Lecture – 5

Combinational Logic Circuits

Lesson Outcomes

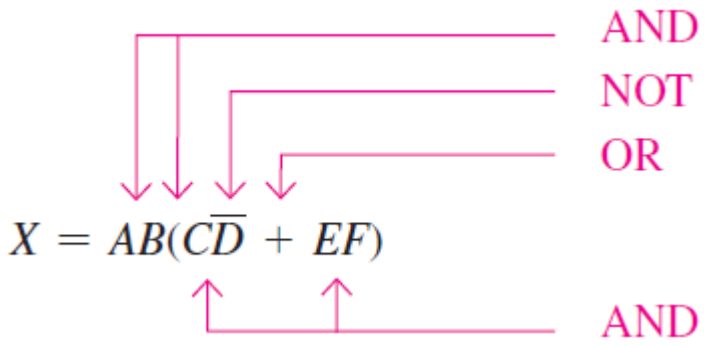
After completing this lecture, students will be able to

- *Implement combinational logic circuits including adder, subtractor, comparator, encoder, decoder, multiplexer, demultiplexers. etc.*
- *Convert from binary to Gray code, and Gray code to binary by using logic devices.*
- *Use parity generators and checkers to detect bit errors in digital systems*

Key Terms

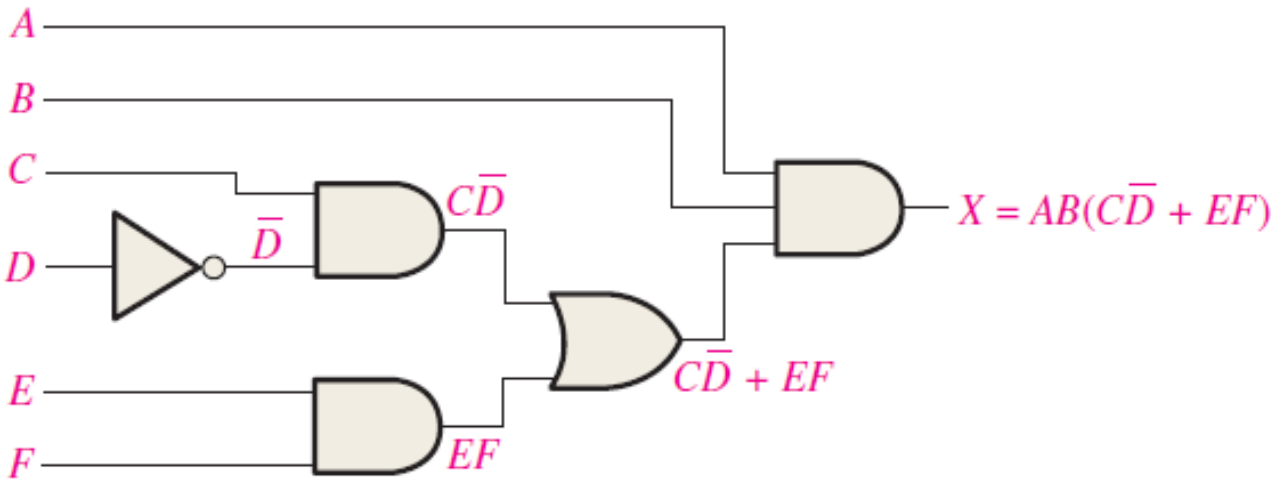
- **Comparator** A digital circuit that compares the magnitudes of two quantities and produces an output indicating the relationship of the quantities.
- **Decoder** A digital circuit that converts coded information into a familiar or noncoded form.
- **Demultiplexer (DEMUX)** A circuit that switches digital data from one input line to several output lines in a specified time sequence.
- **Encoder** A digital circuit that converts information to a coded form.
- **Full-adder** A digital circuit that adds two bits and an input carry to produce a sum and an output carry.
- **Half-adder** A digital circuit that adds two bits and produces a sum and an output carry. It cannot handle input carries.
- **Look-ahead carry** A method of binary addition whereby carries from preceding adder stages are anticipated, thus eliminating carry propagation delays.
- **Multiplexer (MUX)** A circuit that switches digital data from several input lines onto a single output line in a specified time sequence.
- **Parity bit** A bit attached to each group of information bits to make the total number of 1s odd or even for every group of bits.
- **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.

Implementing logic circuit from Boolean expression

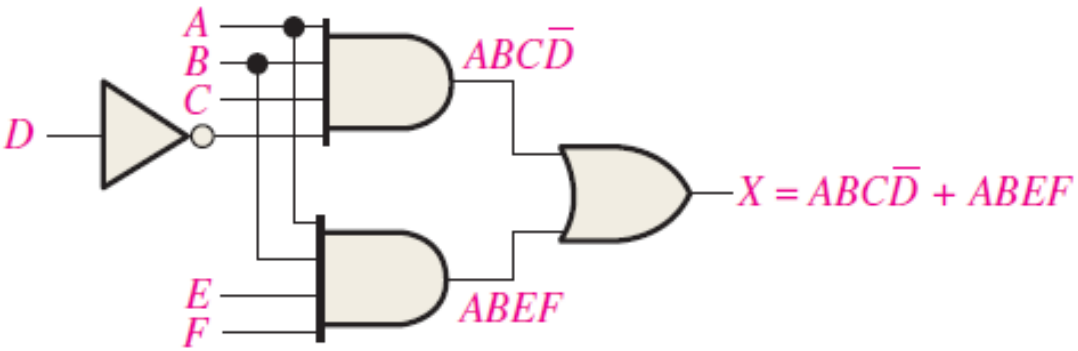


The logic gates required to implement $X = AB(C\overline{D} + EF)$ are as follows:

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



(a)



(b) Sum-of-products implementation of the circuit in part (a)

Combinational logic circuit

EXAMPLE 5–6

Develop a logic circuit with four input variables that will only produce a 1 output when exactly three input variables are 1s.

Solution

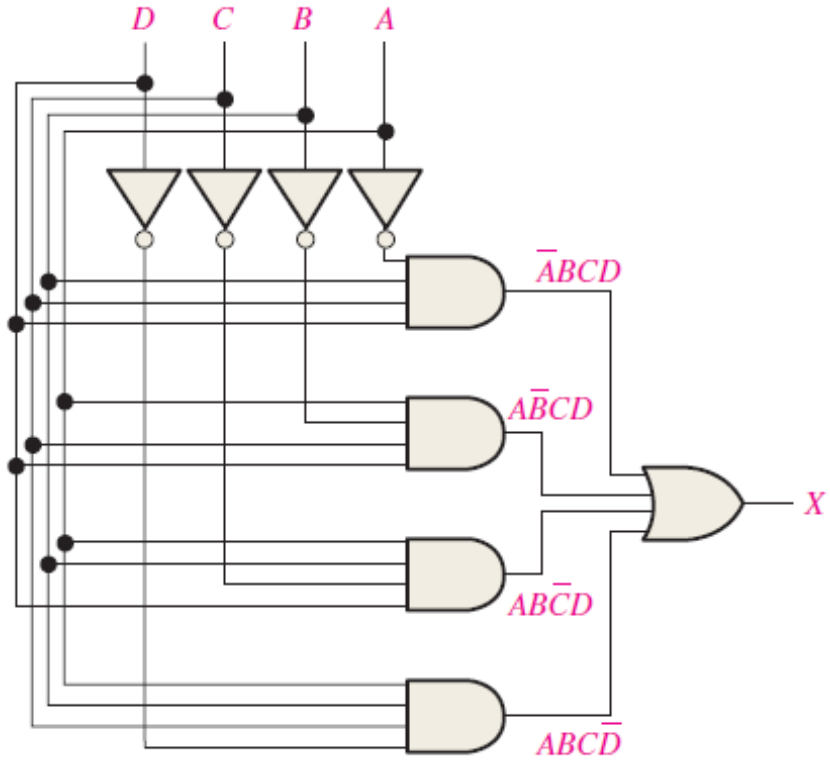
Out of sixteen possible combinations of four variables, the combinations in which there are exactly three 1s are listed in Table 5–5, along with the corresponding product term for each.

TABLE 5–5

A	*B*	*C*	*D*	Product Term
0	1	1	1	$\overline{A}BCD$
1	0	1	1	$A\overline{B}CD$
1	1	0	1	$AB\overline{C}D$
1	1	1	0	$ABC\overline{D}$

The product terms are ORed to get the following expression:

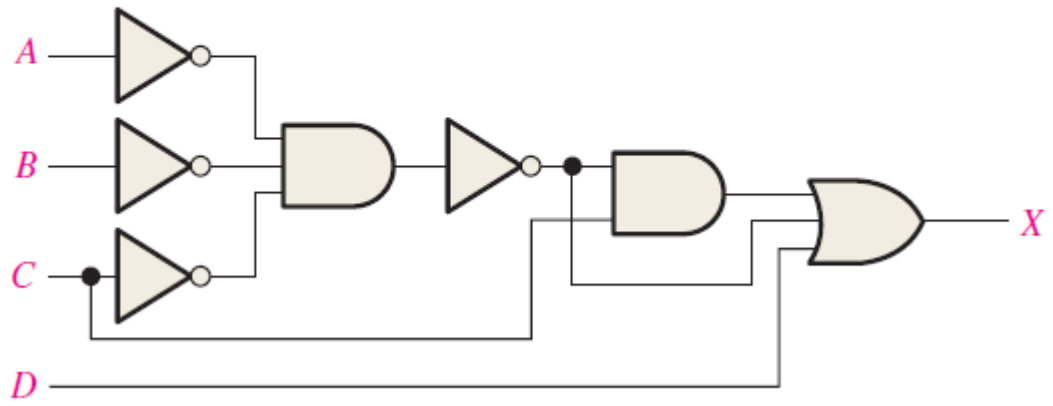
$$X = \overline{A}BCD + A\overline{B}CD + AB\overline{C}D + ABC\overline{D}$$



Combinational logic circuit

EXAMPLE 5–7

Reduce the combinational logic circuit to a minimum form.



