

FIGURE 2-46 Simulation results for the hazard-free circuit in Figure 2-45.

2.6 Building Blocks for Logic Design

Combinational logic encompasses a wide range of functionality and circuit structures, but certain structures and circuits are commonly used in many applications, and it is worthwhile to gain familiarity with them.

2.6.1 NAND–NOR Structures

In complementary metal-oxide semiconductor (CMOS) technology, AND gates and OR gates are not implemented as efficiently as NAND gates and NOR gates. An SOP form or a POS form can always be converted to a NAND logic structure or a NOR logic structure. The NAND gate and NOR gate are universal logic gates—any Boolean function can be realized from only NAND gates or only NOR gates. DeMorgan's laws provide equivalent structures for NAND and NOR gates, shown in Figure 2-47.

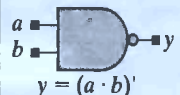
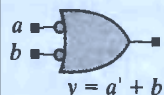
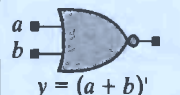
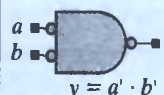
Gate	DeMorgan Equivalent
 $y = (a \cdot b)'$	 $y = a' + b'$
 $y = (a + b)'$	 $y = a' \cdot b'$

FIGURE 2-47 Equivalent circuits that result from DeMorgan's laws.

Networks realizing SOP expressions³ can be transformed by DeMorgan's laws into a circuit that uses only NAND gates and inverters by (1) replacing AND gates by NAND gates in the original AND–OR structure, (2) placing inversion bubbles at the inputs of the OR gates, (3) inserting inverters where needed to match bubbles at the inputs of the OR gates, and (4) substituting a NAND gate for a NOR gate that has inversion bubbles at its inputs. A circuit in POS form⁴ can be transformed into an equivalent circuit that uses only NOR gates and inverters by (1) replacing OR gates by NOR gates, (2) placing inversion bubbles at the inputs of the AND gates, (3) inserting inverters where needed to match bubbles at the inputs of the AND gates, and (4) substituting a NOR gate for a NAND gate that has inversion bubbles at its inputs.

Example 2.35

The function $Y = G + EF + AB'D + CD$ has the two-level circuit realization shown in Figure 2-48(a), which can be transformed into the circuit in Figure 2-48(b) by placing bubbles at the inputs to the OR gate that forms Y and an inverter at input G to match the bubble at the input to the OR gate driven by G . Then DeMorgan's law is applied to replace the OR gate with input bubbles by the NAND gate shown in Figure 2-48(c).

To verify that that circuit of (c) is equivalent to that of (a), we note that

$$\begin{aligned}
 Y &= [(G')(EF)'(AB'D)'(CD)']' \\
 &= (G')' + [(EF)']' + [(AB'D)']' + [(CD)']' \\
 &= G + EF + AB'D + CD
 \end{aligned}$$

End of Example 2.35

³The output of the network must be the output of an OR gate.

⁴The output of the network must be the output of an AND gate.

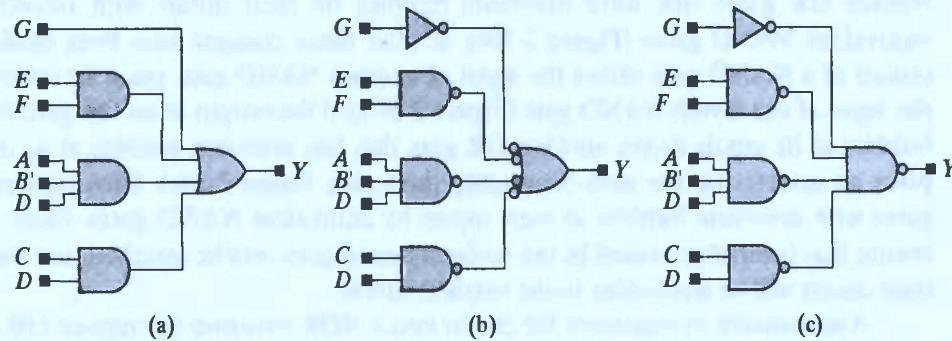


FIGURE 2-48 Circuit transformations to obtain a NAND/Inverter realization of an SOP expression.

