8

Multiplexers and Demultiplexers

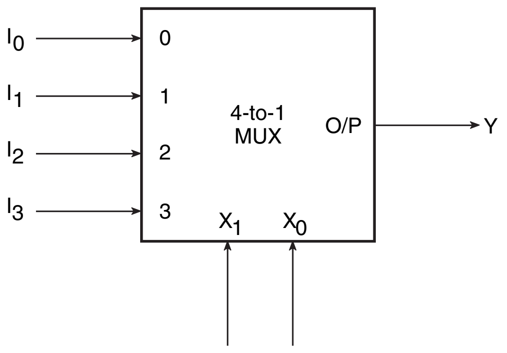
In the previous chapter, we described at length those combinational logic circuits that can be used to perform arithmetic and related operations. This chapter takes a comprehensive look at yet another class of building blocks used to design more complex combinational circuits, and covers building blocks such as multiplexers and demultiplexers and other derived devices such as encoders and decoders. Particular emphasis is given to the operational basics and use of these devices to design more complex combinational circuits. Application-relevant information in terms of the list of commonly used integrated circuits available in this category, along with their functional description is given towards the end of the chapter. The text has been adequately illustrated with the help of a large number of solved examples.

8.1 Multiplexer

A *multiplexer* or *MUX*, also called a *data selector*, is a combinational circuit with more than one input line, one output line and more than one selection line. There are some multiplexer ICs that provide complementary outputs. Also, multiplexers in IC form almost invariably have an ENABLE or STROBE input, which needs to be active for the multiplexer to be able to perform its intended function. A multiplexer selects binary information present on any one of the input lines, depending upon the logic status of the selection inputs, and routes it to the output line. If there are n selection lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n-to-1 multiplexer or $2^n \times 1$ multiplexer. Figures 8.1(a) and (b) respectively show the circuit representation and truth table of a basic 4-to-1 multiplexer.

To familiarize readers with the practical multiplexer devices available in IC form, Figs 8.2 and 8.3 respectively show the circuit representation and function table of 8-to-1 and 16-to-1 multiplexers. The 8-to-1 multiplexer of Fig. 8.2 is IC type number 74151 of the TTL family. It has an active LOW ENABLE input and provides complementary outputs. Figure 8.3 refers to IC type number 74150 of the TTL family. It is a 16-to-1 multiplexer with active LOW ENABLE input and active LOW output.

Digital Electronics: Principles, Devices and Applications Anil K. Maini
© 2007 John Wiley & Sons, Ltd. ISBN: 978-0-470-03214-5

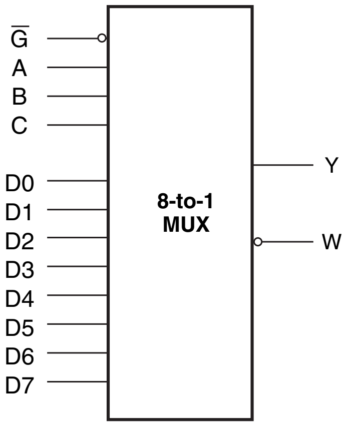


(a)

X_1	X_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b)

Figure 8.1 (a) 4-to-1 multiplexer circuit representation and (b) 4-to-1 multiplexer truth table.



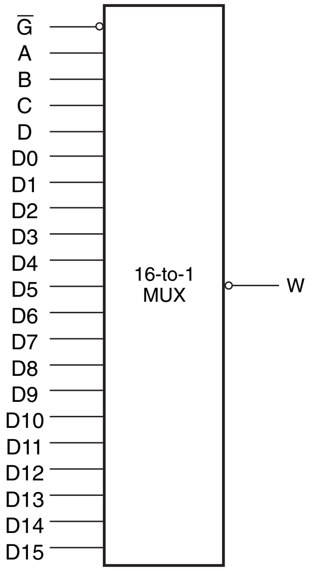
(a)

Inputs				Output	
Select			Enable		
C	B	A	$\overline{G}$	Y	W
X	X	X	H	L	$\overline{H}$
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

$\overline{G}$: ENABLE input
A, B, C : Select inputs
D0-D7 : Data inputs
Y,W : outputs

(b)

Figure 8.2 (a) 8-to-1 multiplexer circuit representation and (b) 8-to-1 multiplexer truth table.



(a)

Inputs					Output
Select				Enable	W
D	C	B	A	$\overline{G}$	
X	X	X	X	H	H
L	L	L	L	L	$\overline{D0}$
L	L	L	H	L	$\overline{D1}$
L	L	H	L	L	$\overline{D2}$
L	L	H	H	L	$\overline{D3}$
L	H	L	L	L	$\overline{D4}$
L	H	L	H	L	$\overline{D5}$
L	H	H	L	L	$\overline{D6}$
L	H	H	H	L	$\overline{D7}$
H	L	L	L	L	$\overline{D8}$
H	L	L	H	L	$\overline{D9}$
H	L	H	L	L	$\overline{D10}$
H	L	H	H	L	$\overline{D11}$
H	H	L	L	L	$\overline{D12}$
H	H	L	H	L	$\overline{D13}$
H	H	H	L	L	$\overline{D14}$
H	H	H	H	L	$\overline{D15}$

(b)

Figure 8.3 (a) 16-to-1 multiplexer circuit representation and (b) 16-to-1 multiplexer truth table.

8.1.1 Inside the Multiplexer

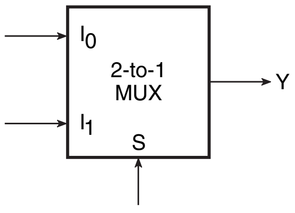
We will briefly describe the type of combinational logic circuit found inside a multiplexer by considering the 2-to-1 multiplexer in Fig. 8.4(a), the functional table of which is shown in Fig. 8.4(b). Figure 8.4(c) shows the possible logic diagram of this multiplexer. The circuit functions as follows:

- For $S = 0$, the Boolean expression for the output becomes $Y = I_0$.
- For $S = 1$, the Boolean expression for the output becomes $Y = I_1$.

Thus, inputs I_0 and I_1 are respectively switched to the output for $S = 0$ and $S = 1$. Extending the concept further, Fig. 8.5 shows the logic diagram of a 4-to-1 multiplexer. The input combinations 00, 01, 10 and 11 on the select lines respectively switch I_0, I_1, I_2 and I_3 to the output.The operation of the circuit is governed by the Boolean function (8.1). Similarly, an 8-to-1 multiplexer can be represented by the Boolean function (8.2):

$$Y = I_0.\overline{S_1}.\overline{S_0} + I_1.\overline{S_1}.S_0 + I_2.S_1.\overline{S_0} + I_3.S_1.S_0 \tag{8.1}$$

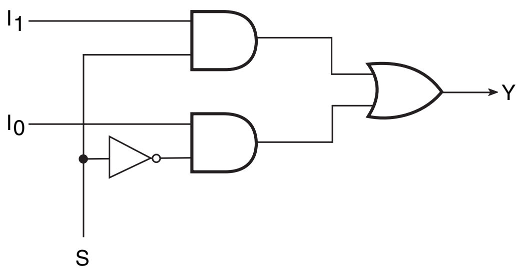
$$Y = I_0.\overline{S_2}.\overline{S_1}.\overline{S_0} + I_1.\overline{S_2}.\overline{S_1}.S_0 + I_2.\overline{S_2}.S_1.\overline{S_0} + I_3.\overline{S_2}.S_1.S_0 + I_4.S_2.\overline{S_1}.\overline{S_0}$$
$$+ I_5.S_2.\overline{S_1}.S_0 + I_6.S_2.S_1.\overline{S_0} + I_7.S_2.S_1.S_0 \tag{8.2}$$



(a)

S	Y
0	I_0
1	I_1

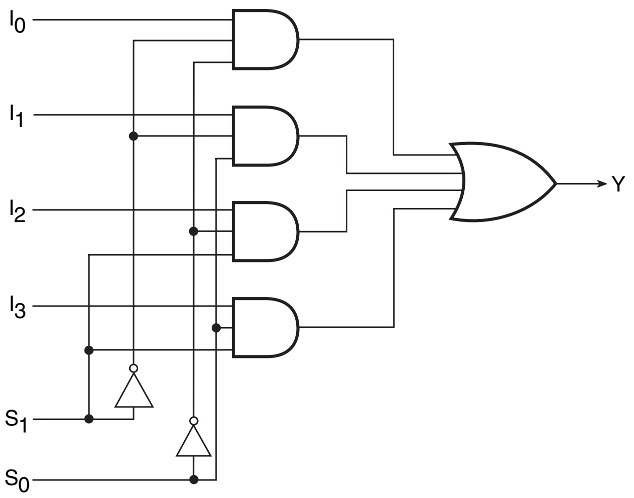
(b)



(c)

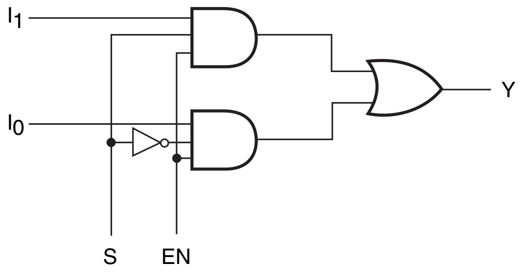
Figure 8.4 (a) 2-to-1 multiplexer circuit representation, (b) 2-to-1 multiplexer truth table and (c) 2-to-1 multiplexer logic diagram.