Solution

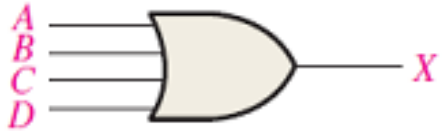
The expression for the output of the circuit is

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

Applying DeMorgan's theorem and Boolean algebra,

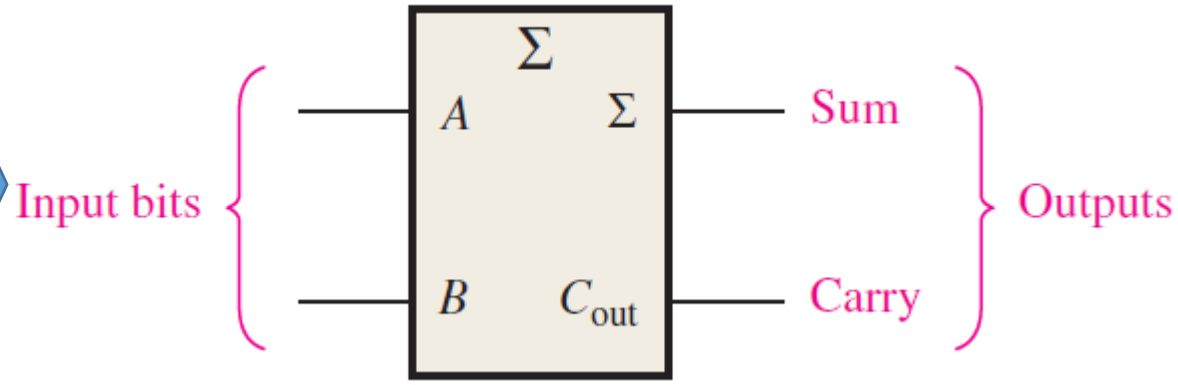
$$\begin{aligned} X &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}})C + \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + D \\ &= AC + BC + CC + A + B + C + D \\ &= AC + BC + C + A + B + C + D \\ &= C(A + B + 1) + A + B + D \\ X &= A + B + C + D \end{aligned}$$

The simplified circuit is a 4-input OR gate as shown in Figure



Half Adder

Half Adder Logic Symbol



Half Adder Logic Diagram

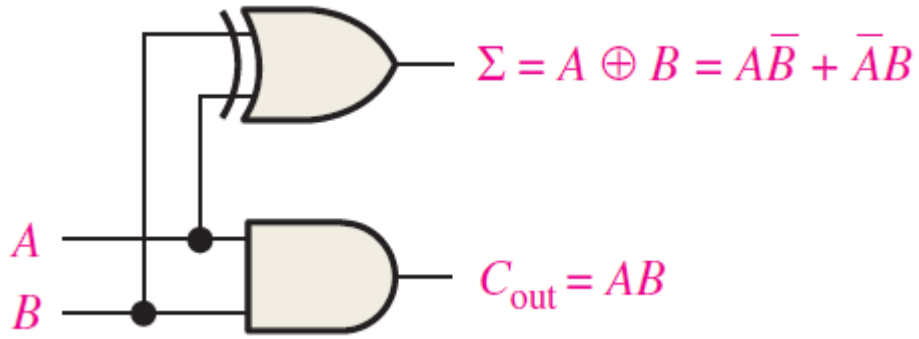


TABLE 6–1

Half-adder truth table.

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

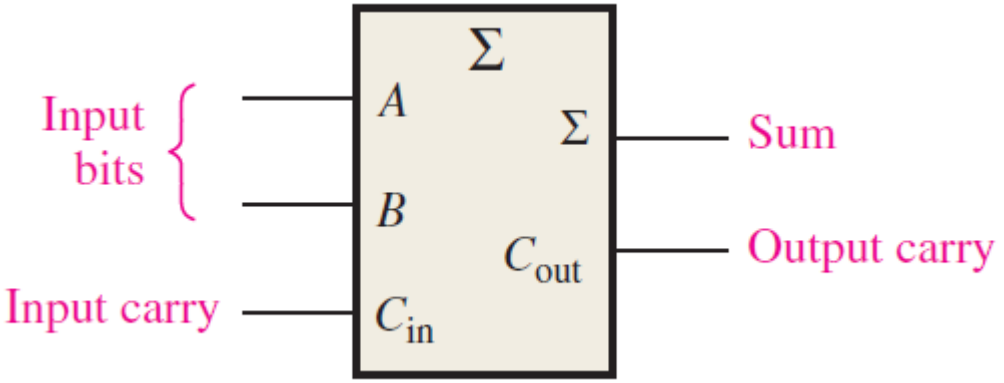
Σ = sum

C_{out} = output carry

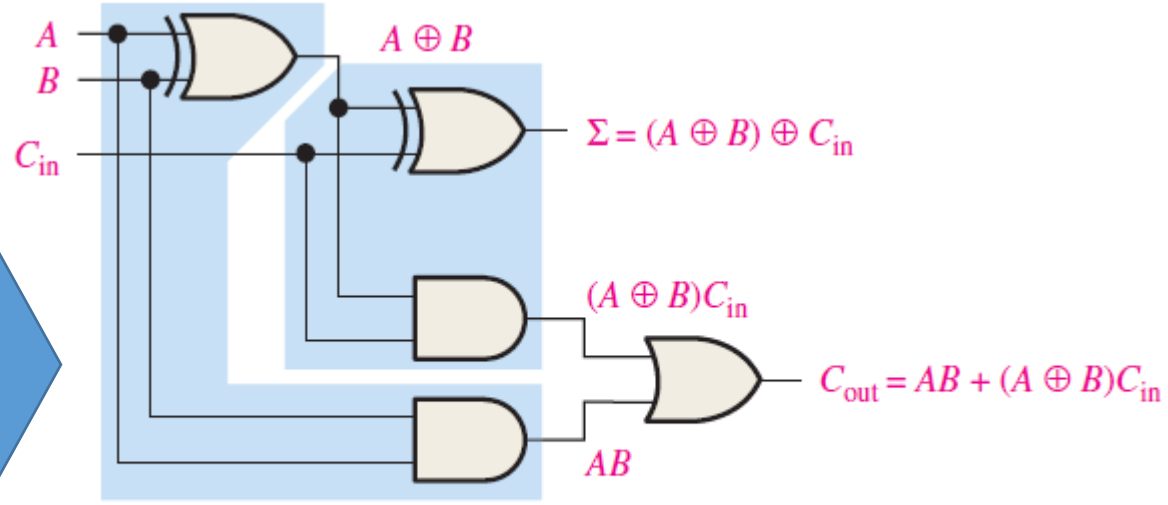
A and B = input variables (operands)

Full Adder

Full Adder Logic Symbol



Full Adder Logic Diagram



(b) Complete logic circuit for a full-adder (each half-adder is enclosed by a shaded area)

TABLE 6–2

Full-adder truth table.

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

C_{in} = input carry, sometimes designated as *CI*
C_{out} = output carry, sometimes designated as *CO*
Σ = sum
A and B = input variables (operands)

3-bit parallel adder

EXAMPLE 6–2

Determine the sum generated by the 3-bit parallel adder in Figure 6–8 and show the intermediate carries when the binary numbers 101 and 011 are being added.

