MORE OPTIMIZATION

Selected topics not covered in the fourth edition of *Logic and Computer Design Fundamentals* are provided here for optional coverage and for self-study. This material fits well with the desired coverage in some programs but not may not fit within others due to time constraints or local preferences. This supplement provides two optimization algorithms for finding a minimum cost two-level circuit. The first algorithm selects prime implicants for a minimum cost implementation of a two-level, sum of products circuit. The second algorithm replaces K-maps with tabular representations that permit the handling of more that the six variables possible using K-maps. Unfortunately, the latter algorithm is difficult to execute manually and the prime implicant generation approach is not the best for program implementation. The prime implicant selection step is more useful for both manual computation for simple problems and computer implementation for more complex ones. Overall, the material in this supplement provides a more rigorous perspective for the cost optimization for two-level circuits and shows some of the potential computational difficulties of rigorous optimization problem solutions for digital circuits.

Both sections require the study of sections 2-4 and 2-5 of the textbook. Coverage of the tabular algorithm is optional, but has as it prerequisite coverage of the prime implicant selection algorithm.

A Prime Implication Selection Algorithm

In Chapter 2 of *Logic and Computer Design Fundamentals* by Mano and Kime, a selection rule for choosing non-essential prime implicants for a sum-of-products expression for a function is given. Recall this selection rule: "Minimize the overlap among prime implicants as much as possible. In particular, in the final solution make sure each prime implicant selected includes at least one minterm not included in any other prime implicant." While it often gives good results, the selection rule does not guarantee a minimum literal or gate input cost solution. The alternative approach given here is systematic and guarantees a minimum cost solution.

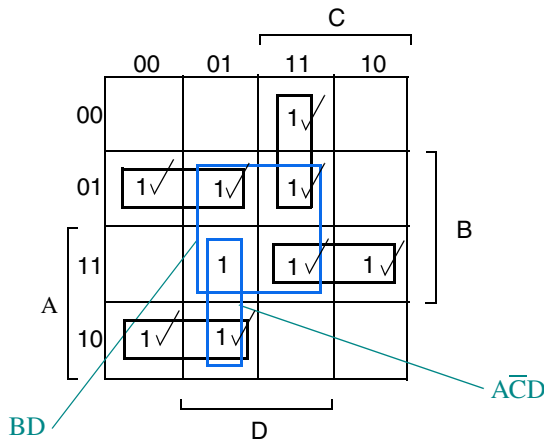[1] © Pearson Education 2008 All rights reserved.

□ FIGURE 1
Example 1 – Need for Prime Implicant Selection Algorithm

WHY IS A PRIME IMPLICANT SELECTION ALGORITHM NEEDED? Figure 1 illustrates the need for a prime implicant selection algorithm that goes beyond the selection rule. In this example, the essential prime implicants have been selected and the minterms covered by these prime implicants have been checked off. The minterm without a check, $AB\overline{C}D$, must be included in a prime implicant in the sum-of-products expression. The two prime implicants that include this minterm are $A\overline{C}D$ and BD. Applying the selection rule, we pick $A\overline{C}D$ since this overlaps other prime implicants in only one minterm. In contrast, BD overlaps other prime implicants in three minterms. But is this the best choice? The answer is clearly no, since $A\overline{C}D$ has three literals and BD has only two. Selection of BD clearly gives a smaller literal cost, demonstrating that the selection rule is inadequate for achieving the minimum literal cost goal.

The question we need to answer is: How can we construct a systematic algorithm for prime implicant selection? This example gives two hints at what is lacking in the selection rule. First of all, we need to take into account the number of literals required to implement the product term corresponding to a prime implicant. Second, we need a more systematic way of determining the "overlap" between prime implicants. The algorithm we will give accomplishes both of these goals. This algorithm solves what is generally referred to as a "minimum covering problem" – hence, we will refer to its solution which covers minterms of the function with prime implicants as a *minimum prime implicant cover*. Two additional concepts related to the goals above and essential to the algorithm appear in the next section.

Less Than PIs and Secondary Essential PIs

For simplicity, we abbreviate the term "prime implicant" using PI. For multiple prime implicants, we add an s to give PIs. In seeking a minimum PI cover, we want to exclude those PIs that *at the given stage of the algorithm execution* would not be
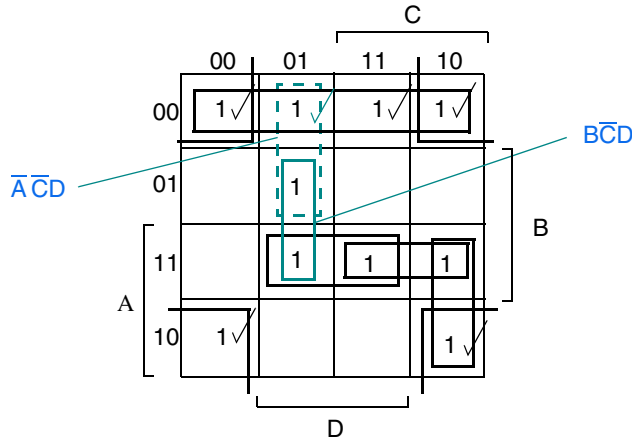
included in the PI cover we are forming. The less than PI concept permits us to do this. Once one or more PIs have been excluded, then some other PIs must be included in the PI cover and hence become essential. These are called *secondary essential PIs.*

