Min: Other Minimization Topics

Min.1 Simplifying Products of Sums

Using the principle of duality, we can minimize product-of-sums expressions by looking at the 0s on a Karnaugh map. Each 0 on the map corresponds to a maxterm in the canonical product of the logic function. The entire process in the preceding subsection can be reformulated in a dual way, including the rules for writing sum terms corresponding to circled sets of 0s, in order to find a *minimal product*. Of course, applying the dual of the rules can be confusing.

minimal product

Fortunately, there's an easier way to find the minimal product for a logic function F, since we already know how to find a minimal sum. The first step is to complement F to obtain F′. Assuming that F is expressed as a minterm list or a truth table, complementing is very easy; the 1s of F′ are just the 0s of F. Next, we find a minimal sum for F′ using the method of the preceding subsection. Finally, we complement the result using the generalized DeMorgan's theorem, which yields a minimal product for (F′)′ = F. (Note that if you simply "add out" the minimal-sum expression for the original function, the resulting product-of-sums expression is not necessarily minimal; for example, see Exercise Min.17.)

In general, to find the lowest-cost two-level realization of a logic function, we have to find both a minimal sum and a minimal product and compare them. If a minimal sum for a logic function has many terms, then a minimal product for the same function may have few terms. As a trivial example of this trade-off, consider a 4-input OR function:

$$\begin{aligned} F &= (W) + (X) + (Y) + (Z) \text{ (a sum of four trivial product terms)} \\ &= (W + X + Y + Z) \text{ (a product of one sum term)} \end{aligned}$$

For a nontrivial example, you're invited to find the minimal product for the function that we minimized in Figure 4-34 on page 219; it has just two sum terms.

The opposite situation is also sometimes true, as trivially illustrated by a 4-input AND:

$$\begin{aligned} F &= (W) \cdot (X) \cdot (Y) \cdot (Z) \text{ (a product of four trivial sum terms)} \\ &= (W \cdot X \cdot Y \cdot Z) \text{ (a sum of one product term)} \end{aligned}$$

A nontrivial example with a higher-cost product-of-sums is the function in Figure 4-30 on page 216.

For some logic functions, both minimal forms have the same cost. For example, consider a 3-input "Exclusive OR" function; both minimal expressions have four terms, and each term has three literals:

$$\begin{aligned} F &= \Sigma_{X,Y,Z}(1,2,4,7) \\ &= (X' \cdot Y' \cdot Z) + (X' \cdot Y \cdot Z') + (X \cdot Y' \cdot Z') + (X \cdot Y \cdot Z) \\ &= (X + Y + Z) \cdot (X + Y' + Z') \cdot (X' + Y + Z') \cdot (X' + Y' + Z) \end{aligned}$$

Still, in most cases, one form or the other will give better results. Looking at both forms is especially useful in PLD-based designs.

Typical PLDs have an AND-OR array corresponding to a sum-of-products form, so you might think that only the minimal sum-of-products is relevant to a PLD-based design. However, most PLDs also have an inverter/buffer at the output of the AND-OR array, which can be programmed to invert or not. Thus, the PLD can utilize the equivalent of the minimal sum by using the AND-OR array to realize the complement of the desired function and then programming the inverter/buffer to invert. Most logic-minimization programs for PLDs automatically find both the minimal sum and the minimal product and select the one that requires fewer terms.

Min.2 "Don't-Care" Input Combinations

Sometimes the specification of a combinational circuit is such that its output doesn't matter for certain input combinations, called *don't-cares*. This may be true because the outputs really don't matter when these input combinations occur, or because these input combinations never occur in normal operation. For example, suppose we wanted to build a prime-number detector whose 4-bit input $N = N_3N_2N_1N_0$ is always a BCD digit; then minterms 10–15 should never occur. A prime BCD-digit detector function may therefore be written as follows:

don't-care

$$F = \Sigma_{N_3,N_2,N_1,N_0}(1,2,3,5,7) + d(10,11,12,13,14,15)$$

The d(...) list specifies the don't-care input combinations for the function, also known as the *d-set*. Here F must be 1 for input combinations in the on-set (1,2,3,5,7), F can have any values for inputs in the d-set (10,11,12,13,14,15), and F must be 0 for all other input combinations (in the 0-set).

d-set

Figure Min-1 shows how to find a minimal sum-of-products realization for the prime BCD-digit detector, including don't-cares. The d's in the map denote the don't-care input combinations. We modify the procedure for circling sets of 1s (prime implicants) as follows:

- Allow d's to be included when circling sets of 1s, to make the sets as large as possible. This reduces the number of variables in the corresponding prime implicants. Two such prime implicants ($N_2 \cdot N_0$ and $N_2' \cdot N_1$) appear in the example.
- Do not circle any sets that contain only d's. Including the corresponding product term in the function would unnecessarily increase its cost. Two such product terms ($N_3 \cdot N_2$ and $N_3 \cdot N_1$) are circled in the example.
- Just a reminder: As usual, do not circle any 0s.

