Fundamentals of Digital Logic and Microcomputer Design. M. Rafiquzzaman

4

COMBINATIONAL LOGIC DESIGN

This chapter describes analysis and design of combinational logic circuits. Topics include BCD to seven-segment code converters, adders, subtractors, comparators, decoders, and multiplexers. An overview of ROMs, PLDs and hardware description languages is also included.

4.1 Basic Concepts

Digital logic circuits can be classified into two types: combinational and sequential. A combinational circuit is designed using logic gates in which application of inputs generates the outputs at any time. An example of a combinational circuit is an adder, which produces the result of addition as output upon application of the two numbers to be added as inputs.

A sequential circuit, on the other hand, is designed using logic gates and memory elements known as "flip-flops." Note that the flip-flop is a one-bit memory. A sequential circuit generates the circuit outputs based on the present inputs and the outputs (states) of the memory elements. The sequential circuit is basically a combinational circuit with memory. Note that a combinational circuit does not require any memory (flip-flops), whereas sequential circuits require flip-flops to remember the present states. A counter is a typical example of a sequential circuit. To illustrate the sequential circuit, suppose that it is desired to count in the sequence 0, 1, 2, 3, 0, 1,... and repeat. In binary, the sequence is 00, 01, 10, 11, 00, 01, ..., and so on. This means that a two-bit memory using two flip-flops is required for storing the two bits of the counter because each flip-flop stores one bit. Let us call these flip-flops with outputs A and B. Note that initially $A = 0$ and $B = 0$. The flip-flop changes outputs upon application of a clock pulse. With appropriate inputs to the flip-flops and then applying the clock pulse, the flip-flops change the states (outputs) to $A = 0, B = 1$. Thus, the count to 1 can be obtained. The flip-flops store (remember) this count. Upon application of appropriate inputs along with the clock, the flip-flops will change the status to $A = 1, B = 0$; thus, the count to 2 is obtained. The flip-flops remember (store) this count at the outputs until a common clock pulse is applied to the flip-flops. The inputs to the flip-flops are manipulated by a combinational circuit based on A and B as inputs. For

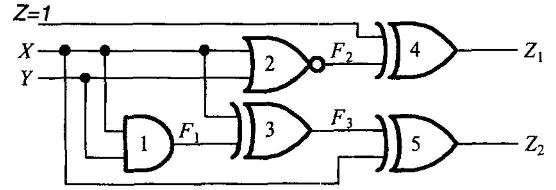


FIGURE 4.1 Analysis of a combinational logic circuit

example, consider $A = 1$, $B = 0$. The inputs to the flip-flops are determined in such a way that the flip-flops change the states at the clock pulse to $A = 1$, $B = 1$; thus, the count to 3 is obtained. The process is repeated.

4.2 Analysis of a Combinational Logic Circuit

A combinational logic circuit can be analyzed by (i) first, identifying the number of inputs and outputs, (ii) expressing the output functions in terms of the inputs, and (iii) determining the truth table for the logic diagram. As an example, consider the combinational circuit in Figure 4.1 There are three inputs (X, Y, and Z) and two outputs (Z_1 and Z_2) in the circuit.

Let us now express the outputs F_1 and F_2 in terms of the inputs. The output F_1 of the AND gate #1 is $F_1 = XY$. The output F_2 of NOR gate #2 can be expressed as $F_2 = \overline{X + Y}$. The output of the XOR gate #3 is

$$F_3 = X \oplus F_1 = (X \oplus XY)$$

Because one of the inputs of the XOR gate #4 is 1, its output is inverted. Therefore,

$$Z_1 = \overline{F_2} = X + Y.$$

Finally,

$$Z_2 = X \oplus F_3 = X \oplus (X \oplus XY)$$

Therefore,

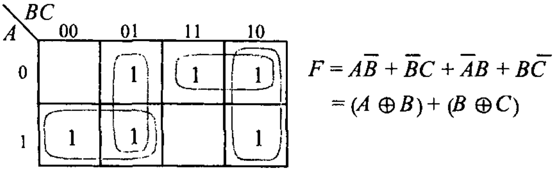
$$\begin{aligned} Z_2 &= X \oplus (X \bullet \overline{XY} + \overline{X} \bullet XY) \\ &= X \oplus (X \bullet (\overline{X} + \overline{Y})) \\ &= X \oplus (X\overline{Y}) \\ &= X(\overline{X\overline{Y}}) + \overline{X}(X\overline{Y}) \\ &= X(\overline{X} + Y) \\ &= XY \end{aligned}$$

TABLE 4.1 Truth Table for Figure 4.1 with Input, $Z = 1$

Inputs		Outputs	
X	Y	Z_1	Z_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

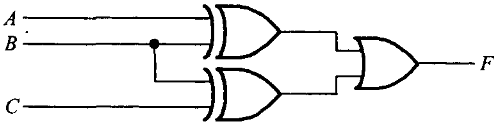
TABLE 4.2 Truth Table for F

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



$$F = A\bar{B} + \bar{B}C + \bar{A}B + B\bar{C}$$
$$= (A \oplus B) + (B \oplus C)$$

(a) K-map for F



(b) Logic Diagram for the output, F

FIGURE 4.2 K-map and the logic diagram for F

Another way of determinig Z_2 is provided below:

$$Z_2 = X \oplus F_3 = X \oplus (X \oplus XY) = X \oplus X \oplus XY = 0 \oplus (X\,Y) = XY$$

The Z_0 truth table shown in Table 4.1 can be obtained by using the logic equations for Z_1 and Z_2.

4.3 Design of a Combinational Circuit

A combinational circuit can be designed using three steps as follows:

1) Determine the inputs and the outputs from problem definition and then derive the truth table.
2) Use K-maps to minimize the number of inputs (literals) in order to express the outputs. This reduces the number of gates and thus the implementation cost.
3) Draw the logic diagram

In order to illustrate the design procedure, consider the following example. Suppose that it is desired to design a combinational circuit with three inputs (A, B, and C) and one output F. The output F is one if A, B, and C are not equal ($A \neq B \neq C$); $F = 0$ otherwise.First, the number of inputs and outputs are identified. There are three inputs (A, B, and C) and one output, F. Next the truth table is obtained as shown in Table 4.2. F in the truth table of Table 4.2 is simplified using a K-map and implemented as shown in Figure 4.2. Note that this is one of the solutions. There are more than one implementation for this problem.

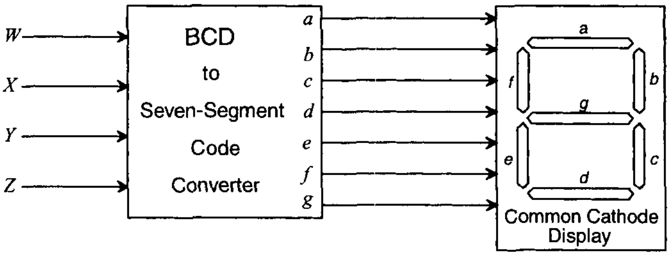


FIGURE 4.3 BCD to seven-segment code converter

4.4 Multiple-Output Combinational Circuits

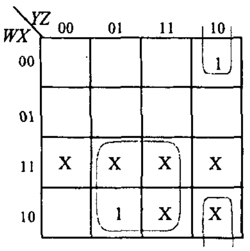
A combinational circuit may have more than one output. In such a situation, each output must be expressed as a function of the inputs. A digital circuit called the "code converter" is an example of multiple-output circuits. A code converter transforms information from one binary code to another. As an example, consider the BCD to seven-segment code converter shown in Figure 4.3. The code converter in the figure can be designed to translate the BCD inputs (*W*, *X*, *Y*, and *Z*) to seven-segment code for displaying decimal digits. The inputs *W*, *X*, *Y*, and *Z* can be entered into the code converter via four switches as was discussed in Chapter 1. A combinational circuit can be designed for the code converter that will translate each digit entered using four bits into seven output bits (one bit for each segment) of the display.

In this case, the code converter has four inputs and seven outputs. This code converter is commonly known as a "BCD to seven-segment decoder." With four bits (W, X, Y, and Z), there are sixteen combinations (0000 through 1111) of 1's and 0's. BCD allows only 10 (0000 through 1001) of these 16 combinations, so the invalid numbers (1010 through 1111) will never occur for BCD and can be considered as don't cares in K-maps because it does not matter what the seven outputs (a through g) are for these invalid combinations.

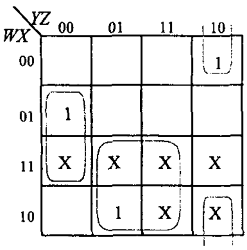
The 7447 (TTL) is a commercially available BCD to 7-segment decoder/driver chip. It is designed for driving a common-anode display. A LOW output will light a segment while a HIGH output will turn it OFF. For normal operation, the LT (Lamp test) and BI/RBO (Blanking Input / Ripple Blanking Input) must be open or conntected to HIGH. The 7448 chip, on the other hand, is designed for driving a common-cathode display.

TABLE 4.3 Truth Table for Converting Decimal Digits (Since common-cathode, a 1 will turn a segment ON and a 0 will turn it OFF)

Decimal Digit to be Displayed	BCD Input Bits				Seven-Segment Output Bits						
	W	*X*	*Y*	*Z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
2	0	0	1	0	1	1	0	1	1	0	1
4	0	1	0	0	0	1	1	0	0	1	1
9	1	0	0	1	1	1	1	0	0	1	1



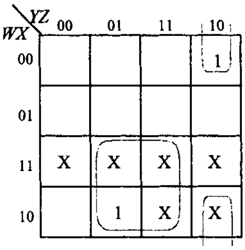
i) K-map for *a*: $a = WZ + \bar{X}Y\bar{Z}$



ii) K-map for b : $b = X\bar{Y}\bar{Z} + WZ + \bar{X}Y\bar{Z}$

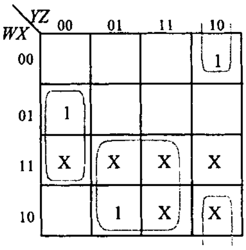
$$= \bar{Z}(X\bar{Y} + \bar{X}Y) + WZ$$

$$= \bar{Z}(X \oplus Y) + WZ$$



WX \ YZ	00	01	11	10
00				1
01				
11	X	X	X	X
10		1	X	X

i) K-map for *a*: $a = WZ + \bar{X}Y\bar{Z}$

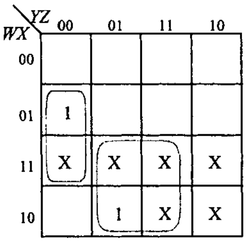


WX \ YZ	00	01	11	10
00				1
01	1			
11	X	X	X	X
10		1	X	X

ii) K-map for b : $b = X\bar{Y}\bar{Z} + WZ + \bar{X}Y\bar{Z}$

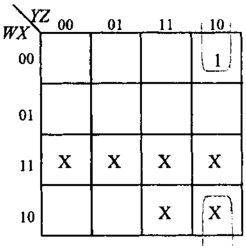
$= \bar{Z}(X\bar{Y} + \bar{X}Y) + WZ$

$= \bar{Z}(X \oplus Y) + WZ$



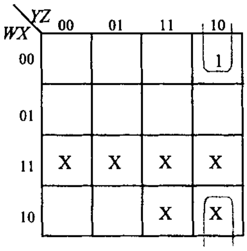
WX \ YZ	00	01	11	10
00				
01	1			
11	X	X	X	X
10		1	X	X

iii) K-map for *c*: $c = X\bar{Y}\bar{Z} + WZ$



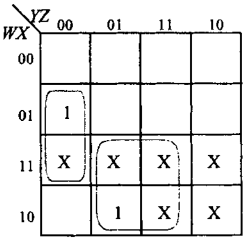
WX \ YZ	00	01	11	10
00				1
01				
11	X	X	X	X
10			X	X

iv) K-map for *d*: $d = \bar{X}Y\bar{Z}$



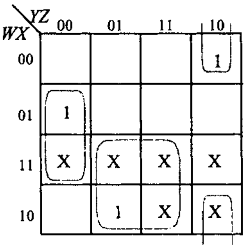
WX \ YZ	00	01	11	10
00				1
01				
11	X	X	X	X
10			X	X

v) K-map for *e*: $e = \bar{X}Y\bar{Z}$



WX \ YZ	00	01	11	10
00				
01	1			
11	X	X	X	X
10		1	X	X

vi) K-map for *f*: $f = X\bar{Y}\bar{Z} + WZ$



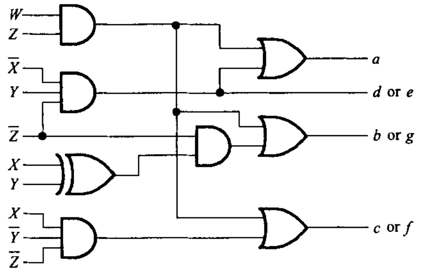
WX \ YZ	00	01	11	10
00				1
01	1			
11	X	X	X	X
10		1	X	X