We will illustrate the concept of a less than relationship between two PIs and then proceed to definitions of less than, a less than PI, and equivalent PIs. Consider the function represented on the K-map in Figure 2. The essential PIs are $\overline{A}\,\overline{B}$ and $\overline{B}\,\overline{D}$. The minterms included in these PIs have been checked off. We now focus on PIs $\overline{A}\,\overline{C}D$ and $B\overline{C}D$. Both of these PIs include unchecked minterm $\overline{A}B\overline{C}D$. But $B\overline{C}D$ also includes unchecked minterm $AB\overline{C}D$. Since we are interested in selecting PIs that include unchecked minterms, would we ever select $\overline{A}\,\overline{C}\,D$ rather than $B\overline{C}D$? The answer is no. First of all, the literal cost of implementing these two PIs is three in both cases, so their cost is the same. Second, $B\overline{C}D$ includes all the unchecked minterms that $\overline{A}\,\overline{C}\,D$ does, but contains an additional unchecked minterm. So $B\overline{C}D$ always contributes more to the goal of including minterms in PIs than $\overline{A}\,\overline{C}\,D$ does and, as a consequence, in moving forward with this solution, we will never want to use $\overline{A}\,\overline{C}\,D$. We denote this by saying that $\overline{A}\,\overline{C}D$ is less than $B\overline{C}D$ and call $\overline{A}\,\overline{C}\,D$ a less than PI. Since we will never use $\overline{A}\,\overline{C}\,D$, we delete it. Then, $B\overline{C}D$ becomes the only PI left that includes $\overline{A}B\overline{C}D$ making it essential at this stage in the solution. As a consequence, we say that $B\overline{C}D$ is a secondary essential PI and add it to the PIs in the solution.

We say that PI_i is *less than* PI_j, denoted $PI_i \leq PI_j$, if: 1) PI_i contains at least as many literals as PI_j and 2) PI_j includes at least all of the as yet unchecked minterms that PI_i includes. Applying this to $\overline{A}\,\overline{C}\,D$ and $B\overline{C}D$, $\overline{A}\,\overline{C}\,D$ contains three literals, as least as many as the three literals in $B\overline{C}D$. Also, $B\overline{C}D$ includes $\overline{A}B\overline{C}D,$ the only unchecked minterm included in $\overline{A}\,\overline{C}\,D$ but includes $AB\overline{C}D$ as well. Thus, $\overline{A}\,\overline{C}\,D \leq B\overline{C}D$.
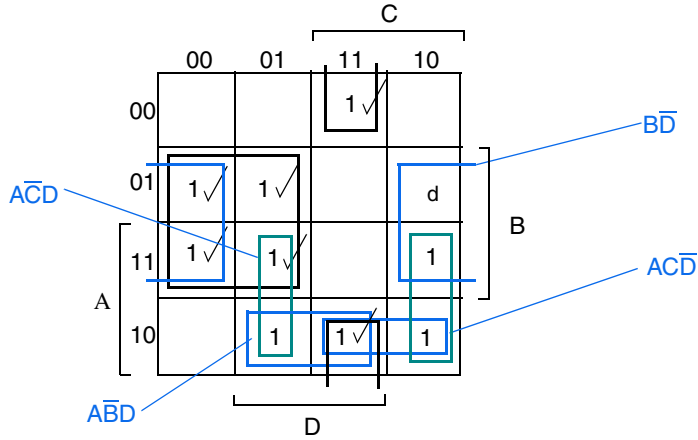
□ FIGURE 3

Example 3 – Less Than Relation between PIs Absent and Equivalent PIs

Assuming that $B\overline{C}D$ is now selected and $\overline{A}B\overline{C}D$ and $AB\overline{C}D$ are now checked, are there other less than relations between the remaining non-selected PIs? Examination of the remaining non-selected PIs yields $ABD \leq ABC$ and $AC\overline{D} \leq ABC$. Deleting the less than PIs ABD and $AC\overline{D}$ makes ABC a secondary essential PI. Adding it to the solution permits us to check off the remaining unchecked minterms and thus completes the solution:

$F(A, B, C, D) = \overline{A}\,\overline{B} + \overline{B}\,\overline{D} + B\overline{C}D + ABC$

Now, suppose we go back to the Example 1 in Figure 1. Is there a less than relation between BD and $A\overline{C}D$? They both include the same unchecked minterm. And the number of literals in BD is two and in $A\overline{C}D$ is three. Thus, $A\overline{C}D \leq BD$ and BD will be selected to complete the solution.