Example 2.36

Now consider the POS expression $Y = D(B + C)(A + E + F')(A + G)$. The circuit in Figure 2-49(b) is formed by replacing OR gates with NOR gates, adding inversion bubbles to the input of the AND gate, and adding an inverter to D to match its input bubble. Then the NAND gate with inversion bubbles at its inputs is replaced by an equivalent NOR gate to form the circuit in Figure 2-49(c).

A check reveals that the transformed circuit is equivalent to the original circuit:

$$\begin{aligned}
 Y &= [D' + (B + C)' + (A + E + F')' + (A + G)]' \\
 &= (D')'[(B + C)'][(A + E + F')'][(A + G)']' \\
 &= D(B + C)(A + E + F')(A + G)
 \end{aligned}$$

A circuit whose structure does not consist of alternating AND gates and OR gates can still be transformed into an equivalent structure that uses only NAND gates and inverters or a structure that uses only NOR gates and inverters. To transform such a circuit into a NAND structure: (1) replace all AND gates by NAND gates (Figure 2-50a), (2) place inversion bubbles at the inputs of all OR gates (Figure 2-50b), and (3)

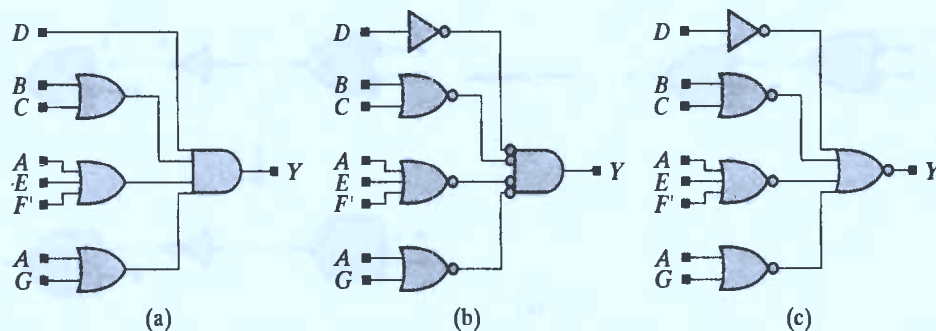


FIGURE 2-49 Circuit transformations to obtain a NOR/Inverter realization of a POS expression.

replace OR gates that have inversion bubbles on their inputs with DeMorgan-equivalent NAND gates (Figure 2-50c). If, after these changes have been made, the output of a NAND gate drives the input of another NAND gate, place an inverter at the input of the driven NAND gate (Figure 2-50d); if the output of an OR gate having bubbles at its inputs drives another OR gate that has inversion bubbles at its inputs, place an inverter on the path connecting them (see Figure 2-50e). Then replace OR gates with inversion bubbles at their inputs by equivalent NAND gates. These steps ensure that inversions caused by the replacement of gates will be matched, and that the final circuit will be equivalent to the original circuit.

Alternatively, to transform the circuit into a NOR structure: (1) replace OR gates by NOR gates (Figure 2-51a), (2) place inversion bubbles at the inputs of any AND gates (Figure 2-51b), and (3) replace AND gates with bubble inputs by DeMorgan-equivalent