As outlined earlier, multiplexers usually have an ENABLE input that can be used to control the multiplexing function. When this input is enabled, that is, when it is in logic ‘1’ or logic ‘0’ state, depending upon whether the ENABLE input is active HIGH or active LOW respectively, the output is enabled. The multiplexer functions normally. When the ENABLE input is inactive, the output is disabled and permanently goes to either logic ‘0’ or logic ‘1’ state, depending upon whether the output is uncomplemented or complemented. Figure 8.6 shows how the 2-to-1 multiplexer of Fig. 8.4 can be modified to include an ENABLE input. The functional table of this modified multiplexer is also shown in Fig. 8.6. The ENABLE input here is active when HIGH. Some IC packages have more than one multiplexer. In that case, the ENABLE input and selection inputs are common to all multiplexers within the same IC package. Figure 8.7 shows a 4-to-1 multiplexer with an active LOW ENABLE input.



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

Figure 8.5 Logic diagram of a 4-to-1 multiplexer.



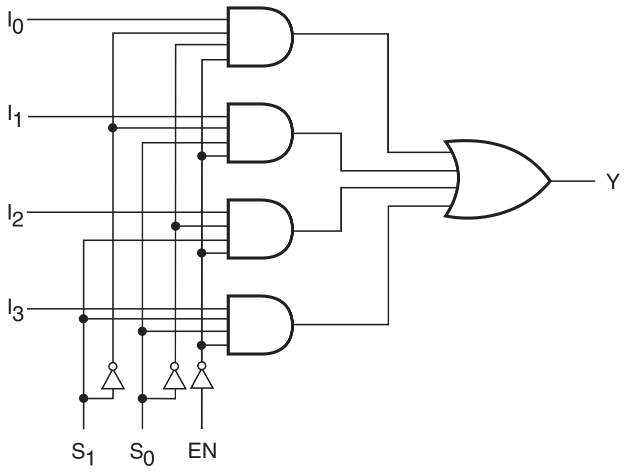
S	EN	Y
X	0	0
0	1	I_0
1	1	I_1

Figure 8.6 2-to-1 multiplexer with an ENABLE input.

8.1.2 Implementing Boolean Functions with Multiplexers

One of the most common applications of a multiplexer is its use for implementation of combinational logic Boolean functions. The simplest technique for doing so is to employ a 2^n-to-1 MUX to implement an n-variable Boolean function. The input lines corresponding to each of the minterms present in the Boolean function are made equal to logic '1' state. The remaining minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'. As an example, Fig. 8.8(a) shows the use of an 8-to-1 MUX for implementing the Boolean function given by the equation

$$f(A, B, C) = \sum 2, 4, 7 \tag{8.3}$$



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

Figure 8.7 4-to-1 multiplexer with an ENABLE input.

In terms of variables A, B and C, equation (8.3) can be written as follows:

$$f(A, B, C) = \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.C \tag{8.4}$$

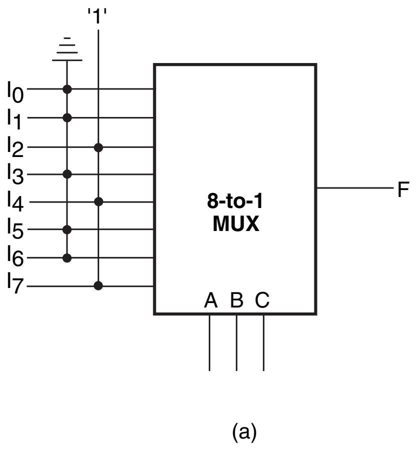
As shown in Fig. 8.8, the input lines corresponding to the three minterms present in the given Boolean function are tied to logic '1'. The remaining five possible minterms absent in the Boolean function are tied to logic '0'.

However, there is a better technique available for doing the same. In this, a 2^n-to-1 MUX can be used to implement a Boolean function with n + 1 variables. The procedure is as follows. Out of n + 1 variables, n are connected to the n selection lines of the 2^n-to-1 multiplexer. The left-over variable is used with the input lines. Various input lines are tied to one of the following: '0', '1', the left-over variable and the complement of the left-over variable. Which line is given what logic status can be easily determined with the help of a simple procedure. The complete procedure is illustrated for the Boolean function given by equation (8.3).

It is a three-variable Boolean function. Conventionally, we will need to use an 8-to-1 multiplexer to implement this function. We will now see how this can be implemented with a 4-to-1 multiplexer. The chosen multiplexer has two selection lines. The first step here is to determine the truth table of the given Boolean function, which is shown in Table 8.1.

In the next step, two of the three variables are connected to the two selection lines, with the higher-order variable connected to the higher-order selection line. For instance, in the present case, variables B and C are the chosen variables for the selection lines and are respectively connected to selection lines S_1 and S_0. In the third step, a table of the type shown in Table 8.2 is constructed. Under the inputs to the multiplexer, minterms are listed in two rows, as shown. The first row lists those terms where remaining variable A is complemented, and second row lists those terms where A is uncomplemented. This is easily done with the help of the truth table.

The required minterms are identified or marked in some manner in this table. In the given table, these entries have been highlighted. Each column is inspected individually. If neither minterm of a certain column is highlighted, a '0' is written below that. If both are highlighted, a '1' is



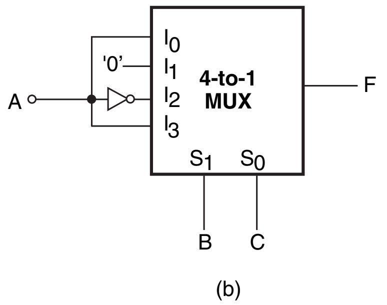


Figure 8.8 Hardware implementation of the Boolean function given by equation (8.3).

Table 8.1 Truth table.

Minterm	*A*	*B*	*C*	*f(A,B,C)*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

written. If only one is highlighted, the corresponding variable (complemented or uncomplemented) is written. The input lines are then given appropriate logic status. In the present case, I_0, I_1, I_2 and I_3 would be connected to A, 0, $\overline{A}$ and A respectively. Figure 8.8(b) shows the logic implementation.

Table 8.2 Implementation table for multiplexers.

	I_0	I_1	I_2	I_3
$\overline{A}$	0	1	**2**	3
A	**4**	5	6	**7**
	A	0	$\overline{A}$	A

Table 8.3 Implementation table for multiplexers.

	I_0	I_1	I_2	I_3
$\overline{C}$	0	**2**	**4**	6
C	1	3	5	**7**
	0	$\overline{C}$	$\overline{C}$	C

It is not necessary to choose only the leftmost variable in the sequence to be used as input to the multiplexer. Any of the variables can be used provided the implementation table is constructed accordingly. In the problem illustrated above, A was chosen as the variable for the input lines, and accordingly the first row of the implementation table contained those entries where 'A' was complemented and the second row contained those entries where A was uncomplemented. If we consider C as the left-out variable, the implementation table will be as shown in Table 8.3.

Figure 8.9 shows the hardware implementation. For the case of B being the left-out variable, the implementation table is shown in Table 8.4 and the hardware implementation is shown in Fig. 8.10.

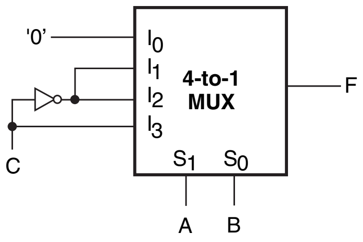


Figure 8.9 Hardware implementation using a 4-to-1 multiplexer.

Table 8.4 Implementation table for multiplexers.