The remainder of the procedure is the same. In particular, we look for distinguished 1-cells and *not* distinguished d-cells, and we include only the corresponding essential prime implicants and any others that are needed to cover

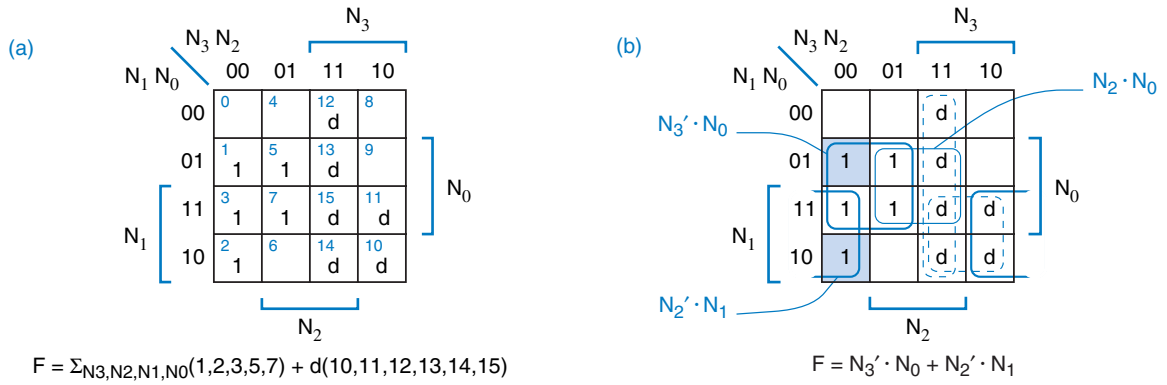


Figure Min-1 Prime BCD-digit detector: (a) initial Karnaugh map; (b) Karnaugh map with prime implicants and distinguished 1-cells.

all the 1s on the map. In Figure Min-1, the two essential prime implicants are sufficient to cover all of the 1s on the map. Two of the d's also happen to be covered, so F will be 1 for don't-care input combinations 10 and 11, and 0 for the other don't-cares.

Some HDLs, including ABEL, provide a means for the designer to specify don't-care inputs, and the logic-minimization program takes these into account when computing a minimal sum.

DON'T-CARES & STATE-MACHINE SYNTHESIS

In Section 7.4.4 we show that designers have a choice of strategies for dealing with unused states in state-machine synthesis. The so-called "minimal-cost" strategy uses don't-care next-states for unused states, and this leads to don't-cares in the Karnaugh maps for the excitation equations.

In the example in the box on page 566, to derive minimal-cost excitation equations, we write "don't-cares" in the next-state entries for the unused states. The colored d's in Figure Min-2 are the result of this choice. The excitation equations obtained from this map are somewhat simpler than we derived using the "minimal-risk" strategy:

$$D1 = 1$$
$$D2 = Q1 \cdot Q3' \cdot A' + Q3 \cdot A + Q2 \cdot B$$
$$D3 = A$$

For a minimal-cost output function, the value of Z is a "don't-care" for the unused states. This leads to an even simpler output function, Z = Q2. The logic diagram for the minimal-cost solution is shown in Figure Min-3.

