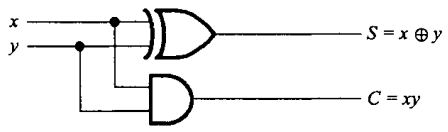



**FIGURE 4.7** Logic diagram of the half-adder

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):

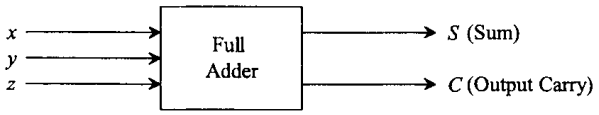



**FIGURE 4.8** Block diagram of a full adder

**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 |

```
        0 <- 1 <- 0 <-- Carries
        0    0    1    0
    +   0    0    1    0
    ---------------------
Sum =   0    1    0    0
Final Carry = 0 <-
```

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

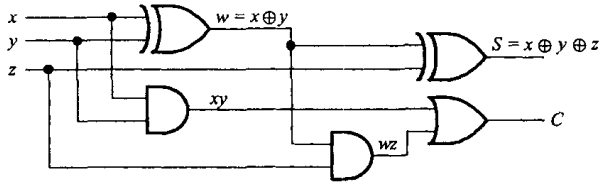



**FIGURE 4.9** Logic diagram of a full adder

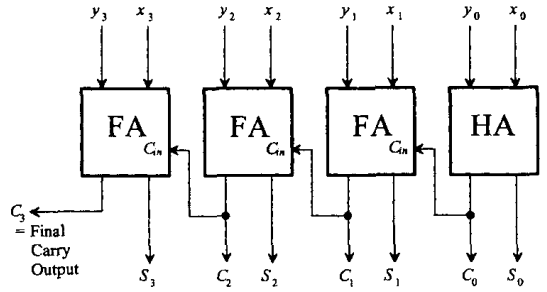



**FIGURE 4.10** 4-bit binary adder using one half-adder and three full adders

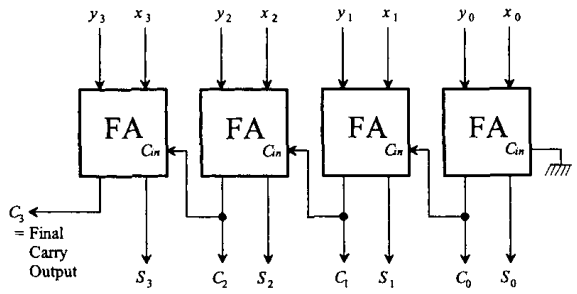



**FIGURE 4.11** Four-bit binary adder using full adders

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.

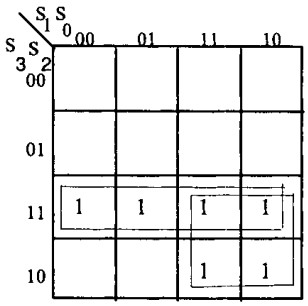
From Chapter 2, addition of two BCD digits is correct if the binary sum is less than or equal to $1001_2$ (9 in decimal). A binary sum greater than $1001_2$ results into an invalid BCD sum; adding $0110_2$ to an invalid BCD sum provides the correct sum with an output carry of 1. Furthermore, addition 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:

| | 111←Intermediate Carries | | |
|---|---|---|---|
| 99 | 1001 | 1001 | BCD for 99 |
| +38 | 0011 | 1000 | BCD for 38 |
| 137 | 1101 | 0001 | invalid sum |
| | +0110 | +0110 | add 6 for correction |
| 0001 | 0011 | 0111 | |
| 1 | 3 | 7 | ← correct answer 137 |

