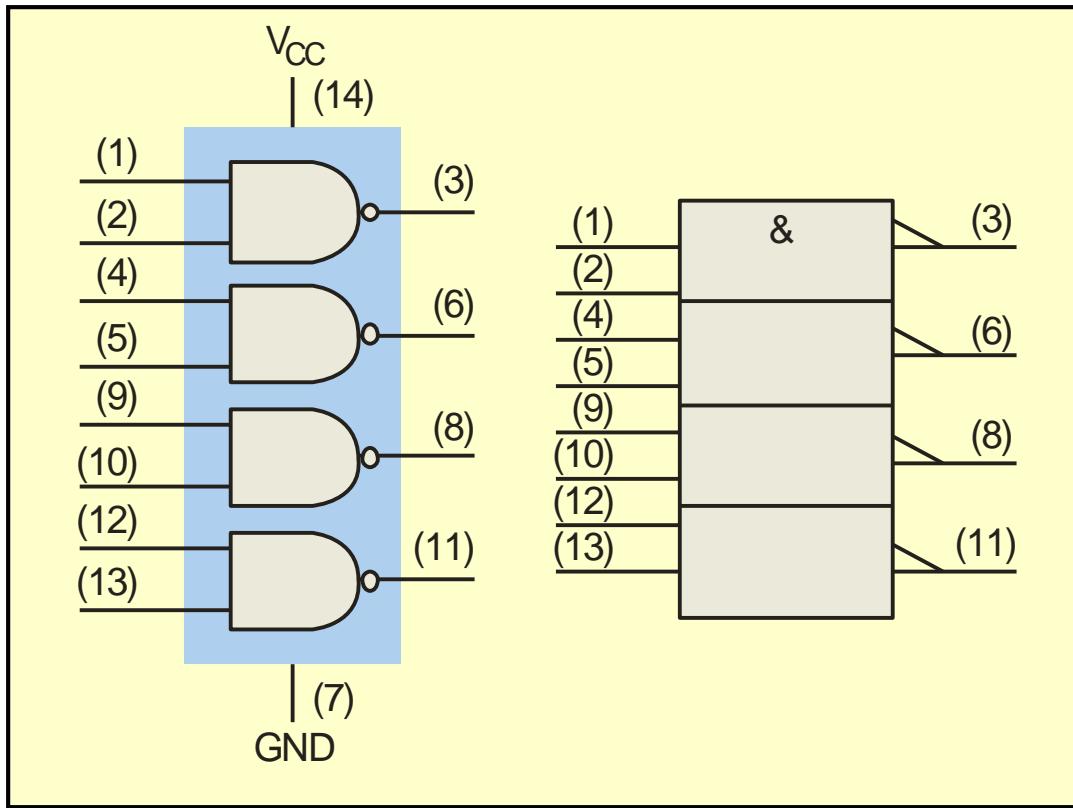
Some common gate configurations are shown.



Summary

Fixed Function Logic

Logic symbols show the gates and associated pin numbers.



Summary

Fixed Function Logic

Data sheets include limits and conditions set by the manufacturer as well as DC and AC characteristics. For example, some maximum ratings for a 74HC00A are:

MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	-0.5 to + 7.0 V	V
V_{in}	DC Input Voltage (Referenced to GND)	-0.5 to V_{CC} +0.5 V	V
V_{out}	DC Output Voltage (Referenced to GND)	-0.5 to V_{CC} +0.5 V	V
I_{in}	DC Input Current, per pin	± 20	mA
I_{out}	DC Output Current, per pin	± 25	mA
I_{CC}	DC Supply Current, V_{CC} and GND pins	± 50	mA
P_D	Power Dissipation in Still Air, Plastic or Ceramic DIP SOIC Package TSSOP Package	750 500 450	mW
T_{stg}	Storage Temperature	-65 to + 150	°C
T_L	Lead Temperature, 1 mm from Case for 10 Seconds Plastic DIP, SOIC, or TSSOP Package Ceramic DIP	260 300	°C

Selected Key Terms

Inverter A logic circuit that inverts or complements its inputs.

Truth table A table showing the inputs and corresponding output(s) of a logic circuit.

Timing diagram A diagram of waveforms showing the proper time relationship of all of the waveforms.

Boolean algebra The mathematics of logic circuits.

AND gate A logic gate that produces a HIGH output only when all of its inputs are HIGH.

Selected Key Terms

OR gate A logic gate that produces a HIGH output when one or more inputs are HIGH.

NAND gate A logic gate that produces a LOW output only when all of its inputs are HIGH.

NOR gate A logic gate that produces a LOW output when one or more inputs are HIGH.

Exclusive-OR gate A logic gate that produces a HIGH output only when its two inputs are at opposite levels.

Exclusive-NOR gate A logic gate that produces a LOW output only when its two inputs are at opposite levels.

Quiz

1. The truth table for a 2-input AND gate is

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

a.

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

b.

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

c.

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

d.

Quiz

2. The truth table for a 2-input NOR gate is

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

a.

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

b.

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

c.

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

d.

Quiz

3. The truth table for a 2-input XOR gate is

a.

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

b.

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

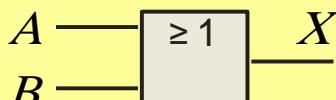
c.

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

d.

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Quiz

4. The symbol  is for a(n)
- a. OR gate
 - b. AND gate
 - c. NOR gate
 - d. XOR gate

Quiz

5. The symbol  is for a(n)

- a. OR gate
- b. NOR gate
- c. XOR gate
- d. XNOR gate

Quiz

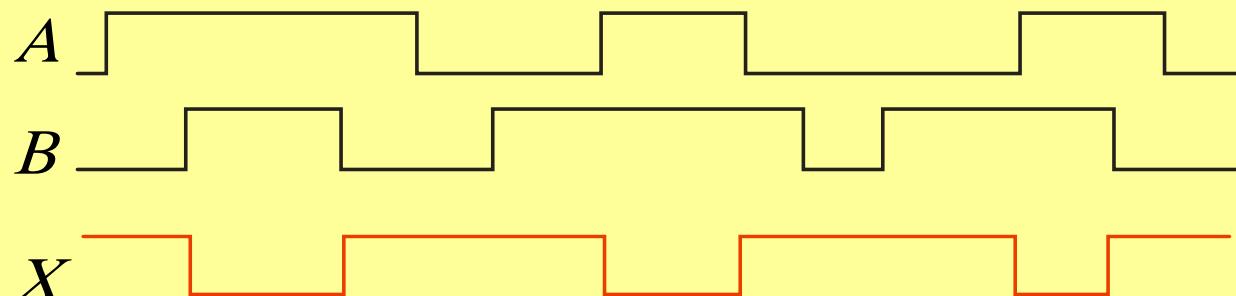
6. A logic gate that produces a HIGH output only when all of its inputs are HIGH is a(n)
- a. OR gate
 - b. AND gate
 - c. NOR gate
 - d. NAND gate

Quiz

7. The expression $X = A \oplus B$ means
- a. A OR B
 - b. A AND B
 - c. A XOR B
 - d. A XNOR B

Quiz

8. A 2-input gate produces the output shown. (X represents the output.) This is a(n)
- a. OR gate
 - b. AND gate
 - c. NOR gate
 - d. NAND gate



Quiz

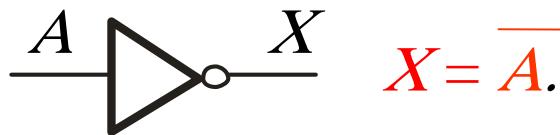
9. A 2-input gate produces a HIGH output only when the inputs agree. This type of gate is a(n)
- a. OR gate
 - b. AND gate
 - c. NOR gate
 - d. XNOR gate

Chapter 4

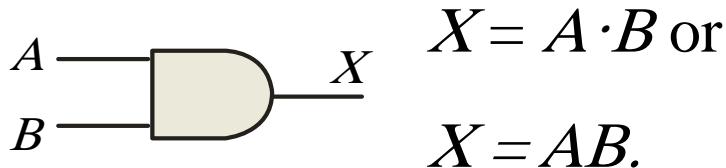
Summary

Boolean Algebra

Boolean algebra is the mathematics of digital logic. A basic knowledge of Boolean algebra is indispensable to the study and analysis of logic circuits.

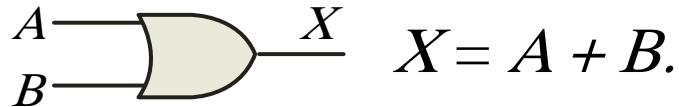


$$X = \overline{A}.$$

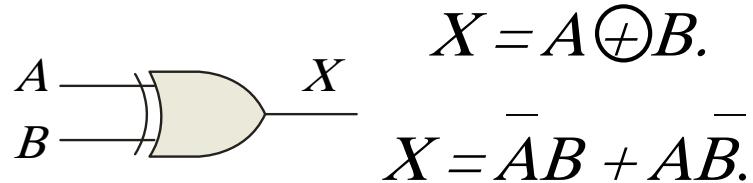


$$X = A \cdot B \text{ or}$$

$$X = AB.$$

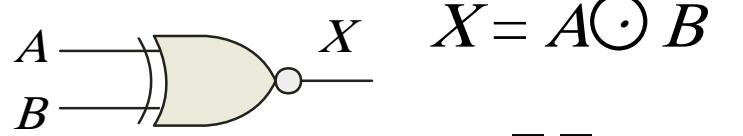


$$X = A + B.$$



$$X = A \oplus B.$$

$$X = \overline{\overline{AB}} + \overline{AB}.$$

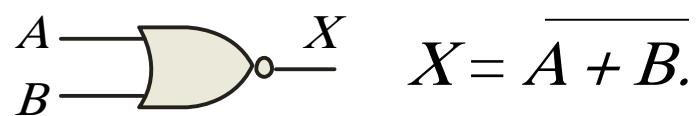
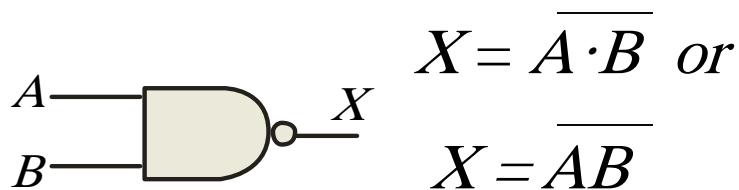


$$X = A \odot B$$

$$X = \overline{\overline{AB}} + \overline{AB}.$$

Summary

Boolean Algebra



In Boolean algebra, a **variable** is a symbol used to represent an action, a condition, or data. A single variable can only have a value of 1 or 0.

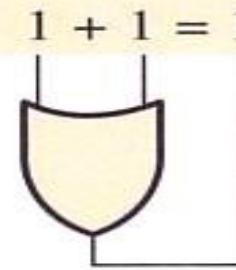
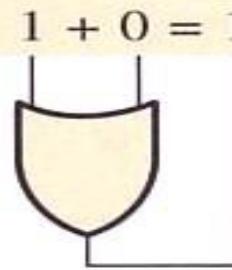
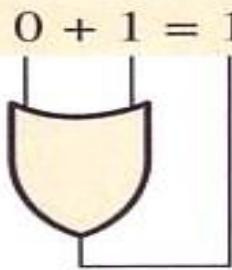
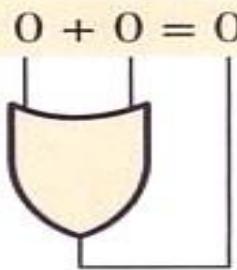
The **complement** represents the inverse of a variable and is indicated with an overbar. Thus, the complement of A is \overline{A} .

A **literal** is a variable or its complement.

Summary

Boolean Addition

Addition is equivalent to the OR operation. The sum term is 1 if one or more of the literals are 1. The sum term is zero only if each literal is 0.



Example

Determine the values of A , B , and C that make the sum term of the expression $\overline{A} + B + \overline{C} = 0$?

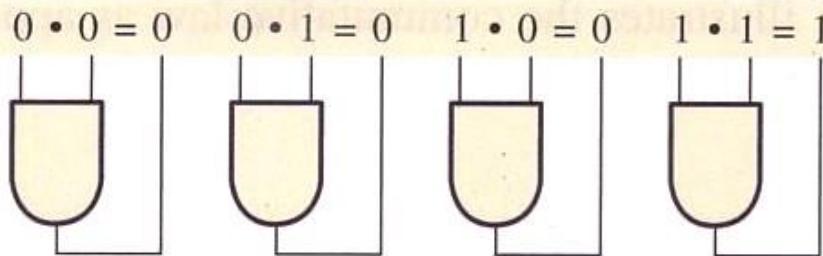
Solution

Each literal must = 0; therefore $A = 1$, $B = 0$ and $C = 1$.

Summary

Boolean Multiplication

In Boolean algebra, multiplication is equivalent to the AND operation. The product of literals forms a product term. The product term will be 1 only if all of the literals are 1.



Example

What are the values of the A , B and C if the product term of $A \cdot \bar{B} \cdot \bar{C} = 1$?

Solution

Each literal must = 1; therefore $A = 1$, $B = 0$ and $C = 0$.



Summary

Laws and Rules of Boolean Algebra

❑ Laws:

- **Commutative Laws**
- **Associative Laws**
- **Distributive Law**

❑ Rules

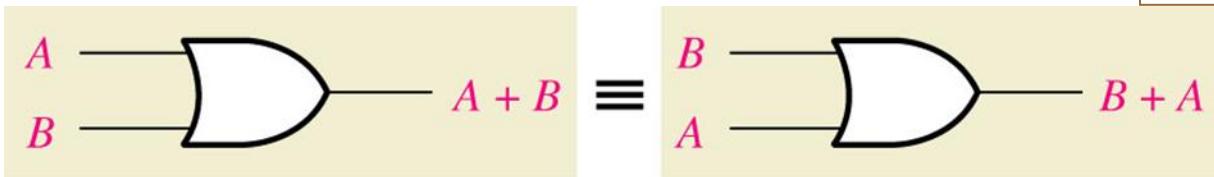
Summary

Commutative Laws

The **commutative laws** are applied to **addition** and **multiplication**. For addition, the commutative law states

In terms of the result, the order in which variables are ORed makes no difference.

$$A + B = B + A$$



For multiplication, the commutative law states

In terms of the result, the order in which variables are ANDed makes no difference.

$$AB = BA$$



Summary

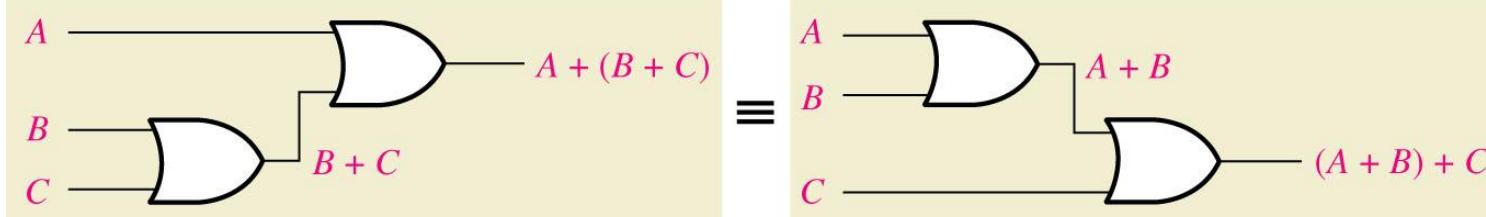
Associative Laws

The **associative laws** are also applied to **addition** and **multiplication**.

For addition, the associative law states

When ORing more than two variables, the result is the same regardless of the grouping of the variables.

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



Application of associative law of addition

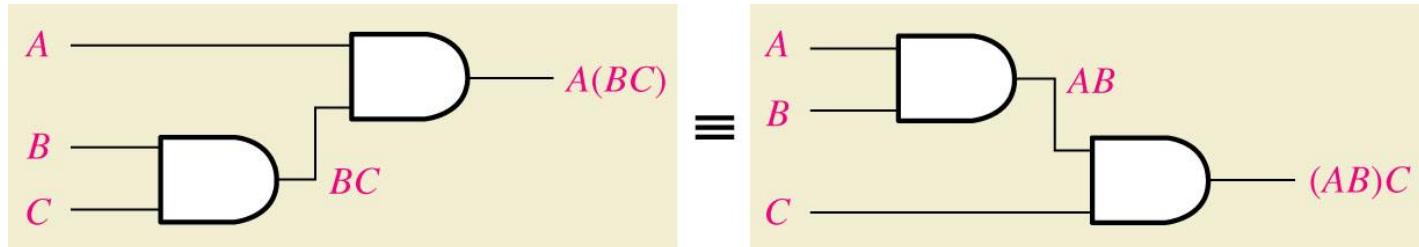
Summary

Associative Laws

For multiplication, the associative law states

When ANDing more than two variables, the result is the same regardless of the grouping of the variables.

$$A(BC) = (AB)C$$



Application of associative law of multiplication

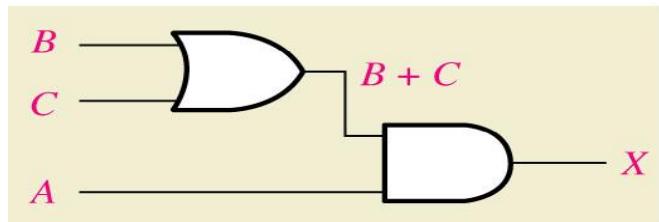
Summary

Distributive Law

The **distributive law** is the factoring law. A common variable can be factored from an expression just as in ordinary algebra. That is

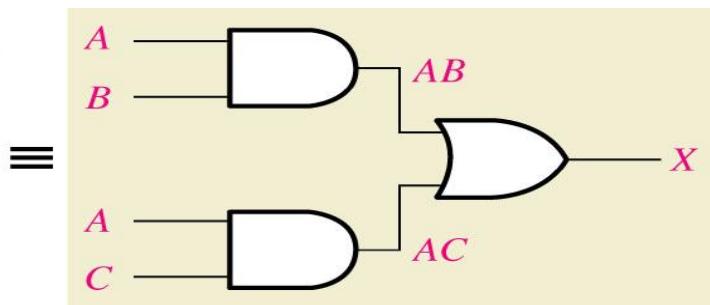
$$AB + AC = A(B + C)$$

The distributive law can be illustrated with equivalent circuits:



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

Application of **distributive law**



$$X = AB + AC$$



Summary

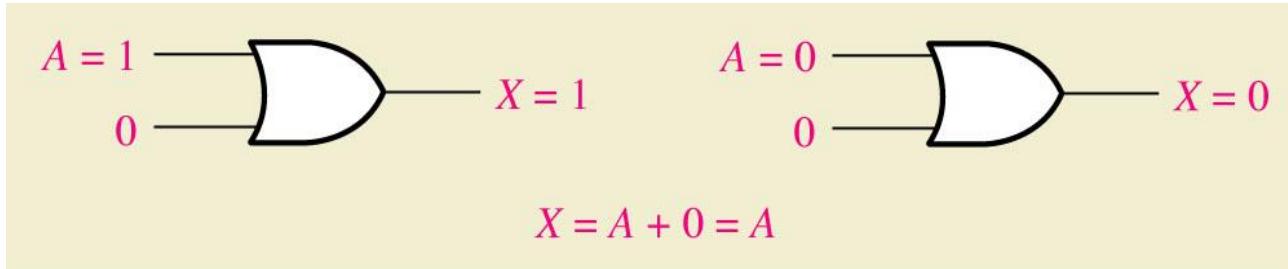
Rules of Boolean Algebra

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = A$
5. $A + A = A$
6. $A + \bar{A} = 1$
7. $A \cdot A = A$
8. $A \cdot \bar{A} = 0$
9. $\bar{\bar{A}} = A$
10. $A + AB = A$
11. $A + \bar{A}B = A + B$
12. $(A + B)(A + C) = A + BC$

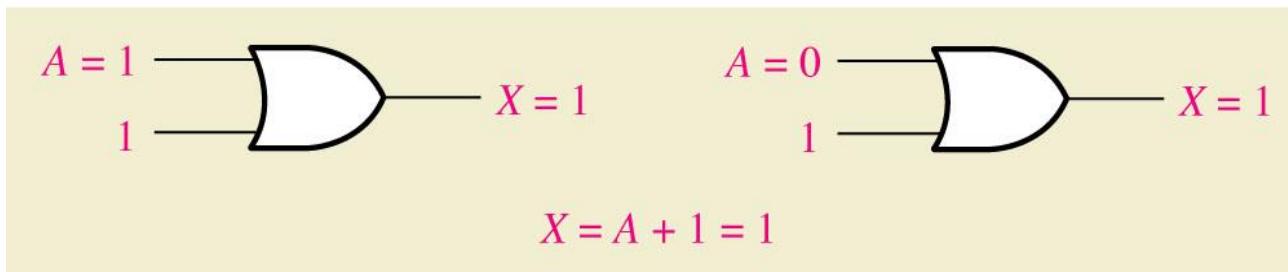
Summary

Rules of Boolean Algebra

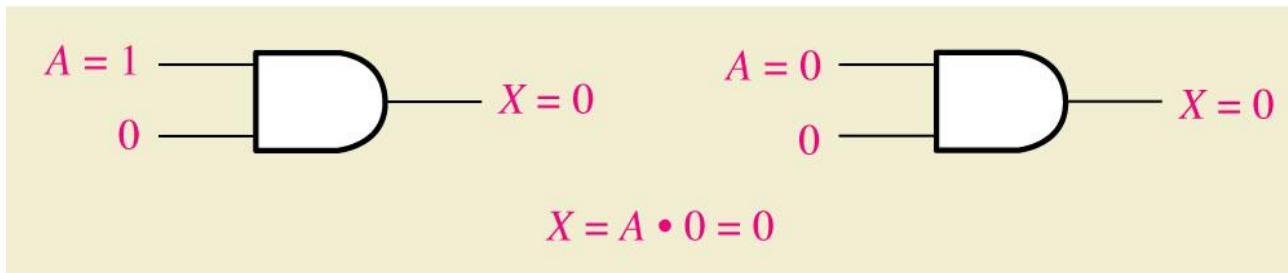
$$1. A + 0 = A$$



$$2. A + 1 = 1$$



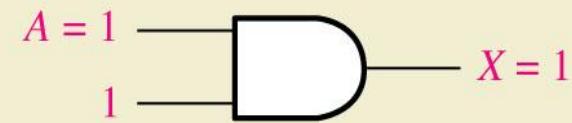
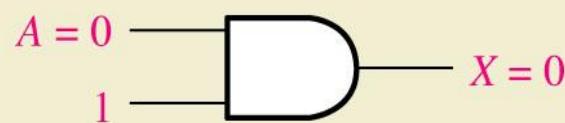
$$3. A \cdot 0 = 0$$



Summary

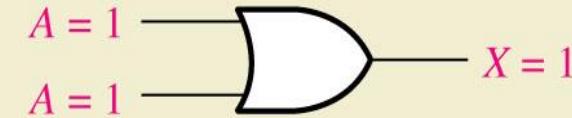
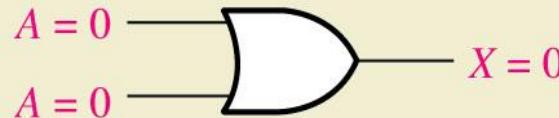
Rules of Boolean Algebra

$$4. A \cdot 1 = A$$



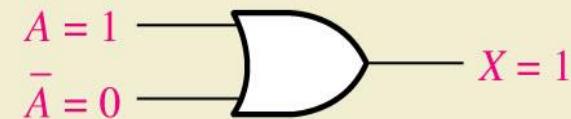
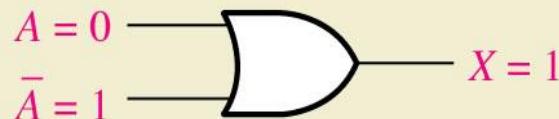
$$X = A \cdot 1 = A$$

$$5. A + A = A$$



$$X = A + A = A$$

$$6. A + \bar{A} = 1$$

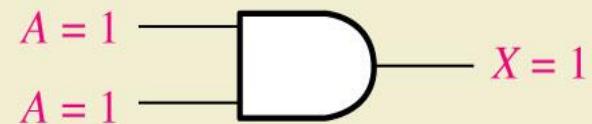
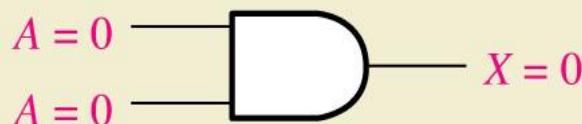


$$X = A + \bar{A} = 1$$

Summary

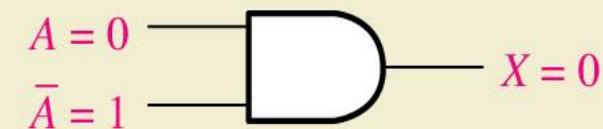
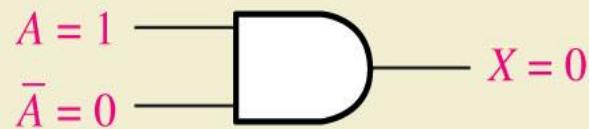
Rules of Boolean Algebra

$$7. A \cdot A = A$$



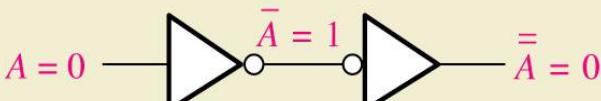
$$X = A \cdot A = A$$

$$8. A \cdot \bar{A} = 0$$

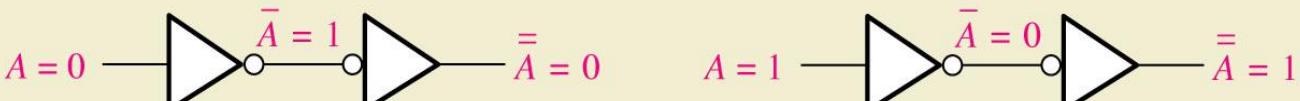


$$X = A \cdot \bar{A} = 0$$

$$9. \bar{\bar{A}} = A$$



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



Summary

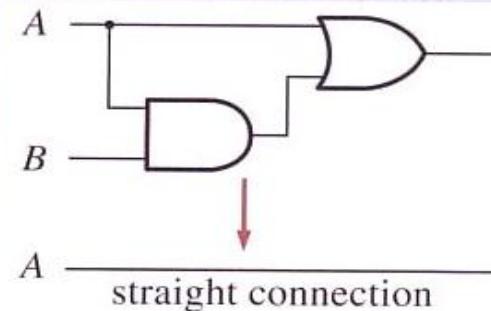
Rules of Boolean Algebra

Rule 10. $A + AB = A$ This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

$$\begin{aligned}A + AB &= A(1 + B) && \text{Factoring (distributive law)} \\&= A \cdot 1 && \text{Rule 2: } (1 + B) = 1 \\&= A && \text{Rule 4: } A \cdot 1 = A\end{aligned}$$

A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

↑ ↑
equal



► TABLE 4-2

Rule 10: $A + AB = A$.



Summary

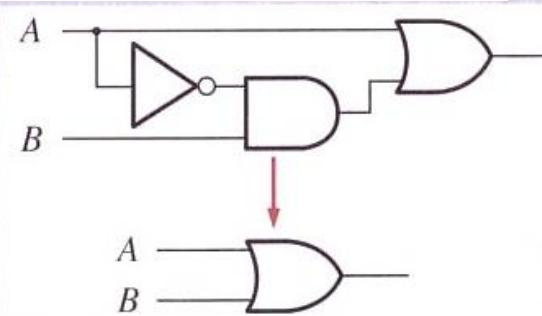
Rules of Boolean Algebra

Rule 11. $A + \bar{A}B = A + B$ This rule can be proved as follows:

$$\begin{aligned}
 A + \bar{A}B &= (A + AB) + \bar{A}B && \text{Rule 10: } A = A + AB \\
 &= (AA + AB) + \bar{A}B && \text{Rule 7: } A = AA \\
 &= AA + AB + A\bar{A} + \bar{A}B && \text{Rule 8: adding } A\bar{A} = 0 \\
 &= (A + \bar{A})(A + B) && \text{Factoring} \\
 &= 1 \cdot (A + B) && \text{Rule 6: } A + \bar{A} = 1 \\
 &= A + B && \text{Rule 4: drop the 1}
 \end{aligned}$$

A	B	$\bar{A}B$	$A + \bar{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

↑ equal ↑



► TABLE 4-3

Rule 11: $A + \bar{A}B = A + B$.



Summary

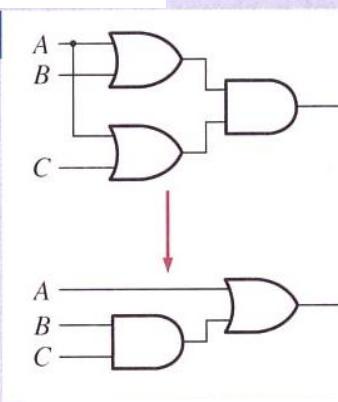
Rules of Boolean Algebra

Rule 12. $(A + B)(A + C) = A + BC$ This rule can be proved as follows:

$$\begin{aligned}
 (A + B)(A + C) &= AA + AC + AB + BC && \text{Distributive law} \\
 &= A + AC + AB + BC && \text{Rule 7: } AA = A \\
 &= A(1 + C) + AB + BC && \text{Factoring (distributive law)} \\
 &= A \cdot 1 + AB + BC && \text{Rule 2: } 1 + C = 1 \\
 &= A(1 + B) + BC && \text{Factoring (distributive law)} \\
 &= A \cdot 1 + BC && \text{Rule 2: } 1 + B = 1 \\
 &= A + BC && \text{Rule 4: } A \cdot 1 = A
 \end{aligned}$$

A	B	C	$A + B$	$A + C$	$(A + B)(A + C)$	BC	$A + BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

↑ equal ↑



▼ TABLE 4-4

Rule 12: $(A + B)(A + C) = A + BC$.



Summary

Rules of Boolean Algebra

Rule 12, which states that $(A + B)(A + C) = A + BC$, can be proven by applying earlier rules as follows:

$$\begin{aligned}(A + B)(A + C) &= AA + AC + AB + BC \\&= A + AC + AB + BC \\&= A(1 + C + B) + BC \\&= A \cdot 1 + BC \\&= A + BC\end{aligned}$$

This rule is a little more complicated, but it can also be shown with a Venn diagram, as given on the following slide...

Summary

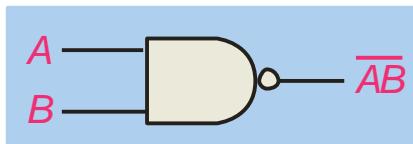
DeMorgan's Theorem

DeMorgan's 1st Theorem

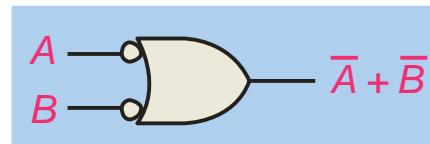
The complement of a product of variables is equal to the sum of the complemented variables.

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

Applying DeMorgan's first theorem to gates:



NAND



Negative-OR

Inputs		Output	
A	B	\overline{AB}	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Summary

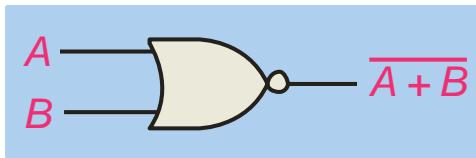
DeMorgan's Theorem

DeMorgan's 2nd Theorem

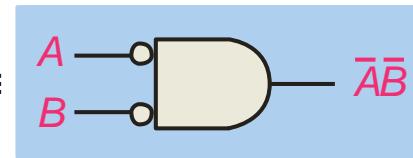
The complement of a sum of variables is equal to the product of the complemented variables.

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

Applying DeMorgan's second theorem to gates:



NOR



Negative-AND

Inputs		Output	
A	B	$\overline{A + B}$	\overline{AB}
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0



Summary

DeMorgan's Theorem

Example

Apply DeMorgan's theorem to remove the overbar covering both terms from the expression $X = \overline{\overline{C} + D}$.

Solution

To apply DeMorgan's theorem to the expression, you can break the overbar covering both terms and change the sign between the terms. This results in $X = \overline{\overline{C}} \cdot \overline{\overline{D}}$. Deleting the double bar gives $X = C \cdot \overline{D}$.

Summary

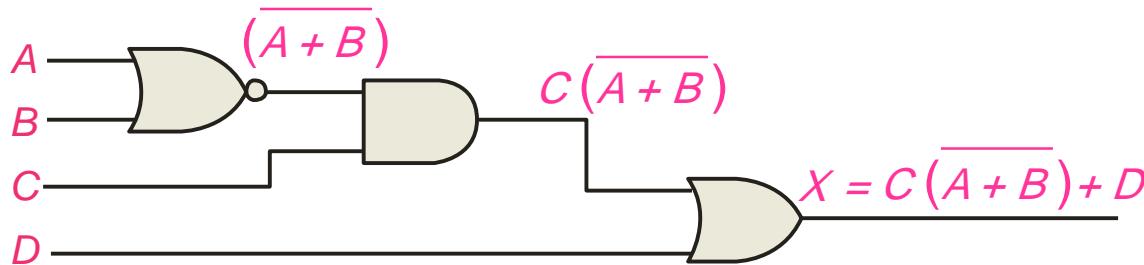
Boolean Analysis of Logic Circuits

Combinational logic circuits can be analyzed by writing the expression for each gate and combining the expressions according to the rules for Boolean algebra.

Example Solution

Apply Boolean algebra to derive the expression for X .

Write the expression for each gate:



Applying DeMorgan's theorem and the distribution law:

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

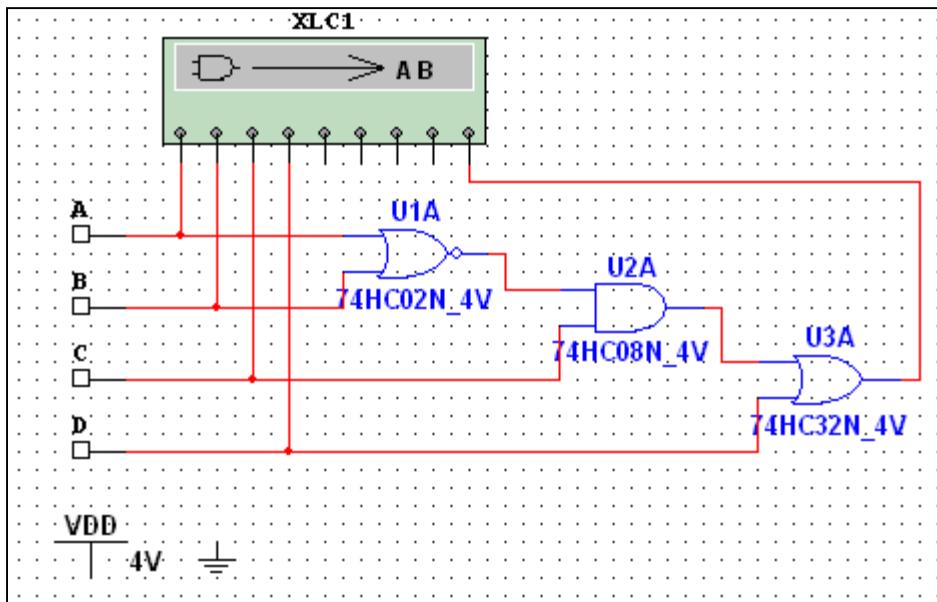
Summary

Boolean Analysis of Logic Circuits

Example Solution

Use Multisim to generate the truth table for the circuit in the previous example.

Set up the circuit using the Logic Converter as shown. (Note that the logic converter has no “real-world” counterpart.)



Double-click the Logic Converter top open it.
Then click on the conversion bar on the right side to see the truth table for the circuit
(see next slide).



Summary

Boolean Analysis of Logic Circuits

The simplified logic expression can be viewed by clicking

The screenshot shows the Logic Converter-XLC1 application window. On the left is a truth table with columns A through H and an output column labeled 'Out'. The rows show binary values from 000 to 111. On the right is a 'Conversions' panel with several options:

- $\oplus \rightarrow 101$
- $101 \rightarrow A \oplus B$
- $101 \xrightarrow{\text{SIMP}} A \oplus B$ (highlighted)
- $A \oplus B \rightarrow 101$
- $A \oplus B \rightarrow \oplus$
- $A \oplus B \rightarrow \text{NAND}$

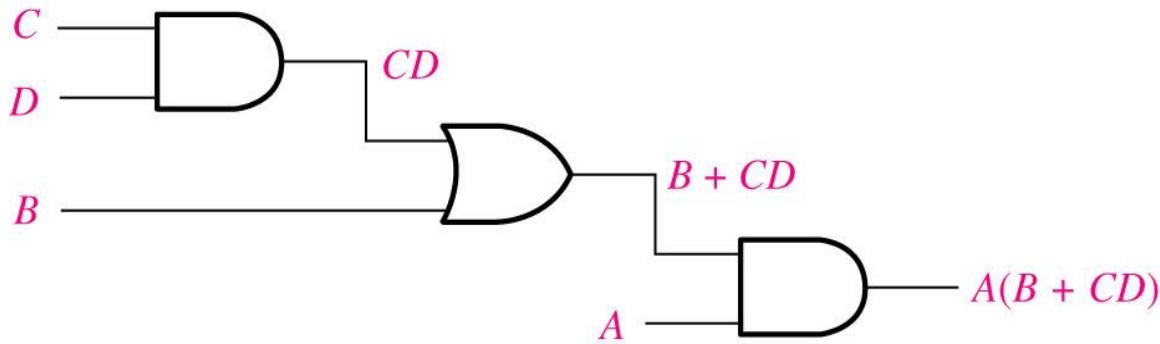
An arrow points from the text "Simplified expression" to the simplified expression field at the bottom of the window, which contains the text $A'B'C + D$.

	A	B	C	D	E	F	G	H	Out
000	0	0	0	0					0
001	0	0	0	1					1
002	0	0	1	0					1
003	0	0	1	1					1
004	0	1	0	0					0
005	0	1	0	1					1
006	0	1	1	0					0
007	0	1	1	1					1
008	1	0	0	0					0
009	1	0	0	1					1
010	1	0	1	0					0
011	1	0	1	1					1
012	1	1	0	0					0
013	1	1	0	1					1
014	1	1	1	0					0
015	1	1	1	1					1

Summary

Boolean Expression for a Logic Circuit

Constructing a Truth Table for a Logic Circuit



Evaluating the expression and putting results
in truth table format

INPUTS				OUTPUT
A	B	C	D	$A(B + CD)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

► TABLE 4-5

Truth table for the logic circuit



Summary

Simplification Using Boolean Algebra

EXAMPLE 4-8

Using Boolean algebra techniques, simplify this expression:

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

Solution The following is not necessarily the only approach.

Step 1. Apply the distributive law to the second and third terms in the expression, as follows:

$$AB + AB + AC + BB + BC$$

Step 2. Apply rule 7 ($BB = B$) to the fourth term.

$$AB + AB + AC + B + BC$$

Step 3. Apply rule 5 ($AB + AB = AB$) to the first two terms.

$$AB + AC + B + BC$$

Step 4. Apply rule 10 ($B + BC = B$) to the last two terms.

$$AB + AC + B$$

Step 5. Apply rule 10 ($AB + B = B$) to the first and third terms.

$$B + AC$$

At this point the expression is simplified as much as possible. Once you gain experience in applying Boolean algebra, you can often combine many individual steps.

Related Problem

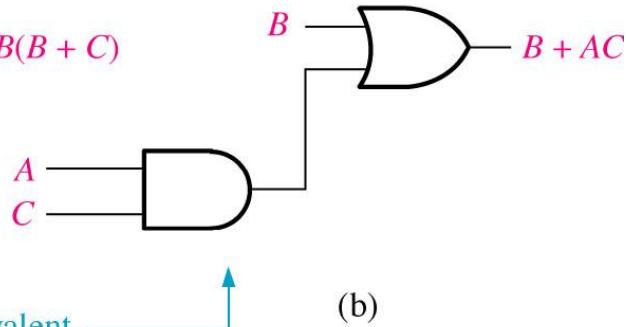
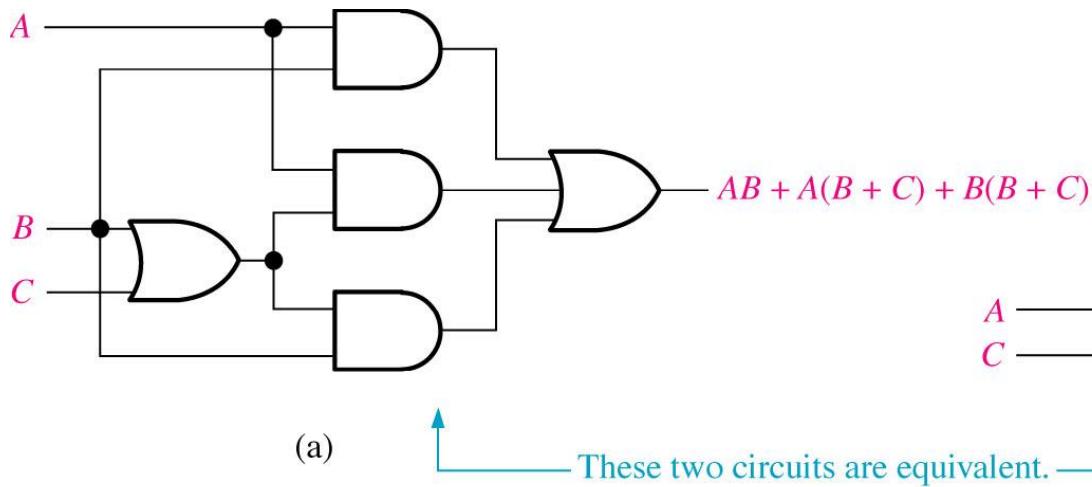
Simplify the Boolean expression $A\bar{B} + A(\bar{B} + \bar{C}) + B(\bar{B} + \bar{C})$.

Summary

Simplification Using Boolean Algebra

Example 4-8: Drawing this expression:

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



Gate circuits for Example 4-8



Summary

Algebraic Manipulation Examples

1. $X + XY = X(1+Y) = X$
2. $XY + XY' = X(Y + Y') = X$
3. $X + X'Y = (X + X')(X + Y) = X + Y$
4. $X(X + Y) = X + XY = X(1 + Y) = X$
5.
$$\begin{aligned}(X + Y)(X + Y') &= X + XY' + XY + YY' \\ &= X(1 + Y') + XY \\ &= X \cdot 1 + XY \\ &= X + XY \\ &= X(1 + Y) \\ &= X\end{aligned}$$
6. $X(X' + Y) = XX' + XY = XY$



Summary

Algebraic Manipulation Examples

Example:

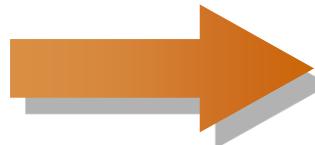
$$\begin{aligned} B(A + C) + AB' + BC' + C \\ &= BA + BC + AB' + BC' + C \\ &= A(B + B') + B(C+C') + C \\ &= A \bullet 1 + B \bullet 1 + C \\ &= A + B + C \end{aligned}$$

Summary

The complement of a function

- ❑ The complement of a function always outputs 0 where the original function outputted 1, and 1 where the original produced 0.
 - ❑ In a truth table, we can just exchange 0s and 1s in the output column(s)
- $$f(x,y,z) = x(y'z' + yz)$$

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



x	y	z	$f'(x,y,z)$
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



Summary

Complementing a function algebraically

- You can use DeMorgan's law to keep "pushing" the complements inwards
- You can also take the dual of the function, and then complement each literal
 - If $f(x,y,z) = x(y'z' + yz)\dots$
 - ...the dual of f is $x + (y' + z')(y + z)\dots$
 - ...then complementing each literal gives $x' + (y + z)(y' + z')\dots$
 - ...so $f'(x,y,z) = x' + (y + z)(y' + z')$

$$f(x,y,z) = x(y'z' + yz)$$

$$\begin{aligned} f'(x,y,z) &= (x(y'z' + yz))' && [\text{complement both sides}] \\ &= x' + (y'z' + yz)' && [\text{because } (xy)' = x' + y'] \\ &= x' + (y'z')' (yz)' && [\text{because } (x + y)' = x' y'] \\ &= x' + (y + z)(y' + z') && [\text{because } (xy)' = x' + y', \text{ twice}] \end{aligned}$$

Selected Key Terms

Variable A symbol used to represent a logical quantity that can have a value of 1 or 0, usually designated by an italic letter.

Complement The inverse or opposite of a number. In Boolean algebra, the inverse function, expressed with a bar over the variable.

Sum term The Boolean sum of two or more literals equivalent to an OR operation.

Product term The Boolean product of two or more literals equivalent to an AND operation.

Quiz

1. The associative law for addition is normally written as

a. $A + B = B + A$

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

c. $AB = BA$

d. $A + AB = A$

Quiz

2. The Boolean equation $B + C = C + B$ illustrates
- a. the distribution law
 - b. the commutative law
 - c. the associative law
 - d. DeMorgan's theorem

Quiz

3. The Boolean equation $AB + AC = A(B + C)$ illustrates

- a. the distribution law
- b. the commutative law
- c. the associative law
- d. DeMorgan's theorem

Quiz

4. The Boolean expression $A \cdot 1$ is equal to

a. A

b. B

c. 0

d. 1

Quiz

5. The Boolean expression $A + 1$ is equal to

a. A

b. B

c. 0

d. 1



Summary

SOP and POS forms

Boolean expressions can be written in the **sum-of-products** form (**SOP**) or in the **product-of-sums** form (**POS**). These forms can simplify the implementation of combinational logic, particularly with PLDs. In both forms, an overbar cannot extend over more than one variable.

An expression is in SOP form when two or more product terms are summed as in the following examples:

$$\overline{A} \overline{B} \overline{C} + A B$$

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

$$C D + \overline{E}$$

An expression is in POS form when two or more sum terms are multiplied as in the following examples:

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

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

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



Summary

SOP Standard form

In **SOP standard form**, every variable in the domain must appear in each term. This form is useful for constructing truth tables or for implementing logic in PLDs.

You can expand a nonstandard term to standard form by multiplying the term by a term consisting of the sum of the missing variable and its complement.

Example Solution

Convert $X = \bar{A} \bar{B} + A B C$ to standard form.

The first term does not include the variable C . Therefore, multiply it by the $(C + \bar{C})$, which = 1:

$$\begin{aligned} X &= \bar{A} \bar{B} (C + \bar{C}) + A B C \\ &= \bar{A} \bar{B} C + \bar{A} \bar{B} \bar{C} + A B C \end{aligned}$$

Summary

SOP Standard form

Example Solution

Convert $X = \bar{A}B\bar{C} + A\bar{B}D + \bar{A}B\bar{C}D$ to standard form.

The first term: multiply it by the $(D + \bar{D})$, which = 1:

The second term: multiply it by the $(C + \bar{C})$, which = 1:

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

Diagram illustrating the terms of the expression $\bar{A}\bar{B}\bar{C} + A\bar{B}D + \bar{A}B\bar{C}D$. The first term $\bar{A}\bar{B}\bar{C}$ is multiplied by $(D + \bar{D})$, resulting in $\bar{A}\bar{B}\bar{C}(D + \bar{D}) = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}$. The second term $A\bar{B}D$ is multiplied by $(C + \bar{C})$, resulting in $A\bar{B}D(C + \bar{C}) = A\bar{B}DC + A\bar{B}D\bar{C}$.

$$\begin{aligned} & \bar{A}\bar{B}\bar{C} \times (D + \bar{D}) \\ & A\bar{B}D \times (C + \bar{C}) \end{aligned}$$

Diagram illustrating the multiplication of each term by its complement. The first term $\bar{A}\bar{B}\bar{C}$ is multiplied by $(D + \bar{D})$, resulting in $\bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}$. The second term $A\bar{B}D$ is multiplied by $(C + \bar{C})$, resulting in $A\bar{B}DC + A\bar{B}D\bar{C}$.

$$\bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}DC + A\bar{B}D\bar{C} + \bar{A}\bar{B}\bar{C}D$$



Summary

POS Standard form

In **POS standard form**, every variable in the domain must appear in each sum term of the expression.

You can expand a nonstandard POS expression to standard form by adding the product of the missing variable and its complement and applying rule 12, which states that $(A + B)(A + C) = A + BC$.

Example

Convert $X = (\bar{A} + \bar{B})(A + B + C)$ to standard form.

Solution

The first sum term does not include the variable C . Therefore, add $C\bar{C}$ and expand the result by rule 12.

$$\begin{aligned} X &= (\bar{A} + \bar{B} + C\bar{C})(A + B + C) \\ &= (\bar{A} + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})(A + B + C) \end{aligned}$$

Summary

POS Standard form

Example

Convert $X = (\bar{A} + \bar{B} + C)(\bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$ to standard form.

Solution

The first sum term: add $D \bar{D}$ & expand the result by rule 12.

The Second sum term: add $A \bar{A}$ & expand the result by rule 12.

$$(A + \bar{B} + C)(\bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$$

$$\begin{array}{c} \swarrow \quad \searrow \\ \bar{D} \quad D \end{array}$$

$$\begin{array}{c} \swarrow \quad \searrow \\ A \quad A \end{array}$$

$$\begin{aligned} & (A + \bar{B} + C) + D \bar{D} \\ & (\bar{B} + C + \bar{D}) + A \bar{A} \end{aligned}$$

Rule 12!

Add:

$$(A + \bar{B} + C + D)(A + \bar{B} + C + \bar{D})(A + \bar{B} + C + D)(\bar{A} + \bar{B} + C + \bar{D})(\bar{A} + \bar{B} + \bar{C} + D)$$

Summary

Converting SOP to Truth Table

- ❑ Examine each of the products to determine where the product is equal to a 1.
- ❑ Set the remaining row outputs to 0.

$$X = \overline{A}\overline{B}\overline{C} + A\overline{B} + A\overline{B}C$$

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Summary

Converting POS to Truth Table

- ❑ Opposite process from the SOP expressions.
- ❑ Each sum term results in a 0.
- ❑ Set the remaining row outputs to 1.

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

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Diagram showing arrows from the terms in the POS expression to the rows of the truth table where the output X is 0. The first arrow points to the row A=0, B=0, C=0. The second arrow points to the row A=0, B=0, C=1. The third arrow points to the row A=0, B=1, C=0. The fourth arrow points to the row A=0, B=1, C=1.

Summary

Converting from Truth Table to SOP and POS

Inputs			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

SOP:

$$X = \overline{A}BC + A\overline{B}\overline{C} + AB\overline{C} + ABC$$

POS:

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



Canonical SOP and minterms

- ❑ An AND term containing every variable of the function (complemented or NOT) is called a minterm.
- ❑ Canonical SOP form is a sum of minterms.
- ❑ Why canonical SOP form?
 - ❑ There is only one way to implement a function in canonical SOP form.



Summary

Canonical POS and Maxterms

- ❑ An OR term containing every variable of the function (complemented or NOT) is called a Maxterm.
- ❑ Canonical POS form is a Product of Maxterms.
- ❑ Why canonical POS form?
 - ❑ There is only one way to implement a function in canonical POS form.



Summary

minterms for three variables, XYZ:

			Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
X	Y	Z										
0	0	0	$\overline{X}\overline{Y}\overline{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}YZ$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{XY}\overline{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	\overline{XYZ}	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\overline{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

- Each binary combination has a related product term.
- Complement variable if 0.
- Uncomplement variable if 1.
- m_j , j corresponds to binary number for which minterm is 1.



Summary

Maxterm for three variables, XYZ:

X	Y	Z	Sum Term	Symbol	M ₀	M ₁	M ₂	M ₃	M ₄	M ₅	M ₆	M ₇
0	0	0	$X + Y + Z$	M ₀	0	1	1	1	1	1	1	1
0	0	1	$X + Y + \bar{Z}$	M ₁	1	0	1	1	1	1	1	1
0	1	0	$X + \bar{Y} + Z$	M ₂	1	1	0	1	1	1	1	1
0	1	1	$X + \bar{Y} + \bar{Z}$	M ₃	1	1	1	0	1	1	1	1
1	0	0	$\bar{X} + Y + Z$	M ₄	1	1	1	1	0	1	1	1
1	0	1	$\bar{X} + Y + \bar{Z}$	M ₅	1	1	1	1	1	0	1	1
1	1	0	$\bar{X} + \bar{Y} + Z$	M ₆	1	1	1	1	1	1	0	1
1	1	1	$\bar{X} + \bar{Y} + \bar{Z}$	M ₇	1	1	1	1	1	1	1	0

- A sum term that contains all the variables in complemented or uncomplemented form.
- Complement variable if 1.
- Uncomplement variable if 0.
- M_j , j denotes binary value for which maxterm is 0.



Summary

- Any truth table can be represented as a Boolean function in canonical SOP form:
just list out minterms.
- Any Boolean function can be expressed algebraically from a given truth table as a product of maxterms.

x	y	z	F	\bar{F}	$F(X,Y,Z)$	Abbreviated as:
0	0	0	1	0	$F = x'y'z' + x'yz' + xy'z + xyz$	
0	0	1	0	1		
0	1	0	1	0	$F = m_0 + m_2 + m_5 + m_7$	
0	1	1	0	1		
1	0	0	0	1	$F = \sum m (0, 2, 5, 7)$	
1	0	1	1	0		
1	1	0	0	1	$F'(X,Y,Z) = \sum m (1, 3, 4, 6)$	
1	1	1	1	0		

Summary

Product of maxterms form

- Every function can be written as a *unique product of maxterms*
- If you have a truth table for a function, you can write a product of maxterms expression function by picking out the rows of the table where the function is 0, and writing the maxterm for that row (the complement of the minterm for the row).

x	y	z	f(x,y,z)	f'(x,y,z)
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$\begin{aligned}f &= (x' + y + z)(x' + y + z')(x' + y' + z') \\&= M_4 M_5 M_7 \\&= \prod M(4,5,7)\end{aligned}$$

$$\begin{aligned}f' &= (x + y + z)(x + y + z')(x + y' + z) \\&\quad (x + y' + z')(x' + y' + z) \\&= M_0 M_1 M_2 M_3 M_6 \\&= \prod M(0,1,2,3,6)\end{aligned}$$

f' contains all the maxterms not in f

Summary

Product of maxterms form

- Every function can be written as a *unique product of maxterms*
- If you have a truth table for a function, you can write a product of maxterms expression function by picking out the rows of the table where the function is 0, and writing the maxterm for that row (the complement of the minterm for the row).

x	y	z	f(x,y,z)	f'(x,y,z)
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$\begin{aligned}f &= (x' + y + z)(x' + y + z')(x' + y' + z') \\&= M_4 M_5 M_7 \\&= \prod M(4,5,7)\end{aligned}$$

$$\begin{aligned}f' &= (x + y + z)(x + y + z')(x + y' + z) \\&\quad (x + y' + z')(x' + y' + z) \\&= M_0 M_1 M_2 M_3 M_6 \\&= \prod M(0,1,2,3,6)\end{aligned}$$

f' contains all the maxterms not in f



Summary

Product of maxterms form

- Find a row in the truth table for f for which the function is 0
- Write down the *complement* of the minterm for that row
 - E.g., if the row is 110, then use DeMorgan's law to change xyz' into $(x' + y' + z)$.
- The product of these *maxterms* is the original function f
- Why does this procedure works?
- $F' = A + B + C + D$ *because we picked the rows for which f is 0*
- $F = A'B'C'D'$ *by DeMorgan's law*



Summary

Minterms and maxterms are related

- Any minterm m_i is the *complement* of the corresponding maxterm M_i
- For example, $m_4' = M_4$ because $(xy'z')' = x' + y + z$

Minterm	Shorthand	Maxterm	Shorthand
$x'y'z'$	m_0	$x + y + z$	M_0
$x'y'z$	m_1	$x + y + z'$	M_1
$x'yz'$	m_2	$x + y' + z$	M_2
$x'yz$	m_3	$x + y' + z'$	M_3
$xy'z'$	m_4	$x' + y + z$	M_4
$xy'z$	m_5	$x' + y + z'$	M_5
xyz'	m_6	$x' + y' + z$	M_6
xyz	m_7	$x' + y' + z'$	M_7



Summary

Converting between standard forms

- We can convert a sum of minterms to a product of maxterms
- In general, just replace the minterms with maxterms, using maxterm numbers that don't appear in the sum of minterms:

From before

and

complementing

so

$$f = \Sigma m(0,1,2,3,6)$$

$$f' = \Sigma m(4,5,7)$$

$$= m_4 + m_5 + m_7$$

$$(f')' = (m_4 + m_5 + m_7)'$$

$$f = m_4' m_5' m_7'$$

$$= M_4 M_5 M_7$$

$$= \prod M(4,5,7)$$

[DeMorgan's law]

[By the previous page]

- The same thing works for converting from a product of maxterms to a sum of minterms.

$$f = \Sigma m(0,1,2,3,6)$$

$$= \prod M(4,5,7)$$

Quiz

1. A Boolean expression that is in standard SOP form is
 - a. the minimum logic expression
 - b. contains only one product term
 - c. has every variable in the domain in every term
 - d. none of the above

Chapter 5

Gate Level Minimization

This is the task of finding an optimal Gate-level optimization of the Boolean functions describing a digital circuit.



-minimum of gates and inputs

Gate Level Minimization

The Map Method

Karnaugh or K-Map

M i n t e r m s

Sum of Products

Product of Sums

map

- diagram of squares
- each square represents one minterm (one row in truth table)
- For 2 variables:
 - 4 minterms
 - $x'y'$, $x'y$, xy' , xy

columns in which $x = 1$

	$x \backslash y$	0	1
0	m_0	m_1	$x'y'$
1	m_2	m_3	xy'

Two Variable Map

m_0	m_1
m_2	m_3

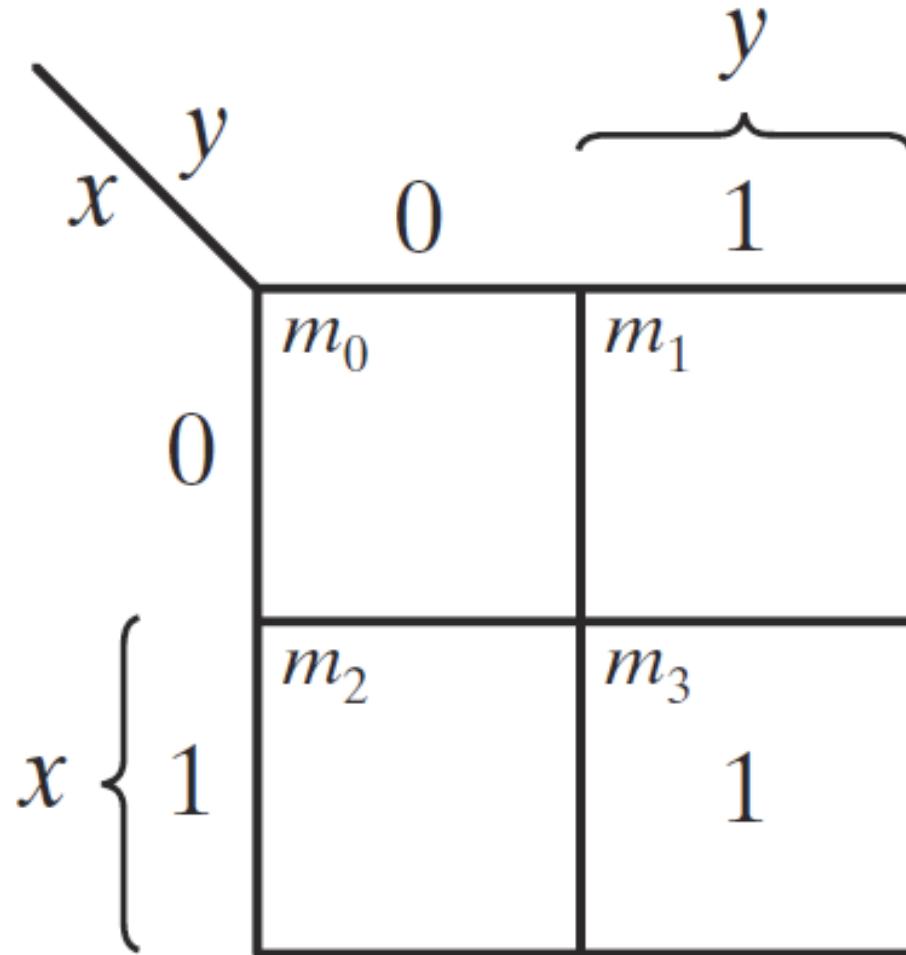
(a)

	m_1	
0	$x'y'$	$x'y$
	m_2	m_3
1	xy'	xy

(b)

two-variable map

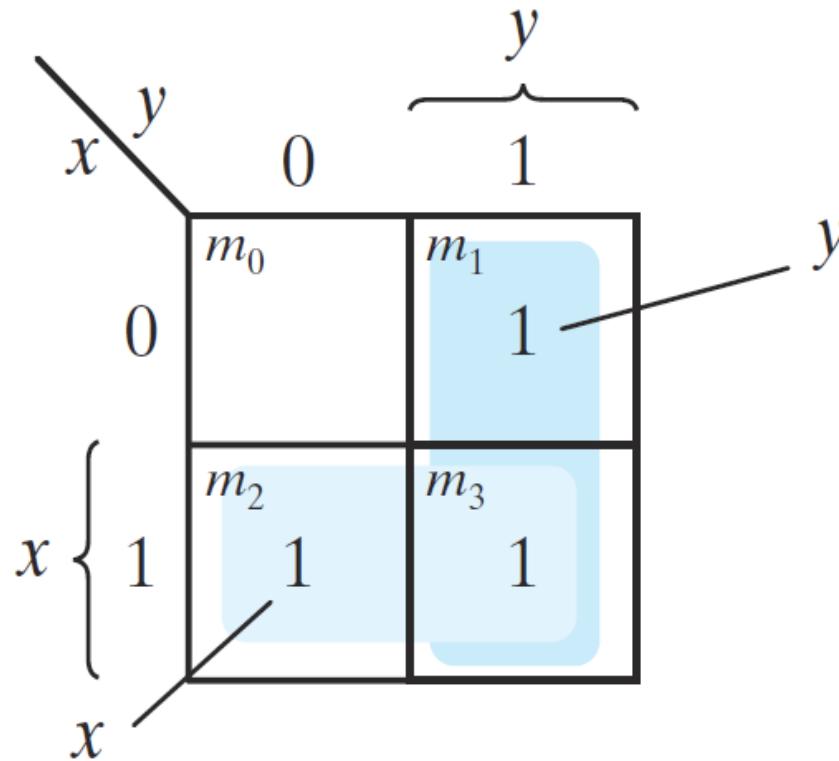
- Example: $F = xy$



two-variable map

- Example: $F = x + y$

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



Gray code

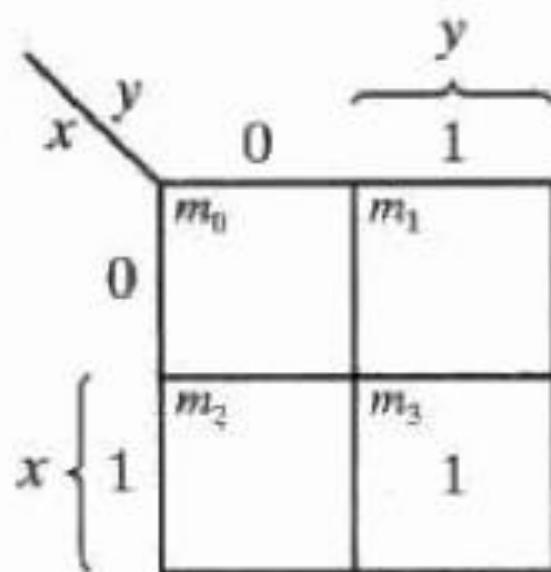
- only one bit changes from number to next:
 - 7 -> 0100
 - 8 -> 1100
- 2-bit Gray code
 - 00, 01, 11, 10

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

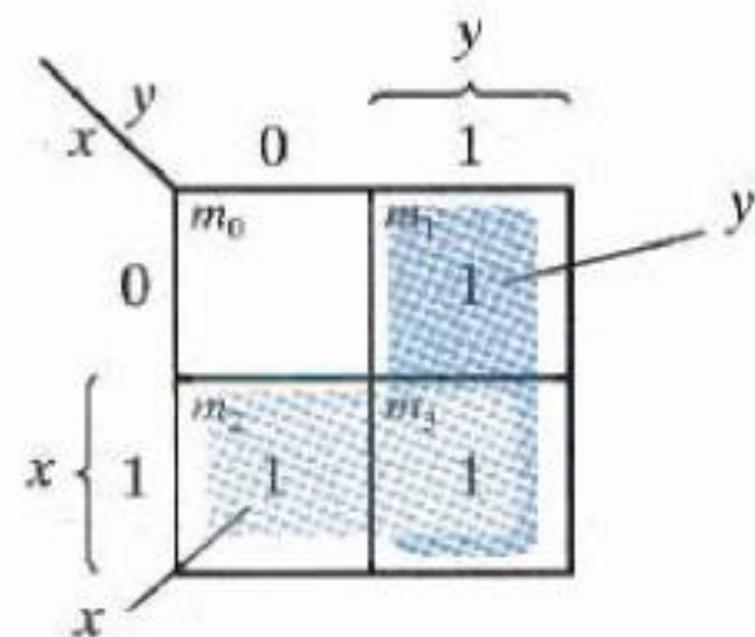
Two Variable Map

$$\begin{aligned}m_1 + m_2 + m_3 &= x' y + x y' + x y \\&= x (y + y') + x' y \\&= x + x' y \\&= x + x' y \\&= (x + x')(x + y) \\&= x + y\end{aligned}$$

Two Variable Map



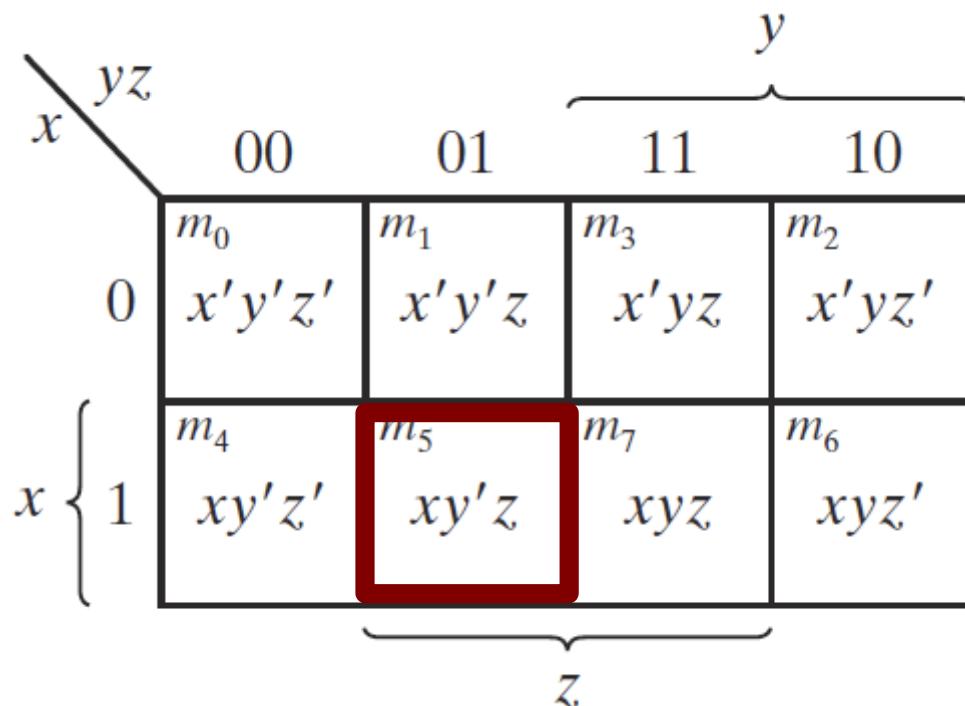
(a) xy



(b) $x + y$

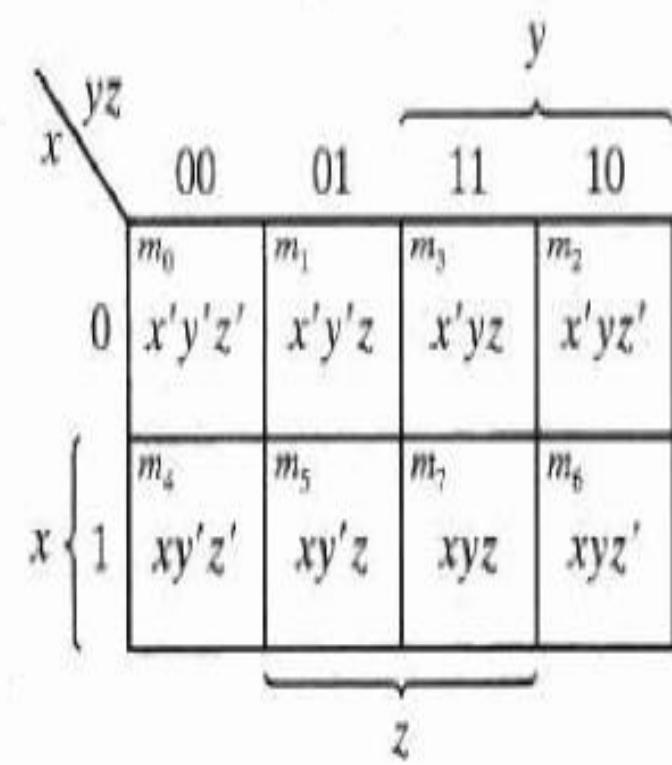
three-variable map

- 8 minterms
- column sequence in Gray code
- row 1 and column 01 -> 1 01 = 5 -> m_5
- row x and column $y'z$ -> $x y'z$



Three Variable Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



3-variable map simplification

- 1 cell -> three literals (e.g., xyz)
- 2 adjacent cells -> two literals (e.g., xy)
- 4 adjacent cells -> one literal (e.g., z)
- 8 adjacent cells -> ??