$g = X\bar{Y}\bar{Z} + WZ + \bar{X}Y\bar{Z}$

$= \bar{Z}(X\bar{Y} + \bar{X}Y) + WZ$

$= \bar{Z}(X \oplus Y) + WZ$

vii) K-map for *g*



viii) Logic diagram assuming both true and complemented values of the inputs are available.

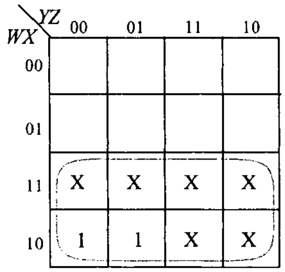
FIGURE 4.4 BCD to seven-segment decoder for decimal digits 2, 4, and 9

To illustrate the design of a BCD to seven-segment decoder, consider designing a code converter for displaying the decimal digits 2, 4, and 9, using the diagram shown in Figure 4.3. First, it is obvious that the BCD to seven-segment decoder has four inputs and seven outputs. Table 4.3 shows the truth table.

For the valid BCD digits that are not displayed (0, 1, 3, 5, 6, 7, 8) in this example, the combinational circuit for the code converter will generate 0's for the seven output bits (*a* through *g*). However, these seven bits will be don't-cares in the K-map for the invalid BCD digits 10 through 15. Figure 4.4 shows the K-maps and the logic diagram.

TABLE 4.4 Truth Table for Example 4.1

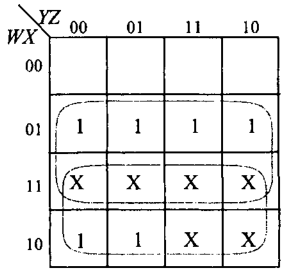
Decimal Digit	Input BCD Code				Output Gray Code			
	W	X	Y	Z	f_3	f_2	f_1	f_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1



WX \ YZ	00	01	11	10
00				
01				
11	X	X	X	X
10	1	1	X	X

a) K-map for f_3

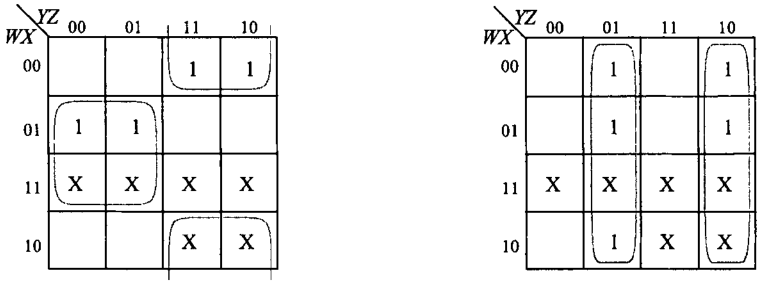
$f_3 = W$



WX \ YZ	00	01	11	10
00				
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

b) K-map for f_2

$f_2 = W + X$

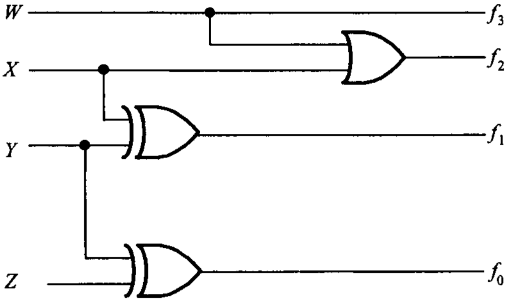


c) K-map for f_1

$$f_1 = X\bar{Y} + \bar{X}Y = X \oplus Y$$

d) K-map for f_0

$$f_0 = \bar{Y}Z + Y\bar{Z} = Y \oplus Z$$



e) Logic diagram for Example 4.1

FIGURE 4.5 K-maps and Logic Circuit for Example 4.1

Example 4.1

Design a digital circuit that will convert the BCD codes for the decimal digits (0 through 9) to their Gray codes.

Solution

Because both Gray code and BCD code are represented by four bits for each decimal digit, there are four inputs and four outputs. Table 4.4 shows the truth table. Note that 4-bit binary

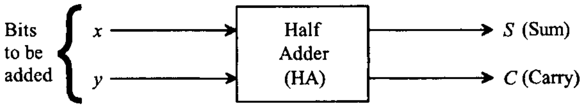


FIGURE 4.6 Block Diagram of a Half-Adder

TABLE 4.5 Truth Table of the Half-Adder

Inputs		Outputs		Decimal Value
x	y	C	S	
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	2

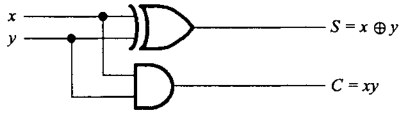


FIGURE 4.7 Logic diagram of the half-adder

combination will provide 16 (2^4) combinations of 1's and 0's. Because only ten of these combinations (0000 through 1001) are allowed in BCD, the invalid combinations 1010 through 1111 can never occur in BCD. Therefore, these six binary inputs are considered as don't cares. This means that it does not matter what binary values are assumed by $f_3 f_2 f_1 f_0$ for $WXYZ$ = 1010 through 1111. Figure 4.5 shows the K-maps and the logic circuit.

4.5 Typical Combinational Circuits

This section describes typical combinational circuits. Topics include binary adders, subtractors, comparators, decoders, encoders, multiplexers, and demultiplexers. These digital components are implemented in MSI chips.

4.5.1 Binary / BCD Adders and Binary Subtractors

When two bits x and y are added, a sum and a carry are generated. A combinational circuit that adds two bits is called a "half-adder." Figure 4.6 shows a block diagram of the half-adder. Table 4.5 shows the truth table of the half-adder. From Table 4.5, $S = \bar{x}y + x\bar{y} = x \oplus y$, $C = xy$

Figure 4.7 shows the logic diagram of the half-adder.

Next, consider addition of two 4-bit numbers as follows (next page):

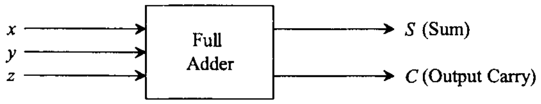
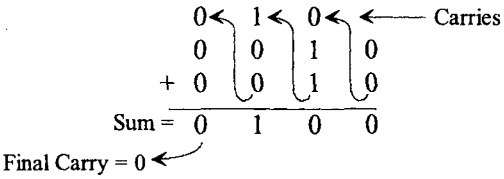


FIGURE 4.8 Block diagram of a full adder

TABLE 4.6 Truth Table of a Full Adder

Inputs			Outputs		Decimal Value
x	y	z	C	S	
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3



This addition of two bits will generate a sum and a carry. The carry may be 0 or 1. Also, there will be no previous carry while adding the least significant bits (bit 0) of the two numbers. This means that two bits need to be added for bit 0 of the two numbers. On the other hand, addition of three bits (two bits of the two numbers and a previous carry, which may be 0 or 1) is required for all the subsequent bits. Note that two half-adders are required to add three bits. A combinational circuit that adds three bits, generating a sum and a carry (which may be 0 or 1), is called a "full adder." Figure 4.8 shows the block diagram of a full adder. The full adder adds three bits, *x*, *y*, and *z*, and generates a sum and a carry. Table 4.6 shows the truth table of a full adder.

From the truth table, $S = \bar{x}\,\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\,\bar{z} + xyz = (\bar{x}y + x\bar{y})\,\bar{z} + (xy + \bar{x}\,\bar{y})\,z$

Let $w = \bar{x}\,y + x\bar{y}$ then $\bar{w} = xy + \bar{x}\,\bar{y}$. Hence, $S = w\,\bar{z} + \bar{w}\,z = w \oplus z = x \oplus y \oplus z$

Also, from the truth table, $C = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = (\bar{x}y + x\bar{y})z + xy(z + \bar{z})$

$= wz + xy$

where $w = (\bar{x}y + x\bar{y}) = x \oplus y$. Hence, $C = (x \oplus y)z + xy$.

Another form of Carry can be written as follows:

$C = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz + xyz + xyz$ (Adding redundant terms *xyz*)

$= yz\,(\bar{x} + x) + xz\,(y + \bar{y}) + xy\,(z + \bar{z}) = yz + xz + xy$

Figure 4.9 shows the logic diagram of a full adder.

Note that the names half-adder and full adder are based on the fact that two half-adders are required to obtain a full adder. This can be obtained as follows. One of the two half-adders with inputs, *x* and y will generate the sum, $S_0 = x \oplus y$ and the carry, $C_0 = xy$. The sum (S_0) output can be connected to one of the inputs of the second half-adder with *z* as

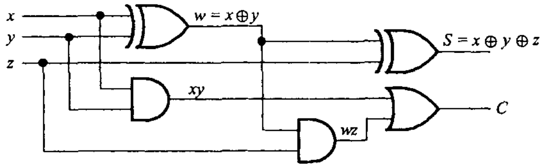


FIGURE 4.9 Logic diagram of a full adder

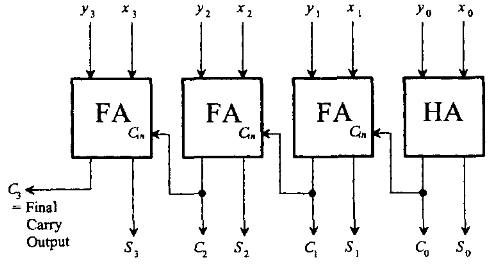


FIGURE 4.10 4-bit binary adder using one half-adder and three full adders

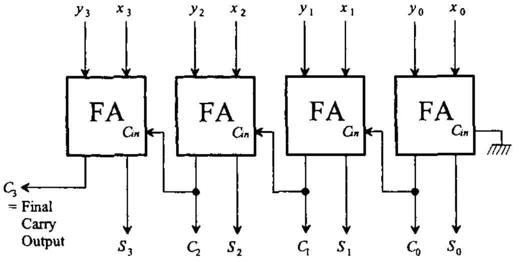


FIGURE 4.11 Four-bit binary adder using full adders

the other input. Thus, the sum output (S) and the carry output (C_1) of the second half-adder will be $S = x \oplus y \oplus z$ and $C_1 = (x \oplus y)z$. The carry outputs of the two half-adders can be logically ORed to provide the carry (C) of the full adder as $C = (x \oplus y)z + xy$. Therefore, two half-adders and a two-input OR gate can be used to obtain a full adder.