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-

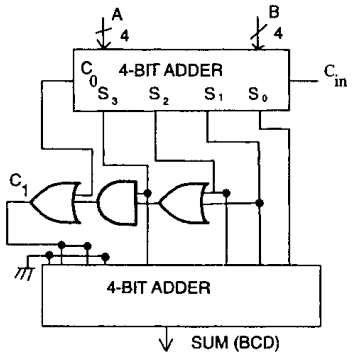



**FIGURE 4.12** BCD Adder

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

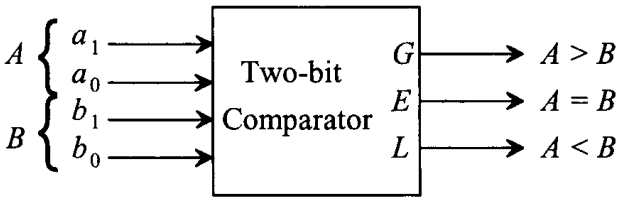
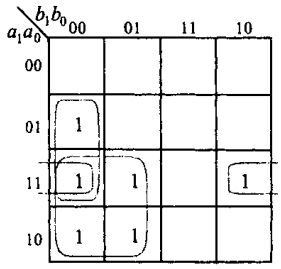



**FIGURE 4.13** Block diagram of a two-bit comparator

**TABLE 4.7** Truth Table for the 2-Bit Comparator

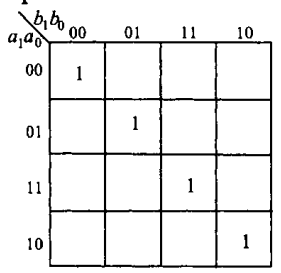
| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| $a_1$ | $a_0$ | $b_1$ | $b_0$ | $G$ | $E$ | $L$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 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 $G$:



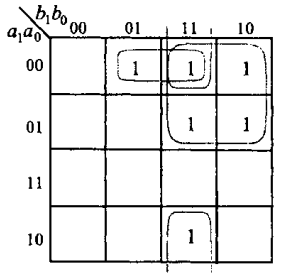

$$G = a_1\overline{b_1} + a_0\overline{b_1}\,\overline{b_0} + a_1a_0\overline{b_0}$$

K-map for $E$:



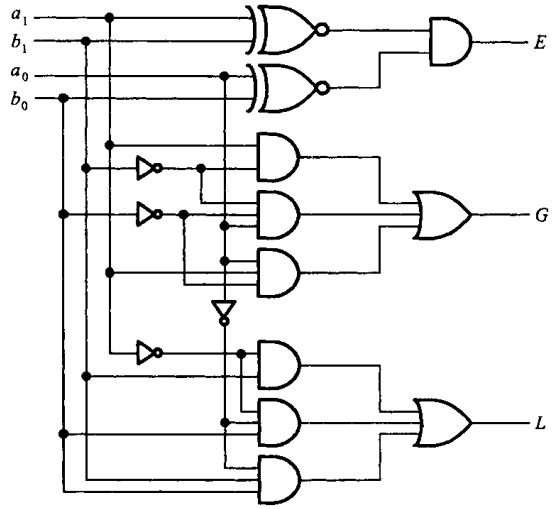

$$\begin{aligned} E &= \overline{a_1}\,\overline{a_0}\,\overline{b_1}\,\overline{b_0} + \overline{a_1}a_0\overline{b_1}b_0 + a_1a_0b_1b_0 + a_1\overline{a_0}b_1\overline{b_0} \\ &= \overline{a_1}\,\overline{b_1}(\overline{a_0}\,\overline{b_0} + a_0b_0) + a_1b_1(a_0b_0 + \overline{a_0}\,\overline{b_0})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

K-map for $L$:




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

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

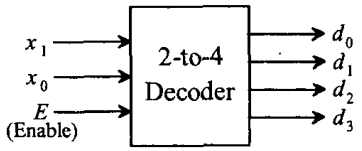



**FIGURE 4.15** Block diagram of the 2-to-4 decoder

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

| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| $E$ | $x_1$ | $x_0$ | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
| 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 |

