A Map Procedure For Two-Level Multiple-Output Logic Minimization

Ali Muhammad Rushdi* and Omar Mohammed Ba-Rukab**

*Department of Electrical and Computer Engineering, King Abdulaziz University, Jeddah, Saudi Arabia
**Department of Computer Technology, College of Telecomm and Electronics, Jeddah, Saudi Arabia

ABSTRACT. A pedagogical treatment of two-level multiple-output logic minimization is presented through the detailed exposition of a novel fast procedure. This procedure is a purely map technique that generalizes the map procedure for single-output minimization. It neither requires the generation of the set of all paramount prime implicants, nor does it require the construction of a cover matrix. Instead, it utilizes certain visual interactions between various groups of maps placed at distinct levels of a Hasse diagram, which is conveniently drawn in a Karnaugh-map layout. The detailed exposition presented herein is intended to enhance what is currently available in undergraduate texts, and it can serve as a supplement rather than a replacement to software experience with computer solutions of the problem. Two illustrative examples demonstrate the details of the proposed procedure for the two dual cases of two-level minimization.

1. Introduction

The design of a compact combinational switching network with multiple outputs is an important problem that frequently arises in logic design practice [1-6], and that is considered by most educators as an essential ingredient of a first course on digital design within the computer engineering curriculum [7-15]. Many software packages for handling the aforementioned problem exist with some of them intended for educational purposes [14]. There is a unanimous agreement, however, that a student will not achieve the desired competency in logic design by solely mastering the use of one or more of these packages. The student needs at least a glimpse of the theoretical basis of the design problem, and an acquaintance with some manual technique for solving it.

Recent popular textbooks on digital design offer different expositions of the aforementioned problem that vary in depth, length, rigor and in the nature of the manual technique or procedure employed. In many texts (see, e.g., [15]) the student is not burdened with any particular details of collective minimization. Instead, the student is instructed to apply separate or individual minimization and then try to identify obvious common product terms so as to share these between the pertinent output functions. Other texts [7,10] introduce students to the classical method of collective minimization through a map derivation of all paramount prime implicants (PPIs), followed by the construction of a Quine-McCluskey
cover matrix and/or a presence or Petrick function for selecting a minimal subset of the set of all PPIs. This method requires further checking for irredundant connections if exact minimization is to be achieved. Some texts present a simplified version of the classical method in which the final solution is generated directly from the set of multiple-output prime implicants (MOPIs) without reducing the MOPI set to its subset of PPIs [8,12,13].

This paper offers a pedagogical treatment of the problem of collective two-level multiple-output logic minimization through the detailed exposition of a novel efficient procedure for tackling this problem. This procedure is a purely map heuristic that generalizes the Karnaugh-map (K-map) procedure for single-output minimization. In fact, the K-map procedure can be viewed as a short-cut technique for obtaining the minimal sum of a switching function without deriving the set of all its prime implicants or its complete sum (Figure 1(a)). Likewise, the procedure proposed herein, is a short-cut technique for obtaining a minimal collective cover without the need for obtaining the set of all MOPIs or even its subset of PPIs (Figure 1(b)). The proposed procedure retains a pure map nature as it avoids any resort to algebraic or tabular techniques in the form of a presence function or cover matrix, respectively. It is of a good pedagogical value because of the pictorial insight it provides, and because it leads to a natural understanding of pertinent terminology such as MOPIs and PPIs. All it requires of a student is some repeated application of the familiar map heuristic, beside the implementation of a simple algorithm that handles visual interactions between various maps. These maps are grouped at distinct levels of a Hasse diagram, which is conveniently drawn in a K-map layout, so that any parent map is easily visualized as adjacent to all its children maps. Another point of strength of the current procedure is that, unlike the classical method, it does not need a final filtering of irredundant connections, as it avoids producing them from the outset.