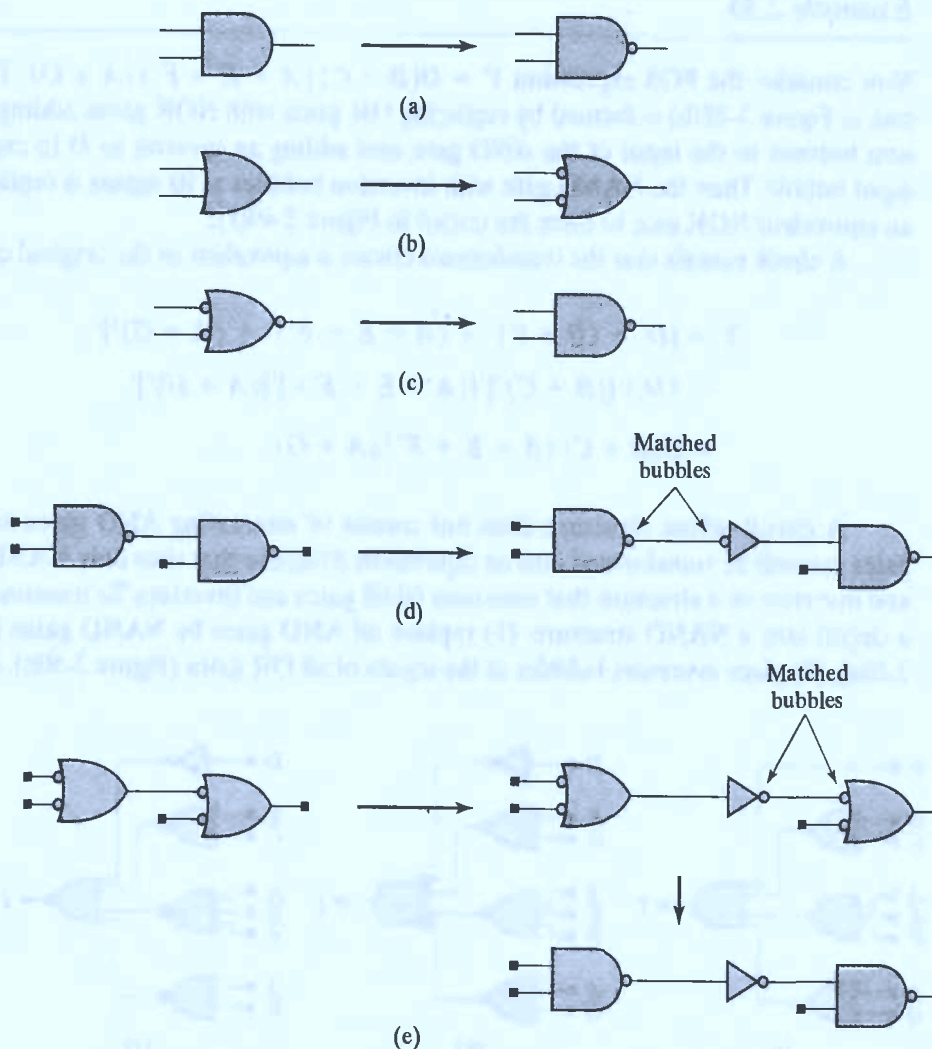


FIGURE 2-50 Circuit transformations for a NAND equivalent circuit.

