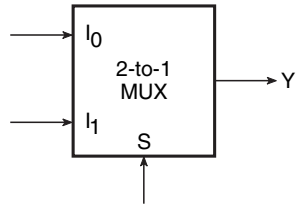
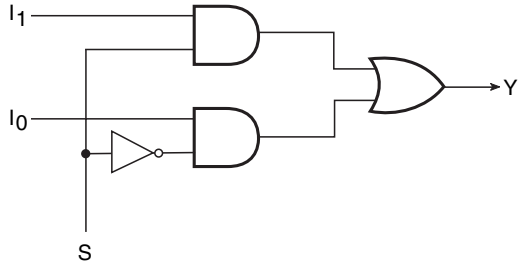


(a)

S	Y
0	I_0
1	I_1

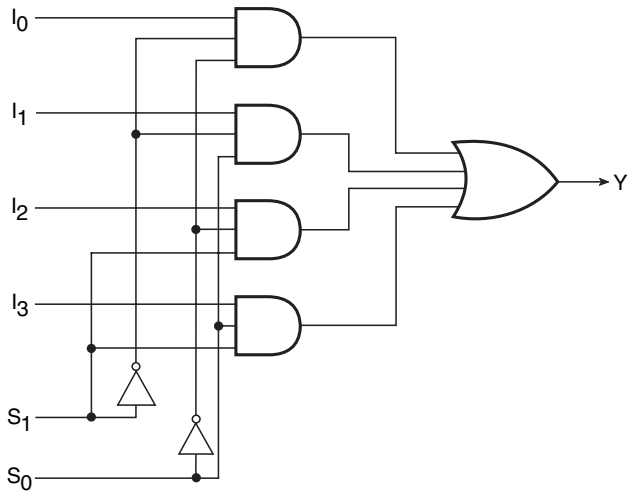
(b)



(c)

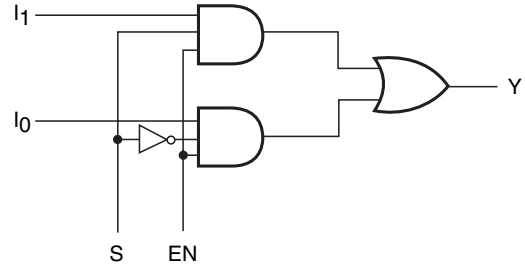
Figure 8.4 (a) 2-to-1 multiplexer circuit representation, (b) 2-to-1 multiplexer truth table and (c) 2-to-1 multiplexer logic diagram.

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.



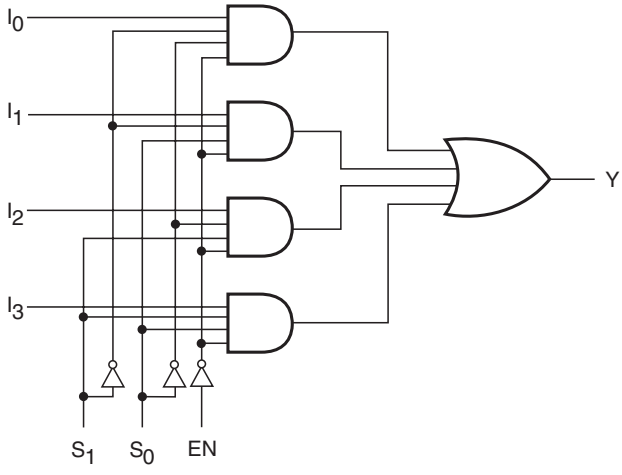
S	EN	Y
X	0	0
0	1	I_0
1	1	I_1

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

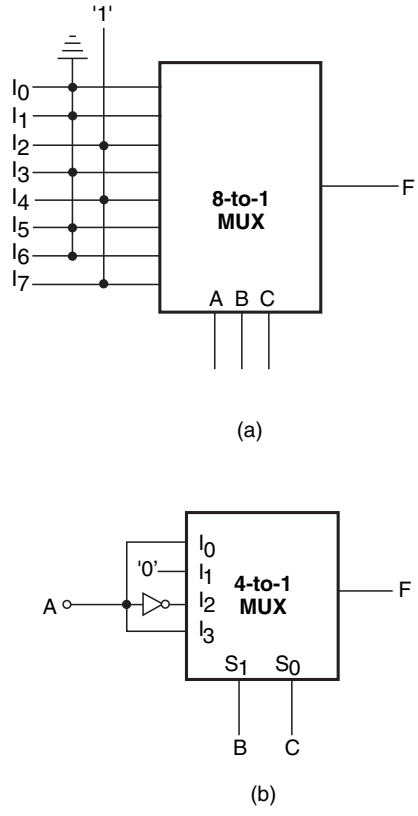


Figure 8.8 Hardware implementation of the Boolean function given by equation (8.3).

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.

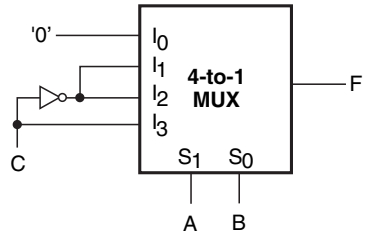


Figure 8.9 Hardware implementation using a 4-to-1 multiplexer.

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

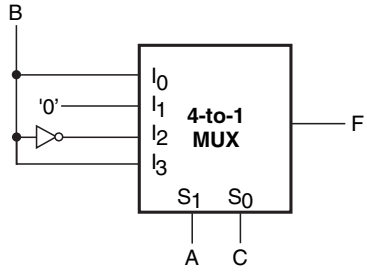


Figure 8.10 Hardware implementation using a 4-to-1 multiplexer.

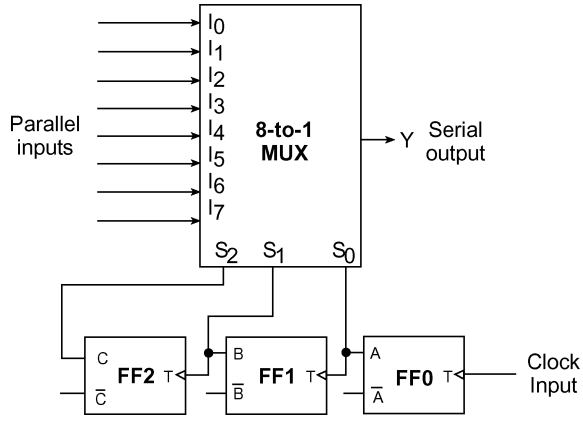


Figure 8.11 Multiplexer for parallel-to-serial conversion.

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 Π1,2,5 by a suitable multiplexer.