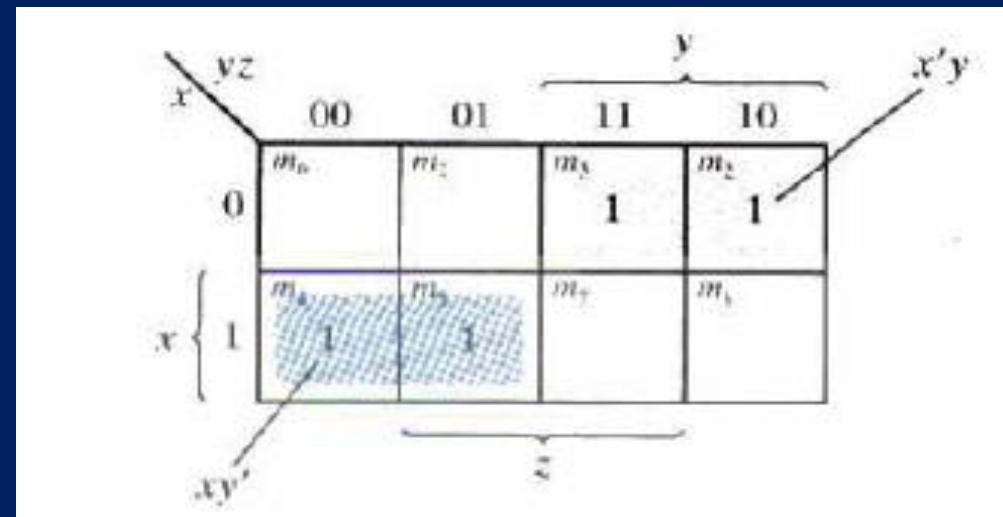
Three-Variable Map

$$\begin{aligned}m_5 + m_7 &= x \bar{y} z + x y \bar{z} \\&= x z (\bar{y} + y) \\&= x z\end{aligned}$$

Three Variable Map

Example 1 :

Simplify the function $F(x, y, z) = \sum (2, 3, 4, 5)$



Three Variable Map

Example 1:

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

$$m_2 + m_3 + m_4 + m_5 =$$

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

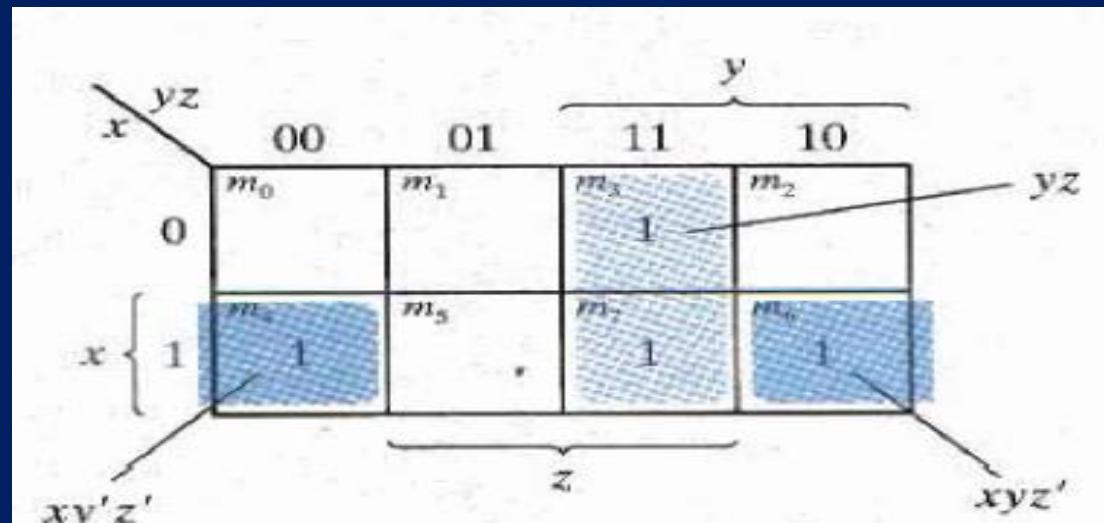
$$= x' y (z' + z) + x y' (z' + z)$$

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

Three Variable Map

Example 2:

Simplify the function $F(x, y, z) = \sum (3, 4, 6, 7)$



Three Variable Map

Adjacent Squares

$$\begin{aligned}m_0 + m_2 &= x` y` z` + x` y z` \\&= x` z` (y` + y) = x` z` \\m_4 + m_6 &= x y` z` + x y z` \\&= x z` (y` + y) \\&= x z`\end{aligned}$$

Three Variable Map

Adjacent Squares

$$m_0 + m_2 + m_4 + m_6 =$$

$$= x` y` z` + x` y z` + x y` z` + x y z`$$

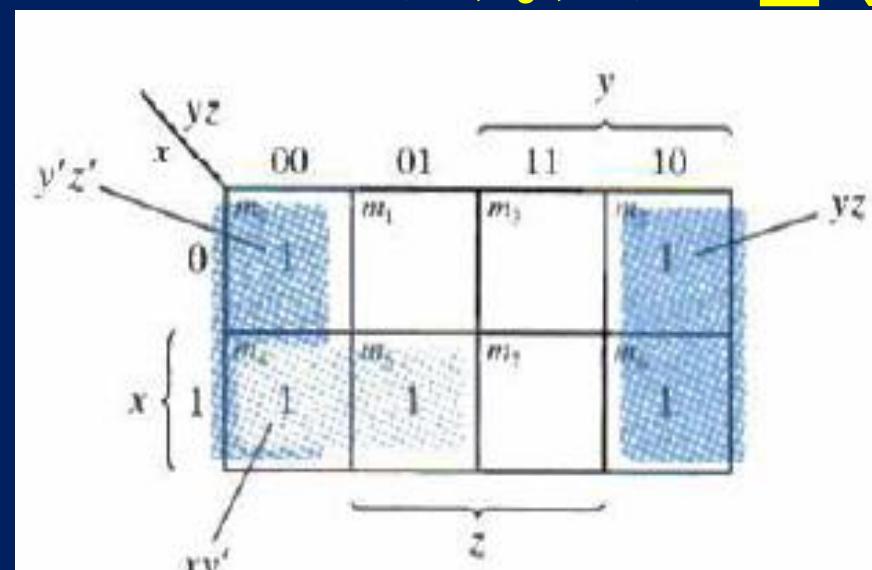
$$= x` z` (y` + y) + x z` (y` + y)$$

$$= x` z` + x z` = z` (x` + x) = z`$$

Three Variable Map

Example 3:

Simplify the function $F(x, y, z) = \sum (0, 2, 4, 5, 6)$



Three Variable Map

Example 3:

Simplify the function $F(x, y, z) = \sum (0, 2, 4, 5, 6)$

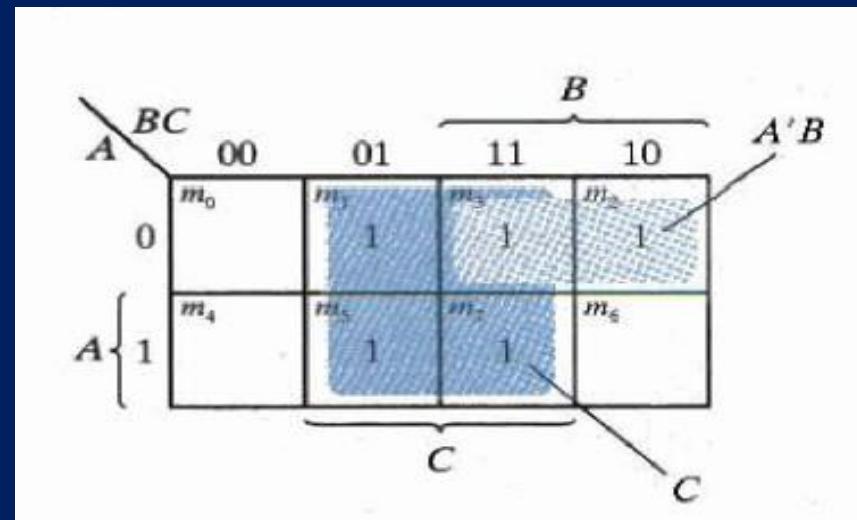
$$m_0 + m_2 + m_4 + m_5 + m_6 =$$

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

Three Variable Map

Example 4:

Simplify the function $F = A'C + A'B + AB'C + BC$



Three Variable Map

Example 4:

Simplify the function $F = A'C + A'B + AB'C + BC$

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

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

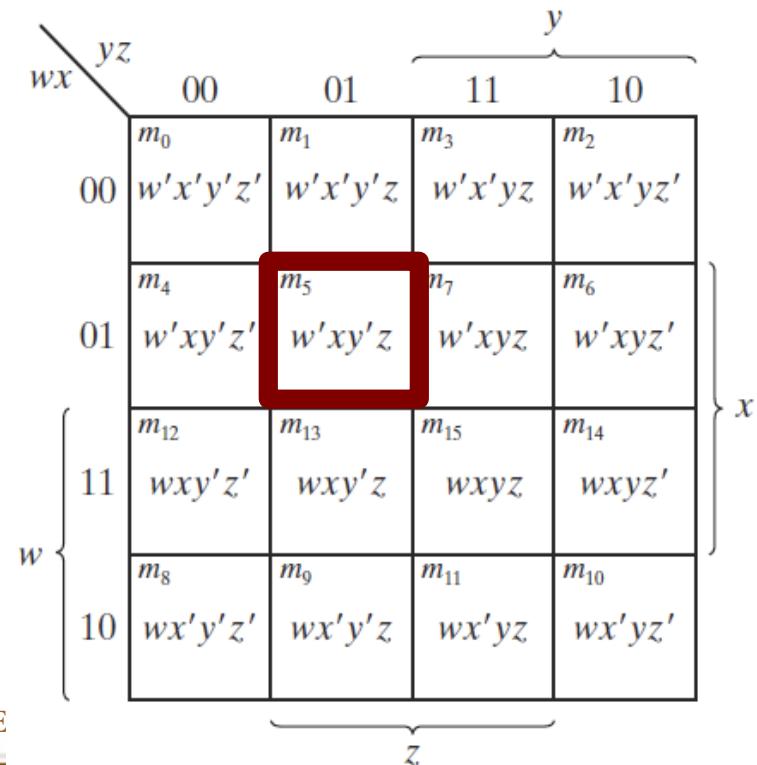
$$= A'B'C + A'BC' + (A' + A)BC + AB'C$$

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

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

four-variable map

- 16 minterms
- column sequence in Gray code
- row 01 and column 01 $\rightarrow 01\ 01 = 5 \rightarrow m_5$
row $w'x$ and column $y'z \rightarrow w'x\ y'z$



Four Variable 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}

(a)

$wx \searrow yz$

		00	01	y	
		11	10		
		00	01	11	10
w	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} $wx'yz'$

x

\overbrace{z}

(b)

Adjacent Squares

One Square represents one Minterm ,gives four literals

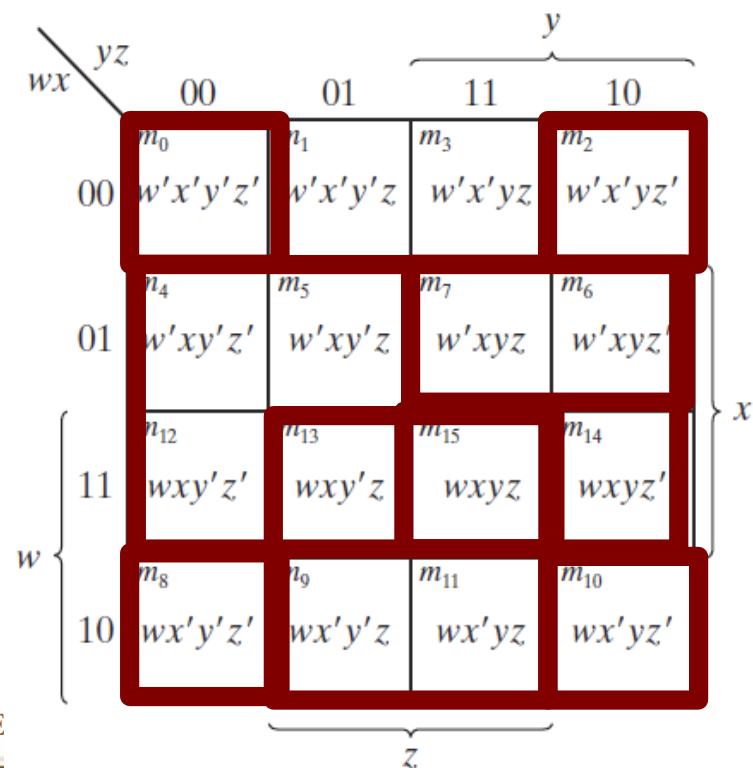
Two adjacent Squares represent a term with three literals

Four adjacent Squares represent a term with two literals

Eight adjacent Squares represent a term with one literal

4-variable map simplification

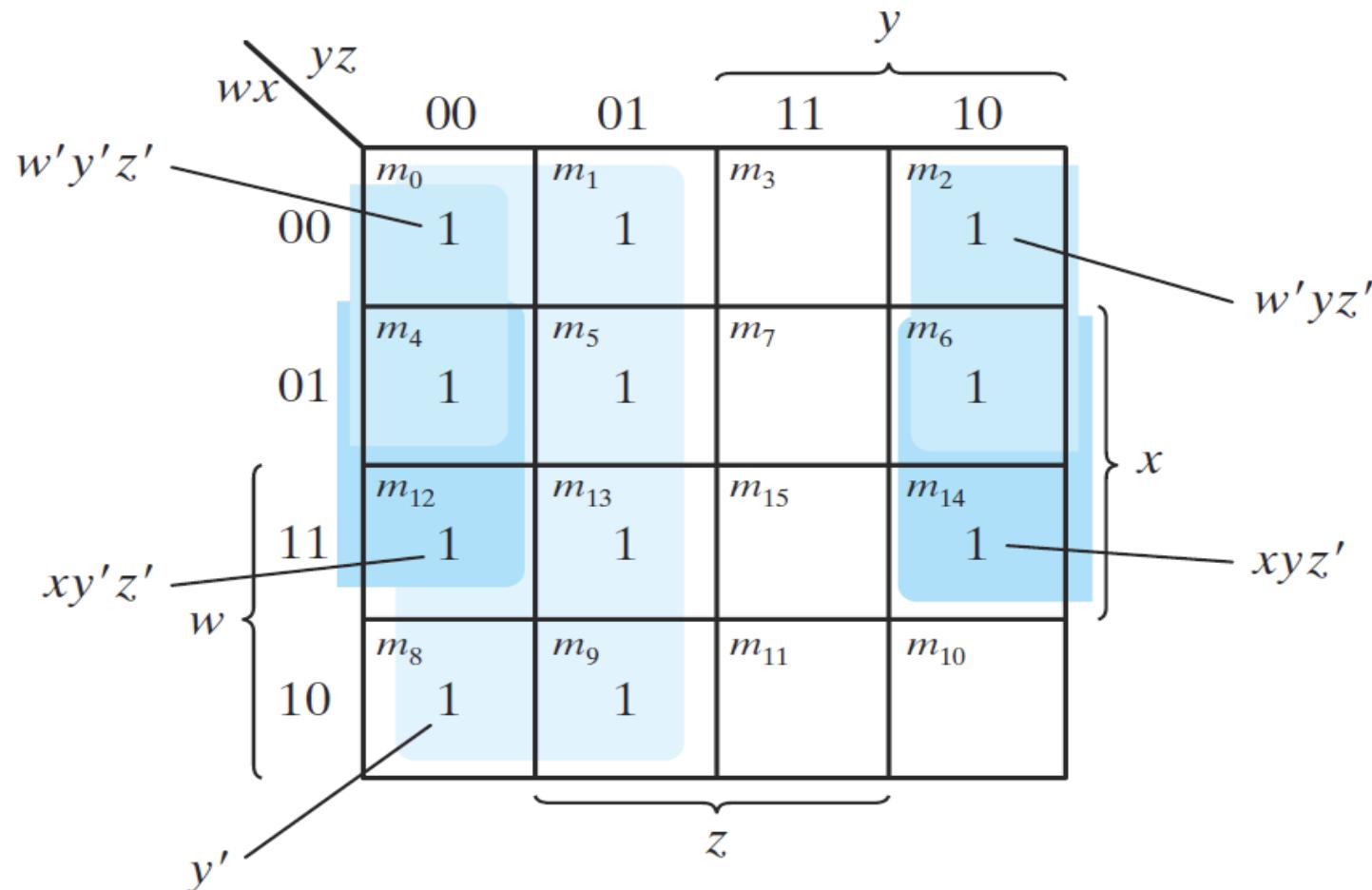
- 1 cell -> four literals (e.g., $wxyz$)
- 2 adjacent cells -> three literals (e.g., $w'xy$)
- 4 adjacent cells -> two literals (e.g., wz)
- 8 adjacent cells -> one literal (e.g., x)
- 16 adjacent cells -> 1
- adjacent?



example 1

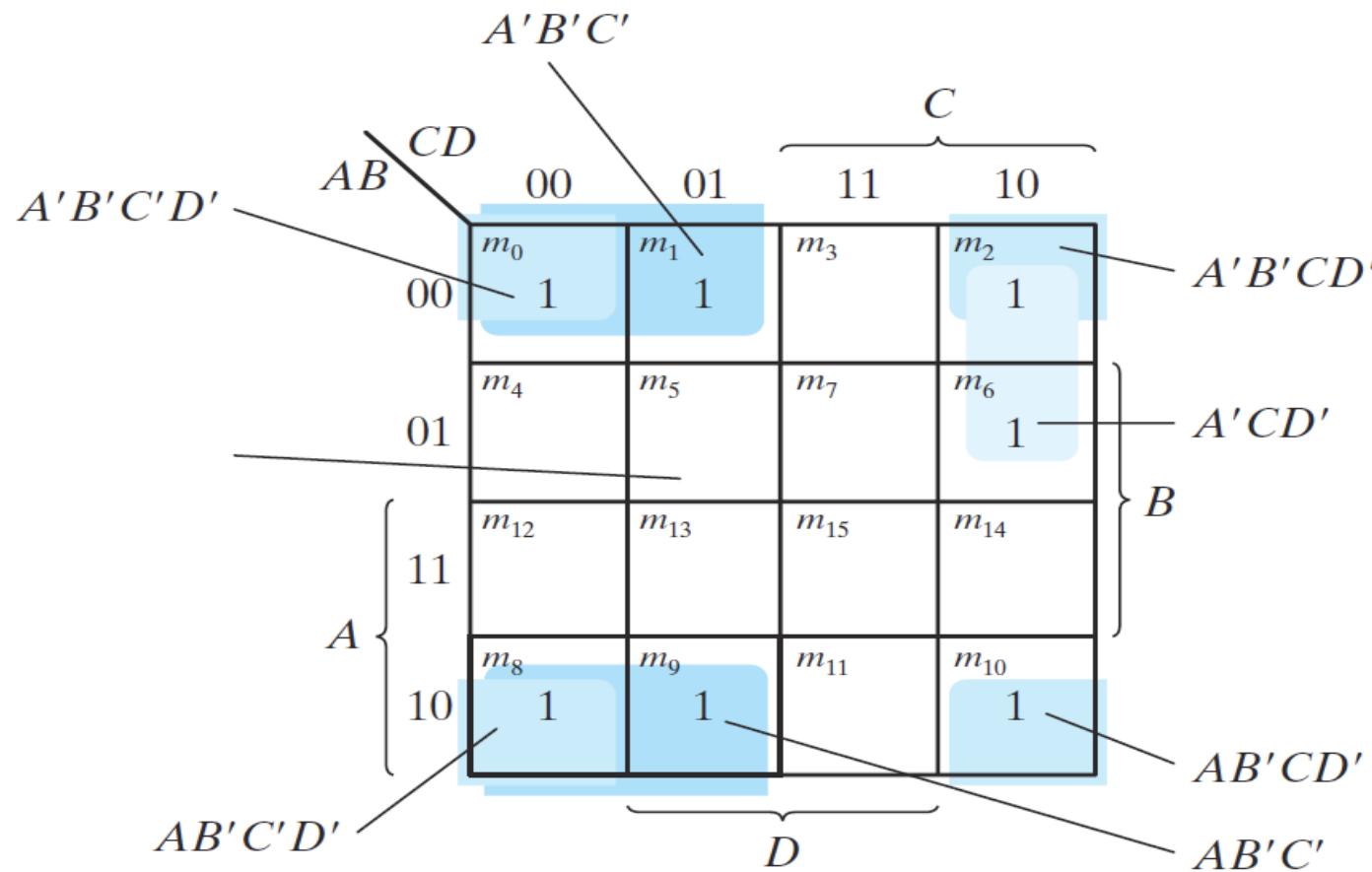
$$F = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

=



example 2

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



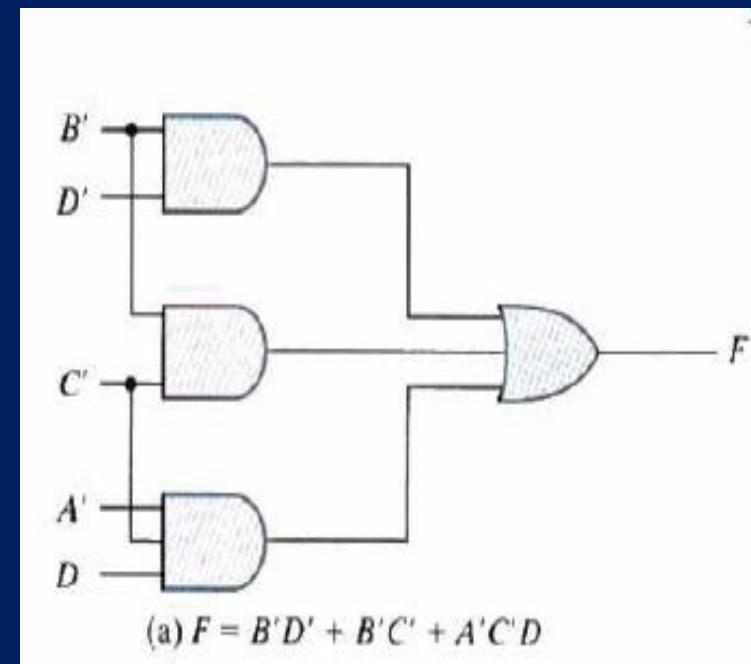
Product of Sums Simplification

$$F(A,B,C,D) = \sum(0,1,2,5,8,9,10)$$

Sum of Products

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

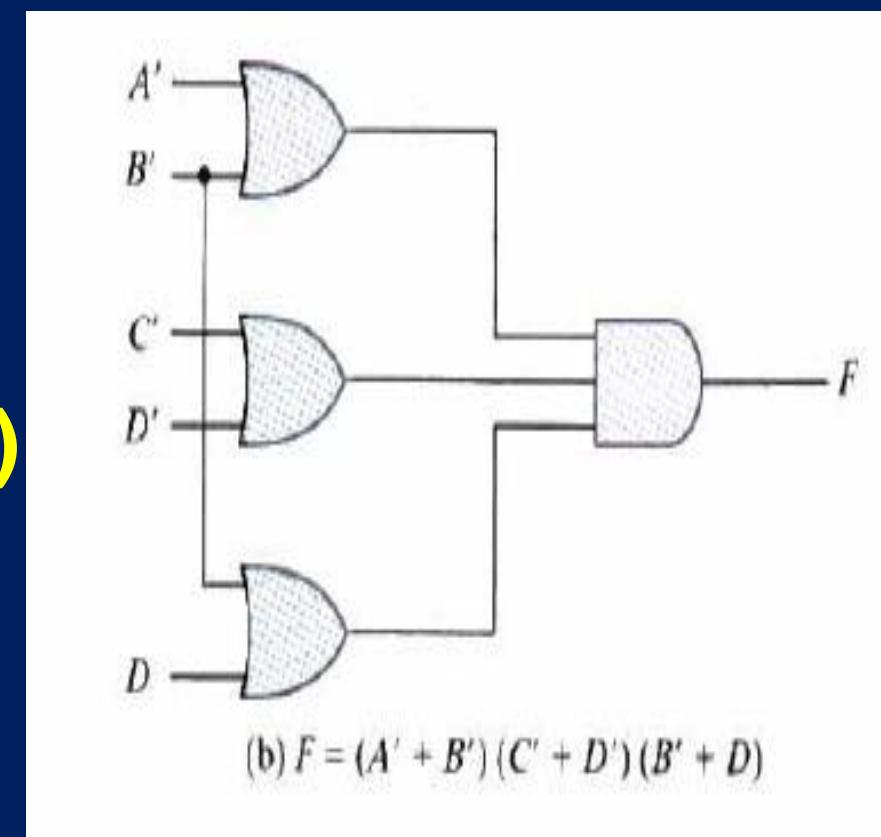
$$F' = AB + CD + BD'$$



Product of Sums Simplification

Product of Sums

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



Don`t Care Condition

Incompletely specified Functions

Steps:

- put X instead of 0 or 1
- Take either 0 or 1
- Select which gives simpler final form.

Don`t Care Condition

Example:

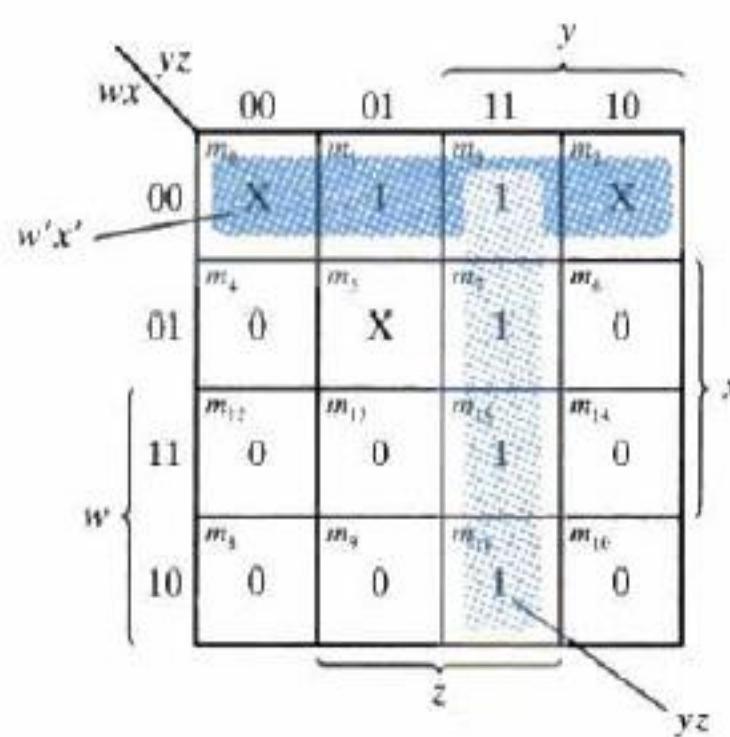
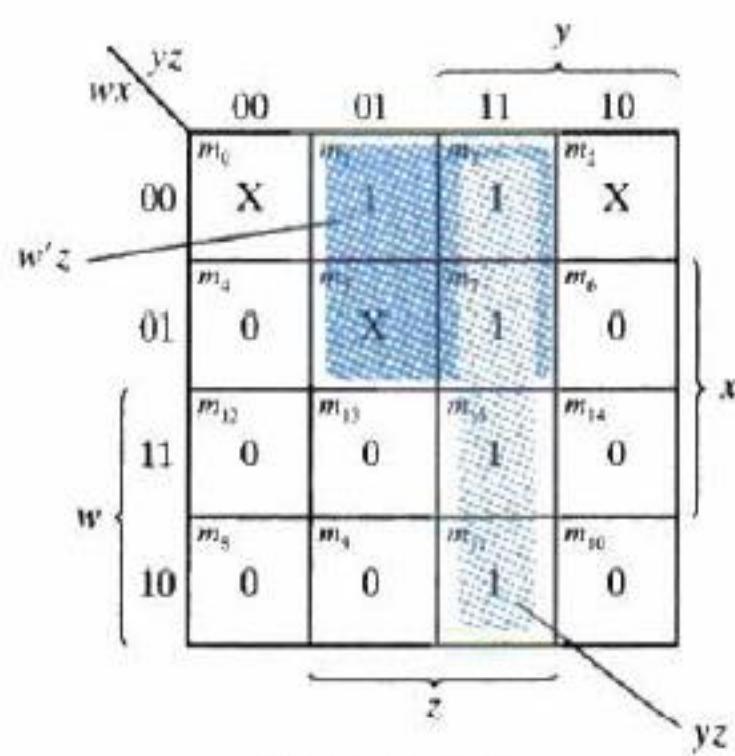
Simplify the function

$$F(w, x, y, z) = \sum (1, 3, 7, 11, 15)$$

With a Don`t Care condition:

$$d(w, x, y, z) = \sum (0, 2, 5)$$

Don't Care Condition

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

Don`t Care Condition

Example(con):

$$F(w, x, y, z) = \sum(0, 1, 2, 3, 7, 11, 15)$$

$$F = yz + w'x'$$

Don`t Care Condition

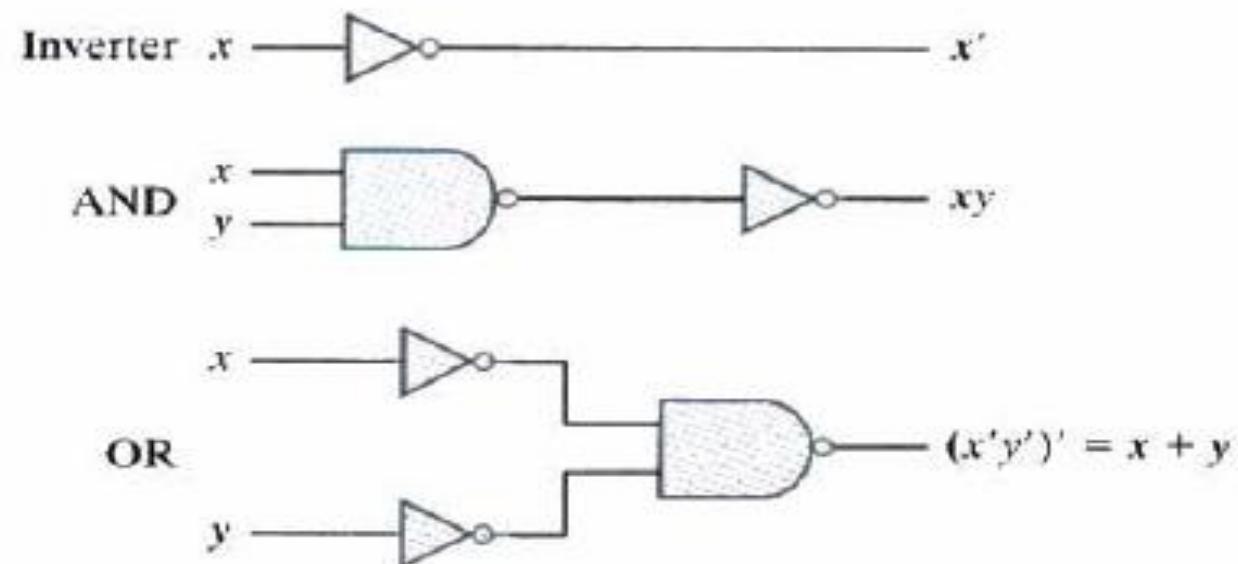
Example (con):

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

$$F = yz + w'z$$

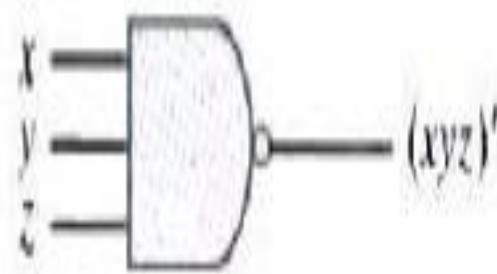
NAND and NOR Implementation

NAND circuits

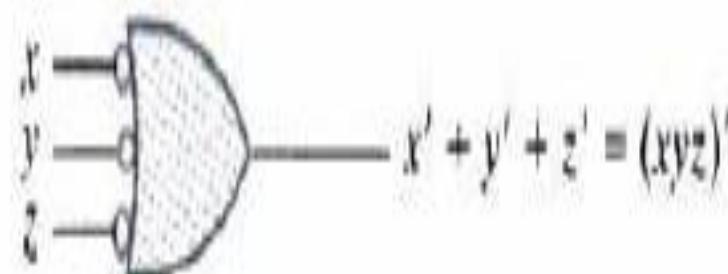


Logic operations with NAND gates

NAND and NOR Implementation



(a) AND-invert

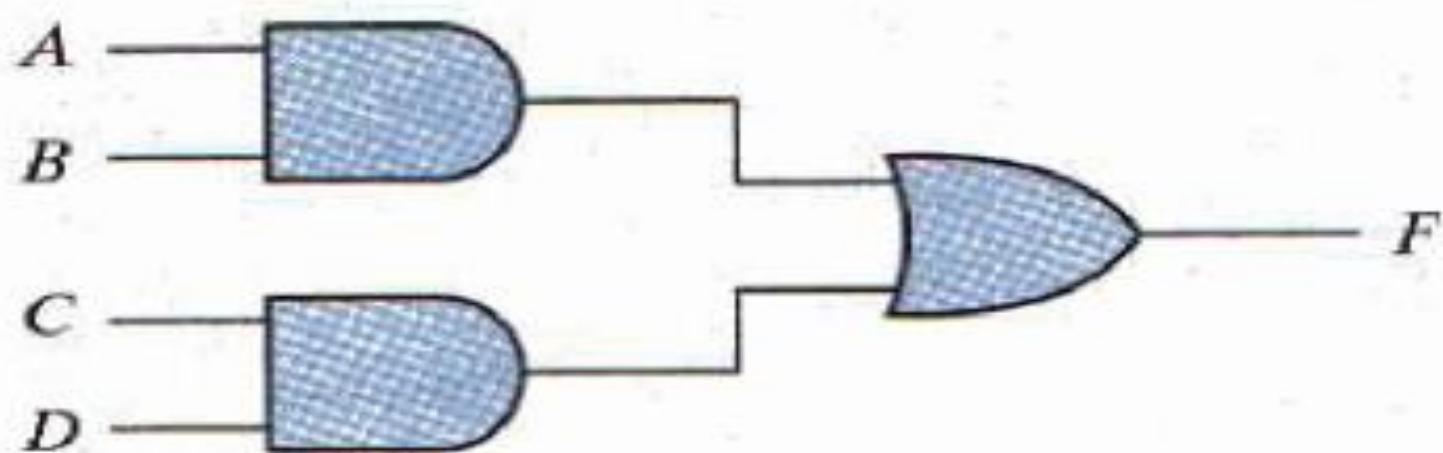


(b) Invert-OR

Two Level Implementation

NAND circuits

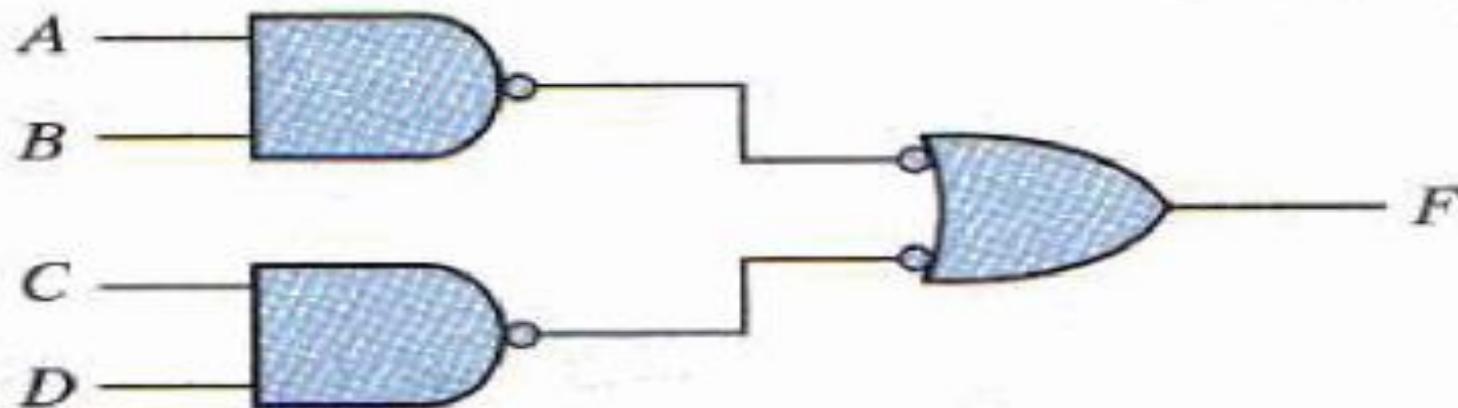
$$F = A \bar{B} + C \bar{D}$$



(a)

Two Level Implementation

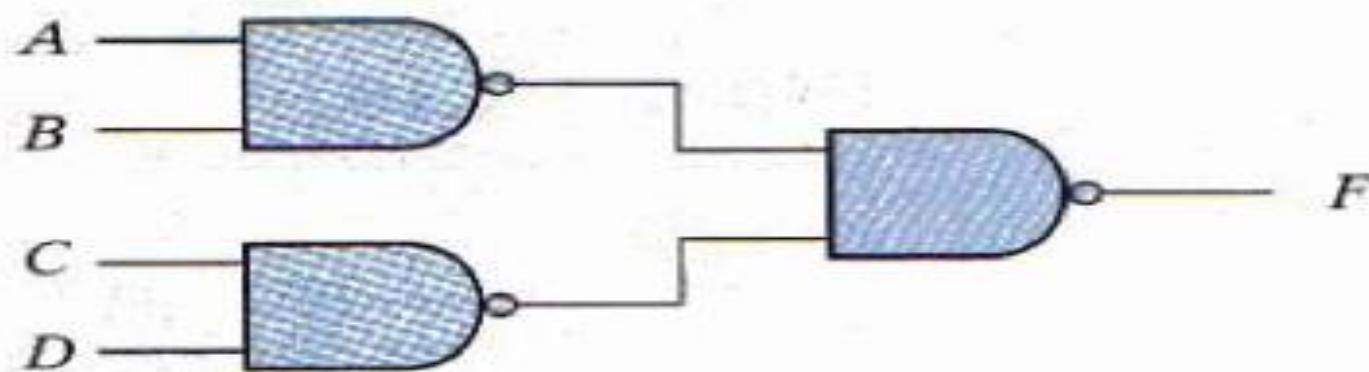
$$F = A \cdot B + C \cdot D$$



(b)

Two Level Implementation

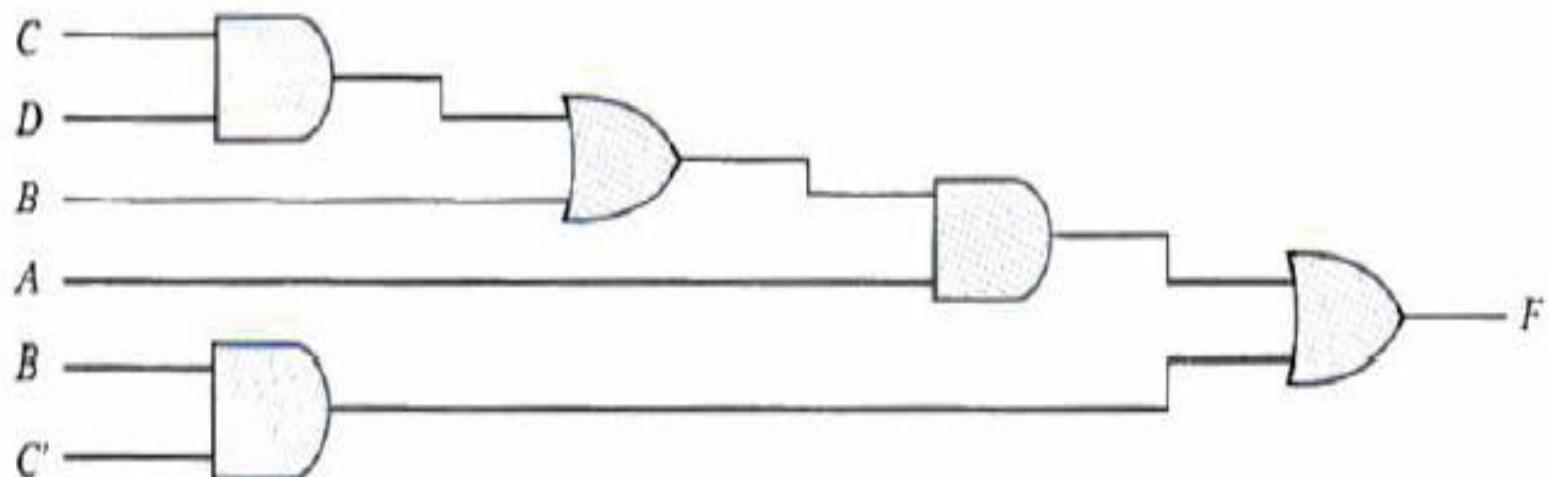
$$F = A \cdot B + C \cdot D$$



(c)

Multi Level NAND Circuits

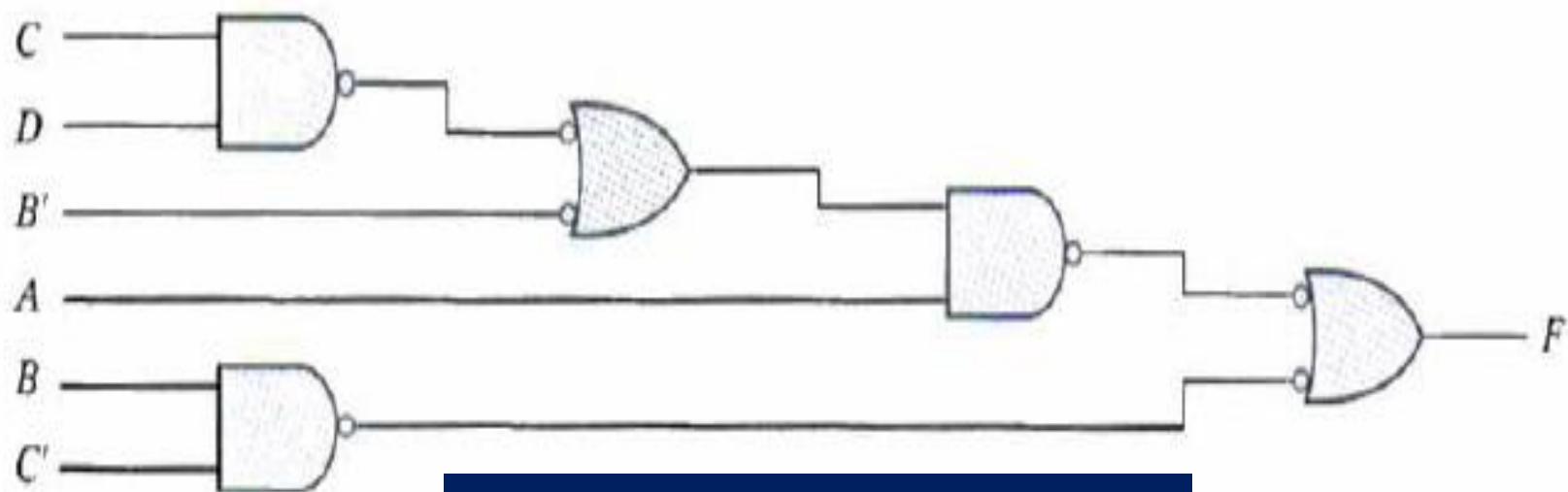
$$F = A(CD + B) + B C'$$



AND-OR Gates

Multi Level NAND Circuits

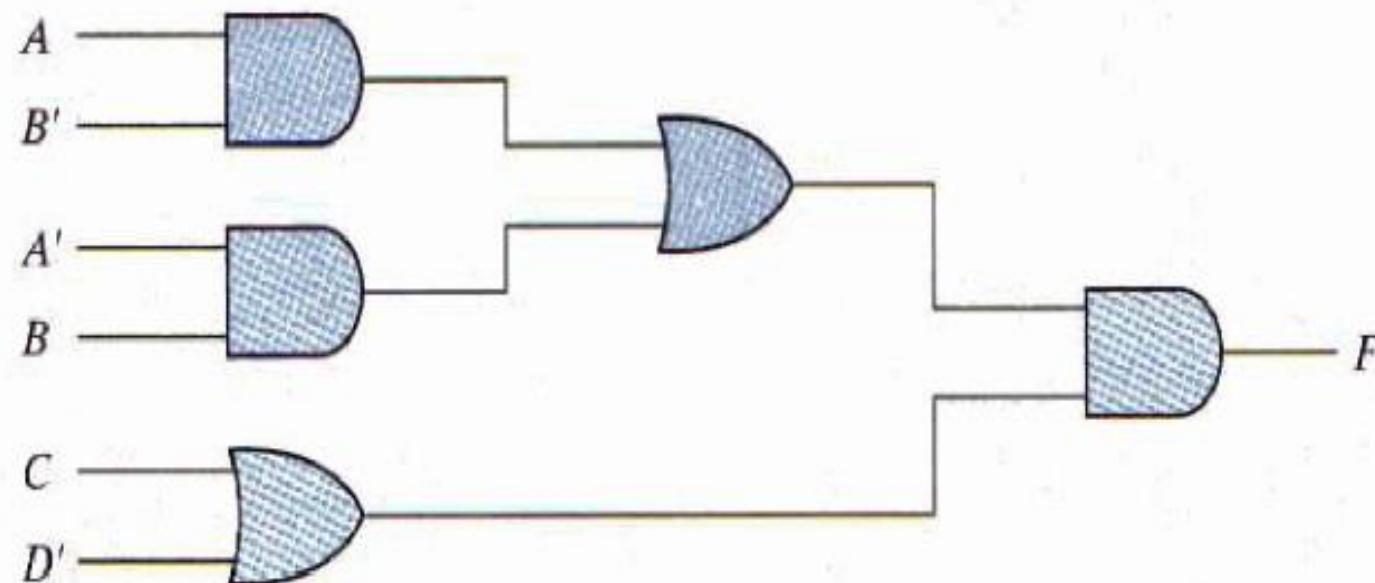
$$F = A(CD+B)+B'C$$



NAND Gates

NAND Implementation

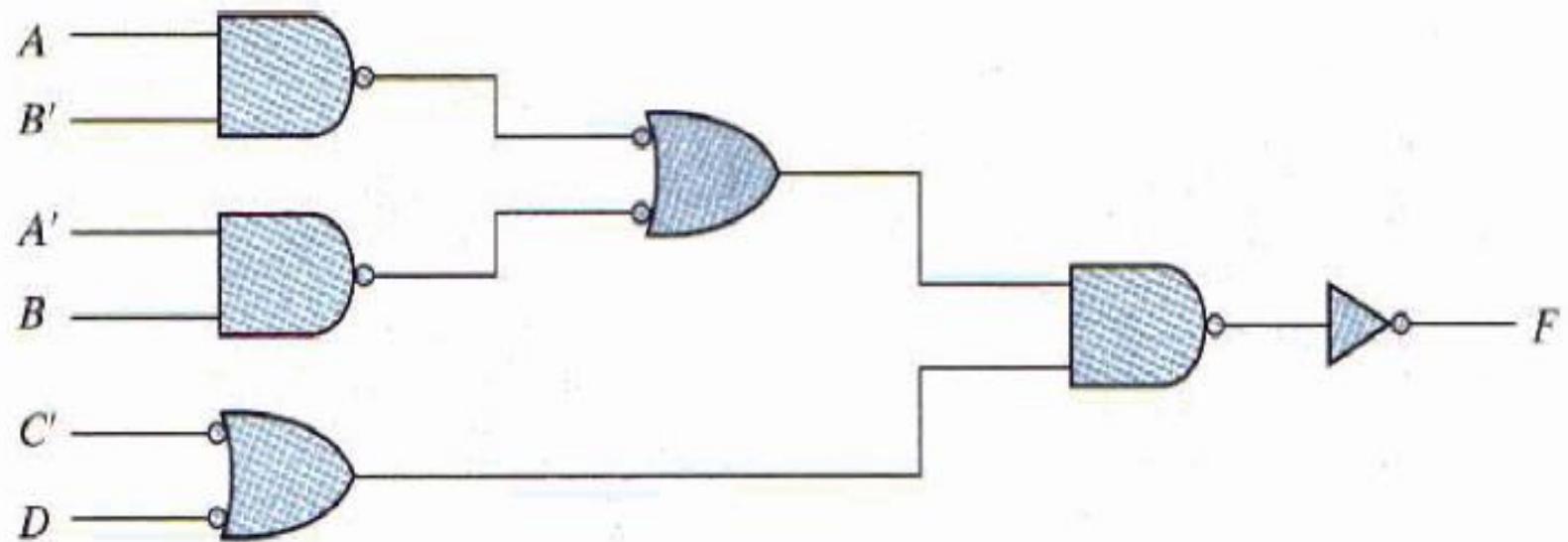
$$F = A B' + A' B (C + D')$$



(a) AND-OR gates

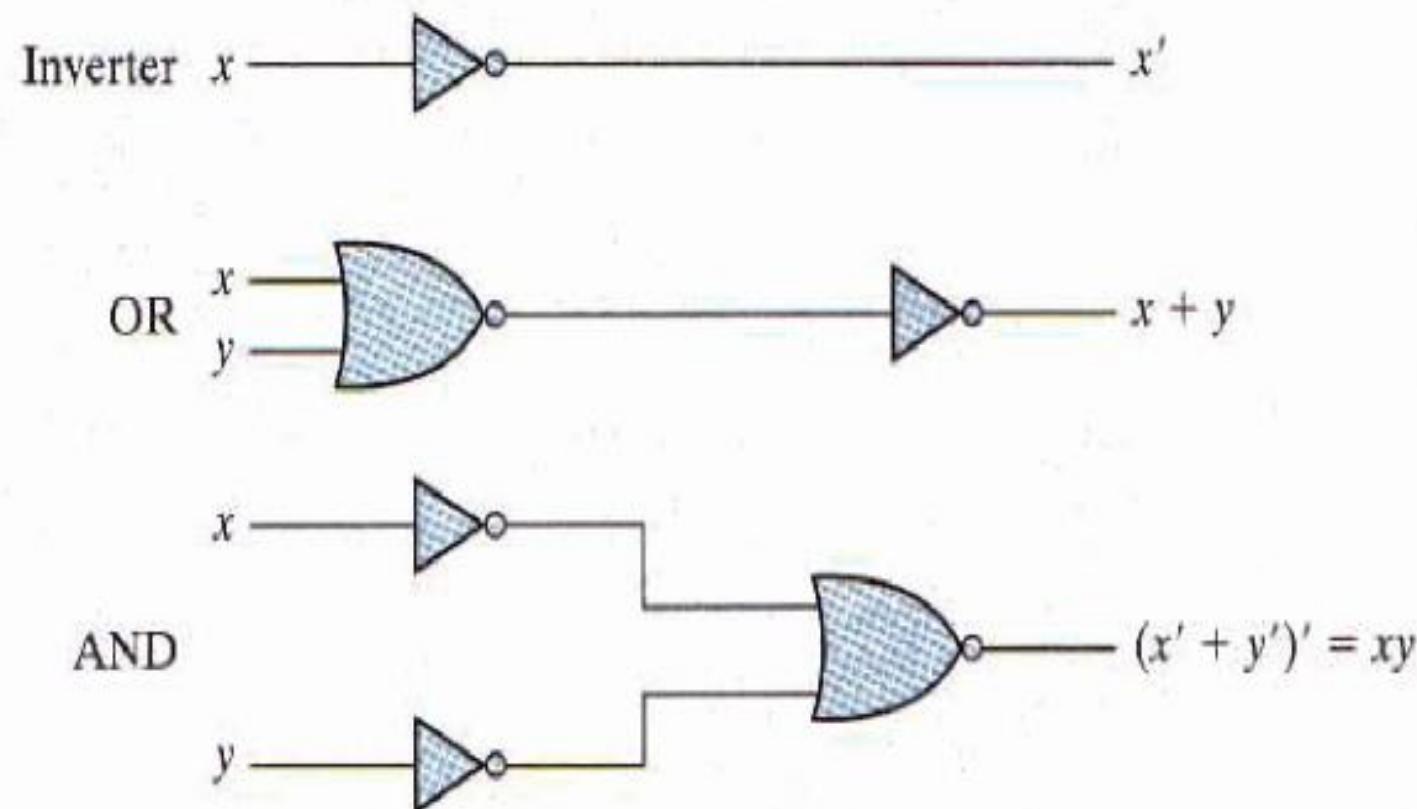
NAND Implementation

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

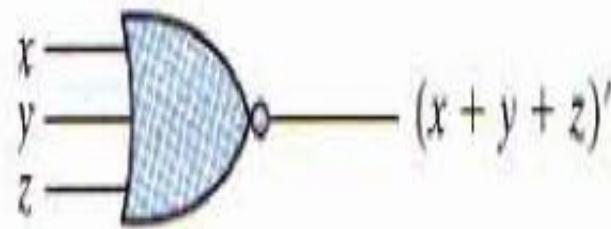


(b) NAND gates

Logic Operations with NOR



Two Graphic Symbols for the NOR Gate



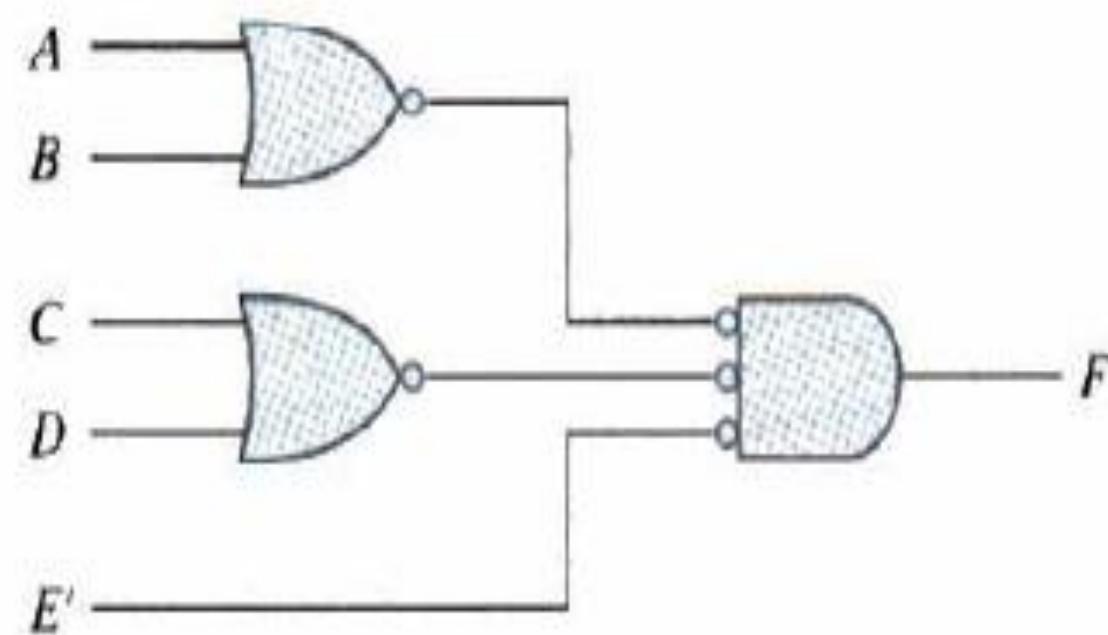
(a) OR-invert



(b) Invert-AND

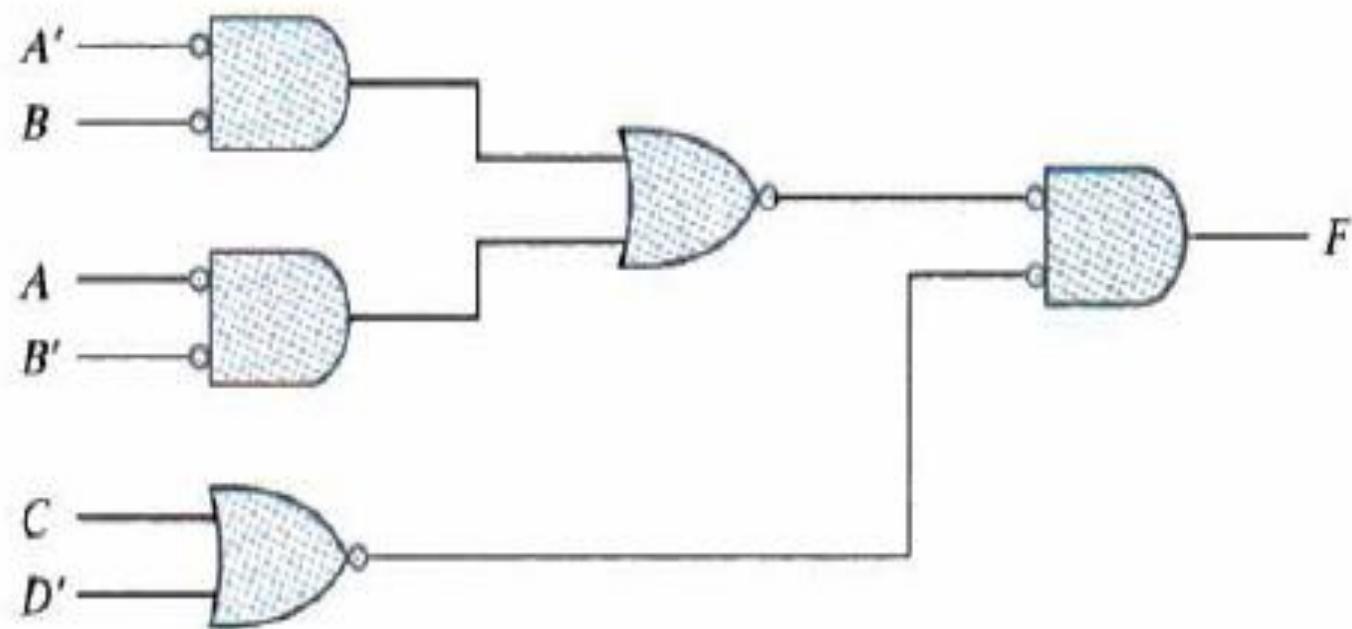
Implementation

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



NOR Implementation

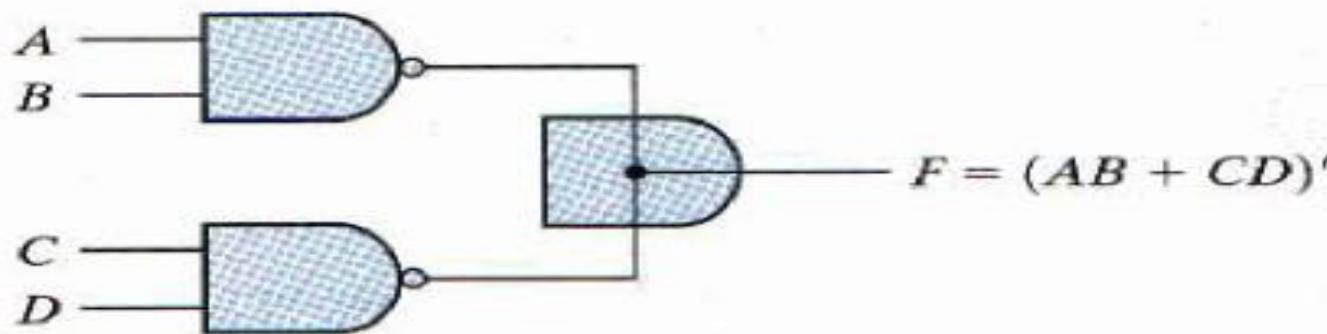
$$F = (A'B' + A'B)(C + D')$$



Other Two Level Implementations

AND-OR-Invert

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



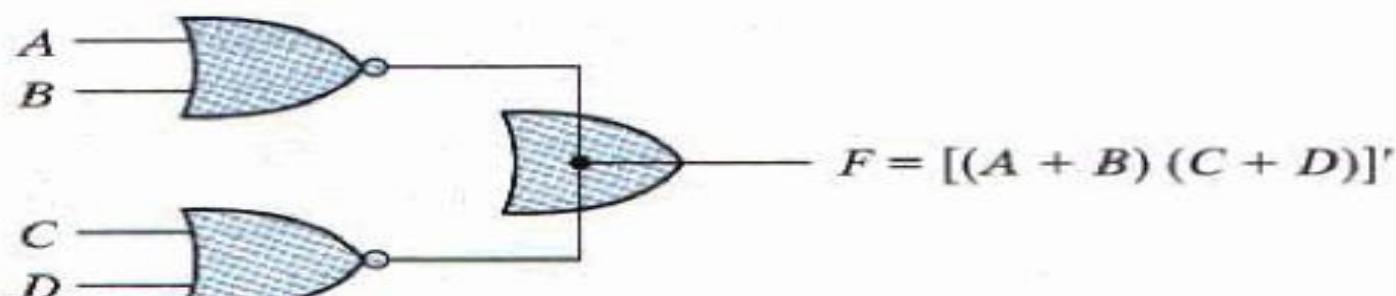
(a) Wired-AND in open-collector
TTL NAND gates.

(AND-OR-INVERT)

Other Two Level Implementations

AND-OR-Invert

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



(b) Wired-OR in ECL gates

(OR-AND-INVERT)

Other Two Level Implementations

Nondegenerate Forms

AND-OR

NAND-NAND

NOR-OR

OR-NAND

OR-AND

NOR-NOR

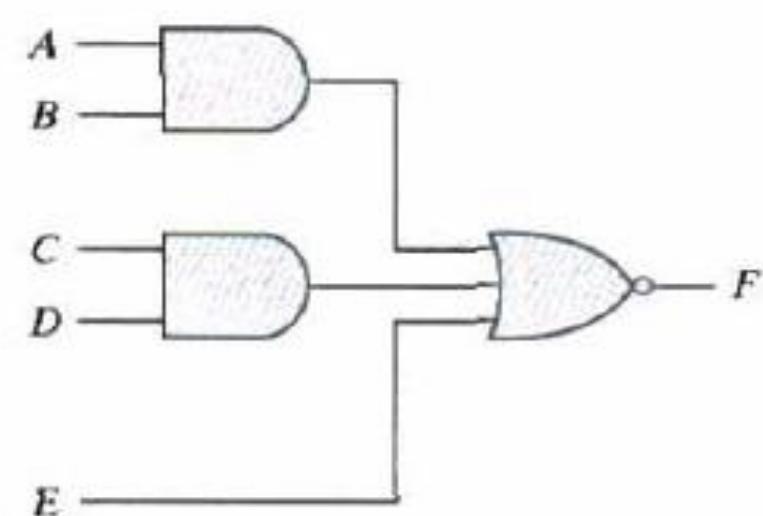
NAND-AND

AND-NOR

Other Two Level Implementations

AND-OR-Invert

$$F = (\overline{A} \overline{B} + \overline{C} \overline{D} + \overline{E})'$$

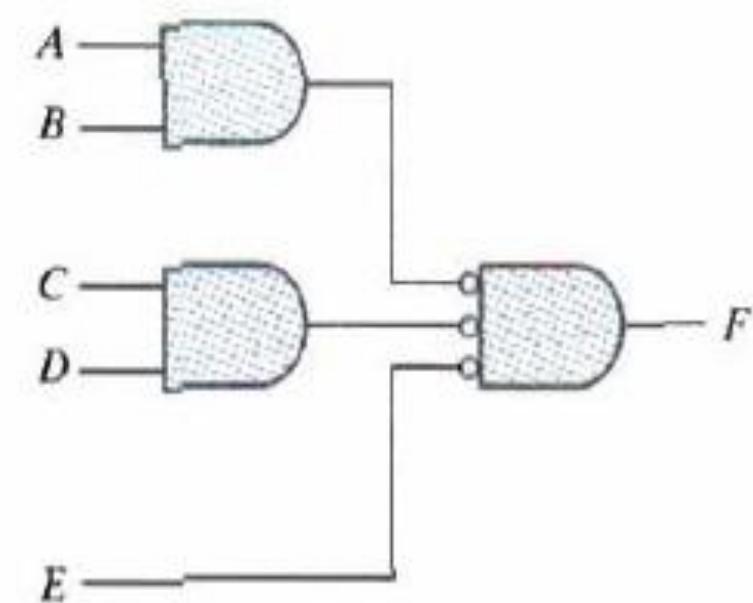


(a) AND-NOR

Other Two Level Implementations

AND-OR-Invert

$$F = (A \cdot B + C \cdot D + E)'$$

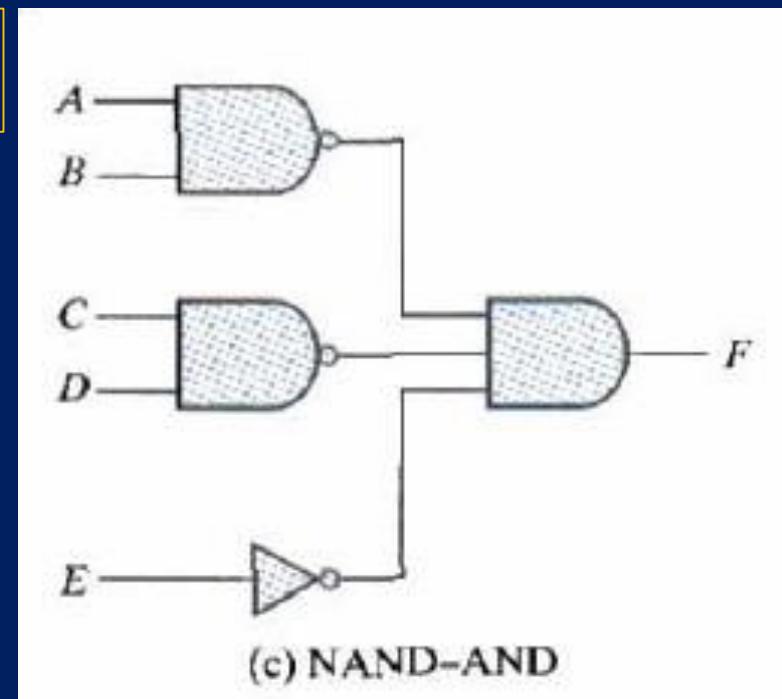


(b) AND-NOR

Other Two Level Implementations

AND-OR-Invert

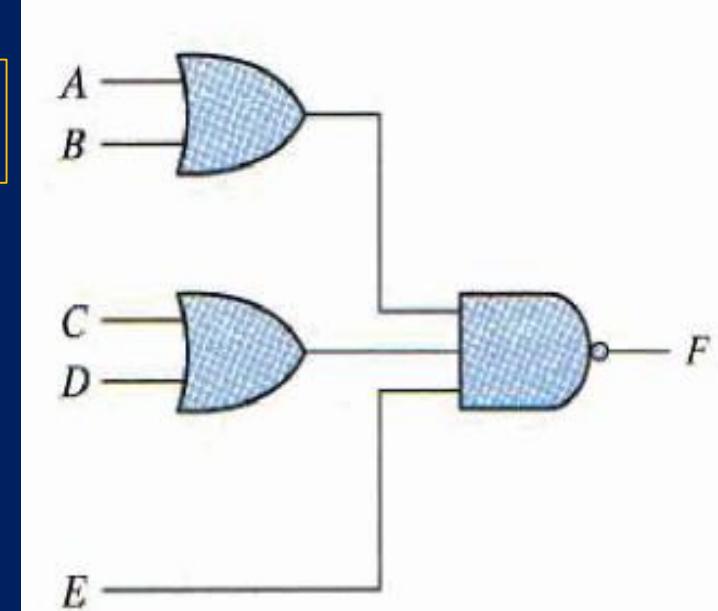
$$F = (A \cdot B + C \cdot D + E)'$$



Other Two Level Implementations

OR-AND-Invert

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

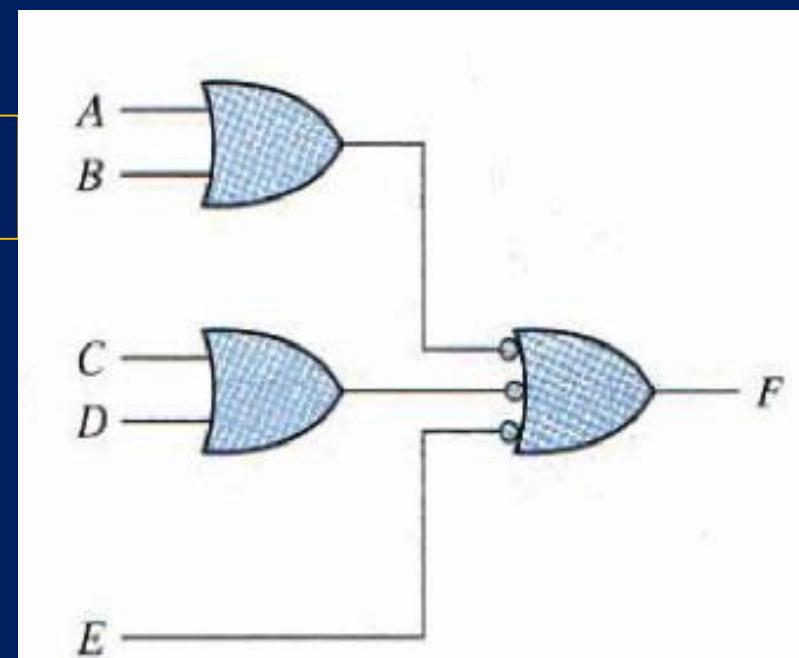


(a) OR-NAND

Other Two Level Implementations

OR-AND-Invert

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

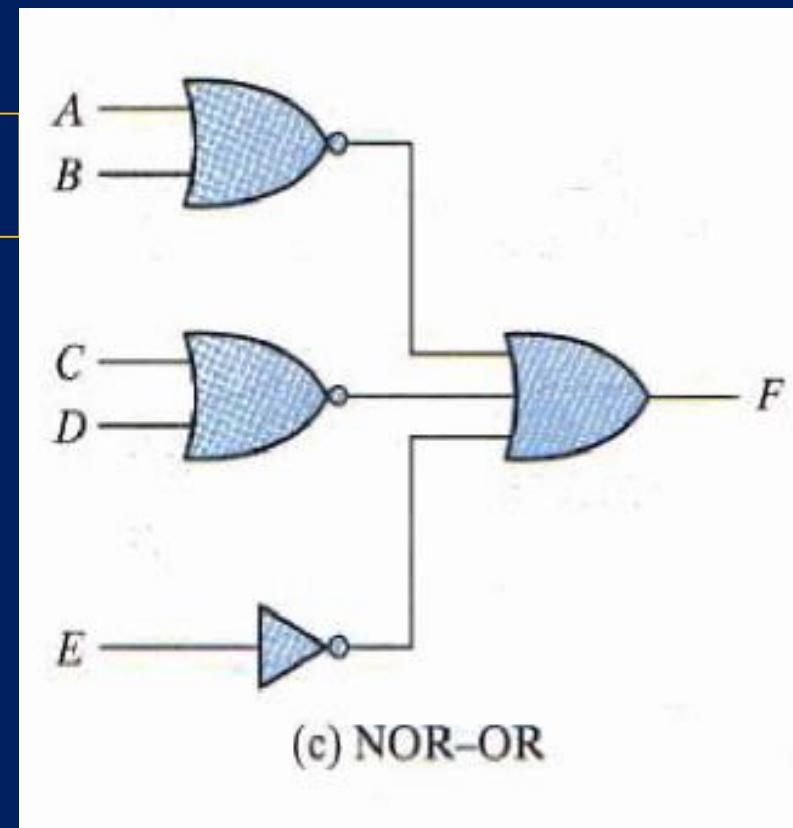


(b) OR-NAND

Other Two Level Implementations

OR-AND-Invert

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



Implementation with Other Two- Level Forms

Equivalent
Non-degenerate
Form

a b

Implements
the Function

Simplify the
function into

AND-NOR

NAND-AND

AND -OR -INVERT

Sum of Products
0's in the map

OR-NAND

NOR-OR

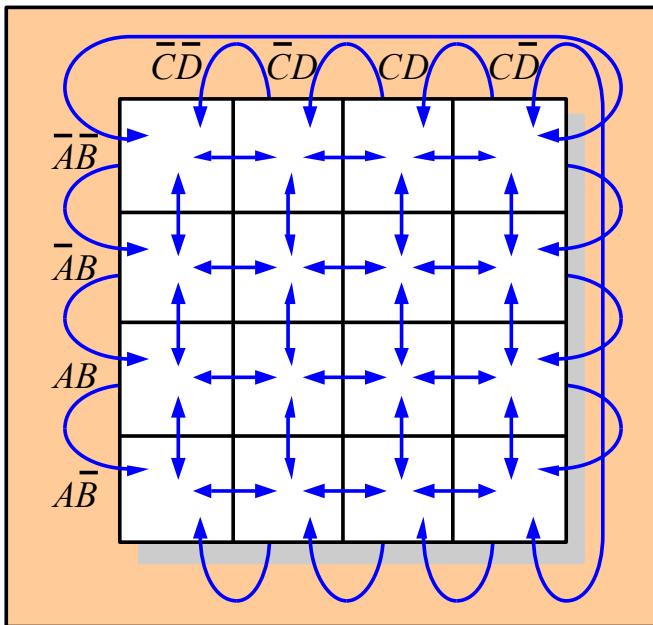
OR –AND -INVERT

Product of sums
1's in the map
and complement

Summary

Karnaugh maps

A 4-variable map has an adjacent cell on each of its four boundaries as shown.



Each cell is different only by one variable from an adjacent cell.

Grouping follows the rules given in the text.

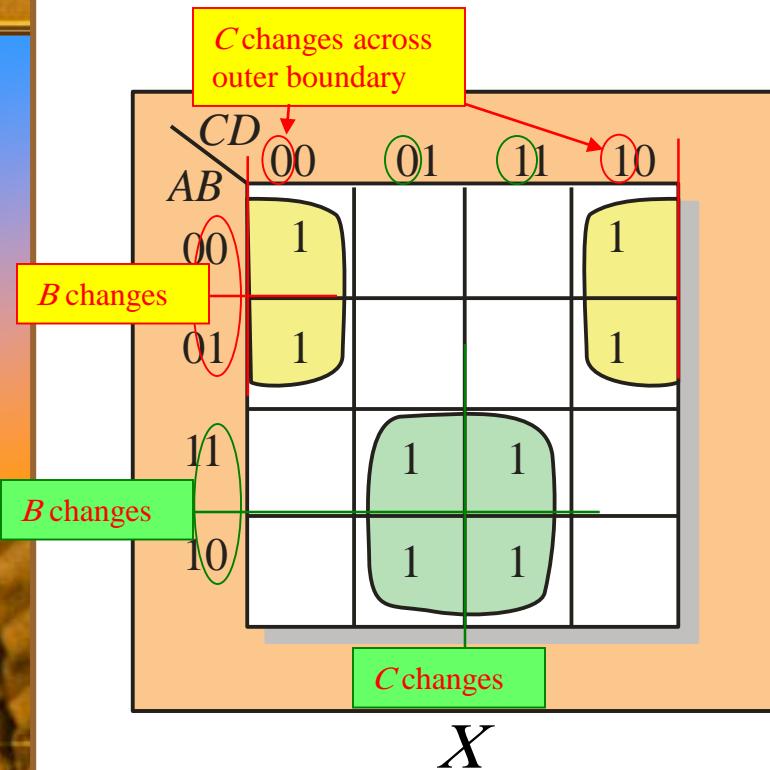
The following slide shows an example of reading a four variable map using binary numbers for the variables...

Summary

Karnaugh maps

Example

Group the 1's on the map and read the minimum logic.