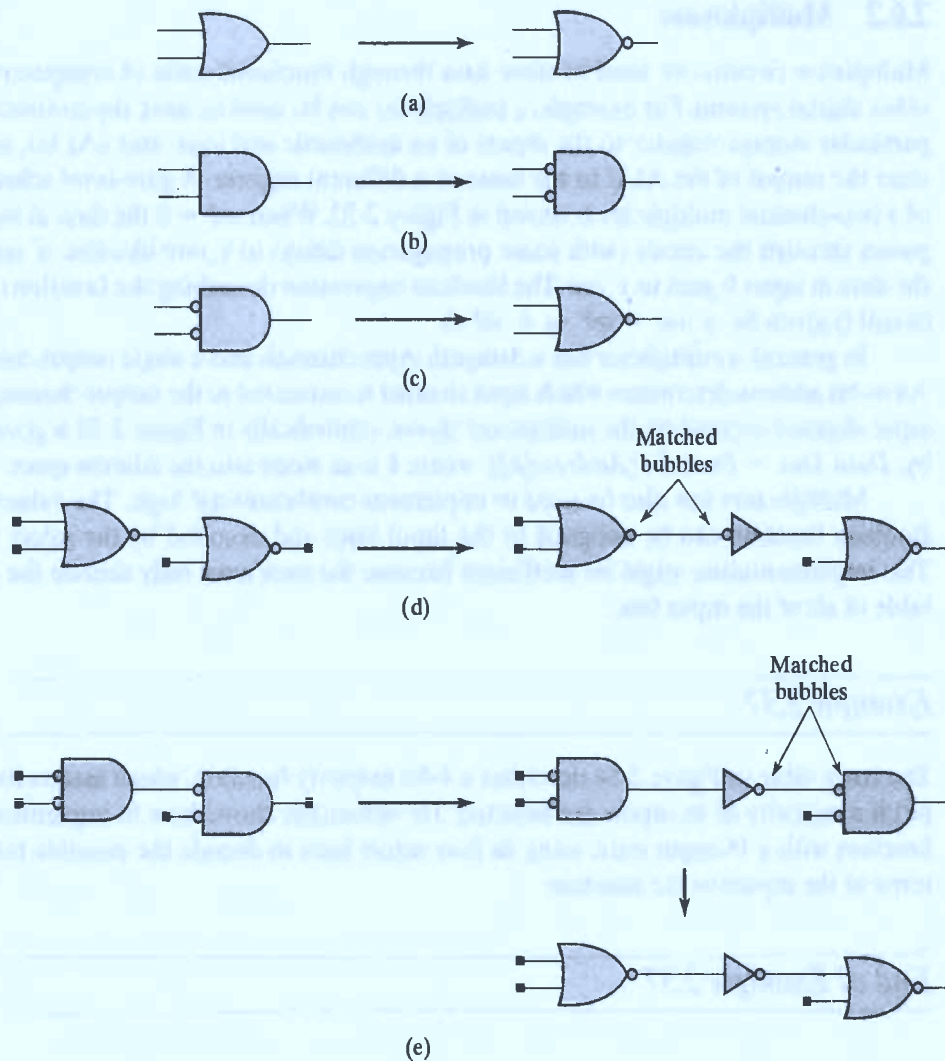


FIGURE 2-51 Circuit transformations for a NOR equivalent circuit.

NOR gates (Figure 2-51c). If, after these changes have been made, the output of a NOR gate drives the input of another NOR gate, place an inverter at the input of the driven NOR gate (Figure 2-51d); if the output of an AND gate with bubbles at its inputs drives another AND gate that has inversion bubbles at its inputs, place an inverter on the path connecting them (see Figure 2-51e). Then replace the AND gate that has inversion bubbles at its inputs with an equivalent NOR gate.

These rules ensure that inversion bubbles will be matched and guarantee that the transformed (NAND or NOR) circuit is equivalent to the original circuit.

End of Example 2.36

2.6.2 Multiplexers

Multiplexer circuits are used to steer data through functional units of computers and other digital systems. For example, a multiplexer can be used to steer the contents of a particular storage register to the inputs of an arithmetic and logic unit (ALU), and to steer the output of the ALU to the same or a different register. A gate-level schematic of a two-channel multiplexer is shown in Figure 2-52. When $sel = 0$ the data at input a passes through the circuit (with some propagation delay) to y_out ; likewise, if $sel = 1$ the data at input b goes to y_out . The Boolean expression describing the function of the circuit is given by: $y_out = sel' \cdot a + sel \cdot b$.

In general, a multiplexer has n datapath input channels and a single output channel. An m -bit address determines which input channel is connected to the output channel. The input channel selected by the multiplexer shown symbolically in Figure 2-53 is governed by: $Data_Out = Data_In[Address[k]]$, where k is an index into the address space.

Multiplexers can also be used to implement combinational logic. The values of a Boolean function can be assigned to the input lines and decoded by the select lines. This implementation might be inefficient because the mux must fully decode the truth table of all of the input bits.

Example 2.37

The truth table in Figure 2-54 describes a 4-bit majority function, which asserts its output if a majority of its inputs are asserted. The schematic shows how to implement the function with a 16-input mux, using its four select lines to decode the possible bit patterns of the inputs to the function

End of Example 2.37

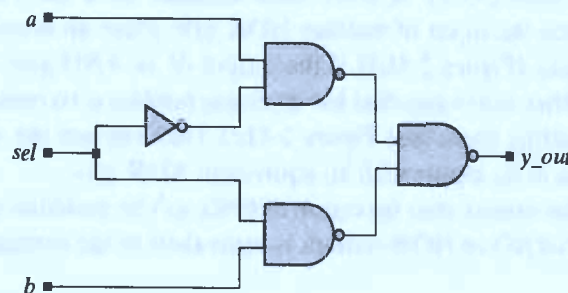


FIGURE 2-52 Gate-level schematic for a two-channel multiplexer circuit.