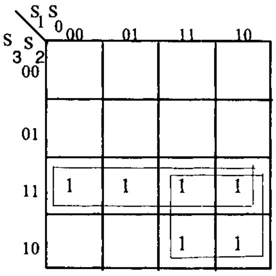
A 4-bit binary adder (also called "Ripple Carry Adder") for adding two 4-bit numbers $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$ can be implemented using one half-adder and three full adders as shown in Figure 4.10. A full adder adds two bits if one of its inputs $C_{in} = 0$. This means that the half-adder in Figure 4.10 can be replaced by a full adder with its C_{in} connected to ground. Figure 4.11 shows implementation of a 4-bit binary adder using four full adders.

From Chapter 2, addition of two BCD digits is correct if the binary sum is less than or equal to 1001_2 (9 in decimal). A binary sum greater than 1001_2 results into an invalid BCD sum; adding 0110_2 to an invalid BCD sum provides the correct sum with an output carry of 1. Furthermore, addition of two BCD digits (each digit having a maximum value of 9) along with carry will require correction if the sum is in the range 16 decimal through 19 decimal. A BCD adder can be designed by implementing required corrections in the result for decimal numbers from 10 through 19 (1010_2 through 10011_2). Therefore, a correction is necessary for the following:

i) If the binary sum is greater than or equal to decimal 16 (This will generate a carry of one)
ii) If the binary sum is 1010_2 through 1111_2. For example, consider adding packed BCD numbers 99 and 38:

$$\begin{array}{llll}
 & 111\leftarrow\text{Intermediate Carries} & & \\
99 & 1001 & 1001 & \text{BCD for 99}\\
\underline{+38} & \underline{0011} & \underline{1000} & \text{BCD for 38}\\
137 & 1101 & 0001 & \text{invalid sum}\\
 & \underline{+0110} & \underline{+0110} & \text{add 6 for correction}\\
\underbrace{0001}_{1} & \underbrace{0011}_{3} & \underbrace{0111}_{7} & \leftarrow \text{correct answer 137}
\end{array}$$

This means that a carry (C_{11}) is generated: i) when the binary sum, $S_3S_2S_1S_0 = 1010_2$ through 1111_2 or ii) when the binary sum is greater than or equal to decimal 16. For case i), using a K-map, $C_{11} = S_1S_3 + S_2S_3$ as follows (next page):



Hence, $C_{11} = S_1S_3 + S_2S_3 = S_3(S_1 + S_2)$. Combining cases i) and ii), $C_1 = C_0 + S_3(S_1 + S_2)$. This is implemented in the Figure 4.12.

Note that C_1 is the output carry of the BCD adder while C_0 is the carry output from the first binary adder. When $C_1 = 0$, zeros are added to $S_3S_2S_1S_0$. This situation occurs when $S_3S_2S_1S_0$ is less than or equal to 1001_2. However, when $C_1 = 1$, the binary number 0110 is added to $S_3S_2S_1S_0$ using the second 4-bit adder. This situation occurs when $S_3S_2S_1S_0$ is greater than or equal to binary 1010 or when $S_3S_2S_1S_0$ is greater than or equal to 16 decimal. The carry output from the second 4-bit adder can be discarded. Note that BCD parallel adder for adding n BCD digits can be obtained using n BCD adders by connecting the output carry (C_1) of each low BCD adder to C_{in} of the next BCD adder.

Next, half-subtractor and full-subtractor will be discussed. Similar to half-adder and full-adder, there are half-subtractor and full-subtractor. Using half- and full-subtractors, subtraction operation can be implemented with logic circuits in a direct manner. A half-subtractor is a combinational circuit that subtracts two bits generating a result (R) bit and a borrow (B) bit. The truth table for the half-subtractor is provided below:

x (minuend)	y (subtrahend)	B (borrow)	R (result)
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

The borrow (B) is 0 if x is greater than or equal to y; B = 1 if x is less than y.
From the truth table, $R = \bar{x}y + x\bar{y} = x \oplus y$ and $B = \bar{x}y$.
A full -subtractor is a combinational circuit that performs the operation among three bits x - y - z generating a result bit (R) and a borrow bit (B). The truth table for the full-

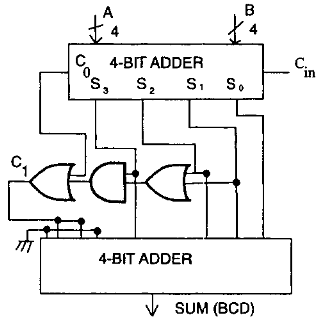


FIGURE 4.12 BCD Adder

subtractor is provided below:

x	y	z	B (Borrow)	R (Result)
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

From the above truth table, the following equations can be obtained:
$R = x \oplus y \oplus z$ and $B = \bar{x}\,y + \bar{x}\,z + yz$.
It is advantageous to implement addition and subtraction with full-adders since both operations can be obtained using a single logic circuit.

4.5.2 Comparators

The digital comparator is a widely used combinational system. Figure 4.13 shows a 2-bit

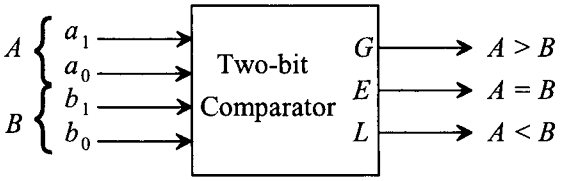
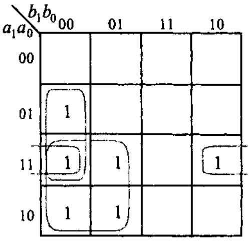


FIGURE 4.13 Block diagram of a two-bit comparator

TABLE 4.7 Truth Table for the 2-Bit Comparator

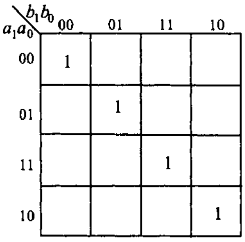
Inputs				Outputs		
a_1	a_0	b_1	b_0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K-map for G:



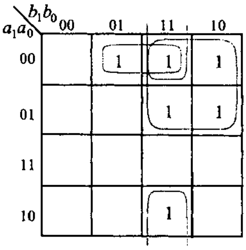
$$G = a_1\overline{b_1} + a_0\overline{b_1}\,\overline{b_0} + a_1 a_0\overline{b_0}$$

K-map for E:



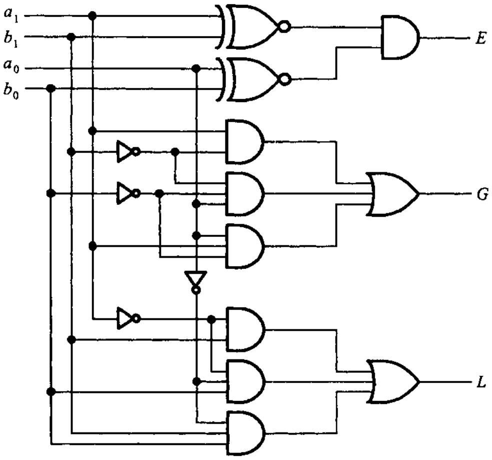
$$\begin{aligned} E &= \overline{a_1}\,\overline{a_0}\,\overline{b_1}\,\overline{b_0} + \overline{a_1}a_0\overline{b_1}b_0 + a_1a_0b_1b_0 + a_1\overline{a_0}b_1\overline{b_0} \\ &= \overline{a_1}\,\overline{b_1}(\overline{a_0}\,\overline{b_0} + a_0b_0) + a_1b_1(a_0b_0 + \overline{a_0}\,\overline{b_0}) \\ &= (a_0b_0 + \overline{a_0}\,\overline{b_0})(a_1b_1 + \overline{a_1}\,\overline{b_1}) \\ &= (a_0 \odot b_0)(a_1 \odot b_1) \end{aligned}$$

K-map for L:



$$L = \overline{a_1}b_1 + \overline{a_0}b_1b_0 + \overline{a_1}\,\overline{a_0}b_0$$

a) K-maps for the 2-bit comparator



b) Logic Diagram of the 2-bit comparator

FIGURE 4.14 Design of a 2-bit comparator

digital comparator, which provides the result of comparing two 2-bit unsigned numbers as follows:

Input Comparison	Outputs G	 E	 L
$A > B$	1	0	0
$A < B$	0	0	1
$A = B$	0	1	0

Table 4.7 provides the truth table for the 2-bit comparator.
Figure 4.14 shows the K-map and the logic diagram:

4.5.3 Decoders

An n-bit binary number provides 2^n minterms or maxterms. For example, a 2-bit binary number will generate 4 (2^2) minterms or maxterms. A decoder is a combinational circuit , when enabled, selects one of 2^n minterms or maxterms at the output based on the input combinations. However, a decoder sometimes may have less than 2^n outputs. For example, the BCD to seven-segment decoder has 4 inputs and 7 outputs rather than 16 (2^4) outputs.

The block diagram of a 2-to-4 decoder is shown in Figure 4.15. Table 4.8 provides

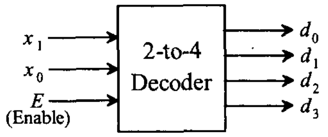


FIGURE 4.15 Block diagram of the 2-to-4 decoder

TABLE 4.8 Truth Table of the 2-to-4 Decoder

Inputs E	 x_1	 x_0	Outputs d_0	 d_1	 d_2	 d_3
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

