# Multilevel Logic Minimization Using Implicit Don't Cares

KAREN A. BARTLETT, ROBERT K. BRAYTON, FELLOW, IEEE, GARY D. HACHTEL, FELLOW, IEEE, REILY M. JACOBY, CHRISTOPHER R. MORRISON, RICHARD L. RUDELL, ALBERTO SANGIOVANNI-VINCENTELLI, FELLOW, IEEE, AND ALBERT R. WANG

*Abstract*—**This paper describes a new approach for the minimization of multilevel logic circuits. We define a multilevel representation of a block of combinational logic called a Boolean network. We propose a procedure, ESPRESSO_MLD, to transform a given Boolean network into a prime, irredundant, and "R-minimal" form. This procedure rests on the extension of the notions of primality and irredundancy, previously used only for two-level logic minimization, to combinational multilevel logic circuits. We introduce the new concept of R-minimality, which implies minimality with respect to cube reshaping, and demonstrate the crucial role played by this concept in multilevel minimization. We give theorems which prove the correctness of the proposed procedure. Finally, we show that prime and irredundant multilevel logic circuits are 100-percent testable for input and output single stuck faults, and that these tests are provided as a by-product of the minimization.**

## I. Introduction

IN THIS PAPER an "efficient" procedures is presented for obtaining high-quality heuristic **multilevel** logic minimization results for a **given** logic network and making it **much more testable** than merely 100-percent testable for the conventional input and output single stuck faults. The approach is based on determining the complete don't care set for each 2-level function embedded in a network of such functions. Once this is done, a 2-level minimizer can be used to minimize the subfunction. The high degree of testability achieved by this approach requires no separate test generation processing, since all tests are produced as a by-product of well-known 2-level minimization procedures.

We use the term "efficient" advisedly. It is clear that all procedures for reducing either two-level or multilevel Boolean networks into prime and irredundant form must be NP-complete or co-NP-complete (i.e., all procedures have $O(2^n)$ complexity). But given this ominous sign of intractability, fairly large Boolean networks may yet be minimized (up to 60 inputs at the time of this writing) on available workstation size computers. In this context we use "efficient" only in comparison to the trivial approach of iteratively calling an automatic test generation tool and modifying the network by hand each time a nontestable fault is discovered (cf. Section VI-B below). Here the "efficiency" of the presented procedures is derived from the application of two-level logic minimization procedures of proven efficiency to the multilevel case (cf. the discussion at the beginning of Section III). We expect that these procedures will excel in applications where individual nodes of the Boolean may have large sum-of-products representations.

The subject of 2-level logic minimization is well developed and well understood [5]. We know exact techniques which provide minimum representations of the given logic (cf. [23], [11], [28]). We also have seen two generations of programs for generating near minimum logic representations (cf. SHRINK [25], MINI [20], ESPRESSO_II [7], ESPRESSO_IIC [28], ESPRESSO_MV [29]). We also know how to determine if two functions are equivalent and when we have irredundant logic (cf. [25], [30], [18]). These notions have been extended to multi-output functions, and functions of multi-valued variables [20], [29]. Significant progress has been made on the state-assignment problem and other encoding problems using two-level logic [13]. In short, this is a well-developed science.

In contrast, multilevel minimization is less structured, more difficult, and relatively new. A worthwhile long-term goal is to bring understanding of this subject up to the level of science currently established for two-level minimization. Multilevel minimization as a science suffers from the same things that make it attractive for implementing logic, namely, it is very flexible. Hence, the problems are not so well defined. In contrast, for two-level minimization, we often have in mind a PLA implementation and, therefore, the minimization problem (i.e., minimize the number of product terms) can be abstracted and made largely independent of the technology of the implementation.

Multilevel synthesis has the advantage over PLA syn-

Manuscript received December 8, 1986; revised November 17, 1987. This work was supported in part by the National Science Foundation under Grant NSF DMC-8419744, by the General Electric Company, and by the IBM Corporation. The review of this paper was arranged by A. J. Strojwas, Editor.

K. A. Bartlett was with the University of Colorado at Boulder, Boulder, CO. She is now at Seattle Silicon Technology, Inc., Bellevue, WA 98005.