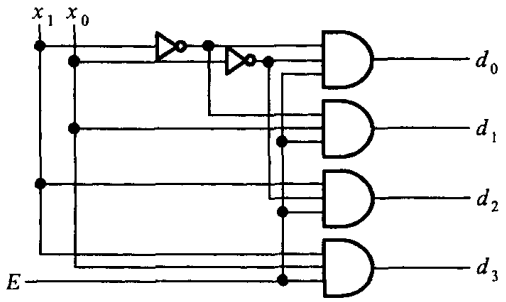



**FIGURE 4.16** Logic diagram of the 2-to-4 decoder

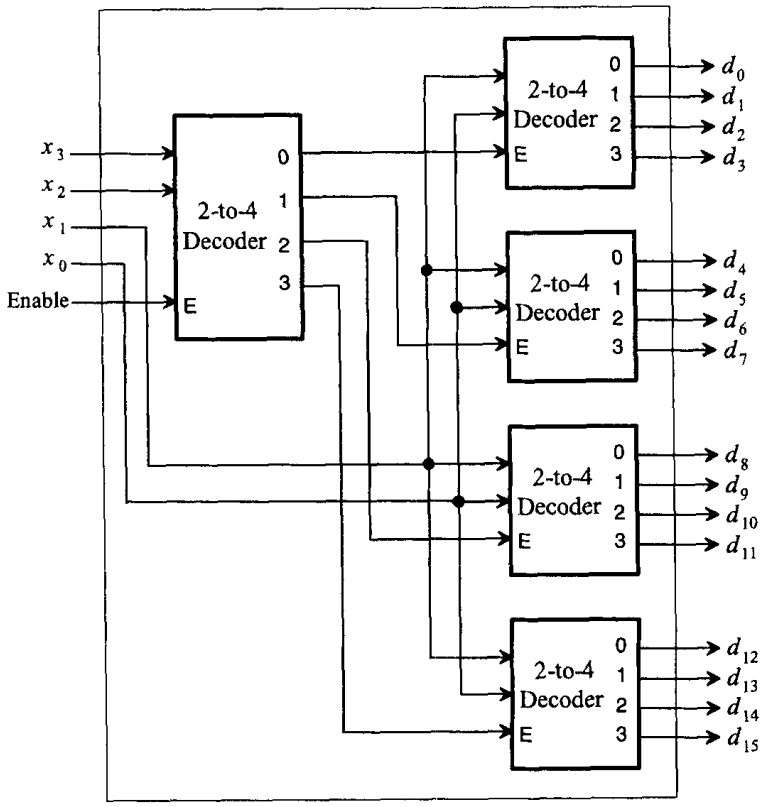



**FIGURE 4.17** Implementation of a 4-to-16 Decoder Using 2-to-4 decoders

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 $d_0 = E\overline{x_1}\,\overline{x_0}$, $d_1 = E\overline{x_1}x_0$, $d_2 = Ex_1\overline{x_0}$, *and* $d_3 = Ex_1x_0$. 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.

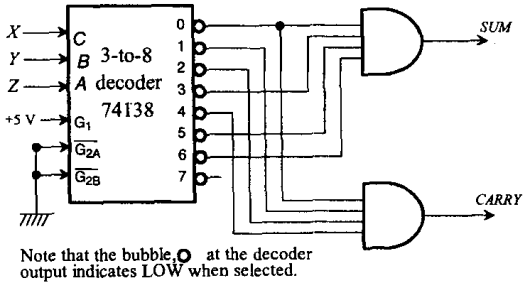



**FIGURE 4.18** Implementation of a Full-adder Using a 74138 Decoder and Two 4-input AND Gates

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:

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

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

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

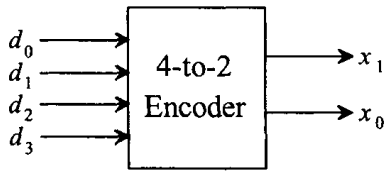



**FIGURE 4.19** Block diagram of a 4-to-2 encoder

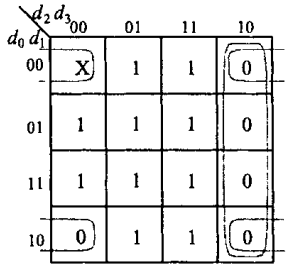
**TABLE 4.9** Truth Table of the 4-to-2 Encoder