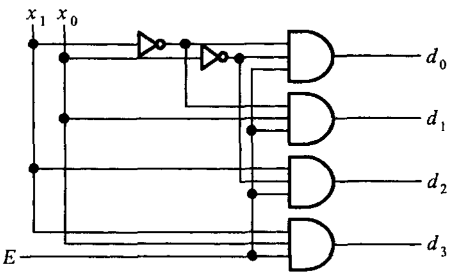


FIGURE 4.16 Logic diagram of the 2-to-4 decoder

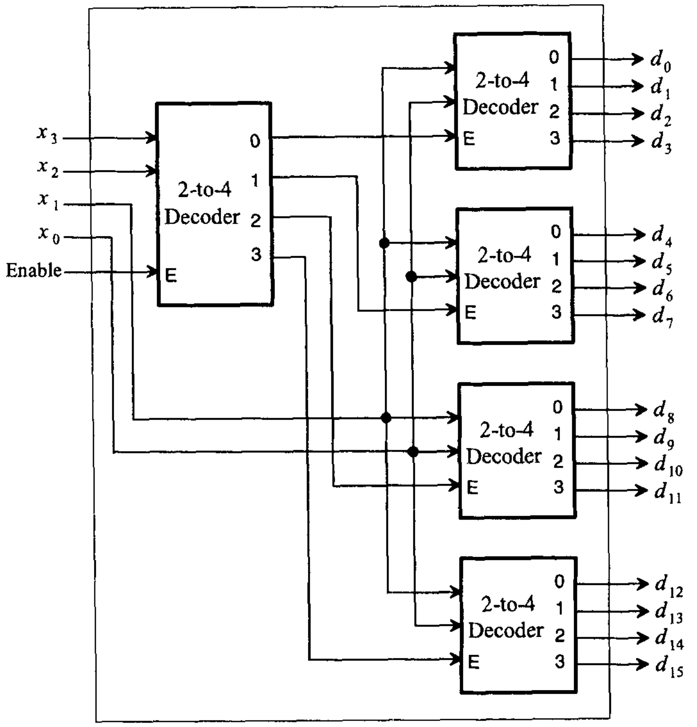


FIGURE 4.17 Implementation of a 4-to-16 Decoder Using 2-to-4 decoders

the truth table. In the truth table, the symbol X is the don't care condition, which can be 0 or 1. Also, $E = 0$ disables the decoder. On the other hand, the decoder is enabled when $E = 1$. For example, when $E = 1, x_1 = 0, x_0 = 0$, and the output d_0 is HIGH while the other outputs d_1, d_2, and d_3 are zero. Note that $d_0 = E\bar{x}_1\bar{x}_0$, $d_1 = E\bar{x}_1 x_0$, $d_2 = E x_1 \bar{x}_0$, *and* $d_3 = E x_1 x_0$. Therefore, the 2-to-4 line decoder outputs one of the four minterms of the two input variables x_1 and x_0 when $E = 1$. In general, for n inputs, the n-to 2^n decoder when enabled selects one of 2^n minterms or maxterms at the output based on the input combinations. The decoder actually provides binary to decimal conversion operation. Using the truth table of Table 4.8, a logic diagram of the 2-to-4 decoder can be obtained as shown in Figure 4.16. Large decoders can be designed using small decoders as the building blocks. For example, a 4-to-16 line decoder can be designed using five 2-to-4 decoders as shown in Figure 4.17.

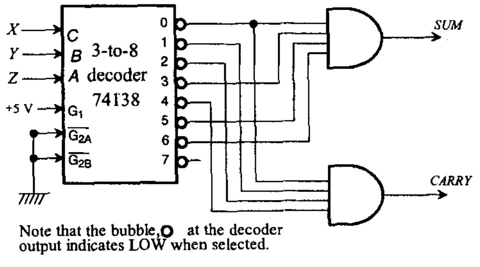


FIGURE 4.18 Implementation of a Full-adder Using a 74138 Decoder and Two 4-input AND Gates

Commercially available decoders are normally built using NAND gates rather than AND gates because it is less expensive to produce the selected decoder output in its complement form. Also, most commercial decoders contain one or more enable inputs to control the circuit operation. An example of the commercial decoder is the 74HC138 or the 74LS138. This is a 3-to-8 decoder with three enable lines G_1, $\overline{G_{2A}}$, and $\overline{G_{2B}}$. When $G_1 = H$, $\overline{G_{2A}} = L$ and $\overline{G_{2B}} = L$, the decoder is enabled. The decoder has three inputs, C, B, and A, and eight outputs Y_0, Y_1, Y_2, ..., Y_7. With $CBA = 001$ and the decoder enabled, the selected output line Y_1 (line 1) goes to LOW while the other output lines stay HIGH.

Because any Boolean function can be expressed as a logical sum of minterms, a decoder can be used to produce the minterms. A Boolean function can then be obtained by logical operation of the appropriate minterms. However, since the 74138 generates a LOW on the selected output line, a Boolean function can be obtained by logically ANDing the appropriate minterms. For example, consider the truth table of the full adder listed in Table 4.6. The inverted sum and the inverted carry can be expressed in terms of minterms as follows:

$$\overline{SUM} = \sum m(0, 3, 5, 6), \quad SUM = \overline{m_0} \bullet \overline{m_3} \bullet \overline{m_5} \bullet \overline{m_6}$$

$$\overline{CARRY} = \sum m(0, 1, 2, 4), \quad CARRY = \overline{m_0} \bullet \overline{m_1} \bullet \overline{m_2} \bullet \overline{m_4}$$

Figure 4.18 shows the implementation of a full adder using a 74138 decoder ($C=X$, $B=Y$, $A=Z$) and two 4-input AND gates. Note that the 74138 in the Manufacturer's data book uses the symbols C, B, A as three inputs to the decoder with C as the most significant

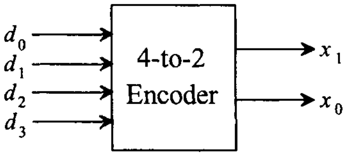


FIGURE 4.19 Block diagram of a 4-to-2 encoder

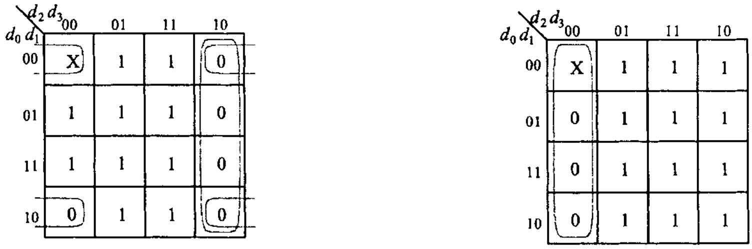
TABLE 4.9 Truth Table of the 4-to-2 Encoder

Inputs				Outputs	
d_0	d_1	d_2	d_3	x_1	x_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

TABLE 4.10 Truth Table of the 4-to-2 Priority Encoder

Inputs				Outputs	
d_0	d_1	d_2	d_3	x_1	x_0
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

X means don't care



d_0d_1 \ d_2d_3	00	01	11	10
00	X	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

d_0d_1 \ d_2d_3	00	01	11	10
00	X	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

a) K-map for $\overline{x_0}$

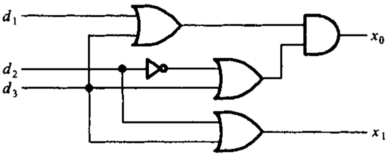
$$\overline{x_0} = \overline{d_1}\,\overline{d_3} + d_2\overline{d_3}$$

$$x_0 = (d_1 + d_3)(\overline{d_2} + d_3)$$

b) K-map for $\overline{x_1}$

$$\overline{x_1} = \overline{d_2}\,\overline{d_3}$$

$$x_1 = d_2 + d_3$$



c) Logic diagram

FIGURE 4.20 K-maps and logic diagram of a 4-to-2 priority encoder

bit and *A* as the least significant bit.

4.5.4 Encoders

An encoder is a combinational circuit that performs the reverse operation of a decoder. An encoder has a maximum of 2^n inputs and n outputs. Figure 4.19 shows the block diagram of a 4-to-2 encoder. Table 4.9 provides the truth table of the 4-to-2 encoder.

From the truth table, it can be concluded that an encoder actually performs

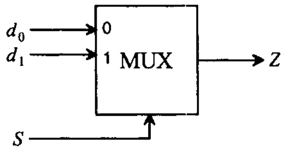


FIGURE 4.21 Block diagram of a 2-to-1 multiplexer

TABLE 4.11 Truth Table of the 2-to-1 Multiplexer

S	d_0	d_1	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

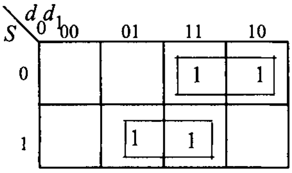


FIGURE 4.22 (a) K-map for the 2-to-1 MUX

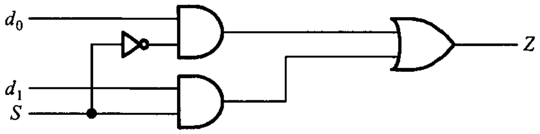


FIGURE 4.22 (b) Logic diagram of the 2-to-1 MUX

decimal-to-binary conversion. In the encoder defined by Table 4.9, it is assumed that only one of the four inputs can be HIGH at any time. If more than one input is 1 at the same time, an undefined output is generated. For example, if d_1 and d_2 are 1 at the same time, both x_0 and x_1 are 1. This represents binary 3 rather than 1 or 2. Therefore, in an encoder in which more than one input can be active simultaneously, a priority scheme must be implemented in the inputs to ensure that only one input will be encoded at the output.

A 4-to-2 priority encoder will be designed next. Suppose that it is assumed that inputs with higher subscripts have higher priorities. This means that d_3 has the highest priority and d_0 has the lowest priority. Therefore, if d_0 and d_1 become one simultaneously, the output will be 01 for d_1. Table 4.10 shows the truth table of the 4-to-2 priority encoder. Figure 4.20 shows the K-maps and the logic diagram of the 4-to-2 priority encoder.

4.5.5 Multiplexers

A multiplexer (abbreviated as MUX) is a combinational circuit that selects one of n input lines and provides it on the output. Thus, the multiplexer has several inputs and only one output. The select lines identify or address one of several inputs and provides it on the output line. Figure 4.21 shows the block diagram of a 2-to-1 multiplexer. The two inputs can be selected by one select line, S. When $S = 0$, input line 0 (d_0) will be presented as the output. On the other hand, when $S = 1$, input line 1 (d_1) will be produced at the output.

Table 4.11 shows the truth table of the 2-to-1 multiplexer. From the truth table, using the K-map of Figure 4.22 (a), it can be shown that $Z = \overline{S}d_0 + Sd_1$. Figure 4.22 (b) shows the logic diagram. In general, a multiplexer with n select lines can select one of 2^n data inputs. Hence, multiplexers are sometimes referred to as “data selectors.”

A large multiplexer can be implemented using a small multiplexer as the building block. For example, consider the block diagram and the truth table of a 4-to-1 multiplexer shown in Figure 4.23 and Table 4.12 respectively. The 4-input multiplexer can be

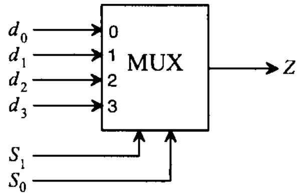


FIGURE 4.23 Block-diagram Representation of a Four-input Multiplexer

TABLE 4.12 Truth Table of the 4-to-1 Input Multiplexer

S_1	S_0	Z
0	0	d_0
0	1	d_1
1	0	d_2
1	1	d_3

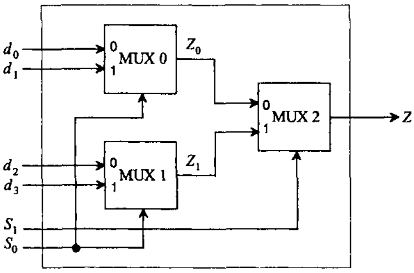


FIGURE 4.24 Implementation of a Four-Input Multiplexer Using Only Two-input Multiplexers

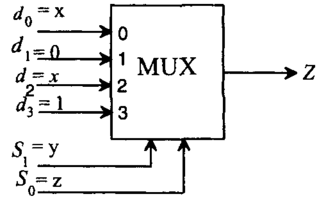


FIGURE 4.25 Implementation of a Boolean equation using a 4-to-1 multiplexer

implemented using three 2-to-1 multiplexers as shown in Figure 4.24.