R. K. Brayton was with IBM T. J. Watson Research Center, Yorktown Heights, NY. He is now with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

G. D. Hachtel, R. M. Jacoby, and C. R. Morrison are with the Departments of Electrical and Computer Engineering and Computer Science, University of Colorado at Boulder, Campus Box 425, Boulder, CO 80309.

R. L. Rudell, A. Sangiovanni-Vincentelli, and A. Wang are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 9472υ.

IEEE Log Number 8719390.

thesis in that it is good for representing and implementing any type of logic. Typically, logic has been divided into two groups, control and data-flow logic, with control logic perceived as suitable for PLA implementation while data-flow usually requires multilevel logic (sometimes called random logic). This sometimes forces an unnatural decomposition, with the control logic made by PLA generators and the data-flow hand designed or obtained from parameterized libraries. Multilevel logic is suitable for all types of logic, and automatic and optimal multilevel logic synthesis forces no such dichotomy on the user. In many applications it is more suitable in fact to mix the two types of logic, for example, for more optimal logic (i.e., by capturing mutual don't care situations), because of layout considerations, or for easier specification at the functional level.

Historically, the literature on multilevel minimization consists mainly of results on factoring (i.e., decomposing) a single Boolean function [22], [9], [4]. Emphasis in the present paper is on optimizing a *given* (i.e., already decomposed) structure. Since multilevel logic is more difficult to optimize, most of the designs involving multilevel logic have been carried out by hand, using a "bag of tricks." Recently, several approaches to automatic multilevel logic optimization have been proposed and have found application in a variety of technologies [12], [6], [17], [14], [21]. In all these approaches, emphasis has been placed on efficient decomposition and factorization techniques which create a certain multilevel logic structure, which in this paper we call a *Boolean network*. Creation of this structure establishes the overall architecture of the logic to be implemented, and roughly establishes the final point to be reached on the area–delay tradeoff curve; it has been shown in [2] that this process alone seldom comes close to realizing the full benefits of minimization. However, two major tasks still need to be accomplished before the full potential of a given decomposition may be reached: a) making the Boolean network minimal with respect to its own intrinsic structure (i.e., finding an optimal point on the area/delay tradeoff curve) and b) making it testable. Fortunately these objectives are not mutually exclusive, and in fact are profoundly related and can be simultaneously realized.

The connection between logic minimization and test generation is well known [27], [31], [26], but has not been systematically investigated. Even modern books on multilevel circuit design [15] and on test generation and design for testability [16] contain no mention of this relationship. The connection rests on the simple observation that the absence of a test is associated with *redundancy* in the Boolean network. In 2-level logic the sources of redundancy are well understood and efficient algorithms are available for making a 2-level representation of an incompletely specified logic function prime and irredundant. However, the equivalent concepts for multilevel representations have not been fully developed, and only the D-algorithm and its variants [16], [24], [3] have been used to identify and remove redundancy. These algorithms often incur great computational expense. An efficient algorithm is badly needed since none of the factorization and decomposition techniques yet proposed is guaranteed to produce irredundant logic. Such an algorithm is the objective of our research, and would have great potential impact because of the testability requirement.

We propose in this paper a don't care algorithm for making a Boolean network prime, irredundant, and R-minimal (this last property is explained below). Further, among different possible prime and irredundant Boolean network representations of a given logic function, the proposed approach utilizes two techniques to choose a superior one. These techniques are: 1) utilization of the EXPAND and IRREDUNDANT_COVER heuristics of the ESPRESSO-II 2-level logic minimizer, and 2) development of ESPRESSO's REDUCE algorithm (which is a limited form of the powerful but expensive decomposition technique known as *Boolean division* [6]) to make the Boolean network R-minimal. Further, we shall show that primality can be regarded as a special type of irredundancy, and that prime and irredundant Boolean networks are 100-percent testable for the usual single stuck faults, as well as for other types of "internal" stuck faults. Thus we believe that the networks produced by our procedures are the first to be synthesized with guaranteed 100-percent testability, as well as minimality comparable to that available with state-of-the-art 2-level minimizers. For example, the work of [26], although based on the D-algorithm, did not claim complete testability, and was designed for the 2-level case. Even if that approach were extended to the multilevel case, it would not be able to promote a general Boolean network to prime, irredundant, and R-minimal status.