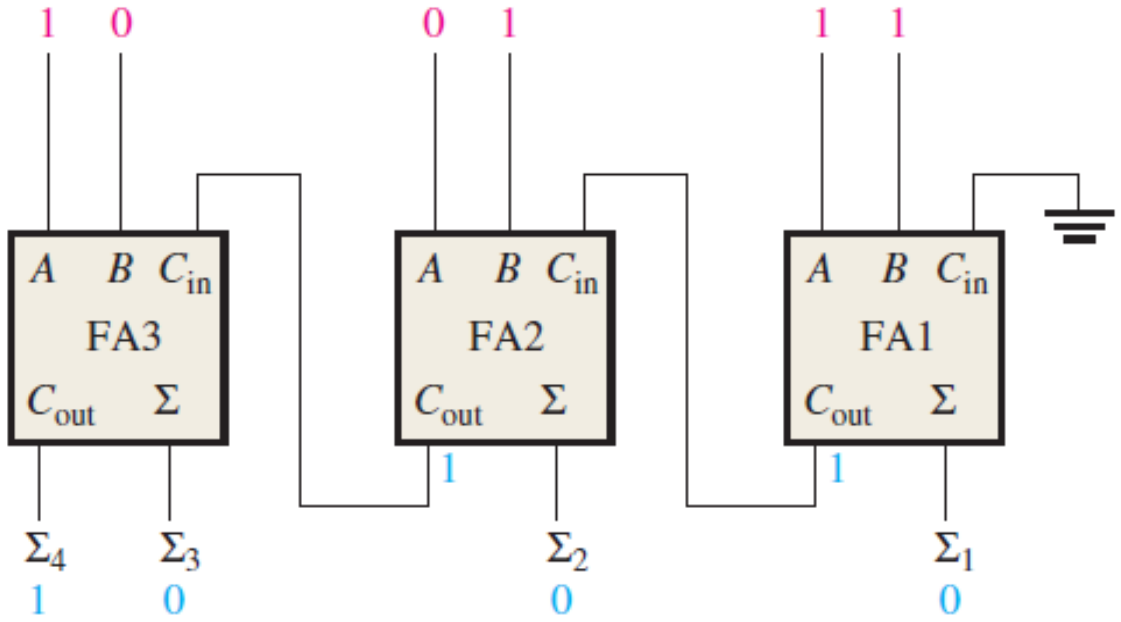
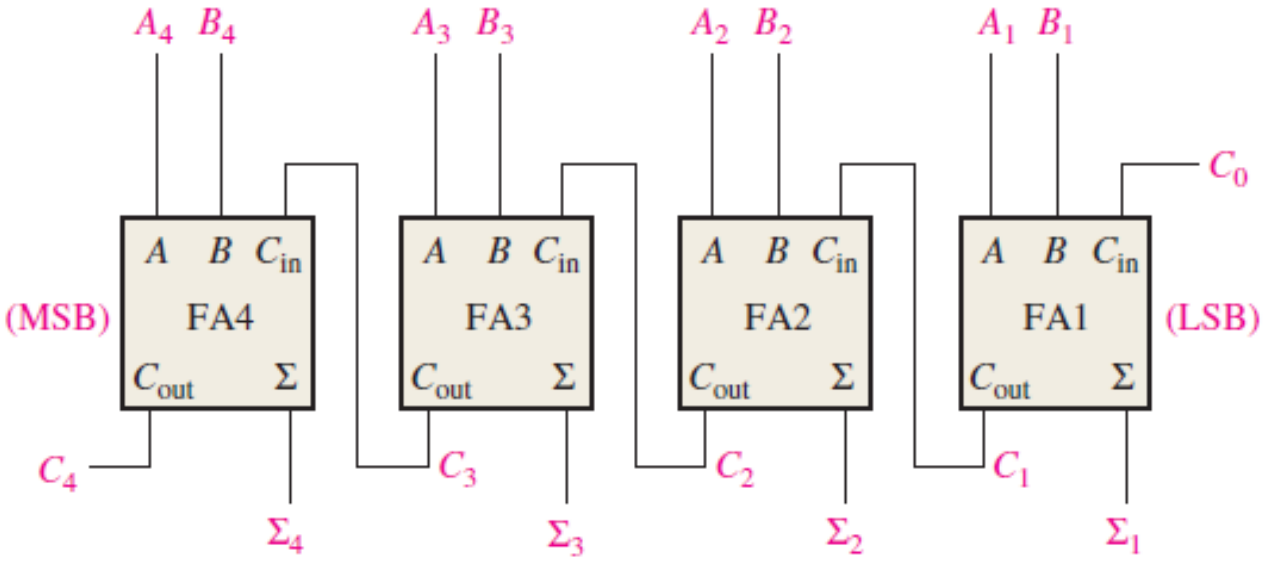
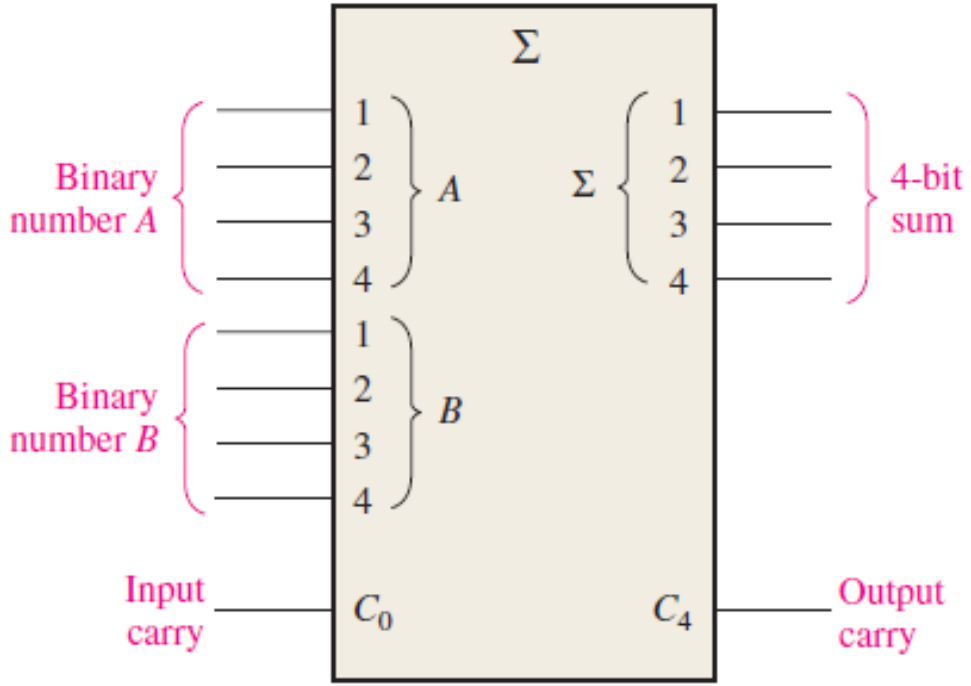


FIGURE 6–8

4-bit parallel adder



(a) Block diagram

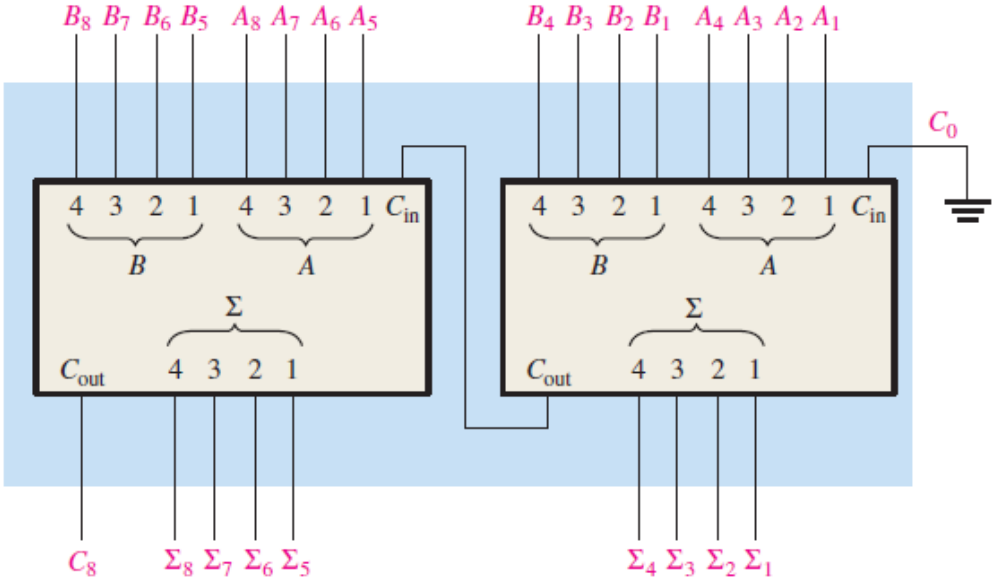


(b) Logic symbol

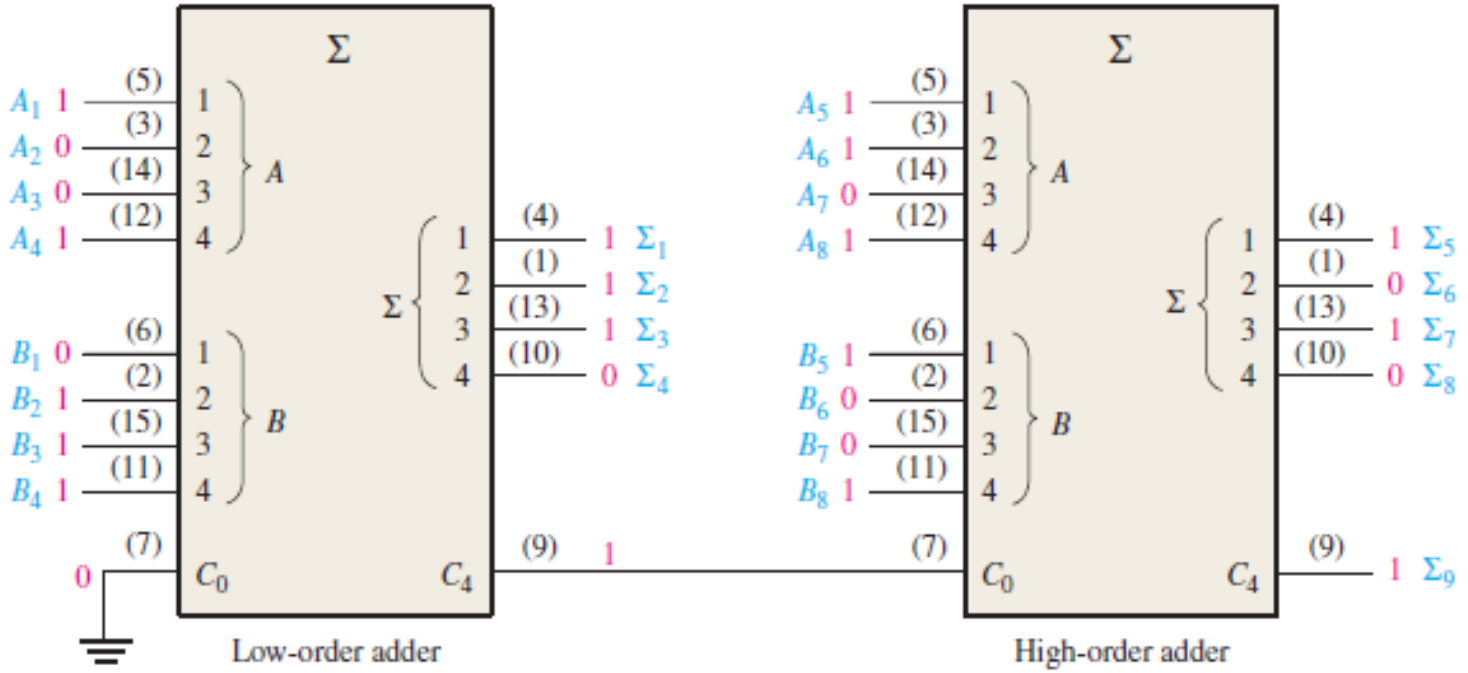
FIGURE 6–9 A 4-bit parallel adder.