In Figure 4.24, the select line S_0 is applied as input to the multiplexers MUX 0 and MUX 1. This means that $Z_0 = d_0$ or d_1 and $Z_1 = d_2$ or d_3, depending on whether $S_0 = 0$ or 1. The select line S_1 is given as input to the multiplexer MUX 2. This implies that $Z = Z_0$ if $S_1 = 0$; otherwise $Z = Z_1$. In this arrangement if $S_1S_0 = 11$, then $Z = d_3$ because $S_0 = 1$ implies that $Z_0 = d_1$ and $Z_1 = d_3$ because $S_1 = 1$, the MUX 2 selects the data input Z_1, and thus $Z = d_3$. The other entries of the truth table of Table 4.12 can be verified in a similar manner.

Multiplexers can be used to implement Boolean equations. For example, consider realizing $f(x,y,z)= x\bar{z}+ yz$ using a 4-to-1 multiplexer. First, the Boolean equation for f(x,y,z) is expressed in minterm form as follows: $f(x,y,z)=x\bar{z}(y+\bar{y}) + yz\ (x + \bar{x})= xy\bar{z} + x\bar{y}\,\bar{z}+ xyz + \bar{x}\,yz$. The next step is to use two of the three variables (x,y,z) as select inputs. Suppose y and z are arbitrarily chosen as select inputs. The four combinations ($\bar{y}\,\bar{z}$, $\bar{y}z$,$y\bar{z}$, yz) of the select inputs, y and z are then required to be factored out of minterm form for f(x,y,z) to determine the inputs to the 4-to-1 multiplexer as follows: $f\,(x,y,z)= \bar{y}\,\bar{z}(x) +\bar{y}z\ (0) +y\bar{z}(x) +yz\ (x + \bar{x}) = \bar{y}\,\bar{z}(x) + \bar{y}z\ (0) +y\bar{z}(x) +yz\ (1)$. Hence, the above equation for f(x,y,z) can be implemented using the 4-to-1 multiplexer of Figure 4.23 as follows: S_1= y, S_0= z, d_0=x, d_1=0, d_2=x, d_3=1. Figure 4.25 shows the implementation.

Next, consider implementing f(a,b,c) = Σm (0, 2, 3, 7) using the 4-to-1 multiplexer of Figure 4.23. The first step is to obtain a table as follows:

a	b	c	f	
0	0	0	1	
0	0	1	0	$f=\bar{c}$
0	1	0	1	
0	1	1	1	$f=1$
1	0	0	0	
1	0	1	0	$f=0$
1	1	0	0	
1	1	1	1	$f=c$

Hence, the 4-to-1 multiplexer of Figure 4.23 can be connected as follows: S_1=a, S_0= b, $d_0=\bar{c}$, d_1=1, d_2=0, d_3=c. Note that the inputs to the multiplexer are selected from the above table. For example, when ab=00, output $f=\bar{c}$ because f=1 when c=0 and f=0 when c=1.

4.5.6 Demultiplexers

The demultiplexer is a combinational circuit that performs the reverse operation of a multiplexer. The demultiplexer has only one input and several outputs. One of the outputs is selected by the combination of 1's and 0's of the select inputs. These inputs determine one of the output lines to be selected; data from the input line is then transferred to the selected output line. Figure 4.26 shows the block diagram of a 1-to-8 demultiplexer. Suppose that i = 1 and $S_2S_1S_0$ = 010; output line d_2 will be selected and a 1 will be output on d_2.

4.6 IEEE Standard Symbols

IEEE has developed standard graphic symbols for commonly used digital components such as adders, decoders, and multiplexers. These are depicted in Figure 4.27.

Example 4.2

Design a combinational circuit using a decoder and OR gates to implement the function depicted in Figure 4.28.

Solution

The truth table is shown in Table 4.13.
From the truth table,

$$Z_1 = \sum m(2, 3, 5, 6, 7)$$
$$Z_2 = \sum m(1, 2, 3, 7)$$

The logic diagram is shown in Figure 4.29.

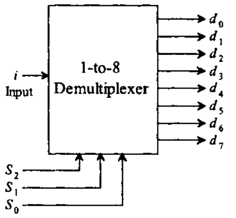
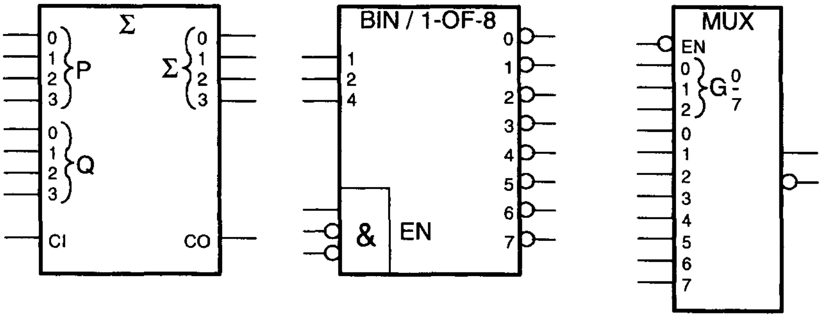


FIGURE 4.26 1-to-8 demultiplexer

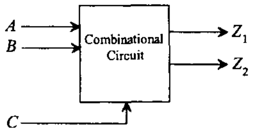


a) 4-bit Binary Adder (74LS283 or 74HC283)

a) 3-to-8 Decoder (74LS138 or 74HC138)

a) 8-to-1 Multiplexer (74LS151 or 74HC151) (providing both true and complemented outputs)

FIGURE 4.27 IEEE Symbols



If $C = 0$, Z_1 follows B and $Z_2 = A + B$.

If $C = 1$, $Z_1 = A + B$ and $Z_2 = AB$.
Assume that the decoder output is HIGH when enabled by E = 1.

FIGURE 4.28 Figure for Example 4.2

Example 4.3

Design combinational circuits using full adders and multiplexers as building blocks to implement (a) a 4-bit adder/subtractor; add when $S = 0$ and subtract when $S = 1$. (b) multiply a 4-bit unsigned number by 2 when $S=0$ and transfer zero to output when $S=1$.

Solution

(a) The subtraction $x - y$ of two binary numbers can be performed using twos complement arithmetic. As discussed before, $x - y = x +$ (ones complement of y) $+ 1$.

Using this concept, parallel subtractors can be implemented. A 4-bit adder/subtractor is shown in Figure 4.30(a). Note that XOR gates (S and y_n as inputs) can be used in place of multiplexers.

The adder/subtractor in Figure 4.30(a) utilizes four MUX's. Each MUX has one select line (S) and is capable of selecting one of two lines, y_n or $\bar{y}_n$.

The 4-bit adder/subtractor of Figure 4.30(a) either adds two 4-bit numbers and performs $(x_3 x_2 x_1 x_0)$ ADD $(y_3 y_2 y_1 y_0)$ when $S = 0$ or performs the subtraction operation $(x_3 x_2 x_1 x_0)$ MINUS $(y_3 y_2 y_1 y_0)$ for $S = 1$. The select bit S can be implemented by a switch. When $S = 0$, each MUX outputs the true value of y_n ($n = 0$ through 3) to the corresponding input of the full adder FA_n ($n = 0$ through 3). Because $S = 0$ (C_{in} for FA_0 $= 0$), the four full adders perform the desired 4-bit addition. When $S = 1$ (C_{in} for FA_0 $= 1$), each MUX generates the ones complement of y_n at the corresponding input of the full adder FA_n. Because $S = C_{in} = 1$, the four full adders provide the following operation: $(x_3 x_2 x_1 x_0) - (y_3 y_2 y_1 y_0) = (x_3 x_2 x_1 x_0) + (\bar{y}_3 \bar{y}_2 \bar{y}_1 \bar{y}_0) + 1$

(b) Assume 4-bit output $S_3 S_2 S_1 S_0$. Figure 4.30(b) shows the implementation.

TABLE 4.13 Truth Table for Example 4.2

Inputs			Outputs	
C	B	A	Z_1	Z_2
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

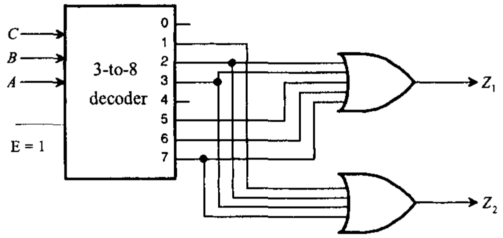


FIGURE 4.29 Implementation of Example 4.2 using a decoder and OR gates

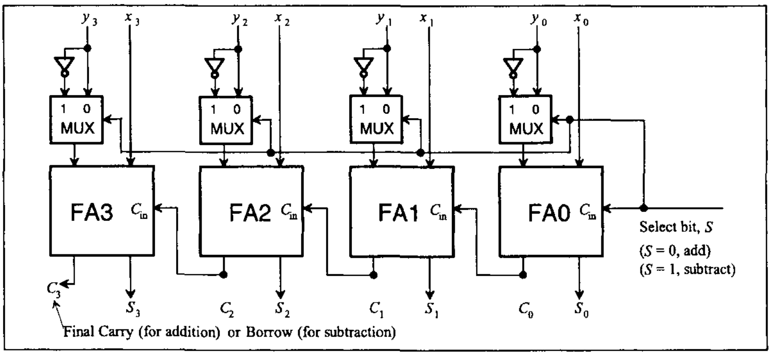


FIGURE 4.30 (a) 4-bit Adder / Subtractor

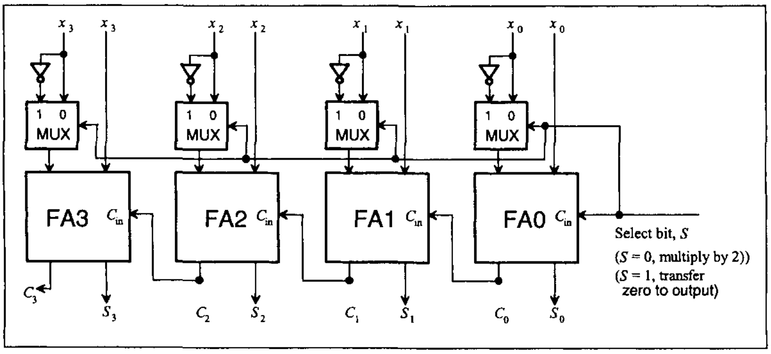


Figure 4.30 (b) Solution to Part (b)

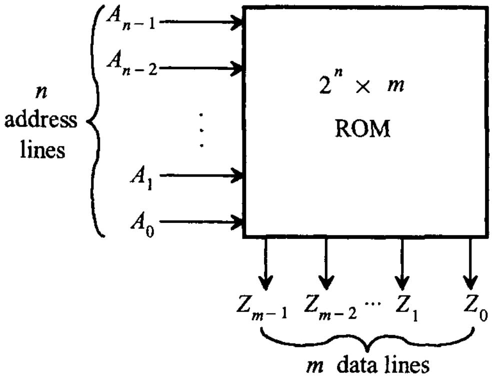


FIGURE 4.31 Block-diagram Representation of a ROM

4.7 Read-Only Memories (ROMs)

Read-only memory, commonly called "ROM," is a nonvolatile memory (meaning that it retains information in case of power failure) that provides read-only access to the stored data. A block-diagram representation of a ROM is shown in Figure 4.31. The total capacity of this ROM is $2^n \times m$ bits. Whenever an n-bit address is placed on the address line, the m-bit information stored in this address will appear on the data lines. The m-bit output generated by the ROM is also called a "word."

For example, a 1K × 8 (1024 × 8)-bit ROM chip contains 10 address pins (2^{10} = 1024 = 1K) and 8 data pins. Therefore, $n = 10$ and $m = 8$. On the other hand, an 8K × 8 (8192 × 8)-bit ROM chip includes 13 address pins (2^{13} = 8192 = 8K) and 8 data pins. Thus, $n = 13$ and $m = 8$.