Briefly stated, R-minimality means that no one of the individual 2-level functions in the Boolean network can be reexpressed in terms of one or more of the others to map the given prime and irredundant Boolean network into another one with less logic cost. This important point is illustrated in Fig. 1, which shows 4 equivalent Boolean networks. The network of Fig. 1(a) has 3 nodes (gates, functions), and is neither prime nor irredundant. Node $F_3$ does not have tests for the following input stuck-at faults: $x_1$, and $x_2$ stuck-at-1 and $y_2$ stuck-at 0 (at the inputs of $F_3$). The equivalent Boolean network of Fig. 1(b) is prime, irredundant, and 100-percent testable and requires 9 literals and 5 product terms. It is *not* R-minimal. The equivalent network of Fig. 1(c) is similarly prime, irredundant, and testable but, by virtue of a call to REDUCE, is R-minimal, and requires only 2 nodes, 5 literals and 3 product terms. This example, as well as the concept of R-minimality, will be examined more closely in Section III below.

We further show that the problem of transforming a given Boolean network into prime, irredundant, and R-minimal form can be reduced to that of solving the same problem on a sequence of 2-level, single-output representations of the incompletely specified logic functions realized at each node of the Boolean network. This is achieved

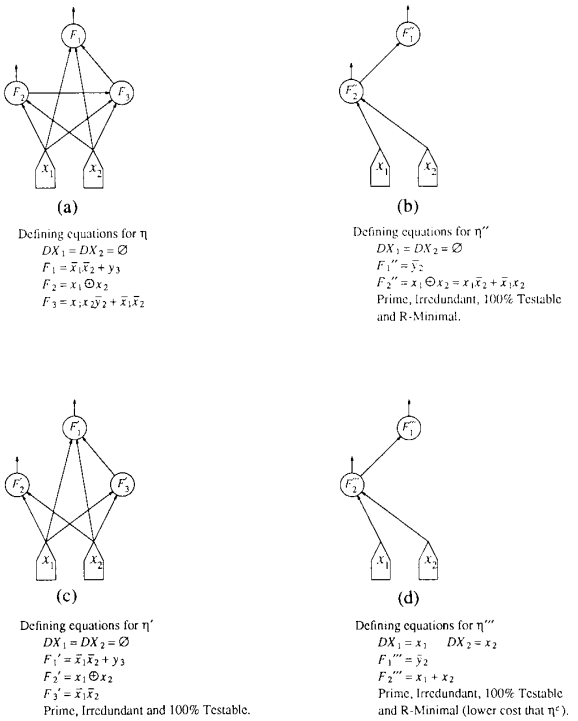



Fig. 1. Progressively optimized Boolean networks.

by determining a representation of the don't care set for each of these incompletely specified functions.

Our approach is rigorous in the sense that we prove that at the end of the proposed procedure, the Boolean network produced is definitely prime, irredundant, and probably R-minimal. This network is not only 100-percent testable, but the stuck fault test vectors "fall out" as a straightforward by-product of the aforementioned minimization of the component 2-level functions.

The sequel begins in Section II with a discussion of basic definitions and background which focuses mainly on the Boolean network concept. In Section III we introduce the topic of multilevel logic minimization and discuss an example in detail. Section III gives a characterization of the don't care sets, shows how to construct them, and proves that this construction is correct. Section IV discusses the proposed procedure, which we call ESPRESSO_MLD (ML is for MultiLevel). In Section V we discuss some experimental results, and in Section VI we present our testability results and discuss, in detail, the connection between multilevel logic minimization and testability. In Section VII, we present conclusions and discuss the prospects for future research.