Example 3 in Figure 3 contains two PIs that overlap but have no less than relation and presents the idea of equivalent PIs. Consider first the two PIs $B\overline{D}$ and $AC\overline{D}$. Since $B\overline{D}$ uses two literals and $AC\overline{D}$ uses three literals, $B\overline{D}$ cannot be less than $AC\overline{D}$. But since $AC\overline{D}$ covers $A\overline{B}C\overline{D}$ as well as $ABC\overline{D}$ which $B\overline{D}$ also covers, $AC\overline{D}$ cannot be less than $B\overline{D}$. Thus, there is no less than relation between these two PIs. Next, consider PIs $A\overline{B}D$ and $A\overline{C}D$. Both use the same number of literals and both included exactly one uncovered minterm $A\overline{B}\,\overline{C}D$. Based on the definition of less than, $A\overline{B}D \leq A\overline{C}D$ and $A\overline{C}D \leq A\overline{B}D$. In such a situation, the PI to be used can be selected arbitrarily and $A\overline{B}D$ is said to be *equivalent* to $A\overline{C}D$. Finally, $A\overline{B}C \leq AC\overline{D}$. Deleting $A\overline{B}C$, $AC\overline{D}$ becomes secondary essential and a part of the solution, making $B\overline{D}$ useless since it does not cover an unchecked 1. In this case, BD is said to be a redundant PI. To complete the solution, we can select either $A\overline{C}D$ or $A\overline{B}D$ since they are equivalent.

With these definition in hand, we now can present an initial PI selection algorithm.

Initial PI Selection Algorithm
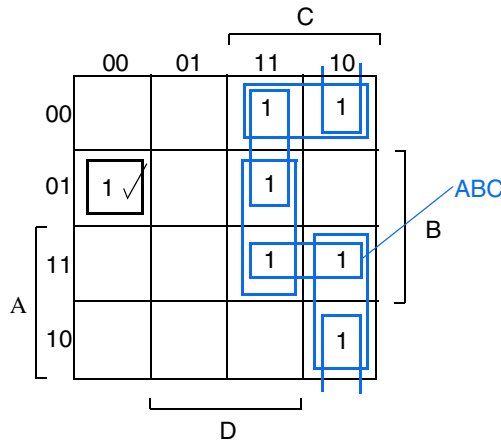
□ FIGURE 4
Example 4 – A Cyclic Structure

The steps of the algorithm developed thus far follow. Note that secondary essential PIs are simply referred to here as essential PIs to simplify the flow of the algorithm.

1. Find all PIs of the function F.
2. Select all essential PIs, checking off the included minterms.
3. Find all less than PIs and delete them. (As a result, some of the other non-selected PIs may become essential.)
4. Repeat 2 and 3 until no more less than PIs appear.
5. Find equivalent PIs and select arbitrarily one PI from each set of equivalent PIs, checking off included minterms.
6. Delete all redundant (unused) PIs.

In many cases, this algorithm succeeds in including all minterms of the function, giving a minimum PI cover. But there are cases in which there are remaining unchecked minterms, but no more less than or equivalent PIs. In these cases, we say that *a cyclic structure* exists.

Cyclic Structures

In the cases in which the initial algorithm does not complete, a pattern of PIs referred to as a cyclic structure is present. This situation is illustrated in Example 4 in Figure 4. The only essential PI is $\overline{A}B\overline{C}\overline{D}$ on the left of the K-map. No essential PIs or less than relations exists among the six PIs on the right of the K-map. Thus, these PIs form a cyclic structure. Note that in this case, the PIs are linked together like a chain or cycle; hence, the term cyclic structure.

The approach to handling a cyclic structure is to arbitrarily select a PI and generate one cover and delete that same PI and generate a second cover. The literal cost of each cover is calculated and the lowest cost one is selected unless they have the same cost in which case the selection is arbitrary. This approach can be illustrated using Example 4. Suppose we pick ABC as the arbitrary PI to use. This

causes BCD and $AC\overline{D}$ to become less than PIs, so these are deleted. The deletion of these PIs makes $\overline{A}CD$ and $\overline{B}C\overline{D}$ secondary essential PIs. Selecting these PIs includes all minterms of the function giving the first solution:

$$F(A, B, C, D) = \overline{A}\ B\ \overline{C}\ \overline{D} + ABC + \overline{A}CD + \overline{B}C\overline{D}$$

Note that the PI $\overline{A}\ \overline{B}C$ is never used in this cover it is a redundant PI.