Solution

1. Group the 1's into two separate groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The upper (yellow) group is read as \overline{AD} .
4. The lower (green) group is read as AD .

$$X = \overline{AD} + AD$$

Example 4--22

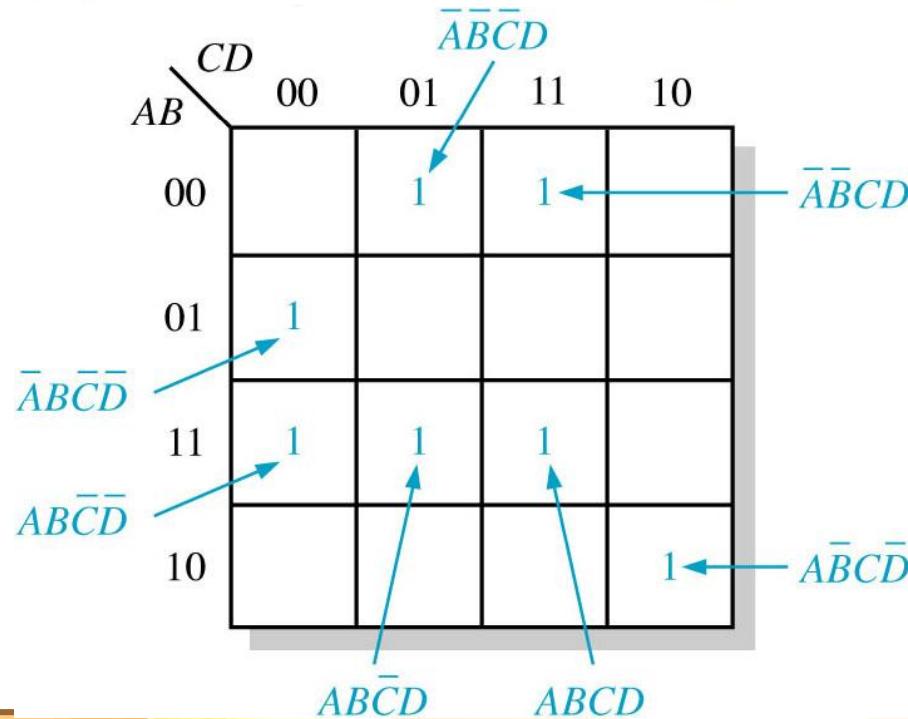


Map the following standard SOP expression on a Karnaugh map:

$$\bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + ABCD + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}CD$$

Solution The expression is evaluated as shown below. A 1 is placed on the 4-variable Karnaugh map in Figure 4–25 for each standard product term in the expression.

$$\begin{array}{ccccccccc} \bar{A}\bar{B}CD & + & \bar{A}\bar{B}\bar{C}\bar{D} & + & A\bar{B}\bar{C}D & + & ABCD & + & A\bar{B}\bar{C}\bar{D} \\ 0011 & & 0100 & & 1101 & & 1111 & & 1100 \\ & & & & & & & & 0001 \\ & & & & & & & & 1010 \end{array}$$



Example 4--24

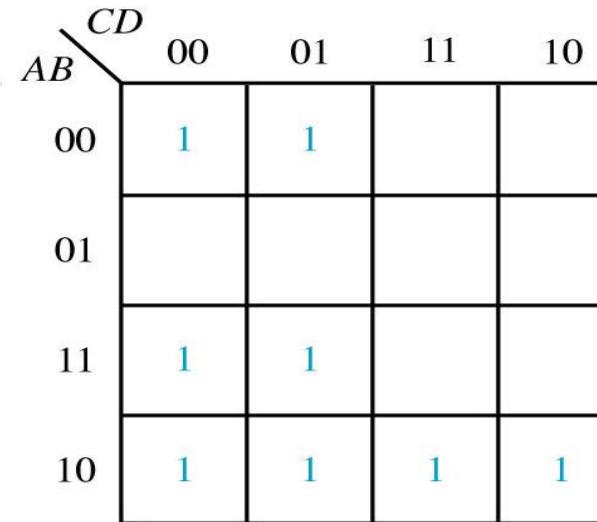


Map the following SOP expression on a Karnaugh map:

$$\bar{B}\bar{C} + A\bar{B} + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}CD$$

Solution The SOP expression is obviously not in standard form because each product term does not have four variables. The first and second terms are both missing two variables, the third term is missing one variable, and the rest of the terms are standard. First expand the terms by including all combinations of the missing variables numerically as follows:

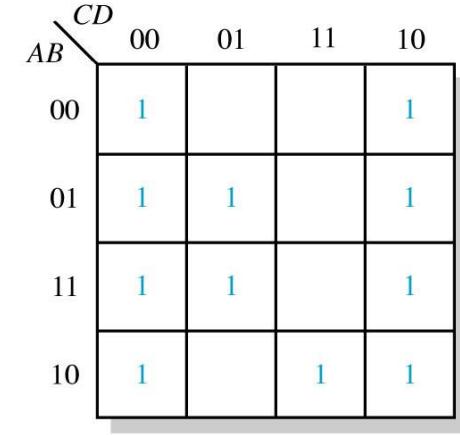
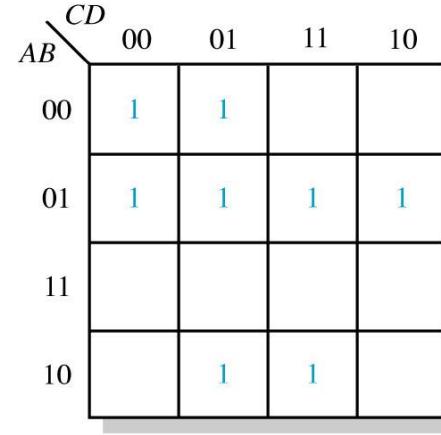
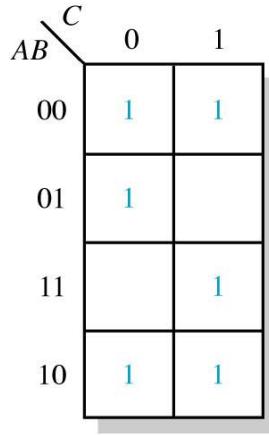
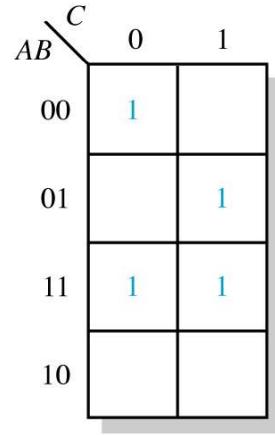
$$\begin{array}{llllll} \bar{B}\bar{C} & + A\bar{B} & + A\bar{B}\bar{C} & + \bar{A}\bar{B}\bar{C}\bar{D} & + \bar{A}\bar{B}\bar{C}D & + A\bar{B}CD \\ 0000 & 1000 & 1100 & 1010 & 0001 & 1011 \\ 0001 & 1001 & 1101 & & & \\ 1000 & 1010 & & & & \\ 1001 & 1011 & & & & \end{array}$$



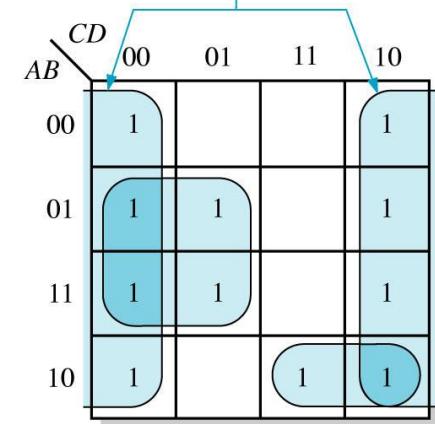
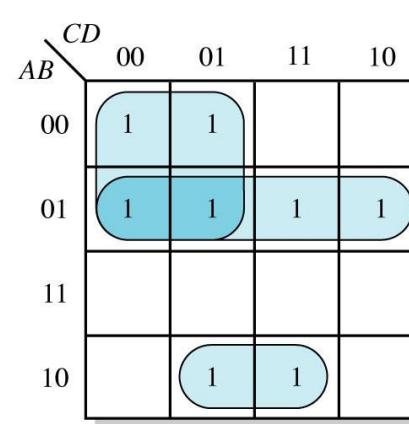
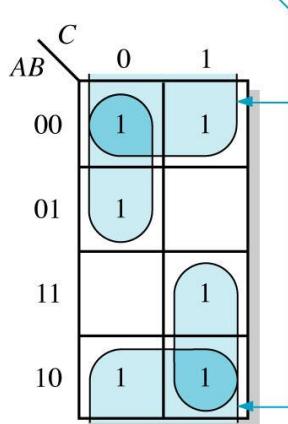
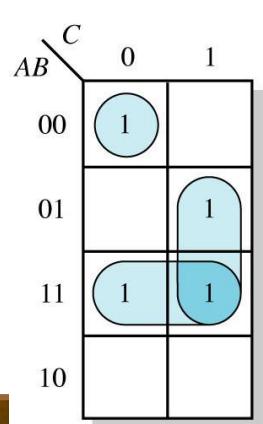


Karnaugh Map Simplification of SOP Expressions

Example 4—25 Group the 1s in each Karnaugh maps



Wrap-around adjacency



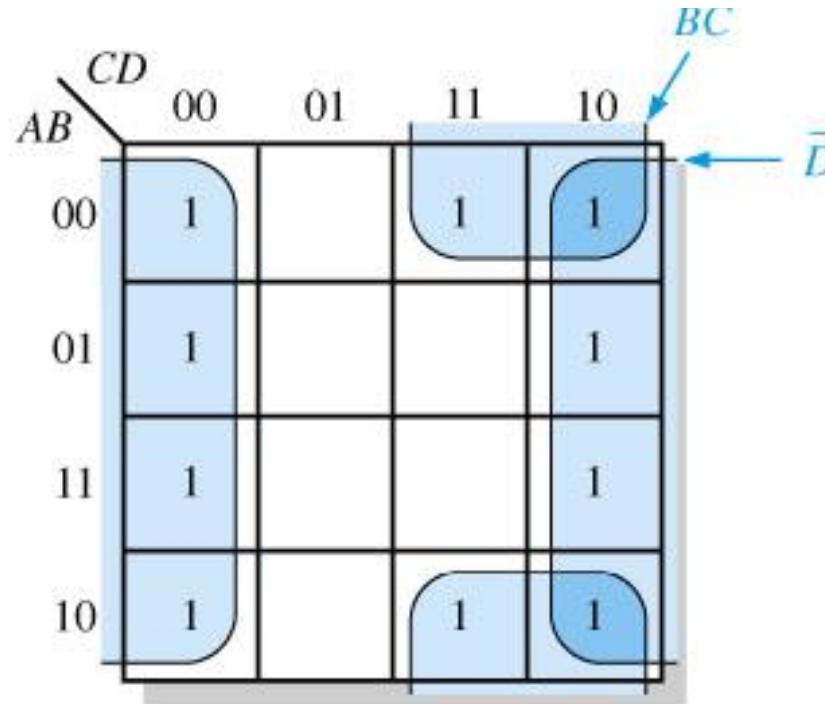
Example 4--29



Use a Karnaugh map to minimize the following SOP expression:

$$\overline{BCD} + \overline{ABC}\overline{D} + A\overline{BC}\overline{D} + \overline{A}\overline{B}CD + A\overline{B}CD + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{ABC}\overline{D} + ABC\overline{D} + A\overline{BC}\overline{D}$$

Solution The first term \overline{BCD} must be expanded into $A\overline{BC}\overline{D}$ and $\overline{ABC}\overline{D}$ to get the standard SOP expression, which is then mapped; and the cells are grouped as shown in Figure 4–33.



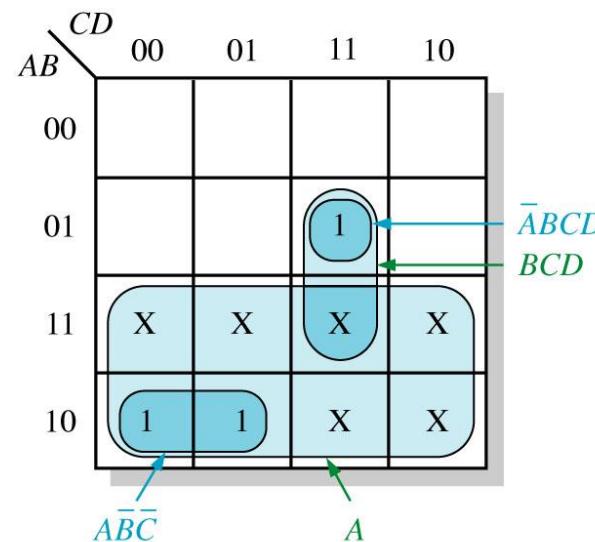
Summary

“Don’t Care” Conditions

Inputs	Output
$A\ B\ C\ D$	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	1
1 0 0 0	1
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

(a) Truth table

Don't cares



(b) Without “don’t cares” $Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}CD$
With “don’t cares” $Y = A + BCD$

Summary

Finding minimal SOP/POS from Boolean Expression

- $F(w,x,y,z) = w'x' + yz' + wxy + w'x'yz' + wxy'z + w'y'z'$

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

- Minimal SOP: $F = w'x' + yz' + w'z' + wxz$

Summary

Practice: Don't Care

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

NOT equivalent,
but both valid.

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

Quiz

1. Adjacent cells on a Karnaugh map differ from each other by
 - a. one variable
 - b. two variables
 - c. three variables
 - d. answer depends on the size of the map

Quiz

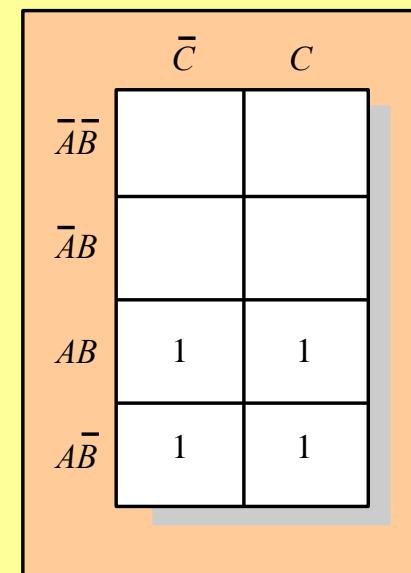
2. The minimum expression that can be read from the Karnaugh map shown is

a. $X = A$

b. $X = \bar{A}$

c. $X = B$

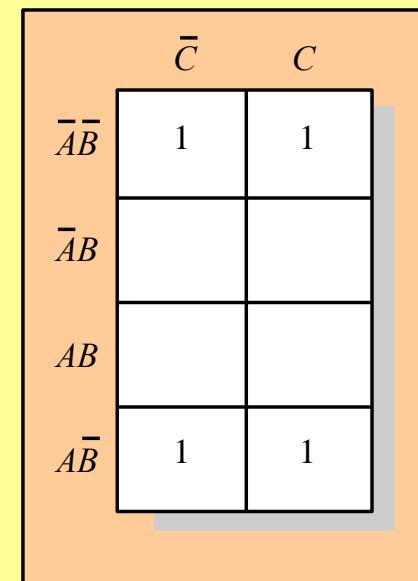
d. $X = \bar{B}$



Quiz

3. The minimum expression that can be read from the Karnaugh map shown is

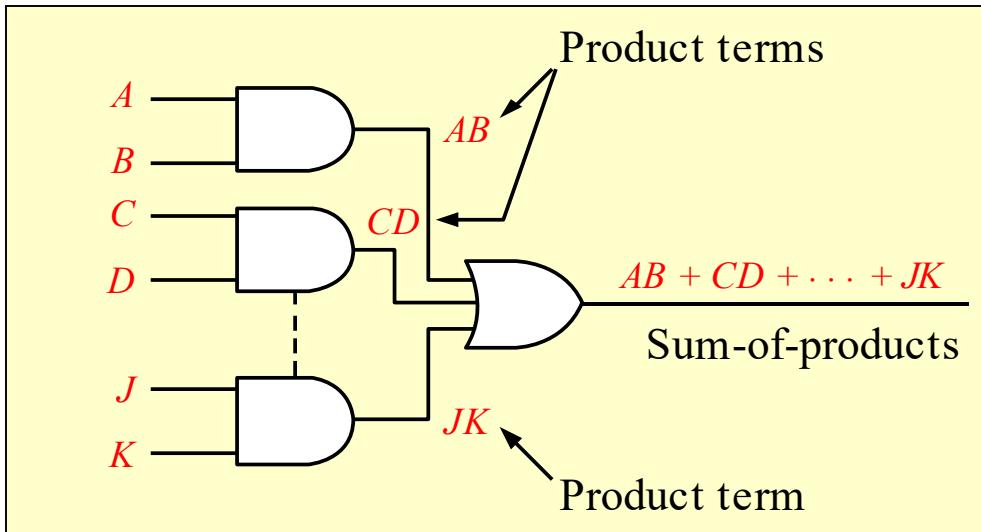
- a. $X = A$
- b. $X = \bar{A}$
- c. $X = B$
- d. $X = \bar{B}$



Summary

Combinational Logic Circuits (AND-OR logic)

In Sum-of-Products (SOP) form, basic combinational circuits can be directly implemented with **AND-OR** combinations if the necessary complement terms are available.

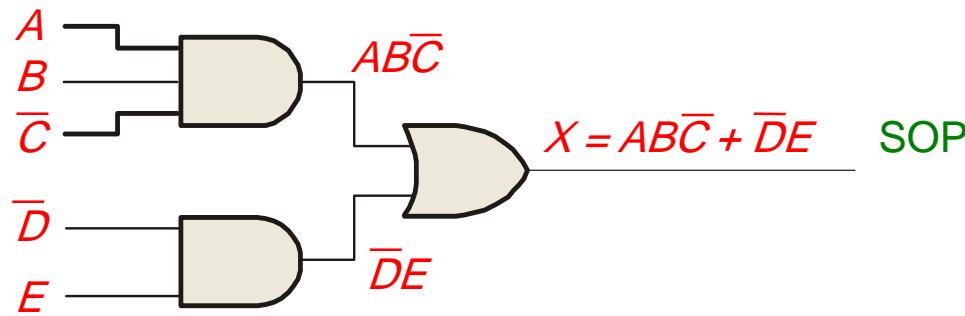


Summary

Combinational Logic Circuits (AND-OR logic)

An example of an SOP implementation is shown. The SOP expression is an AND-OR combination of the input variables and the appropriate complements.

$$X = A\bar{B}\bar{C} + \bar{D}\bar{E}$$



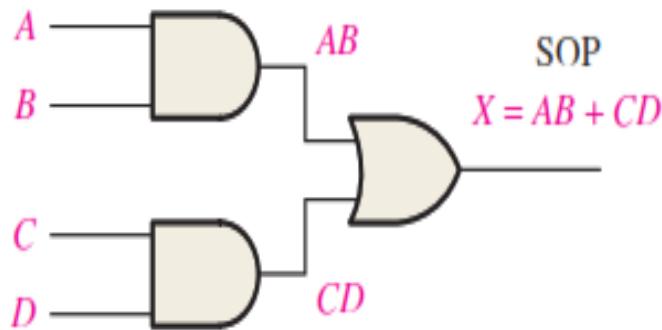
Summary

Combinational Logic Circuits (AND-OR logic)

This figure shows an AND-OR circuit consisting of two 2-input AND gates and one 2-input OR gate

TABLE 5-1

Truth table for the AND-OR logic in Figure 5-1.



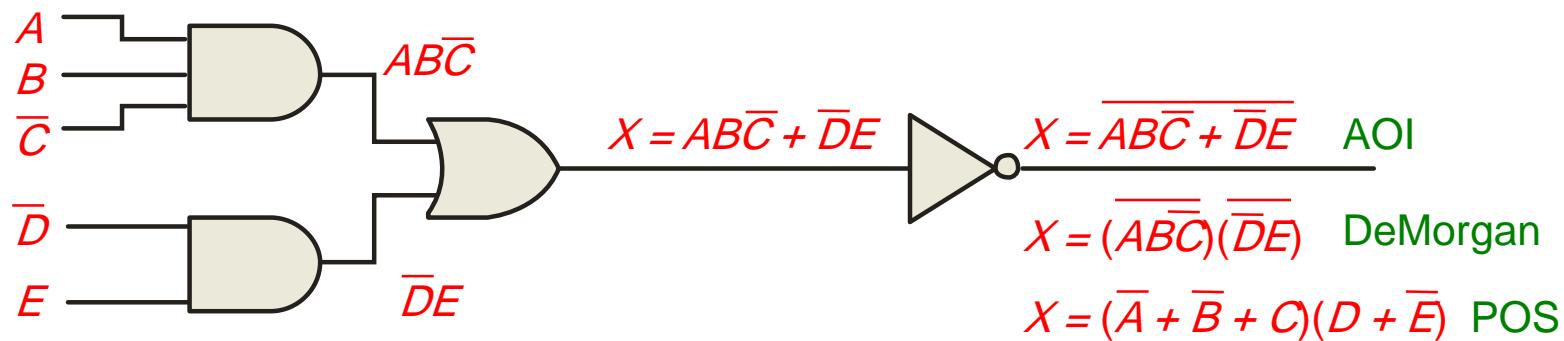
Inputs				AB	CD	Output X
A	B	C	D			
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

Summary

Combinational Logic Circuits (AND-OR-Inverter)

When the output of a SOP form is inverted, the circuit is called an **AND-OR-Invert** circuit. The AOI configuration lends itself to product-of-sums (POS) implementation.

An example of an AOI implementation is shown. The output expression can be changed to a POS expression by applying DeMorgan's theorem twice.



Summary

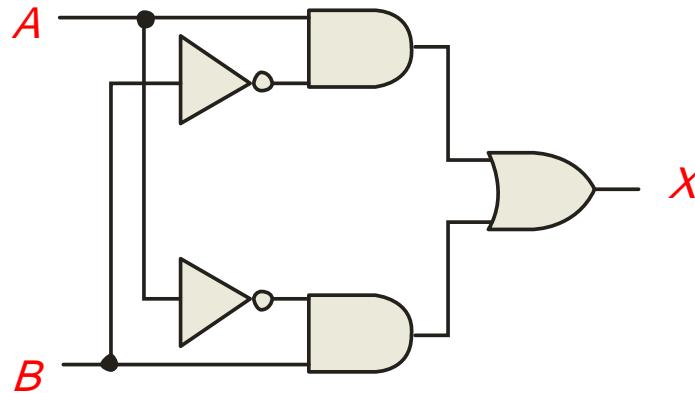
Exclusive-OR Logic

The truth table for an exclusive-OR gate is

Notice that the output is HIGH whenever A and B disagree.

The Boolean expression is $X = \bar{A}B + A\bar{B}$

The circuit can be drawn as

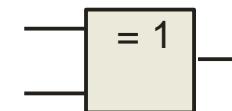


Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Symbols:



Distinctive shape



Rectangular outline

Summary

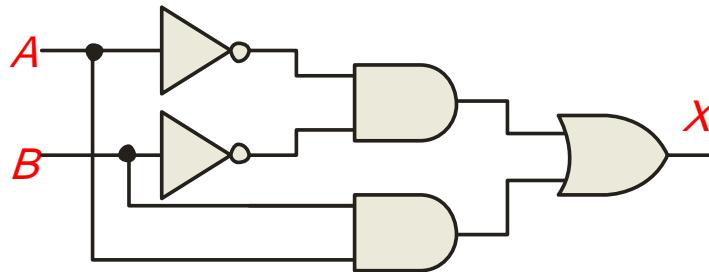
Exclusive-NOR Logic

The truth table for an exclusive-NOR gate is

Notice that the output is HIGH whenever *A* and *B* agree.

The Boolean expression is $X = \bar{A}\bar{B} + AB$

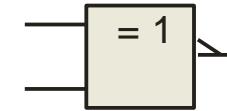
The circuit can be drawn as



Symbols:



Distinctive shape

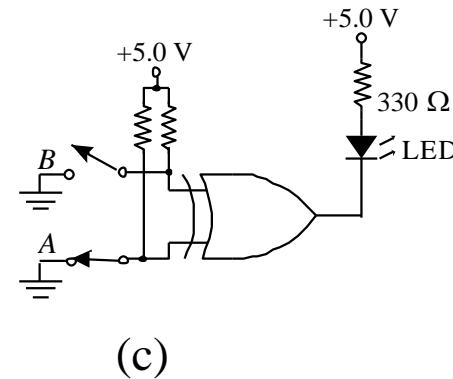
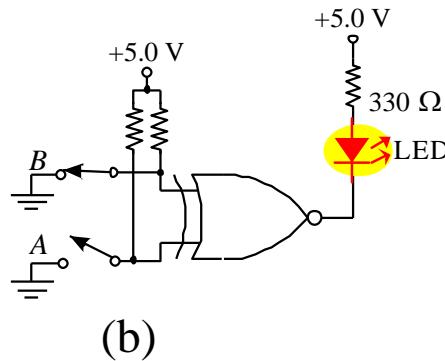
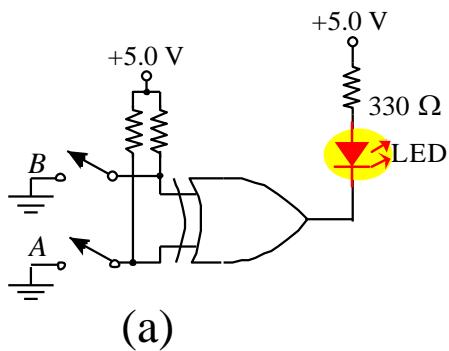


Rectangular outline

Summary

Example

For each circuit, determine if the LED should be on or off.



Solution

Circuit (a): XOR, inputs agree, output is LOW, LED is ON.

Circuit (b): XNOR, inputs disagree, output is LOW, LED is ON.

Circuit (c): XOR, inputs disagree, output is HIGH, LED is OFF.

Summary

Implementing Combinational Logic

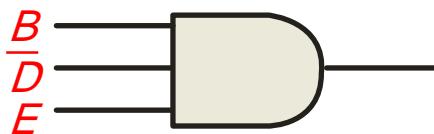
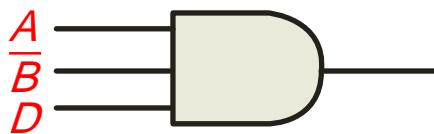
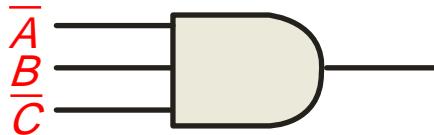
Implementing a SOP expression is done by first forming the AND terms; then the terms are ORed together.

Example

Show the circuit that will implement the Boolean expression $X = \overline{ABC} + \overline{ABD} + \overline{BDE}$. (Assume that the variables and their complements are available.)

Solution

Start by forming the terms using three 3-input AND gates. Then combine the three terms using a 3-input OR gate.



$$X = \overline{ABC} + \overline{ABD} + \overline{BDE}$$

Summary

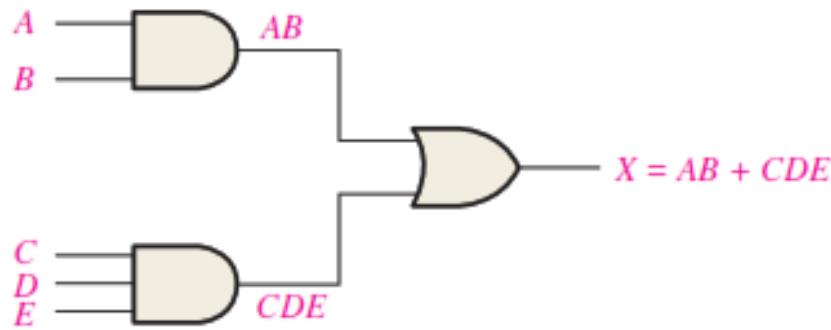
From a Boolean Expression to a Logic Circuit

□ Implementing Combinational Logic

$$X = AB + CDE$$

$$X = \boxed{AB} + \boxed{CDE}$$

AND OR

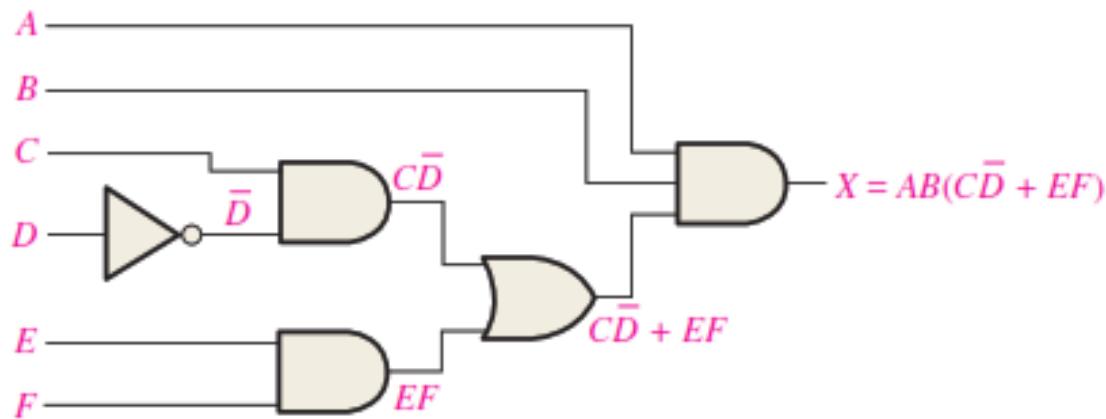
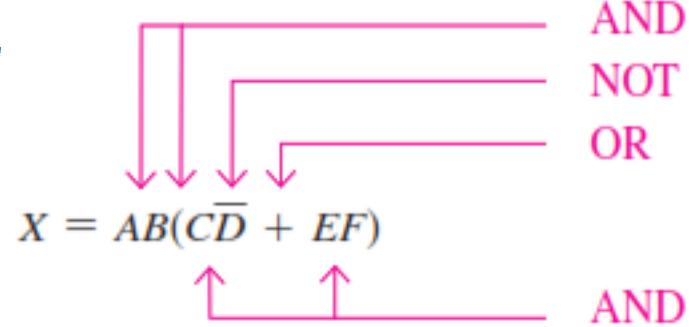


Summary

From a Boolean Expression to a Logic Circuit

The logic gates required to implement $X = AB(\bar{CD} + EF)$ are as follows:

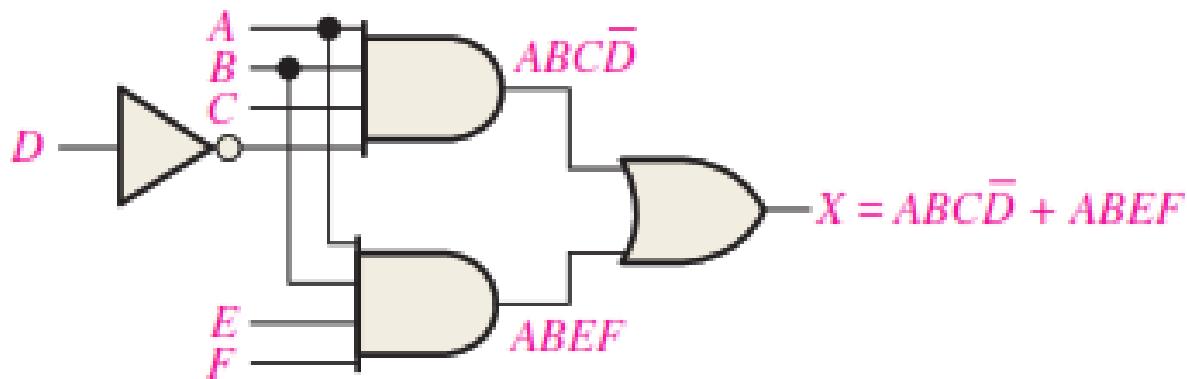
1. One inverter to form \bar{D}
2. Two 2-input AND gates to form \bar{CD} and EF
3. One 2-input OR gate to form $\bar{CD} + EF$
4. One 3-input AND gate to form X



Summary

From a Boolean Expression to a Logic Circuit

Logic circuits for $X = AB(C\bar{D} + EF) = ABC\bar{D} + ABEF$.



Summary

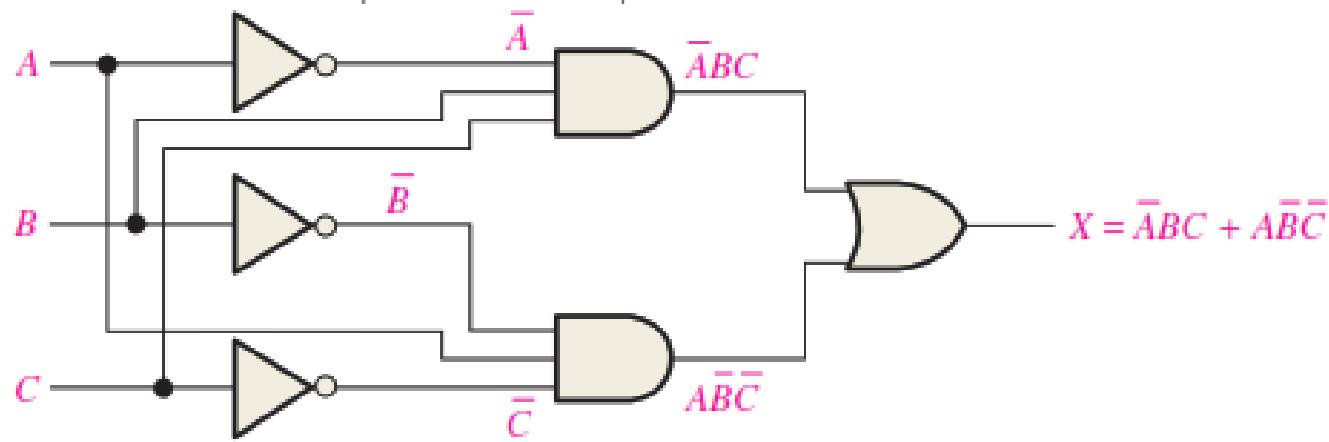
From a Truth Table to a Logic Circuit

☐ Implementing Combinational Logic

TABLE 5-3

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{A}BC$
1	0	0	1	$A\bar{B}\bar{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

$$X = \bar{A}BC + A\bar{B}\bar{C}$$

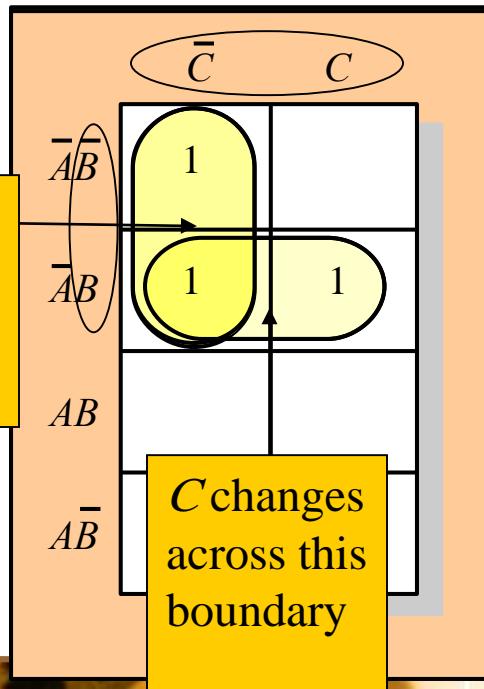


Summary

Karnaugh Map Implementation

For basic combinational logic circuits, the Karnaugh map can be read and the circuit drawn as a minimum SOP.

Example A Karnaugh map is drawn from a truth table. Read the minimum SOP expression and draw the circuit.



Solution

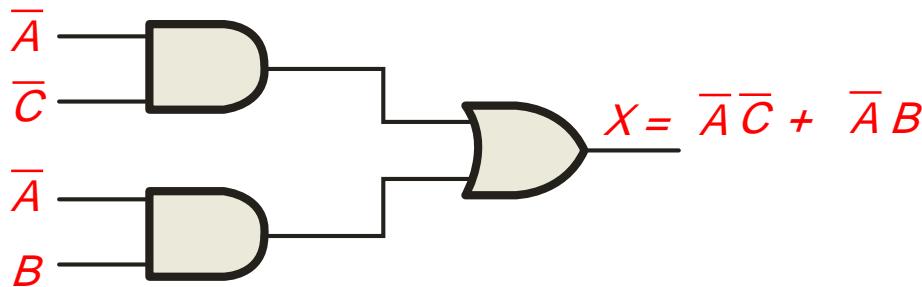
1. Group the 1's into two overlapping groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The vertical group is read $\bar{A}\bar{C}$.
4. The horizontal group is read $\bar{A}B$.

The circuit is on the next slide:

Summary

Solution *continued...*

Circuit:



The result is shown as a sum of products.

It is a simple matter to implement this form using only NAND gates as shown in the text and following example.

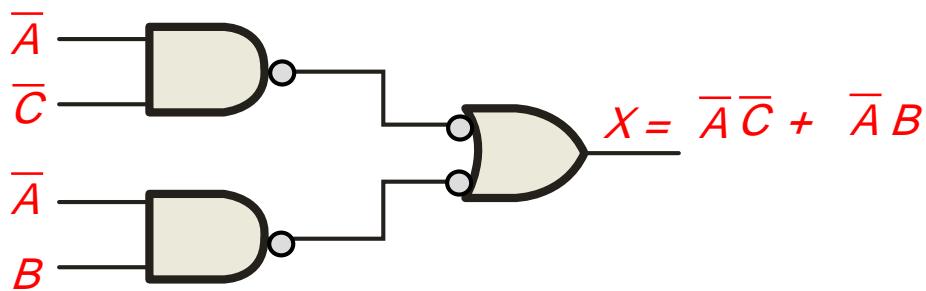
Summary

NAND Logic

Example Convert the circuit in the previous example to one that uses only NAND gates.

Solution

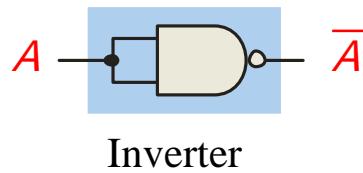
Recall from Boolean algebra that double inversion cancels. By adding inverting bubbles to above circuit, it is easily converted to NAND gates:



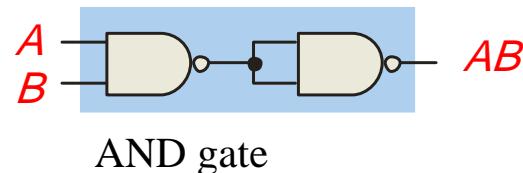
Summary

Universal Gates

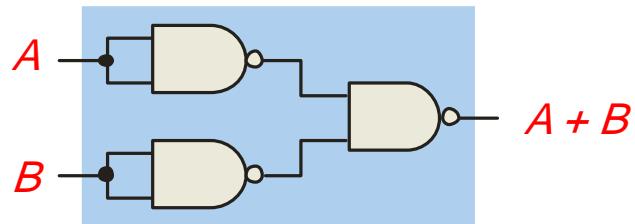
NAND gates are sometimes called **universal** gates because they can be used to produce the other basic Boolean functions.



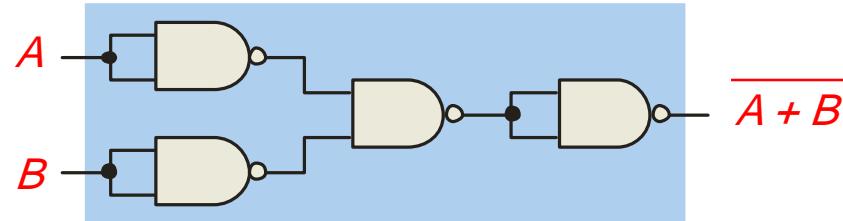
Inverter



AND gate



OR gate

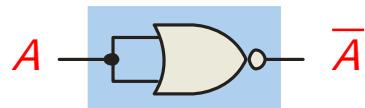


NOR gate

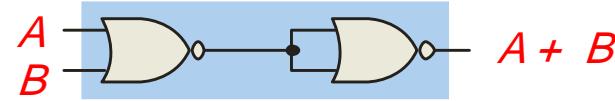
Summary

Universal Gates

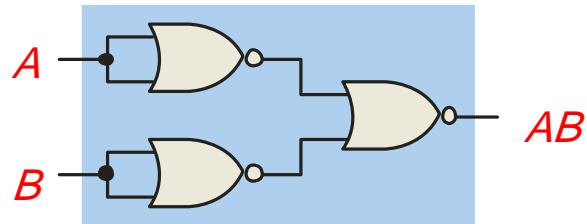
NOR gates are also **universal** gates and can form all of the basic gates.



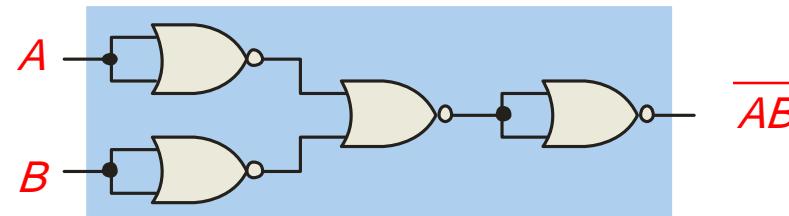
Inverter



OR gate



AND gate

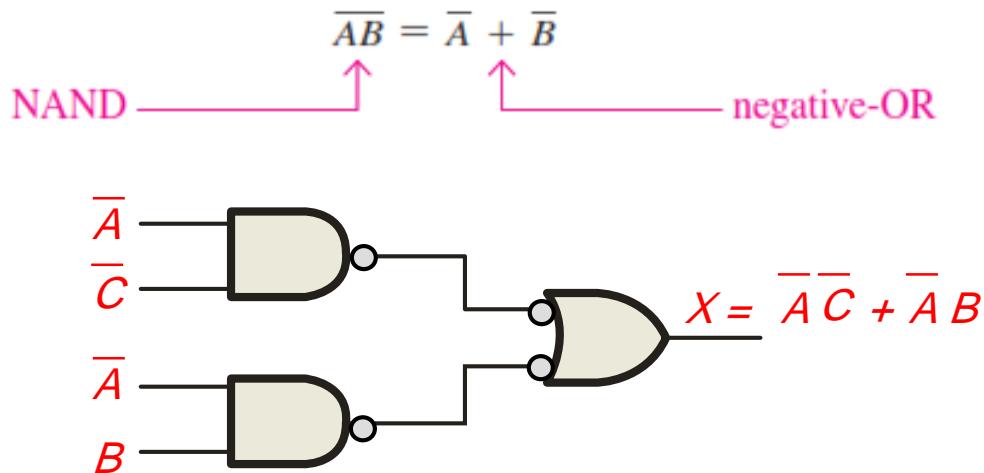


NAND gate

Summary

NAND Logic

Recall from DeMorgan's theorem that $\overline{AB} = \overline{A} + \overline{B}$. By using equivalent symbols, it is simpler to read the logic of SOP forms. The earlier example shows the idea:



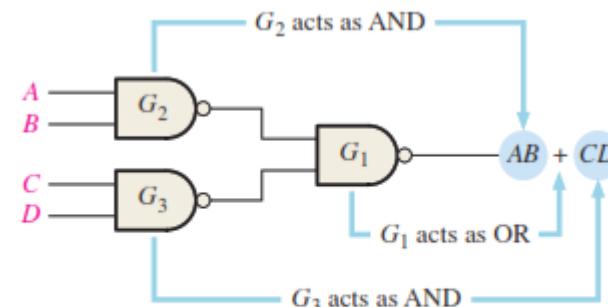
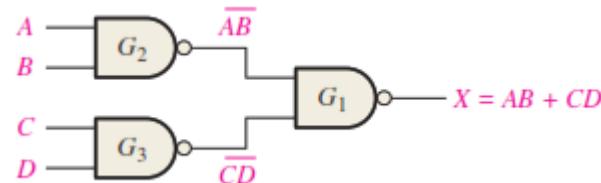
The logic is easy to read if you (mentally) cancel the two connected bubbles on a line.



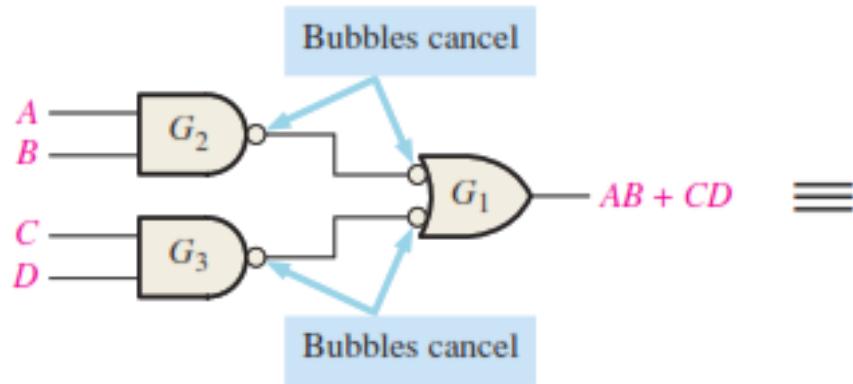
Summary

NAND Logic

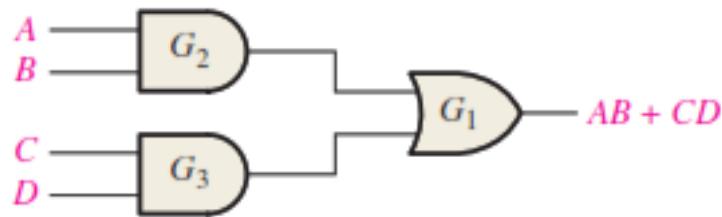
$$\begin{aligned}
 X &= \overline{(AB)(CD)} \\
 &= \overline{(\overline{A} + \overline{B})(\overline{C} + \overline{D})} \\
 &= \overline{\overline{A} + \overline{B}} + \overline{\overline{C} + \overline{D}} \\
 &= \overline{\overline{AB}} + \overline{\overline{CD}} \\
 &= AB + CD
 \end{aligned}$$



(a) Original NAND logic diagram showing effective gate operation relative to the output expression



(b) Equivalent NAND/Negative-OR logic diagram

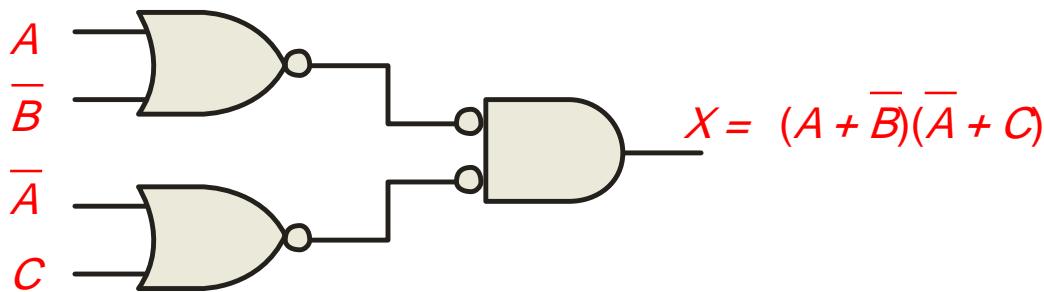


(c) AND-OR equivalent

Summary

NOR Logic

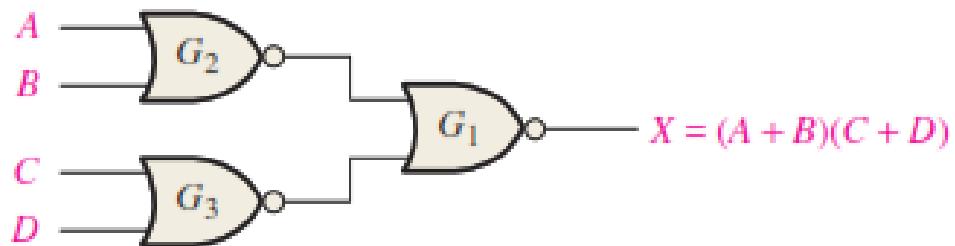
Alternatively, DeMorgan's theorem can be written as $A + B = \overline{AB}$. By using equivalent symbols, it is simpler to read the logic of POS forms. For example,



Again, the logic is easy to read if you cancel the two connected bubbles on a line.

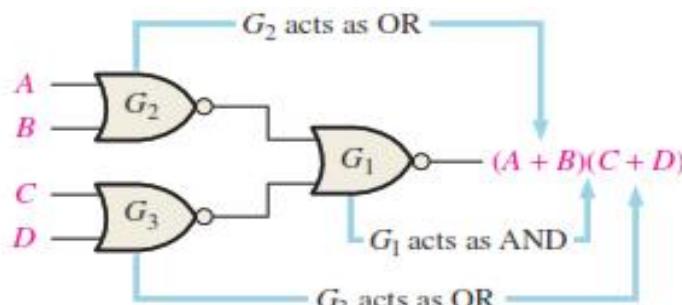
Summary

NOR Logic

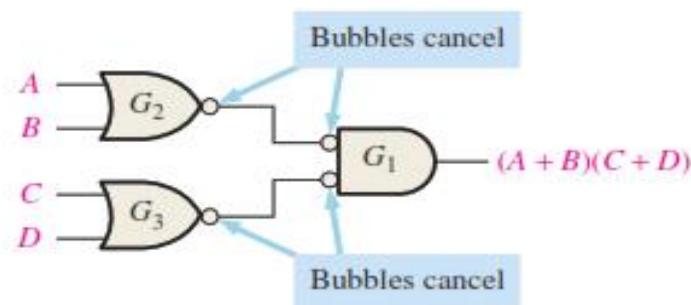


NOR logic for $X = (A + B)(C + D)$.

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



(a)



(b)



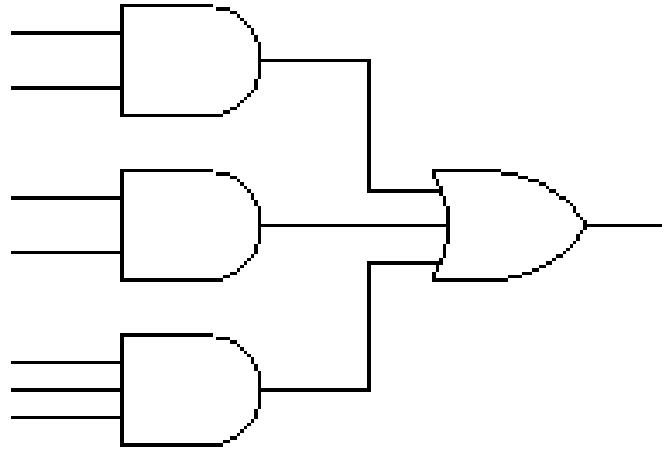
Summary

NAND-NAND / NOR-NOR

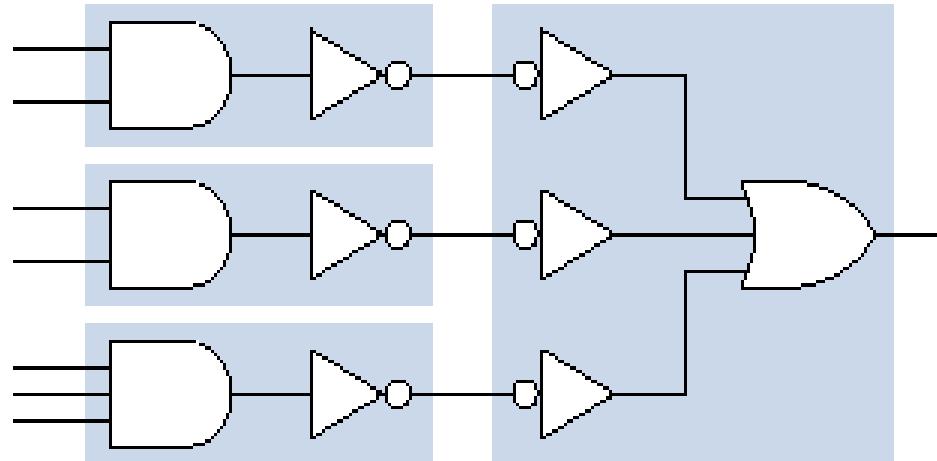
- Any **sum-of-products** expression can be realized in either as an AND-OR circuit or as a **NAND-NAND** circuit.
- Dual statement is also true:
- Any **product-of-sums** expression can be realized in either as an OR-AND circuit or as a **NOR-NOR** circuit.

Summary

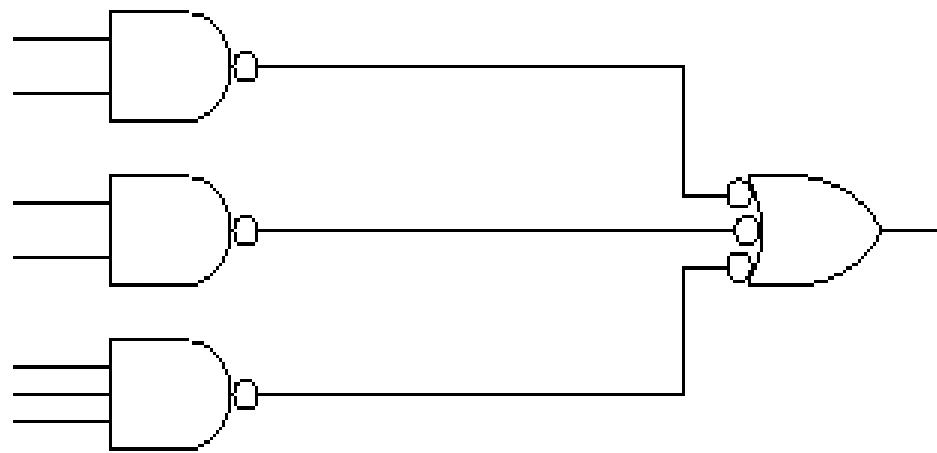
Sum-of-products form



AND-OR

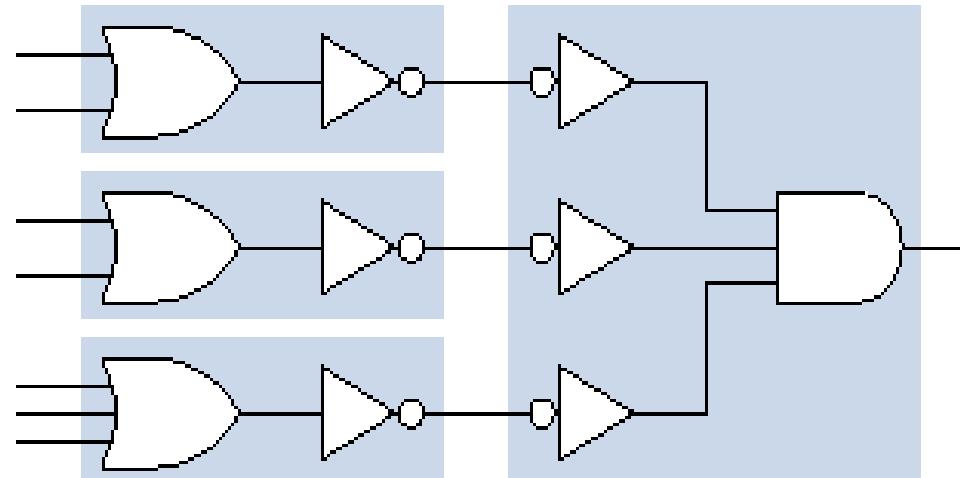
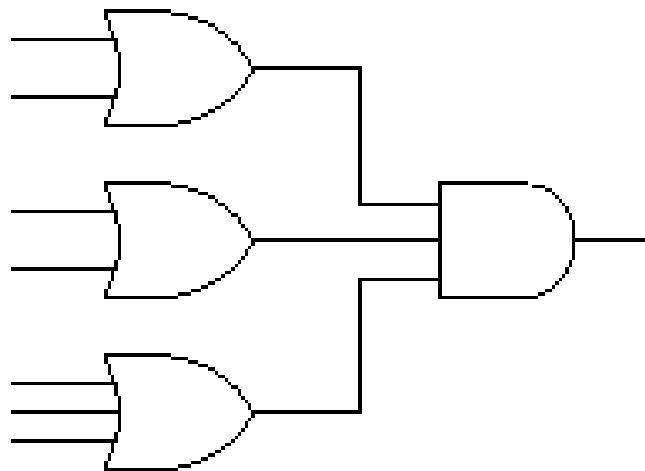


NAND-NAND



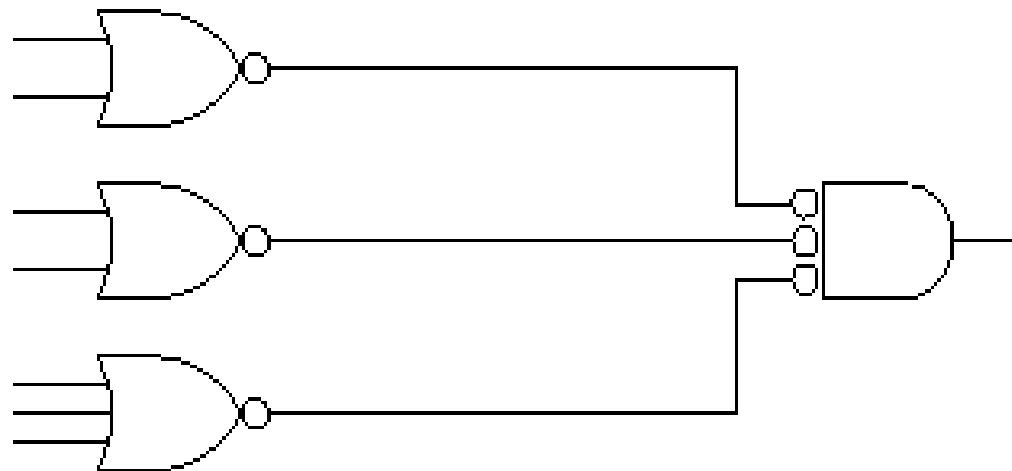
Summary

Product-of-sums form



OR-AND

NOR-NOR



Selected Key Terms

Universal gate

Either a NAND or a NOR gate. The term universal refers to a property of a gate that permits any logic function to be implemented by that gate or by a combination of gates of that kind.

Negative-OR

The dual operation of a NAND gate when the inputs are active-LOW.

Negative-AND

The dual operation of a NOR gate when the inputs are active-LOW.

Quiz

1. Assume an AOI expression is $\overline{AB + CD}$. The equivalent POS expression is
- a. $(A + B)(C + D)$
 - b. $(\overline{A} + \overline{B})(\overline{C} + \overline{D})$
 - c. $(\overline{A + B})(\overline{C + D})$
 - d. none of the above

Quiz

2. The truth table shown is for

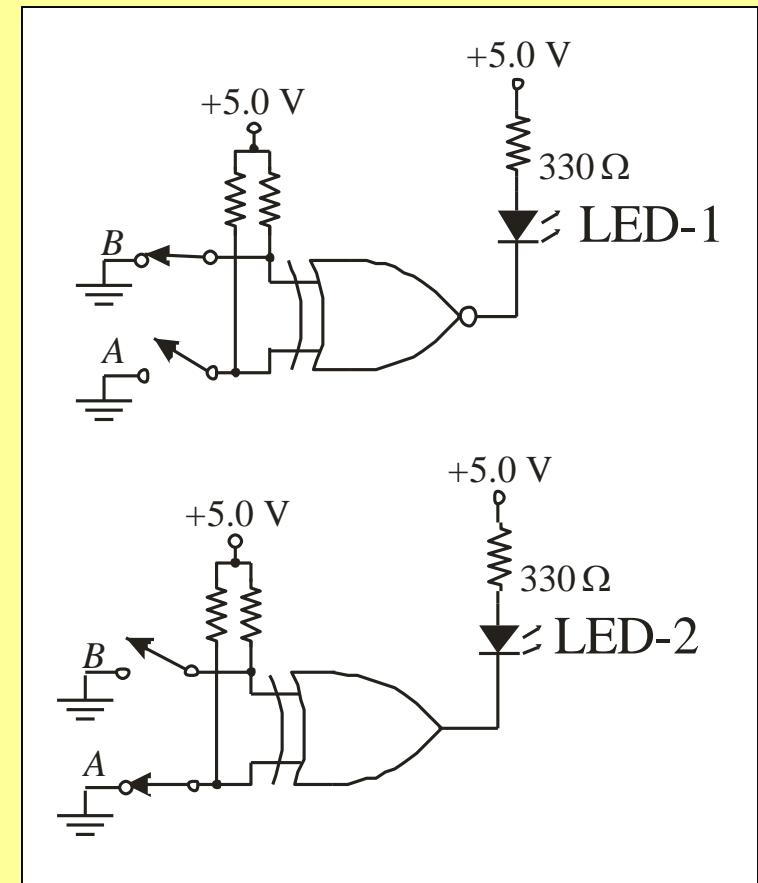
- a. a NAND gate
- b. a NOR gate
- c. an exclusive-OR gate
- d. an exclusive-NOR gate

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Quiz

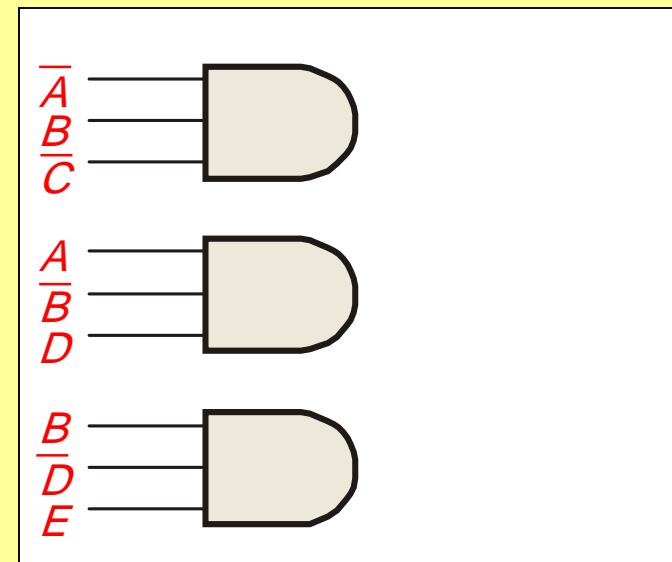
3. An LED that should be ON is

- a. LED-1
- b. LED-2
- c. neither
- d. both



Quiz

4. To implement the SOP expression $X = \bar{A}\bar{B}C + \bar{A}\bar{B}D + B\bar{D}E$ the type of gate that is needed is a
- a. 3-input AND gate
 - b. 3-input NAND gate
 - c. 3-input OR gate
 - d. 3-input NOR gate



Quiz

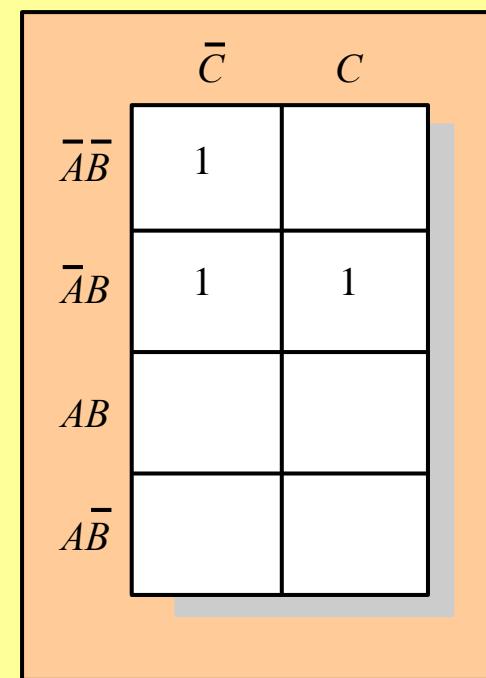
5. Reading the Karnaugh map, the logic expression is

a. $\overline{A}C + \overline{A}B$

b. $\overline{A}B + A\overline{C}$

c. $A\overline{B} + B\overline{C}$

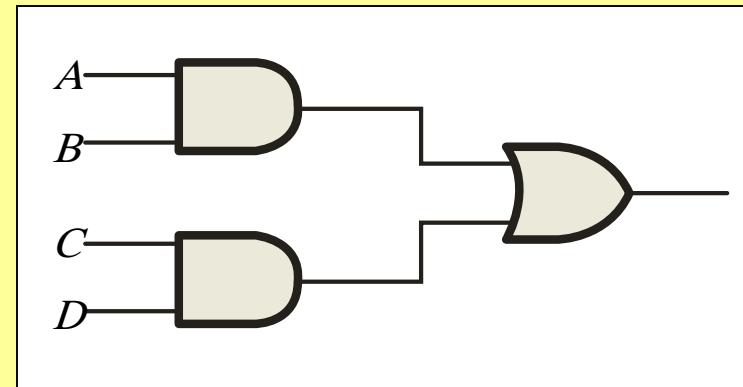
d. $\overline{A}B + \overline{A}\overline{C}$



Quiz

6. The circuit shown will have identical logic out if all gates are changed to

- a. AND gates
- b. OR gates
- c. NAND gates
- d. NOR gates



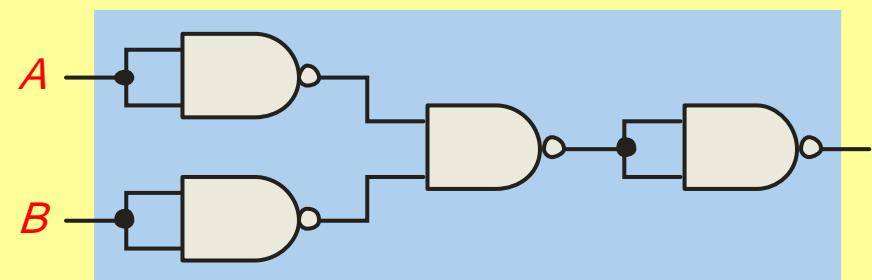
Quiz

7. The two types of gates which are called *universal gates* are
- a. AND/OR
 - b. NAND/NOR
 - c. AND/NAND
 - d. OR/NOR

Quiz

8. The circuit shown is equivalent to an

- a. AND gate
- b. XOR gate
- c. NOR gate
- d. none of the above



Quiz

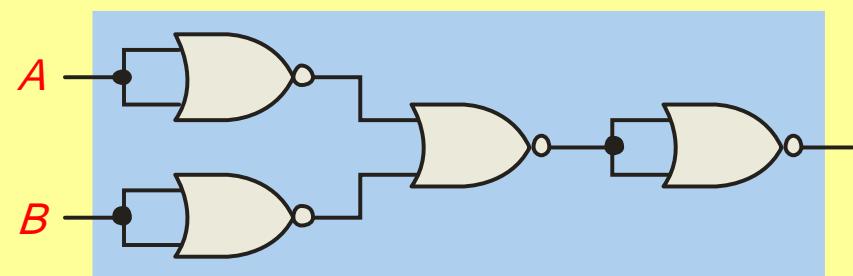
9. The circuit shown is equivalent to

a. NAND gate

b. XOR gate

c. OR gate

d. none of the above



Quiz

Answers:

- | | |
|------|------|
| 1. b | 6. c |
| 2. d | 7. b |
| 3. a | 8. c |
| 4. c | 9. a |
| 5. d | |

Chapter 6

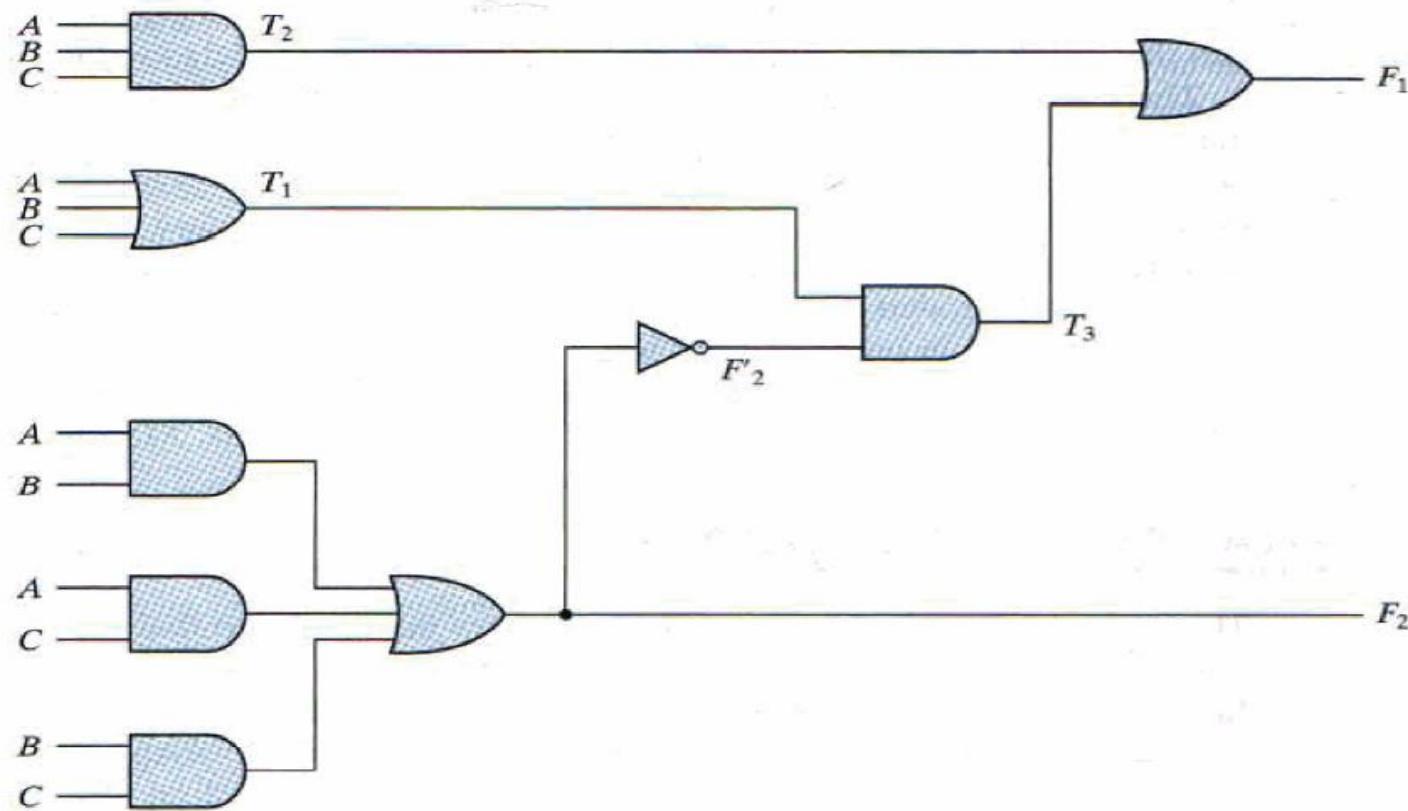
INTRODUCTION

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of current inputs without regard to past inputs.
- A sequential circuit employs memory elements in addition to logic gates (i.e. output is function of present and previous inputs).

Block diagram of a combinational circuit



Combinational Logic



Combinational Logic

$$F2 = AB + AC + BC$$

$$T1 = A + B + C$$

$$T2 = ABC$$

$$T3 = F2 \cdot T1$$

$$F1 = T3 + T2$$

$$F1 = A' B' C' + A' B' C + A B' C' + A B C$$

Combinational Logic

To obtain F_1 as a function of A , B , and C , we form a series of substitutions as follows:

$$\begin{aligned}F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\&= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\&= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\&= A'BC' + A'B'C + AB'C' + ABC\end{aligned}$$

Combinational Logic

Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Combinational Logic

1. Analyze the circuit
2. Determine the number of inputs and outputs and assign a symbol to each.
3. Draw the truth table
4. Obtain the simplified Boolean expressions required
5. Draw the logic diagram and verify correctness either manually or from simulations.

Conversion from BCD to Excess-3

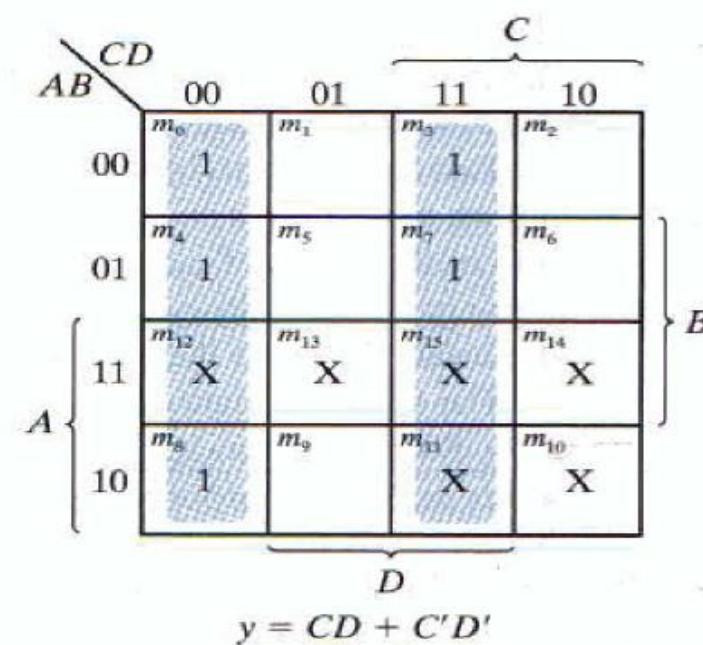
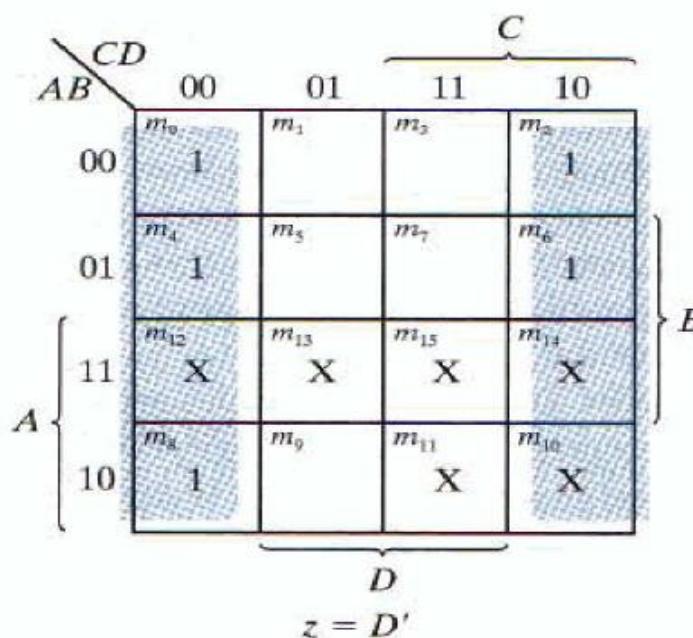
Code Conversion

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

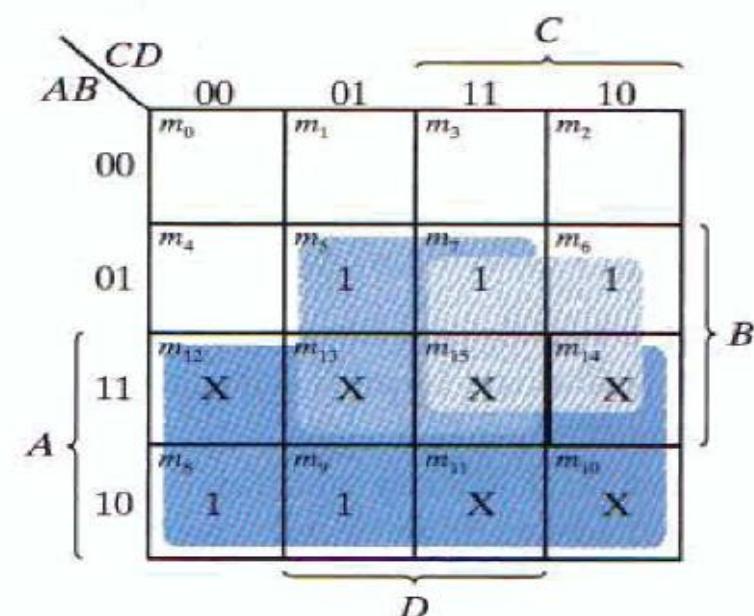
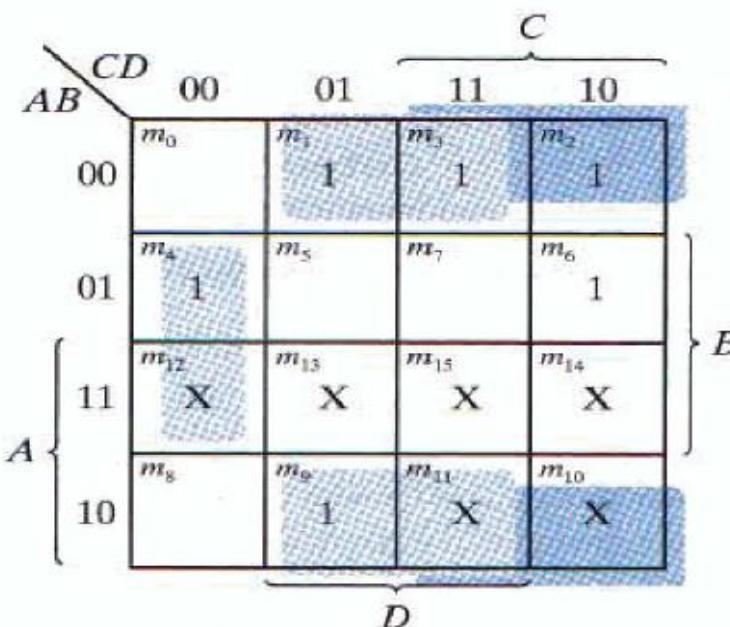
Code Conversion

Conversion from BCD to Excess-3



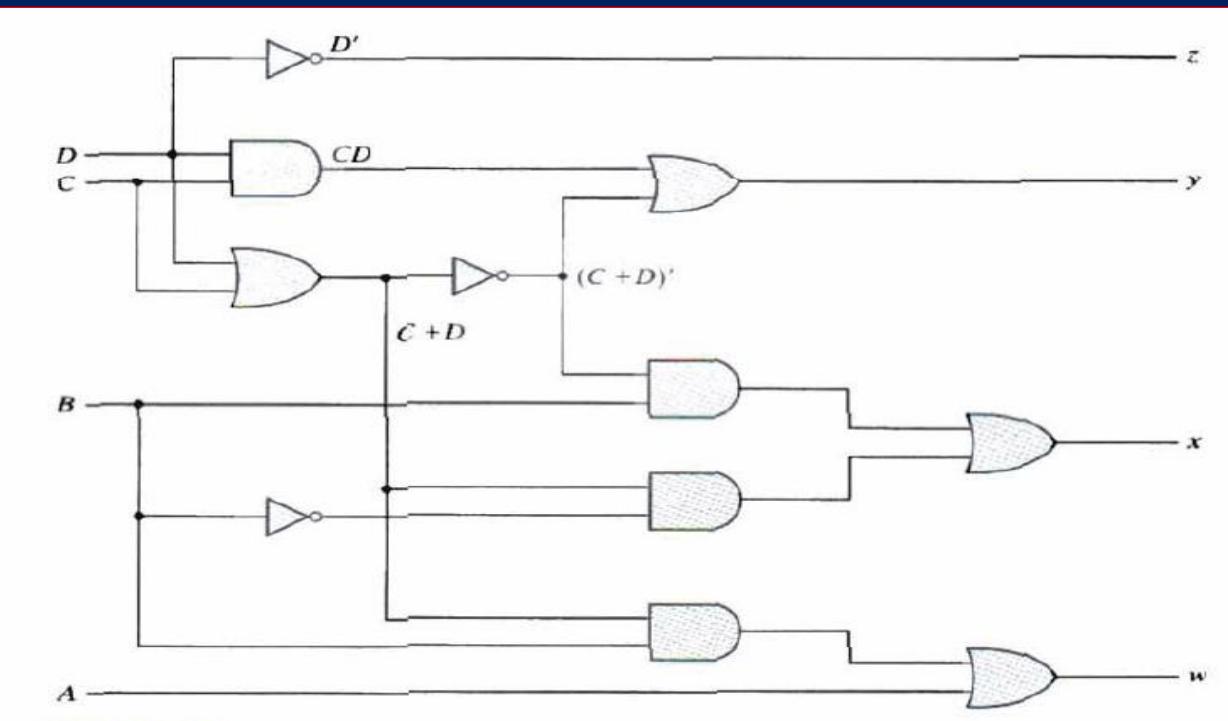
Code Conversion

Conversion from BCD to Excess-3



Code Conversion

Conversion from BCD to Excess-3



Code Conversion

Conversion from BCD to Excess-3

$$z = D'$$

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

$$\begin{aligned}x &= B'C + B'D + BC'D' = B'(C+D) + BC'D' \\&= B'(C+D) + B(C+D)'\end{aligned}$$

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

DESIGN PROCEDURE

The procedure involves the following steps :

1. The problem is started .
2. The number of available input variables and required output variables is determined .
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationships between inputs and outputs is derived .
5. The simplified Boolean function for each output is obtained .
6. The logic diagram is drawn .