## II. Background and Basic Definitions

The primary object in our approach to multilevel logic optimization is a Boolean network, defined formally below, which is a technology-independent multilevel structure for representing an *incompletely specified logic function* [5]. The Boolean network may be regarded as an abstraction of an interconnected set of logic gates, as might be specified by a netlist of standard cells. Considered in isolation, each gate in this network realizes a completely specified logic function, but in the context of the network, it realizes an incompletely specified subfunction. Each interconnection represents a signal net associated with the output of one of the gates. Before formally defining a Boolean network, we briefly introduce the concepts of a) completely and incompletely specified Boolean functions and b) their representations.

An *incompletely specified Boolean function* ( $f$, $d$, $r$) is a set of 3 *completely specified functions* $f: B^t \rightarrow B$ (the on set), $d: B^t \rightarrow B$ (the don't care set), and $r: B^t \rightarrow B$ (the off set). The minterms of $f$, $d$, and $r$ completely *partition* the vertices of the Boolean $t$-cube $B^t$. Here $f$ may be thought of as a function, $f(v)$, of a $t$-dimensional vector $v = (v_1, v_2, \cdots, v_t)$. A simple incompletely specified logic function is illustrated in Fig. 2. In this example $v = (v_1, v_2, v_3)$, $t = 3$, and $f(v) = 1$ for $v \in \{000, 101, 010, 111\}$, else $f(v) = 0$; $d(v) = 1$ for $v \in \{100, 110\}$, else 0; and $r(v) = 1$ for $v \in \{001, 011\}$, else 0. An incompletely specific logic function reduces to a completely specified function when $d = \emptyset$, i.e., there is no don't care set.

Note that a completely specified function $f(v)$ may be *independent* of certain of the $v_i$, and this fact is usually reflected in the selection of a representation $F$ of $f(v)$. The variables *explicitly* represented in $F$ are called the *support* of $F$. One representation of $f$ is the sum of products form, e.g.,

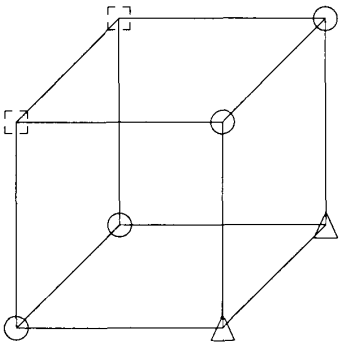
$$F = v_1\bar{v}_3 + \bar{v}_1v_3 + \bar{v}_2 + v_2$$

which is also called the disjunctive normal form. Note that here the *support* of $F$ is $\{v_1, v_2, v_3\}$, and that a variable which a function does *not* depend on, like $v_2$ in the above example, may appear explicitly in the support of the function. Other representations are possible and significant, e.g., conjunctive form or factored form [22]. However, in this paper we use disjunctive form since we rely heavily on 2-level, sum-of-products-based logic minimization procedures such as ESPRESSO_II as subprocedures in our approach to multilevel logic minimization.

Product terms like $v_1\bar{v}_3$ and $\bar{v}_1v_3$ will be called *cubes* in the sequel. Each cube consists of a set of literals, and each literal appears in one of the two forms $v_i$ or $\bar{v}_i$. If "$v_i$" appears it stands for the predicate "$v_i = 1$," and if $\bar{v}_i$ appears it stands for the predicate "$v_i = 0$." Thus the cube $v_1\bar{v}_3$ stands for the conjunction of predicates $v_1 = 1$ and $v_3 = 0$. A cube with fewer literals has, of course, more vertices on the Boolean $t$-cube. In this sense we can view the cube $v_1\bar{v}_3$ as the intersection of the subcubes (half spaces of the Boolean $t$-cube) $v_1 = 1$ and $v_3 = 0$ (cf. Fig. 2, in which the dimension of the Boolean $t$-cube is $t = 3$).