Beginning again, omit ABC. This makes BCD and $AC\overline{D}$ secondary essential PIs. Selecting these generates less than PIs $\overline{A}CD$ and $\overline{B}C\overline{D}$. Deleting the two less than PIs causes $\overline{A}\ \overline{B}C$ to become a secondary essential PI which is selected to complete the second solution:

$$F(A, B, C, D) = \overline{A}B\overline{C}\ \overline{D} + BCD + AC\overline{D} + \overline{A}\ \overline{B}C$$

In this case, the two covers have the same literal cost, so either can be selected as the final minimum literal cost expression.

Final Algorithm

The final minimum prime implicant cover algorithm:

1. Find all PIs of the function F.
2. Select all essential PIs, checking off included minterms.
3. Find all less than PIs and delete them. (As a result, some of the other non-selected PIs may become essential.)
4. Repeat 2 and 3 until no more less than PIs appear.
5. Find equivalent PIs and select arbitrarily one PI from each set of equivalent PIs, checking off included minterms.
6. If minterms remain unchecked and no PI less than relations can be obtained, then a cyclic structure exists. For a cyclic structure, (a) arbitrarily select a PI and repeat steps 1 through 6 and (b) delete the same PI selected and repeat steps 1 through 6. Compare literal cost of the solutions generated and select the minimum literal cost cover.
7. Discard any redundant (unused) PIs.

This algorithm will be illustrated in the next section, after we consider the effects of don't cares on it.

Functions with Don't Cares

The algorithm can easily handle functions with don't cares. The following simple rules apply:

1) Treat don't cares as if they are 1's for PI generation.
2) Immediately delete any PIs that cover only don't cares.
3) Don't care minterms need not be covered by a PI or checked off. (Inclusion of some of them and not others occurs arbitrarily in the course of the solution.)

We now illustrate the final algorithm for Example 5 in Figure 5. Step 1, completed in part (a), yields the six PIs shown. Since there are no PIs that cover only don't cares, none are deleted. Since every minterm (with value 1) is included in at
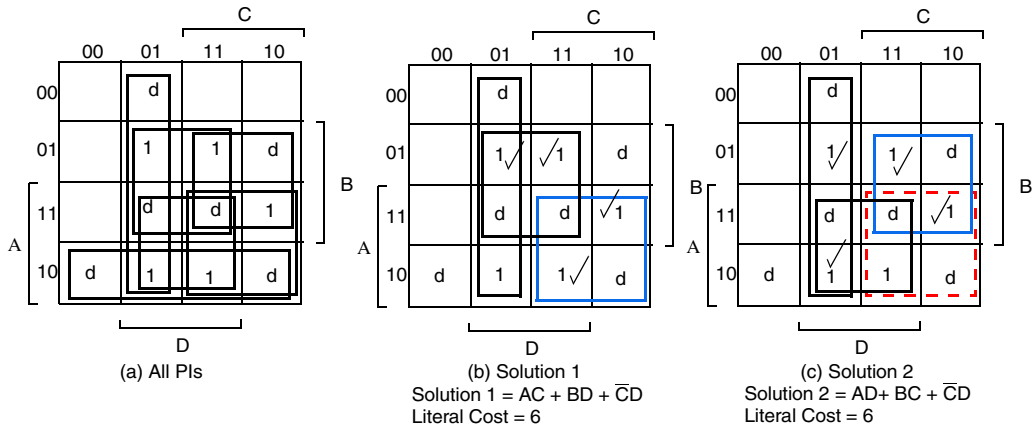
◻ FIGURE 5
Example 5 – Illustration of Final Algorithm

least two PIs, there are no essential PIs found in step 2. Likewise, in step 3, no less than PIs are found. Going to step 5, there are no equivalent PIs, which leads the solution to a cyclic structure and execution of step 6(a). In Figure 5(b), PI AC is arbitrarily picked. This makes $BC \leq BD$ and upon the removal of BC, BD becomes essential. The included minterms are checked. $\overline{C}D$, $A\overline{B}$ and AD are equivalent since each covers $A\overline{B}\,\overline{C}D$ and they have the same literal cost. We arbitrarily pick $\overline{C}D$ to obtain Solution 1 with a literal cost of 6, as shown in Figure 5(b).

We now return to execute step 6(b) and omit AC, the arbitrary PI selected earlier to produce a new solution. This makes BC essential. Inclusion of BC makes $BD \leq \overline{C}D$. Deletion of BD makes $\overline{C}D$ essential. The remaining PIs, $A\overline{B}$ and AD, are equivalent since both cover $A\overline{B}CD$ and they have the same literal cost. Arbitrarily selecting AD gives Solution 2 in Figure 5(c) with a literal cost of 6. Since the two solutions have the same literal cost, either can be selected as a final solution.