	I_0	I_1	I_2	I_3
$\overline{B}$	0	1	**4**	5
B	**2**	3	6	**7**
	B	0	$\overline{B}$	B

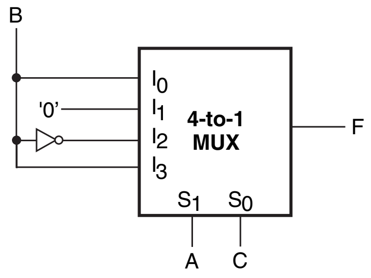


Figure 8.10 Hardware implementation using a 4-to-1 multiplexer.

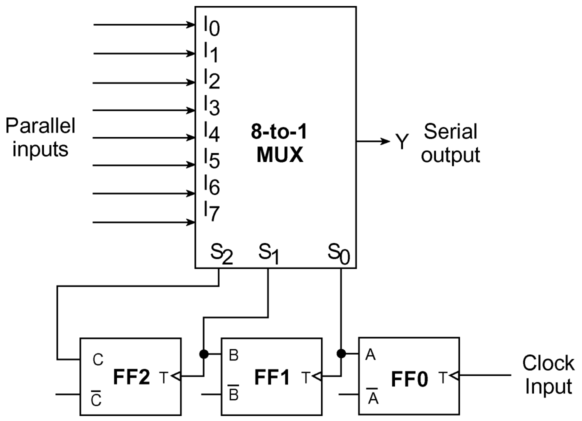


Figure 8.11 Multiplexer for parallel-to-serial conversion.

8.1.3 Multiplexers for Parallel-to-Serial Data Conversion

Although data are processed in parallel in many digital systems to achieve faster processing speeds, when it comes to transmitting these data relatively large distances, this is done serially. The parallel arrangement in this case is highly undesirable as it would require a large number of transmission lines. Multiplexers can possibly be used for parallel-to-serial conversion. Figure 8.11 shows one such arrangement where an 8-to-1 multiplexer is used to convert eight-bit parallel binary data to serial form. A three-bit counter controls the selection inputs. As the counter goes through 000 to 111, the multiplexer output goes through I_0 to I_7. The conversion process takes a total of eight clock cycles. In the figure shown, the three-bit counter has been constructed with the help of three toggle flip-flops. A variety of counter circuits of various types and complexities are, however, available in IC form. Flip-flops and counters are discussed in detail in Chapters 10 and 11 respectively.

Example 8.1

Implement the product-of-sums Boolean function expressed by $\Pi 1,2,5$ by a suitable multiplexer.

Solution

- Let the Boolean function be $f(A, B, C) = \prod 1, 2, 5$.
- The equivalent sum-of-products expression can be written as $f(A, B, C) = \sum 0, 3, 4, 6, 7$.

The truth table for the given Boolean function is given in Table 8.5. The given function can be implemented with a 4-to-1 multiplexer with two selection lines. Variables A and B are chosen for the selection lines. The implementation table as drawn with the help of the truth table is given in Table 8.6. Figure 8.12 shows the hardware implementation.

Table 8.5 Truth table.

C	B	A	$f(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 8.6 Implementation table.

	I_0	I_1	I_2	I_3
$\overline{C}$	**0**	1	2	**3**
C	**4**	5	**6**	**7**
	1	0	C	1

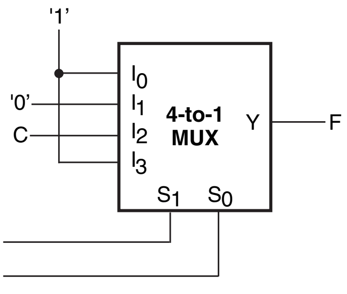


Figure 8.12 Example 8.1.

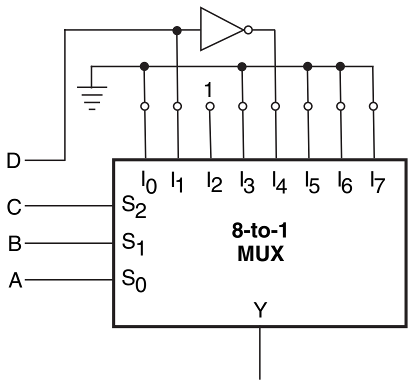


Figure 8.13 Example 8.2.

Example 8.2

Figure 8.13 shows the use of an 8-to-1 multiplexer to implement a certain four-variable Boolean function. From the given logic circuit arrangement, derive the Boolean expression implemented by the given circuit.

Solution

This problem can be solved by simply working backwards in the procedure outlined earlier for designing the multiplexer-based logic circuit for a given Boolean function. Here, the hardware implementation is known and the objective is to determine the corresponding Boolean expression.

From the given logic circuit, we can draw the implementation table as given in Table 8.7. The entries in the first row (0, 1, 2, 3, 4, 5, 6, 7) and the second row (8, 9, 10, 11, 12, 13, 14, 15) are so because the selection variable chosen for application to the inputs is the MSB variable D. Entries in the first row include all those minterms that contain $\overline{D}$, and entries in the second row include all those minterms that contain D. After writing the entries in the first two rows, the entries in the third row can be filled in by examining the logic status of different input lines in the given logic circuit diagram. Having completed the third row, relevant entries in the first and second rows are highlighted. The Boolean expression can now be written as follows:

$$Y = \sum 2, 4, 9, 10 = \overline{D}.\overline{C}.B.\overline{A} + \overline{D}.C.\overline{B}.\overline{A} + D.\overline{C}.\overline{B}.A + D.\overline{C}.B.\overline{A}$$

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

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

Table 8.7 Implementation table.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{D}$	0	1	**2**	3	**4**	5	6	7
D	8	**9**	**10**	11	12	13	14	15
	0	D	1	0	$\overline{D}$	0	0	0

8.1.4 Cascading Multiplexer Circuits

There can possibly be a situation where the desired number of input channels is not available in IC multiplexers. A multiple number of devices of a given size can be used to construct multiplexers that can handle a larger number of input channels. For instance, 8-to-1 multiplexers can be used to construct 16-to-1 or 32-to-1 or even larger multiplexer circuits. The basic steps to be followed to carry out the design are as follows:

1. If 2^n is the number of input lines in the available multiplexer and 2^N is the number of input lines in the desired multiplexer, then the number of individual multiplexers required to construct the desired multiplexer circuit would be 2^{N-n}.
2. From the knowledge of the number of selection inputs of the available multiplexer and that of the desired multiplexer, connect the less significant bits of the selection inputs of the desired multiplexer to the selection inputs of the available multiplexer.
3. The left-over bits of the selection inputs of the desired multiplexer circuit are used to enable or disable the individual multiplexers so that their outputs when ORed produce the final output. The procedure is illustrated in solved example 8.3.

Example 8.3

Design a 16-to-1 multiplexer using two 8-to-1 multiplexers having an active LOW ENABLE input.

Solution

A 16-to-1 multiplexer can be constructed from two 8-to-1 multiplexers having an ENABLE input. The ENABLE input is taken as the fourth selection variable occupying the MSB position. Figure 8.14 shows the complete logic circuit diagram. IC 74151 can be used to implement an 8-to-1 multiplexer.

The circuit functions as follows. When S_3 is in logic '0' state, the upper multiplexer is enabled and the lower multiplexer is disabled. If we recall the truth table of a four-variable Boolean function, S_3 would be '0' for the first eight entries and '1' for the remaining eight entries. Therefore, when $S_3 = 0$ the final output will be any of the inputs from D_0 to D_7, depending upon the logic status of S_2, S_1 and S_0. Similarly, when $S_3 = 1$ the final output will be any of the inputs from D_8 to D_{15}, again depending upon the logic status of S_2, S_1 and S_0. The circuit therefore implements the truth table of a 16-to-1 multiplexer.

8.2 Encoders

An *encoder* is a multiplexer without its single output line. It is a combinational logic function that has 2^n (or fewer) input lines and n output lines, which correspond to n selection lines in a multiplexer. The n output lines generate the binary code for the possible 2^n input lines. Let us take the case of an octal-to-binary encoder. Such an encoder would have eight input lines, each representing an octal digit, and three output lines representing the three-bit binary equivalent. The truth table of such an encoder is given in Table 8.8. In the truth table, D_0 to D_7 represent octal digits 0 to 7. A, B and C represent the binary digits.