![K-map procedure](a)

![Proposed procedure](b)

Fig. (1): A comparison of (a) single-output minimization, and (b) multi-output minimization, stressing the short-cut nature of the proposed procedure.
The procedure stated herein minimizes a set of functions given in disjunctive form and produces an AND-OR implementation. To obtain an OR-AND implementation, many texts (see, e.g., [15]) suggest the application of the same procedure to the complements of the original functions, with these complements being represented in disjunctive forms. However, to stress the concept of duality, an OR-AND implementation is obtained herein via a dual version of the current procedure applied to the original functions represented in conjunctive forms. The dual version of the procedure reads the same as the original one provided that any of the following words be replaced by the word trailing it in parentheses: implicant (implicate), product (sum), sum (product), term (alterm), disjunction (conjunction), conjunction (disjunction), 1 (0), 0 (1), AND (OR), OR (AND), asserted (negatively-asserted) and so on.

The rest of the paper is organized as follows. Section 2 outlines the classical method of collective minimization and sets the stage for understanding technical terms used in latter sections. Section 3 presents the proposed procedure which employs the standard heuristic of map coverage and implements an easy-to-understand algorithm for handling interactions between various maps. Two detailed examples are given in section 4 so as to illustrate the proposed procedure and its dual. Section 5 concludes the paper.

2. The Classical Exposition

Classically, multiple output logic minimization for the set of n functions \( S = \{f_1, f_2, ..., f_n\} \) starts by forming the set \( S_f = \{f_1, f_2, ..., f_n, f_1 f_2, ..., f_1 f_2 f_3, ..., f_1 f_2 f_3 ... f_n\} \) of the products of the functions taken 1, 2, 3, ..., or n at a time. The number of elements of \( S_f \) is \( \sum_{k=1}^{n} \binom{n}{k} = 2^n - 1 \). The next step is to construct the set \( S_H \) of multiple-output prime implicants (MOPIs) for \( S \), which is the union of the sets \( S_i \), 1 \( \leq i \leq 2^n - 1 \), where \( S_i \) contains all the prime implicants of one particular element of \( S_f \). Later, the set \( S_n \) is replaced by an important subset thereof, which is the set \( S_p \) of paramount prime implicants (PPIs). A PPI for the product \( P = f_{i_1} f_{i_2} ... f_{i_n} \) of functions is a prime implicant (PI) for \( P \) which is not a PI for any product of functions subsuming \( P \). Minimization is achieved by selecting appropriate subsets of \( S_p \) via tabular for algebraic techniques such as the Quine-McClusky cover matrix or the Petrick function technique, respectively. Finally, the resulting networks are examined for possible deletion of unnecessary connections and subsequently selecting the most compact network among the final networks.

3. The Proposed Procedure

We note the existence of a partial-order or a hierarchy among the member products of the set \( S_f \). This hierarchy is conveniently illustrated by an n-level Hasse diagram [16]. Figure 2 shows such a diagram for the case \( n = 4 \). A single arrow in the diagram indicates a parent-child relationship while a series of concatenated arrows indicates a more general and encompassing ancestor-descendent relationship, wherein a product \( P_i \) is a descendant (ancestor) of another product \( P_j \) when \( P_i \) subsumes (is subsumed by) \( P_j \). The arrangement
Fig. (2) : A Hasse diagram for the product members of $S_f$ when $n = 4$.

of the product members of $S_f$ in a Hasse diagram is implicit in the classical representation (see, e.g., [7]) and facilitates the derivation of all MOPIs and PPIs. Nodes of the Hasse diagram are at levels $i$ ranging from 1 to $n$. Level $i$ has $\binom{n}{i}$ nodes, each of which is a map representing a product of $i$ functions. The present heuristic constructs a hierarchy of maps each of which representing a member of the set $S_f$ of function products. This hierarchy is explicitly illustrated by a Hasse diagram, but this diagram is constructed in a K-map layout as shown in Figure 3 for the case $n = 4$. A map variable $Y_i \{ i = 1, 2, 3, 4 \}$ in Figure 3 is an indicator variable for the presence of the function $f_i$ in the pertinent product of functions. In fact, a cell of the K-map of Figure 3 is itself a map representing a product of functions in which the function $f_i$ is present if $Y_i = 1$ and is absent otherwise. The layout of Figure 3 is useful because it is more compact than that of Figure 2, and, more importantly, because any parent map appears adjacent (on a toroidal surface) to all its children maps. The current
procedure now covers asserted map entries using the well known map heuristic [7-15]. It then handles interactions between ancestor and descendent maps via the following algorithm:

\[
\begin{align*}
\text{For } \ell &= n \text{ downto } 1 \text{ do begin} \\
(\text{a}) & \text{ Cover all maps at level } \ell \text{ individually using the standard map heuristic.} \\
(\text{b}) & \text{ For every 1-cell that is covered in a certain map in step (a), the asserted entries of all corresponding cells in ancestor maps are crossed out or marked by a slash. A slashed 1 is equivalent to a don't care (d) in further processing.} \\
(\text{c}) & \text{ For every map at level } \ell, \text{ check any loop in a map } P_i \text{ for every slashed or crossed-out 1 it contains against the loop in a descendent map } P_j \text{ which caused the slash. If the latter loop does not cover any asserted cells other than those covered by the former one, e.g., if the latter loop is contained in the former one, then this latter loop is transferred to map } P_i/P_j \text{ and is checked for enlargement therein. Enlargement means the replacement of a certain loop by an admissible larger one that covers the original loop.} \\
(\text{d}) & \text{ Repeat step (c) for transferred loops until no further loop transfer is possible. Note that such a transfer is only possible when a transferred loop contains slashed 1's while covering the loop that caused the slash(es).}
\end{align*}
\]

Fig. (3) : The Hasse diagram in Figure 1 constructed in a K-map layout.
4. Illustrative Examples

4.1 Example 1

The approach outlined in section 3 is used in the collective design of an AND-OR multiple-output network for four 4-variable incompletely-specified functions given in decimal notation as follows:

\[
f_1(X_1, X_2, X_3, X_4) = \sum (1,2,4,6,8,9,11,13) + d(3,12),
\]

\[
f_2(X_1, X_2, X_3, X_4) = \sum (1,2,5,8,10,12,14,15) + d(3,7),
\]

\[
f_3(X_1, X_2, X_3, X_4) = \sum (1,2,5,6,8,10,15) + d(3,4,7,12),
\]

\[
f_4(X_1, X_2, X_3, X_4) = \sum (2,4,5,6,8,13,14) + d(7,12).
\]