ADDERS

Digital computers perform a variety of information processing tasks. Among the basic functions the arithmetic addition is the most basic operation.

The logic operations of addition in the binary system is:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 \text{ carry operation}$$

Summary

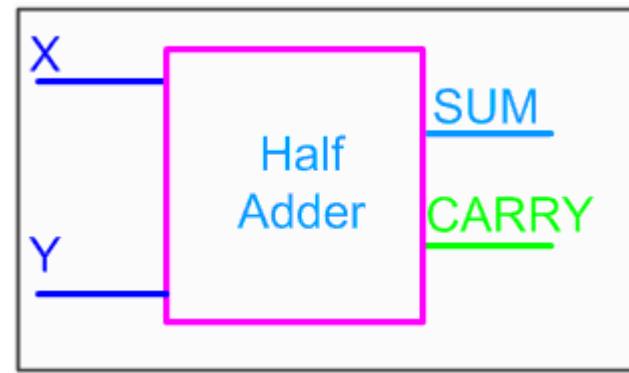
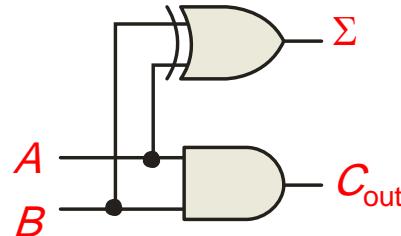
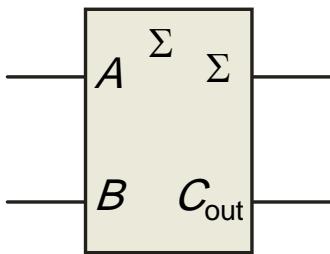
Half-Adder

Basic rules of binary addition are performed by a **half adder**, which has two binary inputs (A and B) and two binary outputs (Carry out and Sum).

The inputs and outputs can be summarized on a truth table.

The logic symbol and equivalent circuit are:

Inputs		Outputs	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

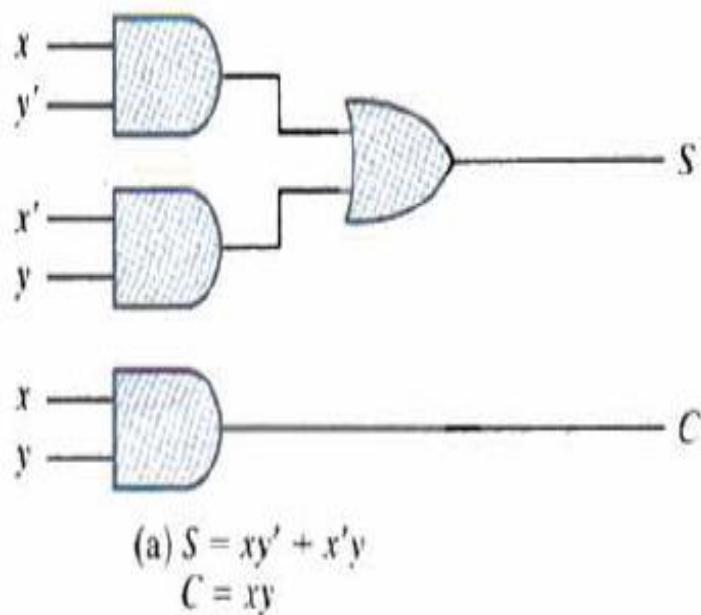


Summary

Half-Adder

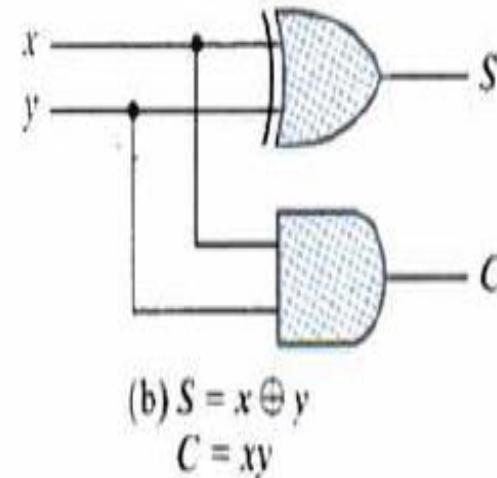
The circuit has 2 inputs and 2 outputs

Adds Two binaries only
S-sum **C-carry**



Half Adder

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

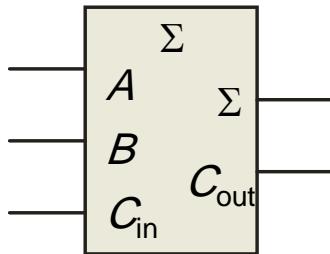


Summary

Full-Adder

A **full adder** has three binary inputs (A , B , and Carry in) and two binary outputs (Carry out and Sum). The truth table summarizes the operation.

Adds Three binaries



Symbol

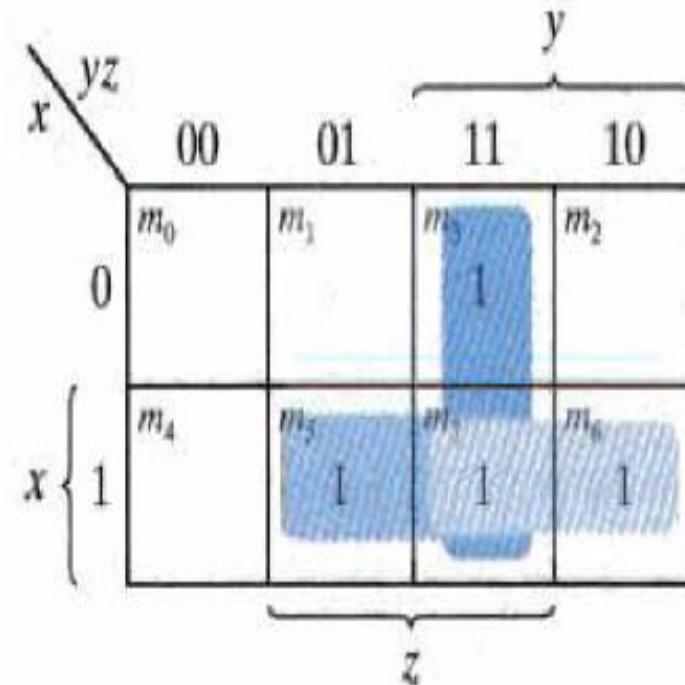
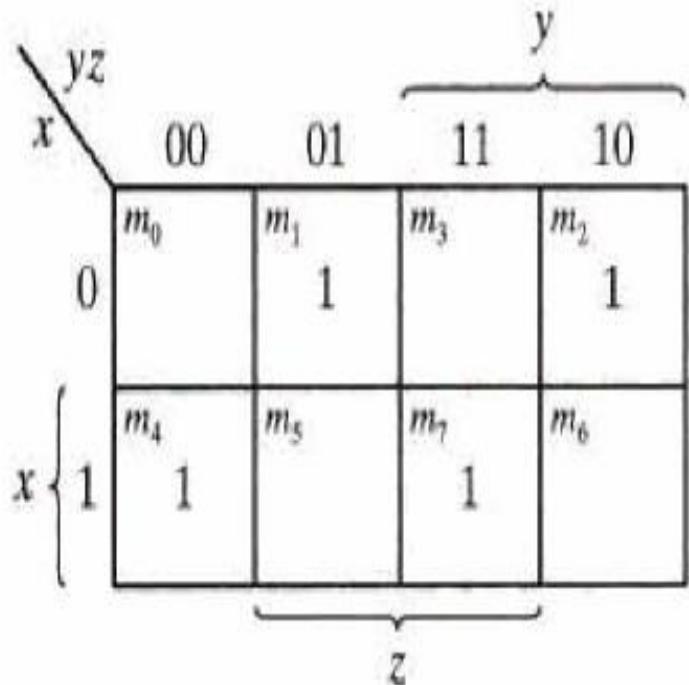
Inputs			Outputs	
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S(A, B, C) = \sum(1, 2, 4, 7)$$

$$C_{out}(A, B, C) = \sum(3, 5, 6, 7)$$

Summary

Full-Adder



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

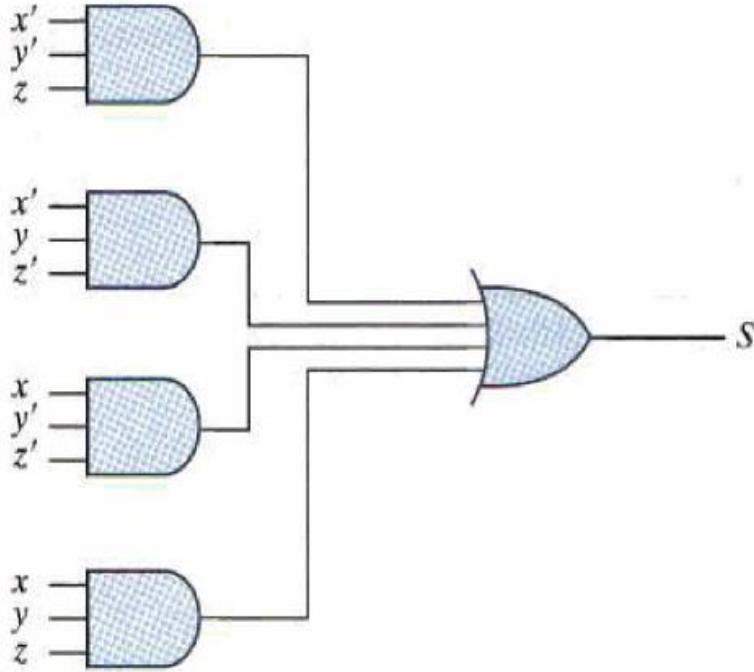
$$S = x' y' z + x' y z' + x y' z' + x y z$$

$$\text{Cout}(x, y, z) = \sum(3, 5, 6, 7)$$

$$C = x y + x z + y z \quad \text{also} \quad C = x' y z + x y' z + x y z$$

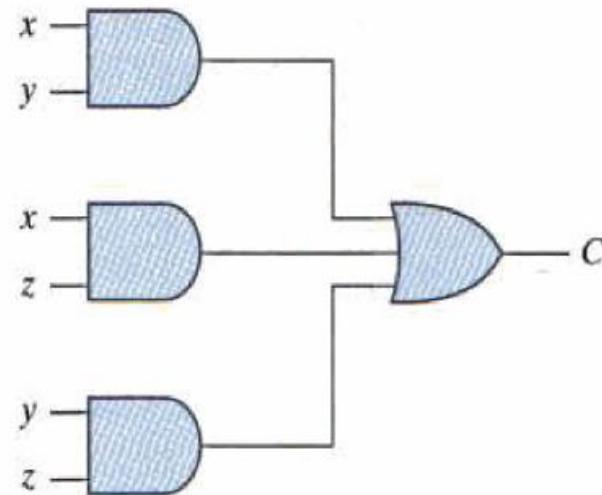
Summary

Full Adder(Sum of products)



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

$$C = x y + x z + y z$$

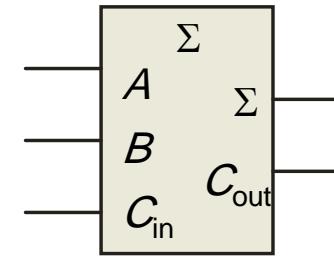
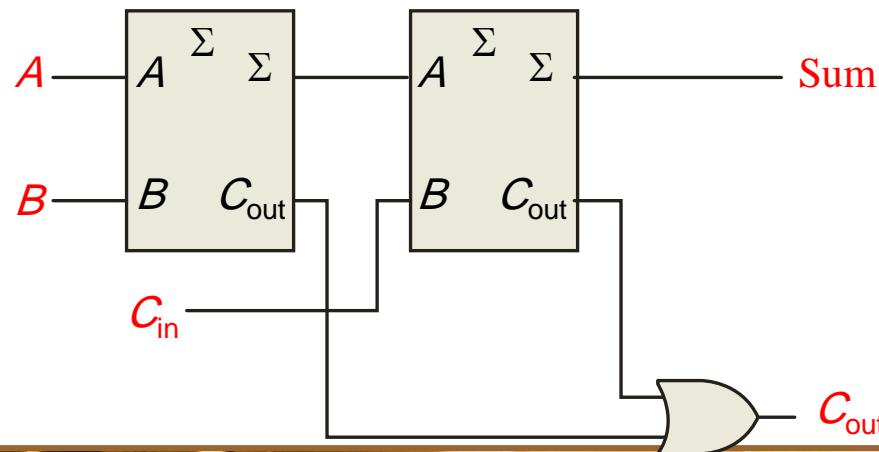
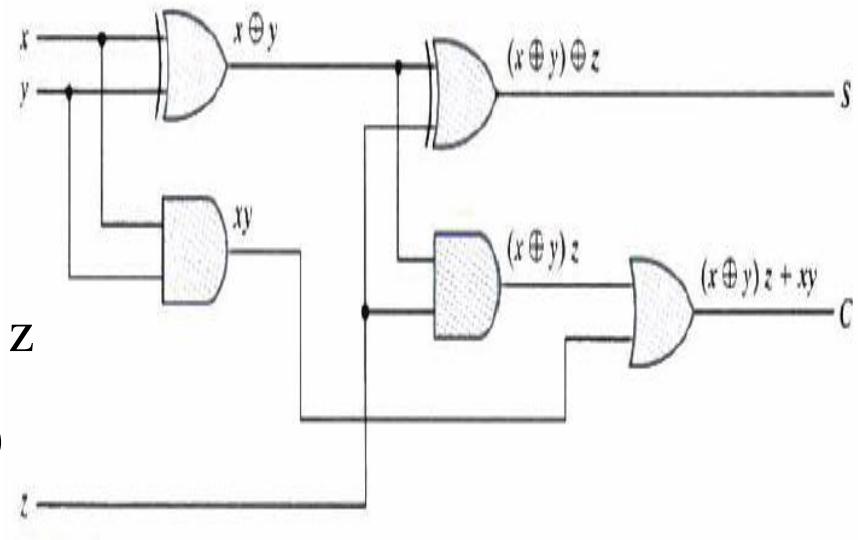


Summary

Full-Adder

A full-adder can be constructed from two half adders as shown:

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



Symbol

Summary

Full-Adder

Example

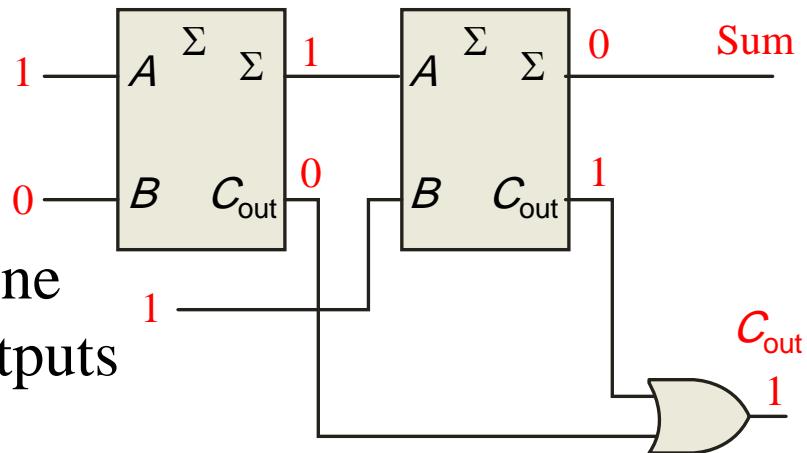
For the given inputs, determine the intermediate and final outputs of the full adder.

Solution

The first half-adder has inputs of 1 and 0; therefore the Sum = 1 and the Carry out = 0.

The second half-adder has inputs of 1 and 1; therefore the Sum = 0 and the Carry out = 1.

The OR gate has inputs of 1 and 0, therefore the final carry out = 1.

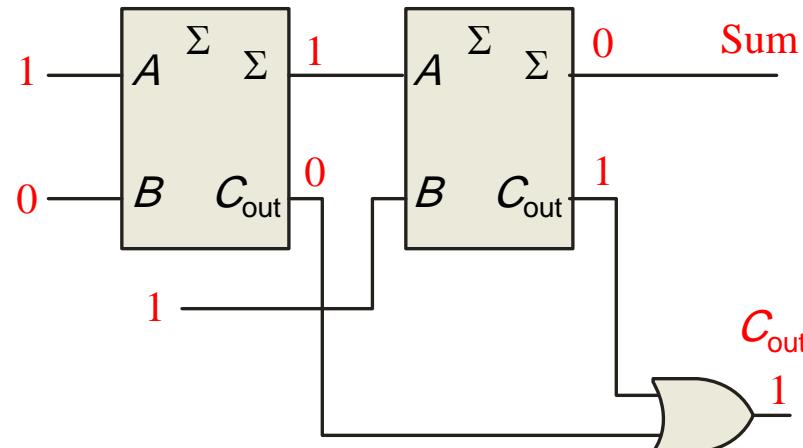


Summary

Full-Adder

Notice that the result from the previous example can be read directly on the truth table for a full adder.

Inputs			Outputs	
A	B	C_{in}	C_{out}	Σ
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





Summary

Binary Adder (with carry operation)

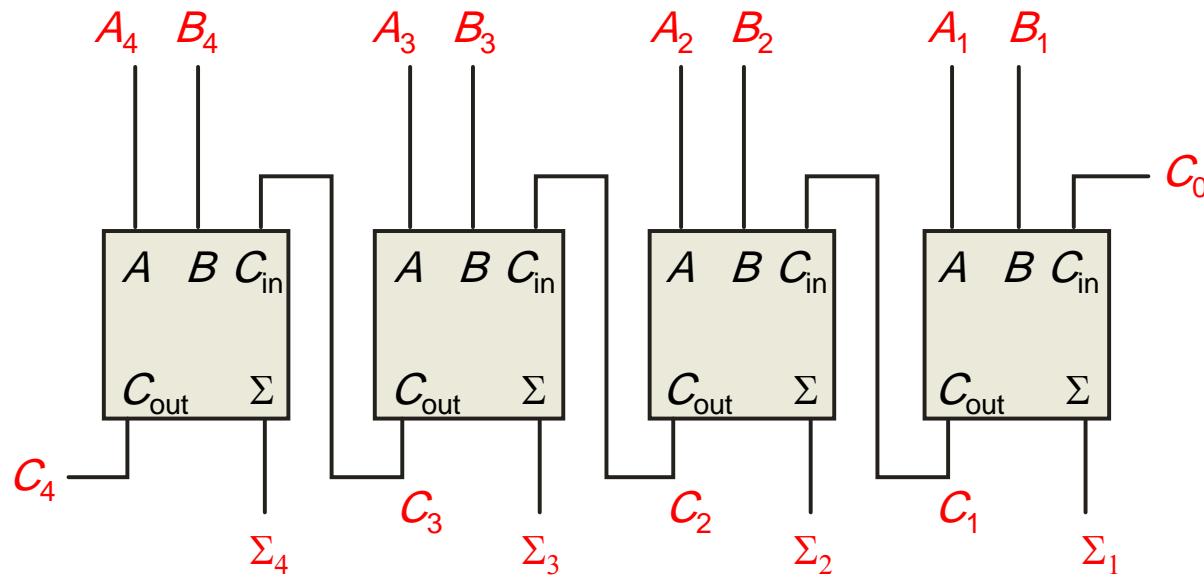
Full adders are combined into parallel adders that can add binary numbers with multiple bits.

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Summary

Parallel Adders

A 4-bit adder is shown.

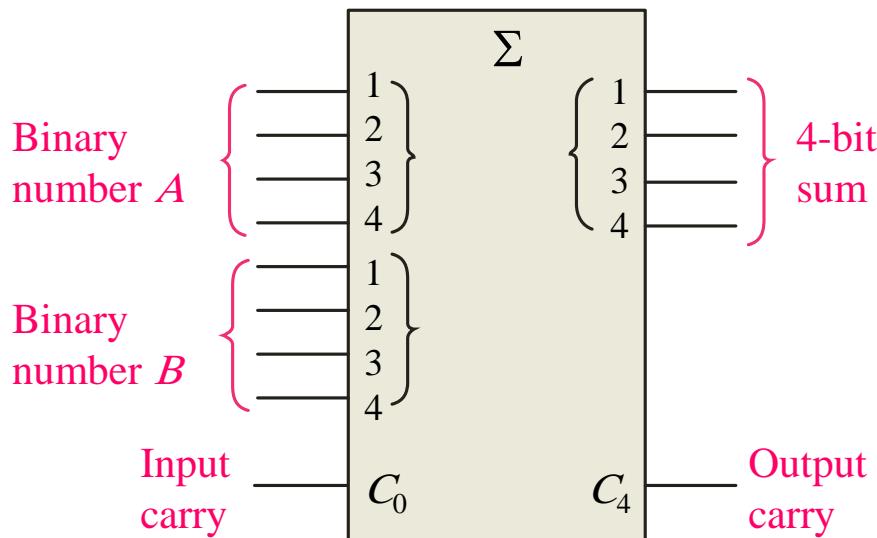


The output carry (C_4) is not ready until it propagates through all of the full adders. This is called *ripple carry*, delaying the addition process.

Summary

Parallel Adders

The logic symbol for a 4-bit parallel adder is shown. This 4-bit adder includes a carry in (labeled C_0) and a Carry out (labeled C_4).



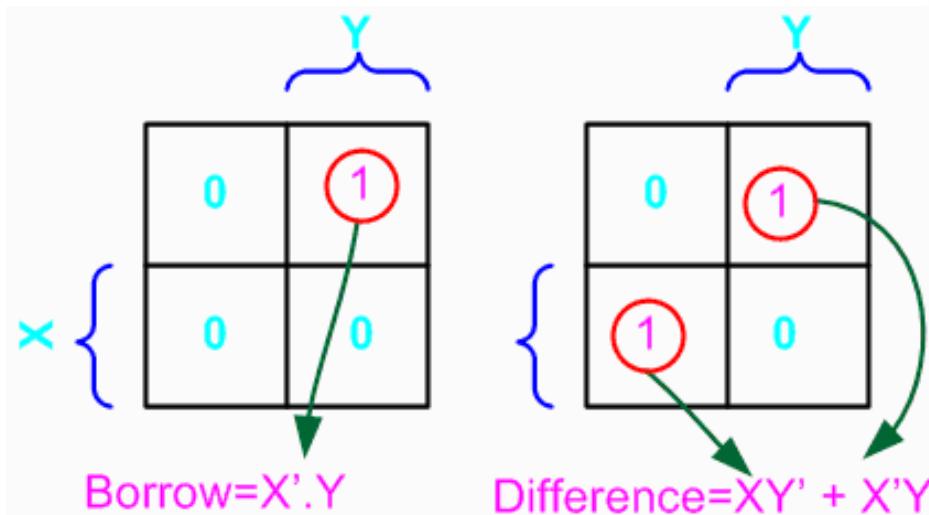
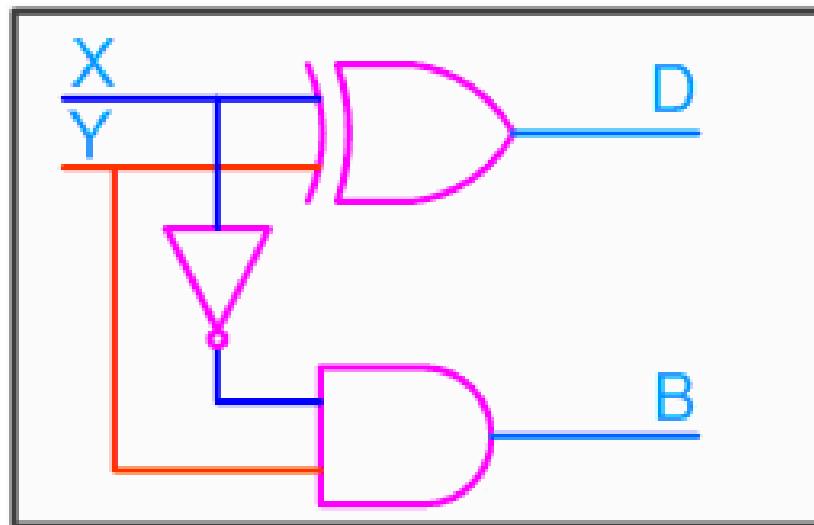
The 74LS283 is an example. It features *look-ahead carry*, which adds logic to minimize the output carry delay. For the 74LS283, the maximum delay to the output carry is 17 ns.

Binary Subtractor

- The subtraction of two binary numbers may be a composed by taking the complement of the subtrahend and adding it to the minuend.
- The subtraction of unsigned binary numbers can be done most conveniently by means of complements.
- Remember that the subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A .

Half Subtractor

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Full Subtractor

X	Y	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = X'Y'Bin + X'YBin' + XY'Bin' + XYBin$$

$$= (X'Y' + XY)Bin + (X'Y + XY')Bin' = (X \oplus Y)'Bin + (X \oplus Y)Bin'$$

$$= X \oplus Y \oplus Bin$$

- $Bout = X'.Y + X'.Bin + Y.Bin$

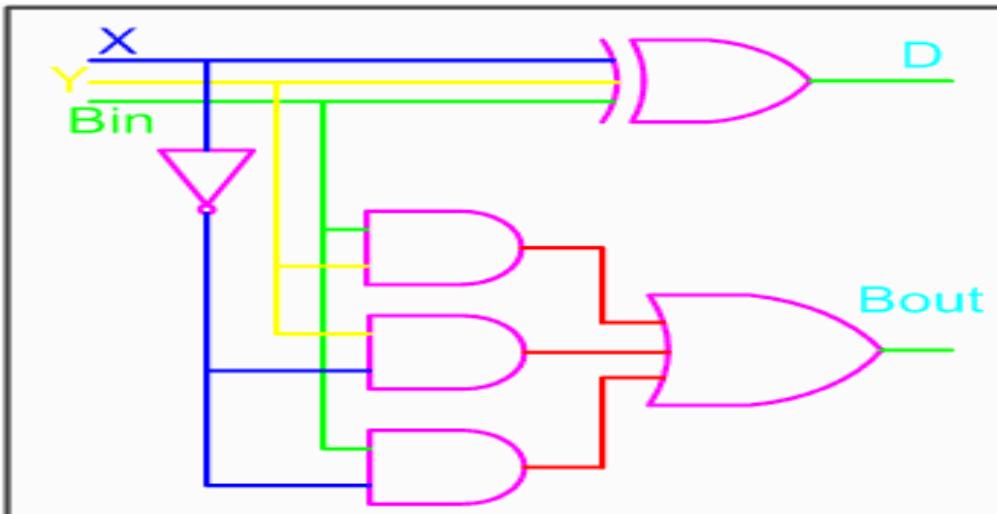
Full Subtractor

X	YBin	00	01	11	10
0		0	1	0	1
1		1	0	1	0

X	YBin	00	01	11	10
0		0	1	1	1
1		0	0	1	0

$$\text{Difference} = X'Y'Bin + X'YBin' + XY'Bin' + XYBin$$

$$\text{Borrow} = X'Bin + X'Y + YBin$$



Four –bit Adder-Subtractor

A-B =?

Inverter before each B

A + complement of B

Add an Exclusive OR gate to overcome

M = 0 Adder

$$B \oplus 0 = B$$

M = 1 Subtractor

$$B \oplus 1 = B^{\complement}, C_0 = 1$$

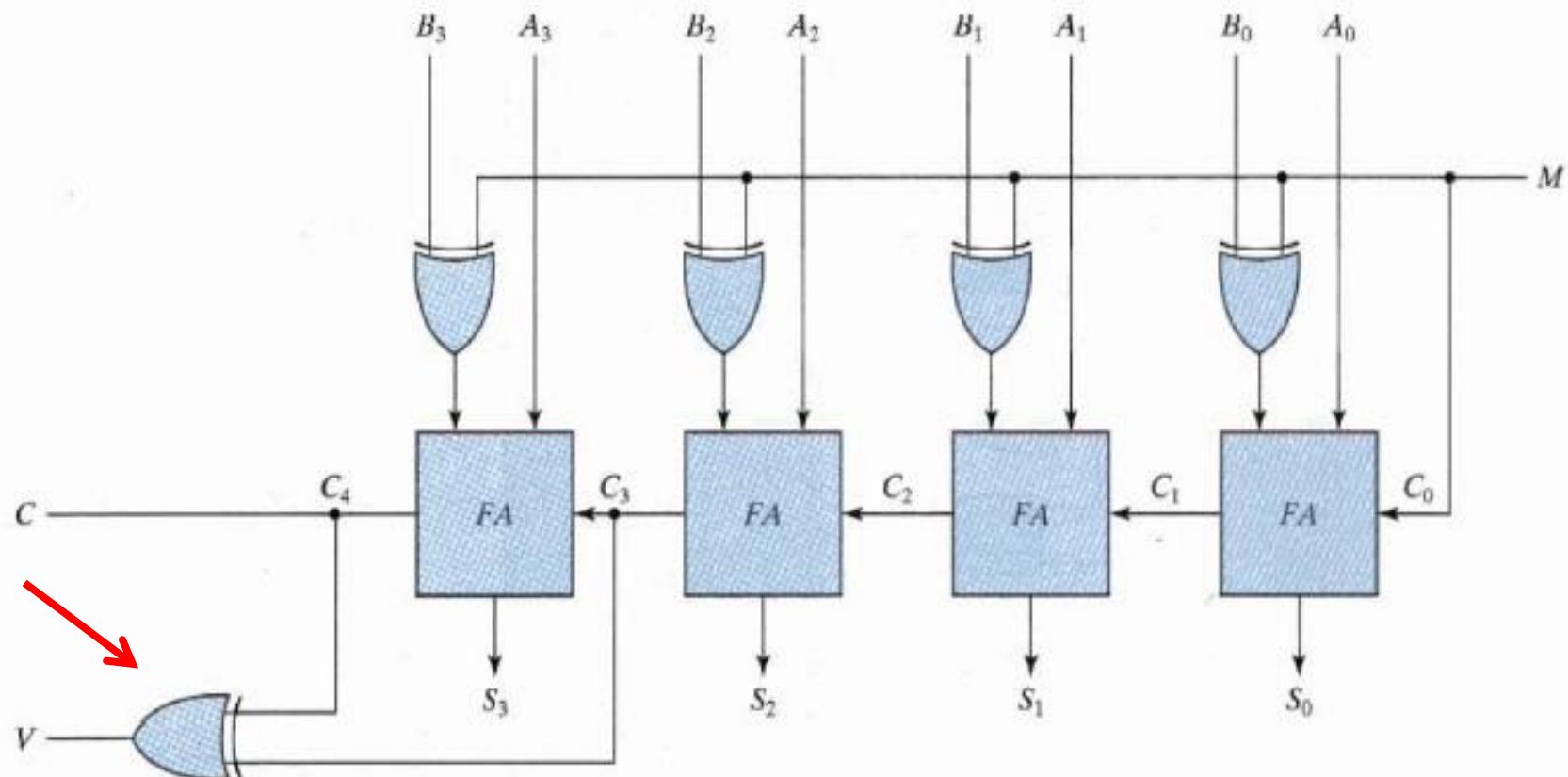
Where the Mode input M controls the operation.

Four –bit Adder-Subtractor

- ❑ The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder. A four-bit adder–subtractor circuit is shown in next slide.
- ❑ The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor.
- ❑ Each exclusive-OR gate receives input M and one of the inputs of B .
- ❑ When $M = 0$, we have $B \oplus 0 = B$. The full adders receive the value of B , the input carry is 0, and the circuit performs A plus B .
- ❑ When $M = 1$, we have $B \oplus 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry.
- ❑ The ~~circuit~~ performs the operation A plus the 2's complement of B .
- ❑ (The exclusive-OR with output V is for detecting an overflow.)

Four –bit Adder-Subtractor

Exclusive OR-gate on output V is to control overflow

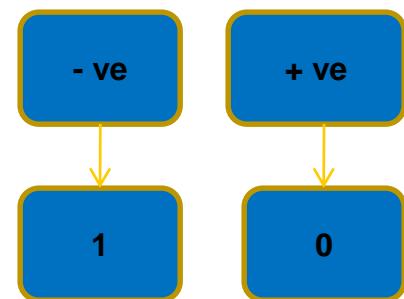


Four-bit adder–subtractor

Overflow

When two numbers with n digits each are added and the sum is a number occupying $n + 1$ digits, we say that an overflow occurred.

n dig + n dig → $n+1$ dig



carries:

$$\begin{array}{r} 0 \ 1 \\ +70 \\ +80 \\ \hline +150 \end{array}$$

$$\begin{array}{r} 0 \ 1 \\ 0 \ 1000110 \\ 0 \ 1010000 \\ \hline 1 \ 0010110 \end{array}$$

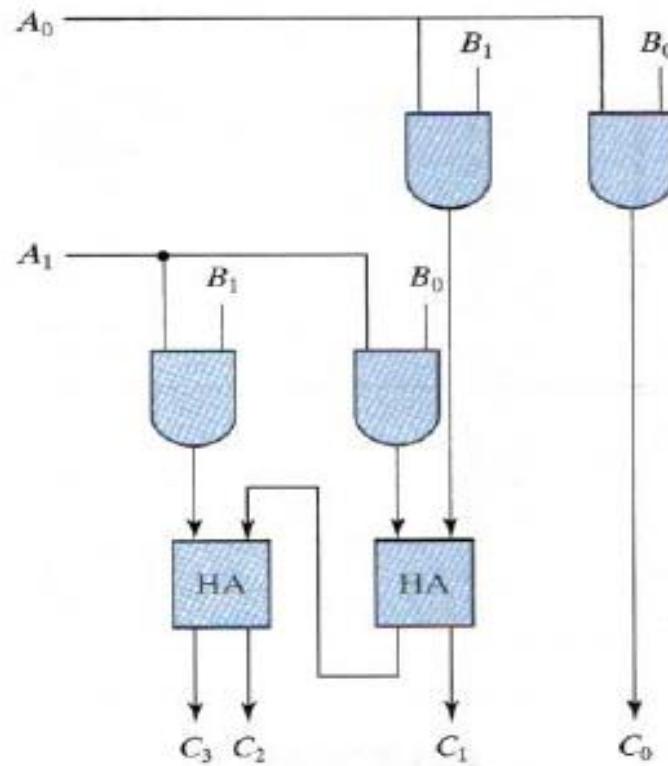
carries:

$$\begin{array}{r} 1 \ 0 \\ -70 \\ -80 \\ \hline -150 \end{array}$$
$$\begin{array}{r} 1 \ 0 \\ 1 \ 0111010 \\ 1 \ 0110000 \\ \hline 0 \ 1101010 \end{array}$$

Summary

Binary Multiplier

$$\begin{array}{r} & B_1 & B_0 \\ & \quad & \\ A_1 & & A_0 \\ \hline A_0B_1 & & A_0B_0 \\ & \hline A_1B_1 & A_1B_0 \\ \hline C_3 & C_2 & C_1 & C_0 \end{array}$$





Binary Multiplier

B1,B0 multiplicand

A1,A0 multipliers

C3C2 C1C0 product

For J multipliers and K multiplicands

(J x k) AND gates and (J-1) k-bit adders

to produce J+K bits

Summary

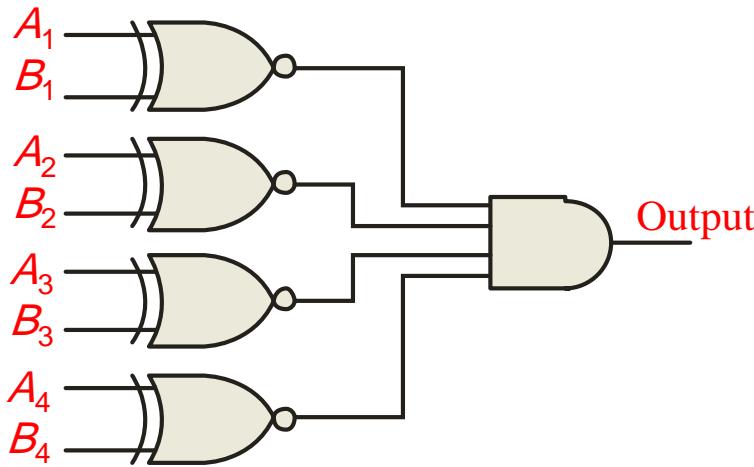
Comparators

The function of a comparator is to compare the magnitudes of two binary numbers to determine the relationship between them. In the simplest form, a comparator can test for equality using XNOR gates.

Example Solution

How could you test two 4-bit numbers for equality?

AND the outputs of four XNOR gates.



Summary

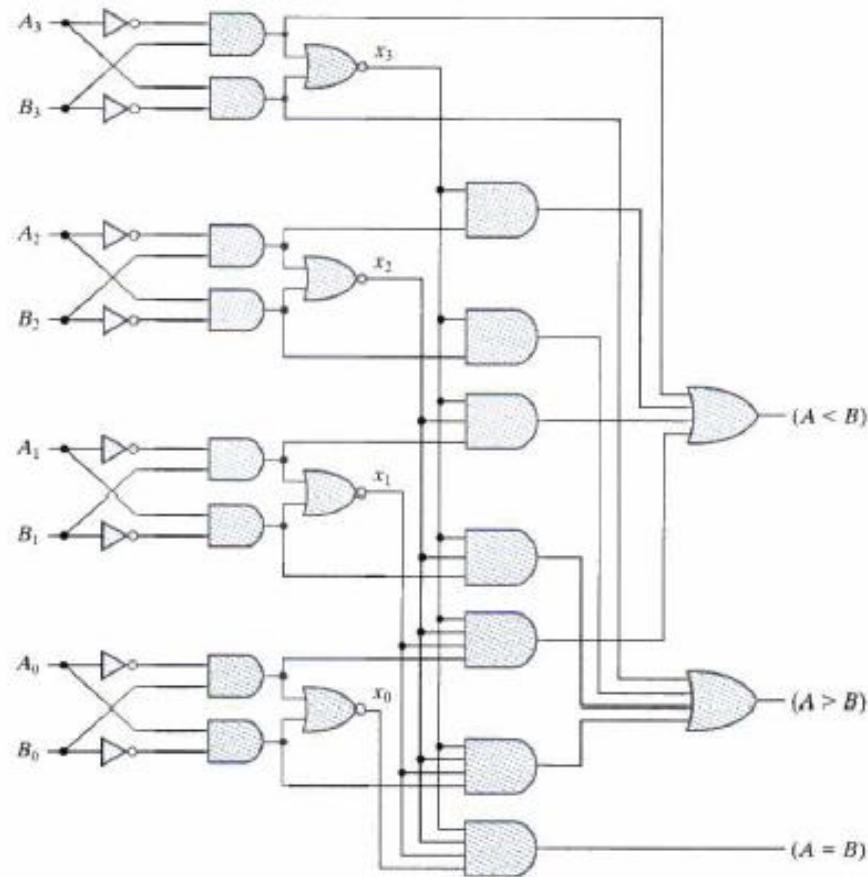
Four-bit Magnitude Comparator

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

$$x_i = A_i B_i + \bar{A}_i \bar{B}_i$$

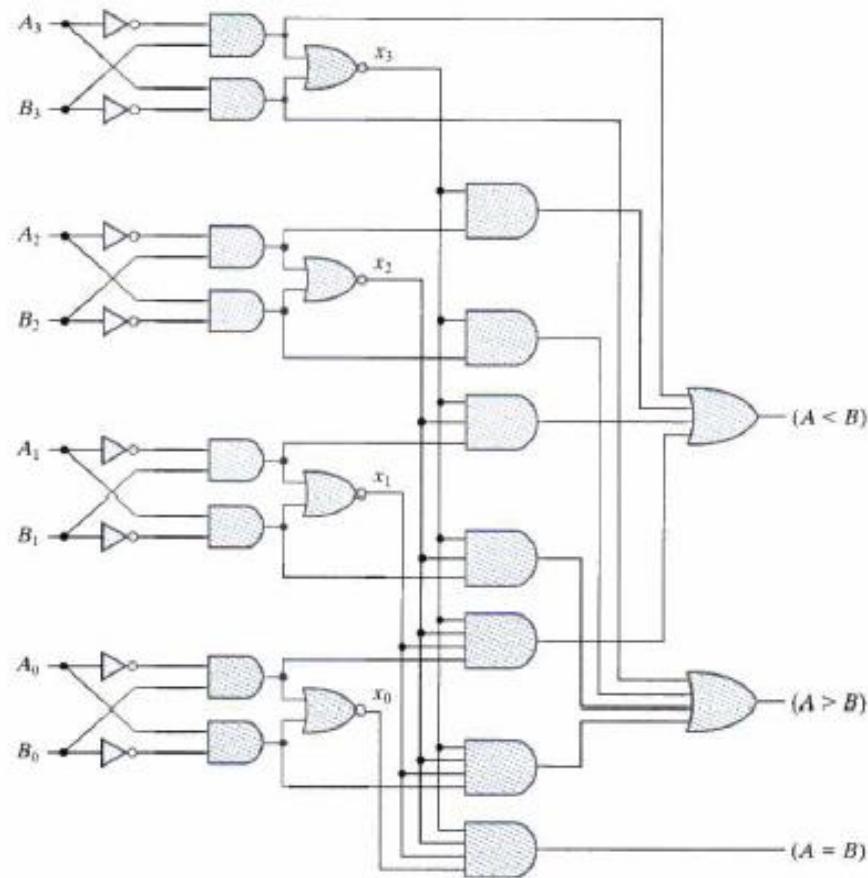
$$i=0,1,2,3$$





Four-bit Magnitude Comparator

- $A = B = x_3 \ x_2 \ x_1 \ x_0$





Summary

Four-bit Magnitude Comparator

A > B

$$= A_3 B^3 + x_3 A_2 B^2 + x_3 x_2 A_1 B^1 + x_3 x_2 x_1 A_0 B^0$$

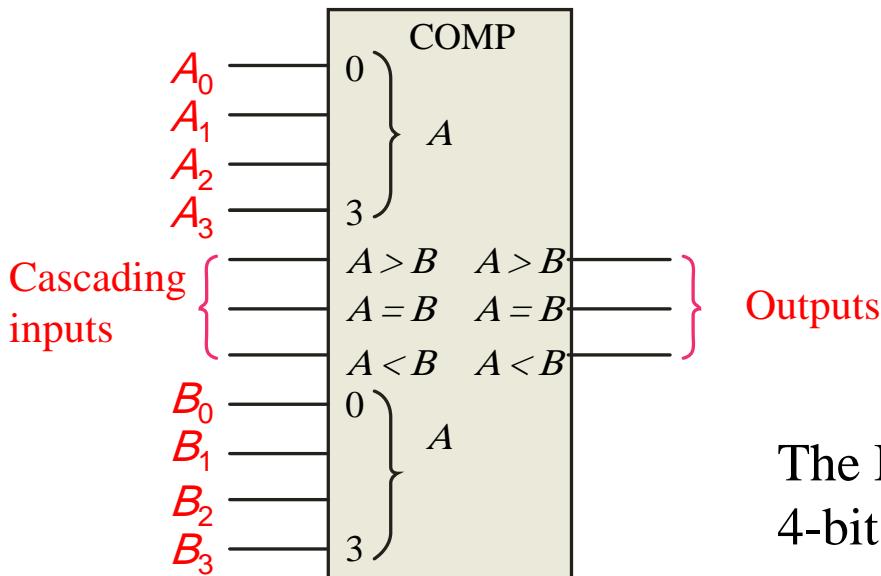
A < B

$$= A^3 B_3 + x_3 A^2 B_2 + x_3 x_2 A^1 B_1 + x_3 x_2 x_1 A^0 B_0$$

Summary

Comparators

IC comparators provide outputs to indicate which of the numbers is larger or if they are equal. The bits are numbered starting at 0, rather than 1 as in the case of adders. Cascading inputs are provided to expand the comparator to larger numbers.

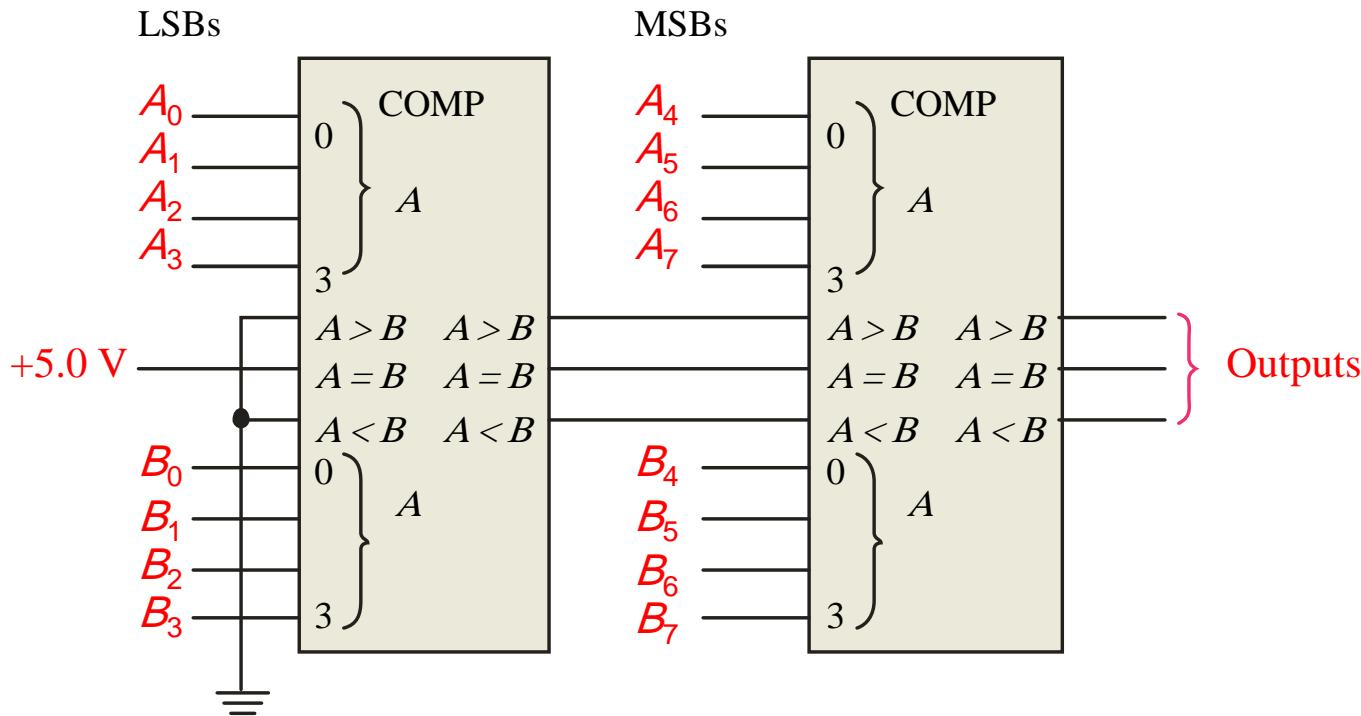


The IC shown is the
4-bit 74LS85.

Summary

Comparators

IC comparators can be expanded using the cascading inputs as shown. The lowest order comparator has a HIGH on the $A = B$ input.





Summary

Introduction: Logic circuits

- Logic circuits for digital systems may be **combinational** or **sequential**.
- A **combinational circuits** consist of logic gates whose outputs at any time are determined directly from the present combination of current inputs without regard to past inputs.
- A **sequential circuit** employ memory elements in addition to logic gates (i.e. output is function of present and previous inputs).

Summary

A combinational circuits

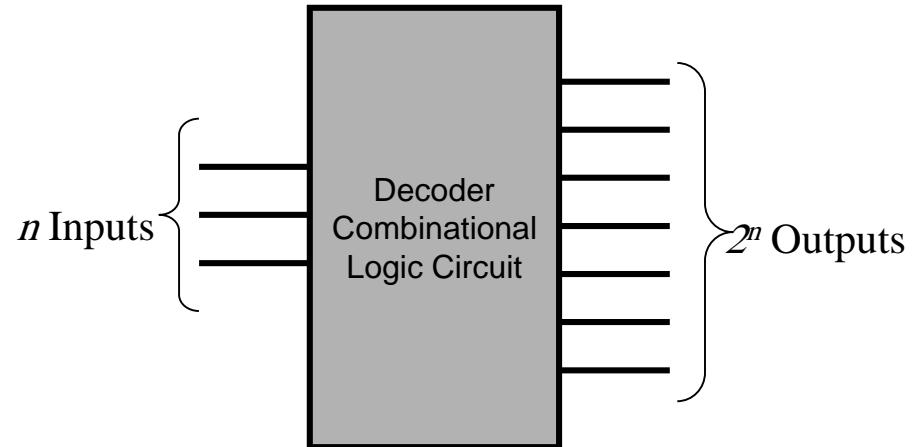
- Outputs (*m*) are a function of the present set of inputs (*n*) only
- Inside of a combinational circuit is made of logic gates
- Combinational logic circuits are important components of digital systems



Summary

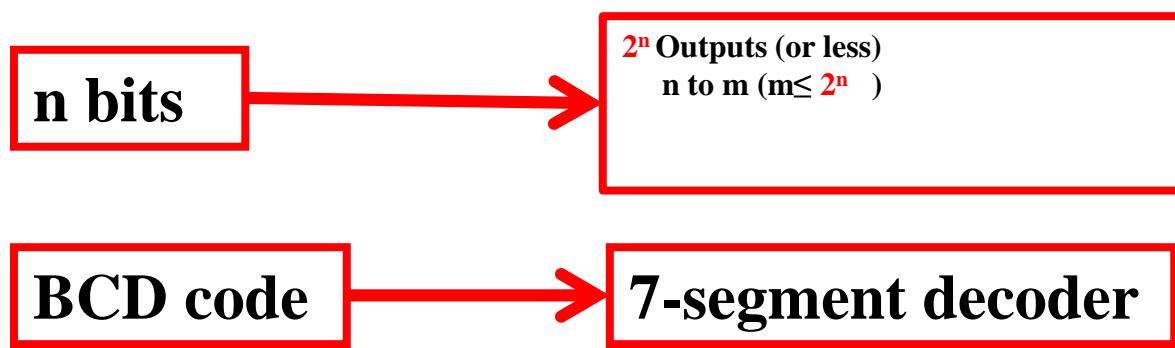
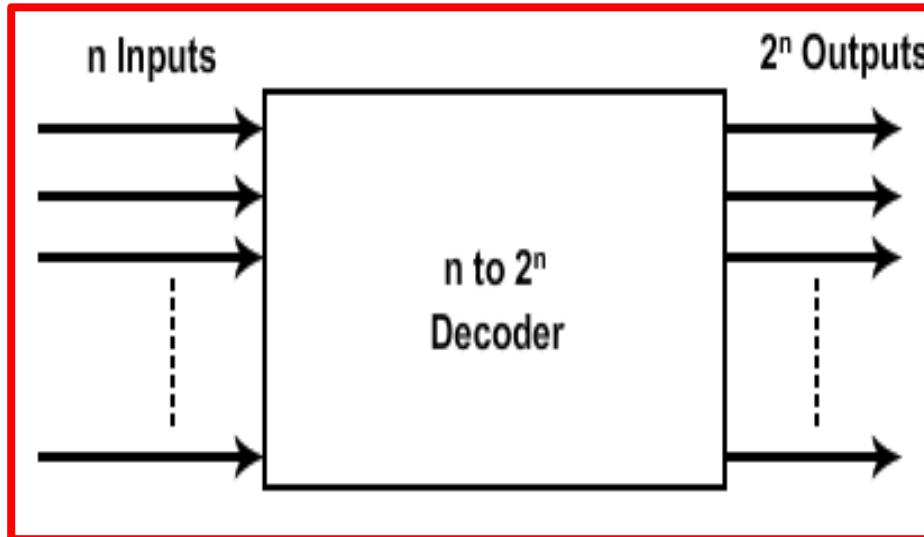
Decoders

- **n to 2^n decoder** - a combinational logic circuit that converts binary information from the **n** coded inputs to a maximum of **2^n** unique outputs.
- Also called the *n-to-m* line deciders for example:
 - 2-to-4 line decoder
 - 3-to-8 line decoder



Summary

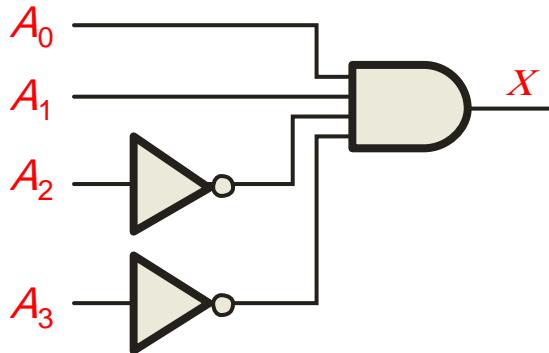
Decoders



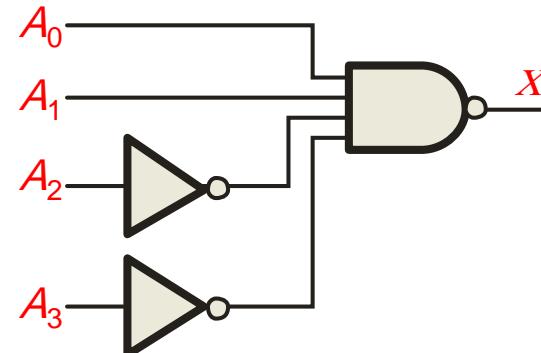
Summary

Decoders

- A **decoder** is a logic circuit that detects the presence of a specific combination of bits at its input.
- Two simple decoders that *detect the presence of the binary code 0011* are shown. The first has an active HIGH output; the second has an active LOW output.



Active HIGH decoder for 0011



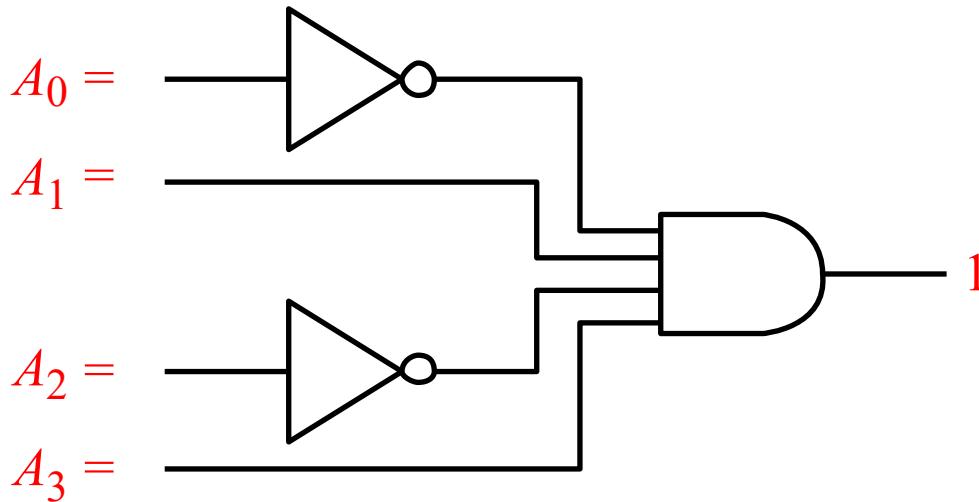
Active LOW decoder for 0011

Summary

Decoders

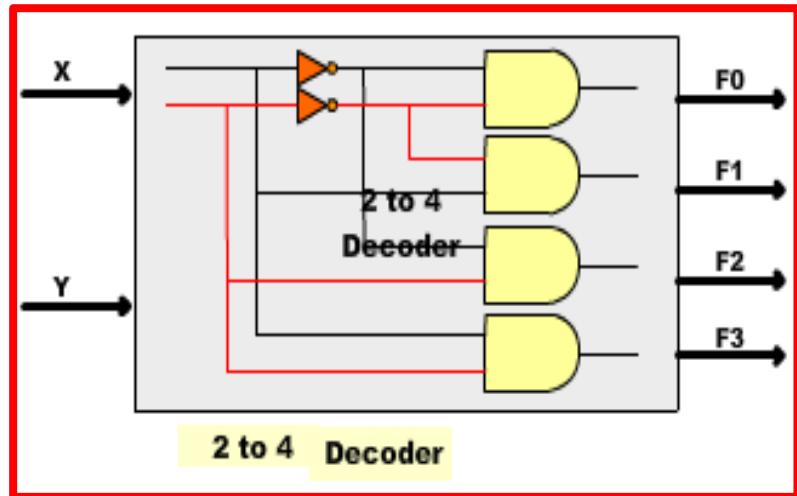
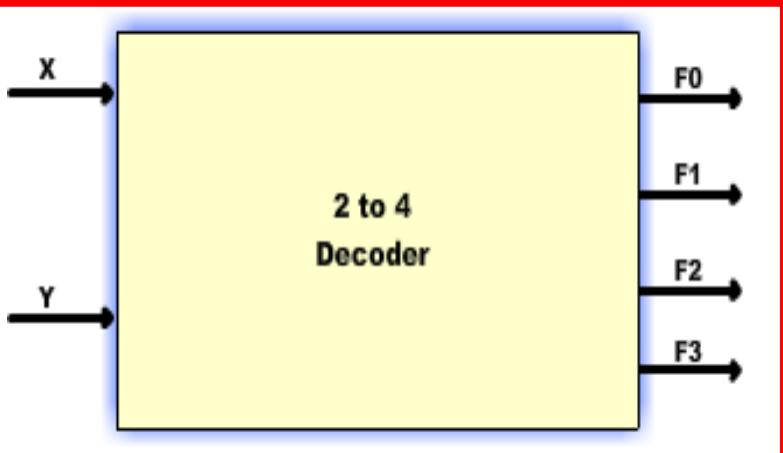
Question

Assume the output of the decoder shown is a logic 1. What are the inputs to the decoder?

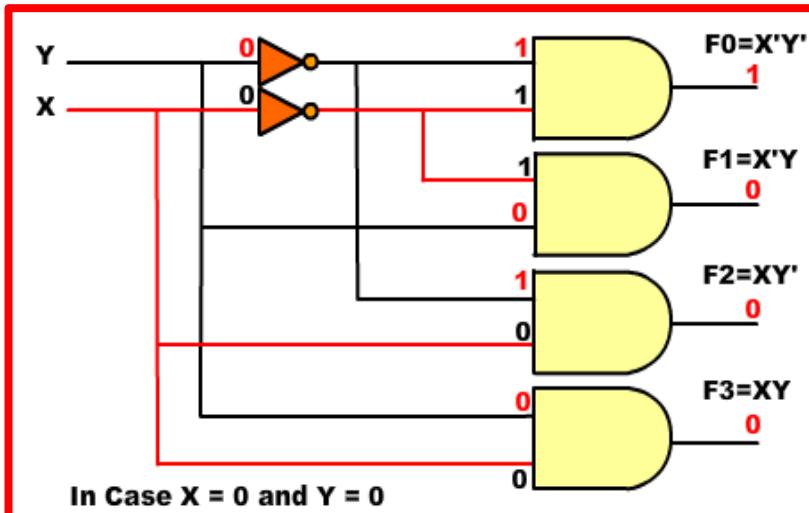


Summary

Decoders

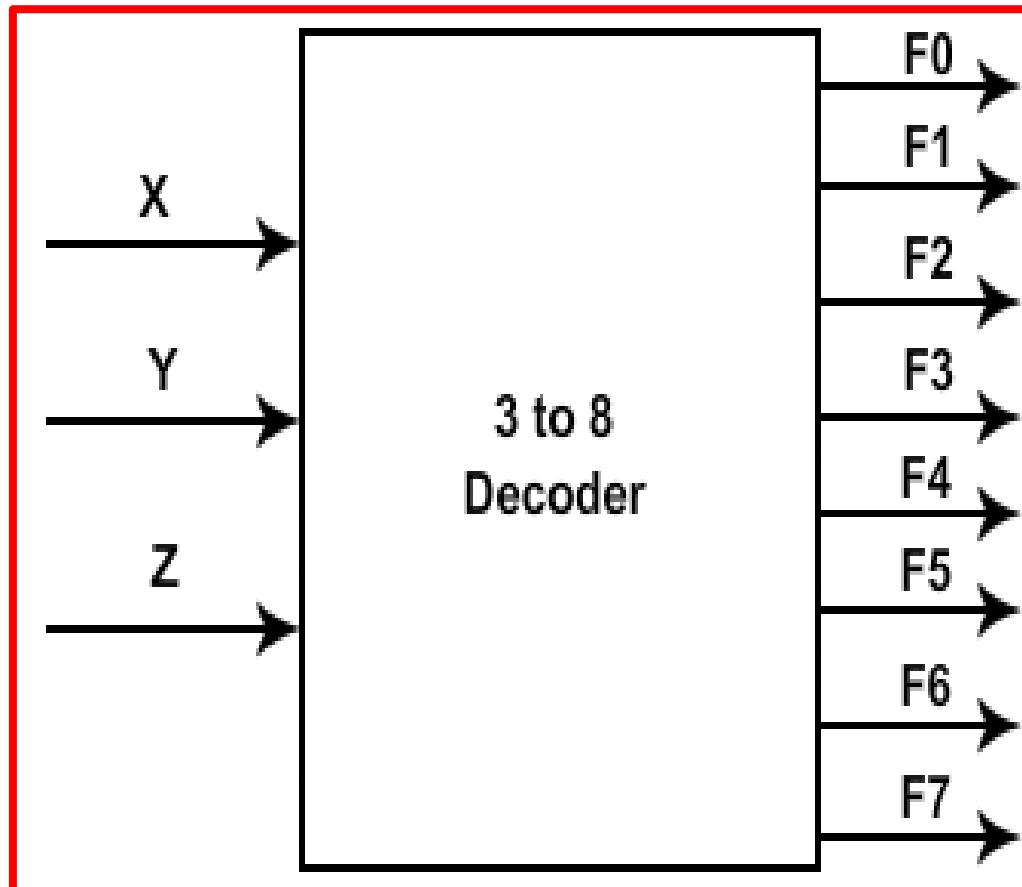


Inputs		Outputs			
X	Y	D ₀	D ₁	D ₂	D ₃
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



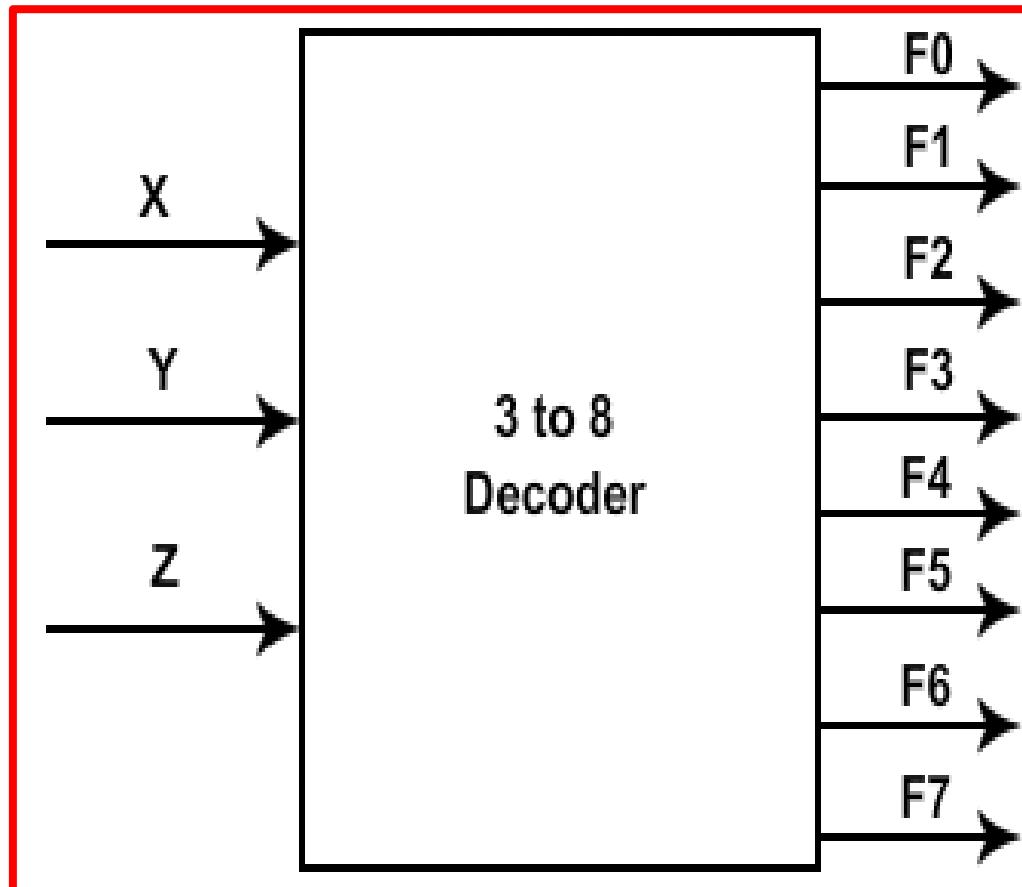


Three-to-eight Line Decoder





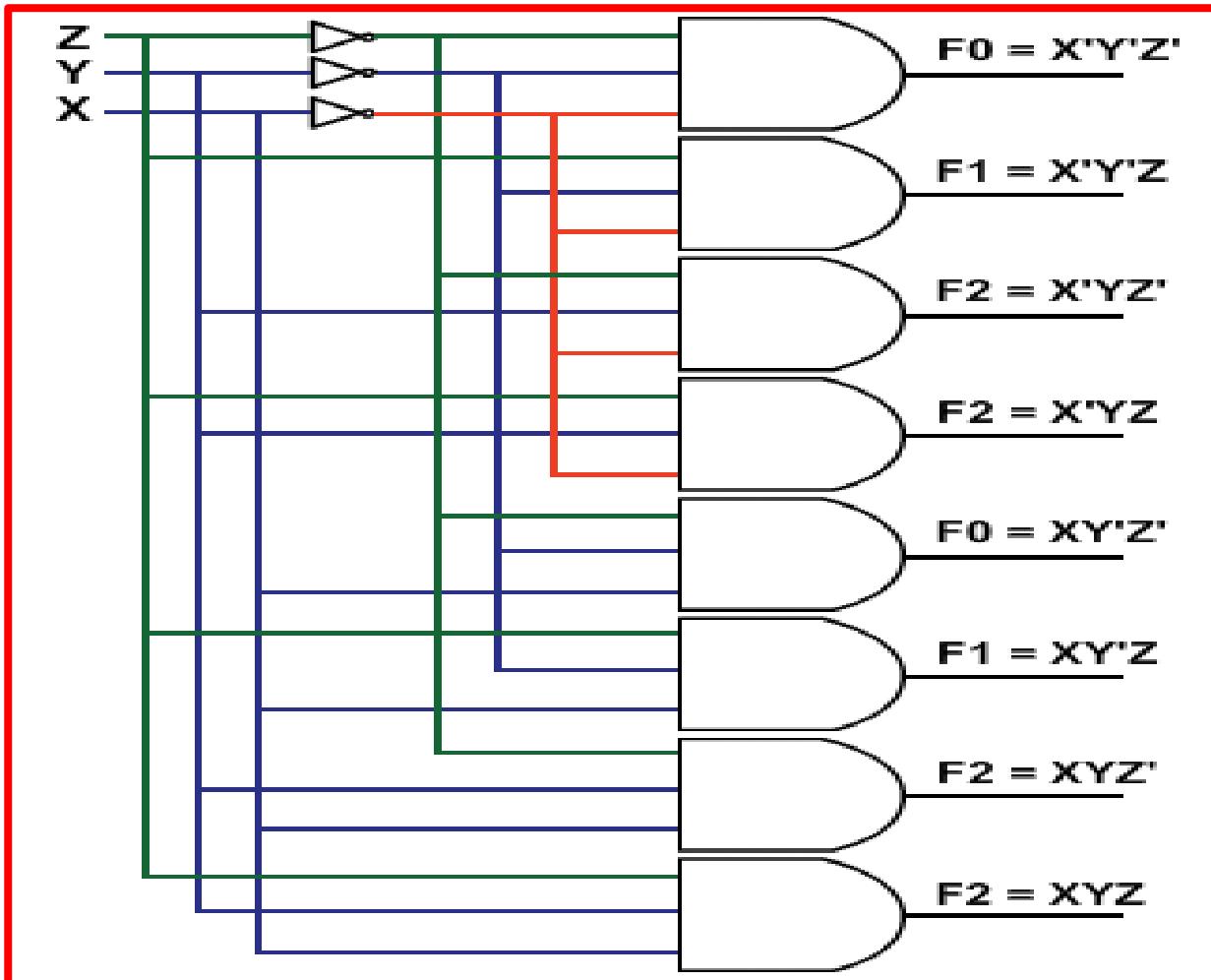
Three-to-eight Line Decoder





Summary

Three-to-eight Line Decoder





Three-to-eight Line Decoder

Truth Table

X	Y	Z	F0	F1	F2	F3	F4	F5	F6	F7
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



Three-to-eight Line Decoder

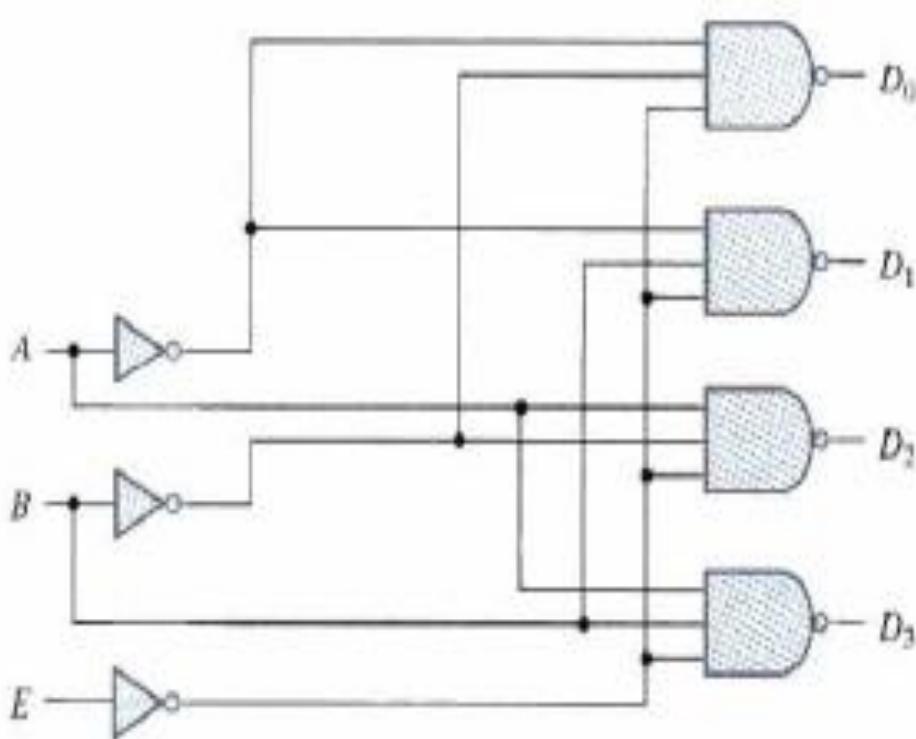
Truth Table of a Three-to-Eight-Line Decoder

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



Summary

3-to-4 Line Decoder with enable input



(a) Logic diagram

E	A	B	D ₀	D ₁	D ₂	D ₃
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Summary

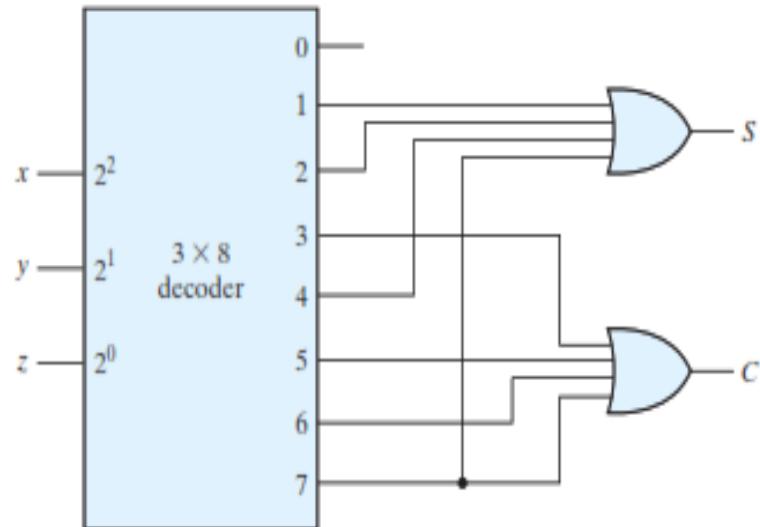
Major application of Decoder

- Decoder is used to implement any combinational circuits (f^n)

- For example the truth table for full adder is

$$S(x,y,z) = \sum(1,2,4,7) \text{ and } C(x,y,z) = \sum(3,5,6,7).$$

- Since the two functions C and S both have the same inputs, we could use just one decoder instead of two.
- The implementation with decoder is



Implementation of a full adder with a decoder



Summary

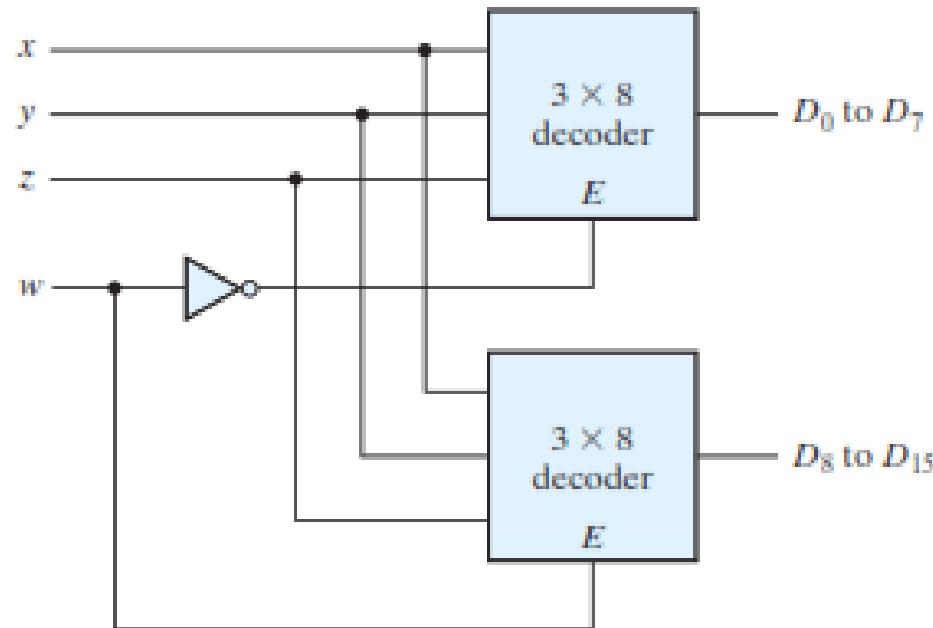
Combinational Circuit Implementation using Decoder

- A n -input decoder generates the minterms corresponding to a boolean function of n -variables
- Can use the outputs of a decoder along with an OR gate to implement a Boolean function in SOP form
- Decoders are sometimes called **minterm generators**.
 - For each of the input combinations, exactly one output is true.
 - Each output equation contains all of the input variables.
 - These properties hold for all sizes of decoders.
- You can implement arbitrary functions with decoders. If you have a sum of minterms equation for a function, you can easily use a decoder (a minterm generator) to implement that function

Summary

Decoder Expansion

- Can use smaller decoders to build bigger decoders.
- e.g. using two 2-4 decoders we can build a 3-8 decoder or using two 3-8 decoders we can build a 4-16 decoder



4 × 16 decoder constructed with two 3 × 8 decoders

Summary

Combinational Logic Implementation - Decoder and OR Gates

□ Implement m functions of n variables with:

- Sum-of-minterms expressions
- One n -to- 2^n -line decoder
- m OR gates, one for each output

□ Approach 1:

- Find the truth table for the functions
- Make a connection to the corresponding OR from the corresponding decoder output wherever a 1 appears in the truth table

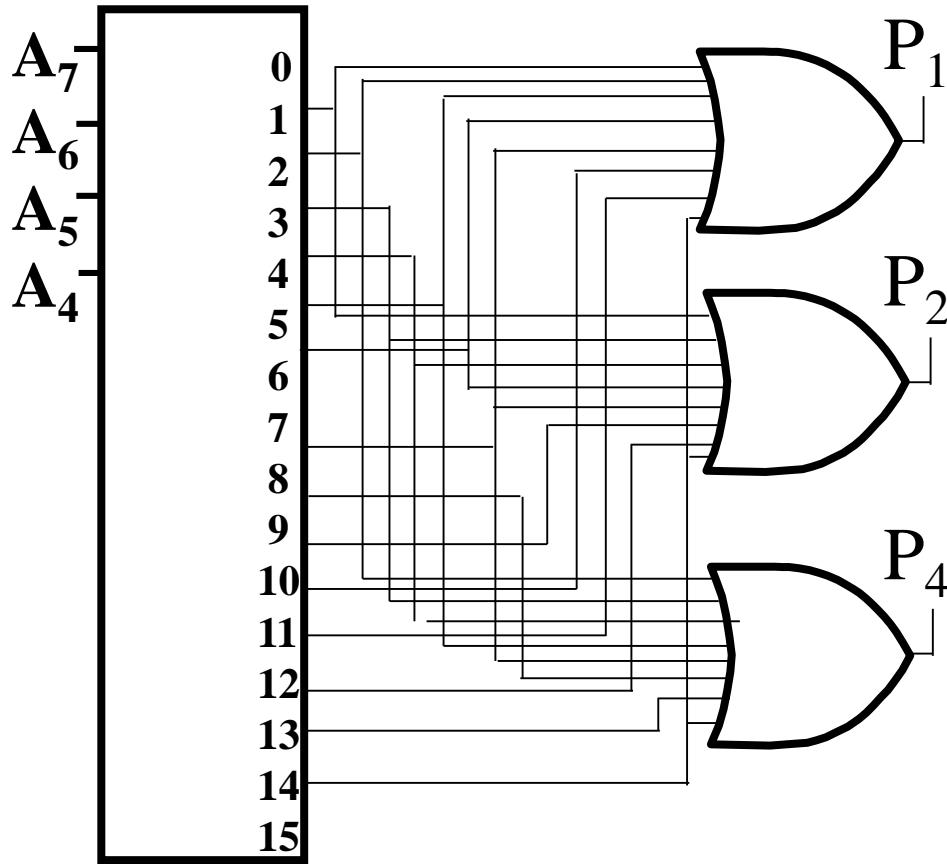
□ Approach 2

- Find the minterms for each output function
- OR the minterms together

Summary

Decoder and OR Gates Example

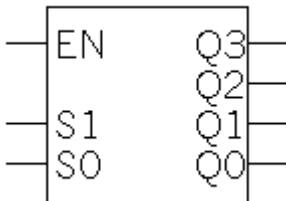
- Finding sum of minterms expressions
- $P_1 = S_m(1,2,5,6,8,11,12,15)$
- $P_2 = S_m(1,3,4,6,8,10,13,15)$
- $P_4 = S_m(2,3,4,5,8,9,14,15)$



Summary

Decoders: A variation of the standard decoder

- Active-high decoders generate minterms, as we've already seen.
- The decoders we've seen so far are **active-high** decoders.



$$\begin{aligned}Q_3 &= S_1 \ S_0 \\Q_2 &= S_1 \ S_0' \\Q_1 &= S_1' \ S_0 \\Q_0 &= S_1' \ S_0'\end{aligned}$$

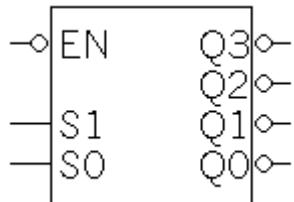
EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

- An **active-low** decoder is the same thing, but with an inverted EN input and inverted outputs
- The output equations for an active-low decoder are mysteriously similar, yet somehow different.

