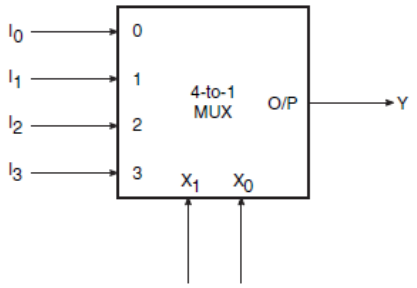
8 MULTIPLEXERS AND DEMULTIPLEXERS

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.

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.



(a)

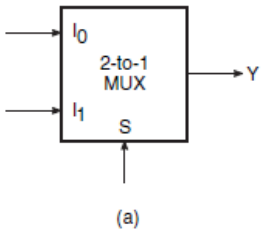
X_1	X_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b)

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

8.1.1 Inside the Multiplexer

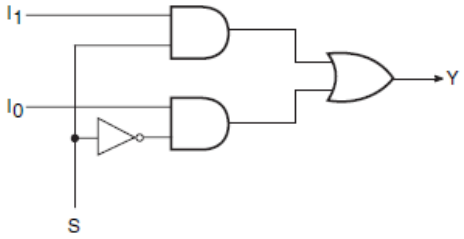
We will briefly describe the type of combinational logic circuit found inside a multiplexer by considering the 2-to-1 multiplexer in Fig. 8.4(a), the functional table of which is shown in Fig. 8.4(b). Figure 8.4(c) shows the possible logic diagram of this multiplexer.



(a)

S	Y
0	I_0
1	I_1

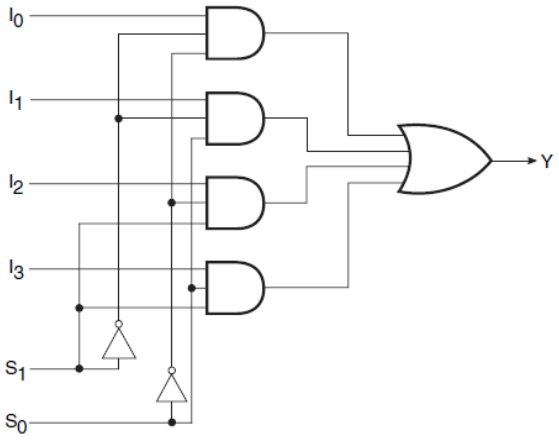
(b)



(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.

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.



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.

The operation of the circuit is governed by the Boolean function (8.1).

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

Similarly, an 8-to-1 multiplexer can be represented by the Boolean function (8.2):

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

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 shows the use of an 8-to-1 MUX for implementing the Boolean function given by the equation

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

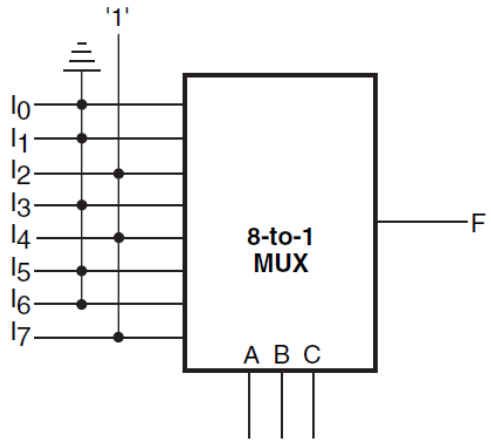


Figure 8.8

Exercise

Implement the product-of-sums Boolean function expressed by Σ1,2,5 by a suitable 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.

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.