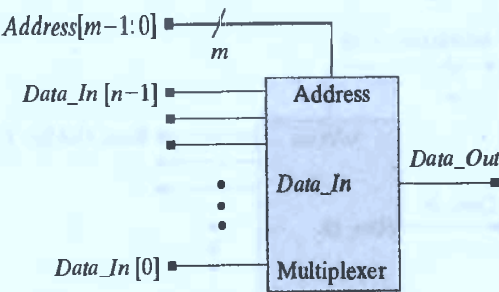


FIGURE 2-53 Schematic symbol for an n -channel multiplexer with an m -bit channel selector address.

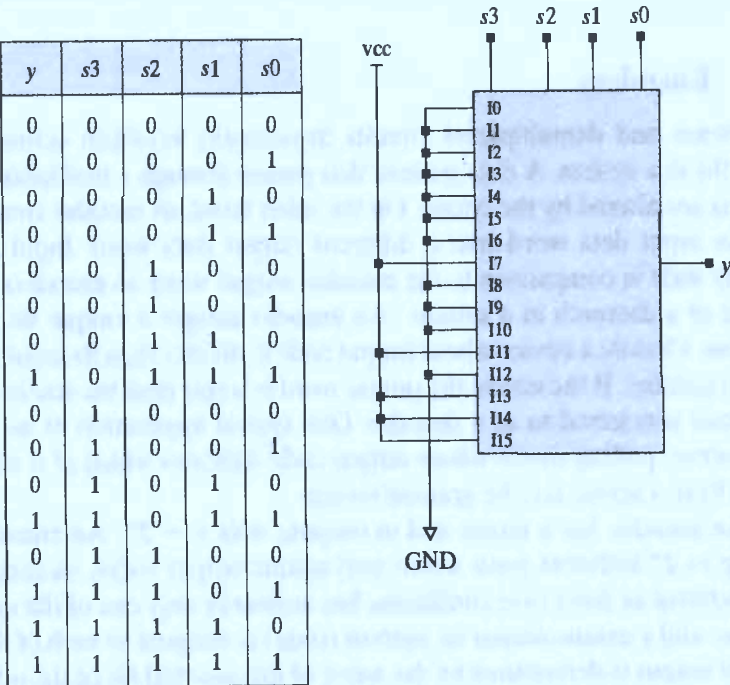


FIGURE 2-54 Truth table and circuit for a 16-input mux implementation of a 4-bit majority function.

2.6.3 Demultiplexers

A demultiplexer circuit implements the reverse functionality of a multiplexer. It has a single input datapath, n output datapaths, and an input m -bit address that determines which of the n outputs is connected to the input. The output channel selected by the demultiplexer shown in Figure 2-55 is determined by $Data_Out[n - 1:0] = Data_In[Address[k]]$, where k is an index into the address space.

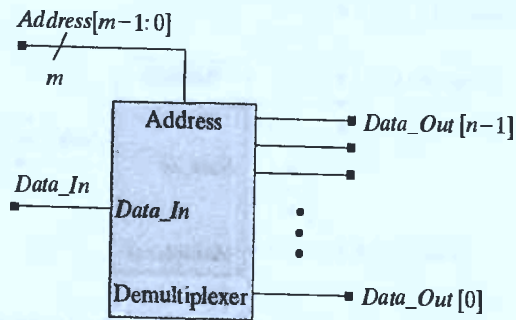


FIGURE 2-55 Gate-level schematic for an n -output demultiplexer circuit having an m -bit destination address.