| 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 |

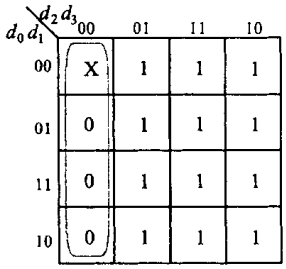
X means don't care




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

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

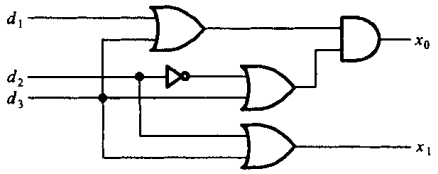
$$x_0 = (d_1 + d_3)(\overline{d_2} + d_3)$$




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

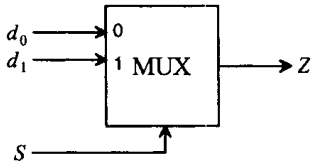



**FIGURE 4.21** Block diagram of a 2-to-1 multiplexer

**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 |

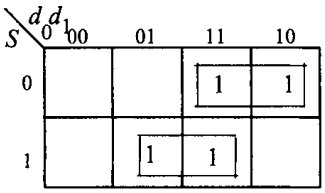



**FIGURE 4.22** (a) K-map for the 2-to-1 MUX

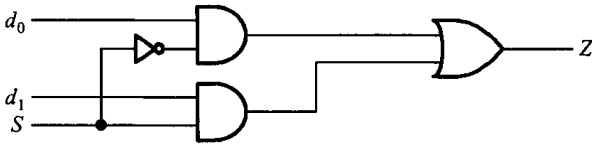



**FIGURE 4.22** (b) Logic diagram of the 2-to-1 MUX

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

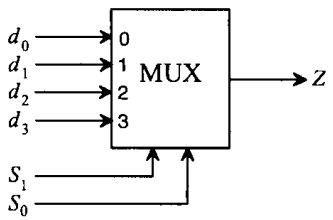



**FIGURE 4.23** Block-diagram Representation of a Four-input Multiplexer

**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 | $d_3$ |

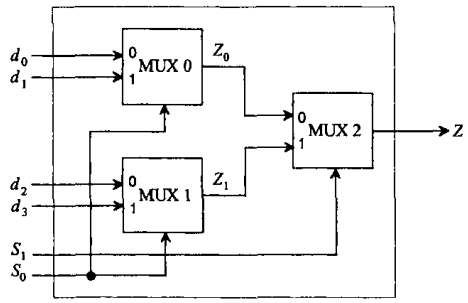



**FIGURE 4.24** Implementation of a Four-Input Multiplexer Using Only Two-input Multiplexers

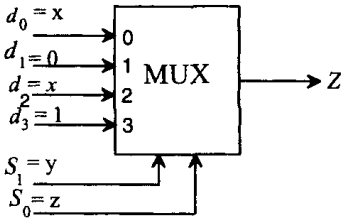



**FIGURE 4.25** Implementation of a Boolean equation using a 4-to-1 multiplexer

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) = $\sum$m (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.

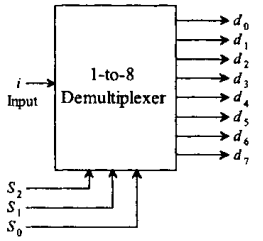
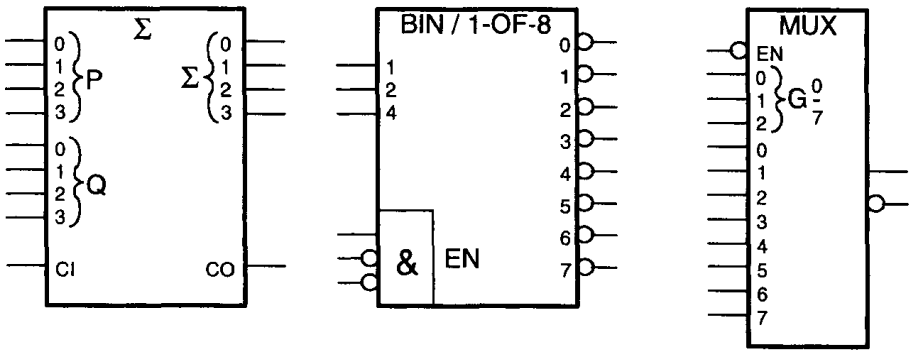