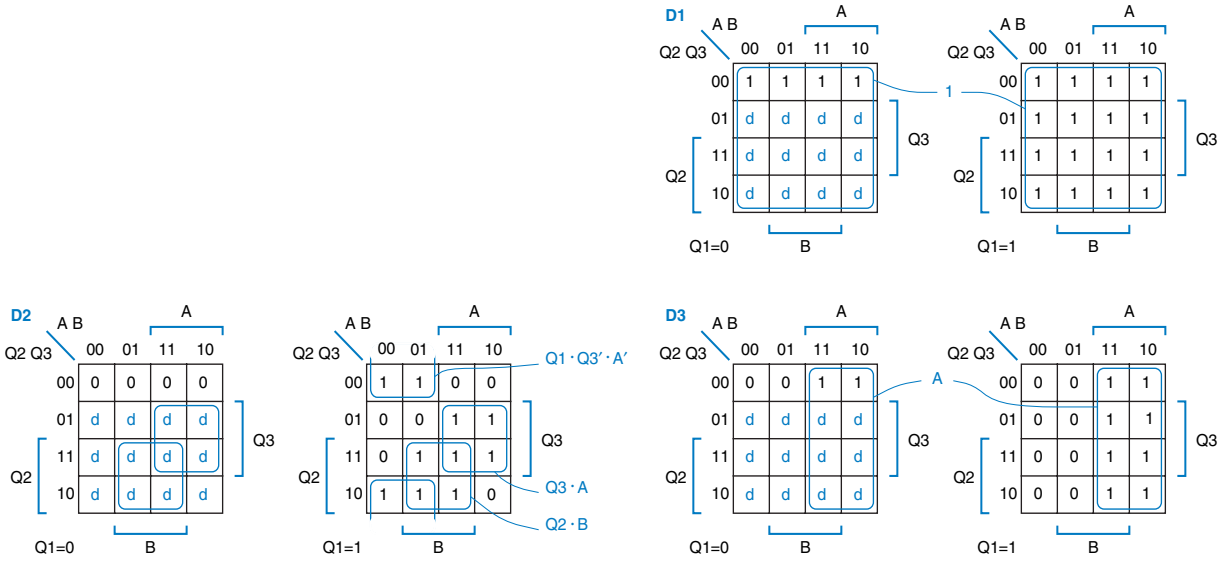


Figure Min-2 Excitation maps for D1, D2, and D3 assuming that next states of unused states are "don't-cares."

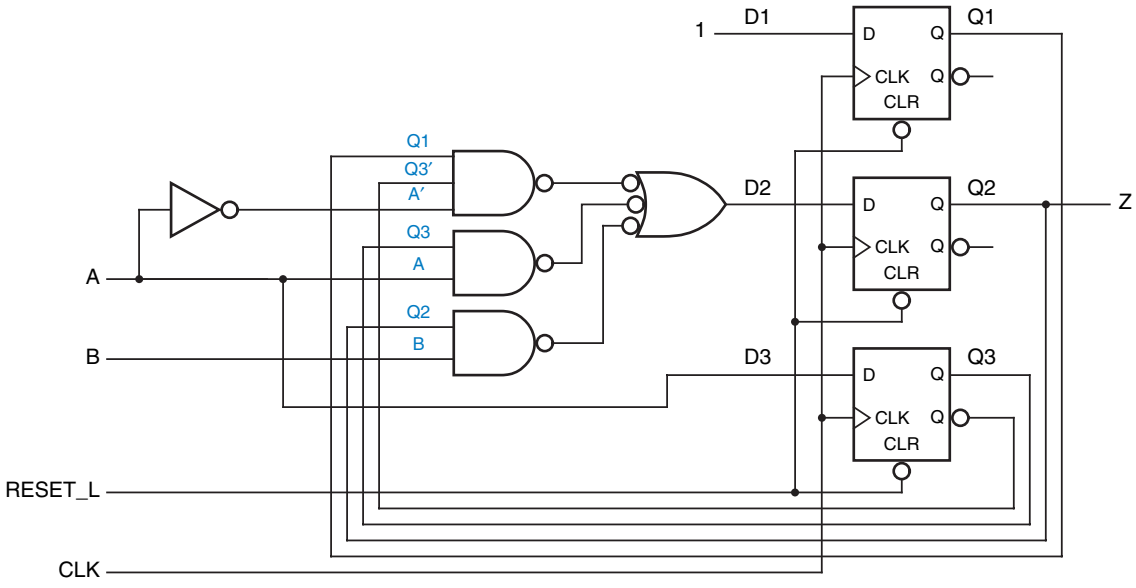


Figure Min-3 Logic diagram resulting from Figure Min-2.