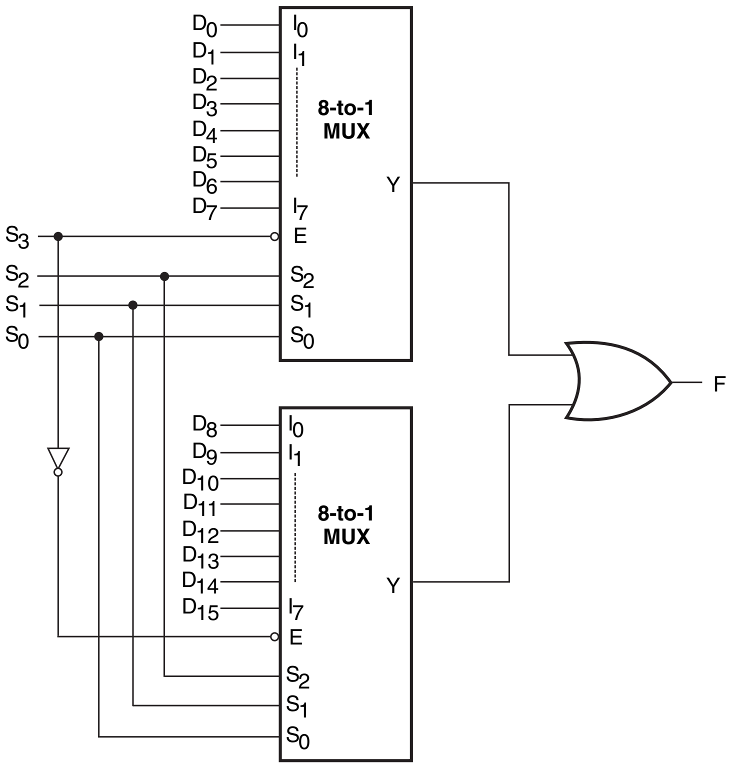


Figure 8.14 Example 8.3.

The eight input lines would have $2^8 = 256$ possible combinations. However, in the case of an octal-to-binary encoder, only eight of these 256 combinations would have any meaning. The remaining combinations of input variables are 'don't care' input combinations. Also, only one of the input lines at a time is in logic '1' state. Figure 8.15 shows the hardware implementation of the octal-to-binary encoder described by the truth table in Table 8.8. This circuit has the shortcoming that it produces an all 0s output sequence when all input lines are in logic '0' state. This can be overcome by having an additional line to indicate an all 0s input sequence.

8.2.1 Priority Encoder

A *priority encoder* is a practical form of an encoder. The encoders available in IC form are all priority encoders. In this type of encoder, a priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded. We will illustrate the concept of priority encoding with the help of an example. Let us assume that the octal-to-binary encoder described in the previous paragraph has an input priority for higher-order digits. Let us also assume that input lines D_2, D_4 and D_7 are all simultaneously in logic '1' state. In that case, only D_7 will be encoded and the output will be 111. The truth table of such a priority

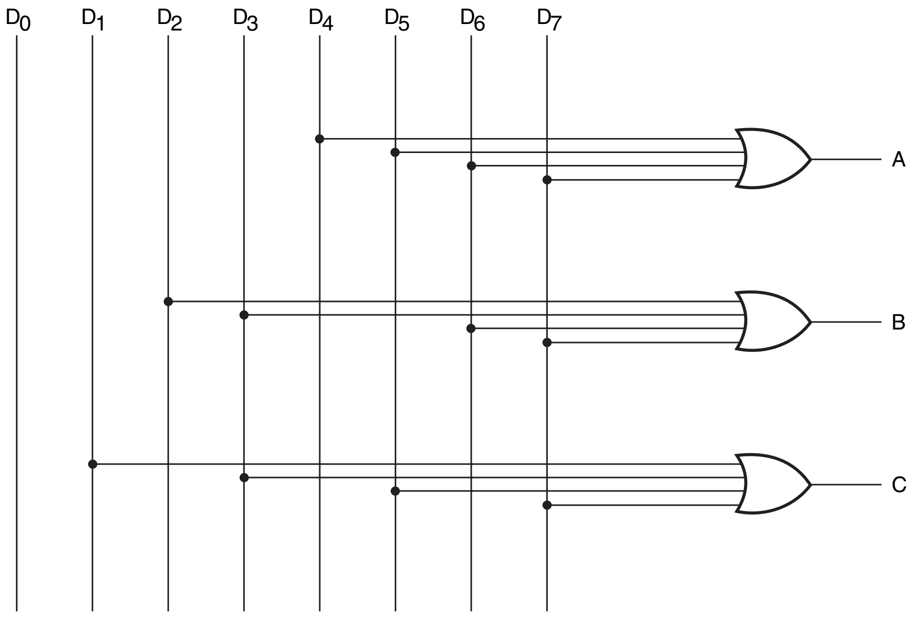


Figure 8.15 Octal-to-binary encoder.

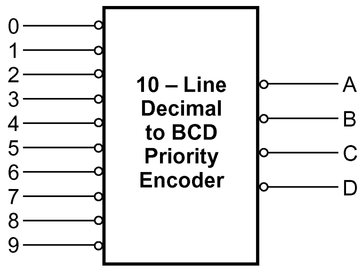
Table 8.8 Truth table of an encoder.

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

encoder will then be modified to what is shown in Table 8.9. Looking at the last row of the table, it implies that, if $D_7 = 1$, then, irrespective of the logic status of other inputs, the output is 111 as D_7 will only be encoded. As another example, Fig. 8.16 shows the logic symbol and truth table of a 10-line decimal to four-line BCD encoder providing priority encoding for higher-order digits, with digit 9 having the highest priority. In the functional table shown, the input line with highest priority having a LOW on it is encoded irrespective of the logic status of the other input lines.

Table 8.9 Priority encoder.

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1
X	X	1	0	0	0	0	0	0	1	0
X	X	X	1	0	0	0	0	0	1	1
X	X	X	X	1	0	0	0	1	0	0
X	X	X	X	X	1	0	0	1	0	1
X	X	X	X	X	X	1	0	1	1	0
X	X	X	X	X	X	X	1	1	1	1



Inputs										Outputs			
0	1	2	3	4	5	6	7	8	9	D	C	B	A
X	X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	X	0	1	1	1	1	1	1	1	1	0	0
X	X	0	1	1	1	1	1	1	1	1	1	0	1
X	0	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 8.16 10-line decimal to four-line BCD priority encoder.

Some of the encoders available in IC form provide additional inputs and outputs to allow expansion. IC 74148, which is an eight-line to three -line priority encoder, is an example. ENABLE-IN (EI) and ENABLE-OUT (EO) terminals on this IC allow expansion. For instance, two 74148s can be cascaded to build a 16-line to four-line priority encoder.

Example 8.4

We have an eight-line to three-line priority encoder circuit with $D_0, D_1, D_2, D_3, D_4, D_5, D_6$ and D_7 as the data input lines. the output bits are A (MSB), B and C (LSB). Higher-order data bits have been assigned a higher priority, with D_7 having the highest priority. If the data inputs and outputs are active when LOW, determine the logic status of output bits for the following logic status of data inputs:

(a) All inputs are in logic '0' state.
(b) D_1 to D_4 are in logic '1' state and D_5 to D_7 are in logic '0' state.
(c) D_7 is in logic '0' state. The logic status of the other inputs is not known.

Solution

(a) Since all inputs are in logic '0' state, it implies that all inputs are active. Since D_7 has the highest priority and all inputs and outputs are active when LOW, the output bits are $A = 0$, $B = 0$ and $C = 0$.
(b) Inputs D_5 to D_7 are the ones that are active. among these, D_7 has the highest priority. Therefore, the output bits are $A = 0$, $B = 0$ and $C = 0$.
(c) D_7 is active. Since D_7 has the highest priority, it will be encoded irrespective of the logic status of other inputs. Therefore, the output bits are $A = 0$, $B = 0$ and $C = 0$.

Example 8.5

Design a four-line to two-line priority encoder with active HIGH inputs and outputs, with priority assigned to the higher-order data input line.

Solution

The truth table for such a priority encoder is given in Table 8.10, with D_0, D_1, D_2 and D_3 as data inputs and X and Y as outputs.

The Boolean expressions for the two output lines X and $Y$are given by the equations

$$X = D_2.\overline{D}_3 + D_3 = D_2 + D_3 \tag{8.5}$$

$$Y = D_1.\overline{D}_2.\overline{D}_3 + D_3 = D_1.\overline{D}_2 + D_3 \tag{8.6}$$

Figure 8.17 shows the logic diagram that implements the Boolean functions given in equations (8.5) and (8.6).