2.6.4 Encoders

Multiplexer and demultiplexer circuits dynamically establish connectivity between datapaths in a system. A data pattern that passes through a multiplexer or a demultiplexer is not altered by the circuit. On the other hand, an encoder circuit acts to transform an input data word into a different output data word. Input data words are typically wide in comparison to the encoded output word, so encoders serve to reduce the size of a datapath in a system. An encoder assigns a unique bit pattern to each input line. Usually, a device whose output code is smaller than its input code is referred to as an encoder. If the size of the output word is larger than the size of the input word, the circuit is referred to as a decoder. One typical application of an encoder is in a client-server polling circuit whose output code indicates which of n clients requesting service from a server is to be granted service.

An encoder has n inputs and m outputs, with $n = 2^m$. An encoder could transform up to 2^m different input words into unique output codes, treating the remaining input patterns as don't-care conditions, but ordinarily only one of the inputs is asserted at a time, and a unique output bit pattern (code) is assigned to each of the n inputs. The asserted output is determined by the index of the asserted bit of the n -bit binary input word. Block diagram symbols for encoders are shown in Figure 2-56.

Example 2.38

A 5:3 encoder having a 5-bit input word is to generate a 3-bit output code indicating the number of bits that are asserted in the input word. The input words and encoded output bit patterns are shown in Figure 2-57. Boolean logic equations can be derived for each bit of the output word.

End of Example 2.38

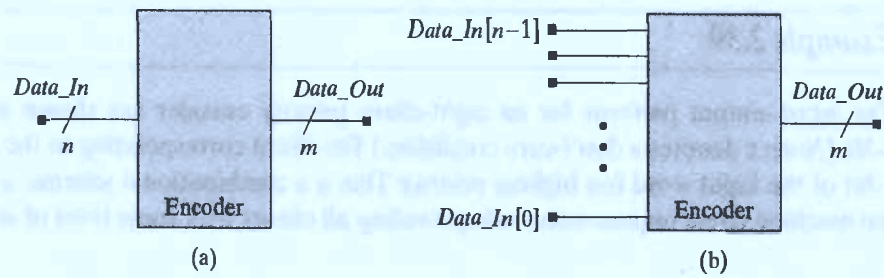


FIGURE 2-56 Schematic symbols for an encoder: (a) an encoder with an n -bit input bus, and (b) an encoder with individual bit-line inputs.

Input	Output	Input	Output
00000	000	10000	001
00001	001	10001	010
00010	001	10010	010
00011	010	10011	011
00100	001	10100	010
00101	010	10101	011
00110	010	10110	011
00111	011	10111	100
01000	100	11000	010
01001	010	11001	011
01010	010	11010	011
01011	011	11011	100
01100	010	11100	011
01101	011	11101	100
01110	011	11110	100
01111	100	11111	101

FIGURE 2-57 Input-output words for a 5:3 encoder that indicates the number of asserted bits in the input word.

2.6.5 Priority Encoder

A priority encoder allows multiple input bits to be asserted simultaneously and uses a priority rule to form an output bit pattern. A priority encoder in a client-server system would identify the client that has the highest priority among multiple clients requesting service.

Example 2.39

The input–output patterns for an eight-client priority encoder are shown in Figure 2-58. (Note: x denotes a don't-care condition.) The client corresponding to the leftmost 1-bit of the input word has highest priority. This is a combinational scheme; a sequential machine could impose some rule providing all clients with some level of service.

Input Word	Output Word
1xxxxxxx	000
01xxxxxx	001
001xxxxx	010
0001xxxx	011
00001xxx	100
000001xx	101
0000001x	110
00000001	111

FIGURE 2-58 Input–output words for an 8:3 priority encoder.

End of Example 2.39

2.6.6 Decoder

A binary decoder interprets an input pattern of bits and forms a unique output word in which only 1 bit is asserted. Decoders are commonly used to extract the opcode from an instruction in a digital computer; row and column address decoders are used to locate a word in memory from its address. Figure 2-59 shows block diagram symbols for