Min.3 Multiple-Output Minimization

Most practical combinational logic circuits require more than one output. We can always handle a circuit with *n* outputs as *n* independent single-output design problems. However, in doing so, we may miss some opportunities for optimization. For example, consider the following two logic functions:

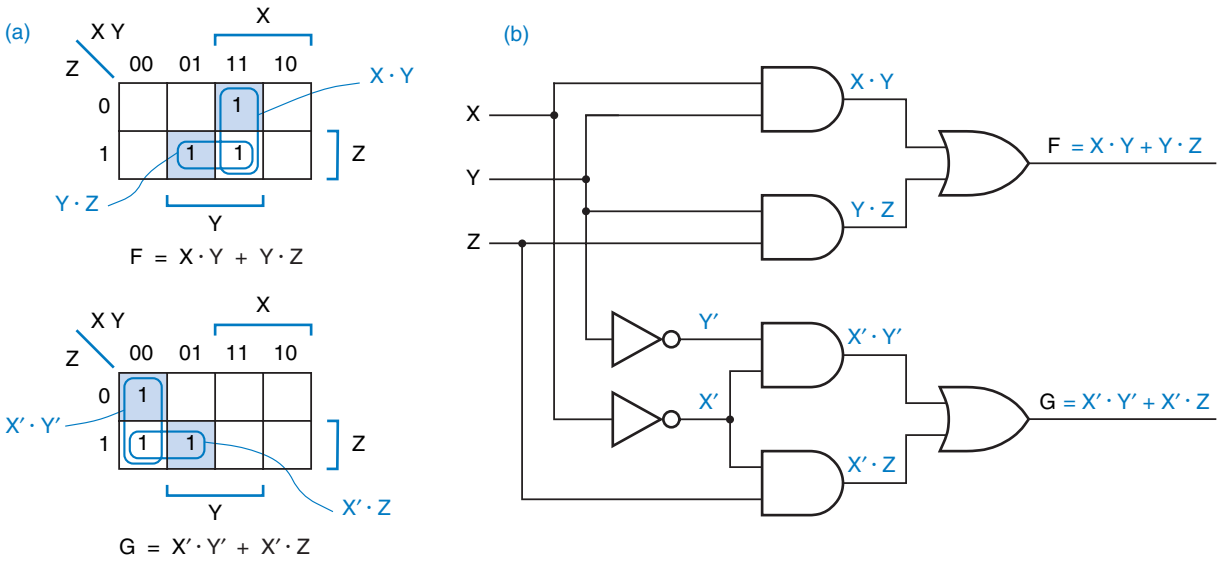
$$F = \Sigma_{X,Y,Z}(3,6,7)$$
$$G = \Sigma_{X,Y,Z}(0,1,3)$$

Figure Min-4 shows the design of F and G as two independent single-output functions. However, as shown in Figure Min-5 on the next page, we can also find a pair of sum-of-products expressions that share a product term, such that the resulting circuit has one fewer gate than our original design.

When we design multiple-output combinational circuits using discrete gates, as in an ASIC, product-term sharing obviously reduces circuit size and cost. In addition, PLDs contain multiple copies of the sum-of-products structure that we've been learning how to minimize, one per output, and some PLDs allow product terms to be shared among multiple outputs. Thus, the ideas introduced in this subsection are used in many logic-minimization programs.

You probably could have "eyeballed" the Karnaugh maps for F and G in Figure Min-5 and discovered the minimal solution. However, larger circuits can be minimized only with a formal multiple-output minimization algorithm. We'll outline the ideas in such an algorithm here; details can be found in the References.

Figure Min-4 Treating a 2-output design as two independent single-output designs: (a) Karnaugh maps; (b) "minimal" circuit.