Two 4-bit parallel adders consists 8-bit adder

$$A_8A_7A_6A_5A_4A_3A_2A_1 = 10111001 \quad \text{and} \quad B_8B_7B_6B_5B_4B_3B_2B_1 = 10011110$$



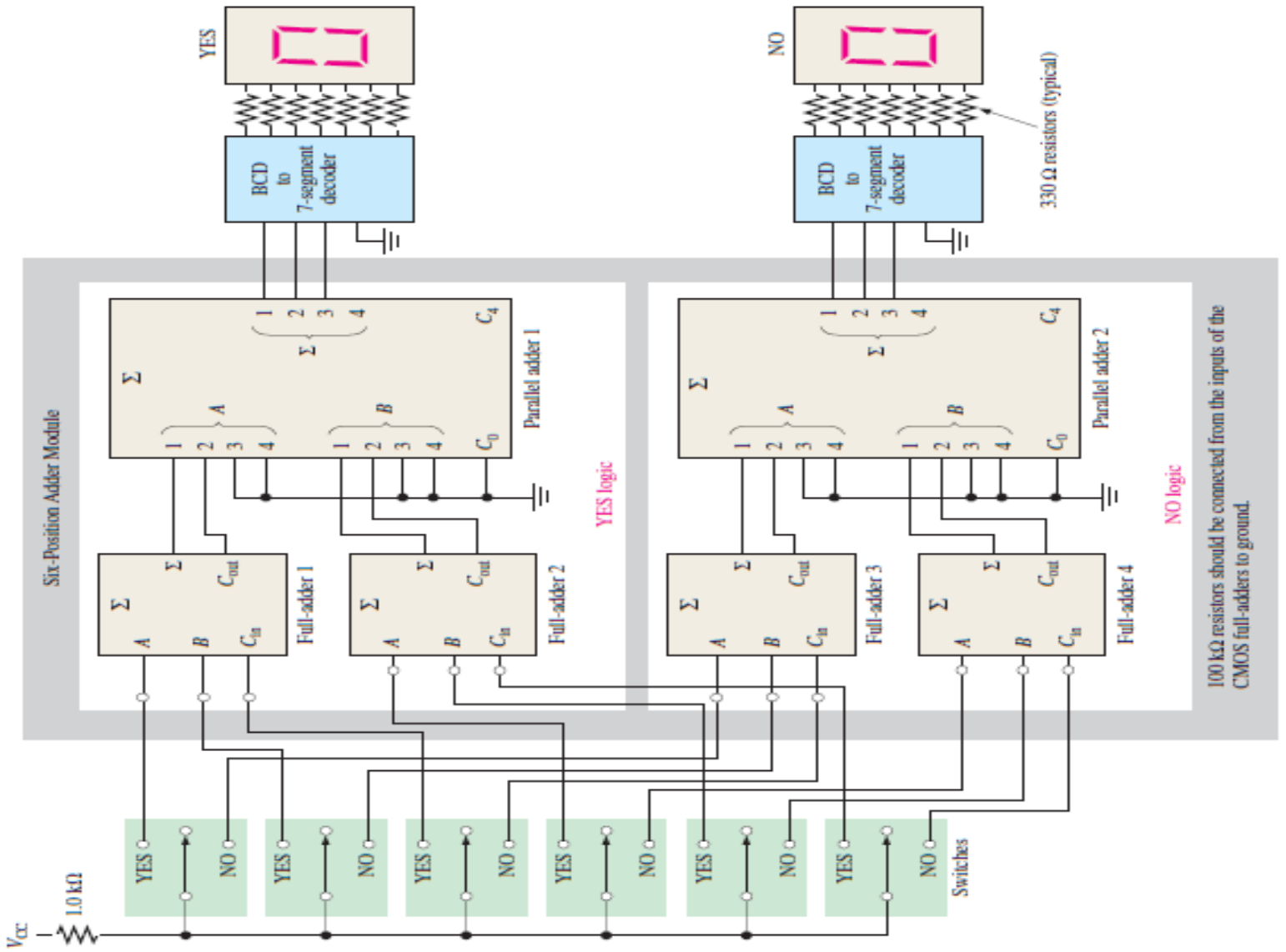
Cascading of two 4-bit adders to form an 8-bit adder.



$$\Sigma_9\Sigma_8\Sigma_7\Sigma_6\Sigma_5\Sigma_4\Sigma_3\Sigma_2\Sigma_1 = 101010111$$

Full adder and binary adder - application

A voting system using full adders and binary adders



Ripple carry adder

- A **ripple carry** adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder).
- The sum and the output carry of any stage cannot be produced until the input carry occurs; this causes a time delay in the addition process, as illustrated in Figure 6–14.
- The **carry propagation delay** for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the *A* and *B* inputs are already present.

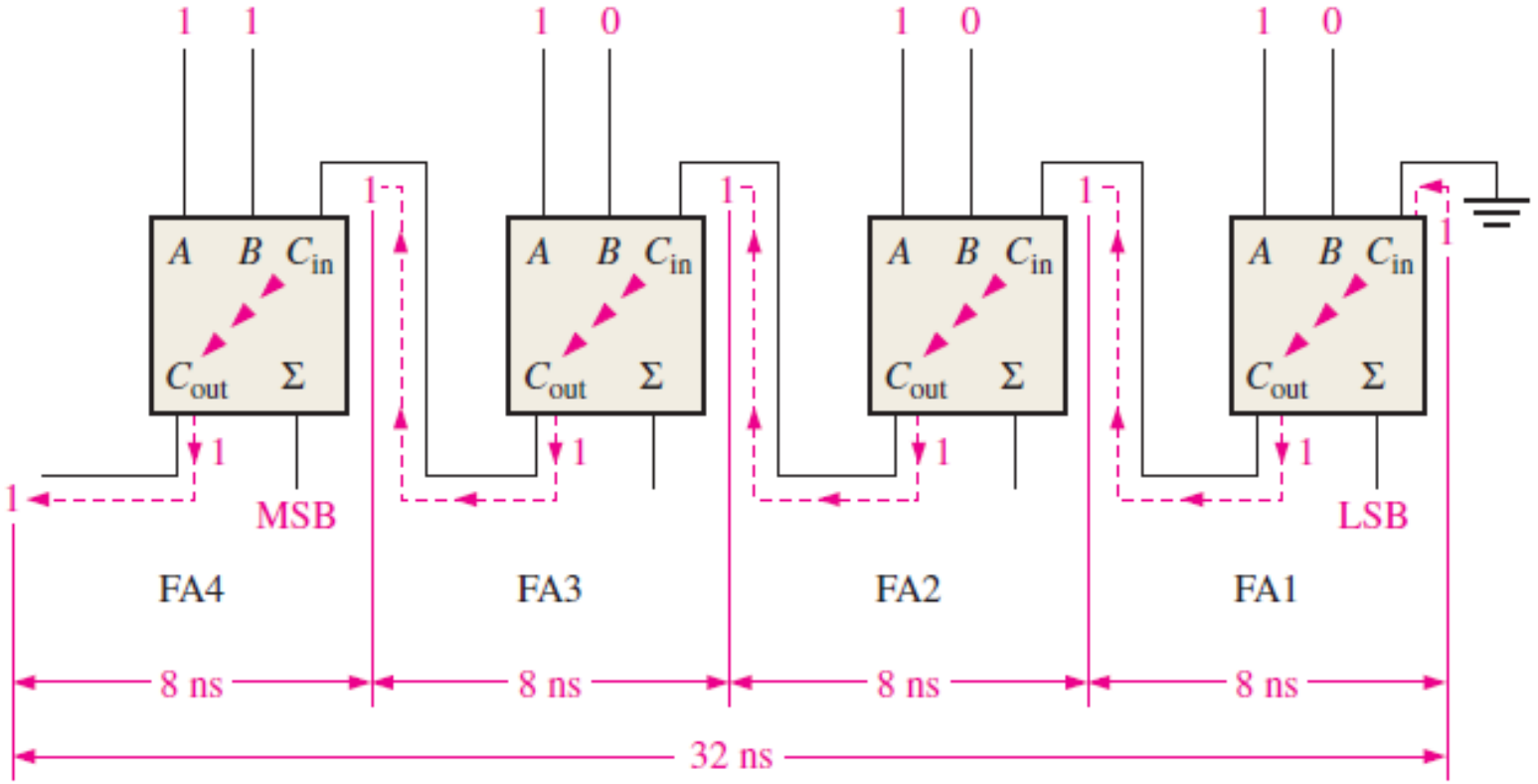


FIGURE 6–14 A 4-bit parallel ripple carry adder showing "worst-case" carry pro delay delays delays.

Subtractors – half and full

Half Subtractor

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

$$D = x'y + x\,y'$$

$$B = x'y$$

Full Subtractor

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = x'y'z + x'yz' + x\,y'z' + x\,yz$$

$$B = x'y + x'z + yz$$

Comparator

EXAMPLE 6–6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–22.

Solution

The number on the A inputs is 0110 and the number on the B inputs is 0011. The $A > B$ **output is HIGH and the other outputs are LOW.**

Related Problem

What are the comparator outputs when $A_3A_2A_1A_0 = 1001$ and $B_3B_2B_1B_0 = 1010$?