Summary

Decoders: A variation of the standard decoder

- An **active-low** decoder is the same thing, but with an inverted EN input and inverted outputs
- It turns out that active-low decoders generate maxterms.



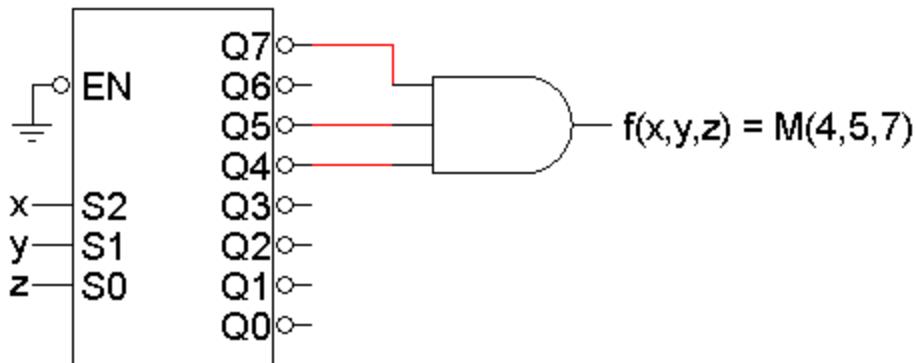
$$\begin{aligned}Q3' &= (S1 \ S0)' = S1' + S0' \\Q2' &= (S1 \ S0')' = S1' + S0 \\Q1' &= (S1' \ S0)' = S1 + S0' \\Q0' &= (S1' \ S0')' = S1 + S0\end{aligned}$$

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

Summary

Decoders: A variation of the standard decoder

- So we can use active-low decoders to implement arbitrary functions too, but as a product of maxterms.
- For example, here is an implementation of the function from the previous page, $f(x,y,z) = \prod M(4,5,7)$, using an **active-low** decoder.
- The “ground” symbol connected to EN represents logical 0, so this decoder is always enabled.
- Remember that you need an AND gate for a product of sums.



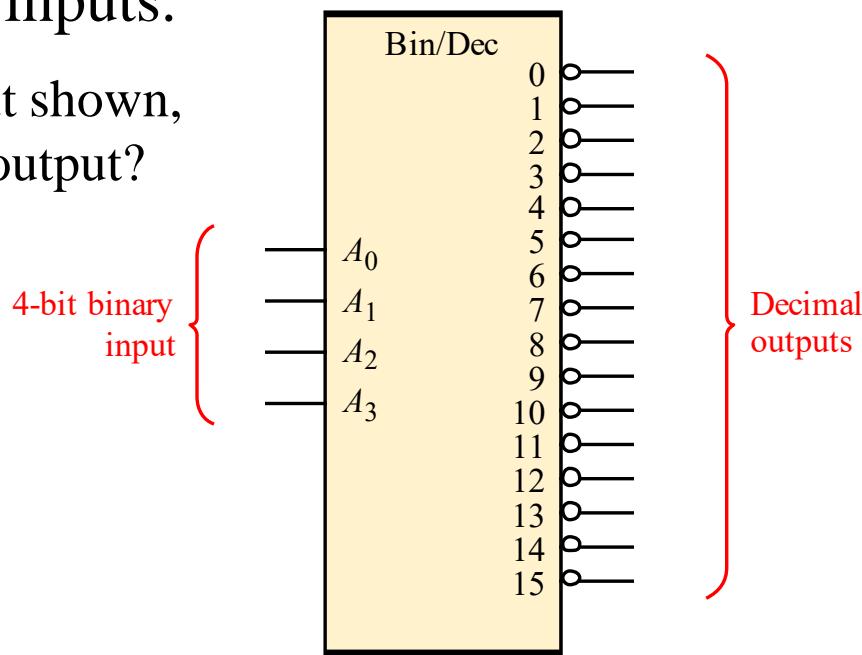
Summary

Decoders

IC decoders have multiple outputs to decode any combination of inputs. For example the binary-to-decimal decoder shown here has 16 outputs – one for each combination of binary inputs.

Question

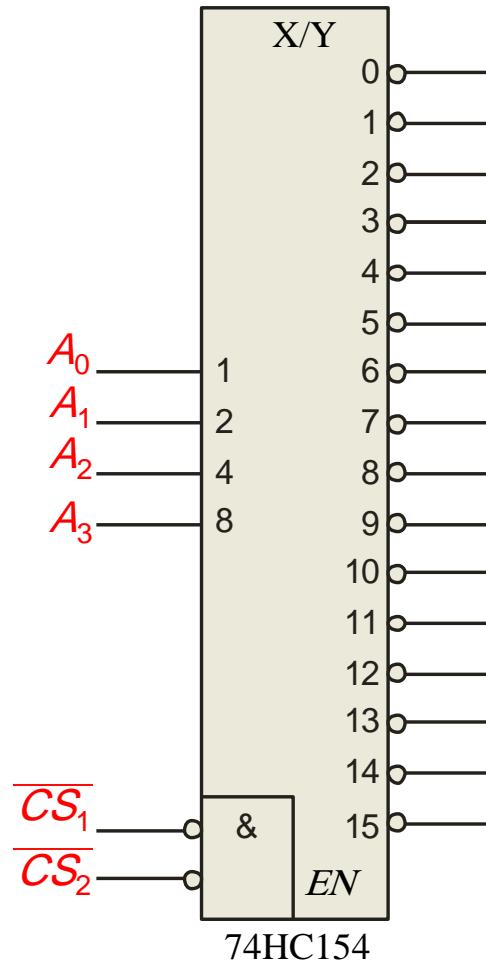
For the input shown,
what is the output?



Summary

Decoders

A specific integrated circuit decoder is the 74HC154 (shown as a 4-to-16 decoder). It includes two active LOW chip select lines which must be at the active level to enable the outputs. These lines can be used to expand the decoder to larger inputs.

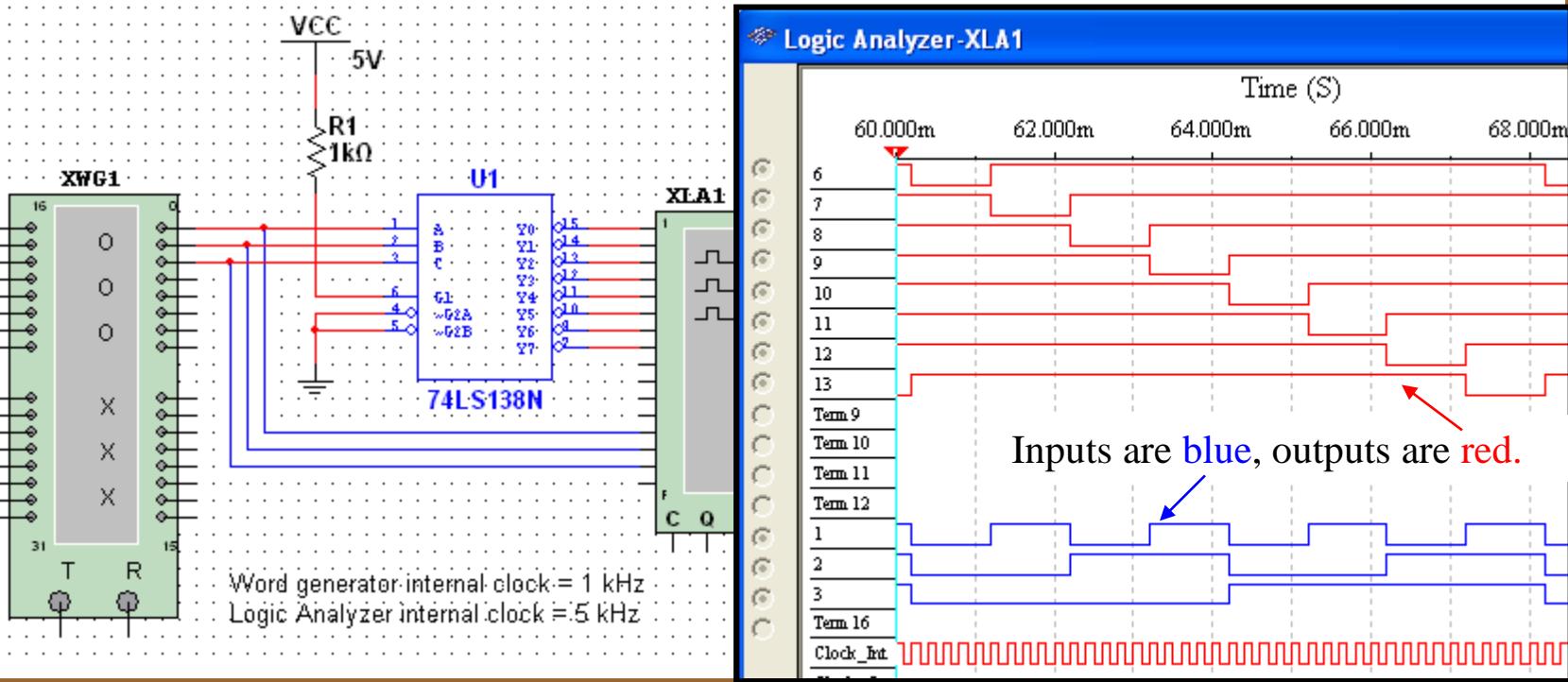




Summary

Decoders

The 74LS138 is a 3-to-8 decoder with three chip select inputs (two active LOW, one active HIGH). In this Multisim circuit, the word generator (XWG1) is set up as an up counter. The logic analyzer (XLA1) compares the input and outputs of the decoder.

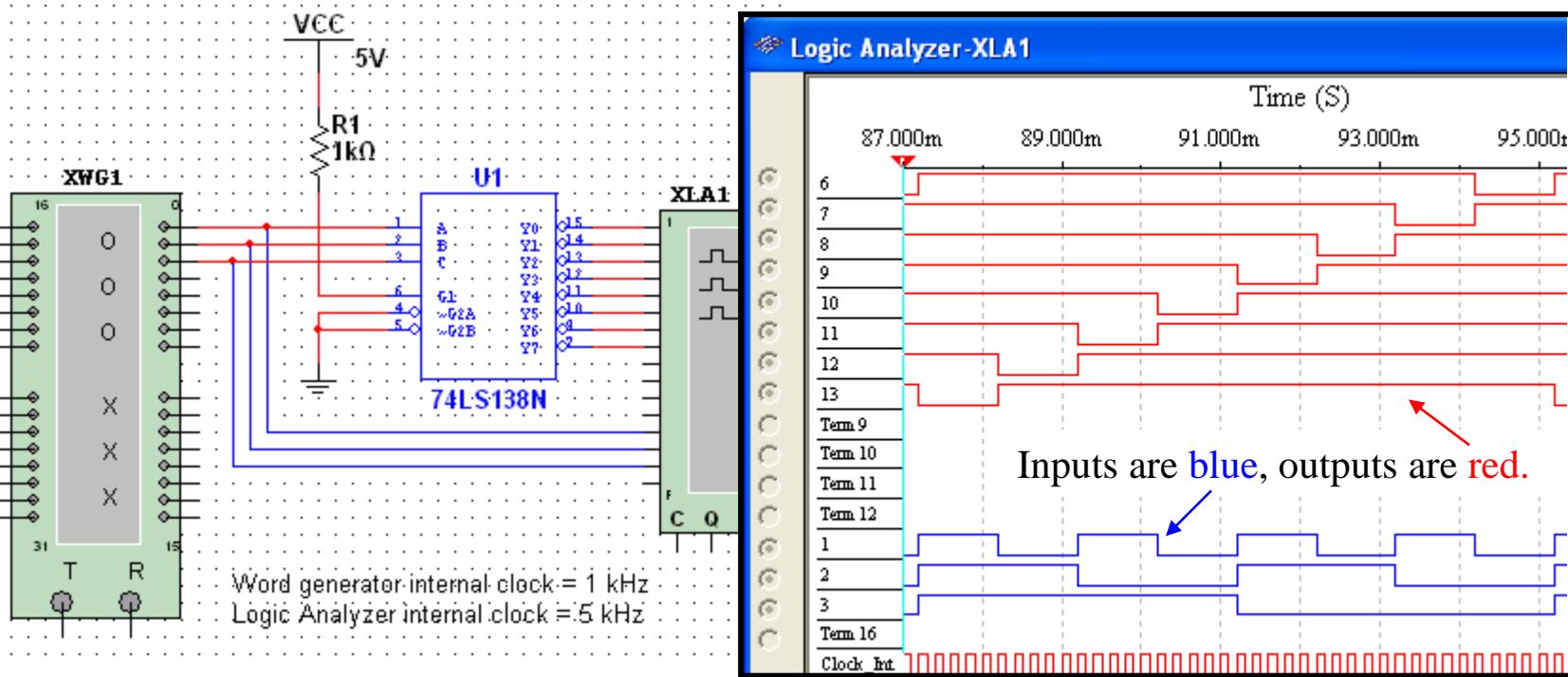


Summary

Decoders

Question

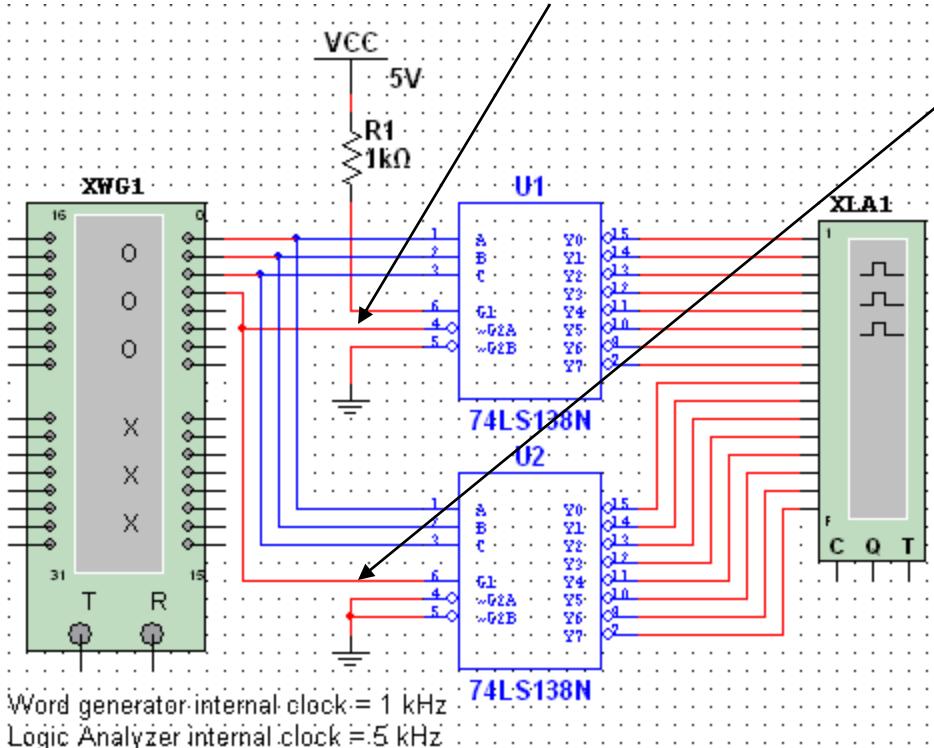
How will the waveforms change if the word generator is configured as a down counter instead of an up counter?





Decoders

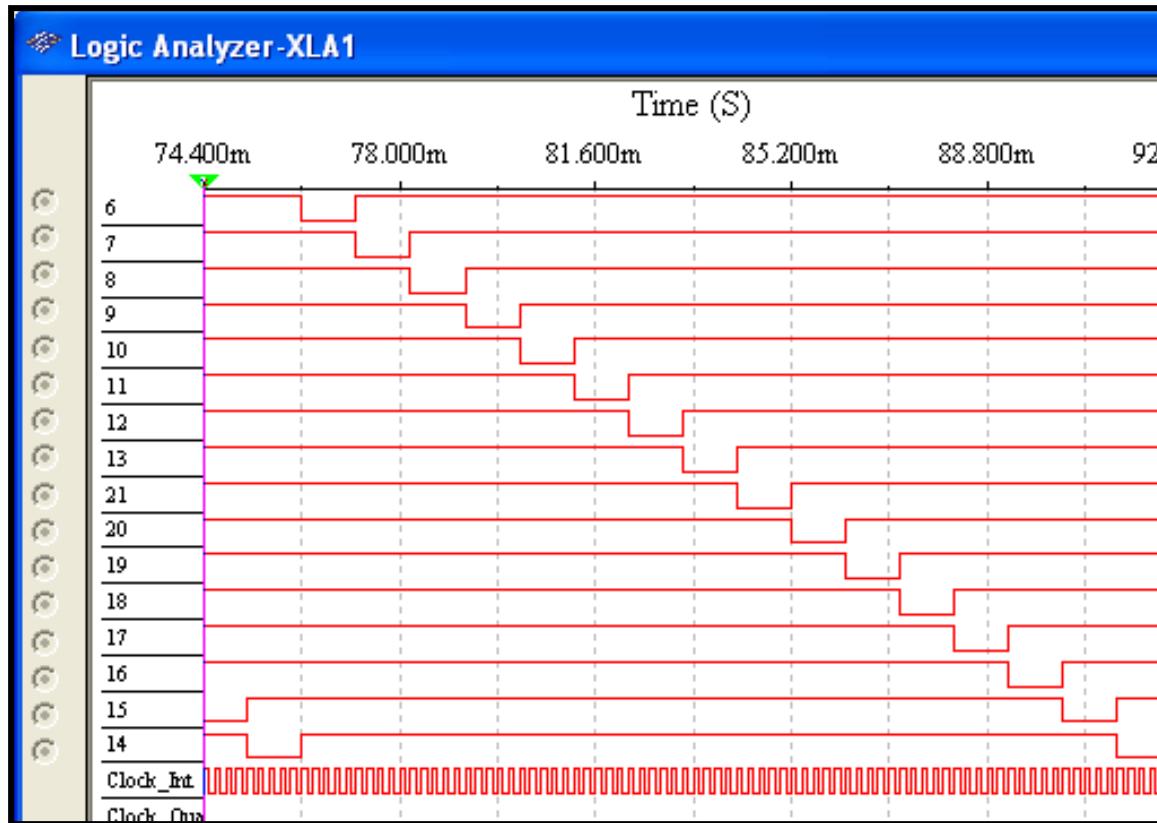
The chip select inputs can be used to expand a decoder. In this circuit, two 74LS138s are configured as a 16 line decoder. Notice how the MSB is connected to one active LOW and one active HIGH chip select.



The next slide
shows the logic
analyzer output...

Summary

Decoders



Question

Is the word generator set as an up counter or a down counter? (The least significant decoder output at the top). **It is an up counter.**

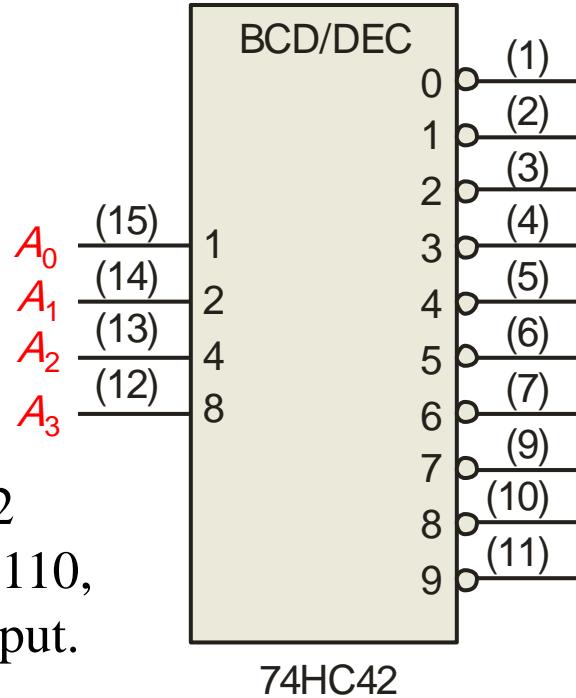
Summary

Decoders

BCD-to-decimal decoders accept a binary coded decimal input and activate one of ten possible decimal digit indications.

Example

Assume the inputs to the 74HC42 decoder are the sequence 0101, 0110, 0011, and 0010. Describe the output.



Solution

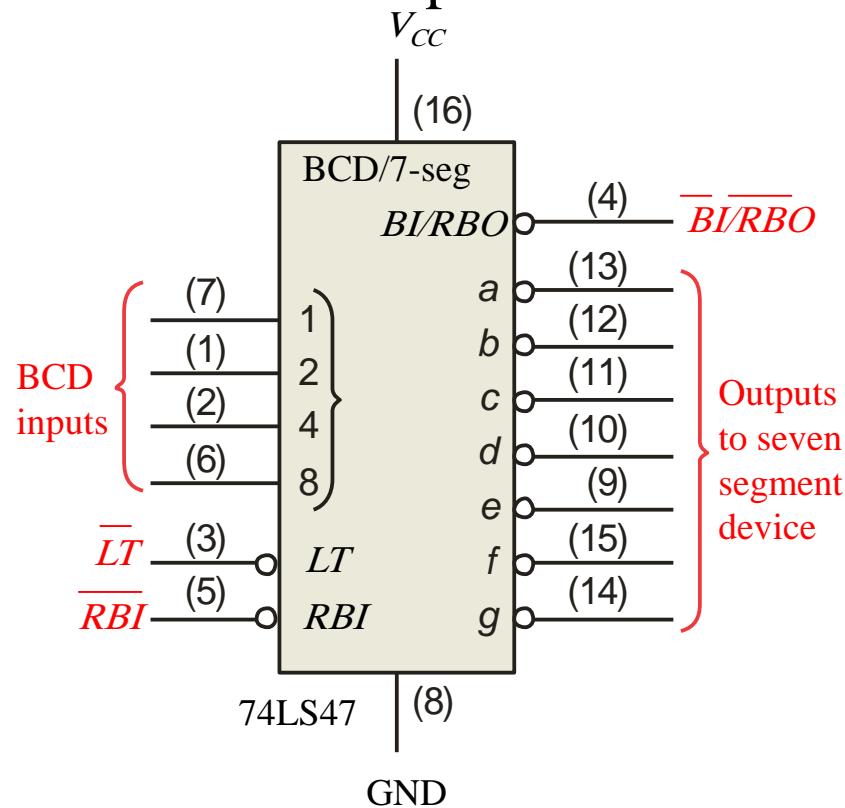
All lines are HIGH except for one active output, which is LOW. The active outputs are 5, 6, 3, and 2 in that order.

Summary

BCD Decoder/Driver

Another useful decoder is the 74LS47. This is a BCD-to-seven segment display with active LOW outputs.

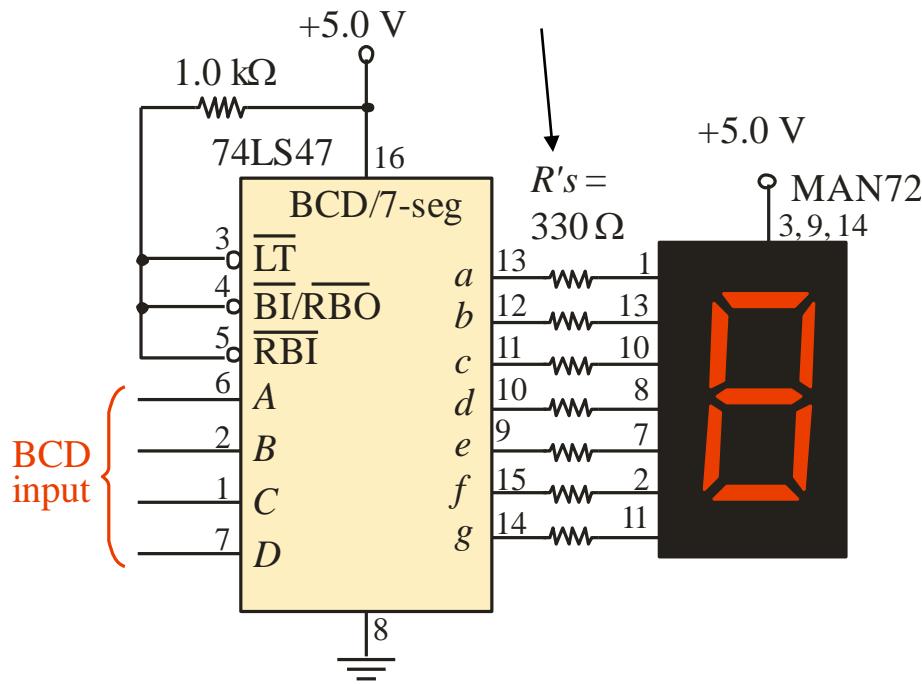
The *a-g* outputs are designed for much higher current than most devices (hence the word driver in the name).



Summary

BCD Decoder/Driver

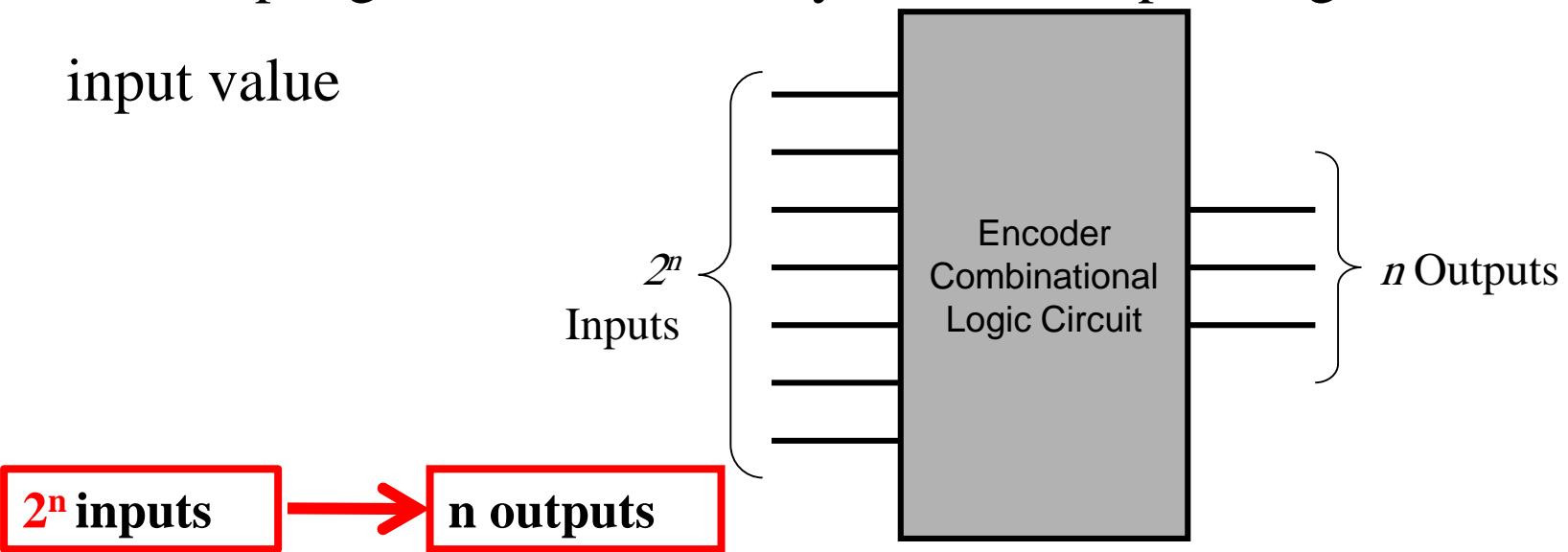
Here the 7447A is connected to an LED seven segment display. Notice the current limiting resistors, required to prevent overdriving the LED display.



Summary

Encoders

- Performs the inverse operation of a decoder
- Has 2^n or fewer input lines and n output lines
- The output generates the binary code corresponding to the input value





Encoders

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

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$



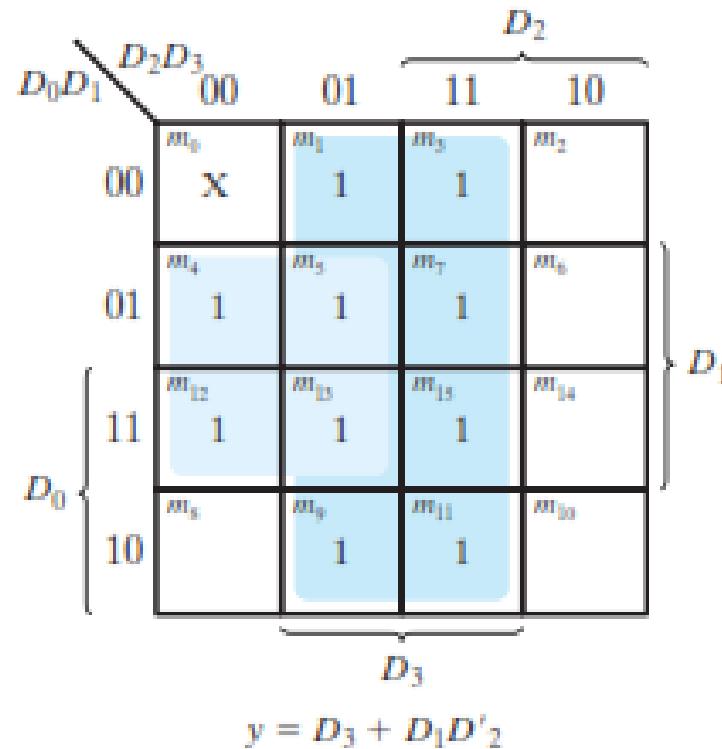
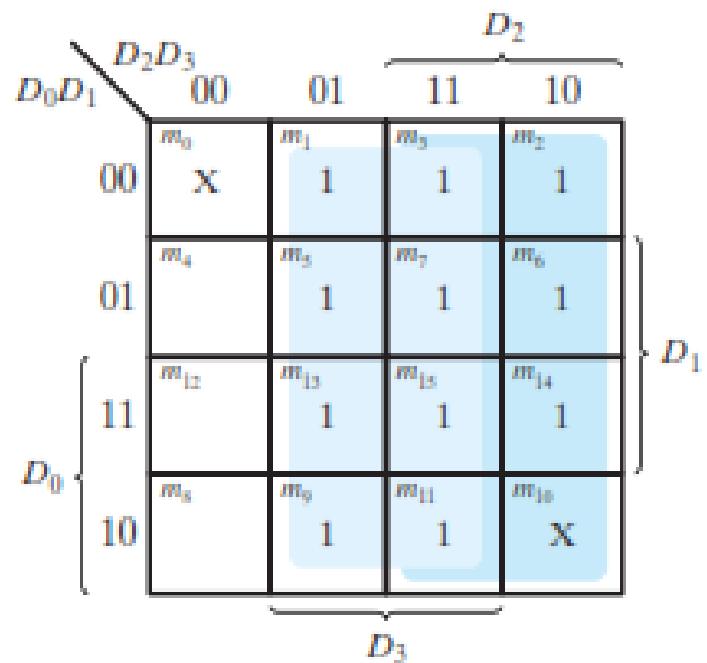
Priority Encoder

Truth Table of a Priority Encoder

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

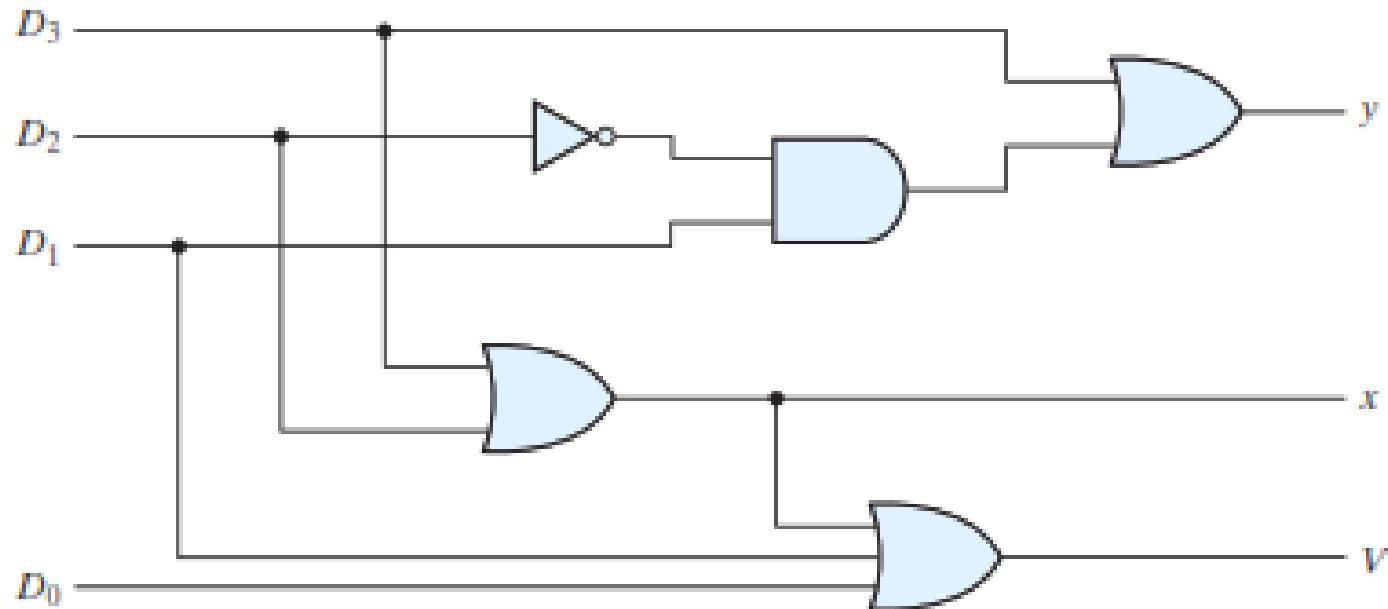


Priority Encoder





Priority Encoder



$$x = D_2 + D_3$$

Four-input priority encoder

$$y = D_3 + D_1 \cdot D_2'$$

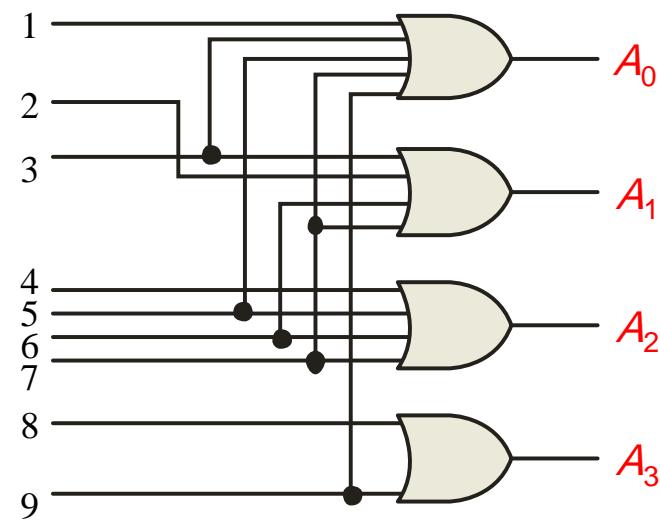
$$v = D_0 + D_1 + D_2 + D_3$$

Summary

Encoders

An **encoder** accepts an active logic level on one of its inputs and converts it to a coded output, such as BCD or binary.

The decimal to BCD is an encoder with an input for each of the ten decimal digits and four outputs that represent the BCD code for the active digit. The basic logic diagram is shown. There is no zero input because the outputs are all LOW when the input is zero.



Summary

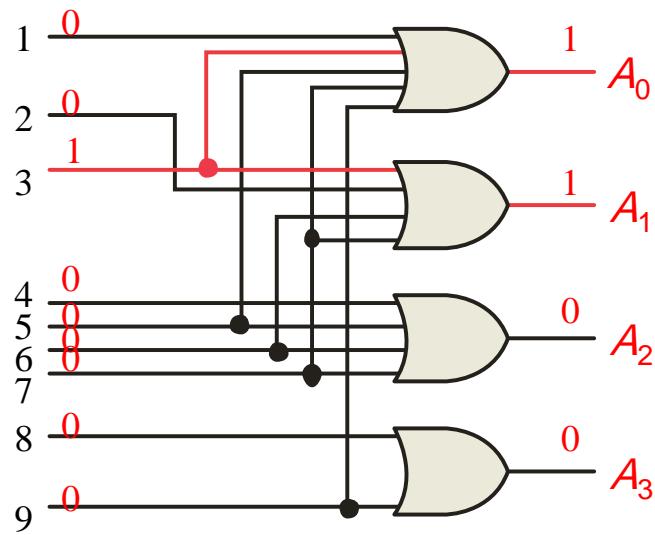
Encoders

Example

Solution

Show how the decimal-to-BCD encoder converts the decimal number 3 into a BCD 0011.

The top two OR gates have ones as indicated with the red lines. Thus the output is 0111.

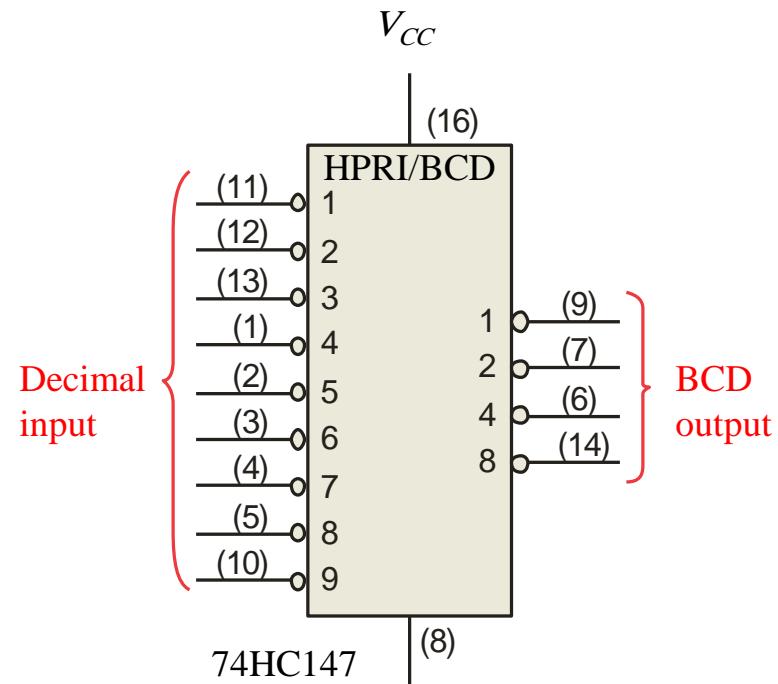


Summary

Encoders

The 74HC147 is an example of an IC encoder. It has ten active-LOW inputs and converts the active input to an active-LOW BCD output.

This device offers additional flexibility in that it is a **priority encoder**. This means that if more than one input is active, the one with the highest order decimal digit will be active.

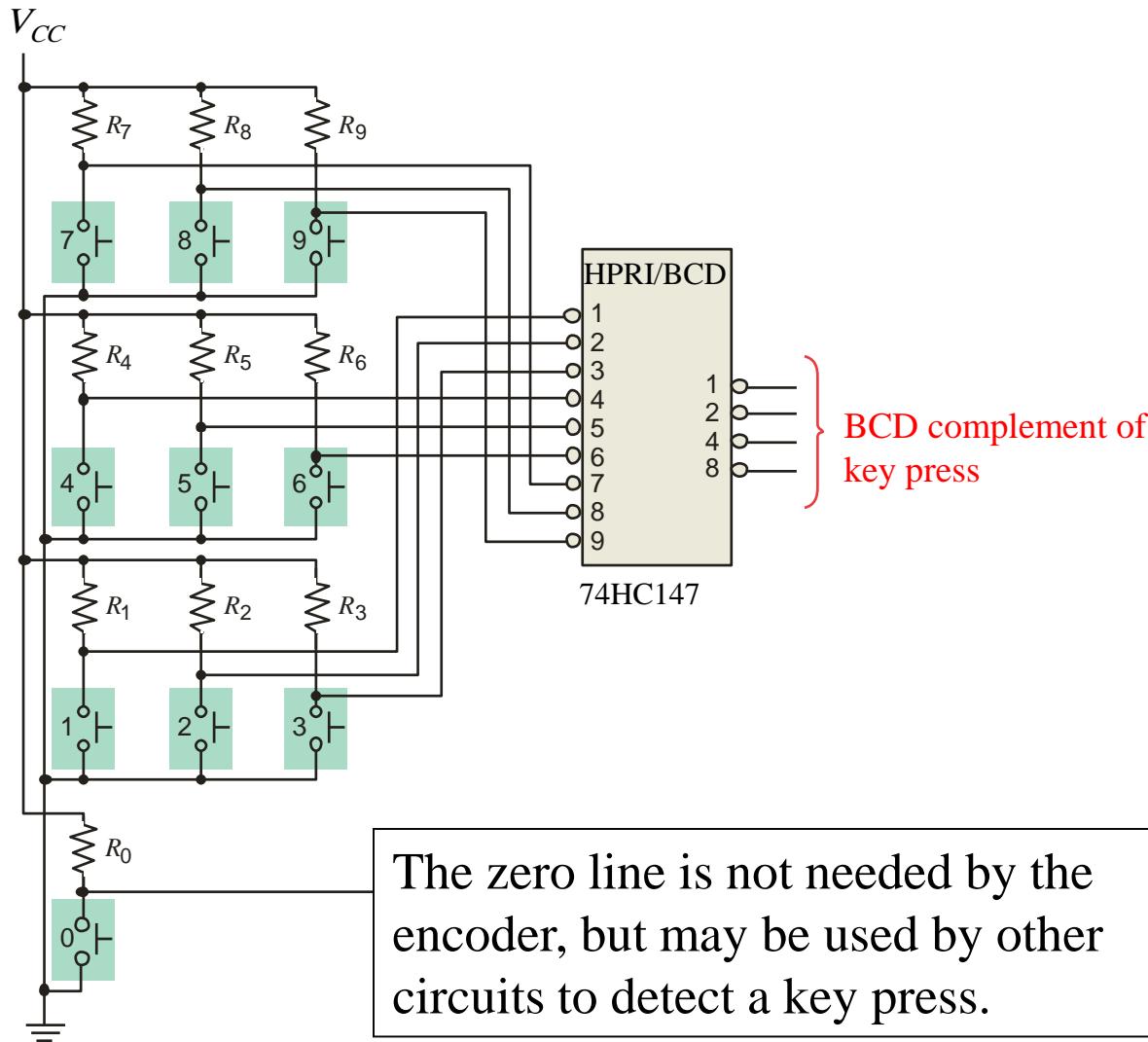


The next slide shows an application ...

Summary

Encoders

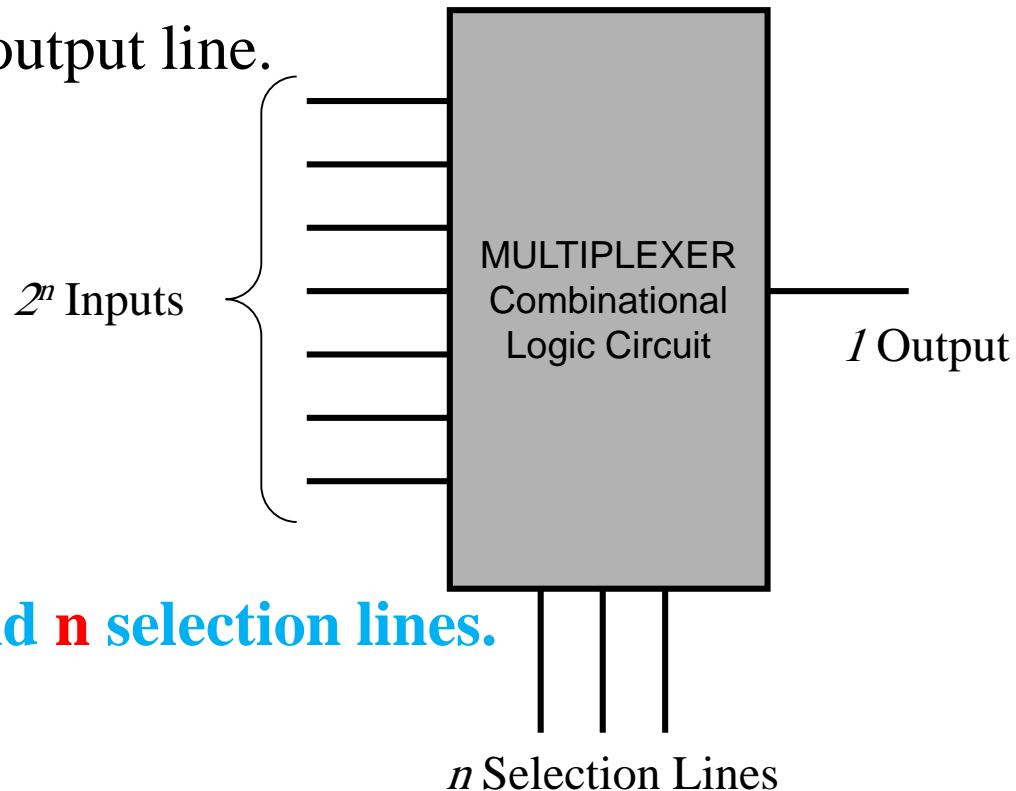
Keyboard
encoder



Summary

Multiplexers

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs the information to a single output line.



Normally 2^n inputs and n selection lines.

Summary

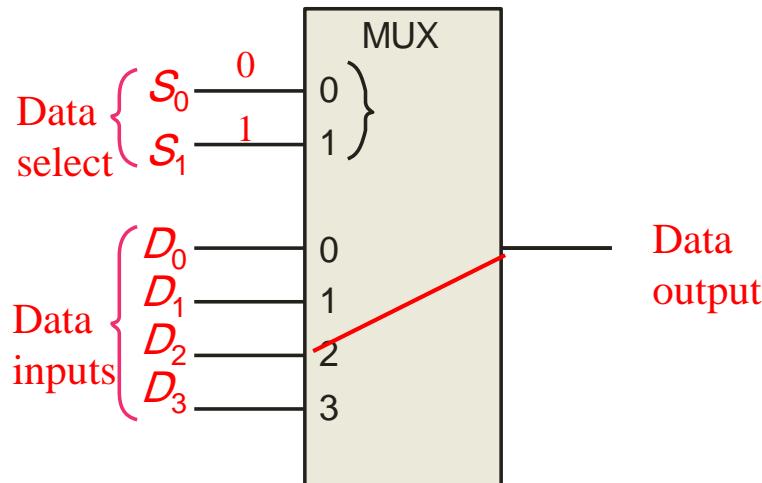
Multiplexers

A multiplexer (MUX) selects one data line from two or more input lines and routes data from the selected line to the output. The particular data line that is selected is determined by the select inputs.

Two select lines are shown here to choose any of the four data inputs.

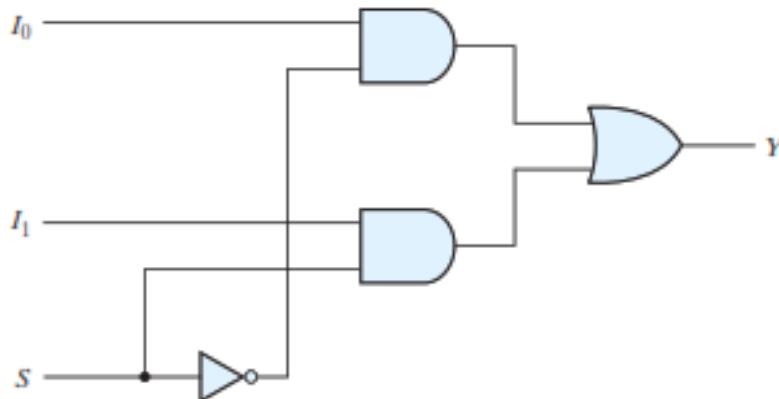
Question

Which data line is selected if $S_1S_0 = 10$? D_2

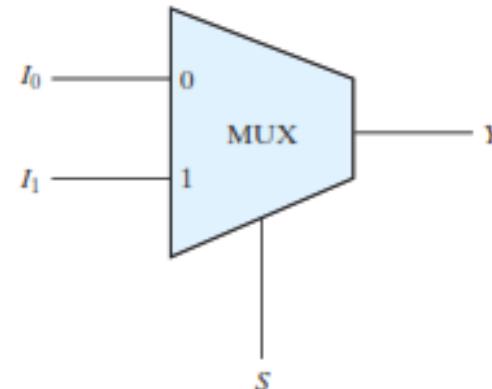


Summary

Two-to-one Multiplexers



(a) Logic diagram



(b) Block diagram

Two 1-bit inputs (I_0, I_1) to one output (y).

Selection through S .

$S=0$, I_0 has a path ,

$S=1$, I_1 has a path

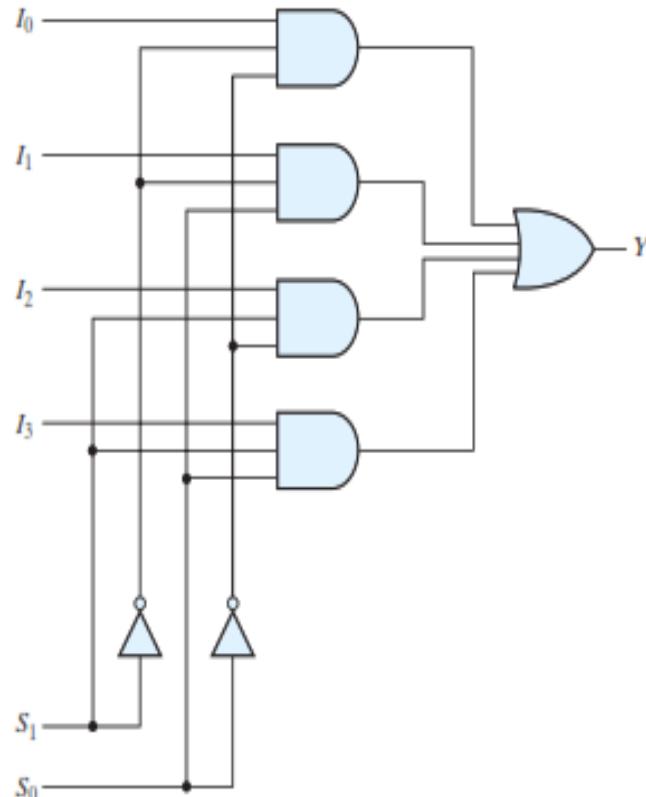
Summary

Four to one Multiplexer

I₀,I₁,I₂,I₃ Inputs

Y- output

S₀,S₁- selection



(a) Logic diagram

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

(b) Function table



Summary

Multiplexers With Boolean Function

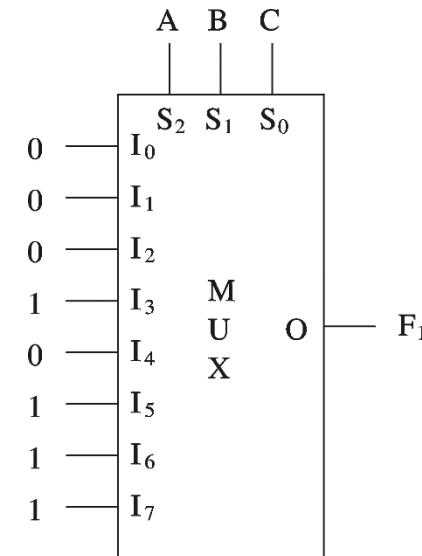
$$F(A,B,C) = \sum(3, 5, 6, 7)$$

Original truth table

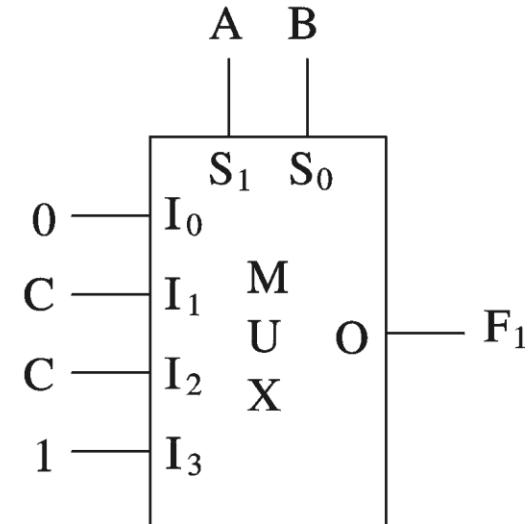
A	B	C	F_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

New truth table

A	B	F_1
0	0	0
0	1	C
1	0	C
1	1	1



Majority function



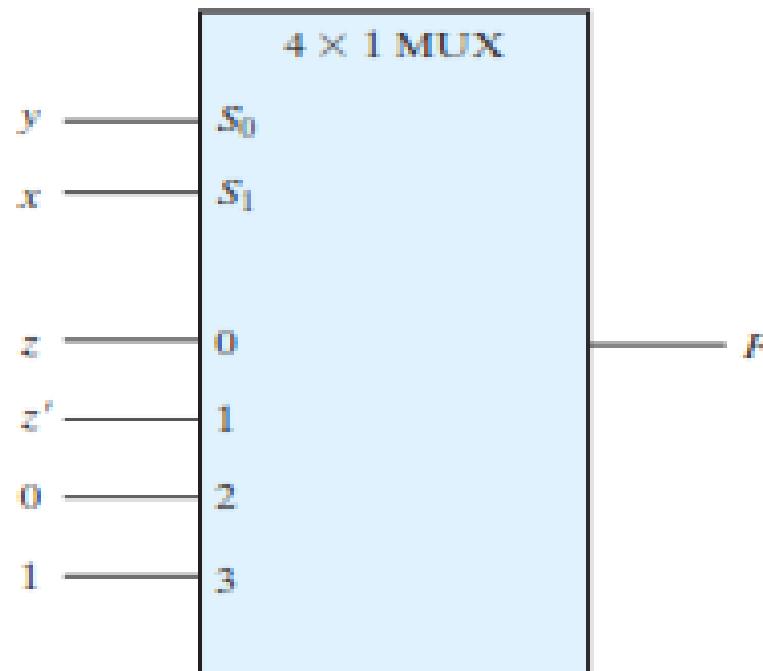
Summary

Multiplexers With Boolean Function

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

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

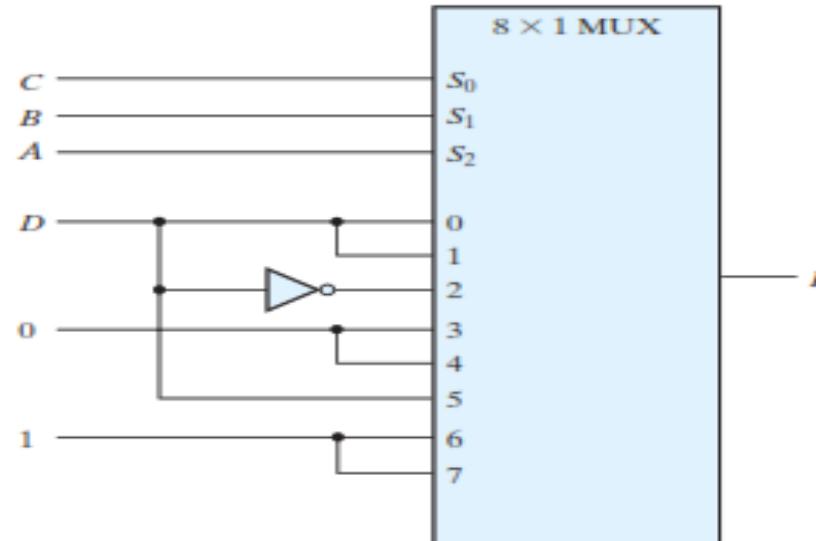


Summary

Multiplexers With Boolean Function

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

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



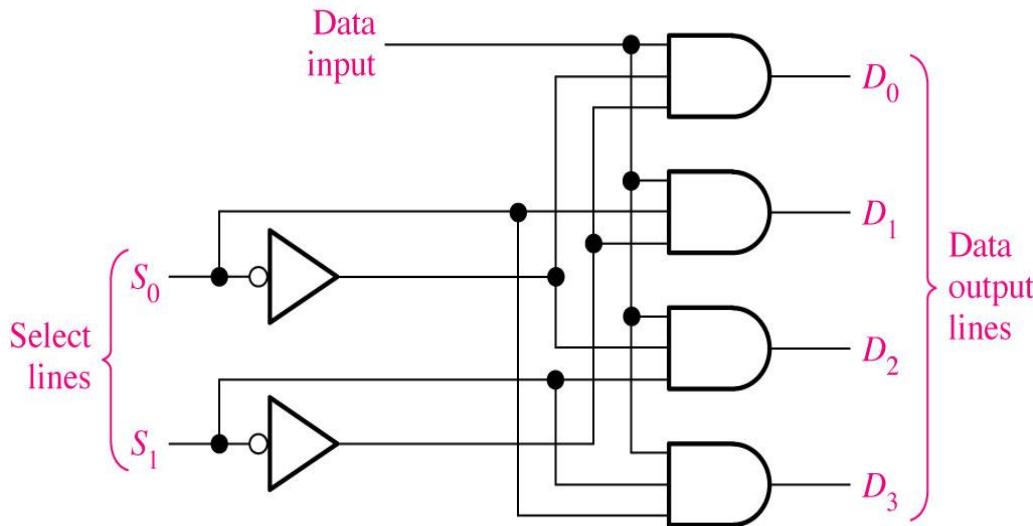
$$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$$

$$ABC=101 \quad F=D$$

Summary

Demultiplexers

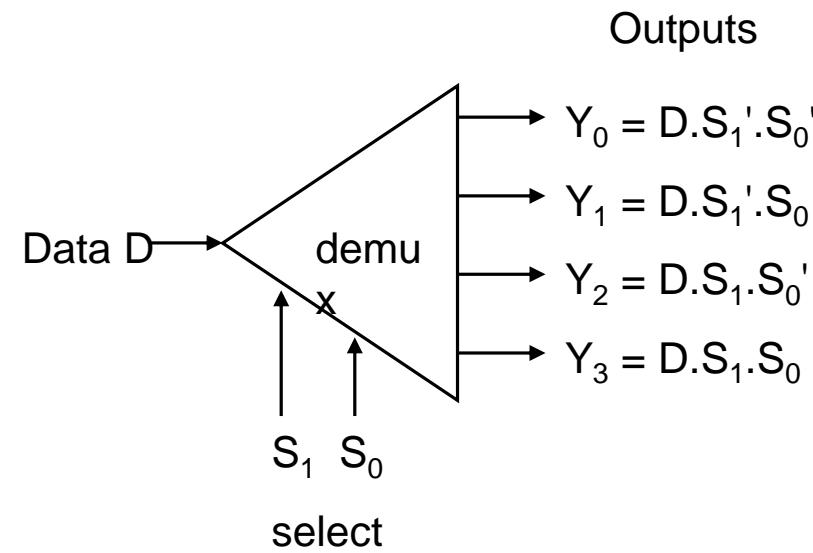
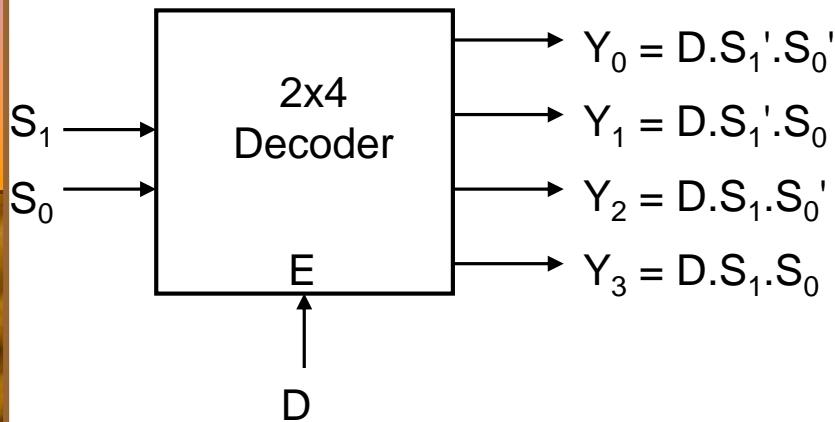
- ❑ Performs inverse operation of a multiplexer
- ❑ A combinational circuit that receives input from a single line and transmits it to one of 2^n possible output lines
- ❑ The selection of the specific output is controlled by the bit combination of n selection lines



Summary

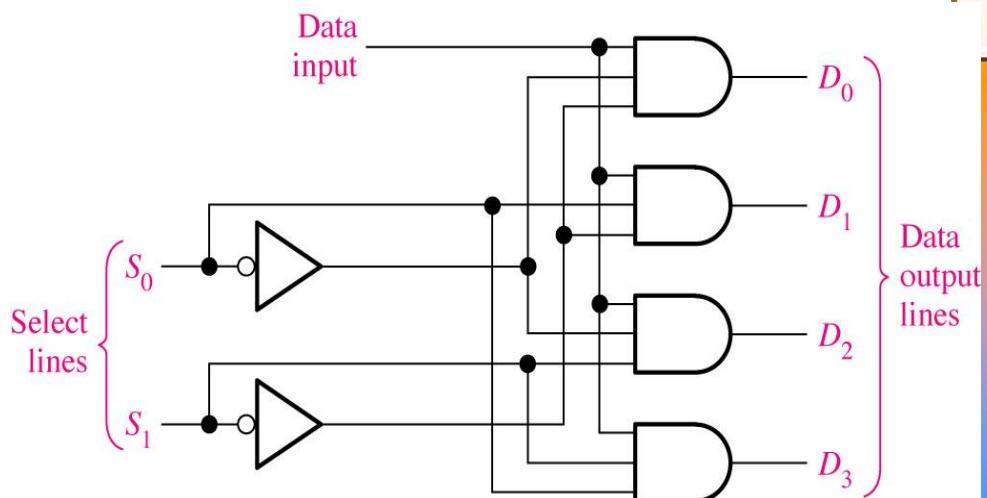
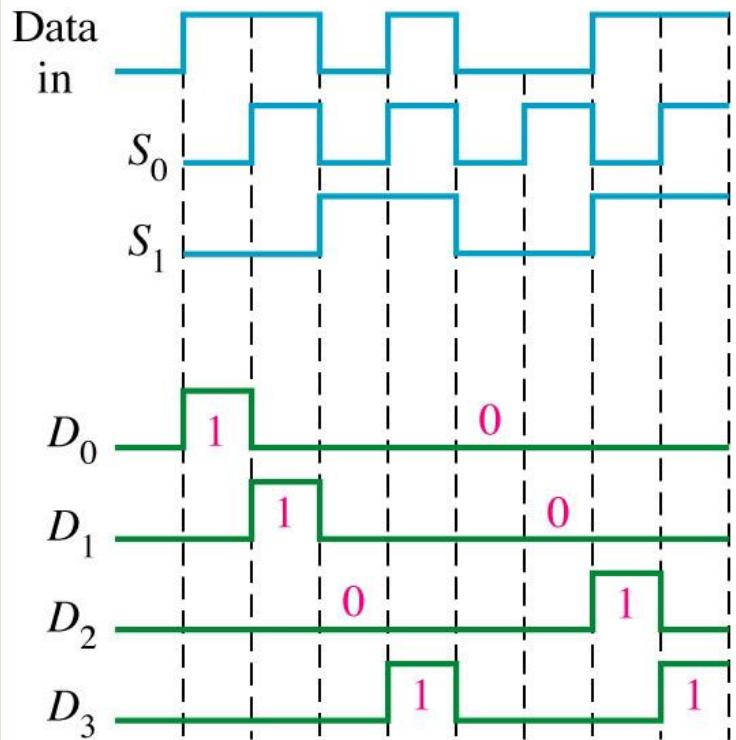
Demultiplexers

- Given an input line & a set of selection lines, the demultiplexer will direct data from input to a selected output line.
- An example of a 1-to-4 demultiplexer:
- Same as a Decoder with an enable – how?



Summary

Demultiplexers



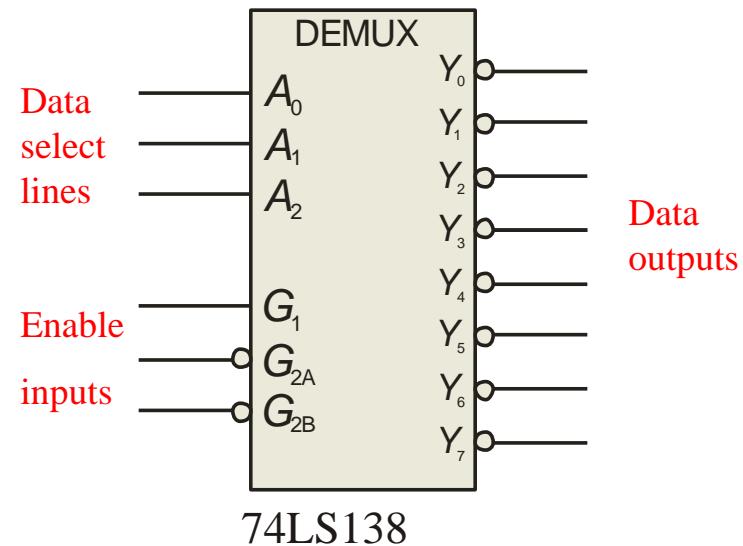
A 1-line-to-4-line demultiplexer.

Summary

Demultiplexers

A demultiplexer (DEMUX) performs the opposite function from a MUX. It switches data from one input line to two or more data lines depending on the select inputs.

The 74LS138 was introduced previously as a decoder but can also serve as a DEMUX. When connected as a DEMUX, data is applied to one of the enable inputs, and routed to the selected output line depending on the select variables. Note that the outputs are active-LOW as illustrated in the following example...

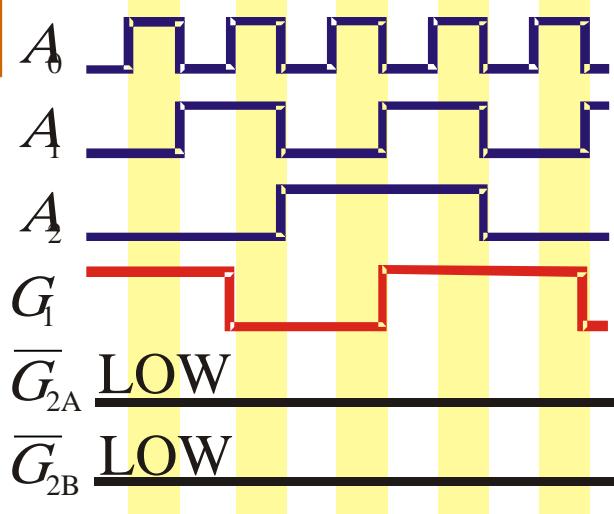
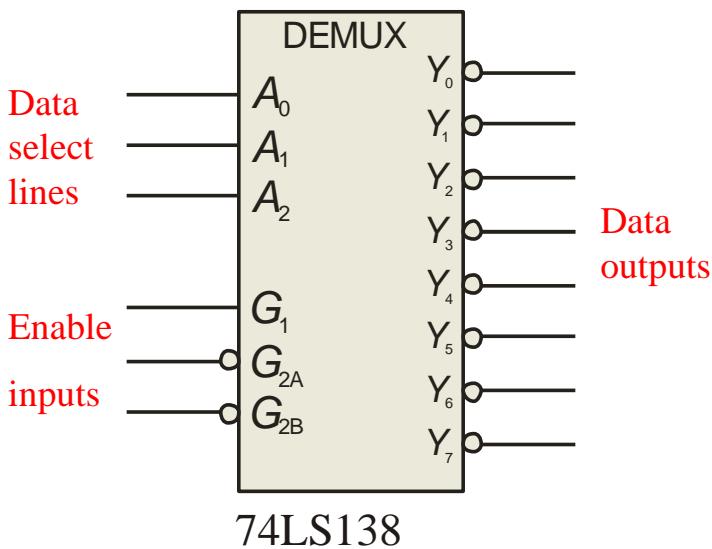


Summary

Demultiplexers

Example Solution

Determine the outputs, given the inputs shown. The output logic is opposite to the input because of the active-LOW convention. (Red shows the selected line).



Summary

Parity Generators/Checkers

Parity is an error detection method that uses an extra bit appended to a group of bits to force them to be either odd or even. In even parity, the total number of ones is even; in odd parity the total number of ones is odd.



Example

The ASCII letter S is 1010011. Show the parity bit for the letter S with odd and even parity.

Solution

S with odd parity = 11010011

S with even parity = 01010011

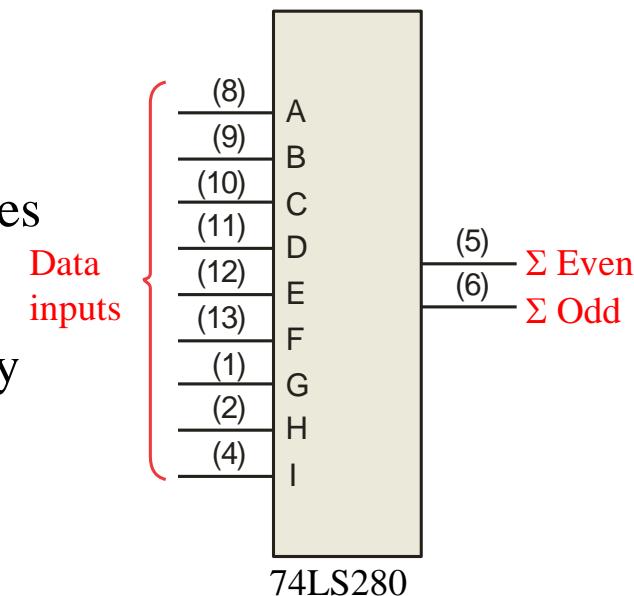
Summary

Parity Generators/Checkers

The 74LS280 can be used to generate a parity bit or to check an incoming data stream for even or odd parity.

Checker: The 74LS280 can test codes with up to 9 bits. The even output will normally be HIGH if the data lines have even parity; otherwise it will be LOW. Likewise, the odd output will normally be HIGH if the data lines have odd parity; otherwise it will be LOW.

Generator: To generate even parity, the parity bit is taken from the odd parity output. To generate odd parity, the output is taken from the even parity output.



Selected Key Terms

- Full-adder** A digital circuit that adds two bits and an input carry bit to produce a sum and an output carry.
- Cascading** Connecting two or more similar devices in a manner that expands the capability of one device.
- Ripple carry** A method of binary addition in which the output carry from each adder becomes the input carry of the next higher order adder.
- Look-ahead carry** A method of binary addition whereby carries from the preceding adder stages are anticipated, thus eliminating carry propagation delays.