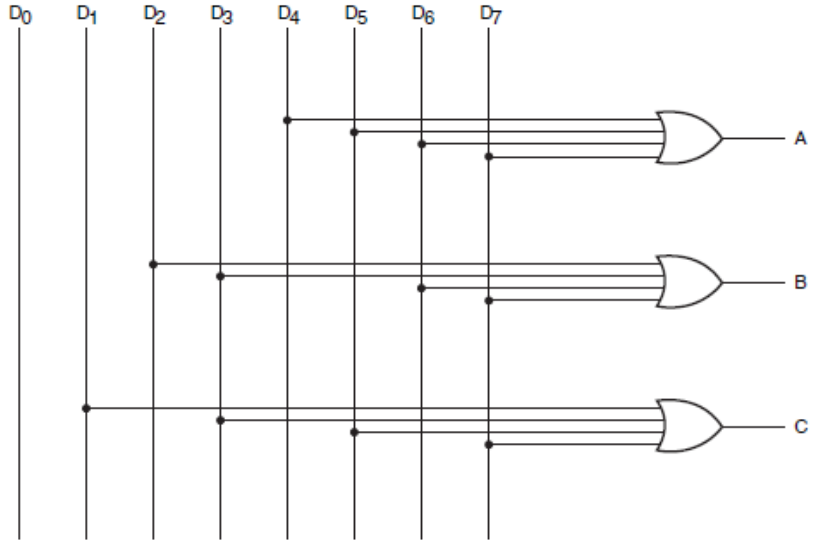


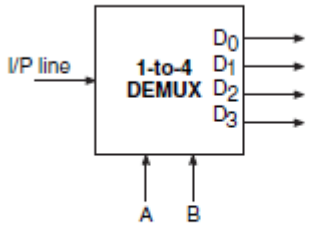
Figure 8.15 Octal-to-binary encoder.

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

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)

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.

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 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.

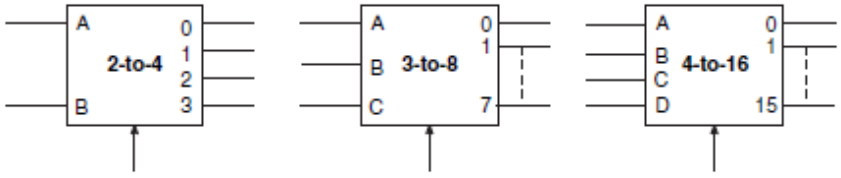
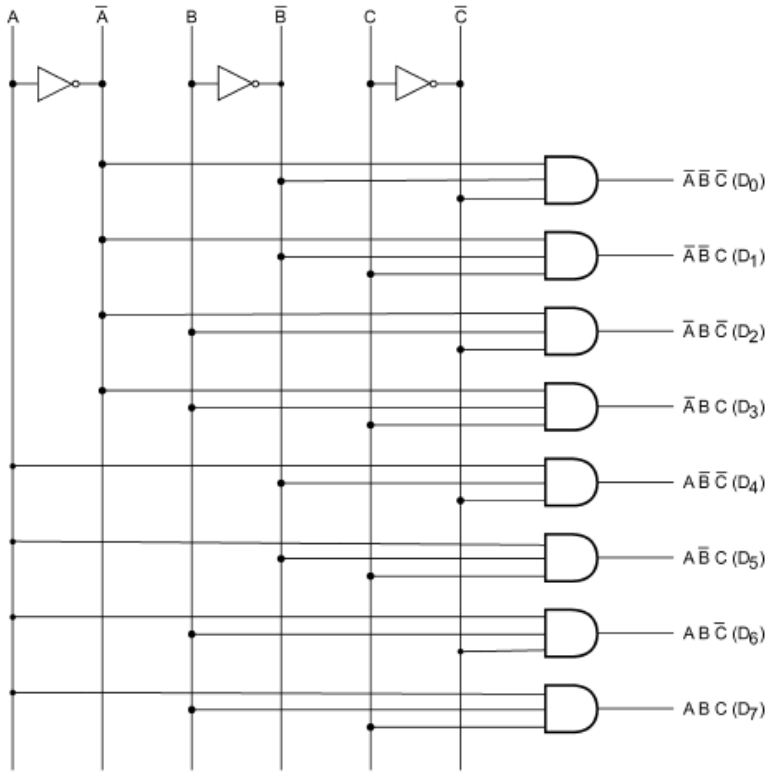


Figure 8.19 Circuit representation of 2-to-4, 3-to-8 and 4-to-16 line decoders.

In order to illustrate further the operation of a decoder, consider the logic circuit diagram in Fig. 8.20.



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.

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

$$Y = A.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.B.C + \overline{A}.\overline{B}.\overline{C} \qquad (8.7)$$

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.

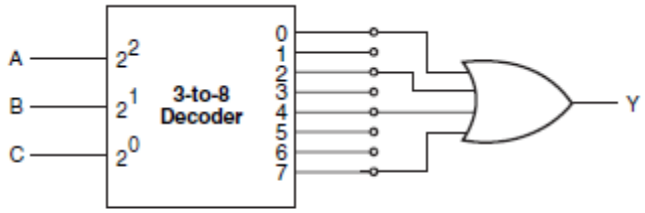


Figure 8.21 Implementing Boolean functions with decoders.