Table 8.10 Example 8.5.

D_0	D_1	D_2	D_3	X	Y
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

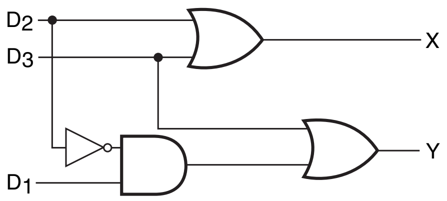
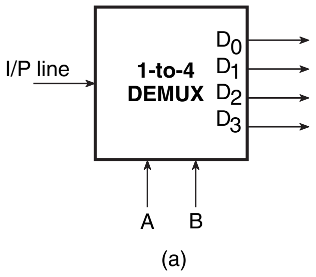


Figure 8.17 Example 8.5.

8.3 Demultiplexers and Decoders

A *demultiplexer* is a combinational logic circuit with an input line, 2^n output lines and n select lines. It routes the information present on the input line to any of the output lines. The output line that gets the information present on the input line is decided by the bit status of the selection lines. A *decoder* is a special case of a demultiplexer without the input line. Figure 8.18(a) shows the circuit representation of a 1-to-4 demultiplexer. Figure 8.18(b) shows the truth table of the demultiplexer when the input line is held HIGH.

A decoder, as mentioned earlier, is a combinational circuit that decodes the information on n input lines to a maximum of 2^n unique output lines. Figure 8.19 shows the circuit representation of 2-to-4, 3-to-8 and 4-to-16 line decoders. If there are some unused or ‘don’t care’ combinations in the n-bit code, then there will be fewer than 2^n output lines. As an illustration, if there are three input lines, it



I/P	Select		O/P			
	A	B	D_0	D_1	D_2	D_3
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

(b)

Figure 8.18 1-to-4 demultiplexer.

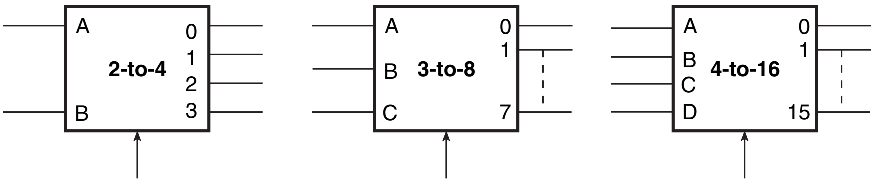


Figure 8.19 Circuit representation of 2-to-4, 3-to-8 and 4-to-16 line decoders.

can have a maximum of eight unique output lines. If, in the three-bit input code, the only used three-bit combinations are 000, 001, 010, 100, 110 and 111 (011 and 101 being either unused or don't care combinations), then this decoder will have only six output lines. In general, if n and m are respectively the numbers of input and output lines, then $m \leq 2^n$.

A decoder can generate a maximum of 2^n possible minterms with an n-bit binary code. In order to illustrate further the operation of a decoder, consider the logic circuit diagram in Fig. 8.20. This logic circuit, as we will see, implements a 3-to-8 line decoder function. This decoder has three inputs designated as A, B and C and eight outputs designated as D_0, D_1, D_2, D_3, D_4, D_5, D_6 and D_7. From the truth table given along with the logic diagram it is clear that, for any given input combination, only one of the eight outputs is in logic '1' state. Thus, each output produces a certain minterm that corresponds to the binary number currently present at the input. In the present case, D_0, D_1, D_2, D_3, D_4, D_5, D_6 and D_7 respectively represent the following minterms:

$$D_0 \rightarrow \overline{A}.\overline{B}.\overline{C}, D_1 \rightarrow \overline{A}.\overline{B}.C, D_2 \rightarrow \overline{A}.B.\overline{C}, D_3 \rightarrow \overline{A}.B.C$$

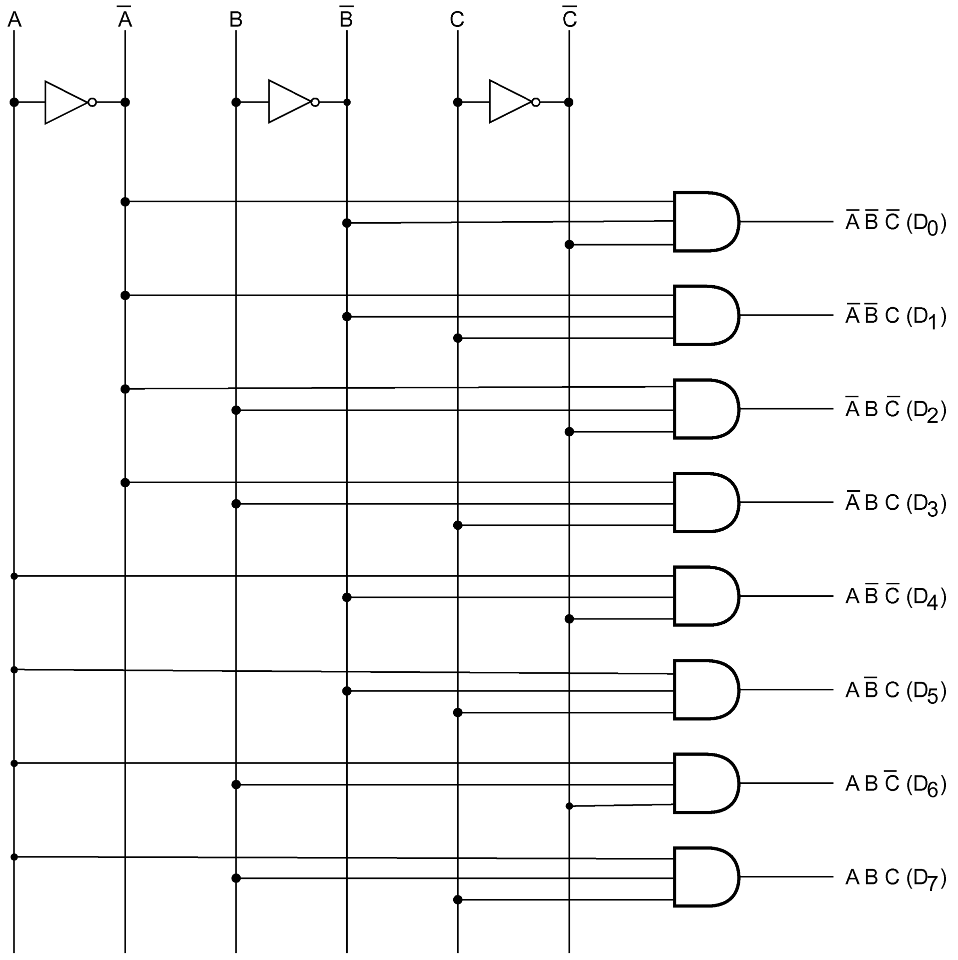
$$D_4 \rightarrow \overline{A}.\overline{B}.\overline{C}, D_5 \rightarrow A.\overline{B}.C, D_6 \rightarrow A.B.\overline{C}, D_7 \rightarrow A.B.C$$

8.3.1 Implementing Boolean Functions with Decoders

A decoder can be conveniently used to implement a given Boolean function. The decoder generates the required minterms and an external OR gate is used to produce the sum of minterms. Figure 8.21 shows the logic diagram where a 3-to-8 line decoder is used to generate the Boolean function given by the equation

$$Y = A.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.B.C + \overline{A}.\overline{B}.\overline{C} \tag{8.7}$$

In general, an n-to-2^n decoder and m external OR gates can be used to implement any combinational circuit with n inputs and m outputs. We can appreciate that a Boolean function with a large number of minterms, if implemented with a decoder and an external OR gate, would require an OR gate with an equally large number of inputs. Let us consider the case of implementing a four-variable Boolean function with 12 minterms using a 4-to-16 line decoder and an external OR gate. The OR gate here needs to be a 12-input gate. In all such cases, where the number of minterms in a given Boolean function with n variables is greater than $2^n/2$ (or 2^{n-1}), the complement Boolean function will have fewer minterms. In that case it would be more advantageous to do NORing of minterms of the complement Boolean function using a NOR gate rather than doing ORing of the given function using an OR gate. The output will be nothing but the given Boolean function.