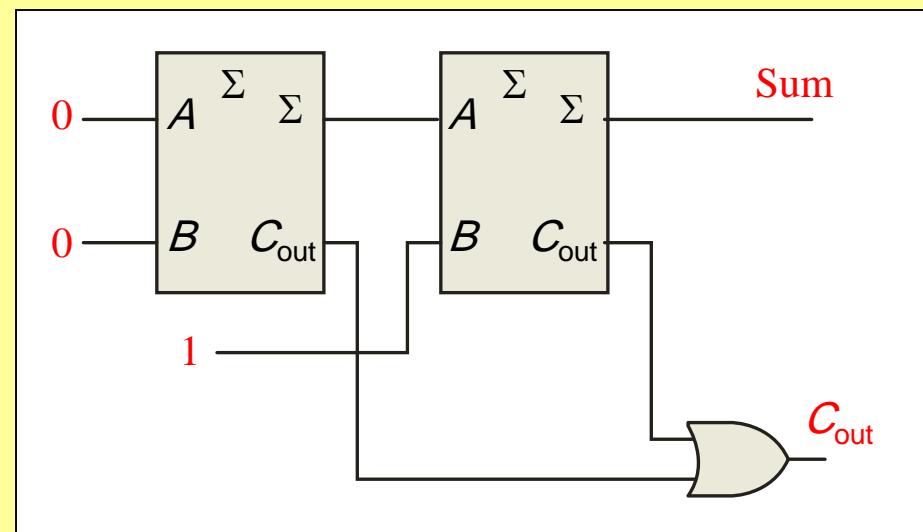
Selected Key Terms

- Decoder** A digital circuit that converts coded information into a familiar or noncoded form.
- Encoder** A digital circuit that converts information into a coded form.
- Priority encoder** An encoder in which only the highest value input digit is encoded and any other active input is ignored.
- Multiplexer (MUX)** A circuit that switches digital data from several input lines onto a single output line in a specified time sequence.
- Demultiplexer (DEMUX)** A circuit that switches digital data from one input line onto a several output lines in a specified time sequence.

Quiz

1. For the full-adder shown, assume the input bits are as shown with $A = 0$, $B = 0$, $C_{\text{in}} = 1$. The **Sum** and C_{out} will be

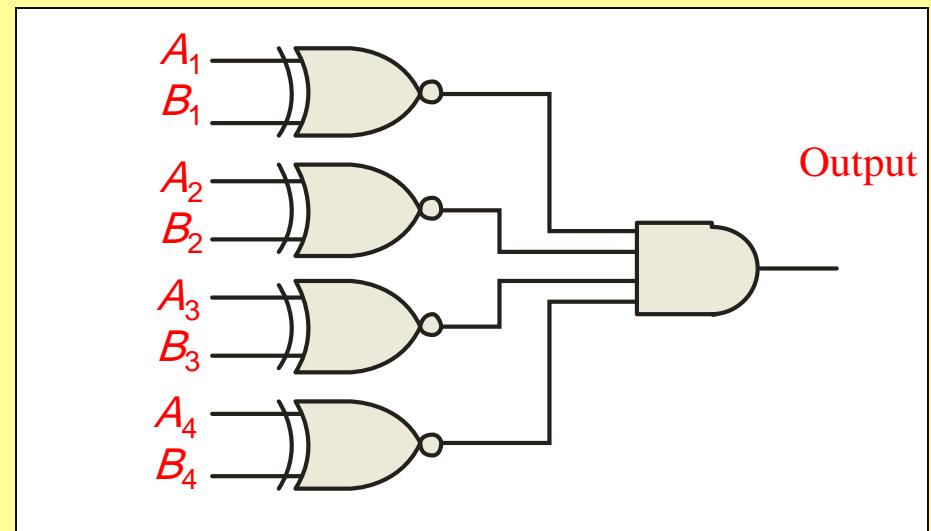
- a. Sum = 0 $C_{\text{out}} = 0$
- b. Sum = 0 $C_{\text{out}} = 1$
- c. Sum = 1 $C_{\text{out}} = 0$
- d. Sum = 1 $C_{\text{out}} = 1$



Quiz

2. The output will be LOW if

- a. $A < B$
- b. $A > B$
- c. both a and b are correct
- d. $A = B$



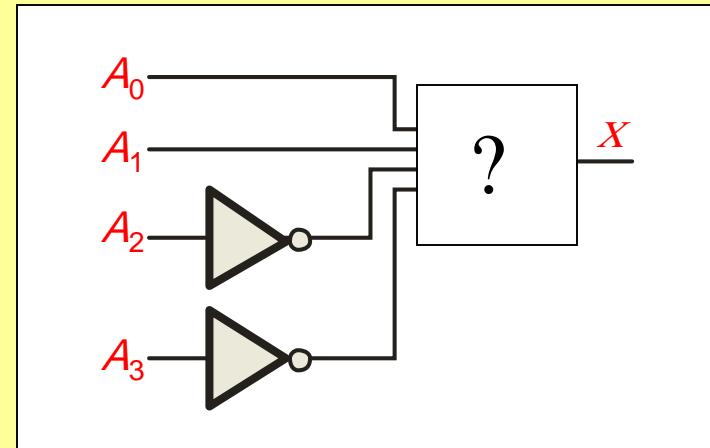
Quiz

3. If you expand two 4-bit comparators to accept two 8-bit numbers, the output of the least significant comparator is
- a. equal to the final output
 - b. connected to the cascading inputs of the most significant comparator
 - c. connected to the output of the most significant comparator
 - d. not used

Quiz

4. Assume you want to decode the binary number 0011 with an active-LOW decoder. The missing gate should be

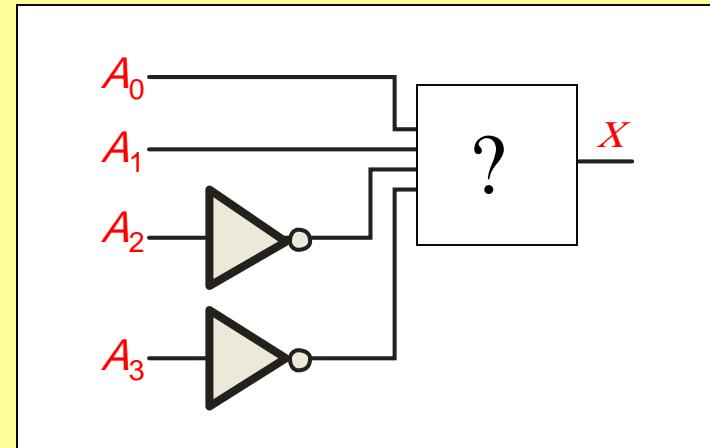
- a. an AND gate
- b. an OR gate
- c. a NAND gate
- d. a NOR gate



Quiz

5. Assume you want to decode the binary number 0011 with an active-HIGH decoder. The missing gate should be

- a. an AND gate
- b. an OR gate
- c. a NAND gate
- d. a NOR gate



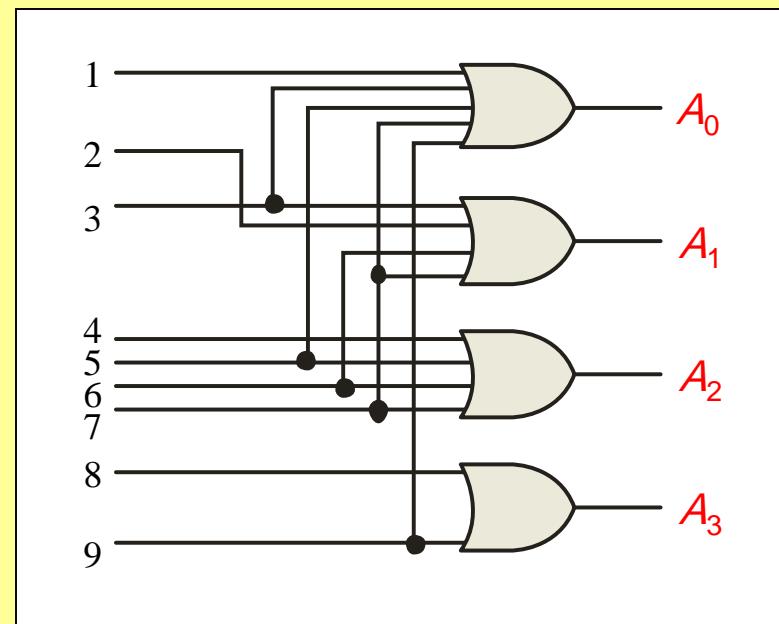
Quiz

6. The 74138 is a 3-to-8 decoder. Together, two of these ICs can be used to form one 4-to-16 decoder. To do this, connect
- a. one decoder to the LSBs of the input; the other decoder to the MSBs of the input
 - b. all chip select lines to ground
 - c. all chip select lines to their active levels
 - d. one chip select line on each decoder to the input MSB

Quiz

7. The decimal-to-binary encoder shown does not have a zero input. This is because

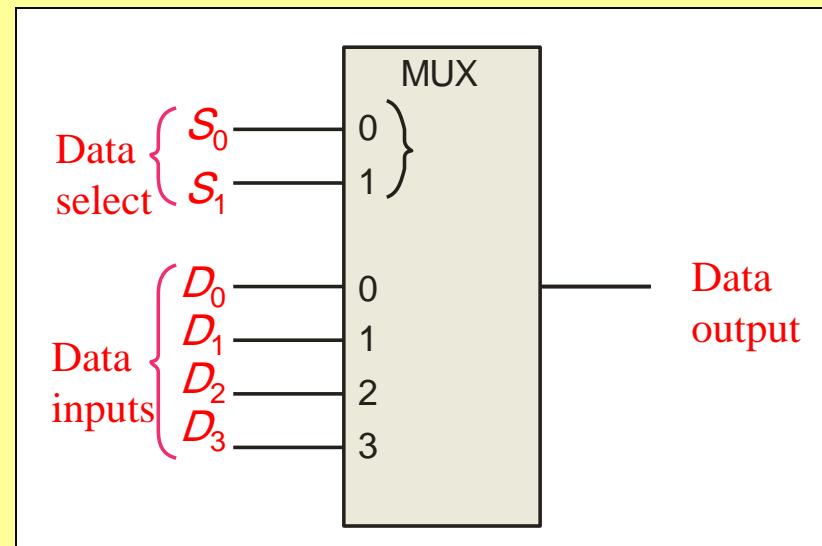
- a. when zero is the input, all lines should be LOW
- b. zero is not important
- c. zero will produce illegal logic levels
- d. another encoder is used for zero



Quiz

8. If the data select lines of the MUX are $S_1S_0 = 11$, the output will be

- a. LOW
- b. HIGH
- c. equal to D_0
- d. equal to D_3



Quiz

9. The 74138 decoder can also be used as
- a. an encoder
 - b. a DEMUX
 - c. a MUX
 - d. none of the above

Quiz

10. The 74LS280 can generate even or odd parity. It can also be used as
- a. an adder
 - b. a parity tester
 - c. a MUX
 - d. an encoder

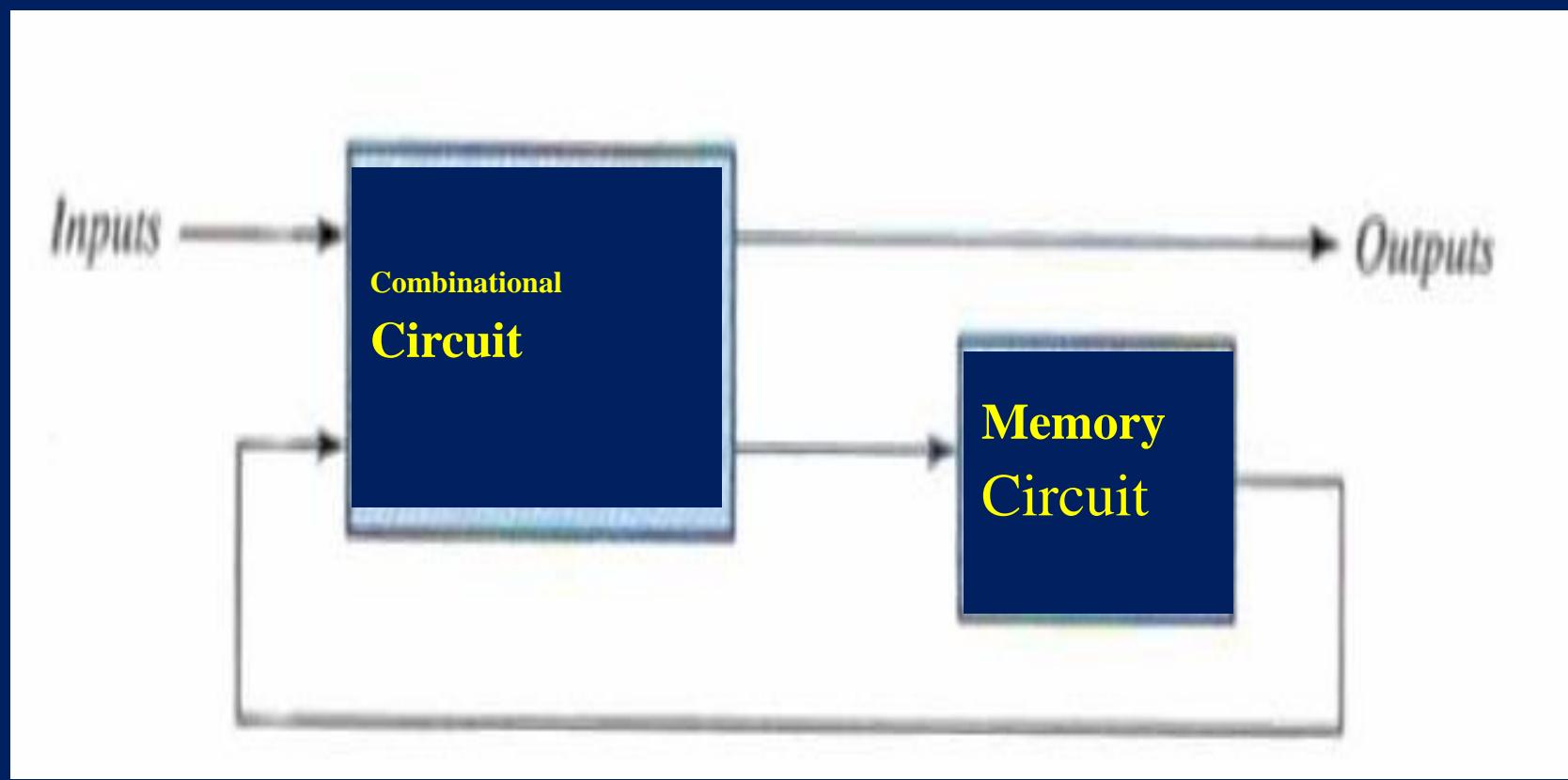
Quiz

Answers:

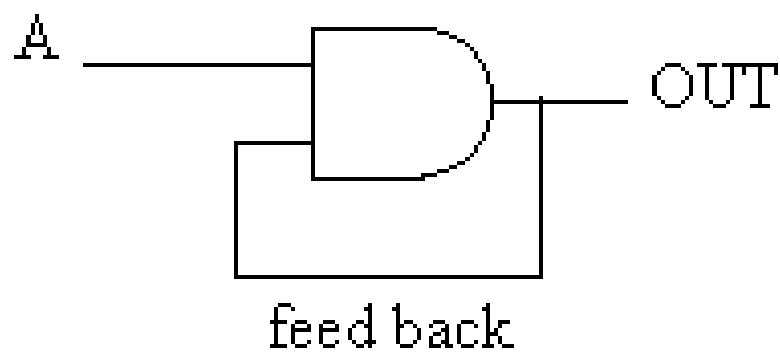
- | | |
|------|-------|
| 1. c | 6. d |
| 2. c | 7. a |
| 3. b | 8. d |
| 4. c | 9. b |
| 5. a | 10. b |

Chapter 7

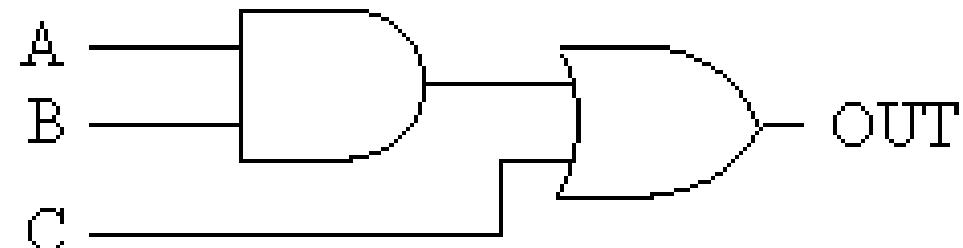
Synchronous Sequential Logic



SEQUENTIAL CIRCUITS AND FEEDBACK



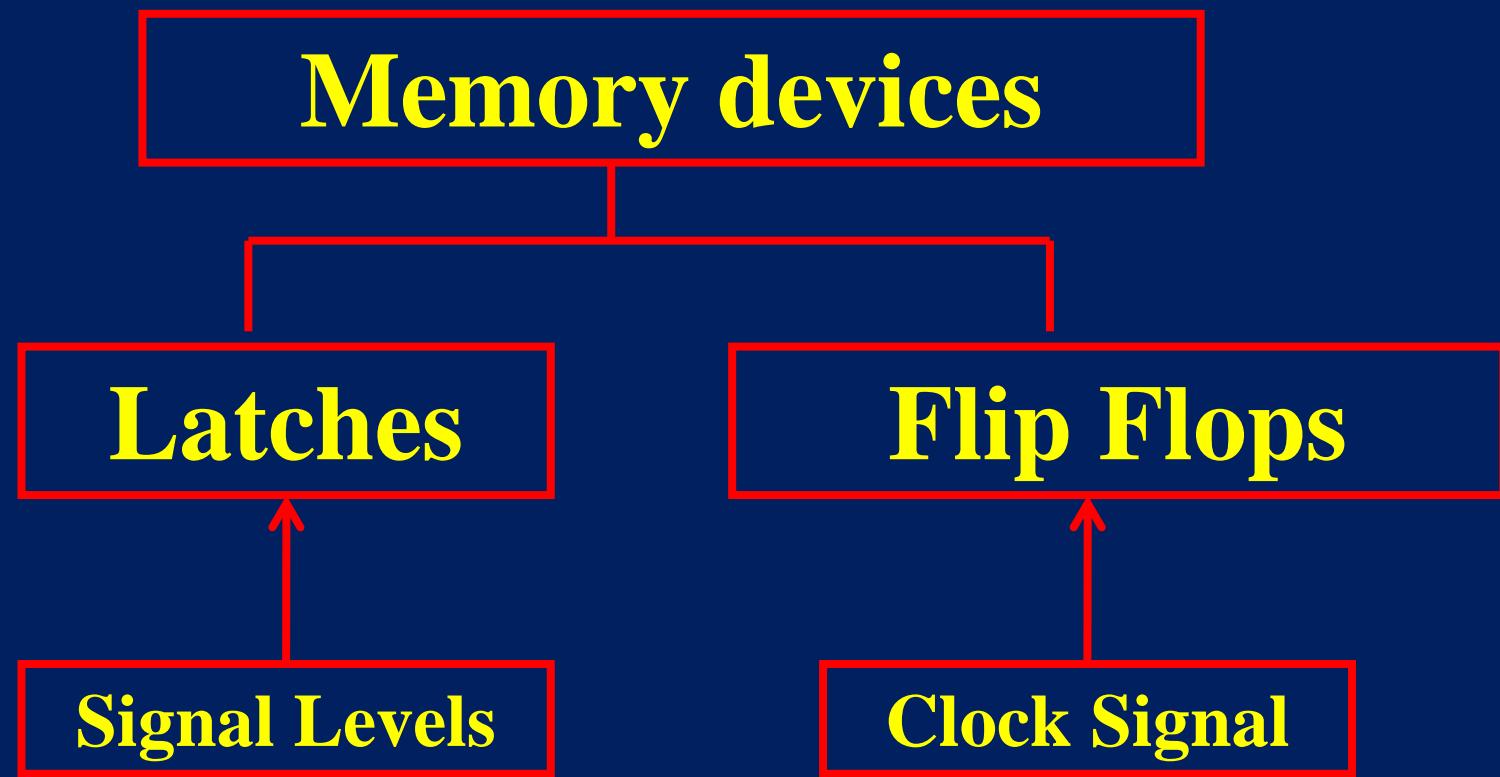
(a)



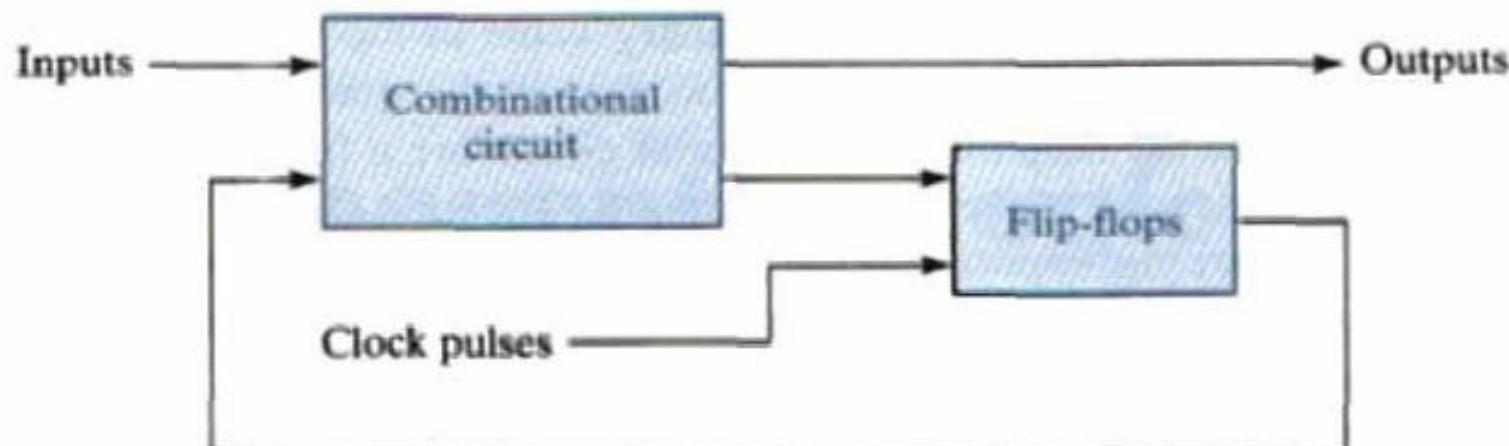
(b)

Fig(1): a: Sequential circuit.
b: Combinational circuit.

Synchronous Sequential circuit (Memory)



Synchronous Sequential Clocked circuit



(a) Block diagram



(b) Timing diagram of clock pulses

Storage Elements

Flip-Flops(triggering)

Latch



(a) Response to positive level

F - F



(b) Positive-edge response



(c) Negative-edge response

SET- RESET (S-R) LATCHES :

- **Cross- NOR S-R latch (active high)**
- **Cross- NAND S-R latch (active low)**

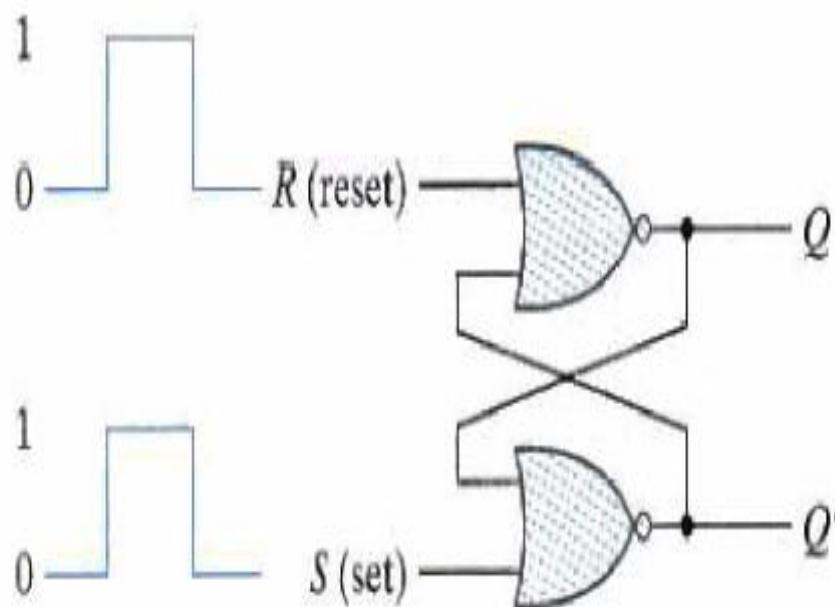
truth table of the NOR gate

A	B	$(A + B)'$
0	0	1
0	1	0
1	0	0
1	1	0

if any of the inputs of the NOR gate is high (logic 1),
the output is low (logic 0)

Storage Elements :Latches

SR Latches with NOR Gates



(a) Logic diagram

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

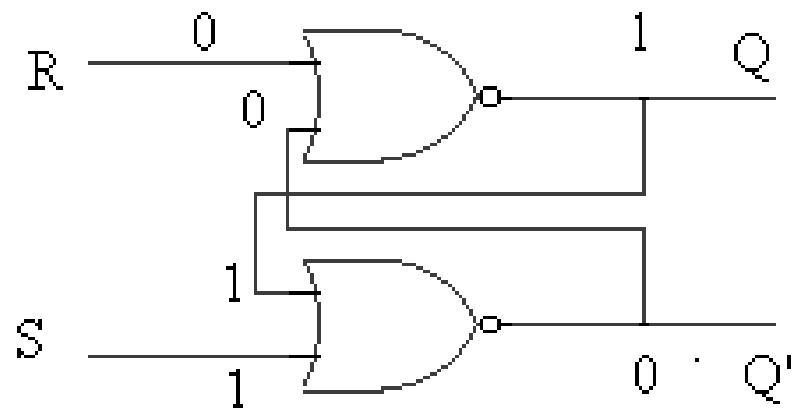
(after $S = 1, R = 0$)

(after $S = 0, R = 1$)

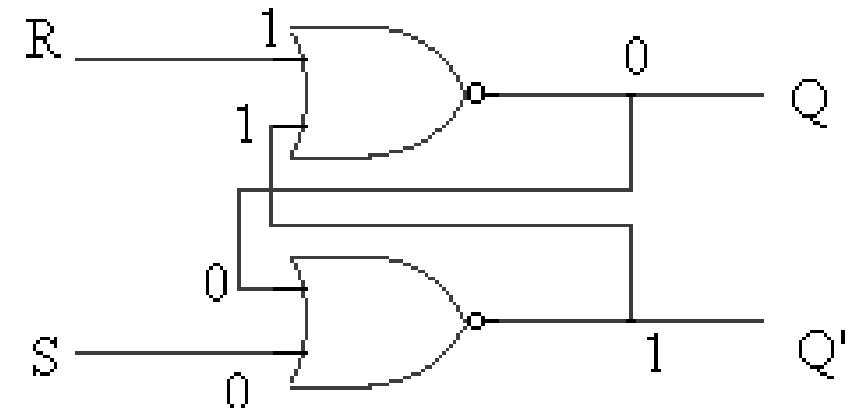
(forbidden)

(b) Function table

Cross-NOR S-R latch (active high)



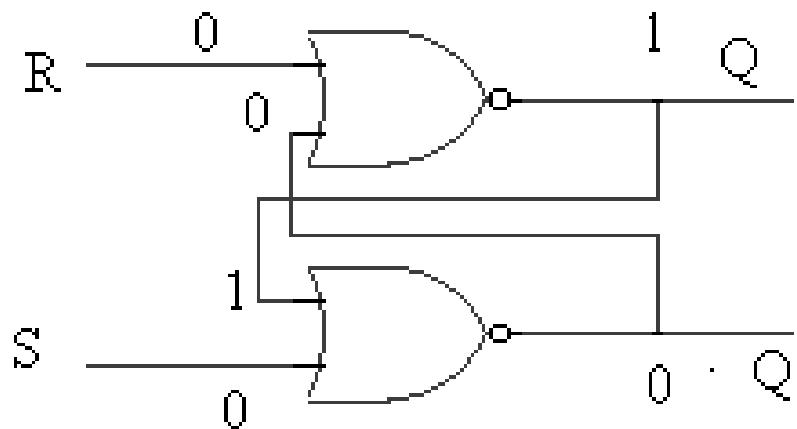
(a): SET condition.



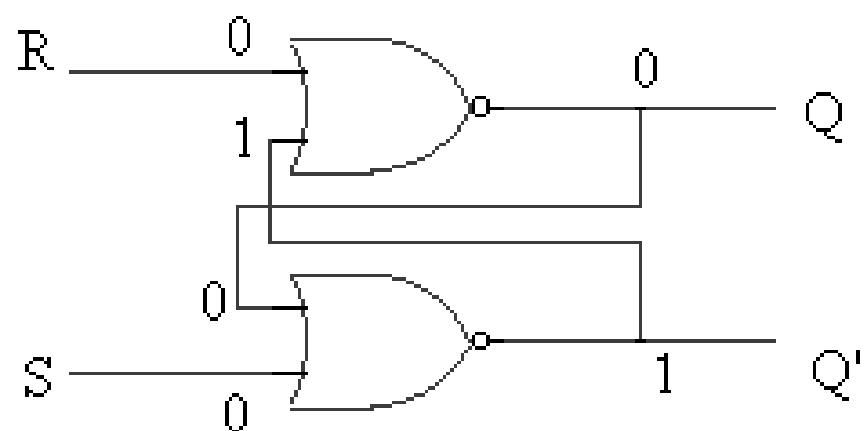
(b): RESET condition.

Fig(3): NOR S-R latch.

No-Change condition in S-R latch



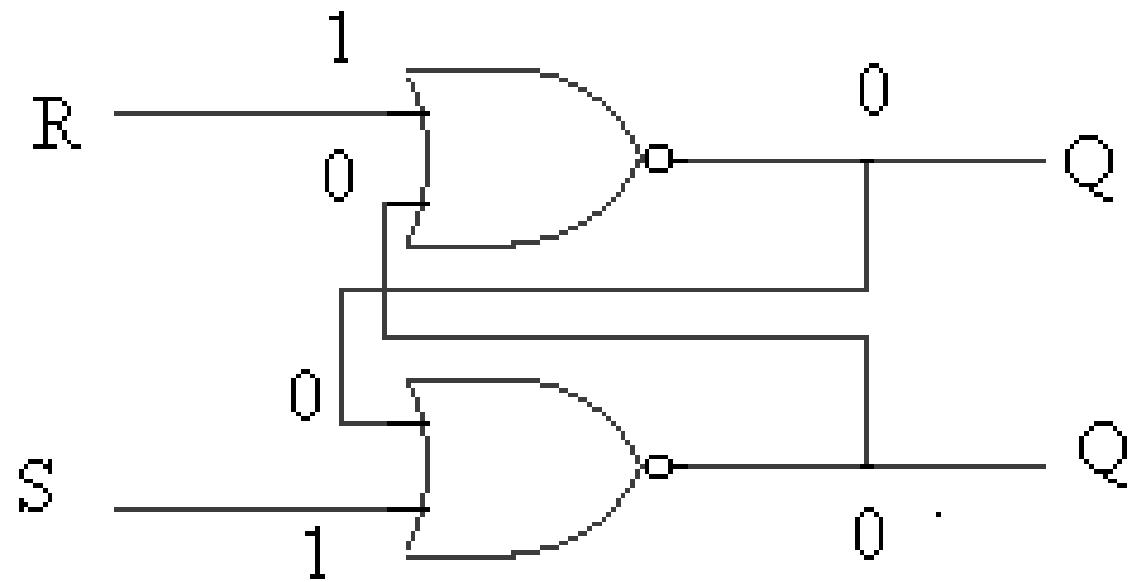
(a): No-Change when the circuit was previously set.



(b): No-Change when the circuit was previously reset.

Fig(4): No-Change condition in S-R latch.

Forbidden condition in S-R latch



Fig(5): Forbidden Condition in S-R Latch.

function table of the NOR S-R latch

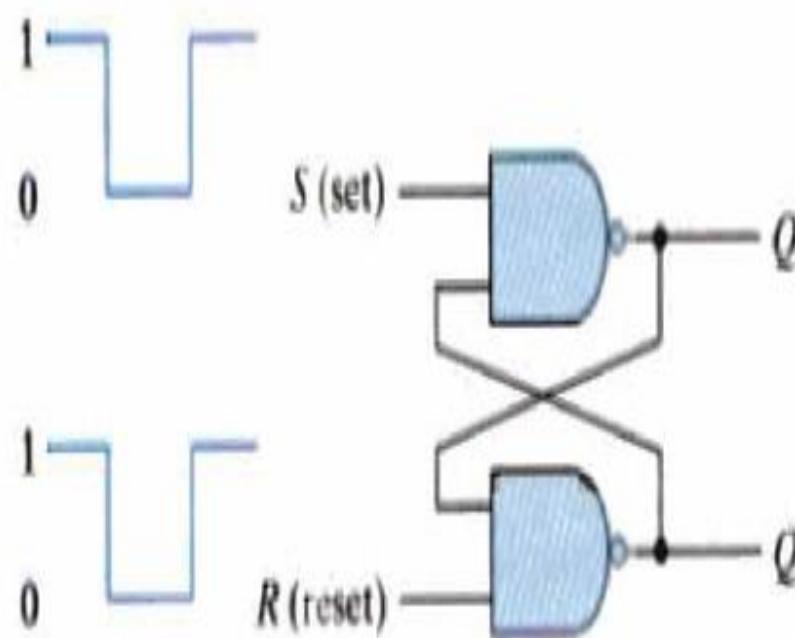
R	S	Q	Q'	Comments
0	0	Q	Q'	No change (hold) condition
0	1	1	0	Set
1	0	0	1	Reset
1	1	0	0	Forbidden, Not used , race

Cross- NAND S-R latch (active low) – S'-R'

- The truth table of NAND gate

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

Storage Elements : SR Latches with NAND Gate

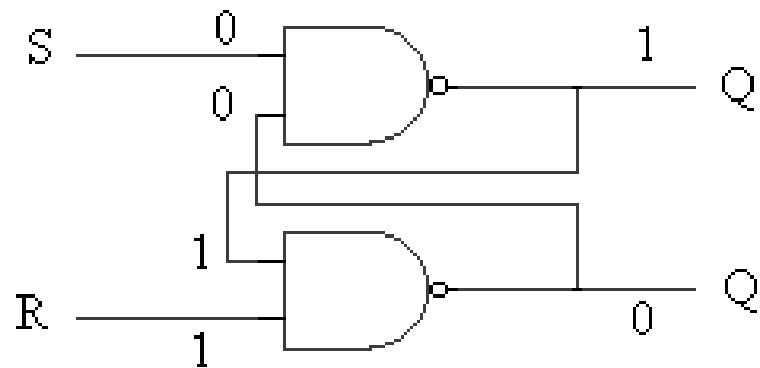


(a) Logic diagram

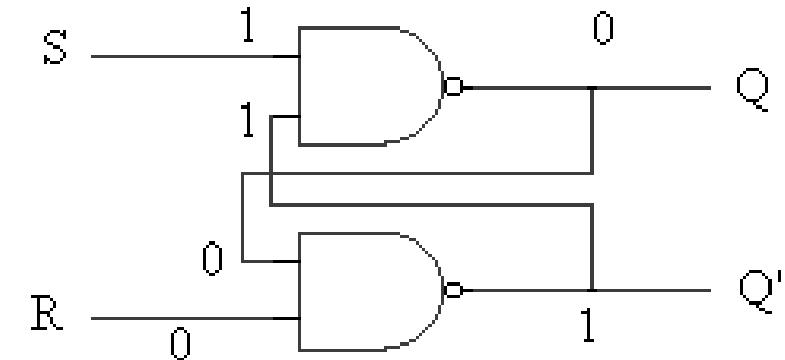
S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	(forbidden)

(b) Function table

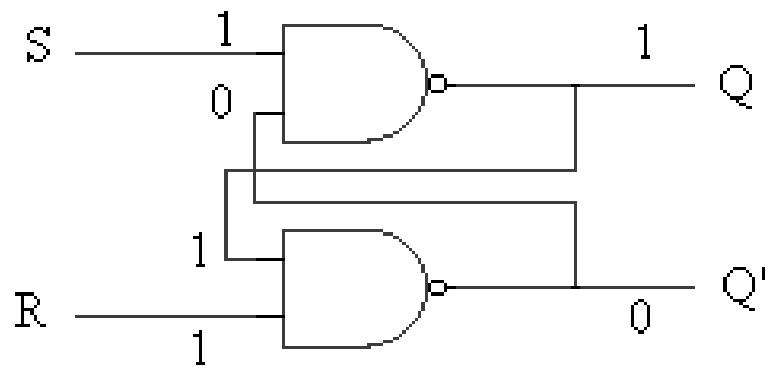
Cross- NAND S-R latch (active low) – S'-R'



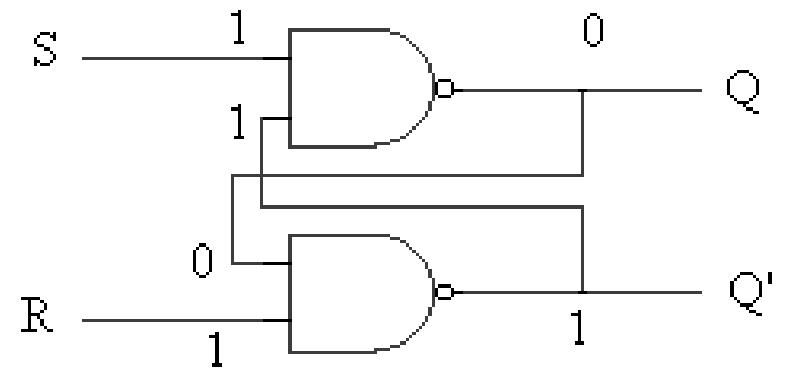
(a): SET condition.



(b): RESET condition.

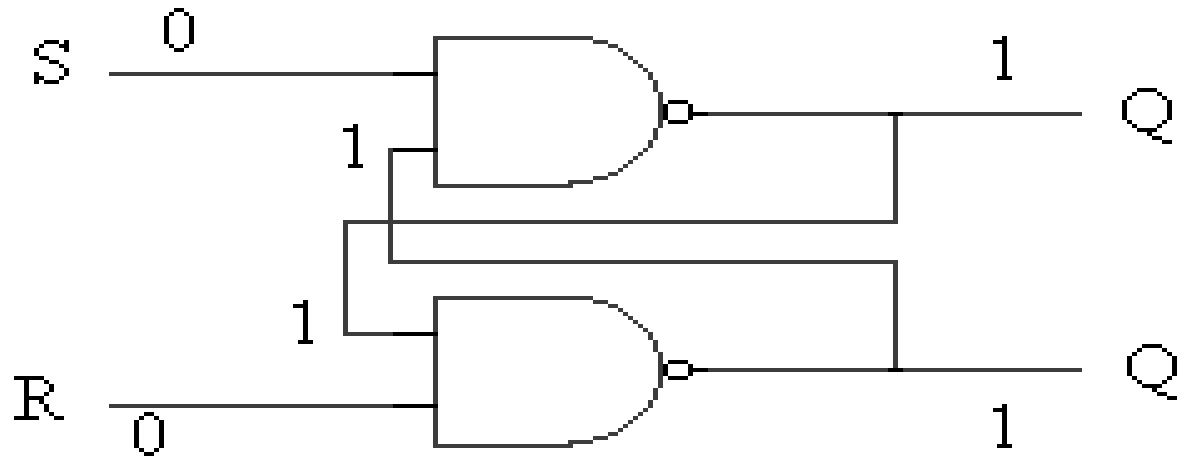


(c): no-change after SET.



(d): no-change after RESET.

Cross- NAND S-R latch (active low) – S'-R'



(e): forbidden or race condition.

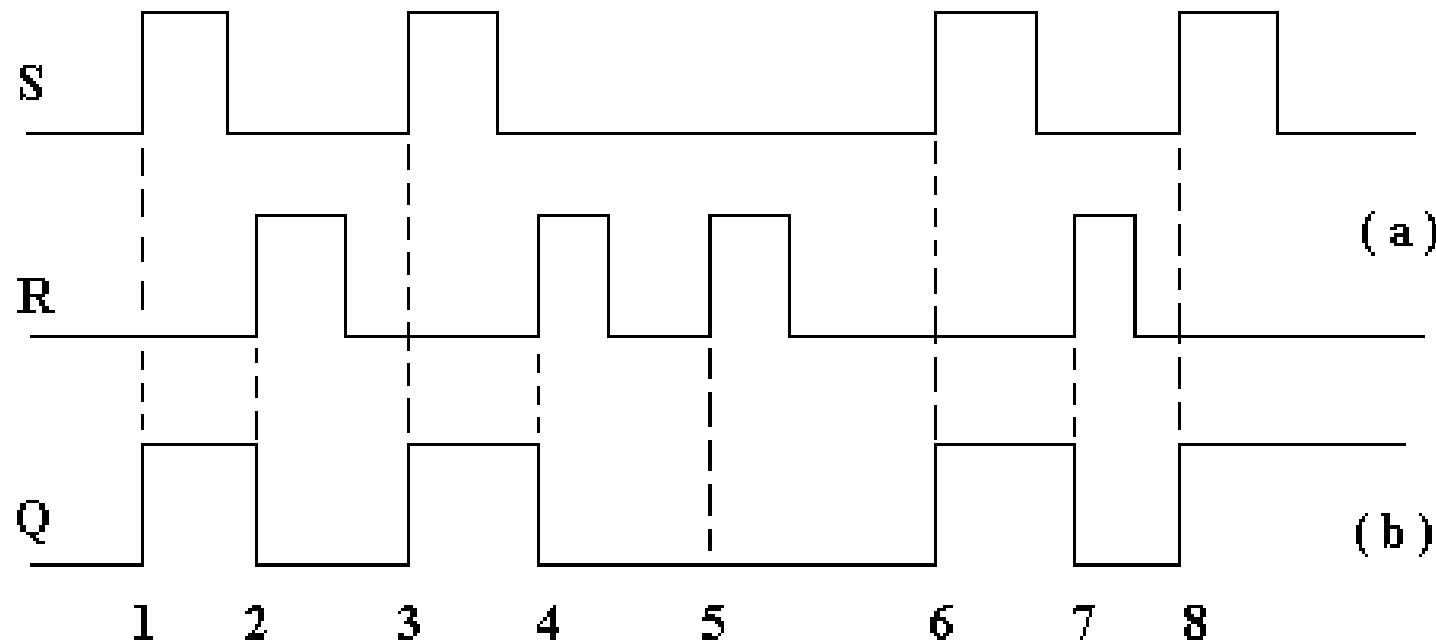
Fig(6): Cross-Coupled NAND Latch.

Functions table of the NAND latch .

R	S	Q	\bar{Q}	Comments
0	0	1	1	Forbidden, not used, race
0	1	0	1	Reset
1	0	1	0	Set
1	1	Q	\bar{Q}	Nochange (hold) condition

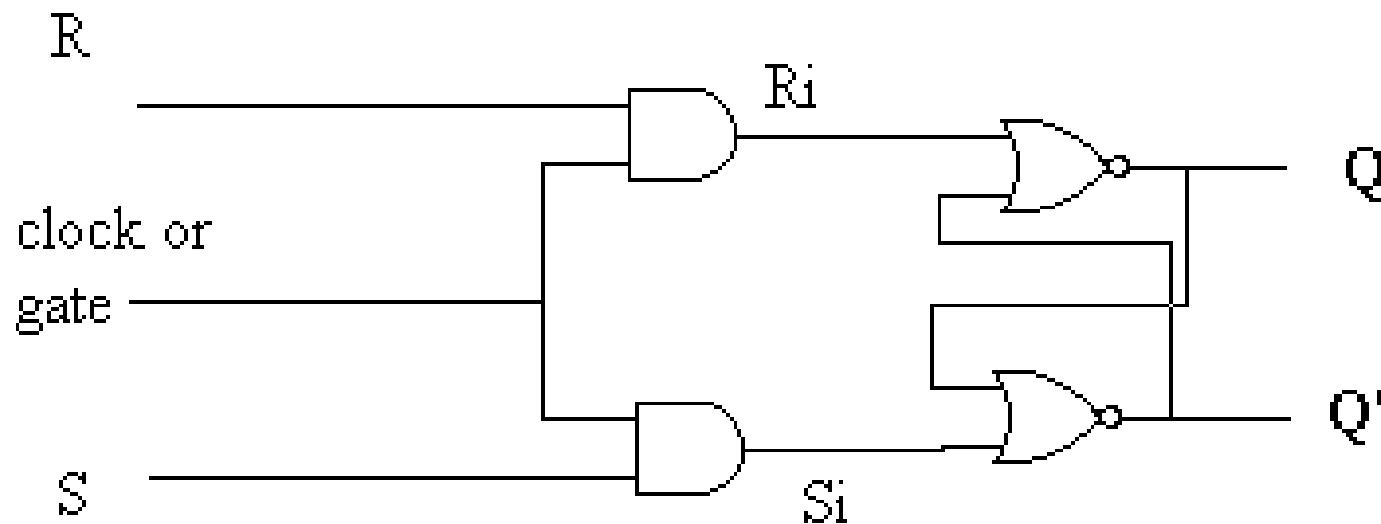
EXAMPLE

- If the S and R waveforms shown in Fig (11.a) are applied to the inputs of the NOR latch, determine the waveform that will be applied on the Q output. Assume that Q is initially low.



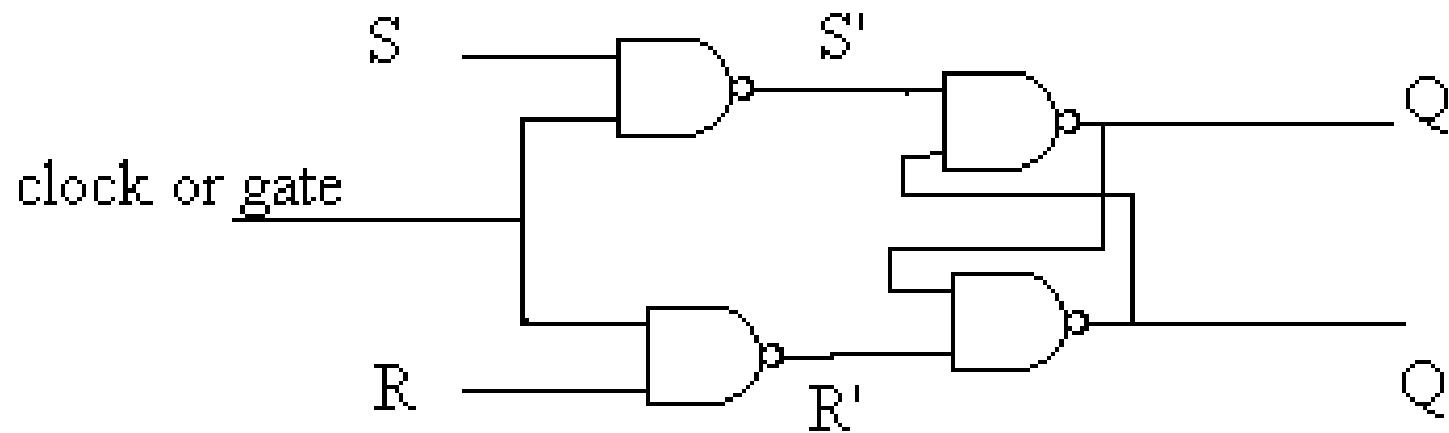
Fig(11)

Clocked SR latches (flip – flops)



Fig(15): Gated S-R flip-flop using cross coupled NOR gates.

clocked (gated) latch using cross – NAND gates



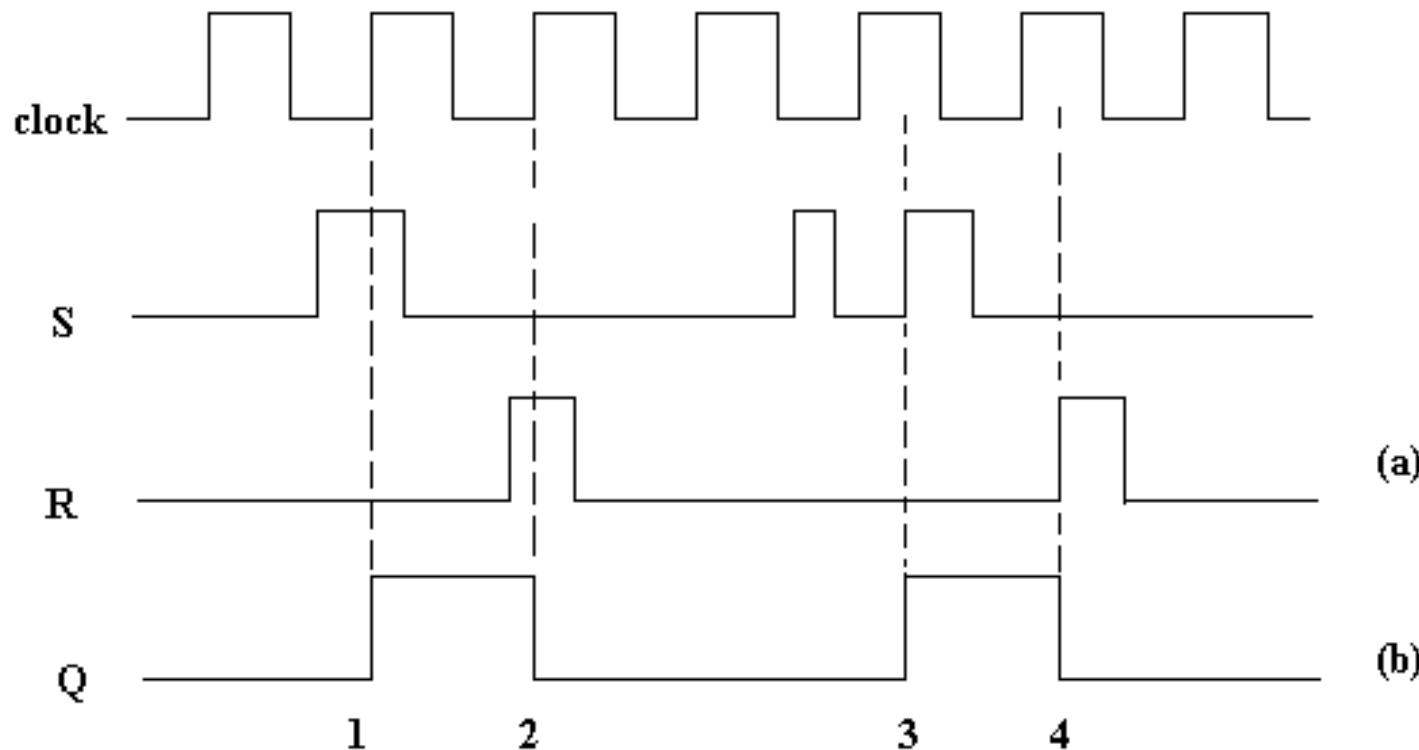
Fig(16): Gated S-R flip-flop using NAND gates.

Function table of gated S-R flip – flop

S	R	Gate	Q	\bar{Q}	Comments
X	X	0	Q	\bar{Q}	The gate is open and the flip flop is in the no change.
0	0	1	Q	\bar{Q}	No change
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	*	*	forbidden

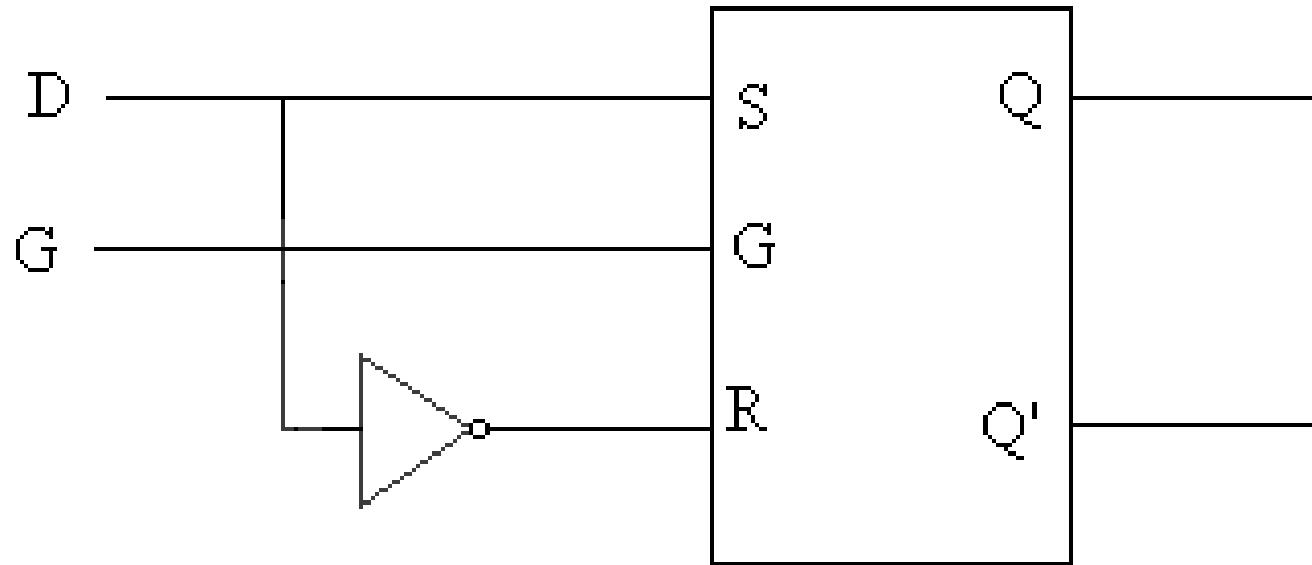
EXAMPLE

- Determine the Q output waveform if the inputs shown in Fig (17-a) are applied to a clocked (gated) S-R latch that is initially RESET.



Fig(17)

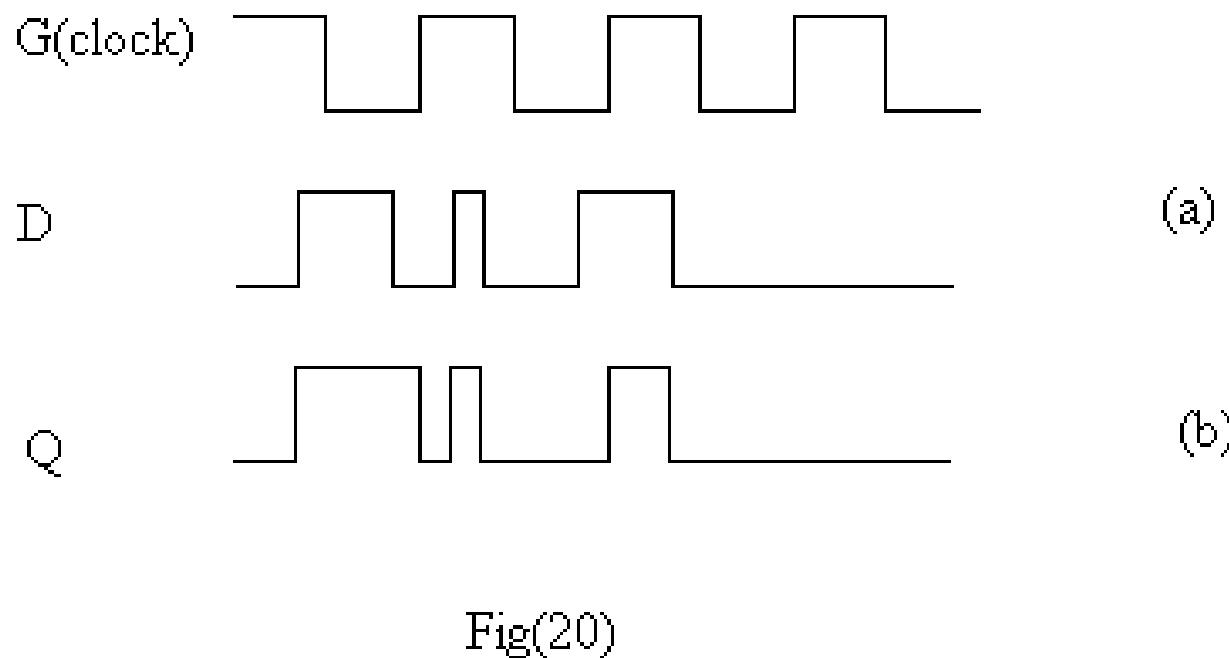
GATED D- latch



Fig(19): Gated D Latch.

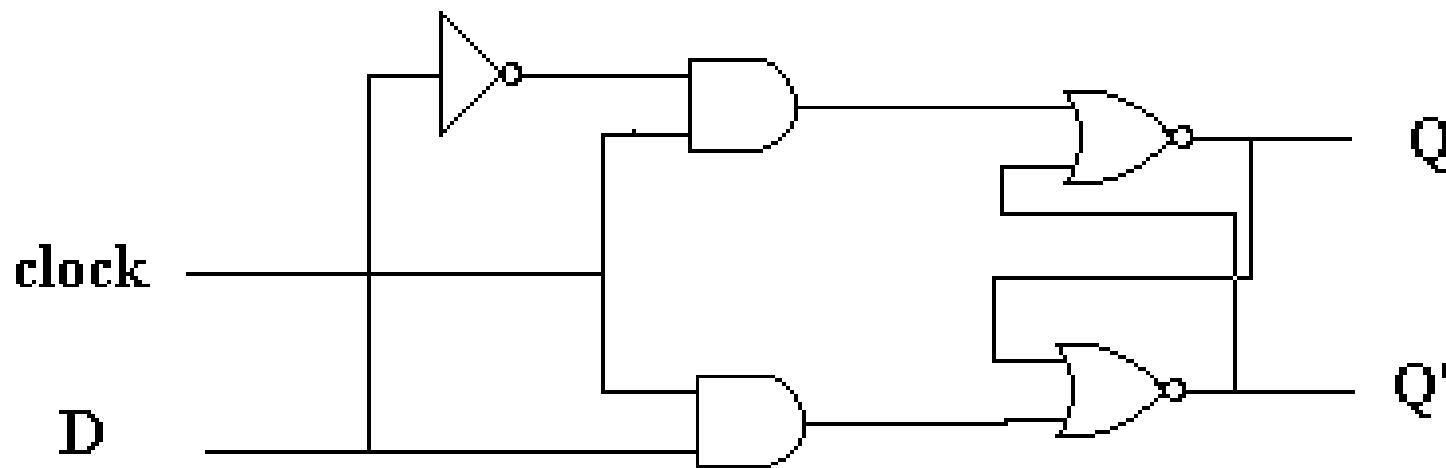
EXAMPLE

- Sketch the output waveform at Q for the inputs at D and G of the gated D latch in Fig (20).



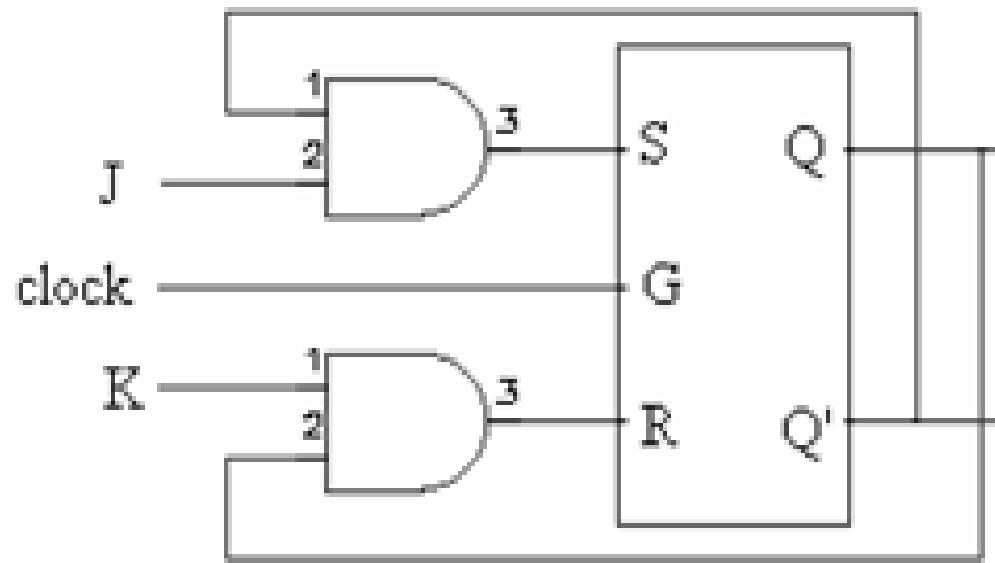
EXAMPLE

- Construct a D flip-flop using NOR and AND gates.

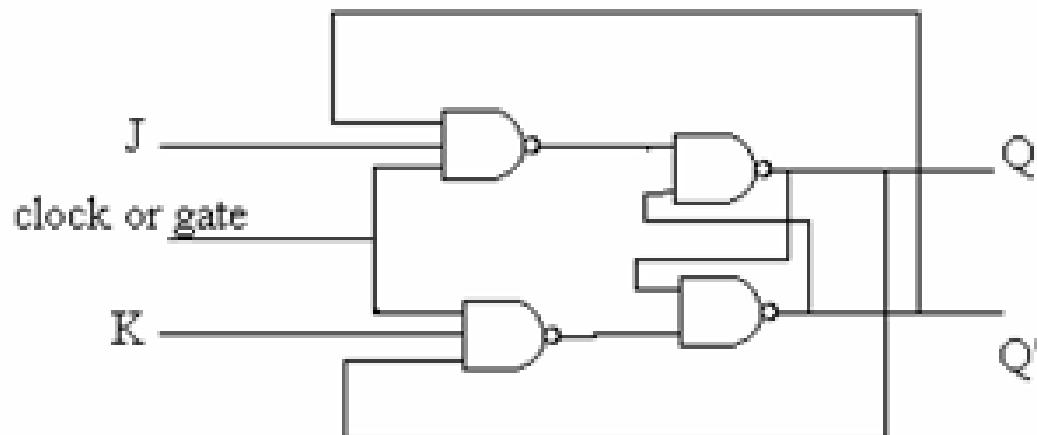
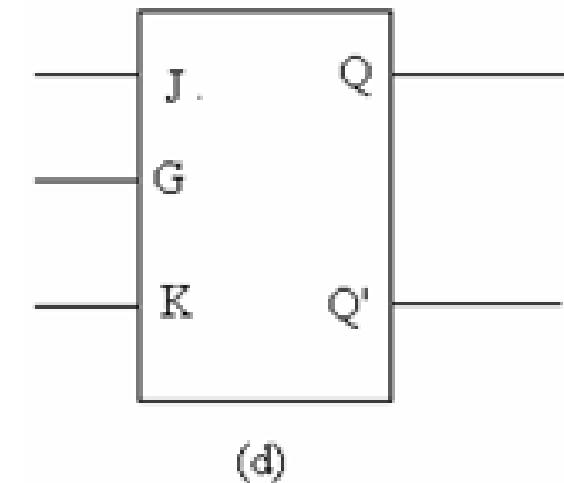
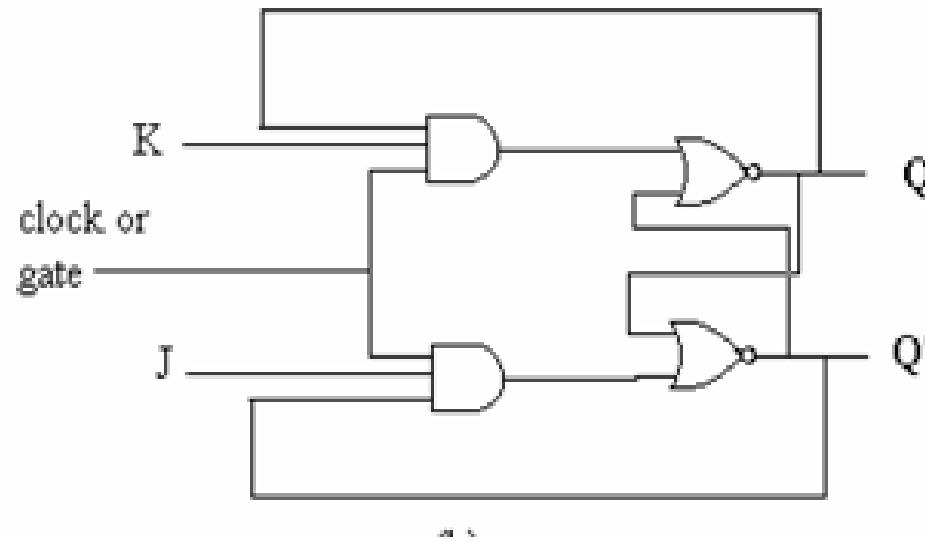


Fig(22)

J-K FLIP – FLOPS

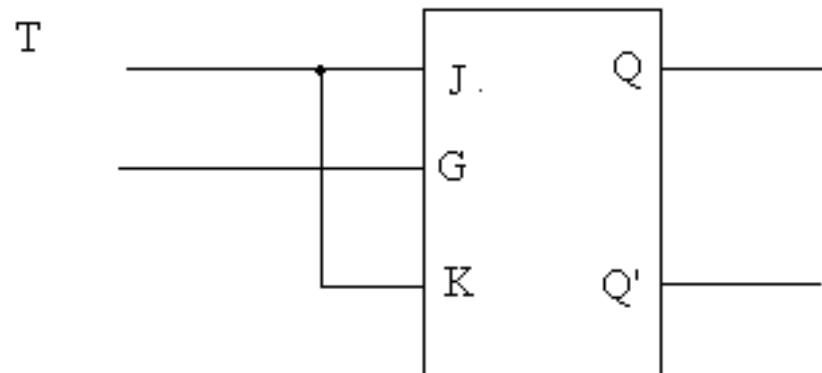


J-K FLIP – FLOPS



J	K	Gate	$Q(t+1)$	$\bar{Q}(t+1)$	Comments
X	X	0	$Q(t)$	$\bar{Q}(t)$	No-change (gate is open)
0	0	1	$Q(t)$	$\bar{Q}(t)$	No-change
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	$\bar{Q}(t)$	$Q(t)$	Toggle (complement)

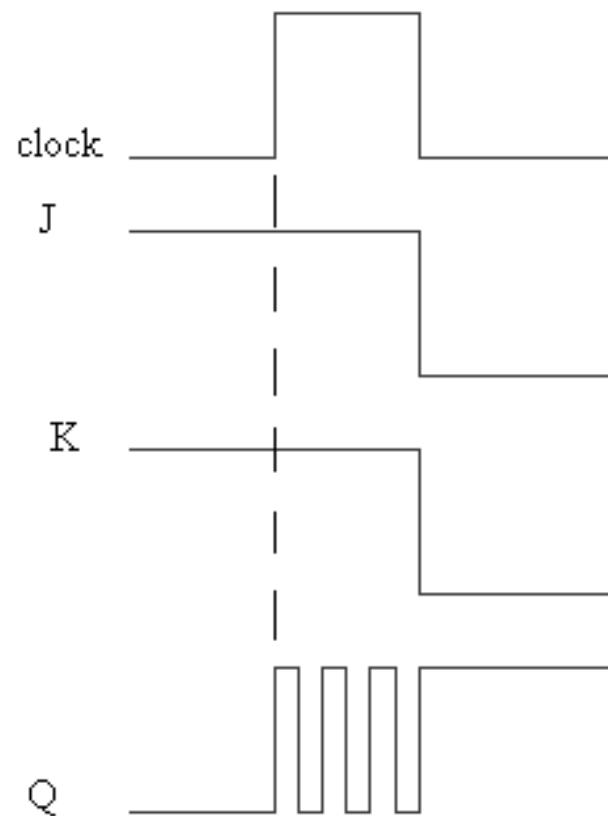
T. (TOGGLE) FLIP-FLOP



T	Gate	$Q(t+1)$	$\bar{Q}(t+1)$	Comments
X	0	$Q(t)$	$\bar{Q}(t)$	No-change (gate is open)
0	1	$Q(t)$	$\bar{Q}(t)$	No-change
1	1	$\bar{Q}(t)$	$Q(t)$	Toggle (complement)

p.

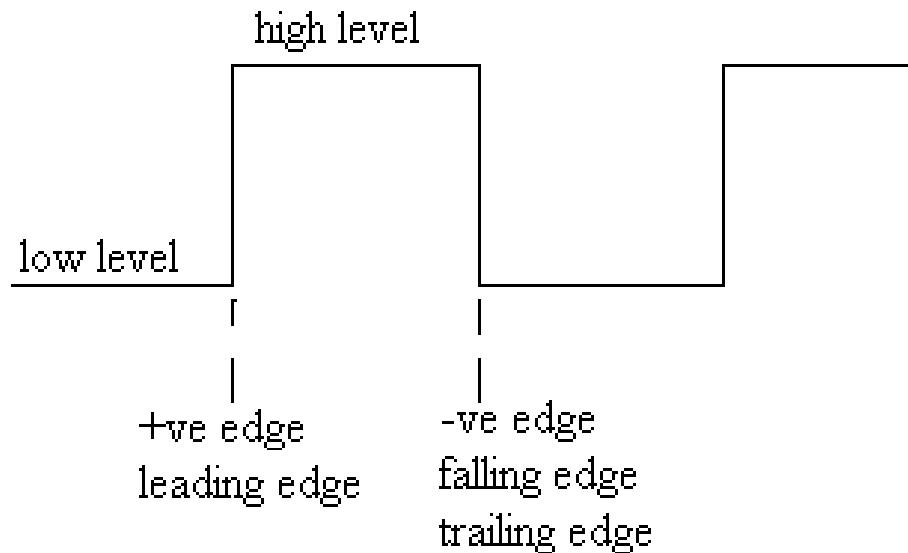
Race problem in level-coded J-K flip-flop



Fig(28):Race problem in level-coded J-K flip-flop.

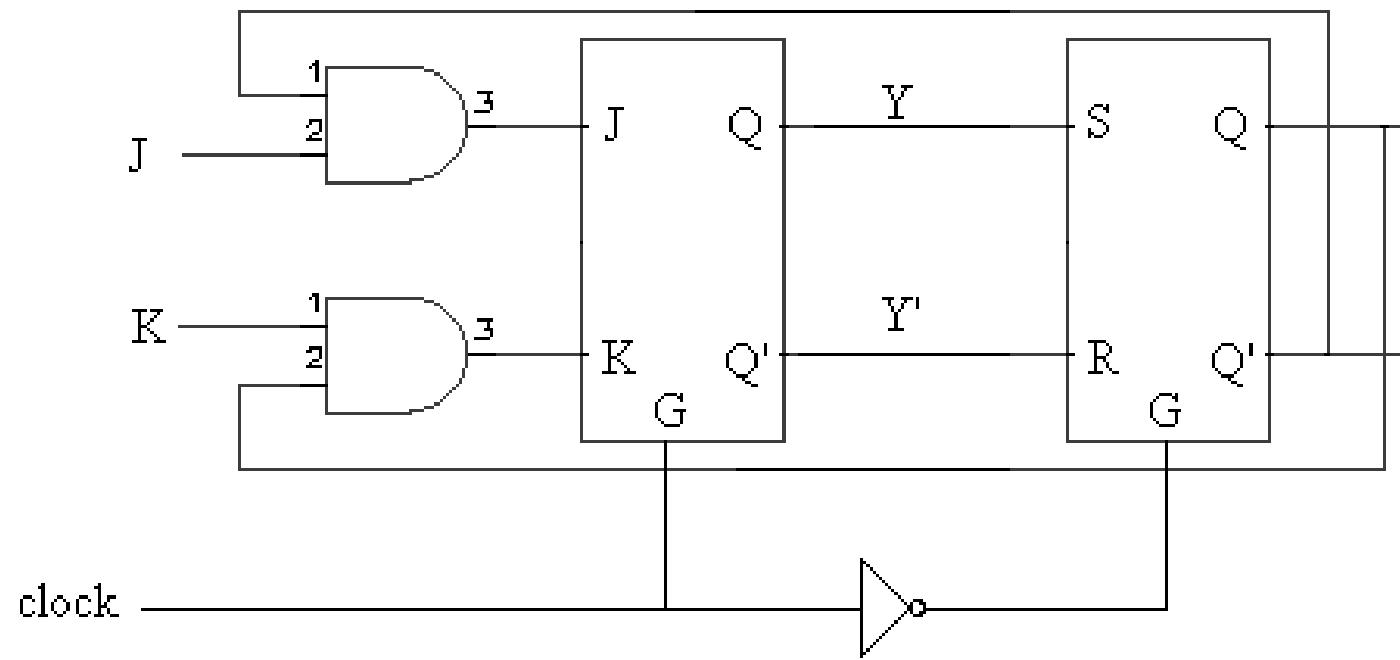
Clock edge and level

- edge – triggered FFs
- Master – slave FFs .



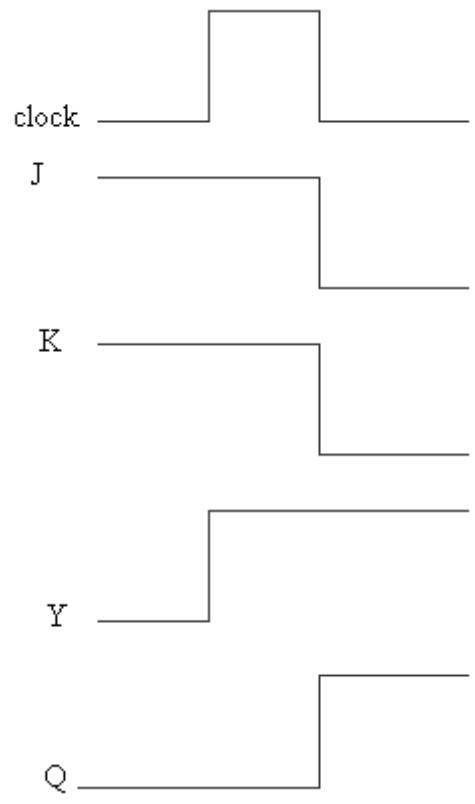
Fig(25): Clock edge and level.

MASTER – SLAVE FLIP-FLOPS



Fig(26): Master-Slave J-K flip-flop.

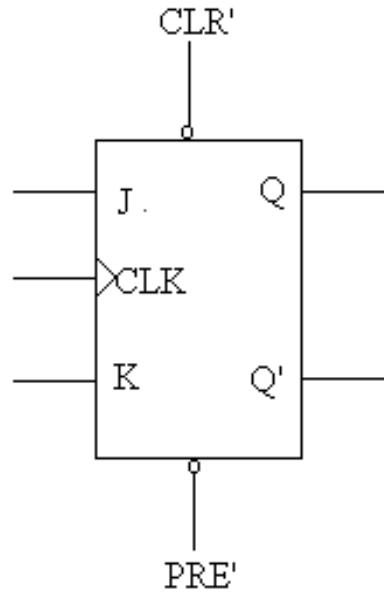
Timing diagram of a master-slave FF



the output changes
at the falling edge.

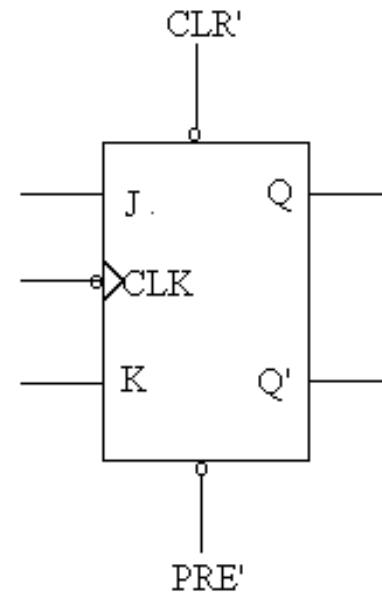
Fig(27): Timing diagram of master-slave flip-flop.