A ROM is an LSI chip that can be designed using an array of semiconductor devices such as diodes, transistors, or MOS transistors. A ROM is a combinational circuit. Internally, a ROM contains a decoder and OR gates; this is illustrated in Figure 4.32. The OR gate of the ROM may be built using diodes. A typical 3-input diode OR gate is shown in Figure 4.33. Resistor R pulls the output down to a LOW level as long as all the inputs are LOW. However, if either input is connected to a high voltage source (3 to 5 volts), the output is pulled HIGH to within one diode drop of the input. Thus, the circuit operates as an OR gate. To illustrate the operation of a ROM, consider the 2 × 4-bit ROM of Figure 4.34. In this system , when $A_1 A_0 = 00$, the decoder output line 0 will be HIGH. This causes the diodes D_{00} and D_{01} to conduct, and thus the output $Z = Z_3 Z_2 Z_1 Z_0 = 0011$. Similarly, when $A_1 A_0 = 01$, the decoder output line 1 goes to high, diode D_{10} conducts, and the output will be $Z = Z_3 Z_2 Z_1 Z_0 = 0100$. Table 4.14 shows the truth table. ROM implementation offers a cost-effective solution for building circuits to perform useful tasks such as square root and transcendental function computations. Although diodes are not normally used for fabricating ROMs, the above diode-based ROM is shown for illustrative purposes.

Figure 4.35 shows the subcategories of ROMs and their associated technologies. The various types of ROMs will be discussed next.

A ROM must be programmed before it can be used. This involves placing the switching devices such as transistors (rather than diodes) at the appropriate intersection points of the row and column lines. For example, in a mask ROM the contents of the ROM are initialized by the manufacturer at the time of its production. This means that

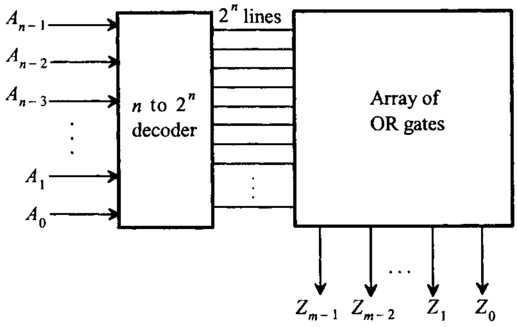


FIGURE 4.32 Internal Structure of a ROM

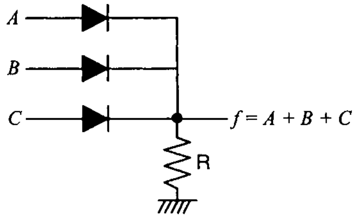


FIGURE 4.33 Diode-OR Gate

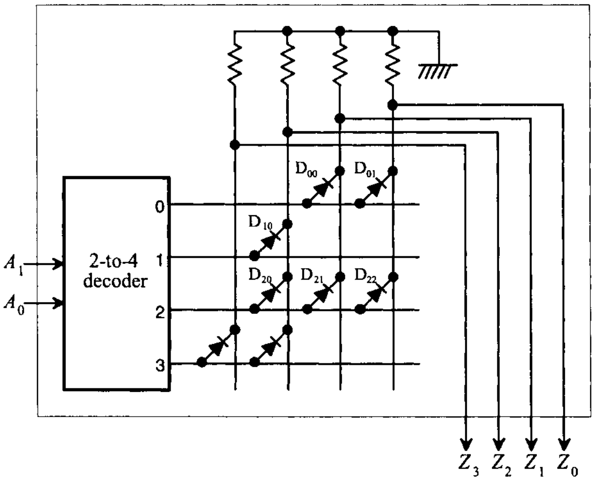


FIGURE 4.34 Hardware Organization of a Typical 2 × 4 ROM

TABLE 4.14 Truth Table implemented by the ROM of Figure 4.34

A_1	A_0	Z_3	Z_2	Z_1	Z_0
0	0	0	0	1	1
0	1	0	1	0	0
1	0	0	1	1	1
1	1	1	1	0	0

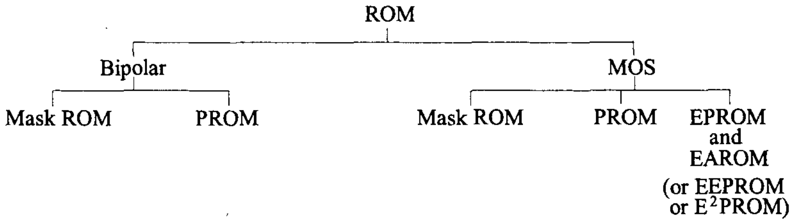


FIGURE 4.35 Subcategories of ROMs

this approach is well suited for producing a standard circuit such as a bar-code generator. Because these types of ROMs are mass-produced, their costs are also very low. However, a mask ROM cannot be reconfigured by a user. That is, a user cannot alter its contents.

Occasionally, a user may wish to develop a specific ROM-based circuit as demanded by the application area. In this case, a ROM that allows a user to initialize its contents is required. A ROM with such a flexibility is known as a PROM (programmable ROM). In this device, the manufacturer places a switching element along with a fusible link at each intersection. This implies that all ROM cells are initialized with a 1. If a user desires to store a zero in a particular cell, the fuse is blown at that point. This activity is called "programming," and it may be accomplished by passing electrical impulses. It should be pointed out that in such a ROM a user can program the ROM only once. That is, it is not possible to reprogram a PROM once the fuse is blown.

When a new product is developed, it may be necessary for the designer to modify the contents of the ROM. A ROM with this capability is referred to as an EPROM (erasable programmable ROM). Usually, the contents of this memory are completely erased by taking the EPROM chip out of the board and exposing the ROM chip to ultraviolet light. Typical erase times vary between 10 and 30 minutes. After erasure the ROM may be reprogrammed by passing voltage pulses at the special inputs. The 2764 chip is a typical example of an EPROM. It is a 28-pin 8K × 8 chip contained in a dual in-line package (DIP). It has 13 address input pins and 8 data output pins. Note that the 2764 needs 13 ($2^{13} = 8192$) pins to address 8192 (8K) locations.

The growth in IC technology allowed the production of another type of ROM whose contents may be erased using electrical impulses. These memory devices are customarily referred to as "electrically alterable ROMs" (EAROMs) or "electrically erasable PROMs" (EEPROMs or E^2PROMs). The main advantage of an EEPROM is that its contents (one or more locations) can be changed without removing the chip from the circuit board. Note that EPROMs and EAROMs are designed using only MOS transistors.

4.8 Programmable Logic Devices (PLDs)

A programmable logic device (PLD) is a generic name for an IC chip capable of being programmed by the user after it is manufactured. It is programmed by blowing fuses. A PLD chip contains an array of AND gates and OR gates. There are three types of PLDs. They are identified by the location of fuses on the AND-OR array. Figure 4.36 shows the block diagrams of these PLDs.

The PROM was discussed in the last section. A PROM contains a number of fixed AND gates and programmable OR gates. The PROM can be programmed to represent

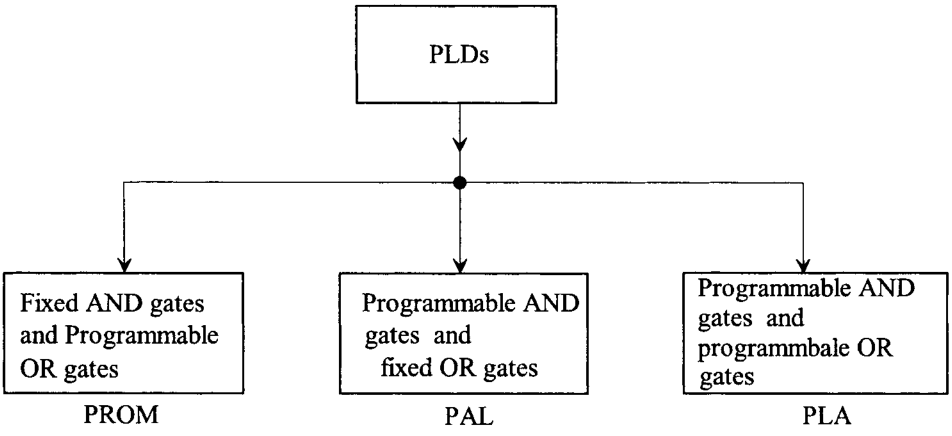
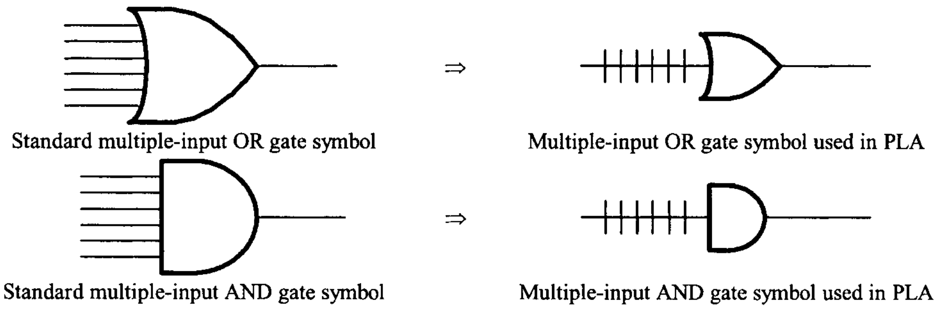

PLDs
Fixed AND gates and Programmable OR gates
Programmable AND gates and fixed OR gates
Programmable AND gates and programmbale OR gates
PROM
PAL
PLA

FIGURE 4.36 Types of PLDs



Standard multiple-input OR gate symbol

Multiple-input OR gate symbol used in PLA

Standard multiple-input AND gate symbol

Multiple-input AND gate symbol used in PLA

FIGURE 4.37 Multiple input AND and OR Gate Symbols for PLA

Boolean functions in sum of products (minterms) form. The PAL, on the other hand, includes programmable AND gates and fixed OR gates. The PAL can be programmed to implement Boolean functions as a logical sum (OR) of product terms. Finally, the PLA (programmable logic array) includes several AND and OR gates, both of which are programmable. The PLA is very flexible in the sense that the necessary AND terms can be logically ORed to provide the desired Boolean functions. Let us explain the basics of PLAs. In order to illustrate a PLA, a special AND gate or OR gate symbol with multiple inputs will be utilized as shown in Figure 4.37. The internal structure of a typical PLA is shown in Figure 4.38. The AND array of this system generates the required product terms, and the OR array is used to OR the product terms generated by the array. As in the case of the ROM, these gate arrays can be realized using diodes, transistors, or MOS devices. The significance of a PLA is explained in the following example.