A
$\bar{A}$
B
$\bar{B}$
C
$\bar{C}$
$\bar{A}\,\bar{B}\,\bar{C}$ (D_0)
$\bar{A}\,\bar{B}\,C$ (D_1)
$\bar{A}\,B\,\bar{C}$ (D_2)
$\bar{A}\,B\,C$ (D_3)
$A\,\bar{B}\,\bar{C}$ (D_4)
$A\,\bar{B}\,C$ (D_5)
$A\,B\,\bar{C}$ (D_6)
$A\,B\,C$ (D_7)

INPUTS			OUTPUTS							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Figure 8.20 Logic diagram of a 3-to-8 line decoder.

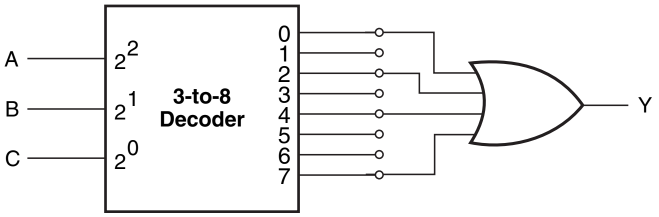


Figure 8.21 Implementing Boolean functions with decoders.

8.3.2 Cascading Decoder Circuits

There can possibly be a situation where the desired number of input and output lines is not available in IC decoders. More than one of these devices of a given size may be used to construct a decoder that can handle a larger number of input and output lines. For instance, 3-to-8 line decoders can be used to construct 4-to-16 or 5-to-32 or even larger decoder circuits. The basic steps to be followed to carry out the design are as follows:

1. If n is the number of input lines in the available decoder and N is the number of input lines in the desired decoder, then the number of individual decoders required to construct the desired decoder circuit would be 2^{N-n}.
2. Connect the less significant bits of the input lines of the desired decoder to the input lines of the available decoder.
3. The left-over bits of the input lines of the desired decoder circuit are used to enable or disable the individual decoders.
4. The output lines of the individual decoders together constitute the output lines, with the outputs of the less significant decoder constituting the less significant output lines and those of the higher–order decoders constituting the more significant output lines. The concept is further illustrated in solved example 8.8, which gives the design of a 4-to-16 decoder using 3-to-8 decoders.

Example 8.6

Implement a full adder circuit using a 3-to-8 line decoder.

Solution

A decoder with an OR gate at the output can be used to implement the given Boolean function. The decoder should at least have as many input lines as the number of variables in the Boolean function to be implemented. The truth table of the full adder is given in Table 8.11, and Fig. 8.22 shows the hardware implementation.

From the truth table, Boolean functions for SUM and CARRY outputs are given by the following equations:

$$\text{Sum output } S = \Sigma\ 1, 2, 4, 7 \tag{8.8}$$

$$\text{Carry output } C_o = \Sigma\ 3, 5, 6, 7 \tag{8.9}$$

Table 8.11 Example 8.6.

A	B	C	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

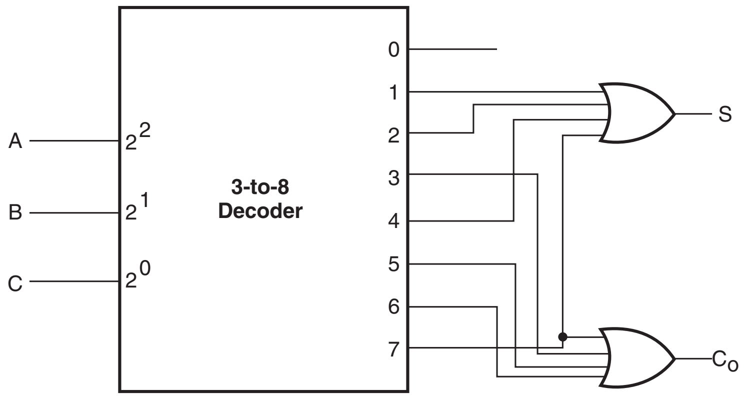


Figure 8.22 Example 8.6.

Example 8.7

A combinational circuit is defined by $F = \Sigma$ 0, 2, 5, 6, 7. Hardware implement the Boolean function F with a suitable decoder and an external OR/NOR gate having the minimum number of inputs.

Solution

The given Boolean function has five three-variable minterms. This implies that the function can be implemented with a 3-to-8 line decoder and a five-input OR gate. Also, $\overline{F}$ will have only three three-variable minterms, which means that F could also be implemented by considering minterms corresponding to the complement function and using a three-input NOR gate at the output. The second option uses a NOR gate with fewer inputs and therefore is used instead. $F = \Sigma$ 0, 2, 5, 6, 7. Therefore, $\overline{F} = \Sigma$ 1, 3, 4.

Figure 8.23 shows the hardware implementation of Boolean function F.

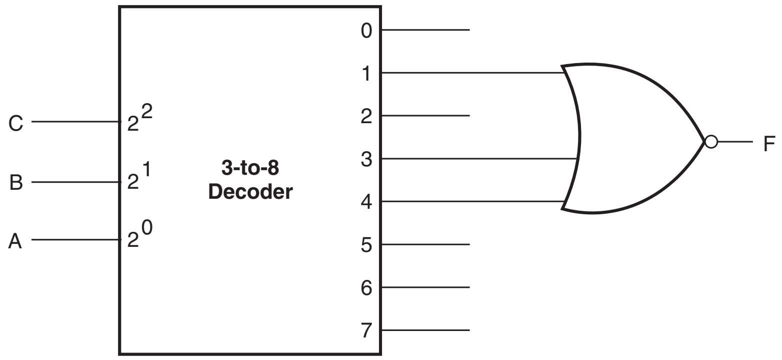


Figure 8.23 Example 8.7.

Example 8.8

Construct a 4-to-16 line decoder with two 3-to-8 line decoders having active LOW ENABLE inputs.

Solution

Let us assume that A (LSB), B, C and D (MSB) are the input variables for the 4-to-16 line decoder. Following the steps outlined earlier, A (LSB), B and C (MSB) will then be the input variables for the two 3-to-8 line decoders. If we recall the 16 possible input combinations from 0000 to 1111 in the case of a 4-to-16 line decoder, we find that the first eight combinations have $D = 0$, with CBA going through 000 to 111. The higher-order eight combinations all have $D = 1$, with CBA going through 000 to 111. If we use the D-bit as the ENABLE input for the less significant 3-to-8 line decoder and the $\overline{D}$-bit as the ENABLE input for the more significant 3-to-8 line decoder, the less significant 3-to-8 line decoder will be enabled for the less significant eight of the 16 input combinations, and the more significant 3-to-8 line decoder will be enabled for the more significant of the 16 input combinations. Figure 8.24 shows the hardware implementation. One of the output lines D_0 to D_{15} is activated as the input bit sequence $DCBA$ goes through 0000 to 1111.

Example 8.9

Figure 8.25 shows the logic symbol of IC 74154, which is a 4-to-16 line decoder/demultiplexer. The logic symbol is in ANSI/IEEE format. Determine the logic status of all 16 output lines for the following conditions:

(a) D = HIGH, C = HIGH, B = LOW, A = HIGH, $\overline{G_1}$ *= LOW and* $\overline{G_2}$ *= LOW.*
(b) D = HIGH, C = HIGH, B = LOW, A = HIGH, $\overline{G_1}$ *= HIGH and* $\overline{G_2}$ *= HIGH.*
(c) D = HIGH, C = HIGH, B = LOW, A = HIGH, $\overline{G_1}$ *= HIGH and* $\overline{G_2}$ *= HIGH.*

Solution

It is clear from the given logic symbol that the device has active HIGH inputs, active LOW outputs and two active LOW ENABLE inputs. Also, both ENABLE inputs need to be active for the decoder to function owing to the indicated ANDing of the two ENABLE inputs.

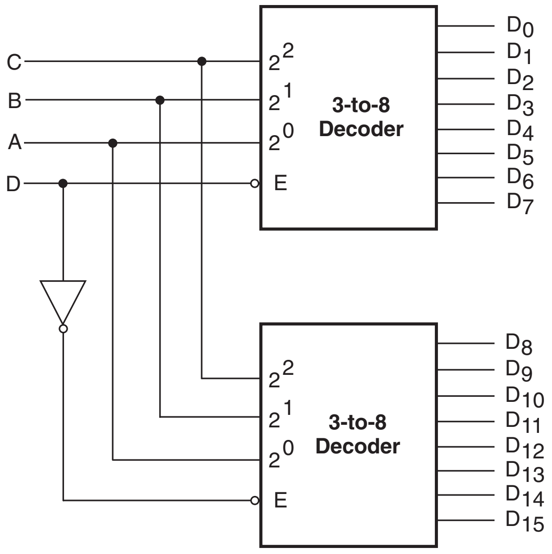


Figure 8.24 Example 8.8.