EDGE – TRIGGERED J K FFS



(a)

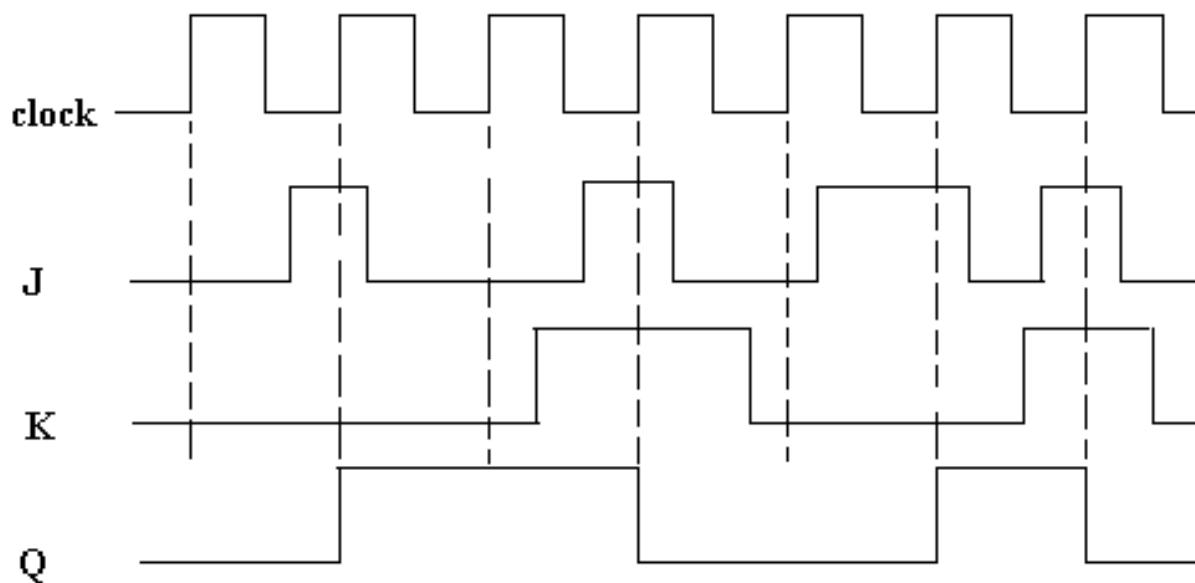
+ve edge triggered



(b)

-ve edge triggered

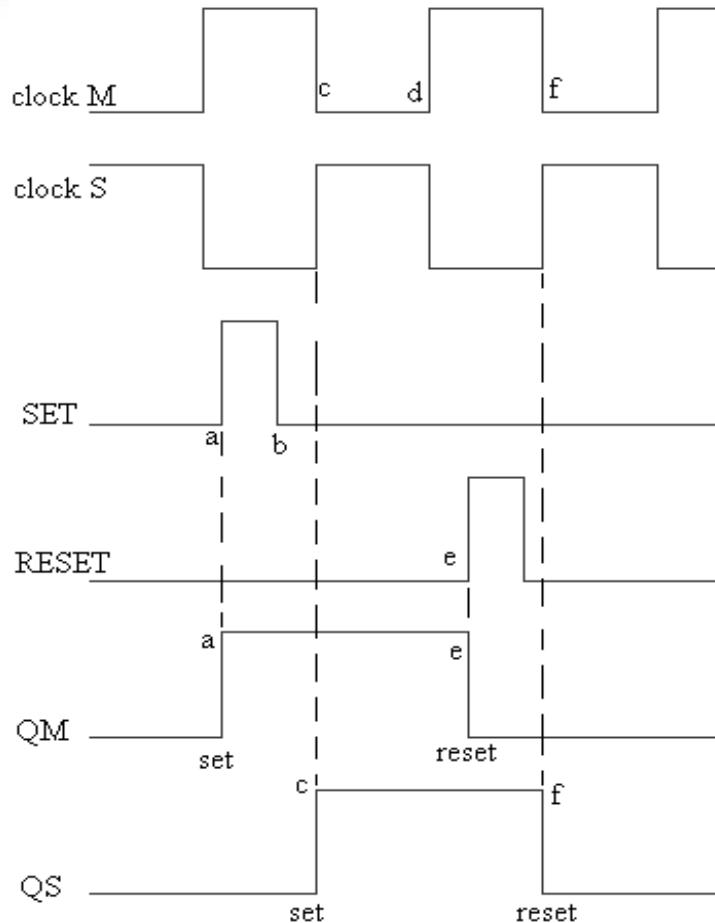
EXAMPLE



Fig(32)

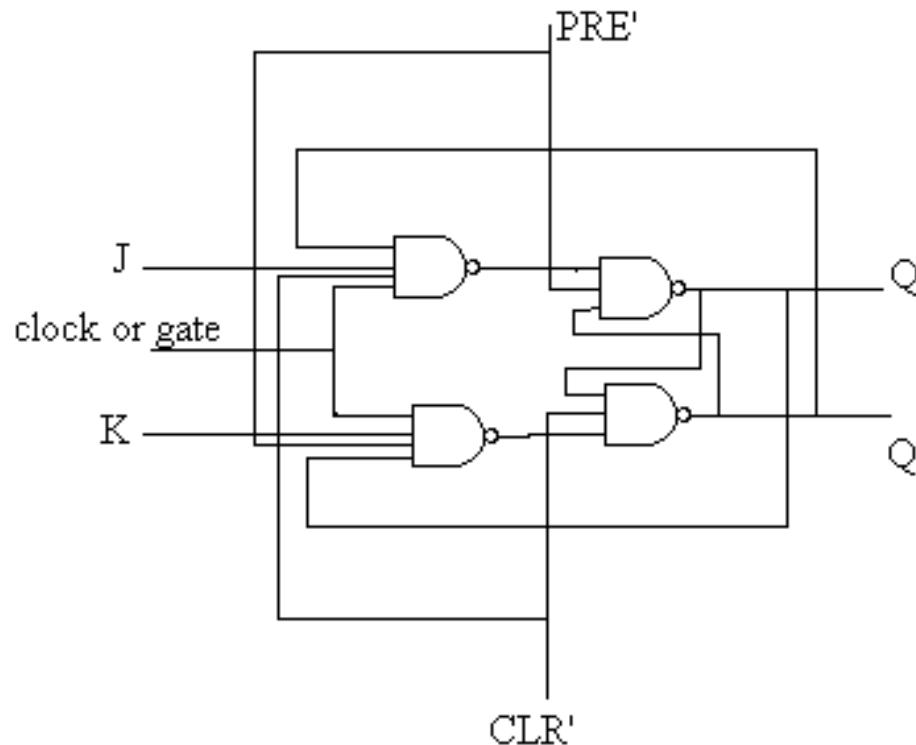
- Determine the Q output waveform if the inputs shown in Fig (32) are applied to a clocked S-R flip-flop that is initially RESET. The flip-flop is triggered at the positive edge.

MASTER-SLAVE FLIP-FLOP AND 1S CATCHING



Fig(36): 1s-catching in Master-Slave S-R flip-flop.

DIRECT (ASYNCHRONOUS) INPUTS



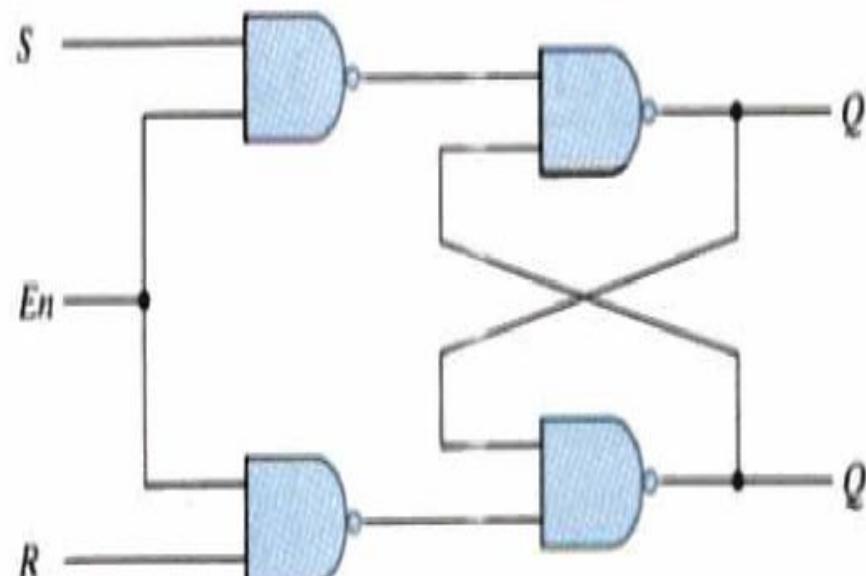
$\overline{\text{PRE}}$	$\overline{\text{CLR}}$	Q
0	0	HI, but unstable
0	1	1
1	0	0
1	1	Clocked operation

Fig(39): Logic diagram for a basic J-K flip-flop with active LOW preset and clear.

FLIP- FLOP OPERATING CHARACTERISTICS

- Propagation Delay times:
 - SET-UP TIME
 - HOLD TIME

Storage Elements : SR Latch with Control Input



(a) Logic diagram

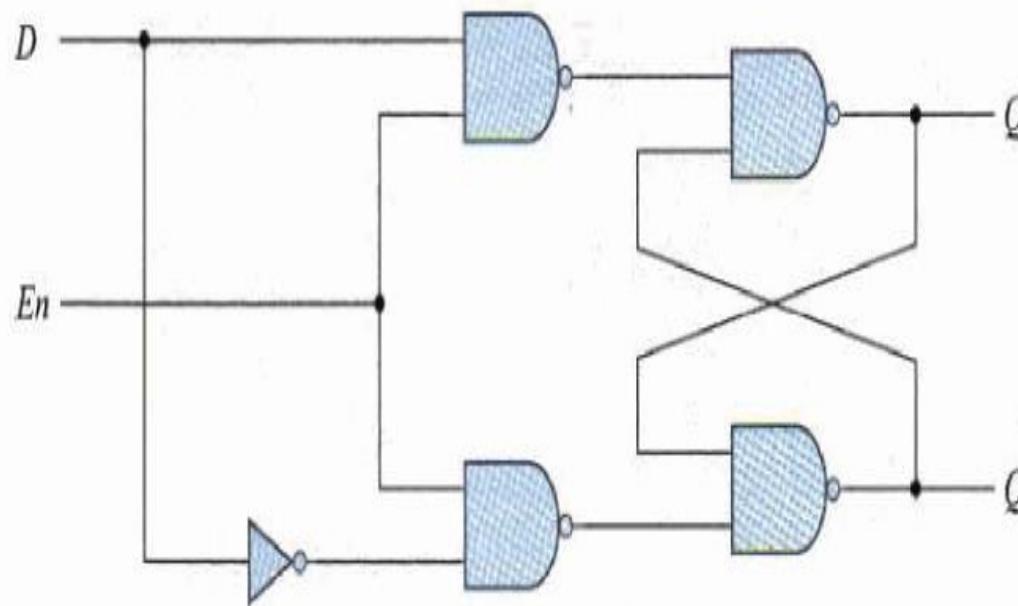
En	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

(b) Function table

Difficult to manage as it is , but useful as we will see

Storage Elements

D Latch(Transparent)

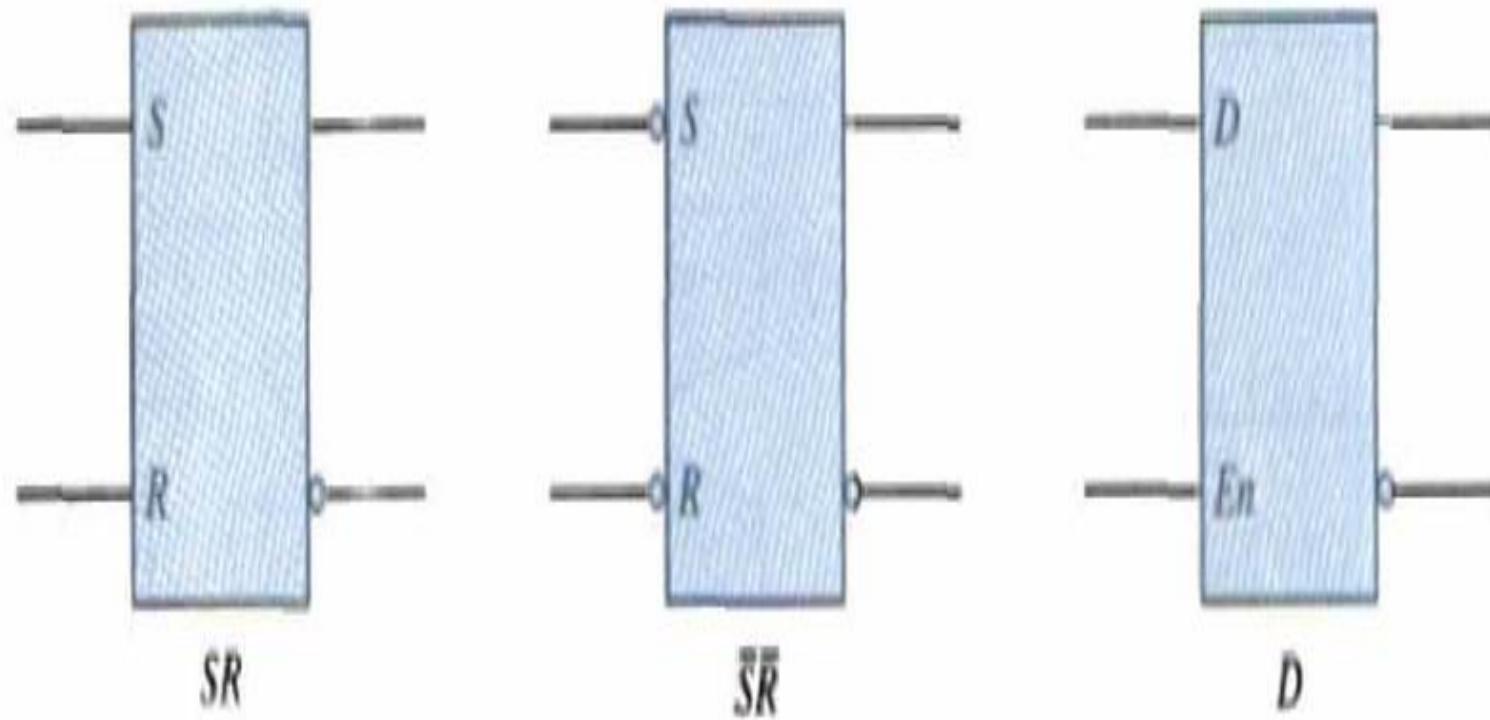


(a) Logic diagram

En	D	Next state of Q
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

(b) Function table

Storage Elements D(transparent) Graphic Symbols for Latches



Storage Elements

Flip-Flops(triggering)

Latch



(a) Response to positive level

F - F



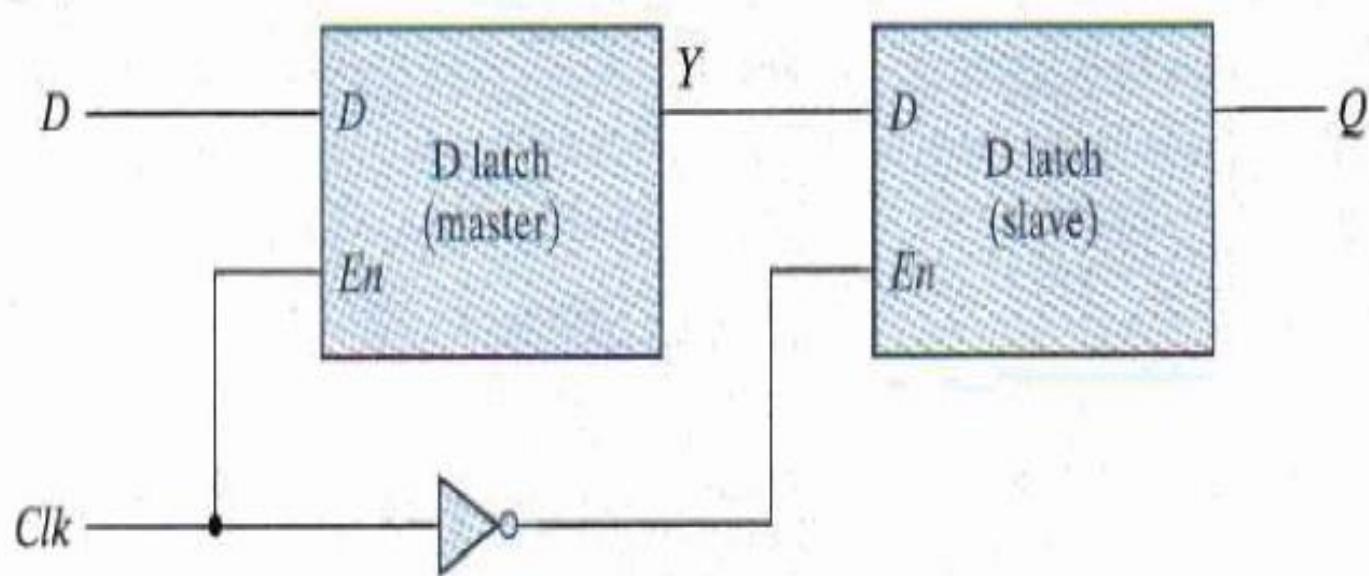
(b) Positive-edge response



(c) Negative-edge response

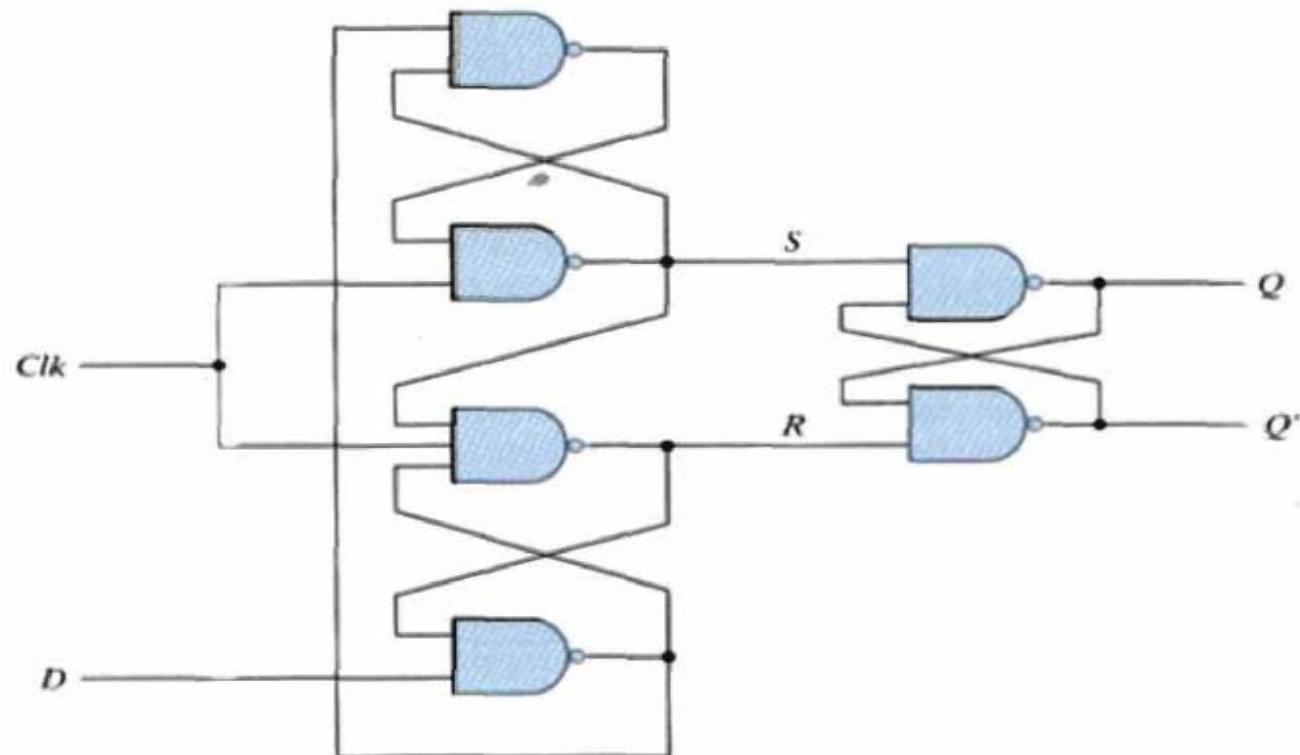
Storage Elements

Edge -Triggered D Flip-Flops

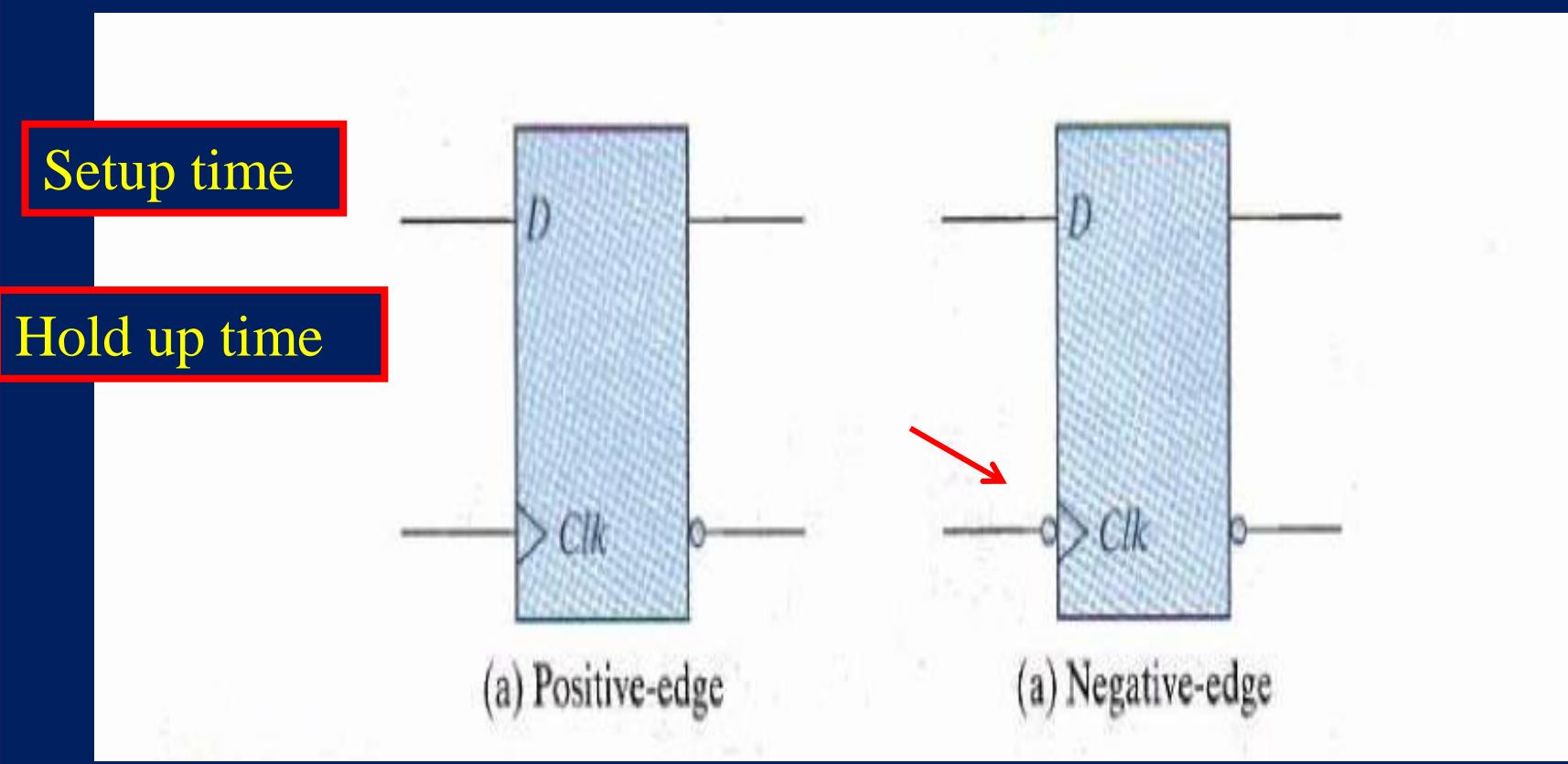


Master –Slave D Flip-Flop

D-Type Positive -Edge -Triggered Flip-Flop

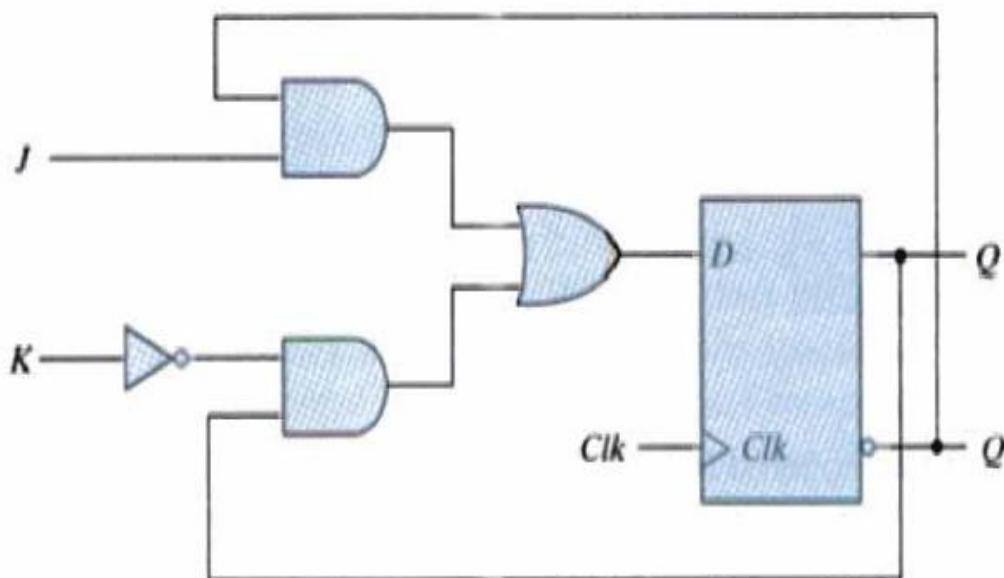


Graphic Symbol for Edge Triggered Flip-Flop

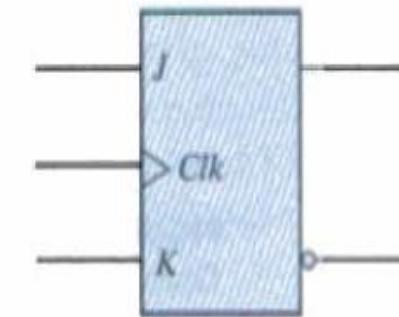


Other Types of Flip-Flops

JK Flip-Flop



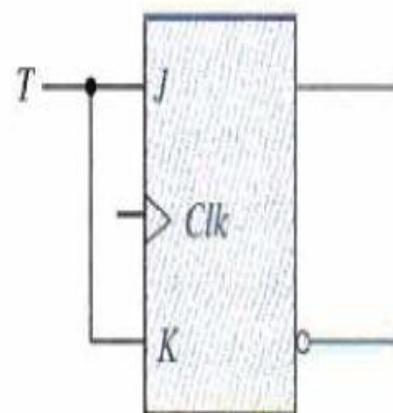
(a) Circuit diagram



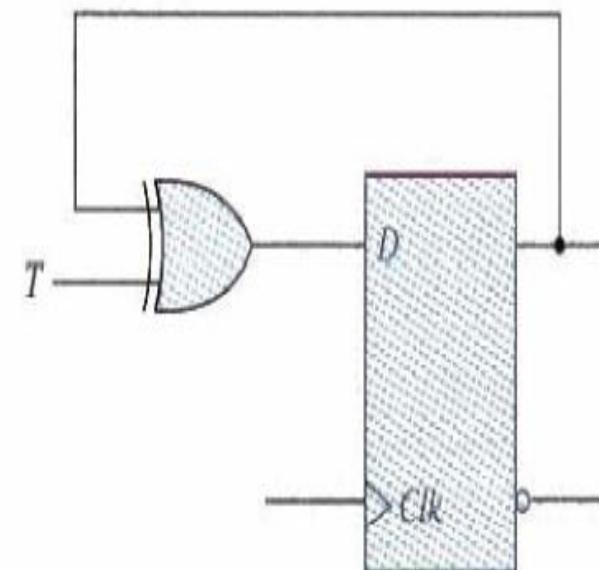
(b) Graphic symbol

$$D = J \bar{Q} + K \bar{Q}$$

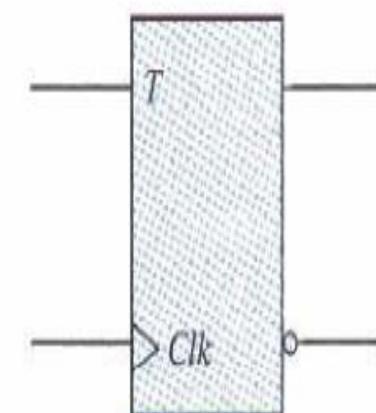
T Flip-Flop (toggle)



(a) From JK flip-flop



(b) From D flip-flop



(c) Graphic symbol

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

Characteristic Tables of Flip-Flops

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

D Flip-Flop

D	Q(t + 1)	
0	0	Reset
1	1	Set

T Flip-Flop

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

Characteristic equations

D FF

$$Q(t+1) = D$$

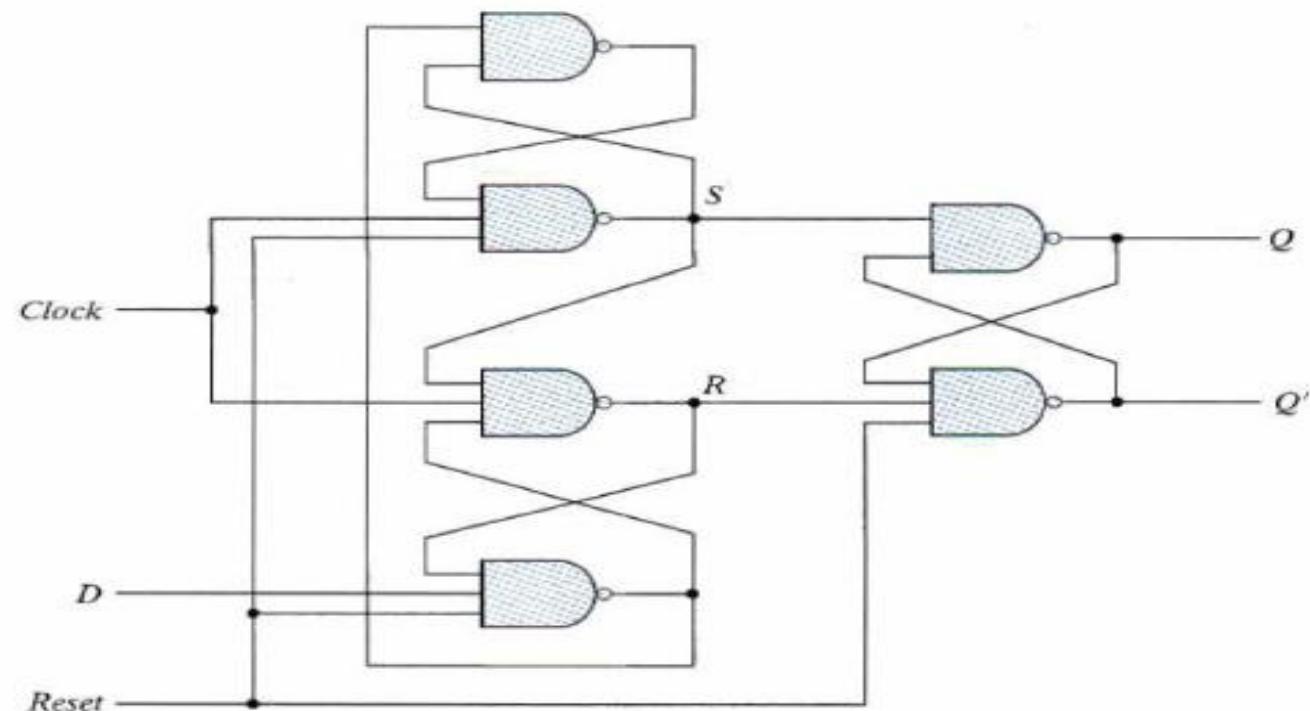
JK FF

$$Q(t+1) = JQ' + K' Q$$

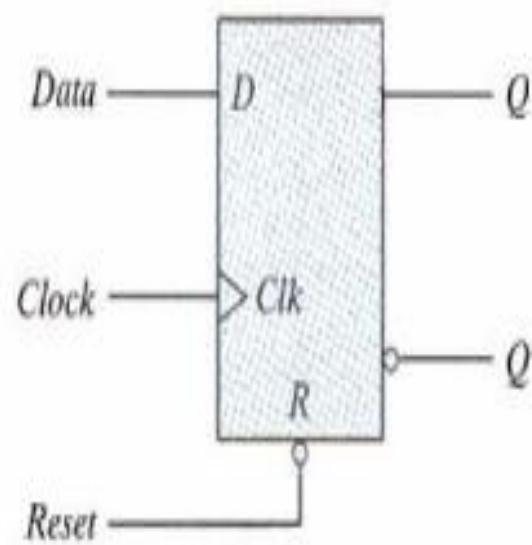
$$T FF Q(t+1) = T \oplus Q = T' Q + T Q'$$

Direct Inputs

Direct Set and Direct Reset



Direct Inputs



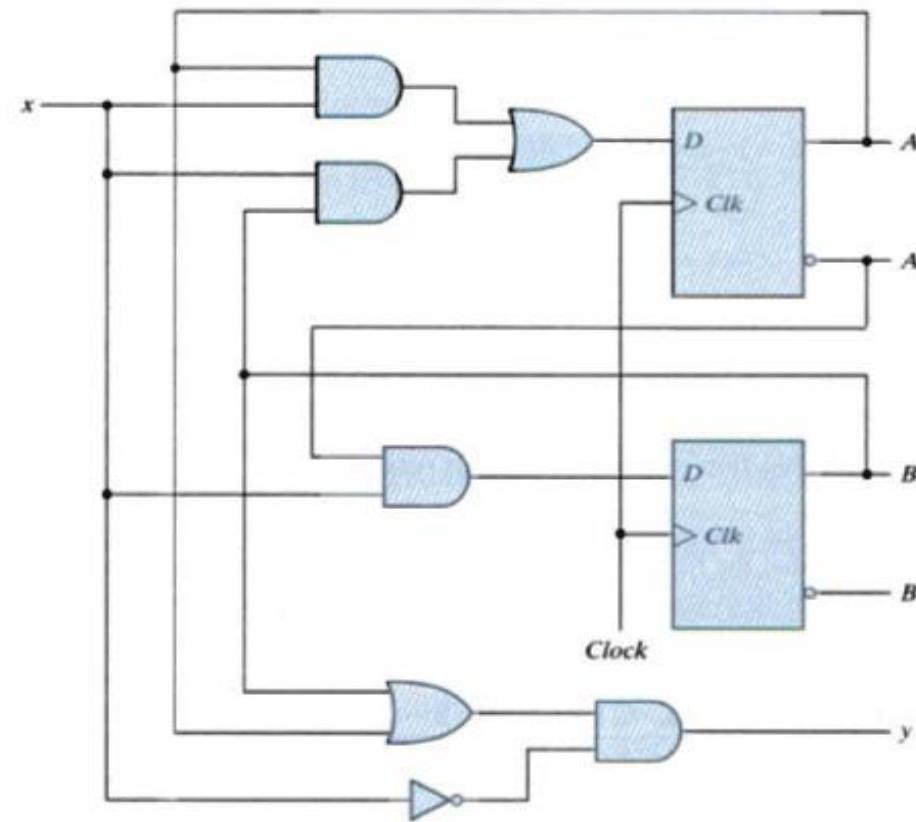
(b) Graphic symbol

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

(b) Function table

Analysis of clocked Sequential Circuits

Example of Sequential Circuit



Analysis of Clocked Sequential Circuits

State Equations

$$A(t+1) = A(t) X(t) + B(t)$$

$$B(t+1) = A^*(t) X(t)$$

$$B(t+1) = A x + B x$$

$$B(t+1) = A^* x$$

$$Y(t) = [A(t)+B(t)] x^*(t)$$

$$Y = (A+B) x^*$$

Analysis of Clocked Sequential Circuits

State Table

Present State		Input <i>x</i>	Next State		Output <i>y</i>
<i>A</i>	<i>B</i>		<i>A</i>	<i>B</i>	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Analysis of Clocked Sequential Circuits

$$A(t+1) = Ax + Bx$$

$$B(t+1) = A^{\sim}x$$

$$y = Ax^{\sim} + Bx^{\sim}$$

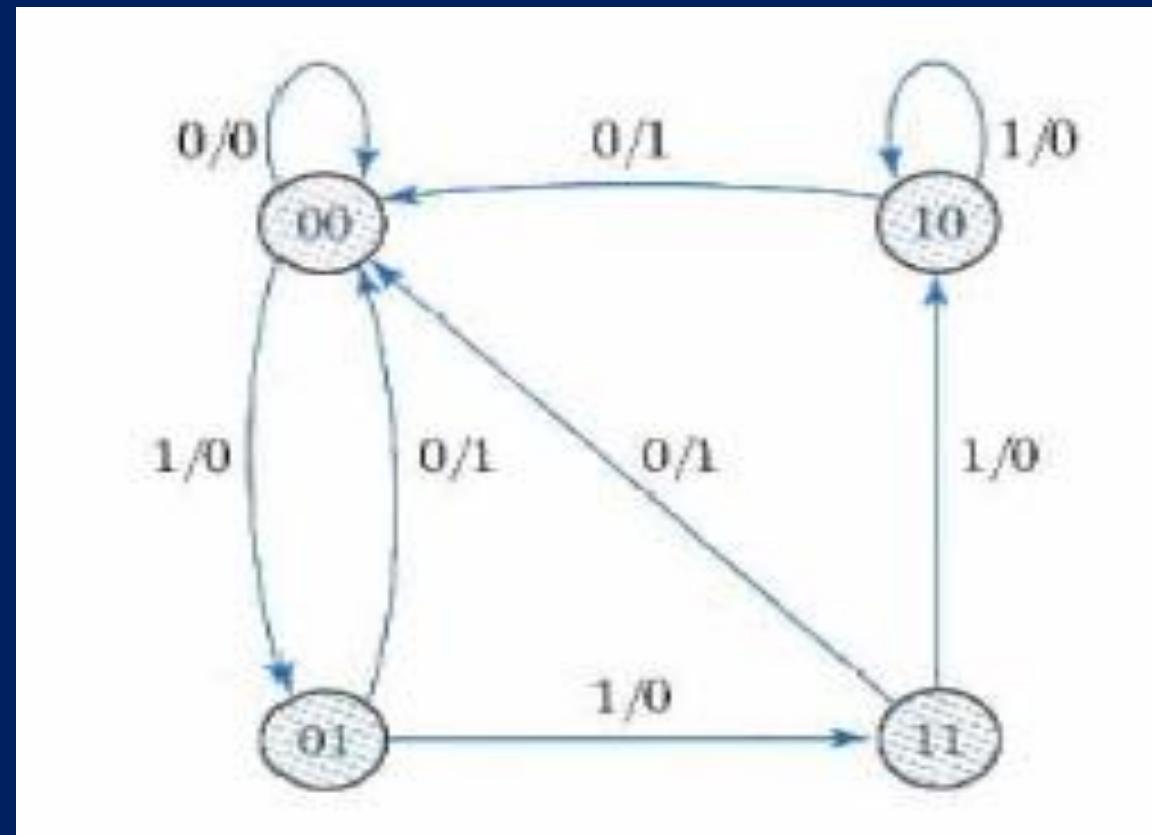
Analysis of Clocked Sequential Circuits

Second Form of State Table

Present State	Next State				Output	
	$x = 0$		$x = 1$			
A	B	A	B	A	B	
0	0	0	0	0	1	0
0	1	0	0	1	1	0
1	0	0	0	1	0	1
1	1	0	0	1	0	1

Analysis of Clocked Sequential Circuits

State Diagram



Flip-Flop Input Equations

$$D_Q = X + Y$$

$$D_A = A X + B \bar{X}$$

$$D_B = A' X$$

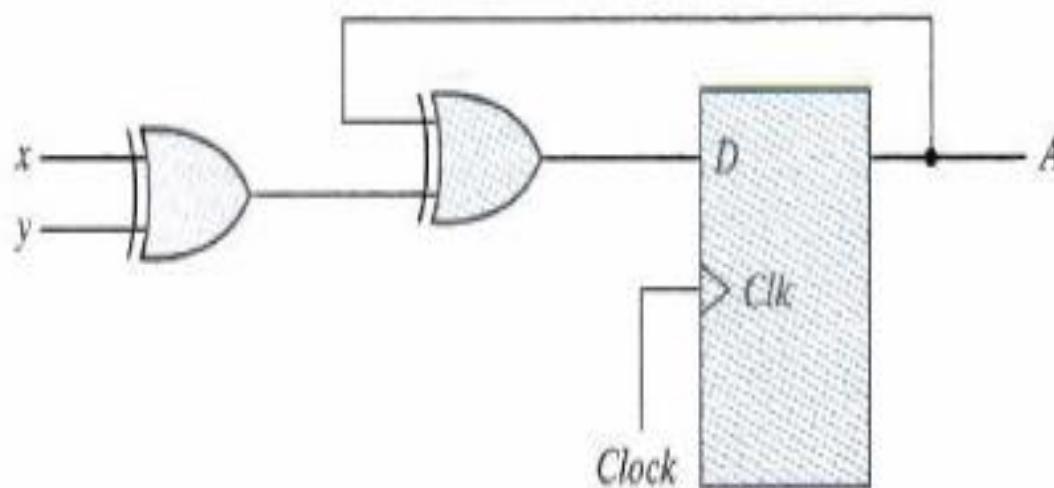
$$Y = (A + B) X'$$

Analysis of D Flip-Flops

$$D_A = A \oplus X \oplus y$$

$$A(t+1) = A \oplus X \oplus y$$

Sequential circuit with D Flip-Flop

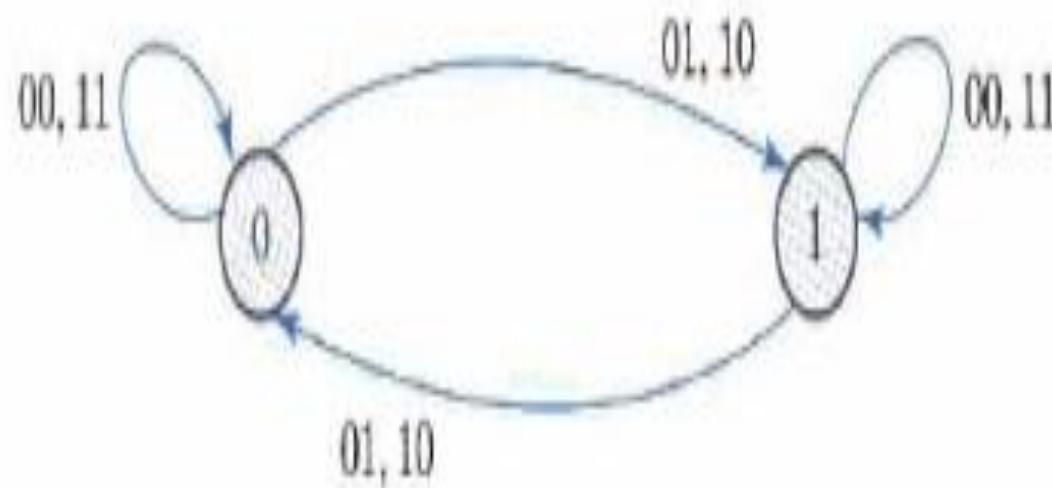


(a) Circuit diagram

Present state	Inputs		Next state
A	x	y	A
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

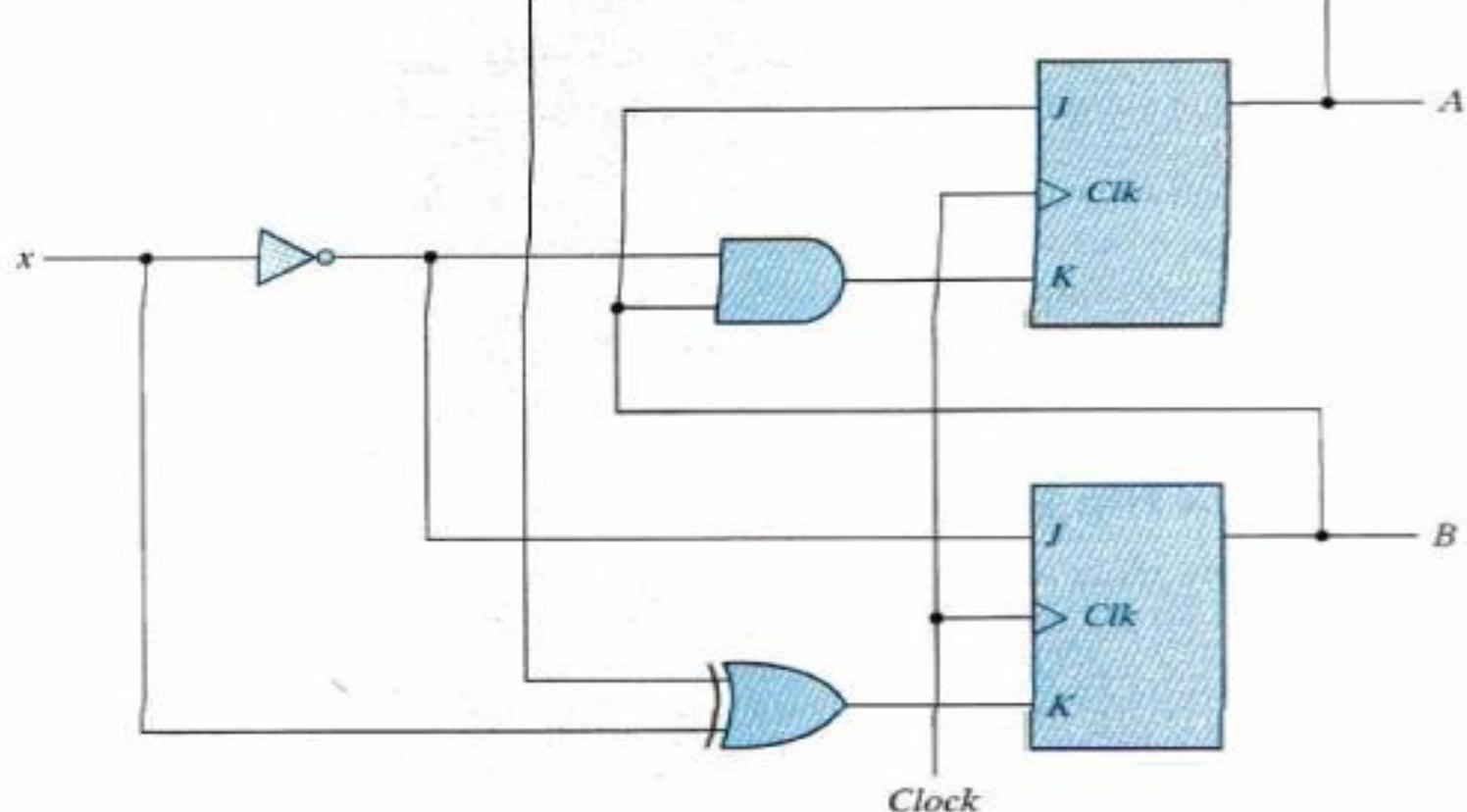
(b) State table

Sequential circuit with D Flip-Flop



(c) State diagram

Analysis with JK Flip-Flop



Analysis with JK Flip-Flop

$$J_A = B$$

$$K_A = B \cdot x'$$

$$J_B = x'$$

$$K_B = A' \cdot x + A \cdot x' = A \oplus x$$

Analysis with JK Flip-Flop

State Table for Sequential Circuit with JK Flip-Flops

Present State		Input x	Next State		Flip-Flop Inputs			
A	B		A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

Analysis with JK Flip-Flop

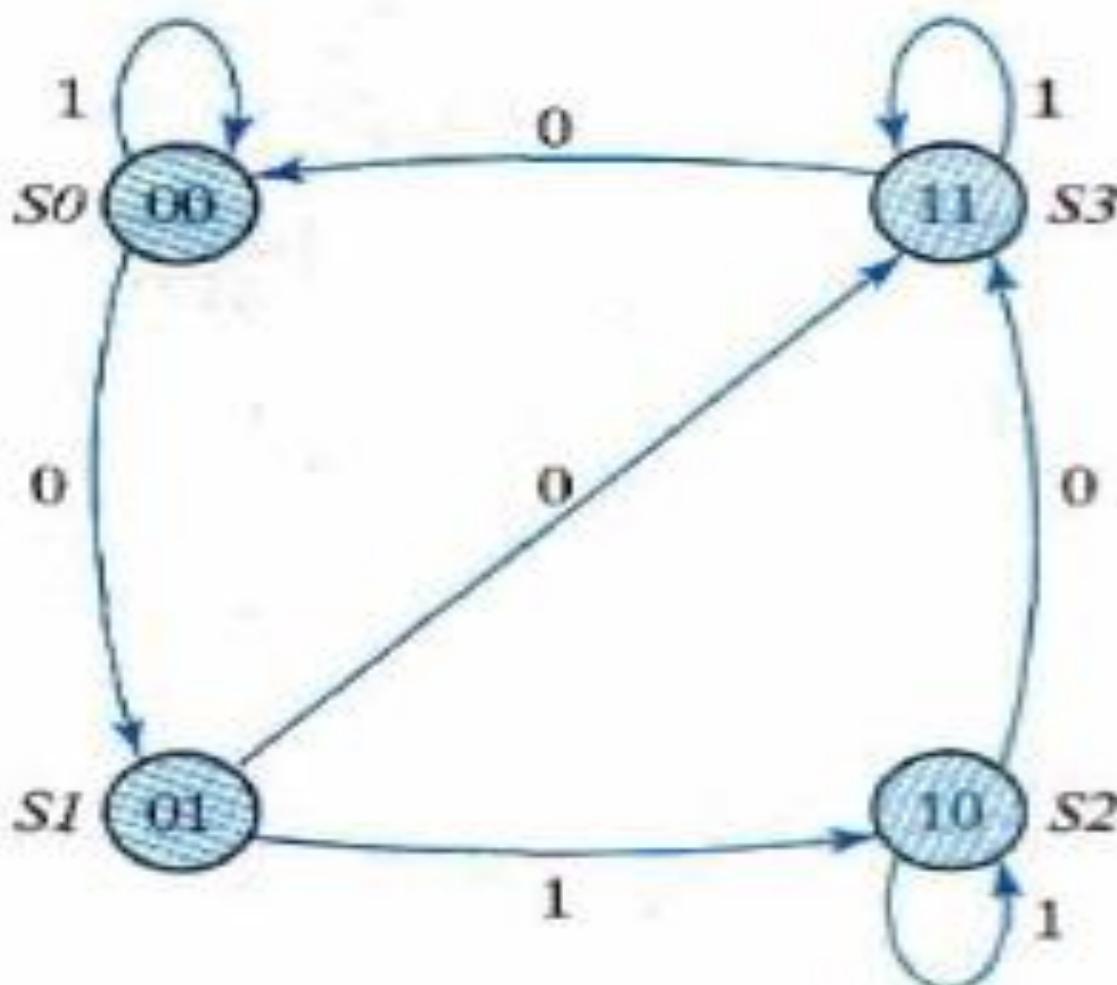
$$A(t+1) = J A' + K A$$

$$B(t+1) = J B' + K B$$

$$A(t+1) = B A' + (B x')' = A' B + A B' + A x$$

$$B(t+1) = x' B' + (A \oplus x)' B = B' x' + A B x + A' B x'$$

Analysis with JK Flip-Flop



Analysis with T Flip-Flops

$$Q(t+1) = T \oplus Q = T' Q + T Q'$$

$$T_A = B x$$

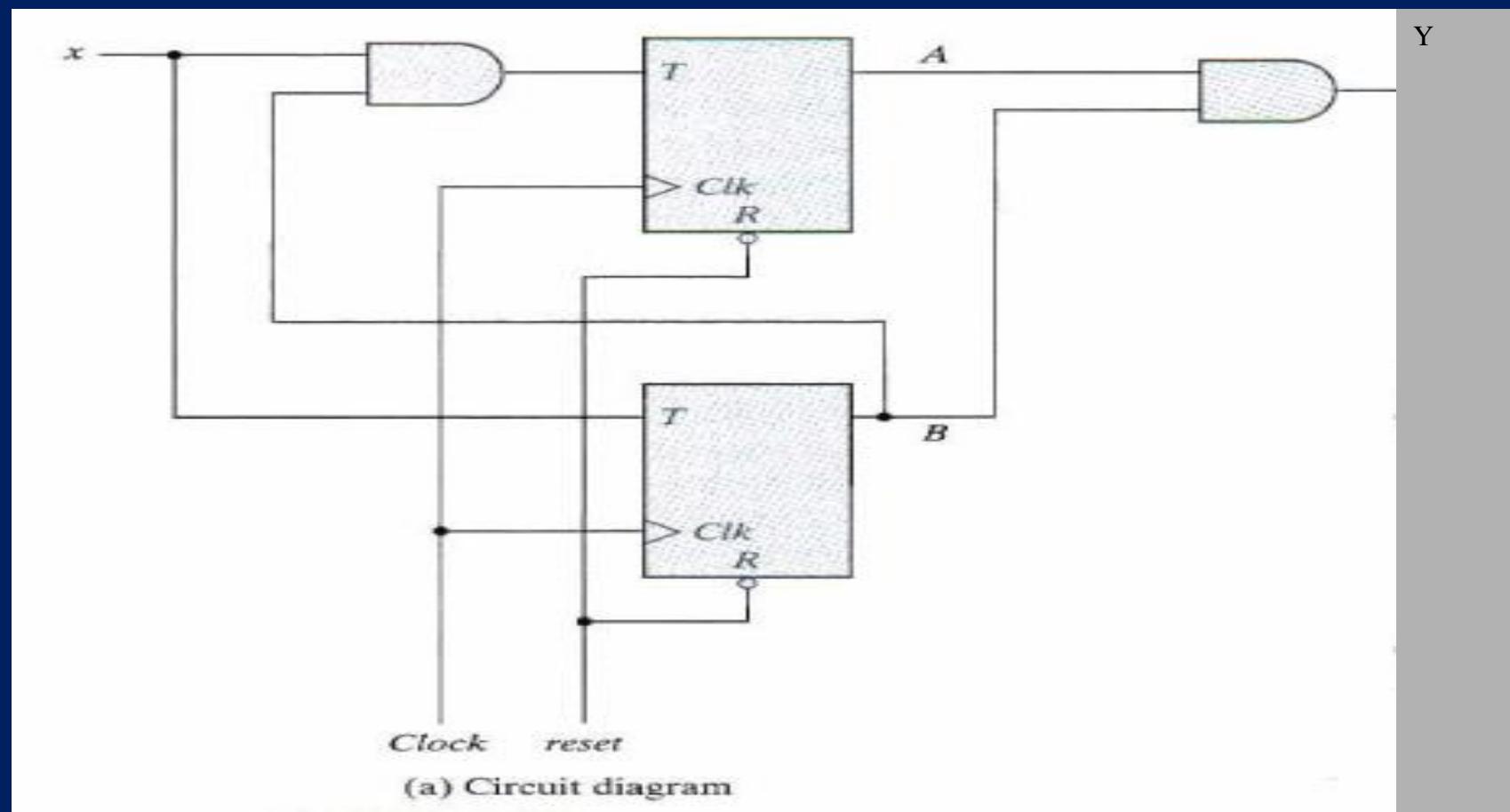
$$T_B = x$$

$$y = A B$$

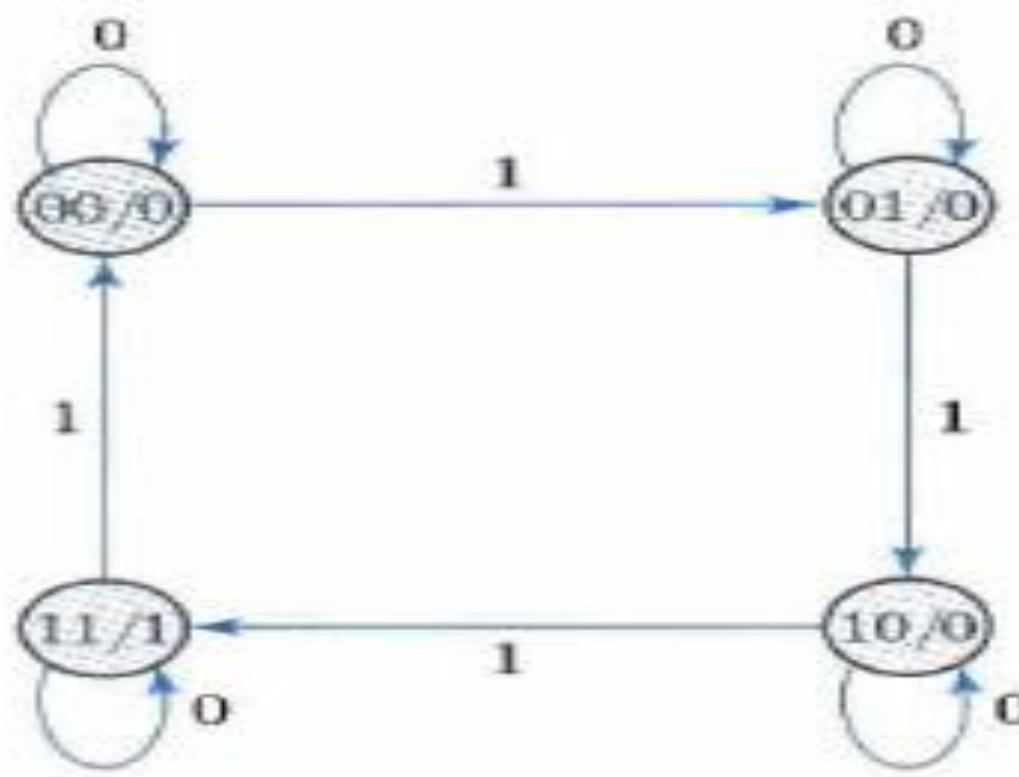
$$A(t+1) = (B x)' A + (B x) A' = A B' + A x' + A' B x$$

$$B(t+1) = x \oplus B$$

Analysis with T Flip-Flops



Analysis with T Flip-Flops



(b) State diagram

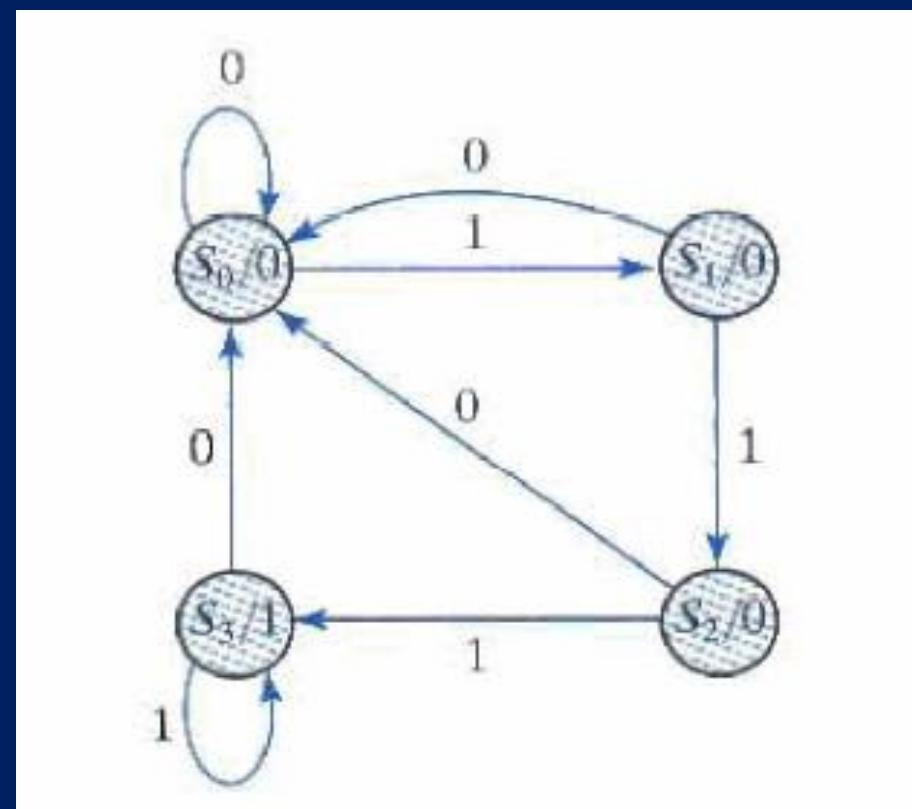
Analysis with T Flip-Flop

State Table for Sequential Circuit with T Flip-Flops

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

Design Procedures

State Diagram for sequence Detector



Design Procedures

D Flip Flops

Input Equations for sequence Detector

$$A(t+1) = D_A(A, B, x) = \sum(3, 5, 7)$$

$$B(t+1) = D_B(A, B, x) = \sum(1, 5, 7)$$

$$y(A, B, x) = \sum(6, 7)$$

Design Procedures

State Table for Sequence Detector

State Table for Sequence Detector

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Design Procedures

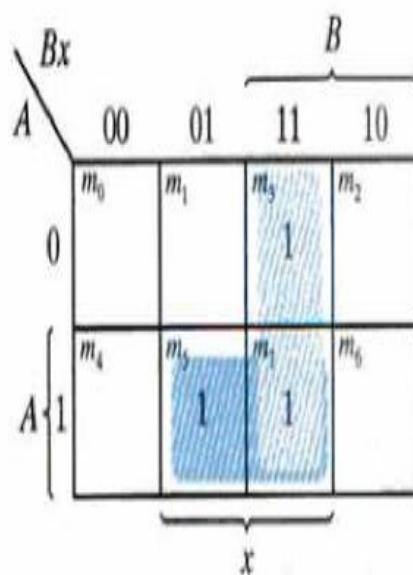
Simplified Equations sequence Detector

$$D_A = A X + B X$$

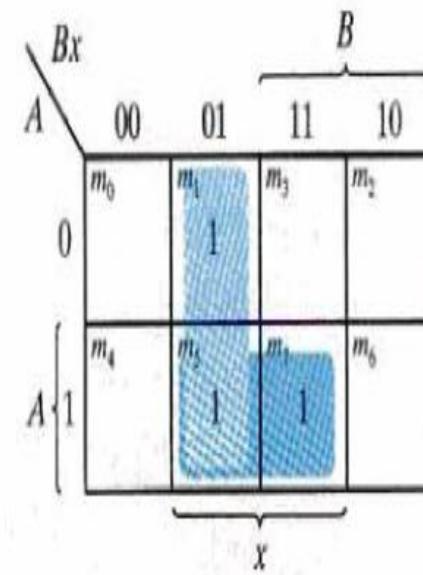
$$D_B = A X + B' X$$

$$y = A B$$

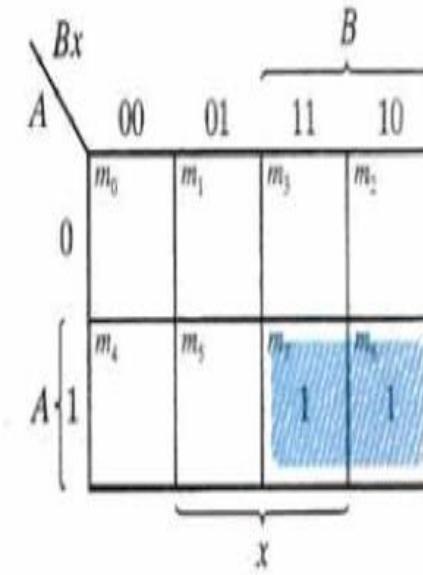
Design Procedures Maps for sequence Detector



$$D_A = Ax + Bx$$



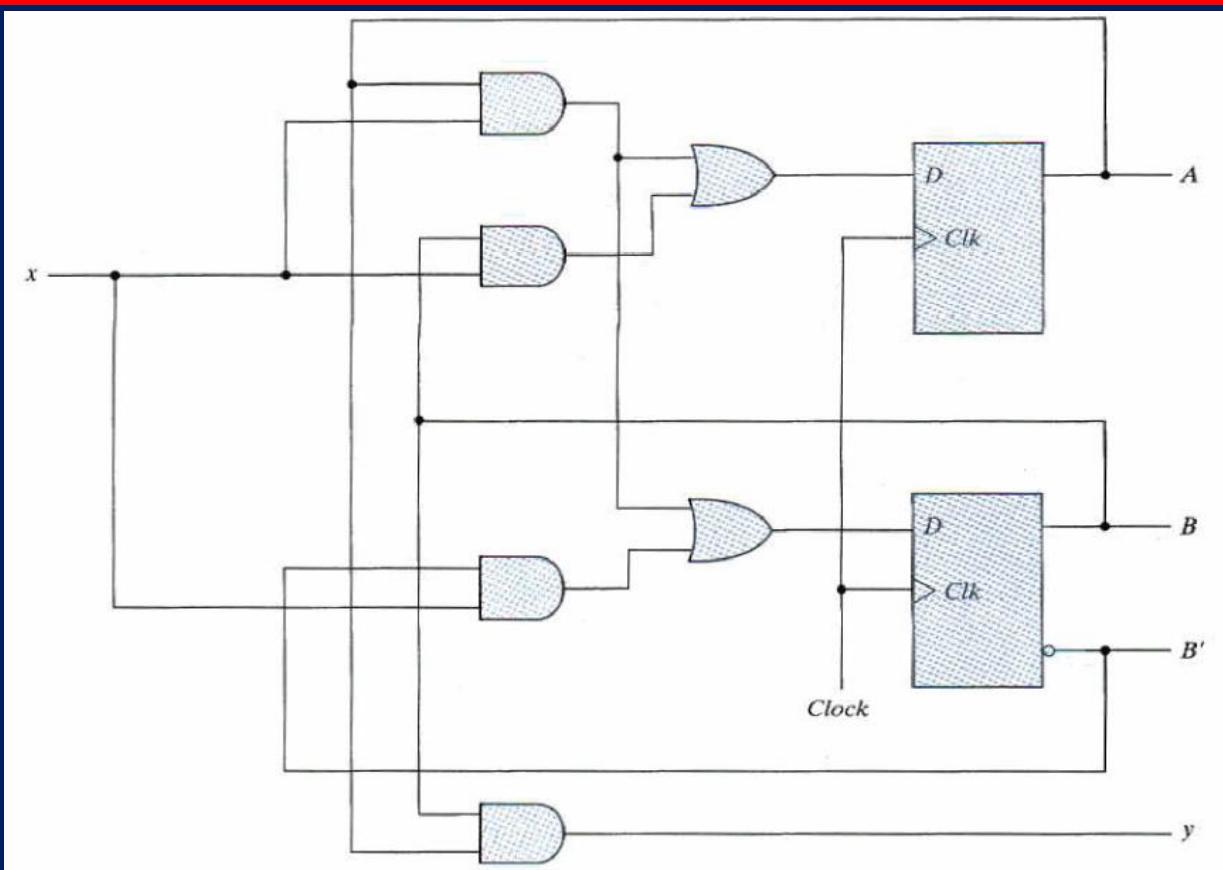
$$D_B = Ax + B^t x$$



$$y = AB$$

Design Procedures

Logic Map for Sequence Detector



Other FF

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

D Flip-Flop

D	Q(t + 1)	
0	0	Reset
1	1	Set

T Flip-Flop

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

Excitation Tables

Table that lists input conditions to achieve the required transition.