*Definition 1 (Boolean Networks)*

As illustrated in Fig. 3, a *Boolean network*, $\eta$, is a pair $(F, PO)$, where $F = \{F_j, j = 1, 2, \cdots, m\}$ is a set of $m$ given representations of the on sets $f_j$ of incompletely

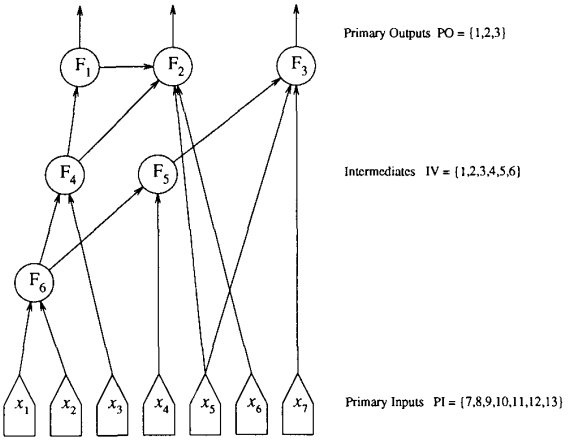


= ON = 1 $F = \bar{x}_1\bar{x}_3 + x_1x_3$

= OFF = 0 $R = \bar{x}_1x_3$

= DON'T CARE = 0 or 1 $D = x_1\bar{x}_3$

Fig. 2. Incompletely specified Boolean function.




$m = 6 = |IV|$
$n = 7 = |PI|$

$F_j: B^{m+n} \to B$
$y_j = F_j(y, x) \quad 1 \le j \le m$

$v_j = y_j \quad 1 \le j \le m$
$v_{j+m} = x_j \quad 1 \le j \le n$

Fig. 3. A multilevel Boolean network.

specified functions $(f_j, d_j, r_j)$, $j = 1, 2, \cdots, m$. With each $F_j$ is associated a "local output" logic variable $y_j$, in the set $IV = \{y_1, y_2, \cdots, y_m\}$, which we call the *intermediate variable* set. The specified primary output set, $PO \subseteq \{1, 2, \cdots, m\}$, identifies the subset $\{y_i \mid i \in PO\} \subseteq IV$ of the outputs of the $F_i$ as observable *primary outputs* of the Boolean network. It is convenient to refer to this subset as the primary output vector $z$, defined so that $z_k = y_{PO(k)}$, $k = 1, 2, \cdots, p$, $p = |PO|$.

Since $\eta$ is completely determined by the pair $(F, PO)$, we write $\eta = (F, PO)$. However, $\eta$ has further structure, determined by the *support sets* $SUPP(F_j)$ of the representations, $F_j$. These sets determine the structure of a *directed graph*, $G = (N, E)$, with nodes

$$N = \{1, 2, \cdots, m, m+1, \cdots, t\}$$

$$t = \left| \bigcup_{j=1}^{m} SUPP(F_j) \right| > m.$$

With each node $i \in N$, we associate a logic variable $v_i$. With the first $m$ nodes of $N$ we associate the representation $F_i$ and its corresponding variable $y_i$, so that $v_i = y_i$, $i = 1, 2, \cdots, m$. (This duplication of notation is quite useful in the sequel). However, *no* $F_i$ is associated with the last $n = t - m$ logic variables in the vector $v$. Instead, these nodes are identified with *primary inputs* of $\eta$, and are associated with duplicate logic variables $PI = \{x_1, x_2, \cdots, x_n\}$. Thus $v_{m+i} = x_i$ $i = 1, 2, \cdots, n$. Note that the vector $v$ can be viewed as the catenation $v = (y, x)$.

For each node $j \in N$, we define the *fan-in set* (for short *fan-in*) $FI_j$, as follows. If $j \le m$ (intermediate variables), $FI_j = \{i \mid v_i \in SUPP(F_j)\}$, but if $j > m$ (primary inputs) $FI_j = \varnothing$. Thus the primary input nodes are terminal nodes of the directed graph $G$. The edge set $E$ of $G$ contains directed edge $(i, j)$ if node $i$ is in the fan-in of node $j$, i.e., $i \in FI_j$. Then we may define the *fan-out*, $FO_j$, of node $j$ to be the set of all nodes $i \in N$ for which there is an edge $(j, i) \in E$. Similarly, we define the *transitive fan-out*, $TFO_j$, to be the set of all nodes $i \in N$ such that there exists a (directed) path from $j$ to $i$ in $G$, and the *transitive fan-in*, $TFI_j$, to be the set of all nodes $i \in N$ for which there exists a path from $i$ to $j$ in $G$. By convention, $j \in TFO_j$, but $j \notin TFI_j$. □