Figure 4 shows K-map representations of these functions and their products taken 2, 3, or 4 at a time. There are \((2^4 - 1) = 15\) maps in Figure 4 arranged according to the K-map layout of a Hasse diagram (see Figs. 2 & 3), so that the ancestor-descendent relationship between any two maps is quite evident. In forming the products in Figure 4, the two identities \(1 \land d = d\) and \(0 \land d = 0\) are invoked. Figure 4 uses solid loops to represent the paramount prime implicants included in the final solution, and uses dotted loops to represent the prime implicants that are included temporarily but later enlarged within the same map or transferred to another map. First, we cover the sole map at the lowest level (level 4), the map of \(f_1f_2f_3f_4\), using essential PI loops \(P_1 = X_1\bar{X}_2\bar{X}_3\bar{X}_4\) and \(P_2 = \bar{X}_1X_2X_3\bar{X}_4\), and then cross out the asserted entries of the cells covered by these two loops in all ancestor maps, i.e., in all the remaining maps. Next, we climb up to level 3 and cover simultaneously the 4 maps of products \(f_1f_2f_3, f_1f_2f_4, f_1f_3f_4,\) and \(f_2f_3f_4\). We add the PI loop \(P_3 = \bar{X}_1\bar{X}_2X_4\) for map \(f_1f_2f_3\), cross out the asserted entries of the cell(s) covered by the \(P_3\) loop in the ancestor maps \(f_1f_2, f_1f_3, f_2f_3, f_1f_2, \) and \(f_3\). Since the single asserted entry within the \(P_3\) loop in the \(f_1f_2f_3\) map is not crossed out, there is no need to compare \(P_3\) with earlier PIs in a descendent map. The next map to consider is that of product \(f_1f_2f_4\) which has all its asserted entries crossed-out and requires no further coverage. Then, we add the PI loop \(P_4 = \bar{X}_1X_2\bar{X}_4\) for map \(f_1f_3f_4\), cross out the asserted entries of the cell(s) covered by the \(P_4\) loop in all ancestor maps, and similarly, the PI loop \(P_5 = \bar{X}_1X_2X_4\) is added in map \(f_2f_3f_4\) and the asserted entries of the cell(s) covered by the \(P_5\) loop in the pertinent ancestor maps are crossed out. Now, the six maps at level 2 are covered simultaneously. No loops are needed to cover the maps \(f_1f_2, f_1f_3, \) and \(f_3f_4\). The PI loop \(P_6 = X_1X_2\bar{X}_3\) is used with map \(f_1f_4\), the asserted entries within the \(P_6\) loop in each of the parent maps \(f_1\) and \(f_4\) are crossed out. The map \(f_2f_3\) requires two PI loops \(P_7 = X_2X_3X_4\) and \(P_8 = \bar{X}_2X_3\bar{X}_4\), and the asserted non-slashed entries of the cells within these loops in the parent maps \(f_2\) and \(f_3\) are crossed out. We note that the cell \(X_1\bar{X}_2X_3\bar{X}_4\) within \(P_8\) is crossed out since it has
already been covered by loop $P_2$ of map $f_1 f_2 f_3 f_4$. This means that $P_2$ is no more needed for $f_2$ or $f_3$ and should be moved from map $f_1 f_2 f_3 f_4$ to map $f_1 f_4$, wherein it is enlarged to loop $P_2' = \overline{X_1} X_3 \overline{X_4}$. This enlargement makes the loop cover cell $\overline{X_1} X_2 X_3 \overline{X_4}$ in addition to cell $\overline{X_1} \overline{X_2} X_3 \overline{X_4}$. If the additional cell $\overline{X_1} X_2 X_3 \overline{X_4}$ had not been slashed, we would have crossed out its entry in parent cells $f_1$ and $f_4$. The last map to be considered at level 2 is that of $f_2 f_4$ which is covered by PI loop $P_0 = X_1 X_2 \overline{X_4}$. The asserted entries of the cells within the $P_0$ loop in the parent maps $f_2$ and $f_4$ are crossed out. The final stage of the process is now reached through covering the four individual maps at level 1. Out of these maps $f_2$, $f_3$ and $f_4$ are already completely covered, while map $f_1$ is to be covered by a single PI loop $P_{10} = \overline{X_2} X_4$. This loop has a slashed entry that is covered by $P_3$ of the descendent map $f_1 f_2 f_3$, where the loop $P_3$ is contained in the loop $P_{10}$. Therefore, the output of the AND gate of $P_3$ should not be used as an input to the OR gate of $f_1$, and serves as an input for $f_2$ and $f_3$ only. Hence, loop $P_3$ is removed to the map $f_2 f_3$, wherein it is enlarged to $P_3' = \overline{X_1} X_4$. A comparison of $P_3'$ with earlier loops in descendent maps reveals

Fig. (4) : The 15 K-maps used in the AND-OR multiple-output minimization process of Example 1. The maps are arranged according to the K-map layout of Figure 3.
that it covers loop \( P_3 \) in map \( f_2f_3f_4 \). Therefore, \( P_3 \) is not needed for \( f_2 \) and \( f_3 \) and should be transferred to the \( f_4 \) map, wherein it is enlarged to \( P'_3 = \bar{X}_1X_2 \). The new loop \( P'_3 \) in the map \( f_4 \) covers loop \( P_4 \) in the map \( f_1f_2f_4 \), and hence this latter map is transferred to the \( f_1f_3 \) map. The design is now complete with the resulting network achieving a minimal number of 14 gates (as a primary objective) and a corresponding minimal number of 47 connections (as a secondary objective), as can be verified by the logic minimizer Espresso [4,11]. To obtain the minimal 2-level network, the aforementioned AND-OR network should be compared to the OR-AND network that can be obtained via the dual version of the present procedure. Solution for this OR-AND network is straightforward, and lacks most of the illustrative details of the AND-OR case (See Figure 5). The OR-AND network turns out to be the minimal 2-level network since it requires only 13 gates and 45 input connections.