Flip-Flop Excitation Tables

$Q(t)$	$Q(t = 1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(a) JK

$Q(t)$	$Q(t = 1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

(b) T

State Table(JK Flip-Flop)

State Table and JK Flip-Flop Inputs

Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J _A	K _A	J _B	K _B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Maps (JK Flip-Flop)

A Karnaugh map for the function $J_A = Bx'$. The columns are labeled 00, 01, 11, 10 and the rows are labeled 0 and 1. The map shows:
Row 0: m_0 , m_1 , m_3 , $m_2 = 1$
Row 1: $m_4 = X$, $m_5 = X$, $m_7 = X$, $m_6 = X$

$$J_A = Bx'$$

A Karnaugh map for the function $K_A = Bx$. The columns are labeled 00, 01, 11, 10 and the rows are labeled 0 and 1. The map shows:
Row 0: m_0 , m_1 , $m_3 = X$, $m_2 = X$
Row 1: m_4 , m_5 , $m_7 = 1$, m_6

$$K_A = Bx$$

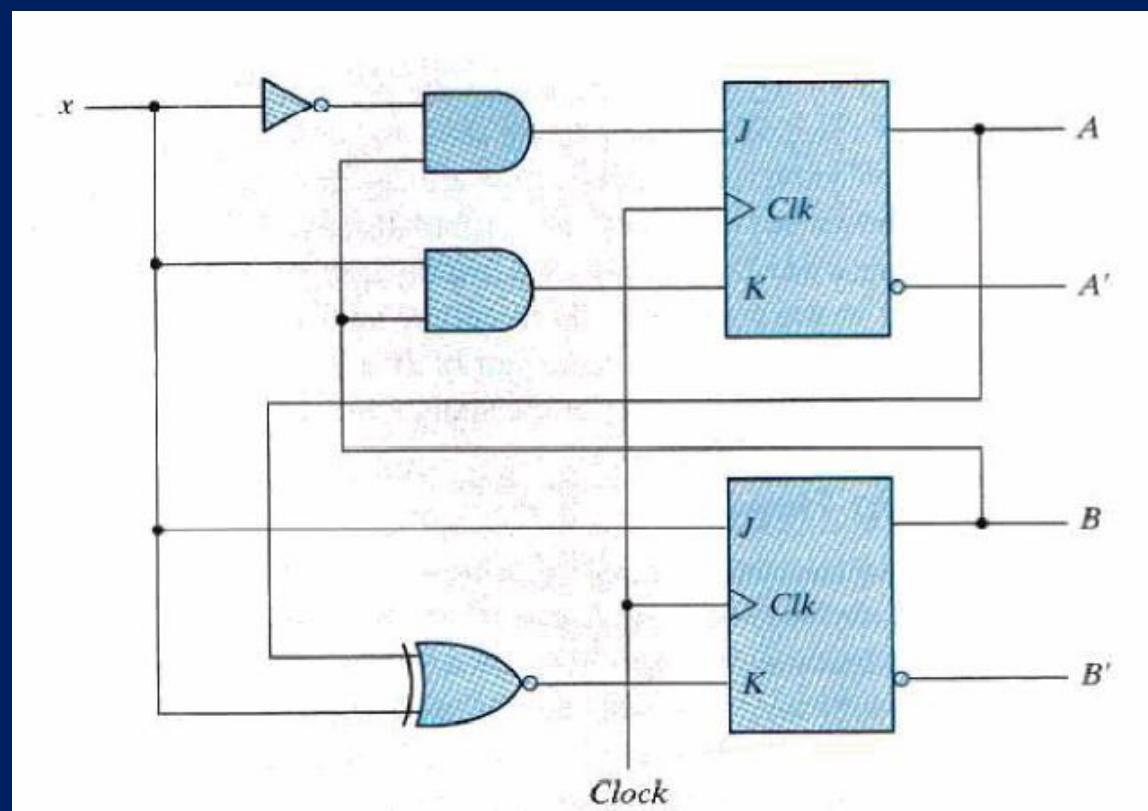
A Karnaugh map for the function $J_B = x$. The columns are labeled 00, 01, 11, 10 and the rows are labeled 0 and 1. The map shows:
Row 0: m_0 , $m_1 = 1$, $m_3 = X$, $m_2 = X$
Row 1: m_4 , $m_3 = 1$, $m_7 = X$, $m_6 = X$

$$J_B = x$$

A Karnaugh map for the function $K_B = (A \oplus x)'$. The columns are labeled 00, 01, 11, 10 and the rows are labeled 0 and 1. The map shows:
Row 0: $m_0 = X$, m_1 , m_3 , $m_2 = 1$
Row 1: m_4 , $m_5 = X$, $m_7 = 1$, m_6

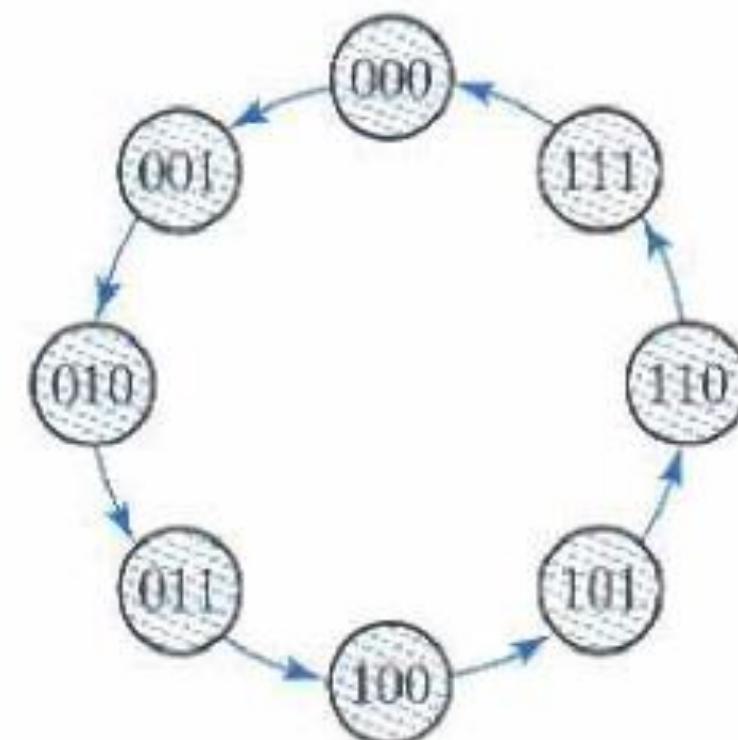
$$K_B = (A \oplus x)'$$

Logic Diagram for sequential circuit(JK FF)



Chapter 8

State Diagram of a 3-bit binary counter

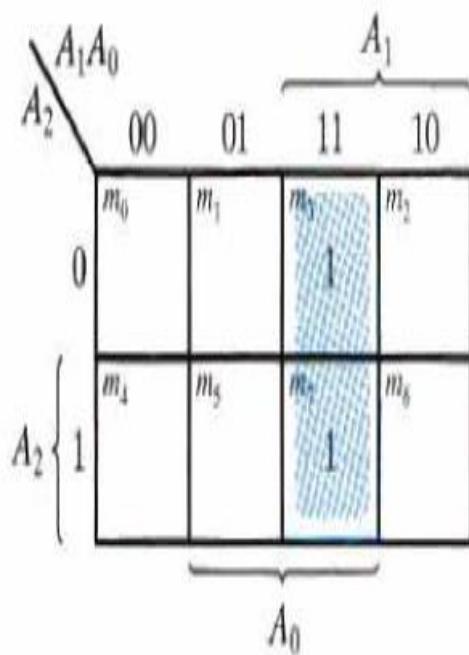


State Table of a 3-bit binary counter

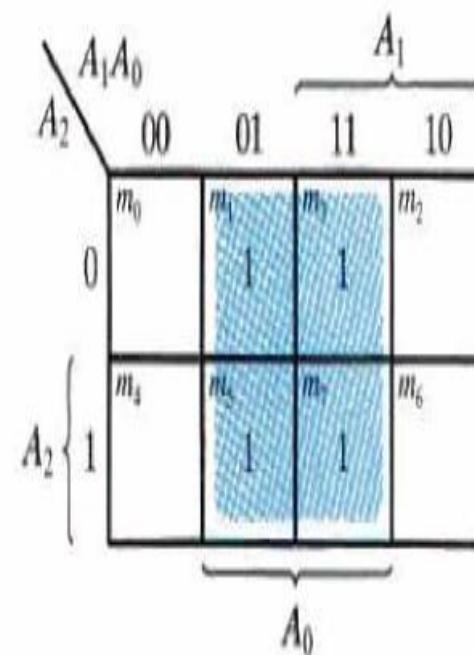
State Table for Three-Bit Counter

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	0	1
1	1	1	0	0	0	1	1	1

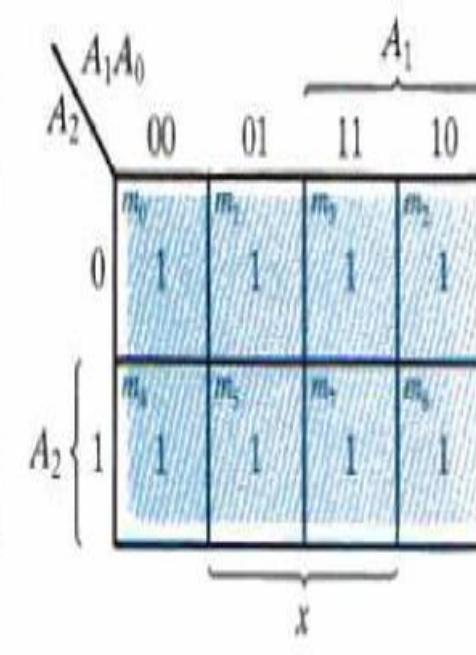
Maps of a 3-bit binary counter



$$T_{A2} = A_1 A_0$$

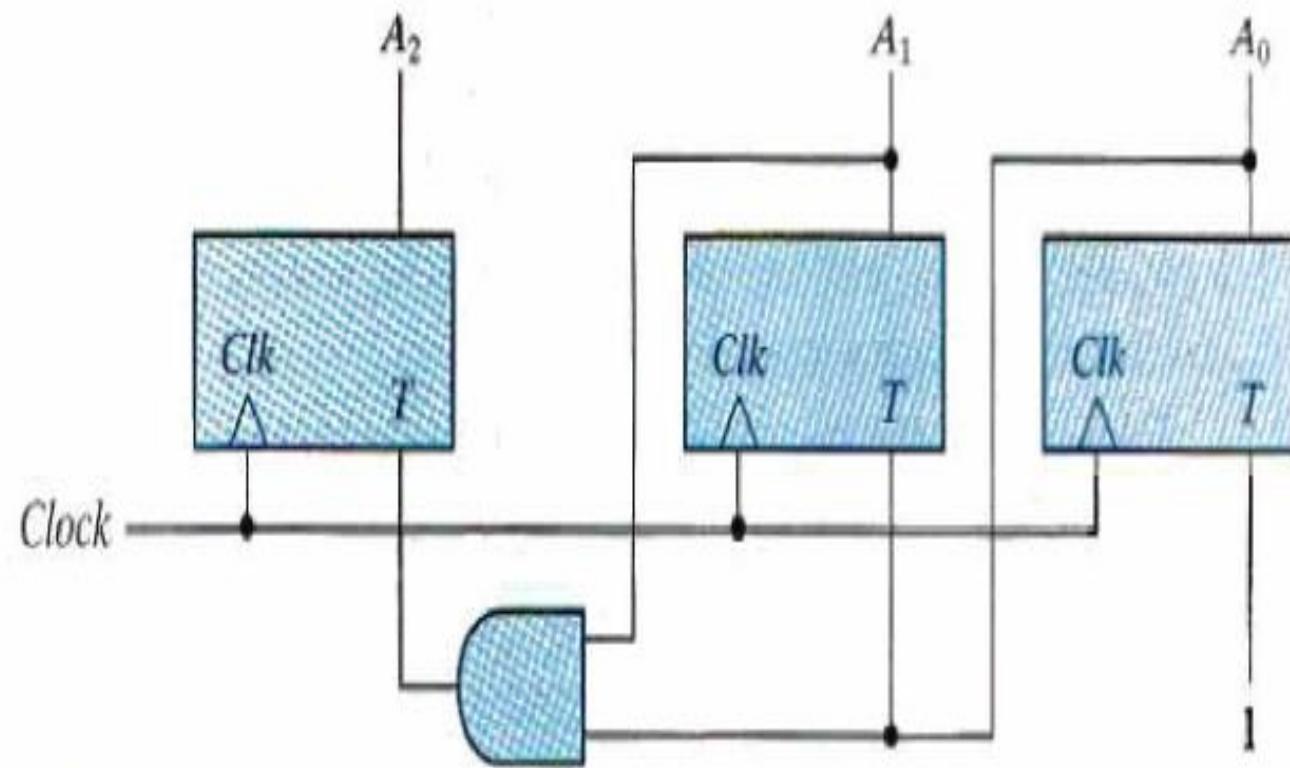


$$T_{A1} = A_0$$



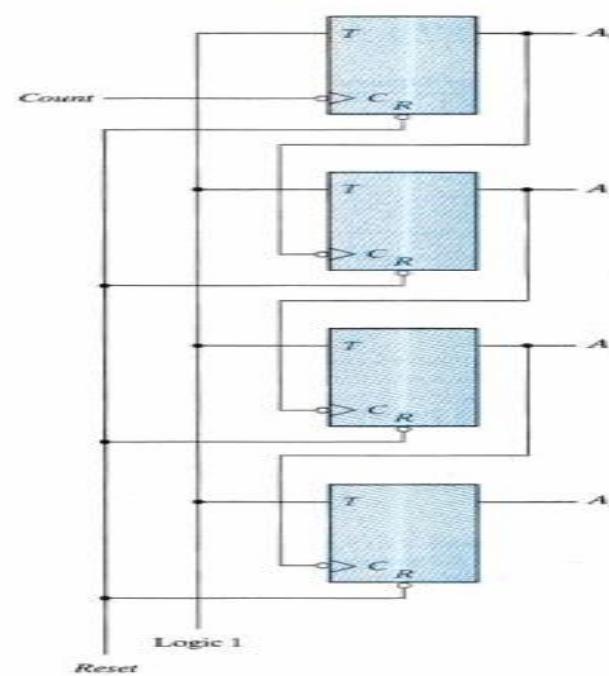
$$T_{A0} = 1$$

Logic Diagram of a 3-bit binary counter



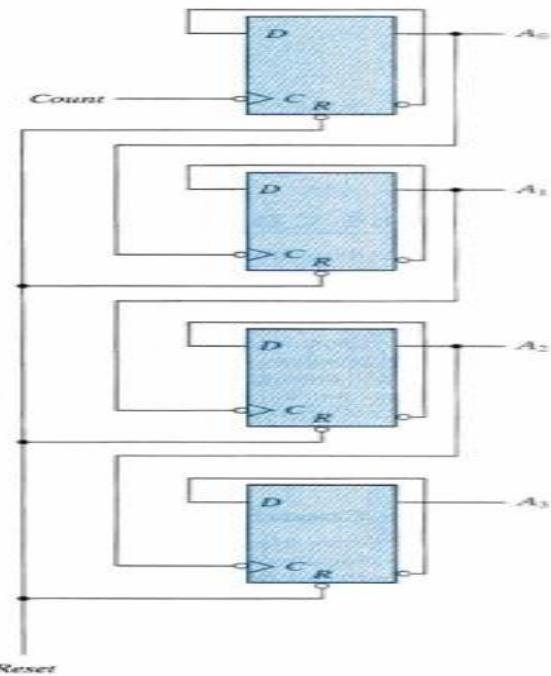
4-bit BINARY RIPPLE COUNTER

With T FF



(a) With T flip-flops

With D FF



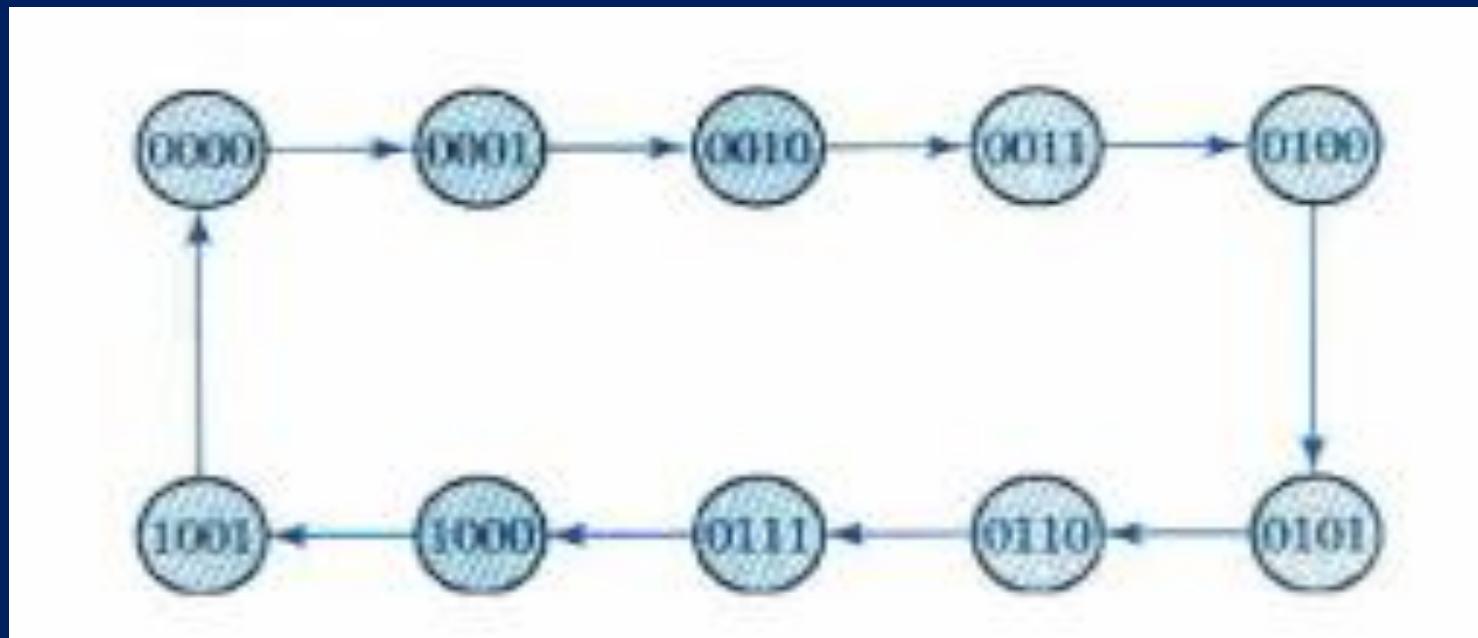
(b) With D flip-flops

4-bit RIPPLE COUNTER

Binary Count Sequence

A_3	A_2	A_1	A_0
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

State Diagram Decimal BCD COUNTER

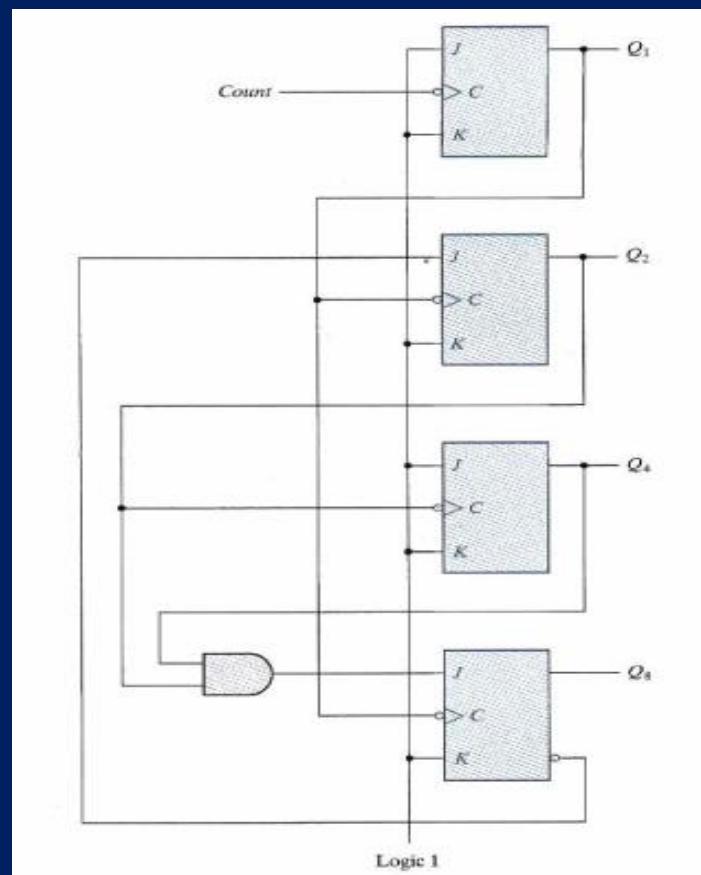


0 → 9 → 0

UP and Down Binary Counter BCD RIPPLE COUNTER

If C goes from 1 to 0,
set if J=1,
cleared if K=1

complemented if J=K=1
unchanged

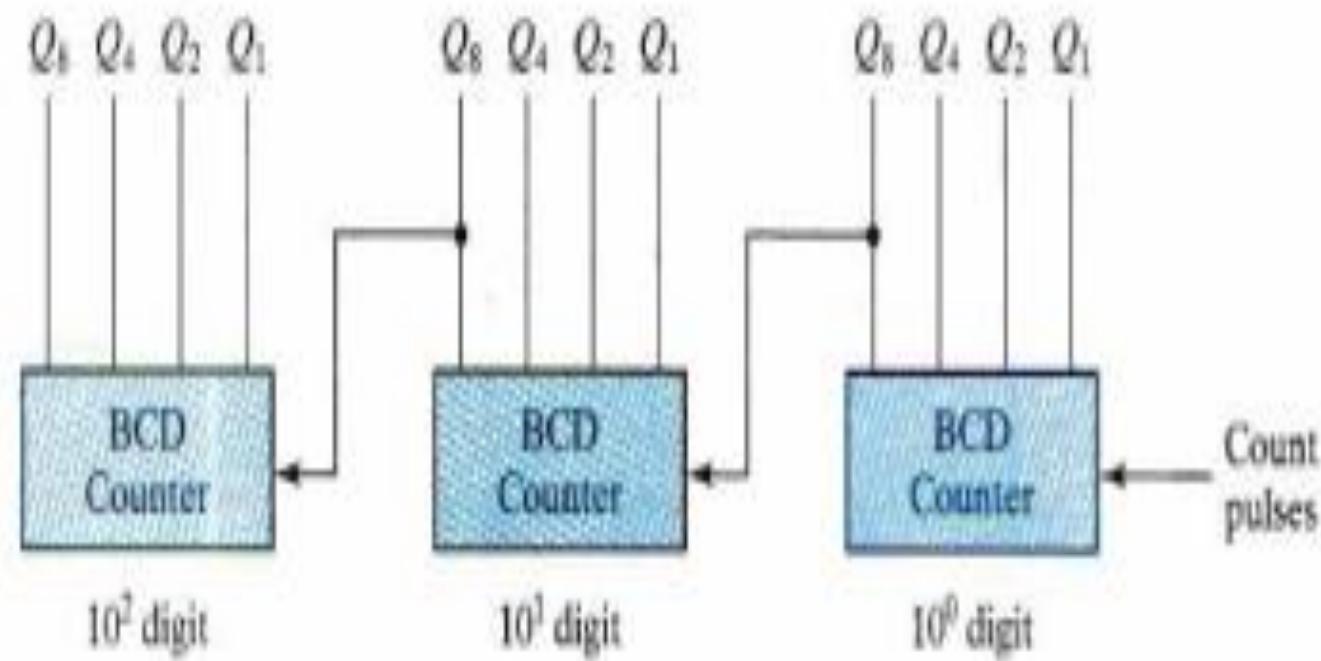


$$Q_1 \rightarrow Q_2, Q_8$$

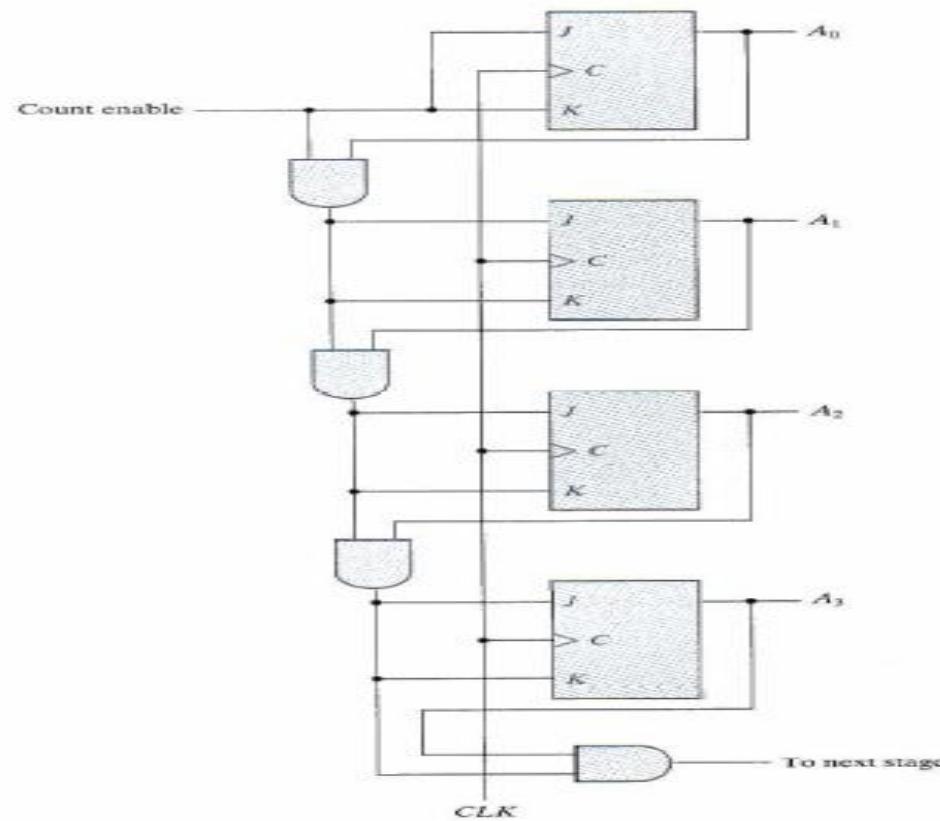
$$Q_2 \rightarrow Q_4$$

Inputs J,K either
- to 1 ,or
- outputs of FF

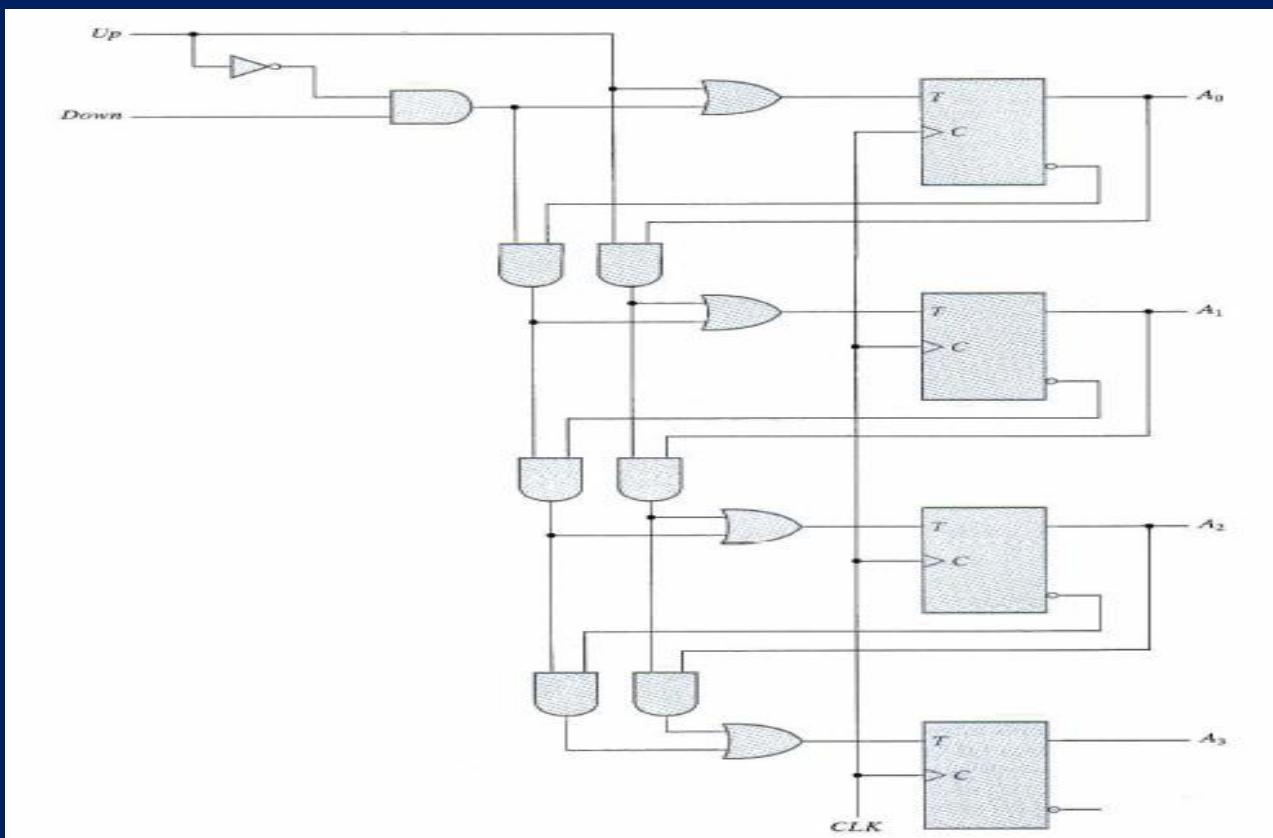
Three decade Decimal BCD Counter 0-999



4-bit Synchronous Binary Counter



4-bit Up-Down Binary Counter



4-bit Binary Counter

State table of BCD Counter

State Table for BCD Counter

Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

BCD COUNTER Equations

$$T_{Q1} = 1$$

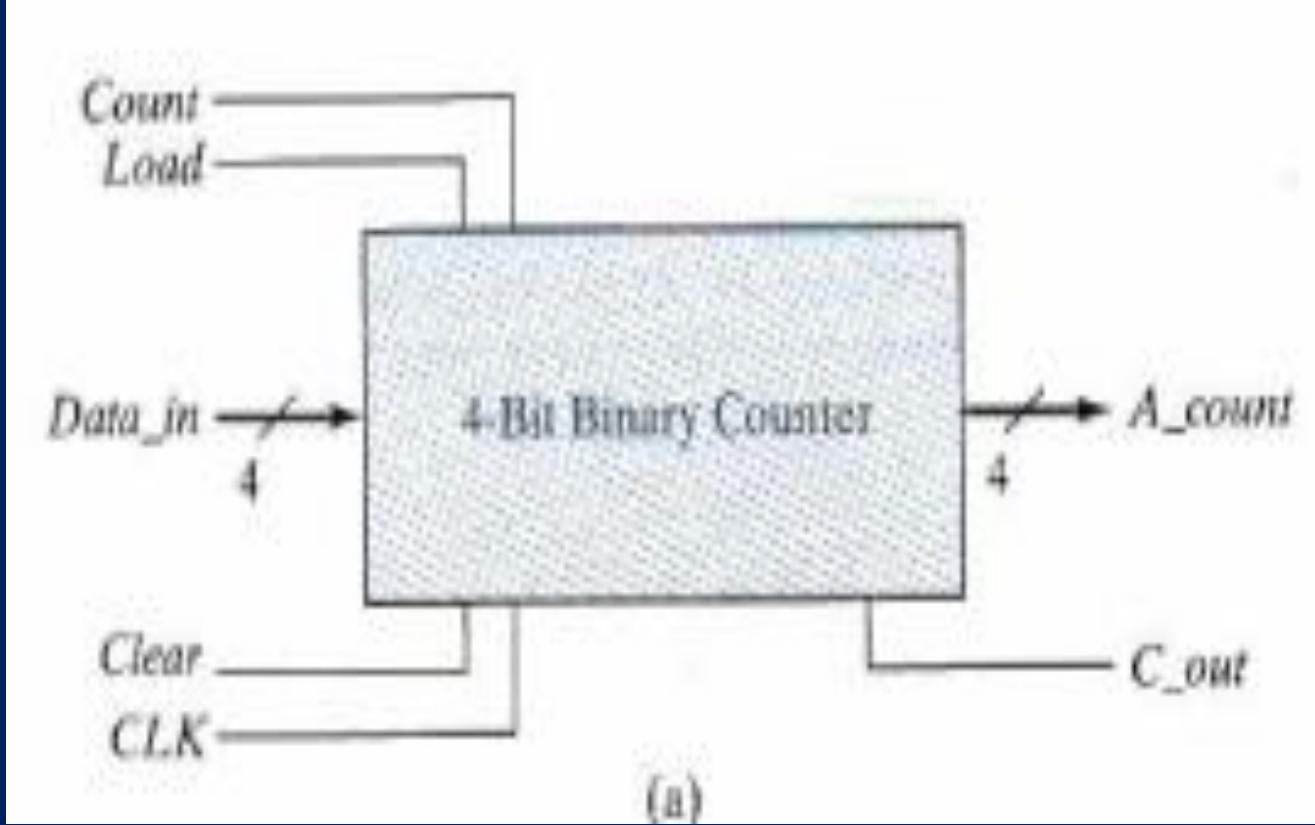
$$T_{Q4} = Q_2 \cdot Q_1$$

$$T_{Q2} = Q_8' \cdot Q_1$$

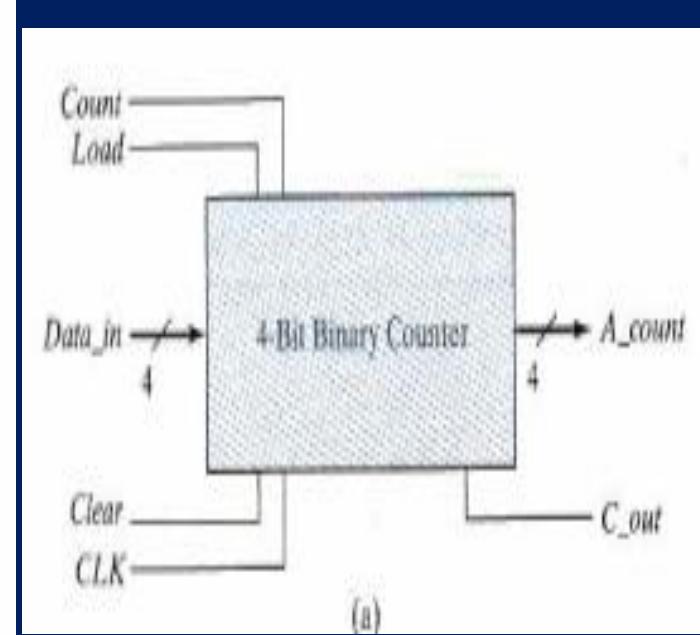
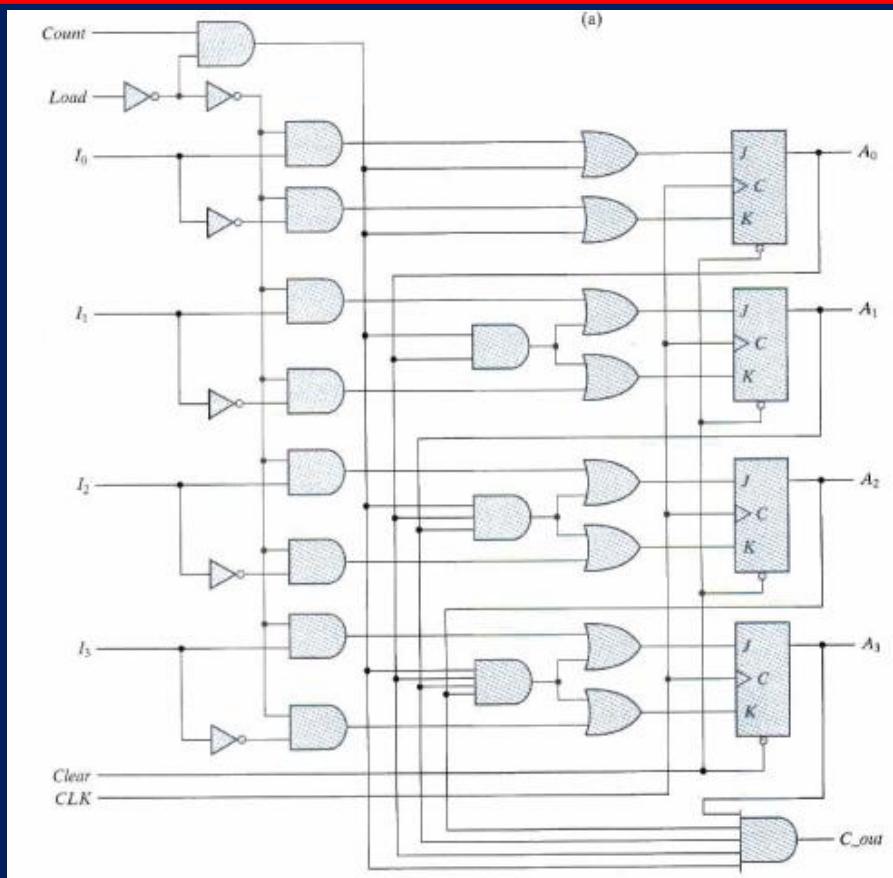
$$T_{Q8} = Q_8 \cdot Q_1 + Q_4 \cdot Q_2 \cdot Q_1$$

$$y = Q_8 \cdot Q_1$$

4-bit Binary Counter With Parallel Load



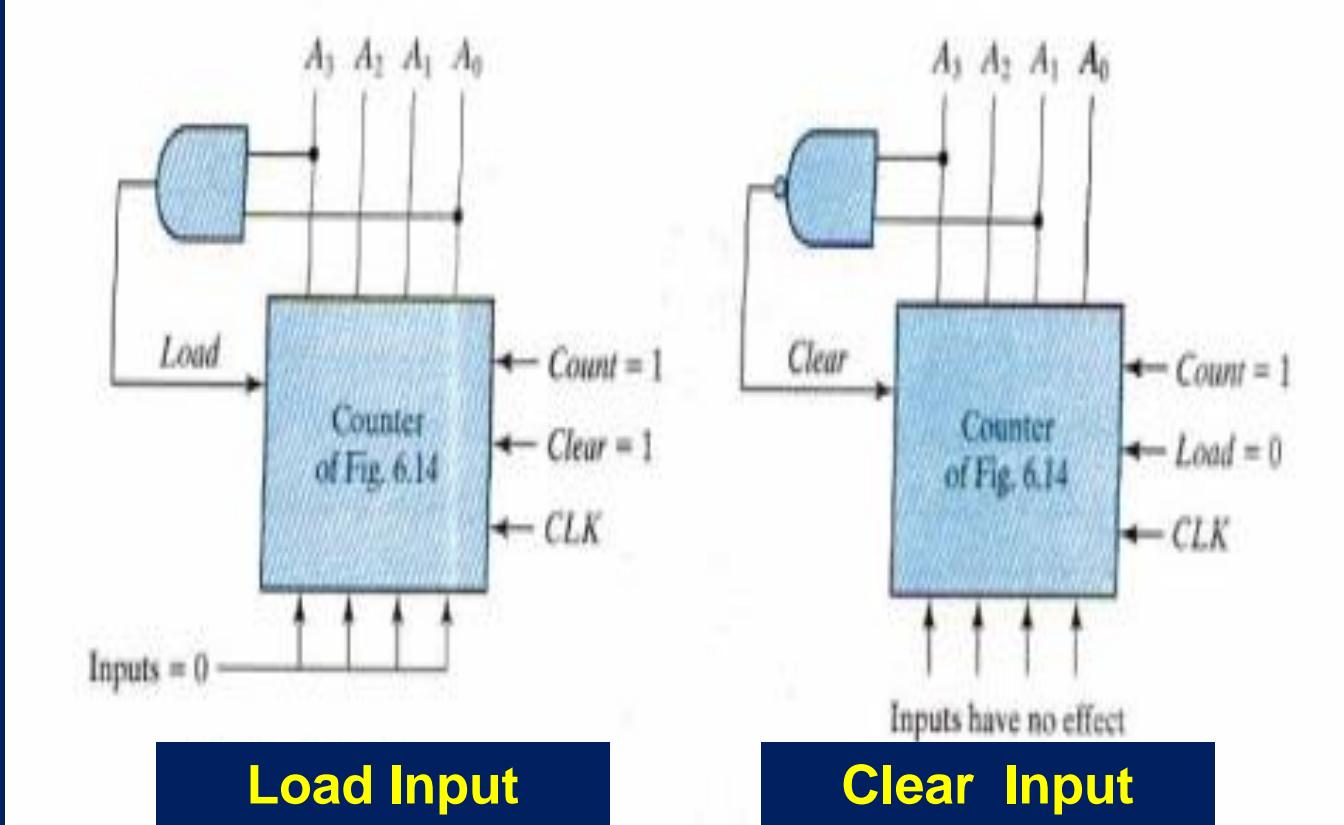
4-bit Binary Counter With Parallel Load



4-bit Binary Counter With Parallel Load Function Table

Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state
1	↑	0	0	No change

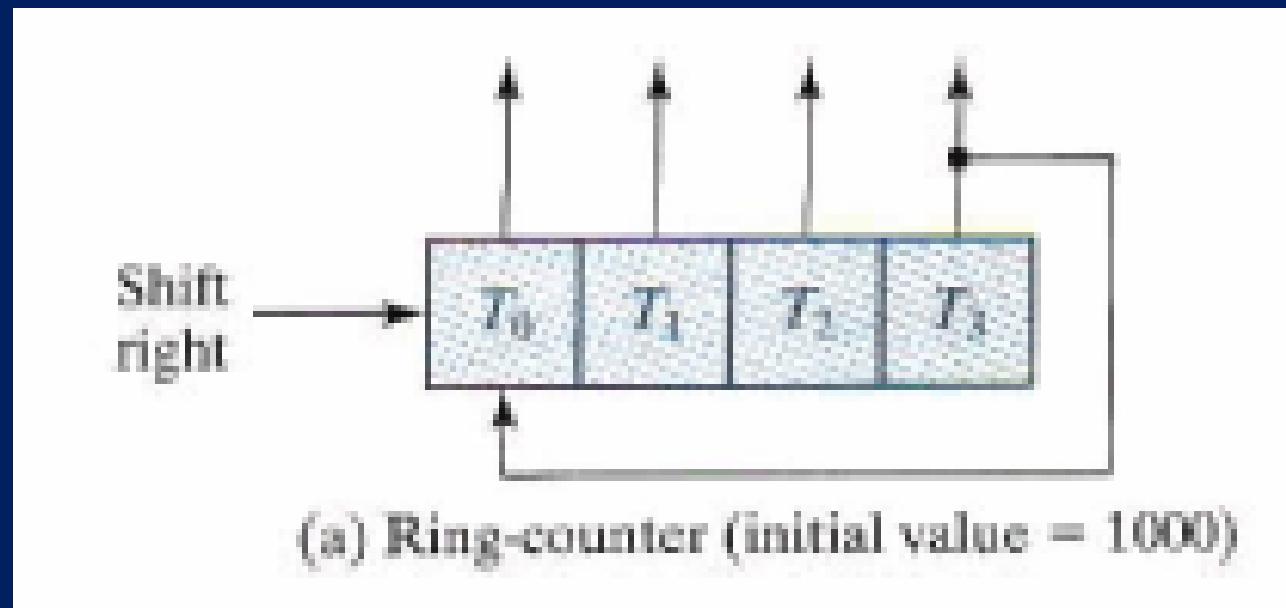
4-bit Binary Counter With Parallel Load Implementation



OTHER COUNTERS

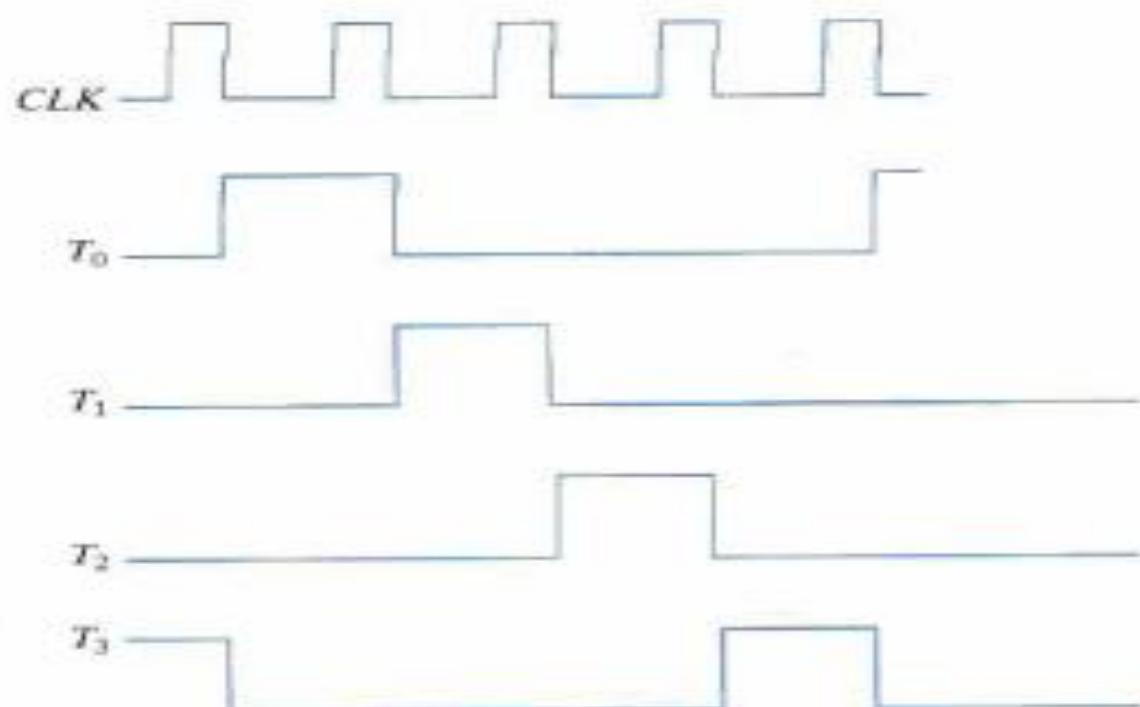
RING COUNTER

Ring Counter: circular shift register, one FF set, all others cleared.



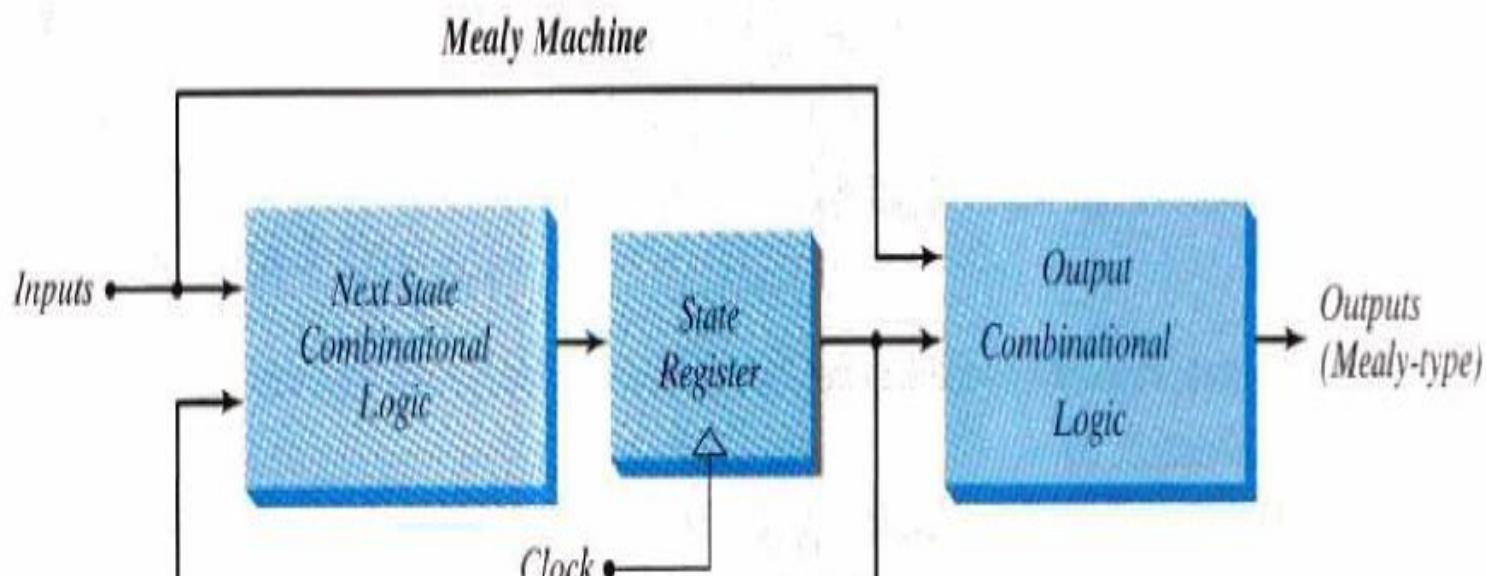
OTHER COUNTERS

RING COUNTER



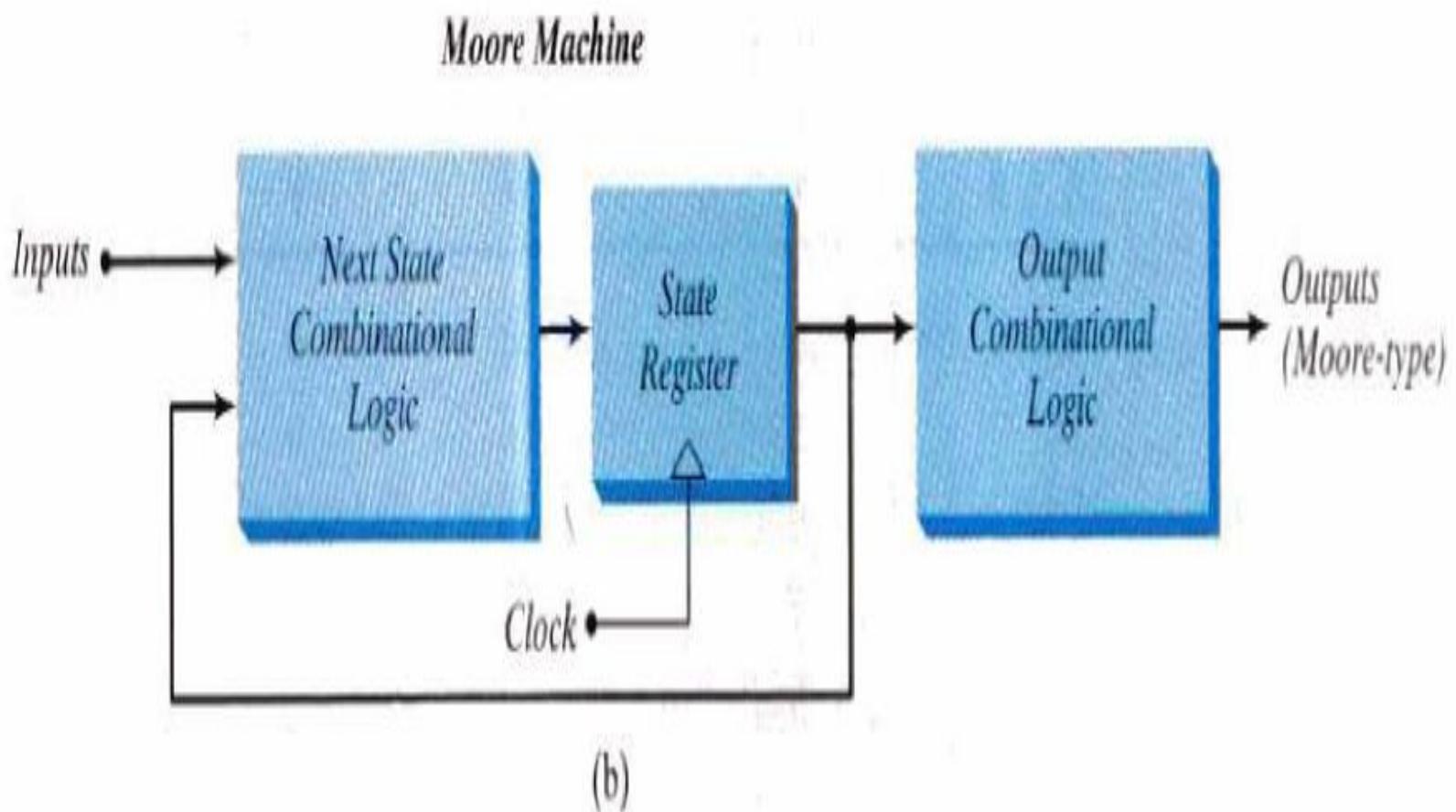
(b) Sequence of four timing signals

Mealy Machine



(a)

Moore Machine



Chapter 9

REGISTERS and COUNTERS

1. REGISTERS

REGISTERS

WHAT IS A REGISTER ?

REGISTERS

**1- A register is a group of flip-flops,
each one of which is capable of
storing one bit of information.**

REGISTERS

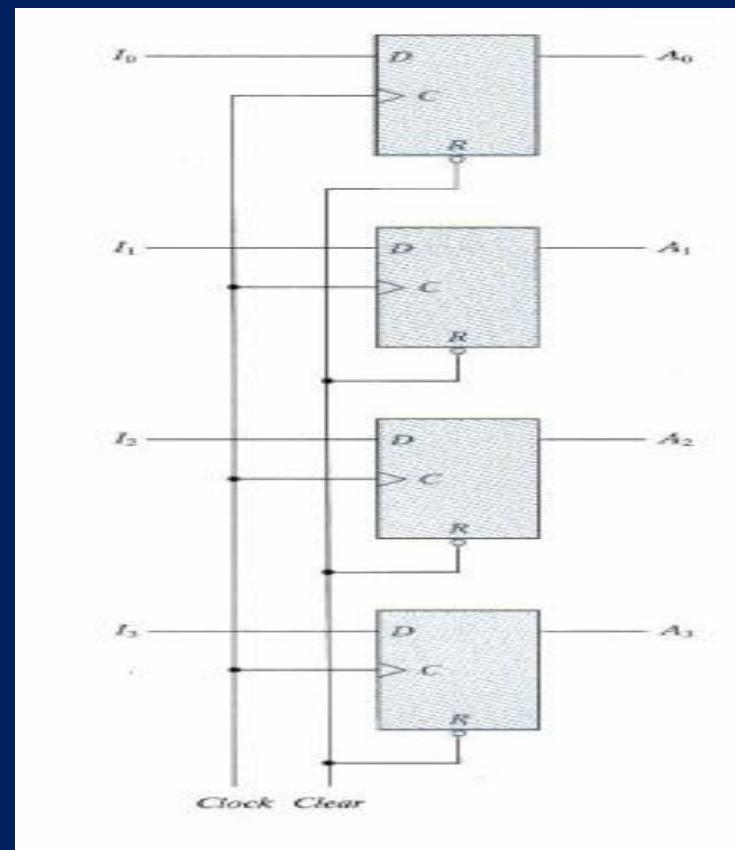
2-An n-bit register consists of a group of *n flip-flops capable of storing n bits of binary information.*

REGISTERS

3 - In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.

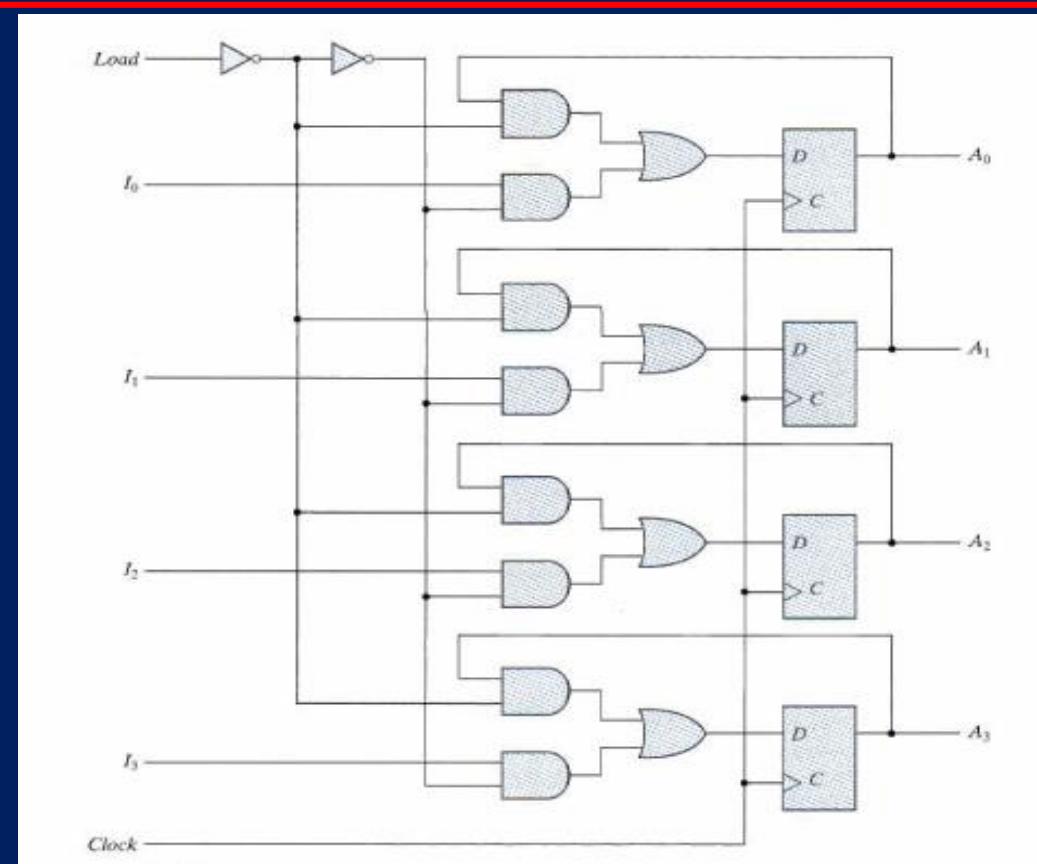
REGISTERS

4-bit REGISTER



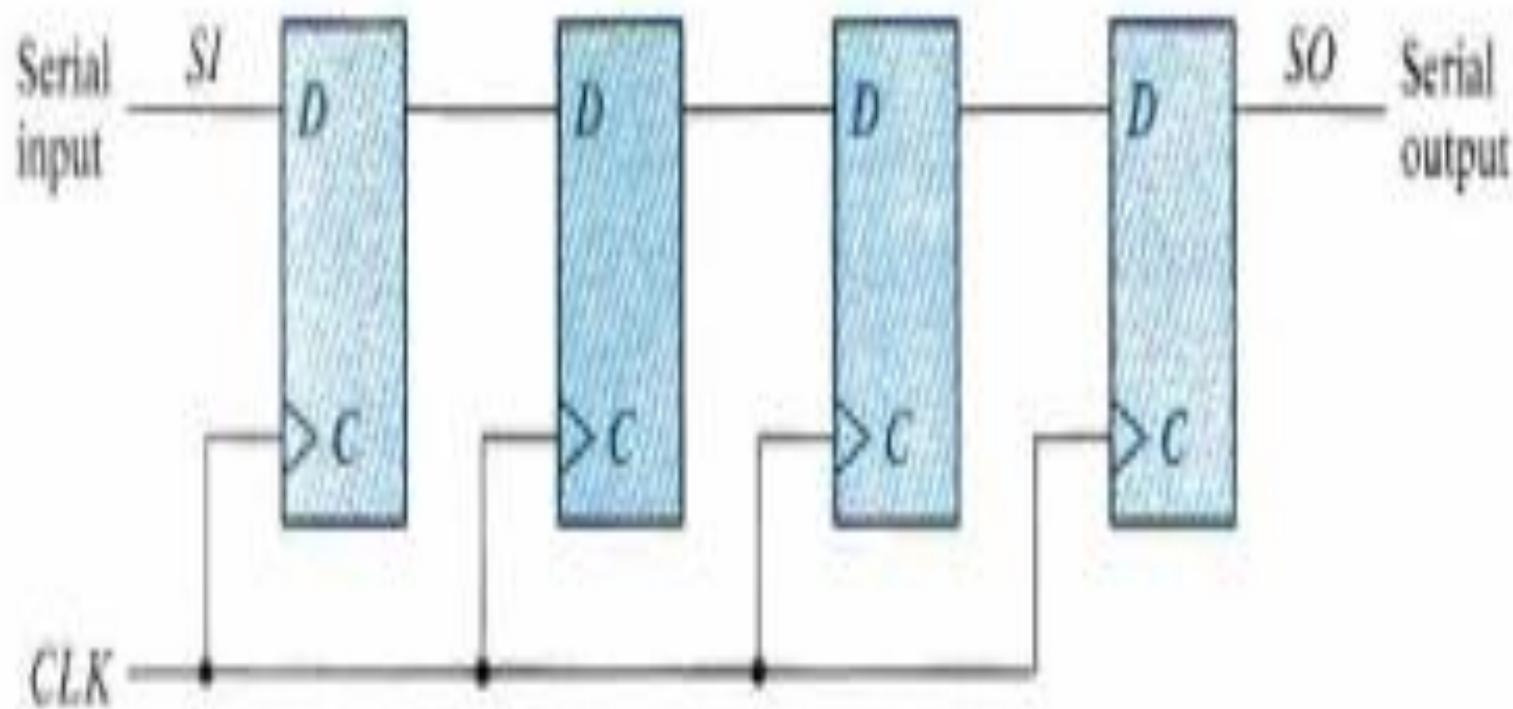
REGISTERS

REGISTER with parallel load



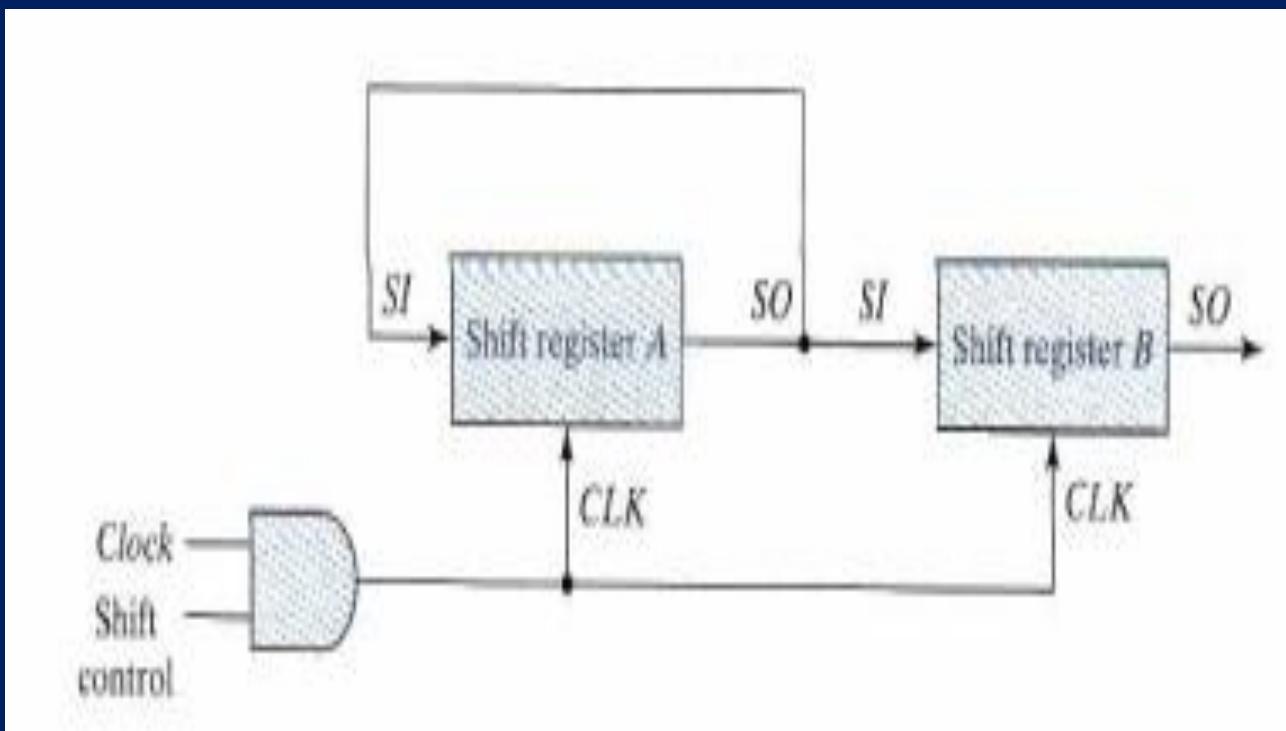
REGISTERS

SHIFT REGISTER



REGISTERS

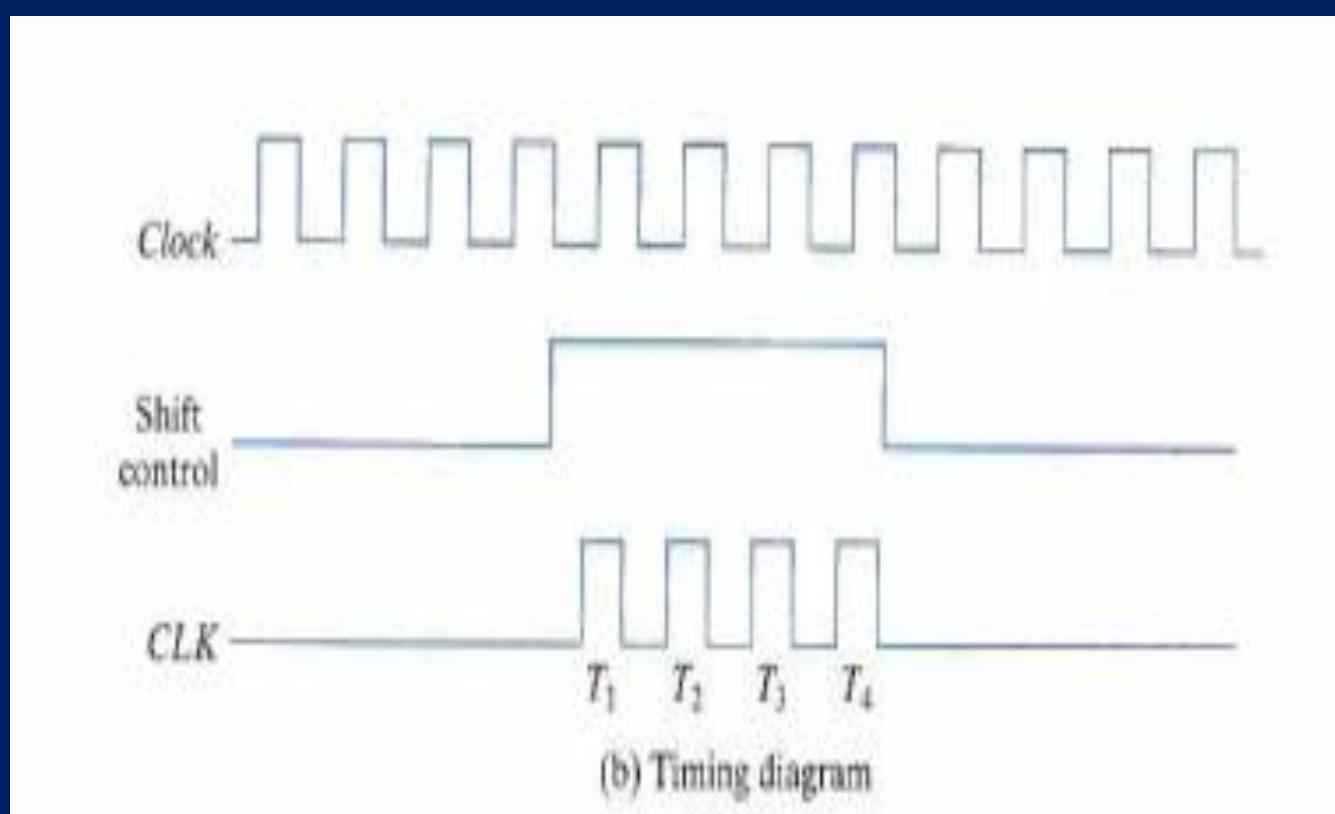
SERIAL TRANSFER



BLOCK DIAGRAM

REGISTERS

SERIAL TRANSFER

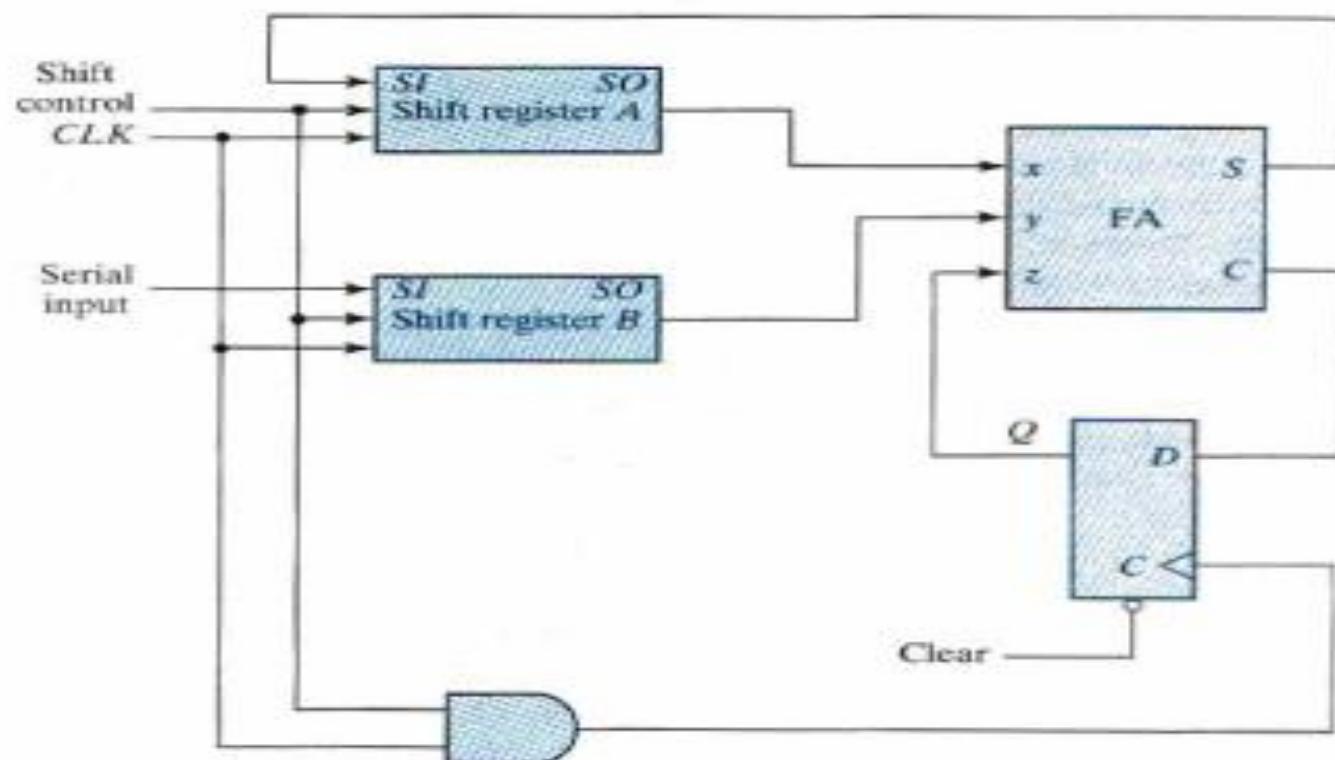


Registers Serial Transfer

Serial-Transfer Example

Timing Pulse	Shift Register A	Shift Register B
Initial value	1 0 1 1	0 0 1 0
After T_1	1 1 0 1	1 0 0 1
After T_2	1 1 1 0	1 1 0 0
After T_3	0 1 1 1	0 1 1 0
After T_4	1 0 1 1	1 0 1 1

REGISTERS SERIAL ADDITION



Registers Serial Adder

$$J_Q = x \cdot y$$

$$K_Q = x' \cdot y' = (x + y)'$$

$$S = x \square y \square Q$$

REGISTERS

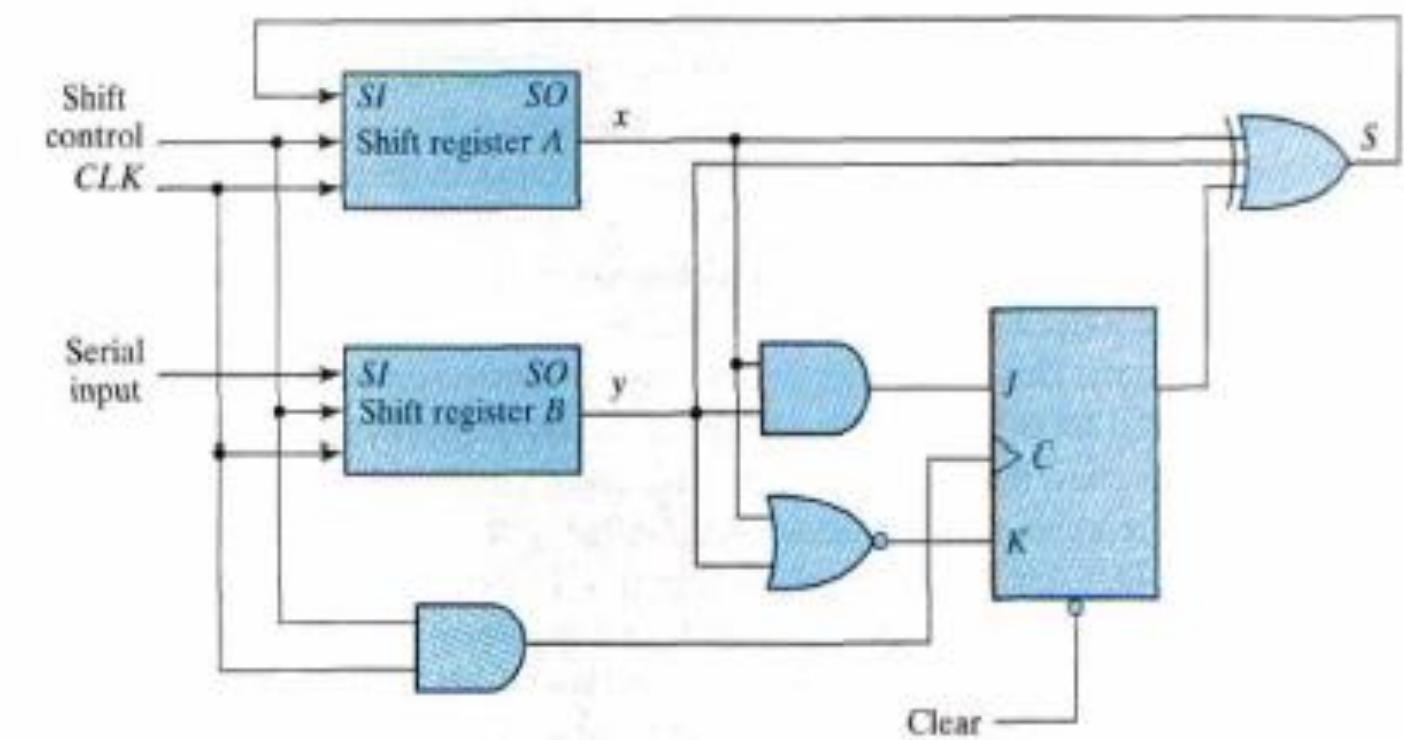
STATE TABLE OF SERIAL ADDER

State Table for Serial Adder

Present State Q	Inputs x y		Next State Q	Output s	Flip-Flop Inputs	
	J_Q	K_Q			J_Q	K_Q
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

REGISTERS

SECOND FORM OF SERIAL ADDER



UNIVERSAL SHIFT REGISTER

1. A clear control to clear the register to 0.
2. A *clock input* to synchronize operations.
3. A shift-right control to enable the shift-right operation and the *serial input and output lines* associated with the shift right.

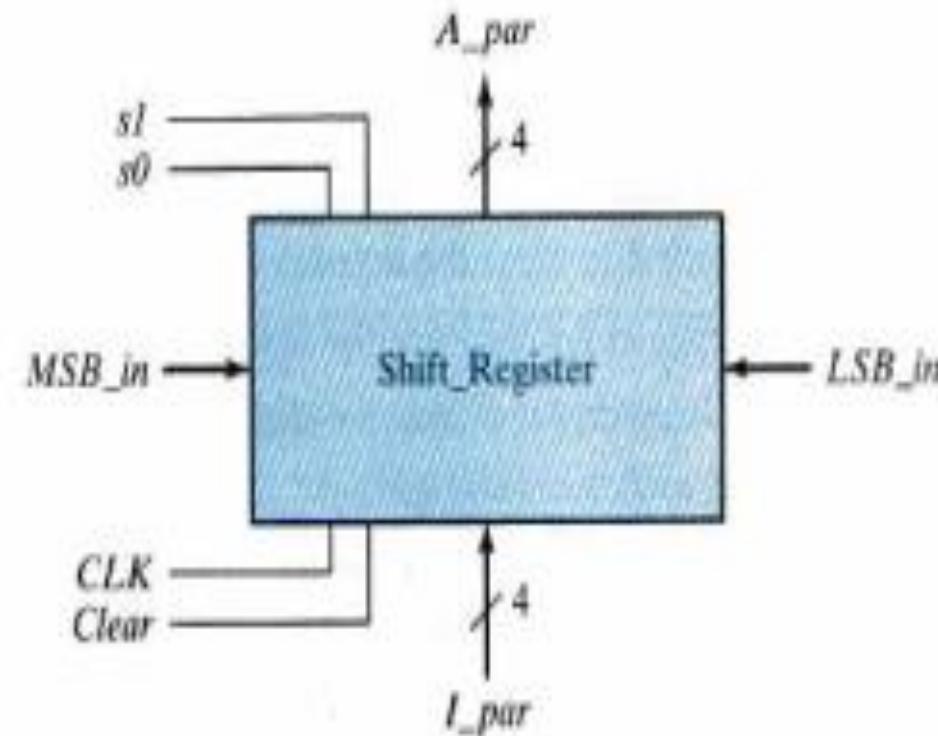
UNIVERSAL SHIFT REGISTER

- 4- A shift-left control to enable the shift-left operation and the *serial input and output* associated with the shift left.
- 5- A *parallel-load control* to enable a *parallel transfer ad the n input lines associated with the parallel transfer.*

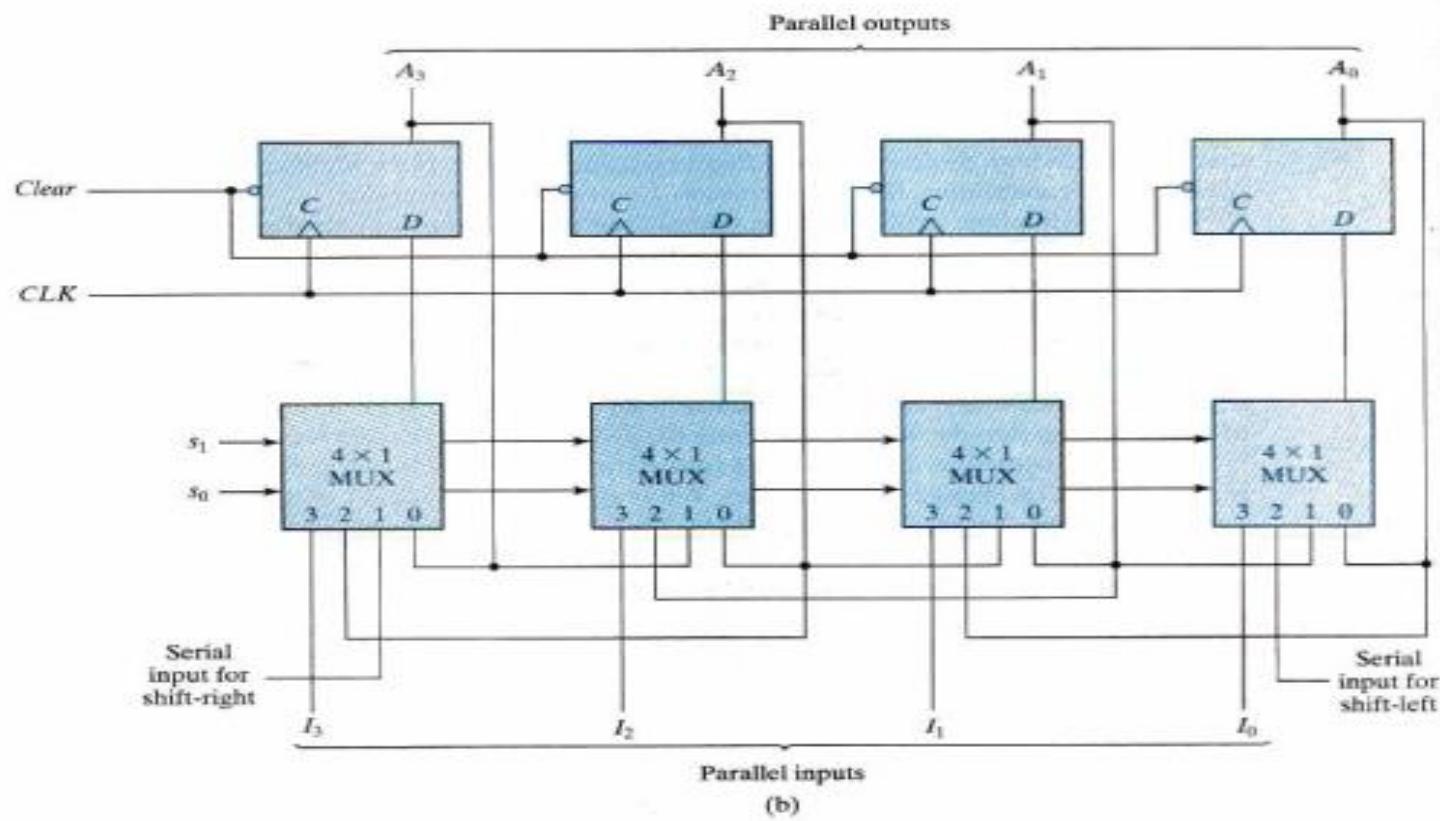
UNIVERSAL SHIFT REGISTER

6. n parallel output lines.
7. A control state that leaves the *information in the register unchanged in response to the clock*. Other shift registers may have only some of the preceding functions, with at least one *shift operation*.

4-bit UNIVERSAL SHIFT REGISTER



4-bit UNIVERSAL SHIFT REGISTER



4-bit UNIVERSAL SHIFT REGISTER

Function Table

Function Table for the Register of Fig. 6.7

Mode Control

s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load