**FIGURE 4.26** 1-to-8 demultiplexer

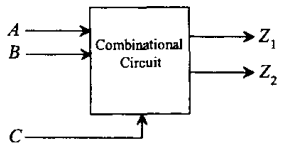



a) 4-bit Binary Adder (74LS283 or 74HC283)

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* $\overline{y_n}$.

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:
$(x_3 x_2 x_1 x_0) - (y_3 y_2 y_1 y_0) = (x_3 x_2 x_1 x_0) + (\overline{y}_3 \overline{y}_2 \overline{y}_1 \overline{y}_0) + 1$

(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 |

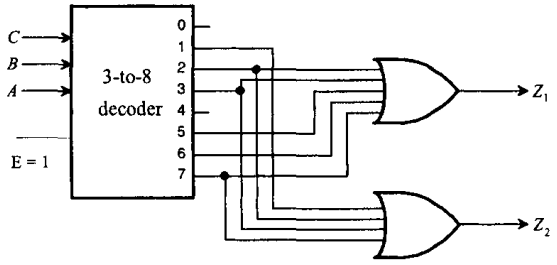



**FIGURE 4.29** Implementation of Example 4.2 using a decoder and OR gates

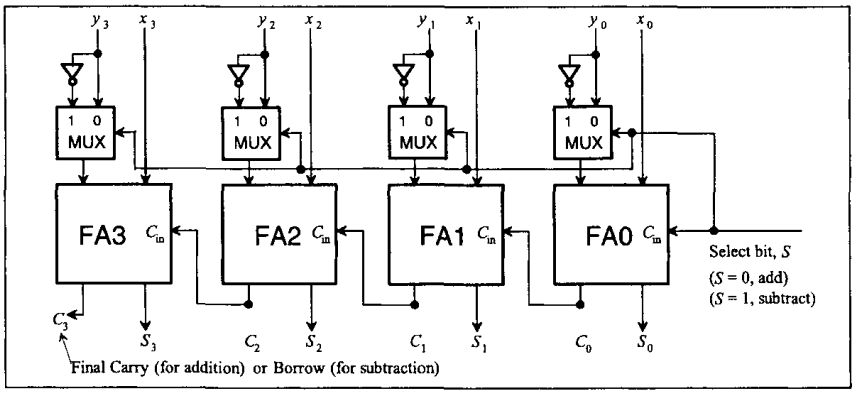



**FIGURE 4.30** (a) 4-bit Adder / Subtractor

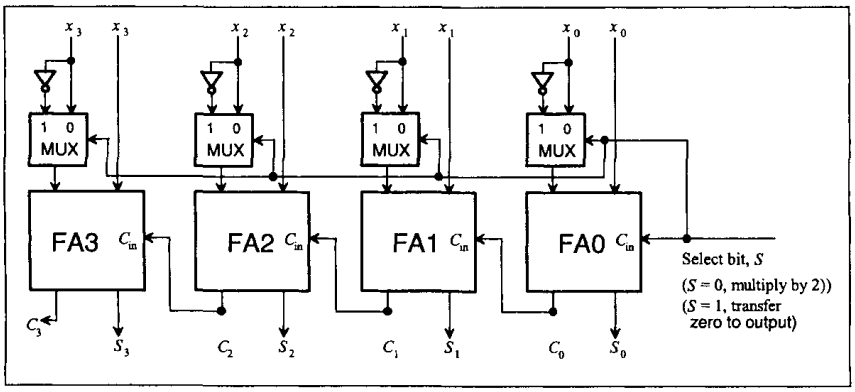



**Figure 4.30** (b) Solution to Part (b)