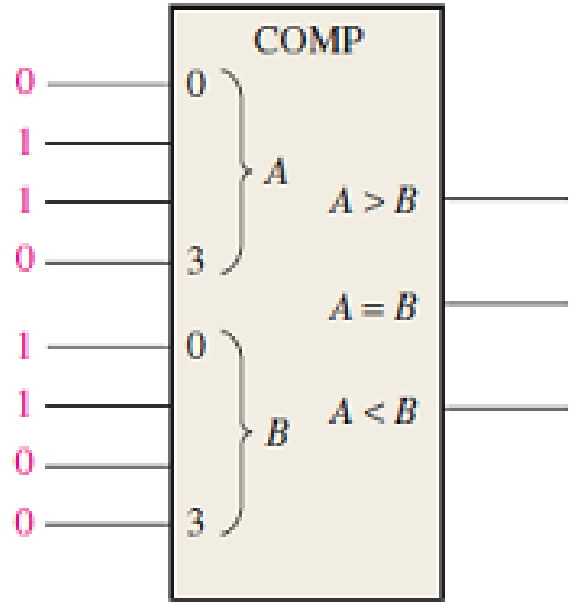
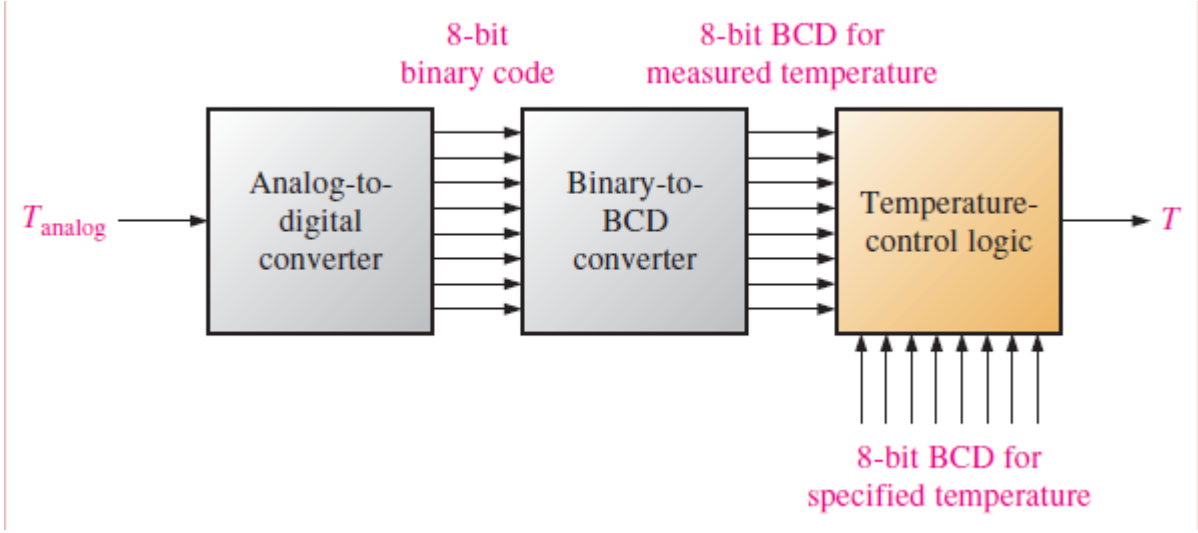
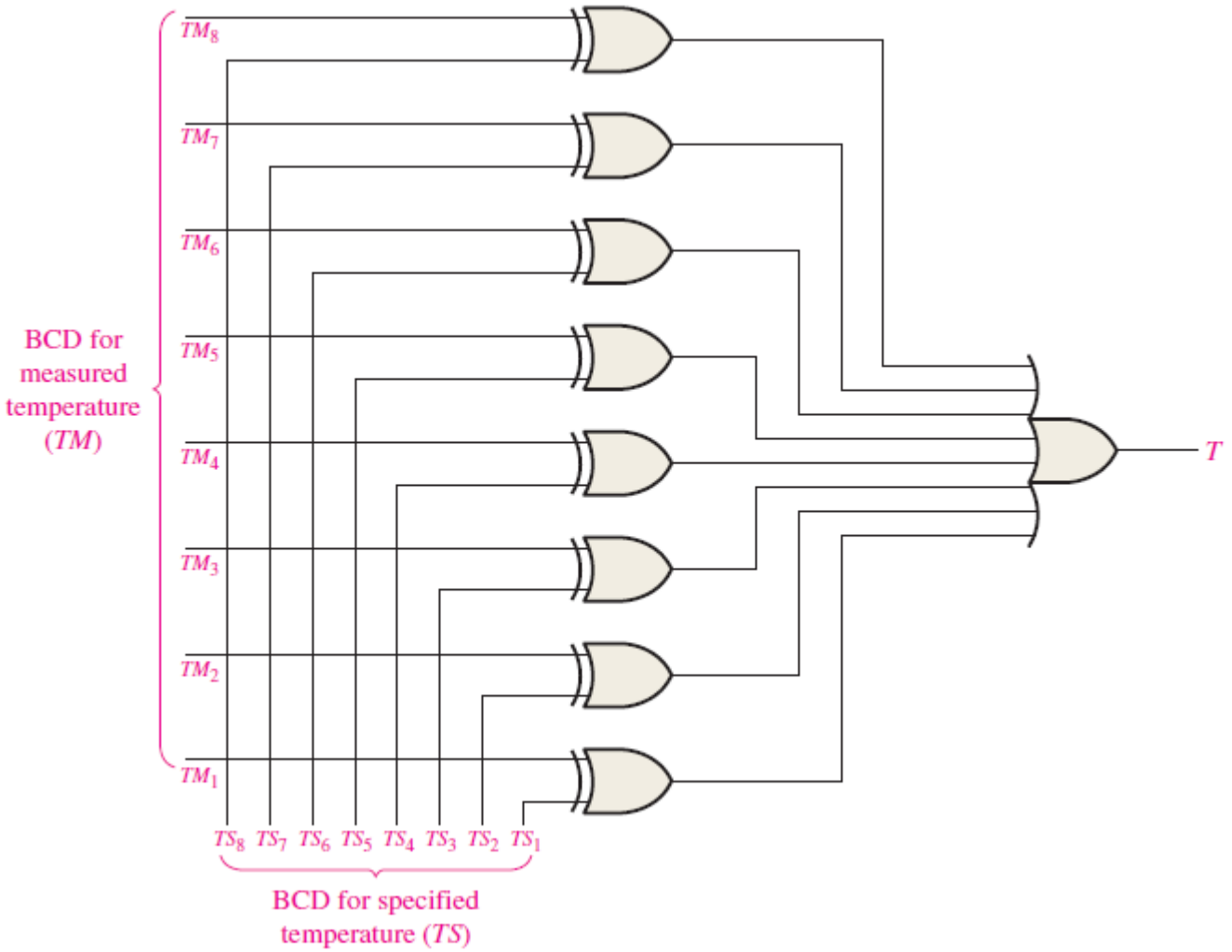


FIGURE 6–22

Comparator – application (temperature control)





When the measured temperature and the specified temperature are the same, the two BCD codes are equal and the *T* output is LOW (0). When the measured temperature falls below the specified value, there is a difference in the BCD codes and the *T* output is HIGH (1), which turns on the heater. The temperature control logic can be implemented with exclusive-OR gates, as shown in Figure.

Decoder: Basic binary decoder

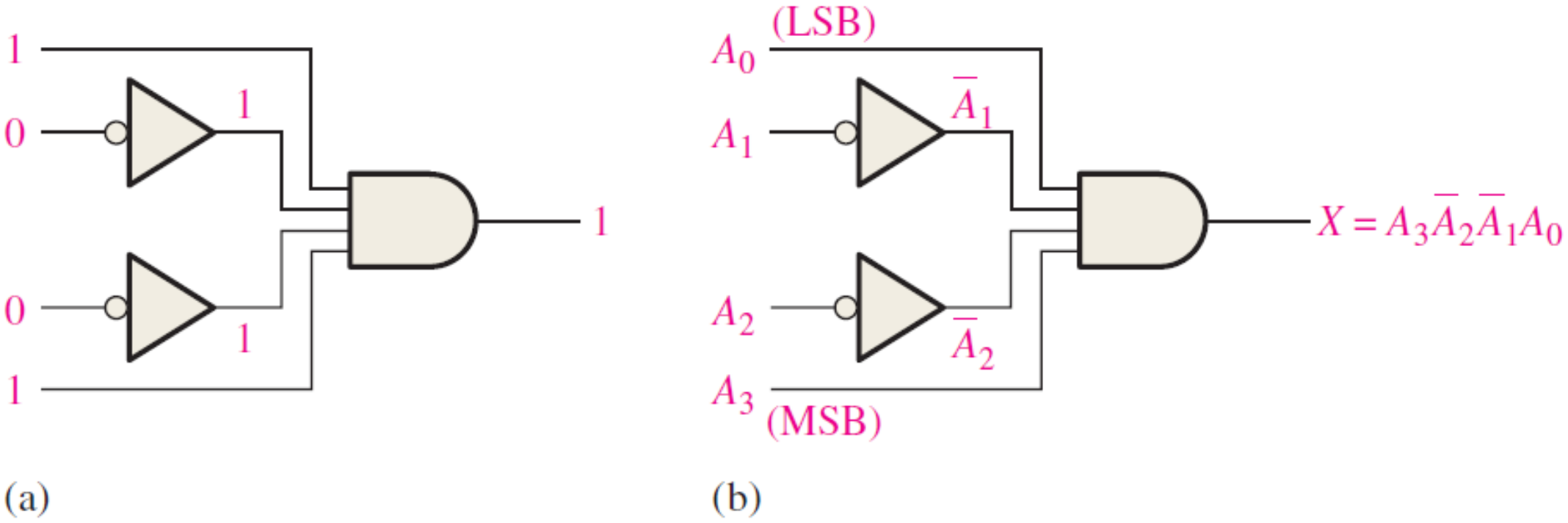


FIGURE 6–26 Decoding logic for the binary code 1001 with an active-HIGH output.

4-bit decoder (4-line-to-16-line or 1-of-16 decoder)

In order to decode all possible combinations of four bits, sixteen decoding gates are required ($2^4 = 16$). For any given code on the inputs, one of the sixteen outputs is activated. A list of the sixteen binary codes and their corresponding decoding functions is given in Table.

TABLE 6–4

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

Decimal Digit	Binary Inputs				Decoding Function	Outputs															
	A_3	A_2	A_1	A_0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

BCD-decimal decoder

- The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications. It is frequently referred as a *4-line-to-10-line decoder* or a *1-of-10 decoder.*
- A list of the ten BCD codes and their corresponding decoding functions is given in Table 6–5.
- Each of these decoding functions is implemented with NAND gates to provide active-LOW outputs. If an active-HIGH output is required, AND gates are used for decoding.

TABLE 6–5

BCD decoding functions.

Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

Encoder: Decimal to BCD encoder

Can be implemented by OR gates

$$A_3 = 8 + 9$$

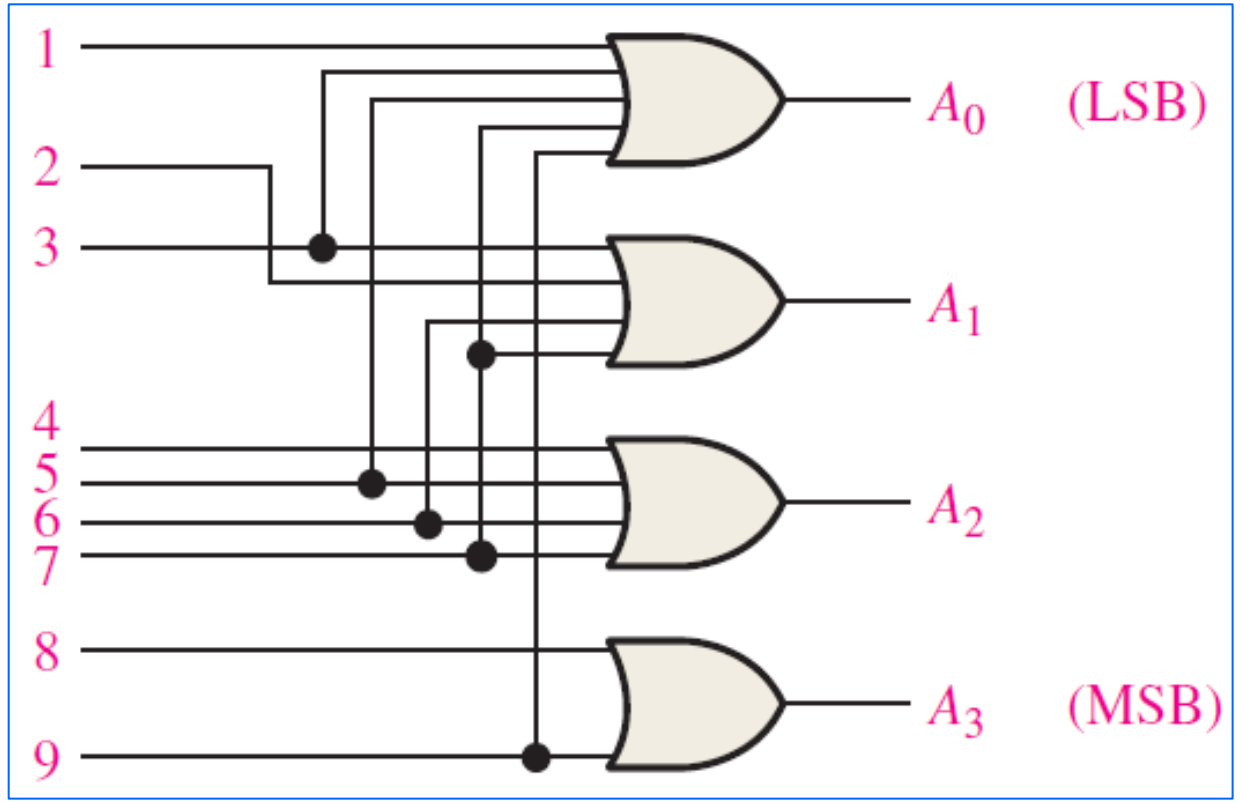
$$A_2 = 4 + 5 + 6 + 7$$

$$A_1 = 2 + 3 + 6 + 7$$

$$A_0 = 1 + 3 + 5 + 7 + 9$$

TABLE 6–6

Decimal Digit	BCD Code A_3	A_2	A_1	A_0
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



Code converter: BCD to binary conversion

The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

Let's examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary. For instance, you already know that the decimal number 87 can be expressed in BCD as

$$\underbrace{1000}_{8} \quad \underbrace{0111}_{7}$$

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1. Within each group, the binary weight of each bit is as follows:

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

BCD to binary conversion

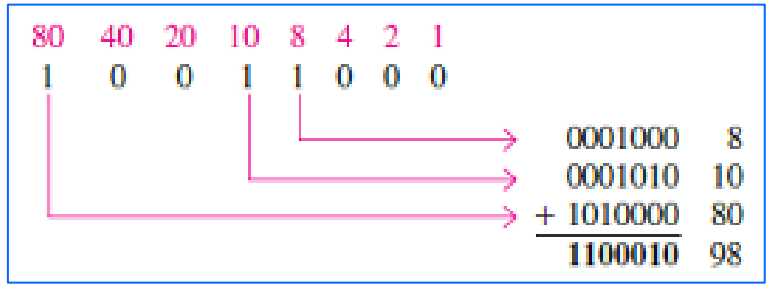
EXAMPLE 6–12

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Solution

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

80	40	20	10	8	4	2	1	
0	0	1	0	0	1	1	1	
							0000001	1
							0000010	2
							0000100	4
							+ 0010100	20
							0011011	Binary number for decimal 27



Gray codes

- The ***Gray code*** is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the *Gray code is that it exhibits only a single bit change from one code word to the next in sequence*.

- ***Gray code*** *is used in* in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence.

TABLE 2–6

Four-bit Gray code.

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

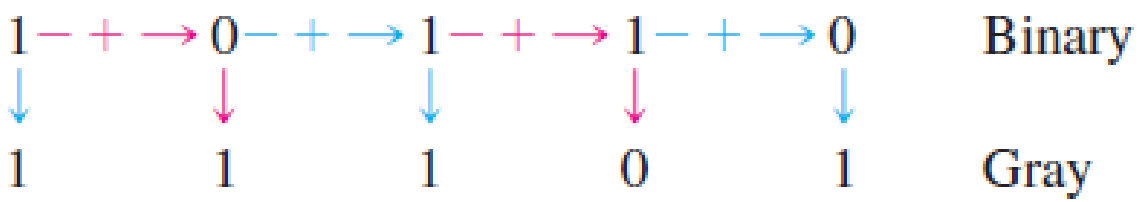
Binary to gray code conversion and vice versa

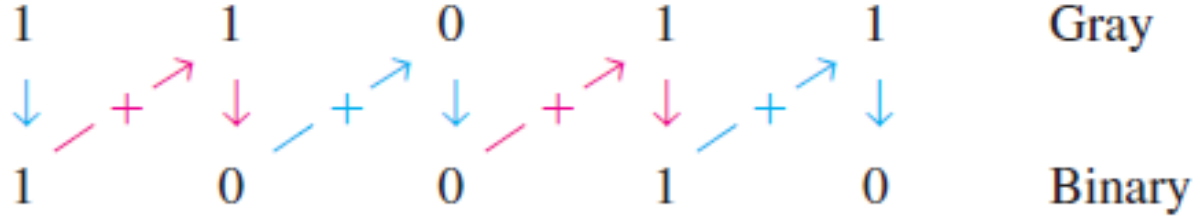
Binary-to-Gray Code Conversion

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

Gray-to-Binary Code Conversion

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.





Binary to gray code conversion and vice versa

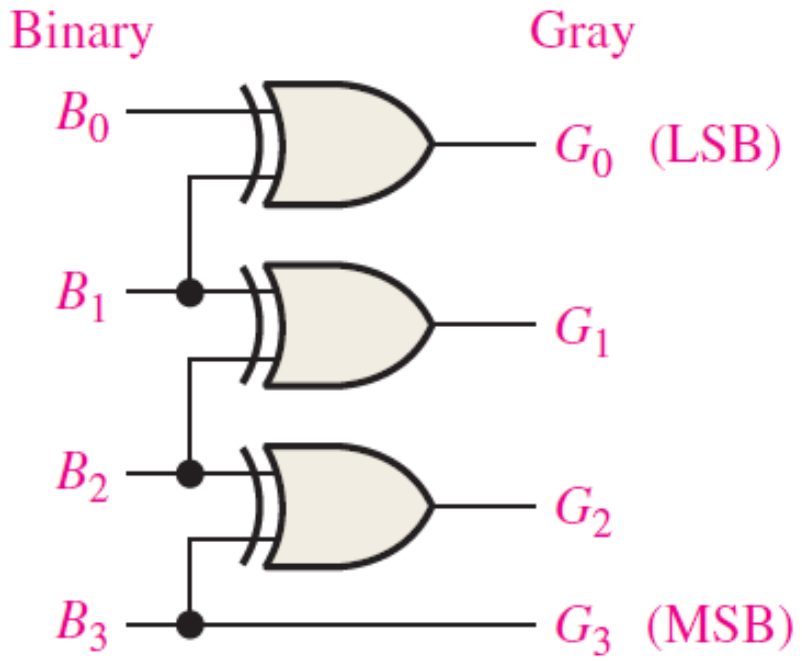


FIGURE 6–40 Four-bit binary-to-Gray conversion logic.

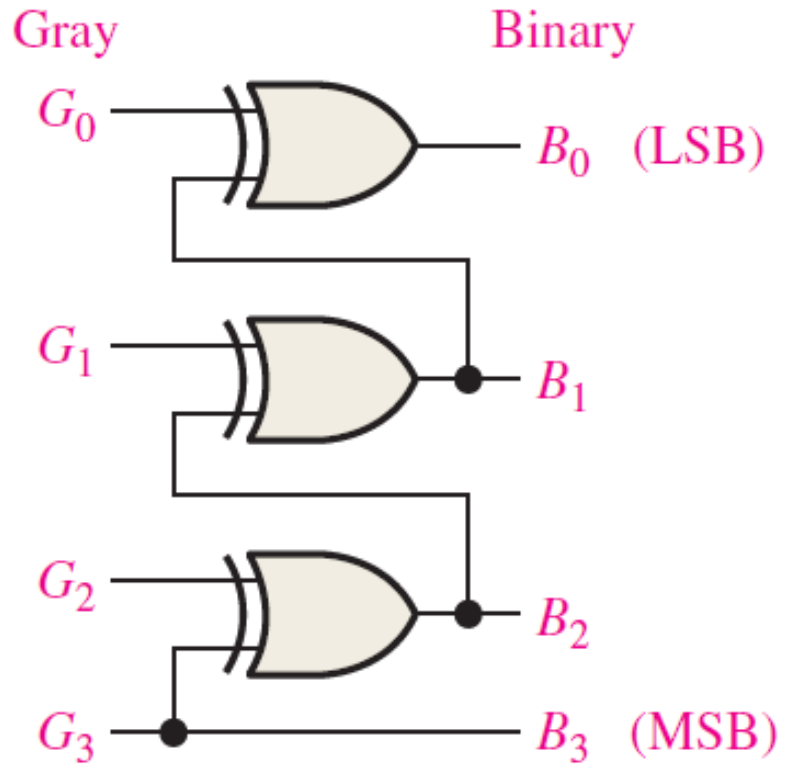


FIGURE 6–41 Four-bit Gray-to-binary conversion logic.

Binary to gray code conversion and vice versa

EXAMPLE 6–13

(a) Convert the binary number 0101 to Gray code with exclusive-OR gates.

(b) Convert the Gray code 1011 to binary with exclusive-OR gates.

Solution

(a) 0101_2 is 0111 Gray. See Figure 6–42(a).

(b) 1011 Gray is 1101_2. See Figure 6–42(b).