The Tabular Algorithm

We have studied K-maps for up to four variables. Five- and six-variable K-maps are logical extensions using multiple four variable maps. For larger numbers of variables, the geometric representation becomes intractable. So an alternative algorithm is needed. The particular algorithm that we will introduce uses the Boolean identity $\overline{X}Y + XY = Y$ known as the minimization theorem for prime implicant generation beginning with minterms. It also implements the prime implicant selection algorithm as discussed in the prior section. The tabular algorithm is also called the Quine-McCluskey algorithm.

The algorithm uses a *cubical notation* for product terms, so before discussing and illustrating the algorithm, we need to introduce this notation. A *cube* is a vector of 1's, 0's and –'s corresponding to a product term. The order of the variables in the product term must be known for the cube to be meaningful. To form a cube from a product term, a true literal is replaced by a 1, a complemented literal by a 0
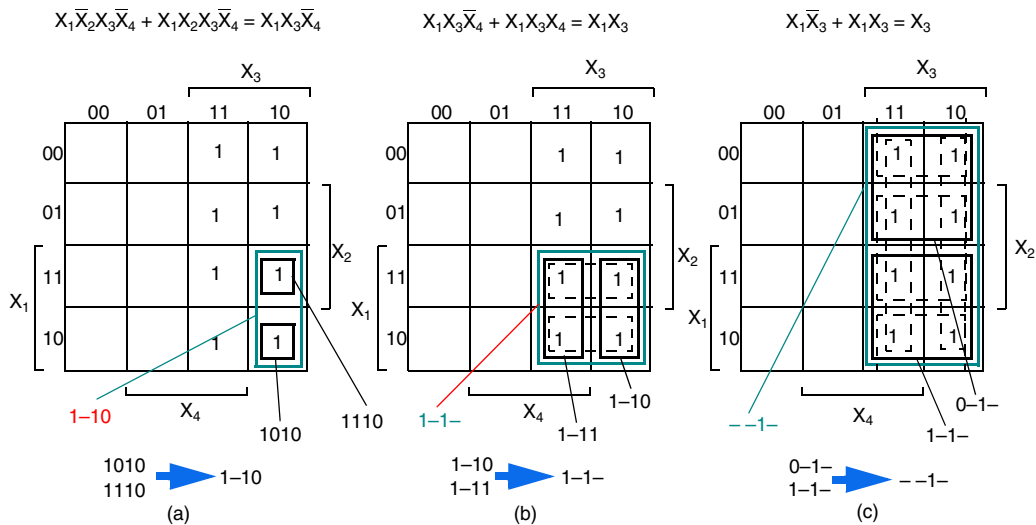
$$X_1\overline{X}_2X_3\overline{X}_4 + X_1X_2X_3\overline{X}_4 = X_1X_3\overline{X}_4 \qquad X_1X_3\overline{X}_4 + X_1X_3X_4 = X_1X_3 \qquad X_1\overline{X}_3 + X_1X_3 = X_3$$

(a) (b) (c)

☐ **FIGURE 6**
Illustration of Minimization Theorem Application

and a missing literal by a blank (–). For example, if the variables are ordered X_1, X_2, X_3, X_4 and the product term is $X_1\overline{X}_2X_4$, then the corresponding cube is 10–1. The minterm $\overline{X}_1X_2\overline{X}_3X_4$ is represented by 0101. In the next subsection, we will see how the minimization theorem can be applied by using this notation and how it can be used to systematically generate all prime implicants for a function.

Prime Implicant Generation

Since the minimization theorem, $\overline{X}Y + XY = Y$, is the basis for this algorithm, we need to represent the application of the theorem in the cubical domain. First, we note that, in the application of the theorem, X is one of the variables, $\overline{X}$ is its complement, and Y is a product of other literals. For example,

$$X_1\overline{X}_2X_3\overline{X}_4 + X_1X_2X_3\overline{X}_4 = X_1X_3\overline{X}_4$$

The first two terms in the above are represented by cubes 1010 and 1110 and the resulting cube is represented by 1–10. Note that the first two cubes match in all positions but one and in that position a 0 appears in one cube and a 1 in the other. In the resulting cube, a – is placed in that position and all of the other positions in which the cubes match are copied. Figure 6(a) shows the application of the theorem for this case. The equation appears first with a representation of the equation on a K-map. From this map, it is apparent that the terms on the left of the equation correspond to two adjacent squares on the map. The product term on the right of the equation corresponds to the rectangle that is the combination of these two squares. By reading off the 0 and 1 entries on the edges of the K-map, we find that the cubes for the two squares are 1010 and 1110. By copying the entries in which these two cubes match and replacing the entry in which they differ by a –, we

8 ☐

| | (b) Minterms in | | |
| (a) Minterms | Index Order | (c) 1-Cubes | (d) 2-Cubes |

(a) Minterms:
0011
0100
0101
0110
1001
1010
1011
1100
1101
1110

(b) Minterms in Index Order:
1   0100 ✓

2   0011 ✓
    0101 ✓
    0110 ✓
    1001 ✓
    1010 ✓
    1100 ✓

3   1011 ✓
    1101 ✓
    1110 ✓

(c) 1-Cubes:
      010– ✓
1,2   01–0 ✓
      –100 ✓

      –011
      –101 ✓
      –110 ✓
      10–1
2,3   1–01
      101–
      1–10
      110– ✓
      11–0 ✓

(d) 2-Cubes:
1,2,3   –1–0
       –10–

☐ **FIGURE 7**

Tabular Prime Implicant Generation for Example 3 in Figure 3

obtain 1–10 as indicated by the large arrow. This cube corresponds as expected to the rectangle on the K-map.