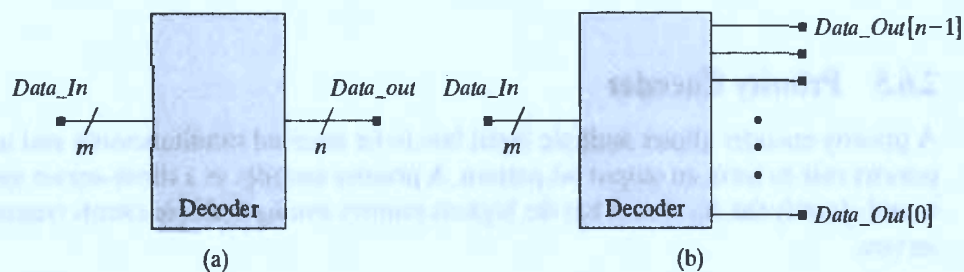


FIGURE 2-59 Block diagram symbols for decoders: (a) decoder with input/output busses, and (b) decoder with an expanded output bus.

a decoder. A binary decoder has m inputs and n outputs, with $n = 2^m$. There are many different possible mappings between input words and output words. An encoder can be built from combinational logic that forms the input–output mapping. (Sequential encoders and decoders are widely used in communication and video transmission circuits.)

Example 2.40

The input–output patterns for an eight-client decoder are shown in Figure 2-60. The arrangement of bits in the output words identifies the client that will be served. This decoder does not resolve contention between multiple clients, and it assumes that only one client at a time will request service.

A binary decoder generates all of the minterms of its inputs. All of the output lines are available to form as many functions of the same inputs as are needed by an application. Binary decoders can be used for small implementations with multiple outputs, but the number of outputs precludes their use in applications with a large number of inputs.

Input Word	Output Word
000	10000000
001	01000000
010	00100000
011	00010000
100	00001000
101	00000100
110	00000010
111	00000001

FIGURE 2-60 The input–output patterns for an eight-client decoder.

End of Example 2.40

Example 2.41

A decoder can be used to implement multiple Boolean functions of the same inputs. The truth table in Figure 2-61 describes f_1 , a majority function, along with some other function, f_2 . Additional logic is used with the decoder to combine its outputs to form the two functions.

End of Example 2.41

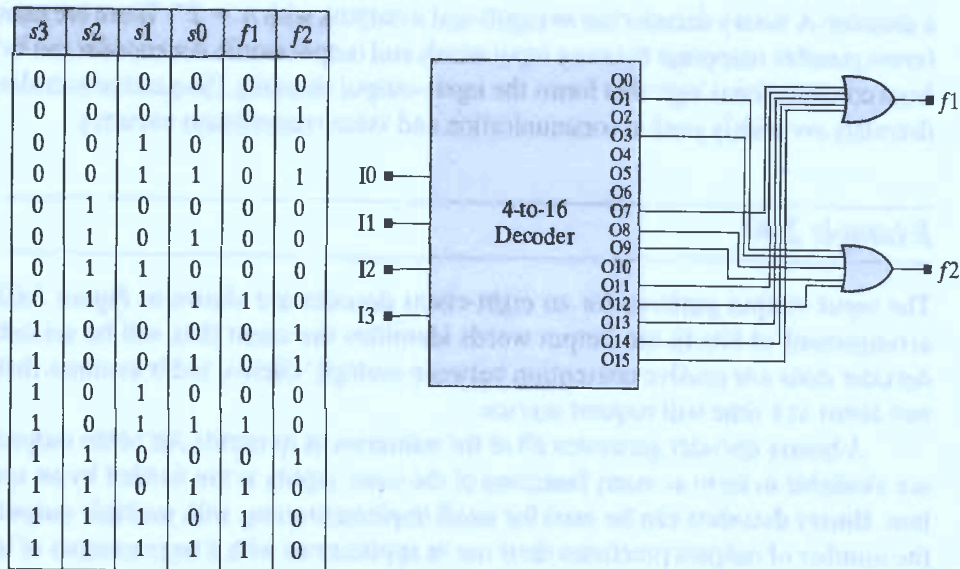


FIGURE 2-61 A truth table for two functions implemented by a single 4-to-16 decoder.

2.6.7 Priority Decoder

A priority decoder can be used in applications in which multiple input codes might imply contention.