Consider the PLA shown in Figure 4.39. This PLA has three inputs, A, B, and C. The AND generates from product terms $\overline{A}\,\overline{B}$, $\overline{A}\,\overline{C}$, BC, and AC. These product terms are logically summed up in the OR array, and the outputs Z_0, Z_1, and Z_2 are generated. Note that the dot in the figure indicates the presence of a switching element such as a diode or transistor. The use of PLAs is very cost-effective when the number of inputs in a

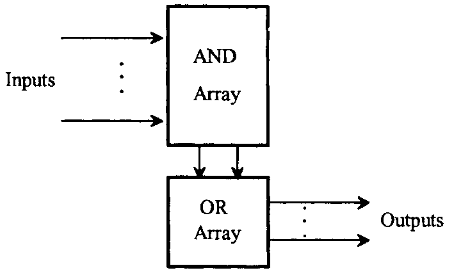


FIGURE 4.38 Internal Structure of a PLA

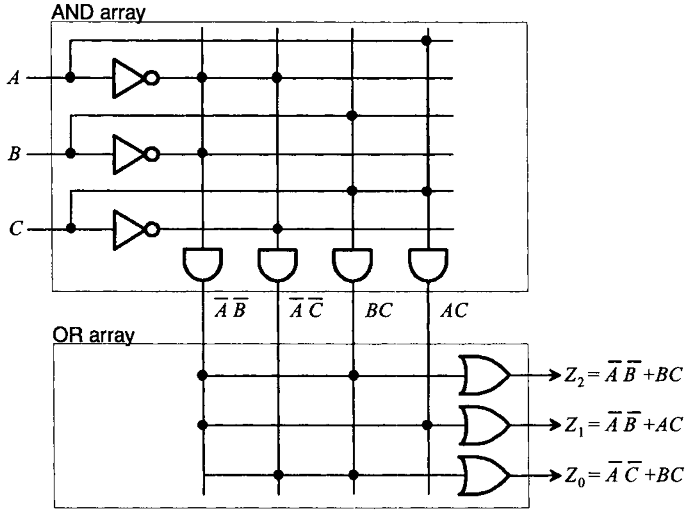

AND array
A
B
C
$\bar{A}\,\bar{B}$
$\bar{A}\,\bar{C}$
BC
AC
OR array
$Z_2=\bar{A}\,\bar{B}+BC$
$Z_1=\bar{A}\,\bar{B}+AC$
$Z_0=\bar{A}\,\bar{C}+BC$

FIGURE 4.39 A PLA with Three Inputs, Four Product Terms, and Three Outputs

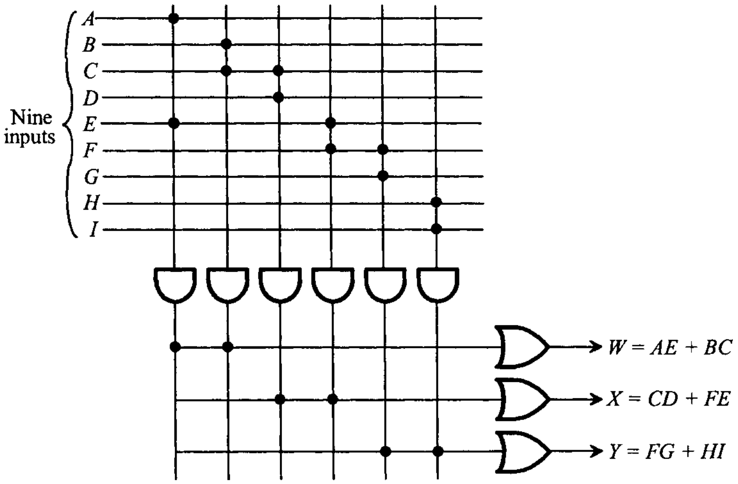


FIGURE 4.40 A PLA with Nine Inputs, Six Product Terms, and Three Outputs

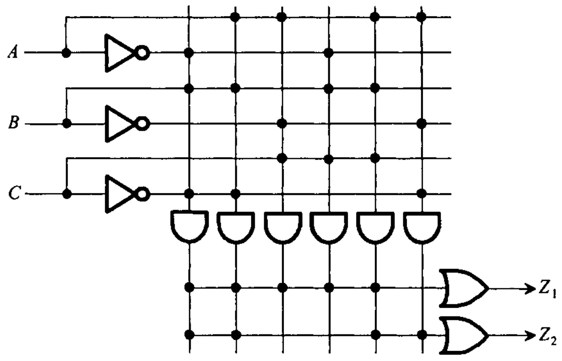


FIGURE 4.41 PLA Implementation of Example 4.3

combinational circuit realized by a ROM is very high and all input combinations are not used. For example, consider the following multiple output functions:

$$W + AE + BC$$
$$X = CD + FE$$
$$Y = FG + HI$$

To implement these Boolean functions in a ROM, a 512 × 3 array is needed because there are nine inputs (A through I) ($2^9 = 512$) and three outputs (W, X, Y), but the same functions can be realized in a PLA using six product terms, nine inputs, and three outputs, as shown in Figure 4.40. Therefore, a considerable savings in hardware can be achieved with PLAs.

<u>Example 4.4</u>

Implement Example 4.2 using PLAs.

Solution

From Example 4.2,

$$Z_1(A, B, C) = \sum m(2, 3, 5, 6, 7)$$
$$= \overline{C}B\overline{A} + \overline{C}BA + C\overline{B}A + CB\overline{A} + CBA$$

$$Z_2(A, B, C) = \sum m(1, 2, 3, 7)$$
$$= \overline{C}\,\overline{B}A + \overline{C}B\overline{A} + \overline{C}BA + CBA$$

Figure 4.41 shows the PLA implementation.

4.9 Commercially Available Field Programmable Devices (FPDs)

Both mask programmable and field programmable PLAs are available. Mask programmable PLAs are similar to mask ROMs in the sense that they are programmed at the time of manufacture. Field programmable PLAs (FPLAs) on the other hand, can be programmed by the user with a computer-aided design (CAD) program to select a minimum number of product terms to express the Boolean functions.

There are three types of commercially available Field Programmable Devices (FPDs). These are Simple PLD (SPLD), Complex PLD (CPLD), and Field Programmable Gate Array (FPGA). Among all SPLDs, PALs are widely used. SPLD uses EPROM technology to implement the switches. Note that PAL is a registered trademark of Advanced Micro Devices, Inc. (AMD). PALs were introduced by Monolithic Memories (a division

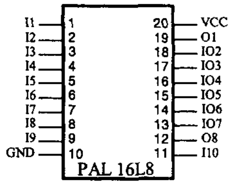


FIGURE 4.42 Pinout for PAL 16L8

of AMD) in 1970. The PAL chips are usually identified by a two-digit number followed by a letter and then one or two digits. The first two-digit number specifies the number of inputs whereas the last one or two digits define the number of outputs. The fixed number of AND gates are connected to either an OR or a NOR gate. The letter H indicates that the output gates are OR gates; the letter L is used when the outputs are NOR gates; the letter C is used when the outputs include both OR and NOR gates. Note that OR outputs generate active HIGH whereas NORs provide active LOW outputs. On the other hand, OR-NOR gates include both active HIGH and active LOW outputs.

For example, the PAL16L8 is a 20-pin chip with a maximum of 16 inputs, up to 8 outputs, one power pin, and one ground pin. The 16L8 contains 10 nonshared inputs, six inputs that are shared by six outputs, and two nonshared outputs. Figure 4.42 shows the pin diagram of the PAL16L8. Note that PEEL (Programmable Electrically Erasable Logic) devices or Erasable PLDs such as 18CV8 or 16V8 are available for instant reprogramming just like an EEPROM. These devices utilize CMOS EEPROM technology. These erasable PLDs use electronic switches rather than fuses so that they are erasable and reprogrammable like EEPROMs.

Due to advent in IC technology, larger PLDs (CPLDs) using SPLDs are designed. The SPLDs cannot be used for larger digital-design applications. Therefore, CPLD (complex PLD) chips are designed by the manufacturers such as Altera and Xlinix to accomplish this. A typical CPLD contains several PLDs (each PLD containing AND and OR gates with EEPROM or EPROM or Flash memory to implement the programmable switches) along with all the interconnections in the same chip. The IC manufacturers such as Altera and Xlinix also took a different approach for handling larger applications. They devised FPGA (Field Programmable Gate Array) chips which can be programmed at the user's location. A typical FPGA chip contains several smaller individual logic blocks (SRAM, multiplexers, gates, and flip-flops) along with all interconnections in a single chip. The FPGA does not use EEPROM technology to implement the switches; the programming information is stored in SRAM (discussed in chapter 5). The SRAM is normally programmed to store a look-up table containing the combinational circuit functions (truth table) for the logic block. The combinaional logic section and the programmed multiplexers provide the flip-flop input equations and the output of the logic block. Application of either CPLD or FPGA depends on the user's choice. Typical examples of CPLD and FPGA chips include Altera Corporation's EPM7032LC44-6(36 user I/O pins) and EPF10K10PLCC(84 user I/O pins) respectively. Products can be developed using either one from conceptual design via prototype to production in a very short time. FPGAs are very popular these days.

4.10 Hardware Description Language (HDL)

Hardware Description Languages (HDLs) such as VHDL or Verilog along with CAD

(Computer-aided design) tools, allow CPLDs and FPGAs to be programmed with millions of gates in a short time. A CAD system contains a number of tools that are used to design a logic circuit. These tools are used in the following sequence:

1. A "Schematic Capture" tool is the first step which is used to design the logic circuit using truth tables. Truth tables are normally used for a small logic function that can be part of a larger circuit. The word schematic means a logic diagram in which logic gates along with their interconnections is shown. Alternatively, the logic circuit can also be designed by a set of waveforms in a timing diagram. The CAD system uses a "Waveform Editor" to draw the timing diagram. The CAD System can then automatically translate this timing diagram to a logic diagram showing logic gates along with their interconnections.

2. The next step is called "Synthesis". The "Synthesis" CAD tool generates a set of logic expressions describing the functions required to obtain the circuit. These initial logic expressions are not in an optimal form. Based upon the designer's input of these initial logic expressions, the CAD system utilizes logic optimization during "Synthesis" to generate a minimum number of equations for obtaining a better circuit.

3. The third step is the "Functional Simulation". A Functional Simulator" tool is to verify the correct operation of the circuit being designed. A "Timing Simulator" can be used for precise simulations that takes into consideration timing details of the implementation technology of the final logic circuit.

Computer-aided design (CAD) software can be used to program CPLD and FPGA chips. Typical PLD programming languages are PALASM (Advanced Micro Devices, Inc.), ABEL (Data I/O Corporation, Inc.), VHDL (U.S. Department of Defense) and Verilog (Cadence Design Systems). ABEL stands for Advanced Boolean Expression Language while PAL Assembler is abbreviated as PALASM. ABEL is supported by a PLD language translator. The purpose of the translator is to provide the fuse pattern from the program written in ABEL in terms of the fuse pattern of a PLD. Note that most PLDs can be programmed using the sum of minterms form. The ABEL translator can minimize the equations in sum of minterms or in almost any other format. ABEL is basically a high-level language for hardware design similar to software design language such as Pascal or C.

VHDL and Verilog are PLD programming languages like ABEL for designing both Combinational and Sequential circuits. VHDL is an acronym for VHSIC Hardware Description Language. VHSIC stands for Very High Speed Integrated Circuits. The design of VHDL evolved from the United States Department of Defense (DOD) VHSIC program. VHDL is based on Ada programming language. The design of VHDL started in 1983 and after going through several versions was formally accepted as an IEEE (Institute of Electrical and Electronics Engineers) standard in 1987.

Verilog (developed by Design Automation in 1984 and later acquired by Cadence Design Systems), another hardware design language, is also popular. Verilog is not an acronym. Verilog (syntax based mostly on C and some Pascal) is easier to learn compared to VHDL (syntax based on Ada). Verilog provides more features than VHDL to support large project development. At present, both VHDL and Verilog have approximately equal market share. Typical Compilers / Simulators for VHDL and Verilog can be downloaded from the Internet.

In order to design systems using HDL, two levels of abstractions or their combinations are used. These are Structural, and Behavioral. The structural level can be used to describe a schematic or a logic diagram (gates and interconnections) of a system. This level makes the designer's task easy for hardware implementation. A "Hierarchical" structural model can be used by the designer to decompose a large digital system into smaller blocks or modules. The designer can define a block that is used repeatedly. This common block can be used by other blocks in the HDL program to accomplish the desired task.

The Behavioral level, on the other hand, is used to describe a system in terms of what it does and how it behaves rather than in terms of its components and their interconnections. Boolean expressions are used to accomplish this. Behavioral level is typically used to describe sequential circuits, although it can also be used to describe combinational circuits. The flow of data in Behavioral model can be represented via concurrent or sequential statements. Concurrent statements are executed in parallel as soon as data is available at the inputs while sequential statements are executed in the order that they are written. Behavioral model uses either sequential statements or concurrent statements. The first method is useful in describing complex digital systems. When behavioral model is described by concurrent statements, it is called Dataflow modeling. The dataflow modeling describes a digital circuit in terms of its function and flow of data through the circuit.

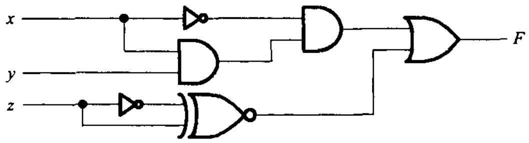
An HDL design program can be written and simulated using software tools provided by manufacturers such as SynaptiCAD (Verilogger Pro), Xlinix (ModelSim simulator / webpack 4.2), and Altera (Quartus II). These software packages are owned and remain the property of the respective manufacturers as indicated. They are protected by international copyrights, and the terms and conditions of the agreements set forth in the web sites of the manufacturers.

Verilogger Pro 8.3 can be downloaded from the web site www.syncad.com. This version allows the user to compile and simulate Verilog programs. However, some features such as save, import, export, and equation-based waveform generation are disabled. ModelSim simulator / webpack 4.2 can be downloaded from Xlinix's web site. This Xlinix software package can be used to compile and simulate VHDL programs. Simulation can be performed on the HDL design program in order to test it. An HDL program called "test bench" can be written to test an HDL design. A test bench program allows the designer to monitor the output(s) based on application of appropriate inputs. These outputs can then be verified for correctness. Test results can be represented in terms of both waveform and tabular form. The waveform typically contains timing diagrams to graphically show the relationship between time, inputs, and outputs.

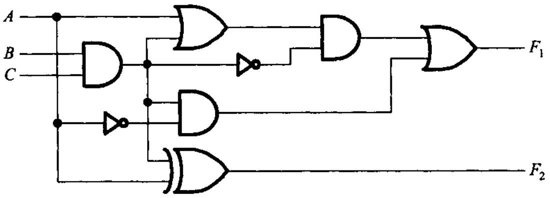
Verilog and VHDL along with examples for synthesizing Combinational circuits and Sequential circuits are discussed in Appendix I and Appendix J respectively.

QUESTIONS AND PROBLEMS

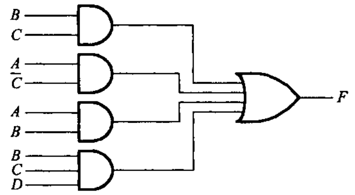
4.1 Find function F for the following circuit:



4.2 Express the following functions F_1 and F_2 in terms of the inputs A, B, and C. What is the relationship between F_1 and F_2?

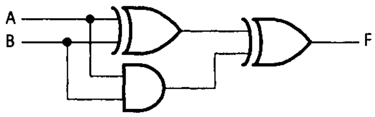


4.3 Given the following circuit:



(a) Derive the Boolean expression for $F(A, B, C, D)$.
(b) Derive the truth table.
(c) Determine the simplified expression for $F(A, B, C, D)$ using a K-map.
(d) Draw the logic diagram for the simplified expression using NAND gates.

4.4 Determine the function F of the following logic diagram and then analyze the function using Boolean identities to show that $F = A + B$.



4.5 Draw a logic diagram to implement $F = ABCDE$ using only 3-input AND gates.

4.6 Draw a logic diagram using two-input AND and OR gates to implement the following function $F = P(P + Q)(P + Q + R)(P + Q + R + S)$ without any simplification; then analyze the logic circuit to verify that $F = P$.

4.7 Design a combinational circuit with three inputs (A, B, C) and one output (F). The output is 1 when $A + C = 0$ or $AC = 1$; otherwise the output is 0. Draw a logic diagram using a single logic gate.

4.8 Design a combinational circuit that accepts a 3-bit unsigned number and generates an output binary number equal to the input number plus 1. Draw a logic diagram.

4.9 Design a combinational circuit with five input bits generating a 4-bit output that is the ones complement of four of the five input bits. Draw a logic diagram. Do not use NOT, NAND, or NOR gates.

4.10 Design a combinational circuit that converts a 4-bit BCD input to its nines complement output. Draw a logic diagram.

4.11 Design a BCD to seven-segment decoder that will accept a decimal digit in BCD and generate the appropriate outputs for the segments to display a decimal digit (0–9). Use a common anode display. Turn the seven segment OFF for non-BCD digits. Draw a logic circuit. What will happen if a common cathode display is used? Comment on the interface between the the decoder and the display.

4.12 Design a combinational circuit using a minimum number of full adders to decrement a 6-bit signed number by 2. Assume 6-bit result. Draw a logic diagram using the block diagram of a full adder as the building block.

4.13 Design a combinational circuit using full adders to multiply a 4-bit unsigned number by 2. Draw a logic diagram using the block diagram of a full adder as the building block.

4.14 Design a combinational circuit that adds two 4-bit signed numbers and generates an output of 1 if the 4-bit result is zero; the output is 0 if the 4-bit result is nonzero. Draw a logic circuit using the block diagram of a 4-bit binary adder as the building block and a minimum number of logic gates.

4.15 Design a 4 x 16 decoder using a minimum number of 74138 and logic gates.

4.16 Design a combinational circuit using a minimum number of 74138s (3 x 8 decoders) to generate the minterms m_1, m_5, and m_9 based on four switch inputs *S3, S2, S1, S0.* Then display the selected minterm number (1 or 5 or 9) on a seven-segment display by generating a 4-bit input (*W, X, Y, Z)* for a BCD to seven-segment code converter. Ignore the display for all other minterms. Note that these four inputs (*W, X, Y, Z)* can be obtained from the selected output line (1 or 5 or 9) of the decoders that is generated by the four input switches (*S3, S2, S1, S0).* Use a minimum number of logic gates. Determine the truth table, and then draw a block diagram of your implementation using the following building blocks (Figure P4.16):

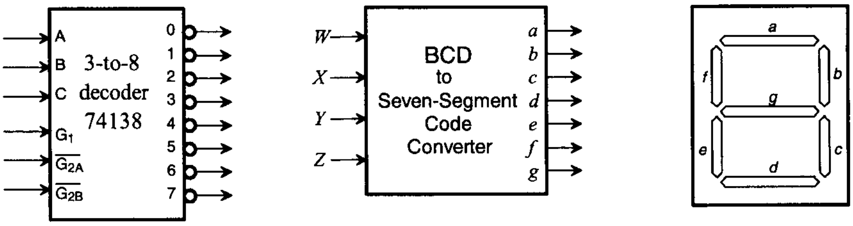


Figure P4.16

4.17 A combinational circuit is specified by the following equations:

$F_0(A, B,C) = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C}$ $F_2(A, B,C) = = \overline{A}\,\overline{B}\,\overline{C} + ABC$

$F_1(A, B,C) = A\overline{B}\,\overline{C} + AB\overline{C}$ $F_3(A, B,C) = = \overline{A}BC + \overline{A}B\overline{C} + AB\overline{C}$

Draw a logic diagram using a decoder and external gates. Assume that the decoder outputs a HIGH on the selected line.

4.18 Draw a logic diagram using a 74138 decoder and external gates to implement the following:
$F_0(A,B,C) = \sum m(1, 3, 4)$, $F_1(A,B,C) = \sum m(0, 2, 4, 7)$,
$F_2(A,B,C) = \sum m(0, 1, 3, 5, 6)$, $F_3(A,B,C) = \sum m(2, 6)$

4.19 Determine the truth table for a hexadecimal-to-binary priority encoder with line 0 having the highest priority and line 15 with the lowest.

4.20 Implement a digital circuit to increment (for $C_{in} = 1$) or decrement (for $C_{in} = 0$) a 4-bit signed number by 1 generating outputs in twos complement form. Note that C_{in} is the input carry to the full adder for the least significant bit. Draw a schematic:
(a) Using only a minimum number of full adders and multiplexers.
(b) Using only a minimum number of full adders and inverters. Do not use any multiplexers.

4.21 Implement each of the following using an 8-to-1 multiplexer:
(a) $F(A, B, C, D) = ABC + \bar{A}BD + \bar{A}\,\bar{B}\,\bar{C} + AC\bar{D}$
(b) $F(W, X, Y, Z) = \Sigma\, m(2, 3, 6, 7, 8, 9, 12, 13, 15)$

4.22 What are the main logic elements/gates in a ROM chip?

4.23 Design a combinational circuit using a 16 X 4 ROM that will increment a 4-bit unsigned number by 1. Determine the truth table and then draw a block diagram of your implementation showing the addresses and their contents in binary along with one Output Enable (OE) input.

4.24 What are the basic differences among PROM, PLA, PAL and PEEL?

4.25 What is the technology used to fabricate EPROMs and EEPROMs?

4.26 Design a 4K × 8 EPROM (with two enable lines, $\overline{CE}$ and $\overline{OE}$) based system to display the squares of BCD digits on seven segment displays using a minimum number of 74LS47 BCD to seven segment converters. Each BCD digit will be input to the EPROM via switches. The square of a particular BCD number will be displayed in BCD each time the 4-bit number is input to the EPROM via the switches. Draw a block diagram of your implementation showing the contents of memory along with addresses in hex.

4.27 Design a 4-bit adder/subtractor (Example 4.3) using only full adders and EXCLUSIVE-OR gates. Do not use any multiplexers.

4.28 Design a combinational circuit using a minimum number of full adders, and logic gates with one BCD to seven-segment converter and one seven-segment display, and which will perform A plus B or A minus B (A and B are signed numbers), depending on a mode select input, M. If M = 0, addition is carried out; if M = 1, subtraction is carried out. Assume A = $A_4 A_3 A_2 A_1 A_0$ and B = $B_4 B_3 B_2 B_1 B_0$ (Two 5-bit numbers). The circuit will be able to carry out the subtraction even if A < B. Use an LED to indicate the sign of the result (LED ON for negative result and LED OFF for positive result). The result of the operation should always appear in BCD form on the single seven-segment display. Assume that the result will be in the range of 0 through +9 in decimal and -1 through -9 in decimal. For example, if five-bit addition or subtraction provides a result of 10111 in binary, the circuit will take the two's complement of the number, and will display minus (Sign LED ON) 9 on the single seven-segment display. The Overflow bit (V) should be indicated by another LED (LED ON for V=1 and LED OFF for V=0). Do not use any multiplexers.