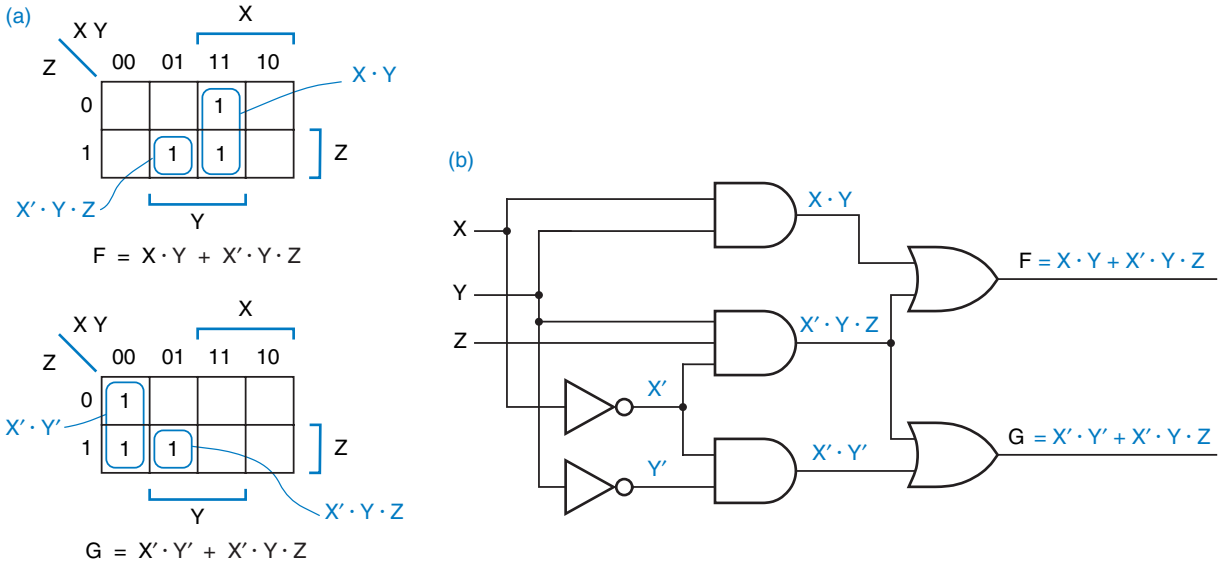


Figure Min-5 Multiple-output minimization for a 2-output circuit: (a) minimized maps including a shared term; (b) minimal multiple-output circuit.

The key to successful multiple-output minimization of a set of n functions is to consider not only the n original single-output functions, but also "product functions." An *m-product function* of a set of n functions is the product of m of the functions, where $2 \le m \le n$. There are $2^n - n - 1$ such functions. Fortunately, $n = 2$ in our example and there is only one product function, F · G, to consider. The Karnaugh maps for F, G, and F · G are shown in Figure Min-6; in general, the map for an m-product function is obtained by ANDing the maps of its m components.

m-product function

A *multiple-output prime implicant* of a set of n functions is a prime implicant of one of the n functions or of one of the product functions. The first step in multiple-output minimization is to find all of the multiple-output prime implicants. Each prime implicant of an m-product function is a possible term to include in the corresponding m outputs of the circuit. If we were trying to minimize a set of 8 functions, we would have to find the prime implicants for $2^8 - 8 - 1 = 247$ product functions as well as for the 8 given functions. Obviously, multiple-output minimization is not for the faint-hearted!

multiple-output prime implicant

Once we have found the multiple-output prime implicants, we try to simplify the problem by identifying the essential ones. A *distinguished 1-cell* of a particular single-output function F is a 1-cell that is covered by exactly one prime implicant of F or of the product functions involving F. The distinguished 1-cells in Figure Min-6 are shaded. An *essential prime implicant* of a particular single-output function is one that contains a distinguished 1-cell. As in single-