Figure 6(b) shows two terms on the left of the equation involving three of the four variables. Investigation of the corresponding K-map shows that these correspond to adjacent rectangles of size two. The cubes for these rectangles can be read as 1–11 and 1–10. These can likewise be obtained from the equations by definition of the cubes. Combining the adjacent rectangles gives 1–1– which is the same as the result obtained by combining the cubes. Note that there are two other rectangles on the K-map shown as dashed lines that also combine to give the same result. These are of note since they will appear and be removed in the algorithm we will use.

Finally, Figure 6(c) shows two terms, X_1X_3 and X_1X_3 involving just two of the four variables. These again correspond to adjacent rectangles which can be combined to yield a larger rectangle. Using cubes, this corresponds to combining 0–1– and 1–1– to give – –1–. These three examples illustrate how the minimization theorem applications correspond in the three different representations. It clearly shows how adjacent rectangles on the K-maps appear as cubes that are identical in all but one entry and contain 0 and 1, respectively, in that entry. It also shows how the cube corresponding to the larger rectangle on the K-map is obtained by copying the identical entries and placing a – in the entry containing the 0 and 1.

In Figure 7, we illustrate the tabular prime implicant generation algorithm for Example 3 in Figure 3. We begin with cubes corresponding to minterms and look for adjacent cubes differing in exactly one position. If we do this to a list of cubes as shown in Figure 7(a), then we compare each cube to all those that follow, considering all possible pairs of cubes. By first arranging the cubes in groups based on the number of 1's in each cube, we can substantially reduce the number of comparisons. The number of 1's is called the *index* of a cube and we need only compare cubes with indexes differing by 1. Figure 7(b) shows such an arrangement of the

cubes from Figure 7(a) with the index of each group given on the left. This arrangement of cubes for the minterms is the first step of the algorithm.

In the second step of the algorithm, we compare each cube with index i to all cubes with index $i + 1$. If two cubes can be combined, then we place a check by each of the cubes and write the resulting cube in the group labeled $(i, i + 1)$ in the next column. A check by a cube indicates that cube has been combined into a larger cube and is not a PI. To illustrate, 0100 combines with 0101 to give 010– and we check off 0100 and 0101. All of the results of this step are shown in Figure 7(c) and are labeled as 1-cubes. The 1-cubes have one variable missing in the corresponding product term.

A form of the second step is then repeated until there are no more cubes that can be combined. In the example, the cubes with indexes 1,2 are compared with those with indices 2,3. Note that the cubes in group 1,2 contain a single 1 and a single – and in group 2,3 contain two 1's and a single –. Thus cubes from these two groups have potential to be adjacent. But cubes from group 0,1 and 2,3 could not be adjacent since they would contain zero 1's and one – and two 1's and one –. Thus, again, only cubes in adjacent groups need to be compared. There is a shortcut that can be used in comparison of cubes containing –'s. In order for two cubes in adjacent groups to combine, they must have –'s in identical positions. These cubes can be visually recognized with ease. This step for the example yields Figure 7(d) consisting of 2-cubes. It is interesting to note that 01–0 and 11–0 form –1–0 and are checked off. But, –100 and –110 also form –1–0. Since –1–0 is already listed, we do not repeat it, but simply check off –100 and –110. This checkoff is based on the equation $X + XY = X$.

The two cubes appearing in the 2-cube column are in the same group and thus cannot be combined, so the algorithm execution is finished. All cubes without a check are prime implicants. The prime implicants shown can be verified by comparing them with the rectangles given in Figure 3.

The prime implicant generation algorithm we have just executed for the example is as follows:

1. Find the cubes corresponding to minterms and arrange them in order based on their index.
2. Compare cubes in groups with indexes differing by 1, forming a new cube in the next column for adjacent cubes if such a cube does not already exist. Check off the cubes from which a new cube has been (or could have been) formed.
3. Repeat step 2 until no new cubes are formed.
4. The cubes not checked off are the prime implicants.