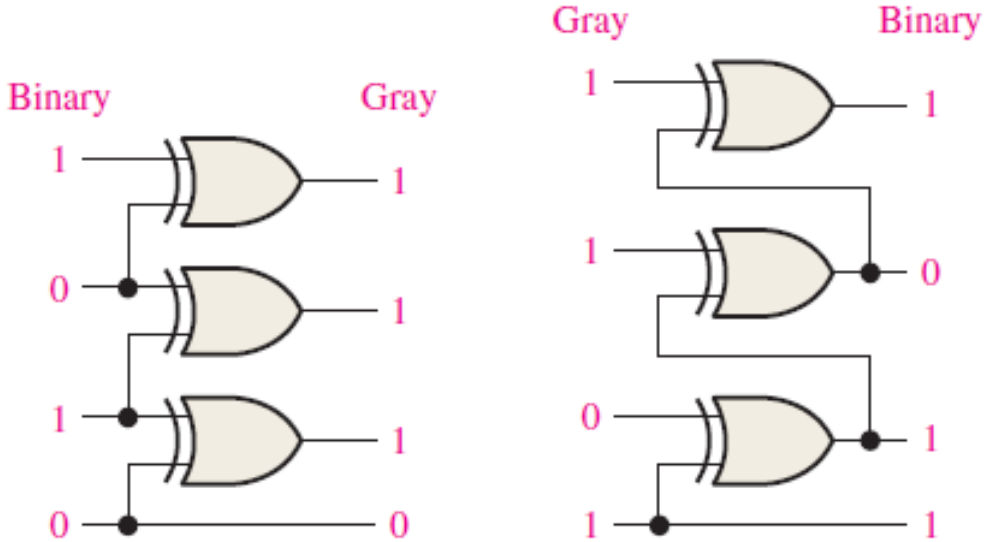
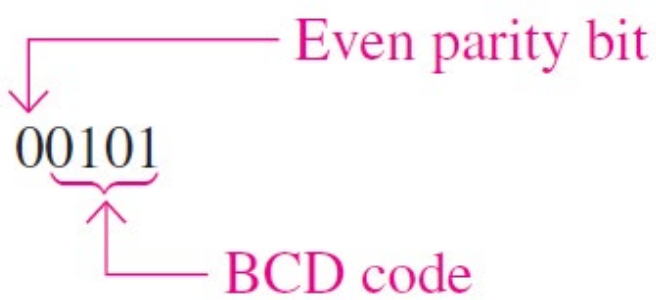


FIGURE 6–42 (a) (b)

Parity method for error detection

- Many systems use a parity bit as a means for bit **error detection**. Any group of bits contain either an even or an odd number of 1s. A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.
- A given system operates with even or odd **parity**, but not both. For instance, if a system operates with even parity, a check is made on each group of bits received to make sure the total number of 1s in that group is even. If there is an odd number of 1s, an error has occurred.



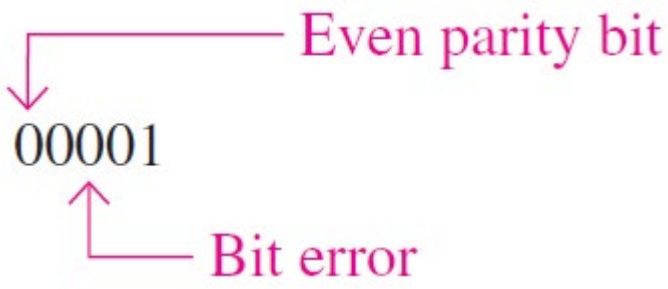


TABLE 2–8

The BCD code with parity bits.

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

Parity checker

EXAMPLE 2–39

Assign the proper even parity bit to the following code groups:

(a) 1010 **(b)** 111000 **(c)** 101101

(d) 1000111001001 **(e)** 101101011111

Solution

Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) **0**1010 **(b)** **1**111000 **(c)** **0**101101

(d) **0**1000111001001 **(e)** **1**101101011111

Multiplexer

- A **multiplexer (MUX)** is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.

TABLE 6–8

Data selection for a 1-of-4-multiplexer.

Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

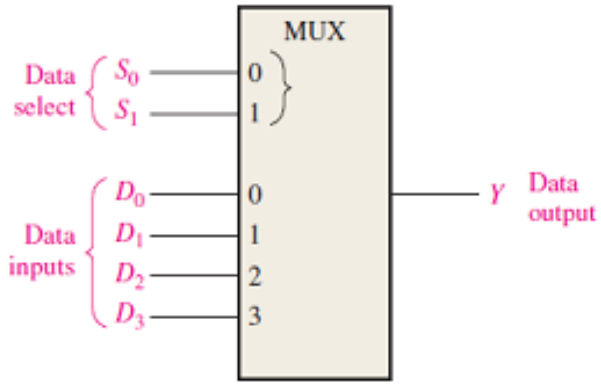


FIGURE 6–43 Logic symbol for a 1-of-4 data selector/multiplexer.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.
The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1S_0$.
The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1\overline{S}_0$.
The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

When these terms are ORed, the total expression for the data output is

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

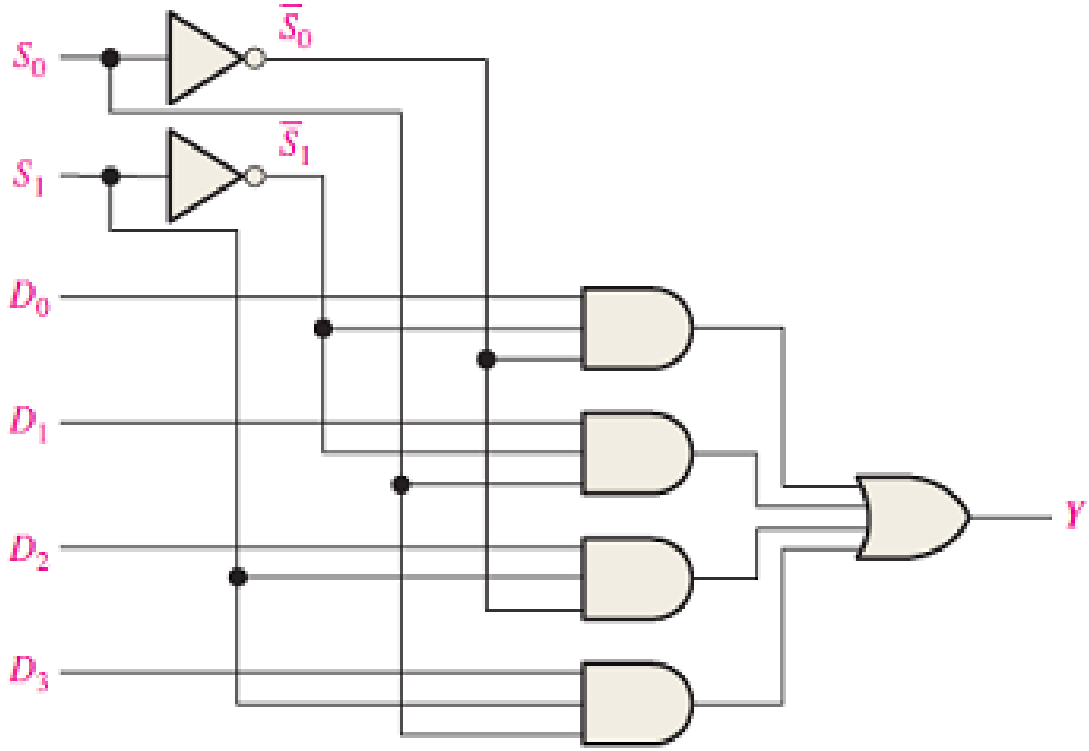


FIGURE 6–44 Logic diagram for a 4-input multiplexer.

Multiplexer application – A 7-segment display multiplexer

- Figure 6–49 shows a simplified method of multiplexing BCD numbers to a 7-segment display. In this example, 2-digit numbers are displayed on the 7-segment readout by the use of a single BCD-to-7-segment decoder.
- The basic operation is as follows. Two BCD digits (*A*3*A*2*A*1*A*0 and *B*3*B*2*B*1*B*0) are applied to the multiplexer inputs. A square wave is applied to the data-select line, and when it is LOW, the *A* bits (*A*3*A*2*A*1*A*0) are passed through to the inputs of the 74HC47 BCD-to-7-segment decoder. The LOW on the data-select also puts a LOW on the *A*1 input of the 74HC139 2-line-to-4-line decoder, thus activating its 0 output and enabling the *A*-digit display by effectively connecting its common terminal to ground. The *A* digit is now *on* and the *B* digit is *off.*
- When the data-select line goes HIGH, the *B* bits (*B*3*B*2*B*1*B*0) are passed through to the inputs of the BCD-to-7-segment decoder. Also, the 74HC139 decoder's 1 output is activated, thus enabling the *B*-digit display. The *B* digit is now *on* and the *A* digit is *off.*

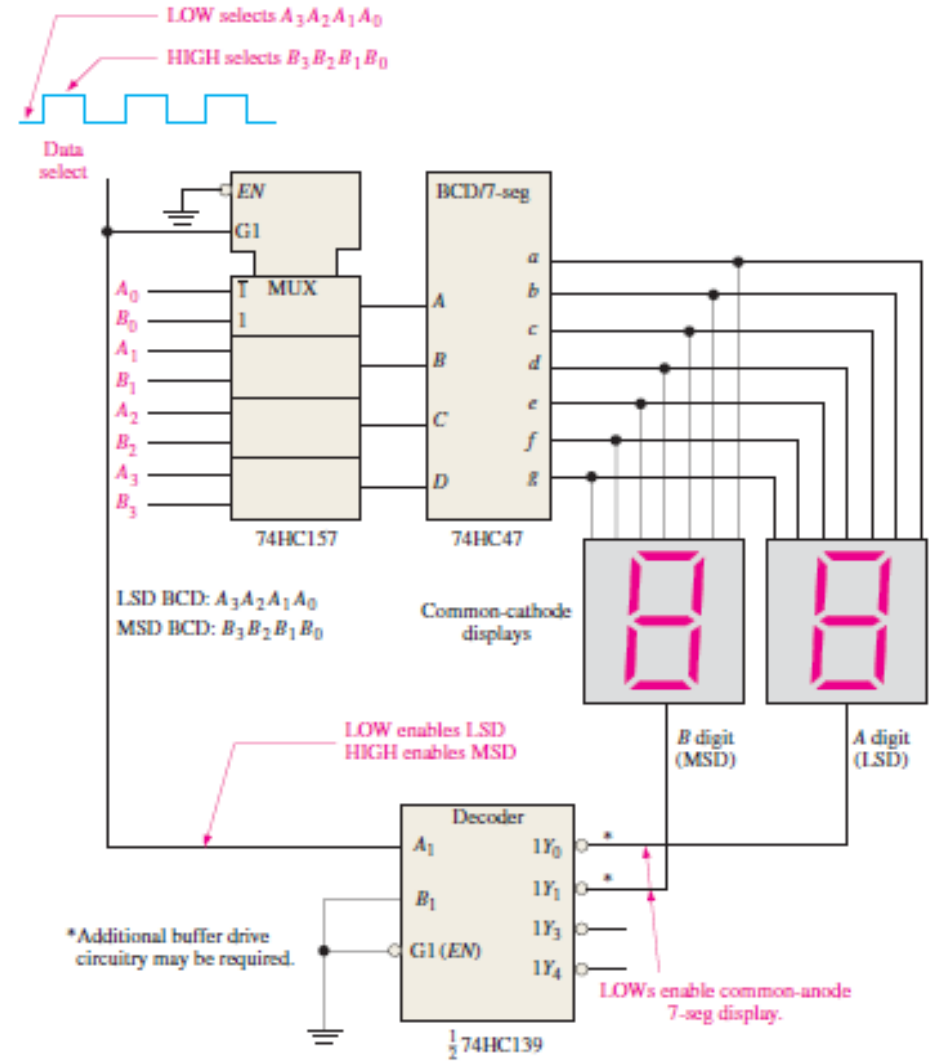


FIGURE 6–49 Simplified 7-segment display multiplexing logic.