It is important to note that although $F_j$ depends explicitly only on the variables $v_k \in SUPP(F_j)$, we may formally view each $F_j$ as a function of the entire vector $v = (y, x)$ (recognizing, of course, that $f_j$ may be functionally independent of many of the $v_k$). Thus we may write

$$v_j = y_j = F_j(v) = F_j(y, x)$$
$$= F_j(y(x), x), \quad j = 1, 2, \cdots, m \quad (2.1)$$

as the basic constitutive relations of $\eta$. Note that as indicated in the last identity, if (2.1) is satisfied for $j = 1, 2, \cdots, m$, then the solution vector $v(x) = (y(x), x)$ is the vector of values appearing at the nodes of the Boolean network in response to the primary input vector $x$. In particular, $z(x)$ represents the values at the primary outputs of $\eta$ in response to $x$, i.e., the same values that would have been obtained by logic simulation of the vector $x$. Thus given that (2.1) is satisfied, $z(x)$ represents the IO (input/output) map of $\eta$.

Thus a Boolean network $\eta$ is a *representation* of a set of incompletely specified functions, $(f(i), d(i), r(i))$, one for each primary output $z_i(x)$. Thus $z_i(x)$ can be regarded as the "IO map" from $PI$ to $PO_i$ of the Boolean network $\eta$. A representation, $DX_i$, of the completely spec-

ified "don't care" function $d(i)$ must come from the system designer. We refer to $DX_i$ as the *external* don't care set, which arises from two phenomena. First, for a particular design the designer may decree that a particular primary input vector $x \in B^n$ *will never occur*. The vector $x$ constitutes a don't care minterm, and such minterms are don't care for *all* primary outputs. The set of all such minterms is labeled *DXP*. Second, the designer may state that for any of the outputs $z_i$, $i \in PO$, the value of $z_i$ *will not be used* for a set of primary input vectors (minterms) in the set $DXO_i$. Thus for each primary output the total external don't care set can be written

$$DX_i = DXP + DXO_i, \qquad i = 1, 2, \cdots, p = |PO|. \tag{2.2}$$

Equation (2.2) gives a representation of the completely specified functions $d(i)$, $i = 1, 2, \cdots, p$ (don't care sets) associated with the primary outputs of a Boolean network. A principal objective of the sequel is to identify representations of the analogous don't care sets for each of the incompletely specified functions associated with the intermediate variables of a given Boolean network (and their corresponding internal nodes).

A key concept in logic optimization is that of Boolean equivalence. In the multilevel context, we wish to establish when a given Boolean network, $\eta$, can be replaced by another one, $\eta'$, with an equivalent IO map, $z'(x) \equiv z(x)$. That is, the relation between primary inputs and primary outputs is preserved. Thus $\eta'$ represents the same set of incompletely specified functions $(f(i), d(i), r(i))$, $\forall i \in PO$.

*Definition 2 (Equivalence)*

Boolean networks $\eta = (\boldsymbol{F}, PO)$ and $\eta' = (\boldsymbol{F}', PO')$ are said to be equivalent (written $\eta = \eta'$) if there exists a permutation $q$ of $\{1, 2, \cdots, p\}$ such that for each primary output $z_i'(x)$ in $PO'$, $z_i'(x) = z_{q(i)}(x)$ for all $x \notin DX_{q(i)}$. □

The permutation, $q$, in Definition 2 is needed to identify the proper correspondence between the primary outputs of the two Boolean networks, which may be very different structurally. For simplicity, we assume, without loss of generality, that $q$ is the identity permutation.

We have, in separate research, demonstrated that a more general definition of equivalence can be stated, but this requires more information about the external environment than just the external single-output don't care set $DX_i$ for $i \in PO$. For example, the external environment may have outputs $i$ and $j$ connected only to the inputs of an exclusive or gate, in which case the environment would be unable to distinguish between outputs $y_i = 1$, $y_j = 0$, and $y_i = 0$, $y_j = 1$. We will treat this more general definition of the concept of don't cares in a later paper. For now, we observe that this generalization will enable us to handle Boolean networks with nodes having multiple-output Boolean functions rather than just single-output Boolean functions.