The Prime Implicant Table

Selection in the overall algorithm begins with step 2 of the final algorithm from the first section, but the information is represented by a *prime implicant table* instead of a K-map. This table has a row for each PI and a column for each minterm for which the function has value 1. If a PI includes a minterm, then an X is placed at the intersection of the PI row and the minterm column. The prime implicant table for Example 3 is shown in Figure 8. Con-

		0011	0100	0101	1001	1010	1011	1100	1101	1110
E	−011	X					X			
EQ	10−1				X		X			
EQ	1−01				X				X	
LT	101−				X	X				
SE	1−10				X					X
RE	−1−0		X					X		X
E	−10−		X	X				X	X	

☐ FIGURE 8

Prime Implicant Table for Example 3 in Figure 3

sider the prime implicant −1−0. By filling in all possible combinations of 0 and 1 for the two − entries, we find that minterms 0100, 0110, 1100, and 1110 are covered. The three minterms 0100, 1100, and 1110 are marked with X's in the −1−0. In the original specification, minterm 0110, however, has a don't care value, permitting it to be either 0 or 1. Since it does not need to be covered, it is omitted from the table.

    In the second step of the PI Selection Algorithm, we select essential prime implicants. To identify essential PIs, we look for columns containing a single X. Such a column identifies a minterm included in a single PI. For the example, there are single X's in columns 0011 and 0101. The essential PIs for these X's are −011 and −10−. We select these PIs and place a check at the bottom of the columns of the minterms included in them. Next, we examine PIs associated with the three unchecked minterms to look for less than relations; we specifically will look at literal costs and the unchecked minterms included in these PIs. Beginning with 1001, we find that 1−01 and 10−1 have the same number of literals and both cover 1001 and no other unchecked PIs. Thus, 1−01 and 10−1 are equivalent. Next, looking at 1010, we find that 101− and 1−10 have the same number of literals and that 1−10 includes 1110 as well as 1010. Thus, 101− ≤ 1−10. Finally, examining the 1110 column, −1−0 has fewer literals than 1−10, but 1−10 includes an added minterm. So no less than relation can be established between −1−0 and 1−10. Deleting less than PI 101−, 1−10 becomes a (secondary) essential PIs. The included minterms are checked off in the second row of checks at the bottom of the table. Since only 1001 is left, we can select either 10−1 or 1−01 to complete the solution.

    Example 4, the final example, employs a function with a simple cyclic structure given as the PI table in Figure 9. Since every column contains at least two X's, there are no essential prime implicants. This cyclic structure has been handled by arbitrarily picking PI 00− to be used in the solution. The selection of this PI causes −01 and 0−0 to be less than PIs. In turn, this forces PI's 1−1 and −10 to be second-

		000	001	010	101	110	111
Pick	00–	X	X				
LT	–01		X		X		
SE	1–1				X		X
RE	11–					X	X
SE	–10			X		X	
LT	0–0	X		X			
		✓	✓	✓	✓	✓	✓

☐ FIGURE 9
Prime Implicant Table for Example 4 – Solution with 00– Picked

ary essential. Their selection covers the remainder of the minterms and PI 11– is not needed and labeled as redundant.

The solution with 00– picked may not be the lowest cost. So, in order to cover the whole solution space, a solution with 00– omitted must be found. This solution is represented by the prime implicant table in Figure 10. Since both solutions have the same cost, either can be used.

Finally, we note that all algorithms can be applied to product-of-sums minimum PI covers by using $\overline{F}$ instead of F with an inverter placed at the output generating F. These may yield better solutions that those obtained for F with little cost for the additional inverter.

		000	001	010	101	110	111
Omit	00–	X̶	X̶				
SE	–01		X		X		
LT	1–1				X		X
SE	11–					X	X
LT	–10			X		X	
SE	0–0	X		X			
		✓	✓	✓	✓	✓	✓

☐ FIGURE 10
Prime Implicant Table for Example 4 – Solution with 00– Omitted

It is clear from the examples in the prior section and this section that solution of the general two-level optimization problem using the Quine-McCluskey algorithm can become complex. Because of this, practical computer implementations of optimization algorithms do not necessarily obtain minimal cost solutions and employ approaches that avoid enumerating minterms and all prime implicants. One such practical algorithm, Espresso, is illustrated in section 2-6 of the text.

References

1. MANO, M. M. AND C. R. KIME. *Logic and Computer Design Fundamentals,* 4th ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

2. DIETMEYER, D. L., *Logic Design of Digital Systems,* 3rd ed. Boston: Allyn Bacon, 1988.

3. WAKERLY, J. F. *Digital Design: Principles and Practices,* 4th ed. Upper Saddle River, NJ: Prentice Hall, 2006.

Problems

The plus (+) indicates a more advanced problem.

1. Use a K-map to generate a list of all prime implicants for the function
    $$E(A, B, C, D) = \Sigma\ m\ (0,1,5,7,8,10, 14, 15).$$

    Then use the prime implicant selection algorithm to find a minimum cost solution in equation form. Identify the type taken on by each of the prime implicants at each step in your solution. The types are: essential (E), less than (L), secondary essential (S), equivalent (EQ), redundant (R), pick (P) and omit (O). The latter two types, pick and omit are applied to one or more prime implicants to produce alternate solutions used to resolve cyclic structures.