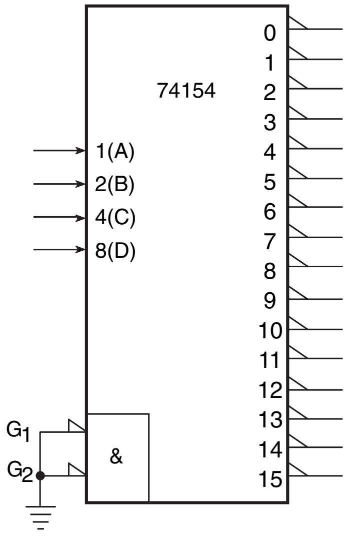


Figure 8.25 Example 8.9.

(a) Since both ENABLE inputs are active, the decoder outputs will therefore be active depending upon the logic status of the input lines. For the given logic status of the input lines, decoder output line 13 will be active and therefore LOW. All other output lines will be inactive and therefore in the logic HIGH state.
(b) Since neither ENABLE input is active, all decoder outputs will be inactive and in the logic HIGH state.
(c) The same as (b).

Example 8.10

The decoder of example 8.9 is to be used as a 1-of-16 demultiplexer. A logically compatible pulsed waveform is to be switched between output line 9 and line 15 when the logic status of an external control input is LOW and HIGH respectively. Draw the logic diagram indicating the logic status of ENABLE inputs and DCBA inputs and the point of application of the pulsed waveform.

Solution

Figure 8.26 shows the logic diagram. When the external control input is in the logic LOW state, $D =$ HIGH, $C =$ LOW, $B =$ LOW and $A =$ HIGH. This means that output line 9 is activated. When the external control input is in the logic HIGH state, $D =$ HIGH, $C =$ HIGH, $B =$ HIGH and $A =$ HIGH. This means that output line 15 is activated. In the logic diagram shown in Fig. 8.26, the two ENABLE inputs are tied together and the pulsed waveform is applied to a common point. This means that either both ENABLE inputs are active (when the input waveform is in the logic LOW state) or inactive (when the input waveform is in the logic HIGH state). Thus, when the input waveform is in the logic LOW state, output line 9 will be in the logic LOW state and all other output lines will be in the logic HIGH state provided the external control input is also in the logic LOW state. If the external

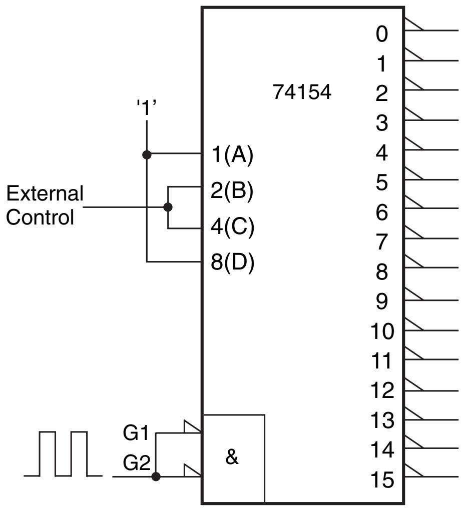


Figure 8.26 Example 8.10.

control input is in the logic HIGH state, logic LOW in the input waveform appears at output line 15. In essence, the logic status of the input waveform is reproduced at either line 9 or line 15, depending on whether the external control signal is LOW or HIGH.

8.4 Application-Relevant Information

Table 8.12 lists commonly used IC type numbers used as multiplexers, encoders, demultiplexers and decoders. Application-relevant information such as the pin connection diagram, truth table, etc., in respect of the more popular of these type numbers is given in the companion website.

Table 8.12 Commonly used IC type numbers used as multiplexers, encoders, demultiplexers and decoders.

IC Type number	Function	Logic family
7442	1-of-10 decoder	TTL
74138	1-of-8 decoder/demultiplexer	TTL
74139	Dual 1-of-4 decoder/demultiplexer	TTL
74145	1-of-10 decoder/driver (open collector)	TTL
74147	10-line to four-line priority encoder	TTL
74148	Eight-line to three-line priority encoder	TTL
74150	16-input multiplexer	TTL
74151	Eight-input multiplexer	TTL
74152	Eight-input multiplexer	TTL
74153	Dual four-input multiplexer	TTL
74154	4-of-16 decoder/demultiplexer	TTL
74155	Dual 1-of-4 decoder/demultiplexer	TTL
74156	Dual 1-of-4 decoder/demultiplexer (open collector)	TTL
74157	Quad two-input noninverting multiplexer	TTL
74158	Quad two-input inverting multiplexer	TTL
74247	BCD to seven-segment decoder/driver (open collector)	TTL
74248	BCD to seven-segment decoder/driver with Pull-ups	TTL
74251	Eight-input three-state multiplexer	TTL
74253	Dual four-input three-state multiplexer	TTL
74256	Dual four-bit addressable latch	TTL
74257	Quad two-input non-inverting three-state multiplexer	TTL
74258	Quad two-input inverting three-state multiplexer	TTL
74259	Eight-bit addressable latch	TTL
74298	Dual two-input multiplexer with output latches	TTL
74348	Eight-line to three-line priority encoder (three-state)	TTL
74353	Dual four-input multiplexer	TTL
74398	Quad two-input multiplexer with output register	TTL
74399	Quad two-input multiplexer with output register	TTL
4019	Quad two-input multiplexer	CMOS
4028	1-of-10 decoder	CMOS
40147	10-line to four-line BCD priority encoder	CMOS
4511	BCD to seven-segment latch/decoder/driver	CMOS
4512	Eight-input three-state multiplexer	CMOS
4514	1-of-16 decoder/demultiplexer with input latch	CMOS

(*continued overleaf*)

Table 8.12 *(continued).*

IC Type number	Function	Logic family
4515	1-of-16 decoder/demultiplexer with input latch	CMOS
4532	Eight-line to three-line priority encoder	CMOS
4539	Dual four-input multiplexer	CMOS
4543	BCD to seven-segment latch/decoder/driver for LCD displays	CMOS
4555	Dual 1-of-4 decoder/demultiplexers	CMOS
4556	Dual 1-of-4 decoder/demultiplexers	CMOS
4723	Dual four-bit addressable latch	CMOS
4724	Eight-bit addressable latch	CMOS
10132	Dual two-input multiplexer with latch and common reset	ECL
10134	Dual multiplexer with latch	ECL
10158	Quad two-input multiplexer (non-inverting)	ECL
10159	Quad two-input multiplexer (inverting)	ECL
10161	3-to-8 line decoder (LOW)	ECL
10162	3-to-8 line decoder (HIGH)	ECL
10164	Eight-line multiplexer	ECL
10165	Eight-input priority encoder	ECL
10171	Dual 2-to-4 line decoder (LOW)	ECL
10172	Dual 2-to-4 line decoder (HIGH)	ECL
10173	Quad two-input multiplexer/latch	ECL
10174	Dual 4-to-1 multiplexer	ECL

Review Questions

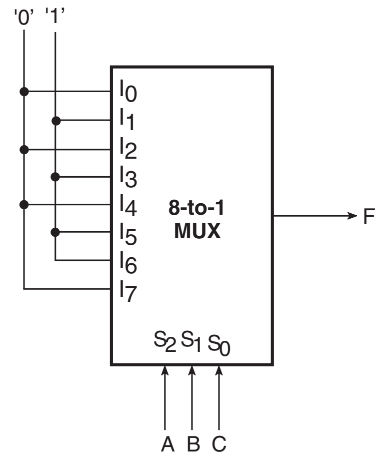
1. What is a multiplexer circuit? Briefly describe one or two applications of a multiplexer?
2. Is it possible to enhance the capability of an available multiplexer in terms of the number of input lines it can handle by using more than one device? If yes, briefly describe the procedure to do so, with the help of an example.
3. What is an encoder? How does a priority encoder differ from a conventional encoder? With the help of a truth table, briefly describe the functioning of a 10-line to four-line priority encoder with active LOW inputs and outputs and priority assigned to the higher-order inputs.
4. What is a demultiplexer and how does it differ from a decoder? Can a decoder be used as a demultiplexer? If yes, from where do we get the required input line?
5. Briefly describe how we can use a decoder optimally to implement a given Boolean function? Illustrate your answer with the help of an example.
6. Draw truth tables for the following:

   (a) an 8-to-1 multiplexer with active LOW inputs and an active LOW ENABLE input;
   (b) a four-line to 16-line decoder with active HIGH inputs and active LOW outputs and an active LOW ENABLE input;
   (c) an eight-line to three-line priority encoder with active LOW inputs and outputs and an active LOW ENABLE input.

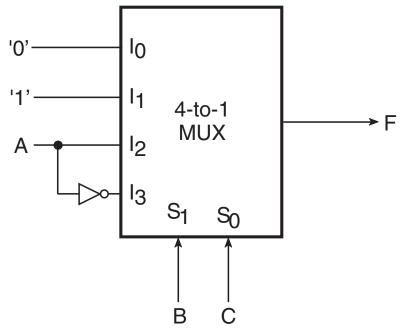
Problems

1. Implement the three-variable Boolean function $F(A, B, C) = \overline{A}.C + A.\overline{B}.C + A.B.\overline{C}$ using (i) an 8-to-1 multiplexer and (ii) a 4-to-1 multiplexer.

(i) Fig. 8.27(a); (ii) Fig. 8.27(b)



(a)



(b)

Figure 8.27 Problem 1.

2. Design a 32-to-1 multiplexer using 8-to-1 multiplexers having an active LOW ENABLE input and a 2-to-4 decoder.

Fig. 8.28

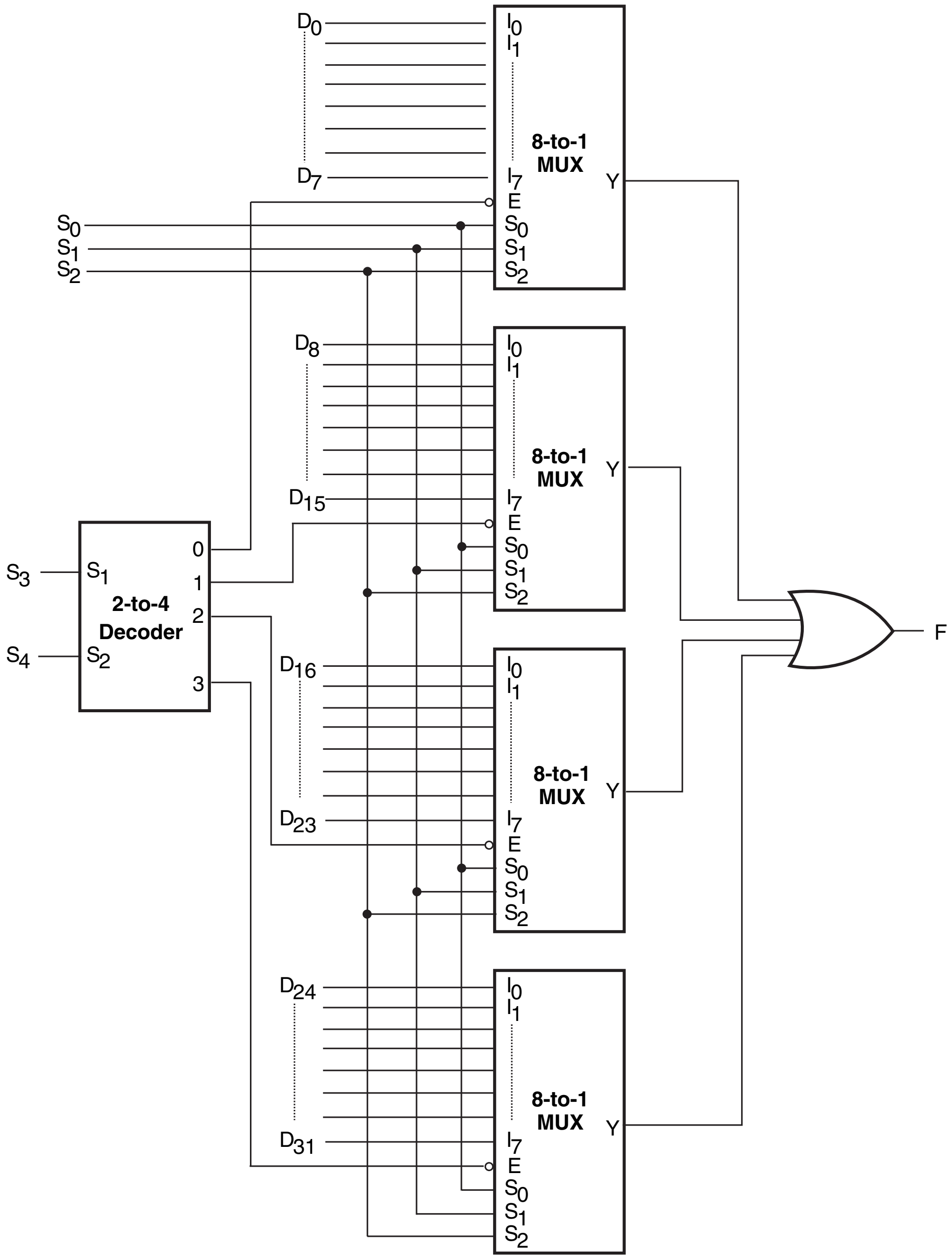


Figure 8.28 Answer to problem 2.

3. Determine the function performed by the combinational circuit of Fig. 8.29.

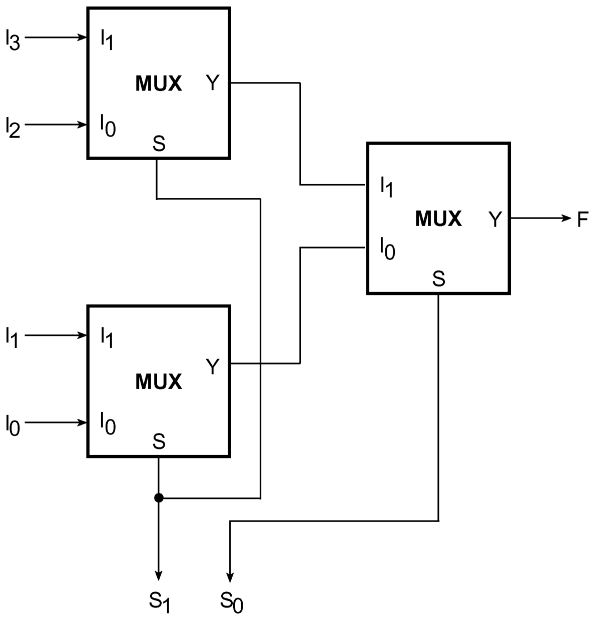


Figure 8.29 Problem 3.

4-to-1 multiplexer

4. Implement a full subtractor combinational circuit using a 3-to-8 decoder and external NOR gates.

Fig. 8.30

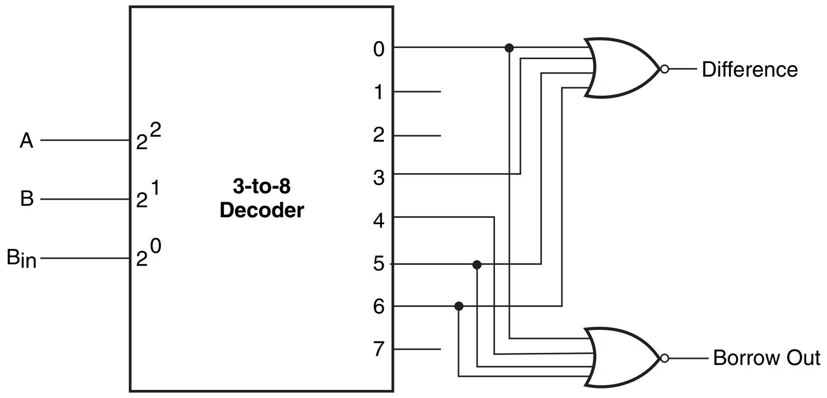


Figure 8.30 Answer to problem 4.

Further Reading

1. Floyd, T. L. (2005) *Digital Fundamentals*, Prentice-Hall Inc., USA.
2. Tokheim, R. L. (1994) *Schaum's Outline Series of Digital Principles*, McGraw-Hill Companies Inc., USA.
3. Tocci, R. J. (2006) *Digital Systems – Principles and Applications*, Prentice-Hall Inc., NJ, USA.
4. Cook, N. P. (2003) *Practical Digital Electronics*, Prentice-Hall, NJ, USA.
5. Rafiquzzaman, M. (2005) *Fundamentals of Digital Logic and Microcomputer Design*, Wiley-Interscience, New York, USA.
6. Morris Mano, M. and Kime, C. R. (2003) *Logic and Computer Design Fundamentals*, Prentice-Hall Inc., USA.