Note that Definition 2 requires only that the *primary outputs* of two Boolean networks match for each *care* input vector. In particular, it is *not* necessary to have identity or even correspondence between the intermediate variables of the two networks. For example, a 4-level network could be equivalent to a 2-level network. The 2-level network specified by the following equations is equivalent to those of Fig. 1:

$$F_1 = x_1x_2 + \bar{x}_1\bar{x}_2 \tag{2.3a}$$

$$F_2 = x_1\bar{x}_2 + \bar{x}_1x_2. \tag{2.3b}$$

The task of minimizing a Boolean network $\eta$ consists of iteratively transforming $\eta$ into an equivalent network $\eta'$ where $\eta'$ is smaller than $\eta$ in some sense. Two properties of minimality, similar to those for the classical 2-level case, are especially relevant to the multilevel case (since Boolean networks having these properties are shown below to be 100-percent testable for stuck faults).

*Definition 3 (Prime and Irredundant Boolean Networks)*

Given a Boolean network $\eta = (\boldsymbol{F}, PO)$, a cube $c$ of the 2-level representation of $F_i$ is *prime* if no literal of $c$ can be removed without causing the resulting network $\eta'$ to be *not* equivalent to $\eta$. In more formal terms, $\eta' = (\boldsymbol{F}', PO)$ is a Boolean network for which $F_j' = F_j$, $\forall j \neq i$ and $F_i' = (F_i - \{c\}) \cup c'$, where $c'$ is $c$ with one of its literals removed. Similarly, a cube $c$ of $F_i$ is *irredundant* if $c$ cannot be removed from the representation of $F_i$ without causing the resulting network $\eta'$ to be *not* equivalent to $\eta$. A *Boolean network* $\eta = (F, PO)$ is said to be *prime* if all the cubes in each of the representations $F_i$ of $\eta$ are prime, and *irredundant* if all of these cubes are irredundant. □

Note that these two concepts are associated with local minima of a cost function which is nondecreasing in the total number of cubes and literals required to represent the incompletely specified logic functions, realized by the given Boolean network.

We complete this section by defining the *cofactor* operation on both representations of functions and on Boolean networks.

*Definition 4 (Cofactor Operation)*

The *cofactor* of a sum-of-products representation, $F = \{c_i\}$, of a Boolean function with respect to a literal $v_j$ is defined to be

$$(F)_{v_j} = \bigcup_i (c_i)_{v_j}.$$

Here if literal $v_j$ is contained in $c_i$, $(c_i)_{v_j}$ is just $c_i$ with literal $v_j$ deleted, else if literal $\bar{v}_j$ appears in $c_i$, $(c_i)_{v_j} = \varnothing$. If neither $v_j$ or $\bar{v}_j$ appears in $c_i$, then $(c_i)_{v_j} = c_i$.

The *cofactor of a Boolean network* $\eta = (\boldsymbol{F}, PO)$ with respect to a literal $v_j$ is a Boolean network, $\eta_{v_j} = (\boldsymbol{F}_{v_j}, PO)$, where $\boldsymbol{F}_{v_j} = \{(F_i)_{v_j}\}$ is the set of cofactors of the representations $\{F_i\}$ of the original Boolean network. We denote the vector of logic variables in this cofactored network to be $(v)_{v_j}$, with components $(v_1)_{v_j}, (v_2)_{v_j}, \cdots, (v_t)_{v_j}$.

Similar definitions apply when the cofactor is with respect to the literal $\bar{v}_j$. □

This definition is crucial to computation of the representation, $D_i$, of the don't care sets, $d_i$, of the incompletely specified functions ( $f_i$, $d_i$, $r_i$ ) which implicitly define the structure of the Boolean network. Note in particular that the edges of the Boolean network $\eta_{v_j}$ are defined by the *support* of the $(F_i)_{v_j}$, from which the variable $v_j$ is now totally missing. