functions might be more efficiently implemented using particular modules (comparator, adder, etc.). At the end of the section we compare the various approaches.

Implementation with Decoder and OR Gates

An n-input binary decoder and an OR gate can realize any switching function of n variables. Since the ith output of the decoder corresponds to the minterm $m_i(\underline{x})$, the implementation of a canonical expression is obtained by OR-ing the outputs of the decoder that correspond to the minterms in the expression (Figure 4.3.1). Several functions of the same variables can be generated with one decoder.

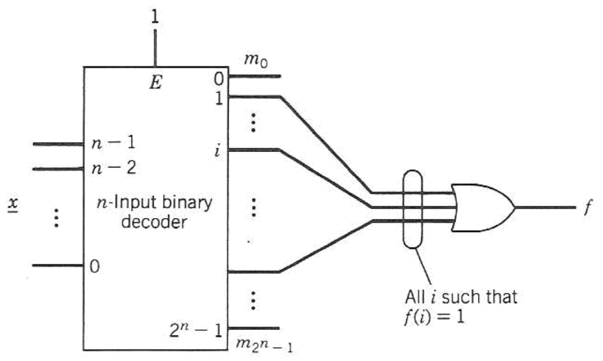


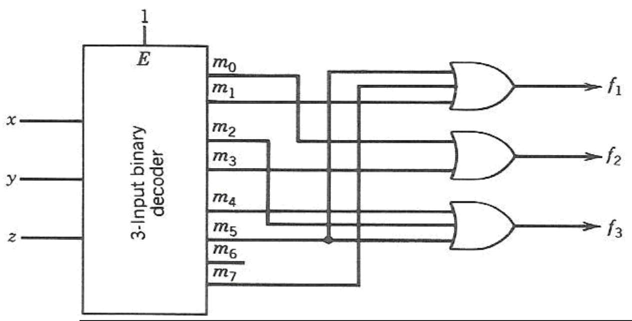
FIGURE 4.3.1 Implementation with decoder and OR gate.

Example 4.3.1

Consider the functions of three variables whose canonical expressions are

$$
\begin{aligned}
f_1(x,y,z) &= \sum m(1,5,7)\\
f_2(x,y,z) &= \sum m(0,3)\\
f_3(x,y,z) &= \sum m(2,4,5)
\end{aligned}
$$

The implementation using a binary 3-input decoder and OR gates is shown in Figure 4.3.2.



If the functions implemented do not share minterms, wired-ORs can be used in some technologies to eliminate the OR gates.

Implementation with Multiplexers

A 2^n input ***multiplexer*** can be used to implement a switching function of n variables. Recall that the output expression of the enabled ($E=1$) multiplexer is

$$y = \sum_{i=0}^{2^n-1} I_i m_i(\underline{s})$$

where I_i is the ith input and $m_i(\underline{s})$ is the ith minterm of the n select variables. To implement a function $f(\underline{x})$, the variables $\underline{x}$ are applied to the select inputs $\underline{s}$ and the data inputs are made equal to the value of the function for the corresponding assignment of input variables, that is, $I_i=f(i)$ (Figure 4.3.3). In other words, a 2^n input multiplexer provides the realization for

$$f(\underline{x}) = \sum_{i=0}^{2^n-1} f(i) m_i(\underline{x})$$

Since any function of n variables can be implemented with a 2^n-input multiplexer, this is a universal module.

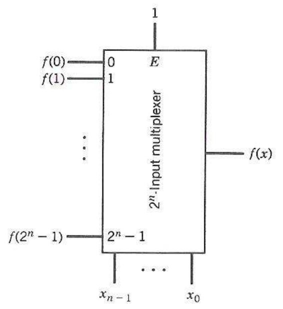


FIGURE 4.3.3 Multiplexer as a universal module.

Example 4.3.2

An implementation of

$$f(x,y,z) = \sum m(0,3,4,7)$$

using an 8-input multiplexer is illustrated in Figure 4.3.4.

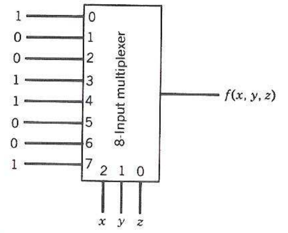


FIGURE 4.3.4 Implementation with multiplexer. ∎

Example 4.3.3

As indicated in the last section, Mead and Conway (1980) have proposed an ALU implementation using multiplexers to obtain the regularity and low interconnection complexity required by a VLSI implementation. In Figure 4.3.5, we show again one bit slice of this ALU. It consists of three multiplexers and Manchester carry-propagate chain. In the case of a ripple-carry adder, the multiplexers generate the carry-propagate, the carry-kill, and the sum while the pass transistor controls the propagation of the carry. Since multiplexers are universal modules, it is possible to extend the functionality of each of them to have an ALU instead of just an adder. Each multiplexer can perform 16 functions, resulting in at most 16^3 functions for the three multiplexers. This does not produce all possible functions of the inputs, since the number of inputs is three (a, b, and c_{in}) and the number of outputs is two (s and c_{out}), which results in a total number of possible functions equal to $4^8=16^4$. The important fact is that many useful arithmetic and logic functions can be realized. An illustration of the control values for some of these functions is given in Table 4.3.1. In the table the values of PC, KC, and RC are integers that identify (as in Table 2.4.3) the two-variable function implemented by the corresponding multiplexer.

In the same way as for the SN74181 module, for the analysis it is more convenient to consider separately specific values of the control variables, rather than getting a unique expression describing the complete function of the module. In a modular analysis, we first obtain expressions for the functions implemented by the multiplexers and then use substitution to obtain the function of the module. For example, for the second entry in Table 4.3.1 we have:

$$p_i = (A_i \oplus B_i)'$$
$$k_i = A_i' B_i$$
$$z_i = c_i \oplus p_i$$