Demultiplexer

- A **demultiplexer (DEMUX)** basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. As you will learn, decoders can also be used as demultiplexers.

- Figure 6–52 shows a 1-line-to-4-line demultiplexer (DEMUX) circuit. The data-input line goes to all of the AND gates. The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.

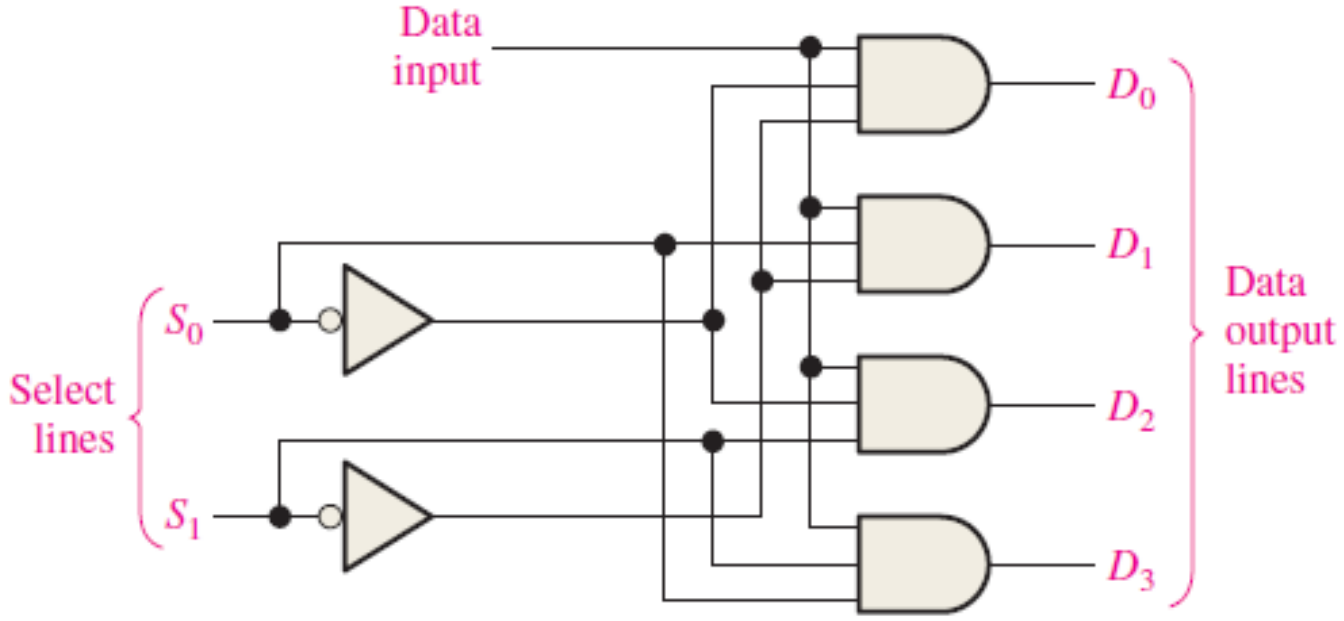


FIGURE 6–52 A 1-line-to-4-line demultiplexer.

Demultiplexer

PROBLEM. The serial data-input waveform (Data in) and data-select inputs (*S*0 and *S*1) are shown in Figure 6–53. Determine the data-output waveforms on *D*0 through *D*3 for the demultiplexer in Figure 6–52.

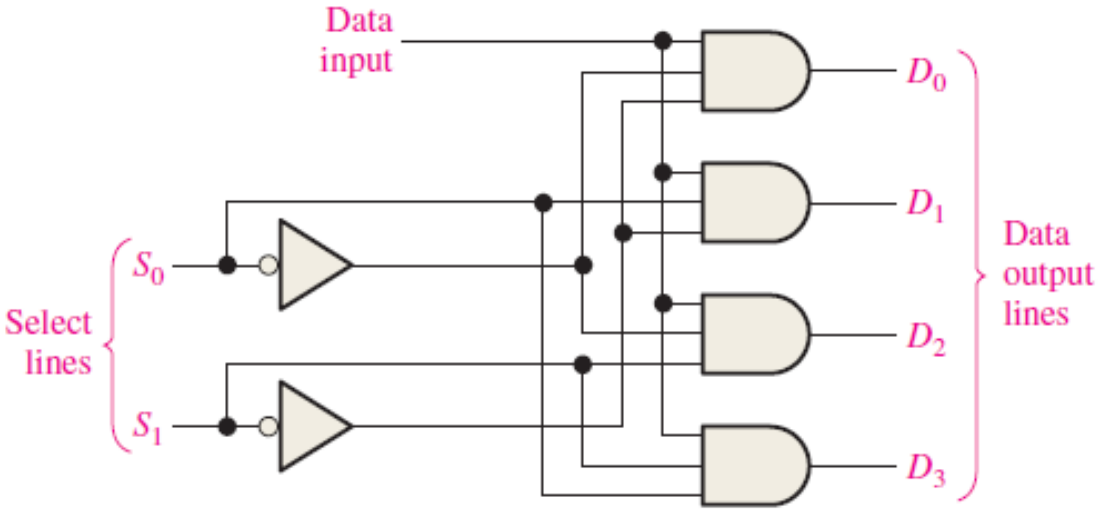


FIGURE 6–52 A 1-line-to-4-line demultiplexer.

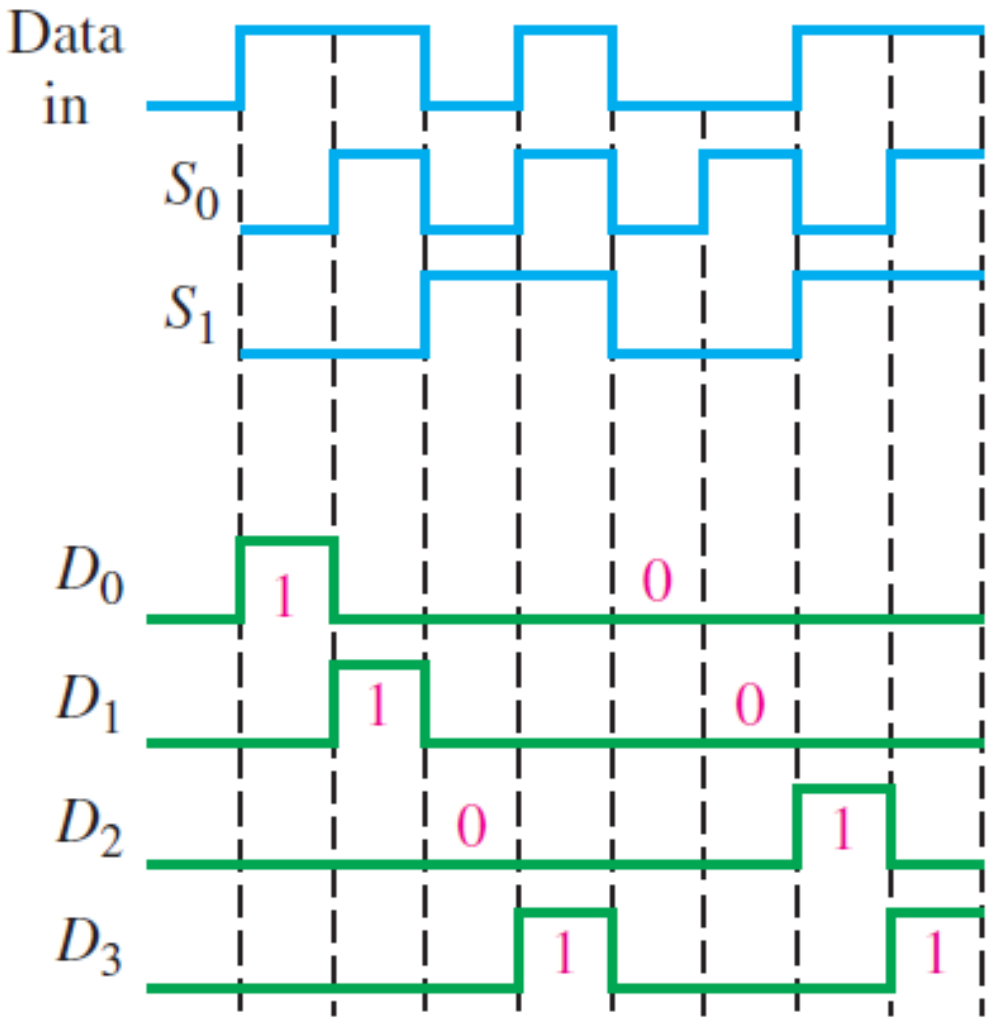


FIGURE 6–53

References

1. ***Digital Fundamentals*** by Thomas Floyd, Pearson International Edition, 11th Edition, Chapter 6, Page 313-386.

Next class

Latches, Flip-Flops and Timers