2. Repeat problem 1 for the function
    $$F(W, X, Y, Z) = \Sigma\ m\ (0,1,2,3,4,8,10,11,12,13, 15).$$

3. Repeat problem 1 for the function:
    $$G(A, B, C, D) = \Sigma\ m\ (0,1,2,5,8,10,11,15),$$
    $$G_d(A, B, C, D) = \Sigma_d\ m\ (7,13).$$

4. Repeat problem 1 for the function:
    $$H(A, B, C, D) = \Sigma\ m\ (0,5,10,15),$$
    $$H_d(A, B, C, D) = \Sigma_d m\ (1,2,4,7,8,11,13,14).$$

5. Repeat problem 1 for the function:
    $$I\ (W, X, Y, Z) = \Sigma\ m\ (0,2,6,7,9,13,15),$$
    $$I_d(W, X, Y, Z) = \Sigma_d m\ (8,10)$$
    finding solutions for both I and $\bar{I}$. Compare the costs and select the one with the least cost.

6. Use the tabular algorithm to find a list of all prime implicants for $\bar{F}$ in problem 2. Using your prime implicants, use the tabular algorithm to find a

minimum cost solution. On the table(s), be sure to indicate the type (See problem 1) of each PI with respect to your solution.

7. Repeat Problem 6 for the function $\overline{H}$ from problem 4.

8. The PIs for a particular function M(A, B, C, D) are as follows: $\overline{B}\,\overline{D}$, $\overline{A}\,\overline{C}\,D$, $\overline{A}\,B\overline{C}$, $B\overline{C}\,D$, $A\,\overline{C}\,D$, $A\,\overline{B}\,\overline{C}$, and $A\,C\,D$. The minterm $\overline{A}\,\overline{B}\,C\,\overline{D}$ is a don't care. Use a prime implicant table(s) to find a minimum cost solution. On the table(s), be sure to indicate the type (See problem 1) of each PI with respect to your solution.

9. Repeat problem 8 for the function N(A, B, C, D, E) with the following prime implicants: $\overline{A}\,\overline{B}\,\overline{D}$, $\overline{A}\,\overline{B}\,\overline{C}\,E$, $\overline{A}\,\overline{C}\,D\,E$, $\overline{A}\,C\,\overline{D}\,\overline{E}$, $B\,C\,\overline{D}\,\overline{E}$, $B\,\overline{C}\,D\,E$, $A\,B\,D$, and $A\,B\,C\,\overline{E}$. The minterms $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}$, $\overline{A}\,\overline{B}\,C\,\overline{D}\,E$, $A\,B\,\overline{C}\,D\,\overline{E}$, and $A\,B\,C\,D\,E$ are don't cares.

10. +Repeat problem 8 for the function P(A, B, C, D, E) with the prime implicant table given below. Hint: Note that the table is broken up by double lines into four regions with cyclic structures in region 1, in region 2, and a single cyclic structure that spans regions 3 and 4. All of these cyclic structures consists of 2-cubes. It is suggested that you make four copies of the table to resolve the cyclic structures in regions 1 and 2. The four solutions will be produced as follows: (Pick PI 00-0-, Pick PI 1-00-), (Pick PI 00-0-, Omit PI 1-00-), (Omit PI 00-0-, Pick PI 1-00-), and (Omit PI 00-0-,

Type	A B C D E																								
	E	0	1	0	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1
	D	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0
	C	0	0	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	1
	B	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1	0	1	0	1	0	1	0	1
	A	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	0 0 – 0 –	X	X	X	X																				
	0 0 – – 1		X		X	X	X																		
	0 – – 1 1					X	X	X	X																
	0 1 – 1 –							X	X	X	X														
	0 1 – – 0									X	X	X	X												
	0 – – 0 0	X		X								X	X												
	1 – 0 0 –													X	X	X	X								
	1 – 0 – 0													X		X		X	X						
	1 – – 1 0																	X	X	X	X				
	1 – 1 1 –																			X	X	X	X		
	1 – 1 – 1																					X	X	X	X
	1 – – 0 1														X		X							X	X
	– 0 0 0 –	X	X											X	X										
	– 0 – 0 1		X		X										X									X	
	– 0 1 – 1				X		X															X		X	
	– – 1 1 1						X		X													X	X		
	– 1 1 1 –								X		X										X		X		
	– 1 – 1 0									X	X								X		X				
	– 1 0 – 0									X		X				X			X						
	– – 0 0 0	X										X		X		X									

(Table regions are annotated 1, 2, 3, and 4 in the four quadrants.)

Omit PI 1-00-). The PIs in the cycle in regions 3 and 4 become all less thans in these solutions, so are not used. Compare the four solutions and pick one that has a minimum cost.