Example 2.42

The input code for a client-server system that is to serve eight clients will contain a 1 in any bit position that corresponds to a request from a client for service. The server must determine which of multiple clients is to be served. One simple rule assigns a unique priority to each client. The input-output codes in Figure 2-62 assign the highest priority to the client associated with the leftmost bit of the input code. The table accounts for all possible input patterns. A sequential decoder circuit could base service on other considerations, such as whether a client has been blocked from service by higher-priority clients for too long.

End of Example 2.42

Input Word	Output Word
1xxxxxxx	10000000
01xxxxxx	01000000
001xxxxx	00100000
0001xxxx	00010000
00001xxx	00001000
000001xx	00000100
0000001x	00000010
00000001	00000001

FIGURE 2-62 The input-output patterns for an eight-client priority decoder.

REFERENCES

1. Breuer MA, Friedman AD. *Diagnosis & Design of Reliable Digital Systems*. Rockville, MD: Computer Science Press, 1976.
2. Fabricius ED. *Introduction to VLSI Design*. New York: McGraw-Hill, 1990.
3. Bryant RE. "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, C-35, 677–691, 1986.
4. Tinder RF. *Engineering Digital Design*, 2nd ed. San Diego: Academic Press, 2000.
5. Katz RH. *Contemporary Logic Design*. Redwood City, CA: Benjamin Cummings, 1994.
6. McCluskey EJ. "Minimization of Boolean Functions," *Bell Systems Technical Journal*, 35, 1417–1444, 1956.
7. McCluskey EJ. *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill, 1965.
8. McCluskey EJ. *Logic Design Principles*, Upper Saddle River, NJ: Prentice-Hall, 1986.
9. Wakerly JF. *Digital Design Principles and Practices*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2000.
10. Brayton RK. et al. *Logic Minimization Algorithms for VLSI Synthesis*. Boston: Kluwer, 1984.

PROBLEMS

1. Find the canonical SOP form of the following Boolean function:
 $F(a, b, c) = \Sigma(1, 3, 5, 7)$.
2. Find the canonical POS form of the following Boolean function:
 $F(a, b, c, d) = \Pi(0, 1, 2, 3, 4, 5, 12)$.
3. Express the function $F = a'b + c$ as a sum of minterms.
4. Express the function $F = a'bcd' + a'bcd + a'b'c'd' + a'b'c'd$ as a sum of minterms.
5. Express the function $G = (a'bcd' + a'bcd + a'b'c'd' + a'b'c'd)'$ as a sum of minterms.

6. Find a NAND circuit realization of the function $f = ac' + bcd + a'd$.
7. Find a NOR circuit realization of the function $f = (b + c + d)(a' + b + c)(a' + d)$.
8. Find the complement of the following expressions:
 - a. $ab' + a'b$
 - b. $b + (cd' + e)a'$
 - c. $(a' + b + c)(b' + c')(a + c)$
9. Simplify the following Boolean functions to a minimum number of literals:
 - a. $F = a + a'b$
 - b. $F = a(a' + b)$
 - c. $F = ac + bc' + ab$
10. Using Karnaugh maps, simplify the following Boolean functions:
 - a. $F(a, b, c) = \Sigma(0, 2, 4, 5, 6)$
 - b. $F(a, b, c) = \Sigma(2, 3, 4, 5)$
 - c. $F = bc' + ac' + a'bc + ab$
 - d. $F = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$
 - e. $F = a'b'c' + b'cd' + a'bcd' + ab'c'$
11. Find a NAND gate realization of the following Boolean function: $F(a, b, c) = \Sigma(0, 6)$
12. Using the K-map diagram below:
 - a. Draw a K-map for $f = \Sigma m(0, 4, 6, 8, 9, 11, 12, 14, 15)$

		<i>cd</i>			
		00	01	11	10
<i>b</i>	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}

FIGURE P2-12

- b. Identify the prime implicants of f .
 - c. Identify the essential prime implicants of f .
 - d. Find all minimal expressions of f and identify those that use only essential prime implicants.
13. Design a two-level circuit that implements a 4-bit majority function, that is, the output is a 1 if three or more inputs are asserted.
14. Example 2.37 showed how to implement a 4-bit majority function with a 16-input mux. Show that it is also possible to implement this function with a 8-input mux.