distinguished 1-cell

essential prime implicant

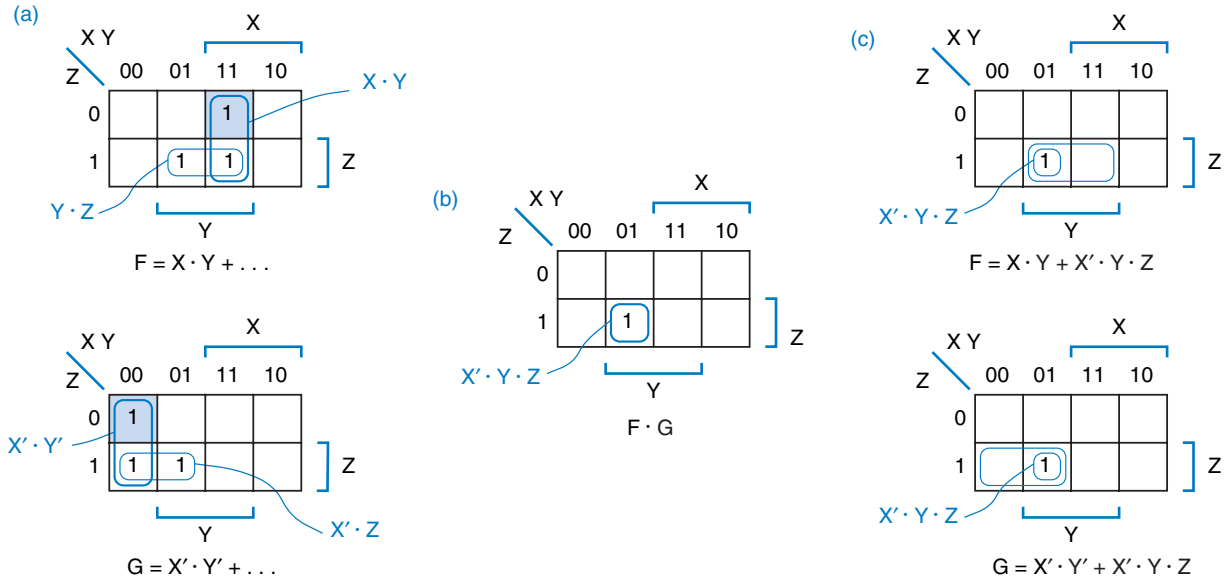


Figure Min-6 Karnaugh maps for a pair of functions: (a) maps for F and G; (b) 2-product map for F · G; (c) reduced maps for F and G after removal of essential prime implicants and covered 1-cells.

output minimization, the essential prime implicants must be included in a minimum-cost solution. Only the 1-cells that are not covered by essential prime implicants are considered in the remainder of the algorithm.

The final step is to select a minimal set of prime implicants to cover the remaining 1-cells. In this step we must consider all n functions simultaneously, including the possibility of sharing; details of this procedure are discussed in the References. In the example of Figure Min-6(c), we see that there exists a single, shared product term that covers the remaining 1-cell in both F and G.

Exercises

Min.1 Find a minimal product-of-sums expression for each of the following logic functions, using the method of Section Min.1. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{X,Y,Z}(1,3,5,6,7)$

(b) $F = \Sigma_{W,X,Y,Z}(1,4,5,6,7,9,14,15)$

(c) $F = \prod_{W,X,Y}(1,4,5,6,7)$

(d) $F = \Sigma_{W,X,Y,Z}(0,1,6,7,8,9,14,15)$

(e) $F = \prod_{A,B,C,D}(4,5,6,13,15)$

(f) $F = \Sigma_{A,B,C,D}(4,5,6,11,13,14,15)$

Min.2 Find a minimal product-of-sums expression for each of the following logic functions, using the method of Section Min.1. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{X,Y,Z}(1,3,5,6,7)$ (b) $F = \Sigma_{W,X,Y,Z}(1,4,5,6,7,9,14,15)$

(c) $F = \prod_{W,X,Y}(0,1,3,4,5)$ (d) $F = \Sigma_{W,X,Y,Z}(0,2,5,7,8,10,13,15)$

(e) $F = \prod_{A,B,C,D}(1,7,9,13,15)$ (f) $F = \Sigma_{A,B,C,D}(1,4,5,7,12,14,15)$

Min.3 Using the method of Section Min.1, find a minimal product-of-sums expression for the function in each of the following figures and compare its cost with the previously found minimal sum-of-products expression: (a) Figure 4-28; (b) Figure 4-30; (c) Figure 4-34.

Min.4 Find a minimal product-of-sums expression for each of the following logic functions, using the method of Section Min.1. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{A,B,C}(0,1,2,4)$ (b) $F = \Sigma_{W,X,Y,Z}(1,4,5,6,11,12,13,14)$

(c) $F = \prod_{A,B,C}(1,2,6,7)$ (d) $F = \Sigma_{W,X,Y,Z}(0,1,2,3,7,8,10,11,15)$

(e) $F = \Sigma_{W,X,Y,Z}(1,2,4,7,8,11,13,14)$ (f) $F = \prod_{A,B,C,D}(1,3,4,5,6,7,9,12,13,14)$

Min.5 Using Karnaugh maps, find a minimal sum-of-products expression for each of the following logic functions, using the method of Section Min.2. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{W,X,Y,Z}(0,1,3,5,14) + d(8,15)$ (b) $F = \Sigma_{W,X,Y,Z}(0,1,2,8,11) + d(3,9,15)$

(c) $F = \Sigma_{A,B,C,D}(1,5,9,14,15) + d(11)$ (d) $F = \Sigma_{A,B,C,D}(1,5,6,7,9,13) + d(4,15)$