---

**Fig. (5)**: The 15 K-maps used in the AND-OR multiple-output minimization process of Example 1. The maps are arranged according to the K-map layout of Figure 3.

### 4.2 Example 2
This example illustrates the dual procedure for the collective design of an OR-AND multiple-output network for three 4-variable incompletely specified functions given in decimal notation as follows:

\[ f_1(X_1, X_2, X_3, X_4) = \Pi(3,5,6,7,8,13,14,15) + d(4,10), \]  

(5) \[ f_2(X_1, X_2, X_3, X_4) = \Pi(1,3,4,6,7,8,9,12) + d(2,10,15), \]  

(6) \[ f_3(X_1, X_2, X_3, X_4) = \Pi(0,2,3,6,7,8,12) + d(5,15). \]  

(7)

Figure 6 shows K-map representations of these functions and their sums taken 2 or 3 at a time. The 0 and d entries in these maps are shown explicitly, while the 1-entered cells are left blank. In forming the sums in Figure 6, the two identities \(0 \lor d = d\) and \(1 \lor d = 1\) are used. Figure 6 uses solid loops to represent the paramount prime implicates included in the final solution, and uses dotted loops to represent the prime implicates that are included temporarily but later enlarged within the same map or transferred to another map. Among the implicates shown \(P_1\), \(P_2\) and \(P_3\) are circled since they have been transferred (with possible enlargement) to pertinent ancestor maps. The resulting OR-AND network uses 12 gates and 35 input connections, and can be shown to be minimal as verified by Espresso. The minimal 2-level network; however, is the dual (AND-OR) network and requires only 10 gates and 31 input connections (See Figure 7).

Fig. (6) : The 15 K-maps used in the OR-AND multiple-output minimization process of Example 2.
The maps are arranged (with duality) according to the layout of the upper half of the K-map of Figure 3.

Figure (7) : The 7 K-maps used in the AND-OR multiple-output minimization of Example.

5. Conclusions

This paper has presented a novel and efficient procedure for two-level multiple-output logic minimization. This procedure offers a powerful visual aid that enables students to quickly construct minimal or near minimal solutions by sharing terms (or alterms) among individual functions through a collective handling of the maps of these functions and the maps of their various products (or sums). All the maps considered are partially ordered by a Hasse diagram constructed in a K-map layout so that a parent-child relation between two maps is manifested as an adjacency between their K-map cells. Two-way interactions among maps at different levels of the Hasse diagram are easily and pictorially implemented. A solution is achieved neatly and quickly without any need to (a) generate the set of all MOPIs or that of all PPIs, (b) construct a cover matrix or a presence function, or (c) inspect resulting connections for possible deletions of redundant ones. In short, the technique proposed herein is a pure map procedure, and is believed to be the natural generalization of the single-output K-map procedure to the multiple-output case.

The heuristic in Hong et al. [3] is similar to the present procedure, since both of them avoid the generation of all PPIs. However, while the procedure in Hong et al. [3] obtains its final solution from an initial one by using certain subprocesses for iterative improvement, the current procedure does not use iteration as its solution evolves gradually through the employment of a direct algorithm.

The current procedure generalizes the map technique of single-output minimization to handle the more difficult problem of multiple-output minimization as it originally stands, i.e., without attempting to simplify or reduce it. By contrast, there are other techniques which proceed by initially converting the original problem into an equivalent one. The equivalent problem could be one of the minimization of a single incompletely-specified auxiliary function with an additional input dimension equal to the number of outputs $n$ [1,3,6]. Alternatively, the equivalent problem may deal with a constrained minimization of a single
completely-specified auxiliary function with an additional input dimension of value \((n-1)\) only [17]. Finally, there is the possibility of an equivalent problem of obtaining the solution of a system of Boolean equations [18-20].

In conclusion, we reiterate that, from the pedagogical point of view, the present procedure should be viewed as a supplement rather than a replacement to automated computational experience in logic design. Such an experience may be obtained from a variety of software educational packages including different versions of Espresso [4,11], LogicAid [15], and MAX+plus [14].

References