(e) $F = \Sigma_{W,X,Y,Z}(3,5,6,7,13) + d(1,2,4,12,15)$

Min.6 Find a minimal product-of-sums expression for each of the following logic functions, using the methods of Section Min.1 and Section Min.2. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{W,X,Y,Z}(0,1,3,5,14) + d(8,15)$ (b) $F = \Sigma_{W,X,Y,Z}(0,1,2,8,11) + d(3,9,15)$

(c) $F = \Sigma_{A,B,C,D}(1,5,9,14,15) + d(11)$ (d) $F = \Sigma_{A,B,C,D}(1,5,6,7,9,13) + d(4,15)$

(e) $F = \Sigma_{W,X,Y,Z}(3,5,6,7,13) + d(1,2,4,12,15)$

Min.7 For each logic function in the two preceding exercises, determine whether the minimal sum-of-products expression equals the minimal product-of-sums expression. Also compare the circuit cost for realizing each of the two expressions.

Min.8 Using Karnaugh maps, find a minimal sum-of-products expression for each of the following logic functions, using the method of Section Min.2. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{W,X,Y,Z}(0,1,3,5,14) + d(8,15)$ (b) $F = \Sigma_{W,X,Y,Z}(0,1,2,8,11) + d(3,9,15)$

(c) $F = \Sigma_{A,B,C,D}(4, 6, 7, 9, 13) + d(12)$ (d) $F = \Sigma_{A,B,C,D}(1, 5, 12, 13, 14, 15) + d(7, 9)$

(e) $F = \Sigma_{W,X,Y,Z}(4, 5, 9, 13, 15) + d(0, 1, 7, 11, 12)$

Min.9 Find a minimal product-of-sums expression for each of the following logic functions, using the methods of Section Min.1 and Section Min.2. Indicate the distinguished 1-cells in each map.

(a) $F = \Sigma_{W,X,Y,Z}(0,1,3,5,14) + d(8,15)$ (b) $F = \Sigma_{W,X,Y,Z}(0,1,2,8,11) + d(3,9,15)$

(c) $F = \Sigma_{A,B,C,D}(4, 6, 7, 9, 13) + d(12)$ (d) $F = \Sigma_{A,B,C,D}(1, 5, 12, 13, 14, 15) + d(7, 9)$

(e) $F = \Sigma_{W,X,Y,Z}(4, 5, 9, 13, 15) + d(0, 1, 7, 11, 12)$

Min.10 For each logic function in the two preceding exercises, determine whether the minimal sum-of-products expression equals the minimal product-of-sums expression. Also compare the circuit cost for realizing each of the two expressions.

Min.11 Find the minimal product-of-sums expressions for the logic functions in Figures 4-34 and 4-37.

Min.12 Use switching algebra to show that the logic functions obtained in Exercise Min.15 equal the AND-OR functions obtained in Figures 4-34 and 4-37.

Min.13 Determine whether the product-of-sums expressions obtained by "adding out" the minimal sums in Figures 4-34 and 4-37 are minimal.

Min.14 Repeat Exercise 4.40, finding a minimal product-of-sums expression for each logic function.

Min.15 Find the minimal product-of-sums expressions for the logic functions in Figures 4-28 and 4-30.

Min.16 Use switching algebra to show that the logic functions obtained in Exercise Min.15 equal the AND-OR functions obtained in Figures 4-28 and 4-30.

Min.17 Determine whether the product-of-sums expressions obtained by "adding out" the minimal sums in Figure 4-28 and 4-30 are minimal.

Min.18 Repeat Exercise 4.58, finding a minimal product-of-sums expression for each logic function.

Min.19 Derive the minimal product-of-sums expression for the prime BCD-digit-detector function of Figure Min-1. Determine whether or not the expression algebraically equals the minimal sum-of-products expression and explain your result.

Min.20 Repeat Exercise 4.59, finding a minimal product-of-sums expression for each logic function.

Min.21 Prove that a two-level OR-AND circuit corresponding to the complete product of a logic function is always hazard free.

Min.22 Redraw the transition table and the resulting Karnaugh maps for the state-machine design in Section 7.4.4, assuming that the next-state entries for unused states are "don't cares." Derive the resulting excitation equations, and make sure they're the same as the ones given in the box on page 566.

Min.23 What changes would be made to the excitation and output equations for the combination-lock machine in Section 7.4.6 as the result of performing the formal multiple-output minimization procedure on the five functions? You need not construct 31 product maps and go through the whole procedure; you should be able to "eyeball" the excitation and output maps in Section 7.4.6 to see what savings are possible.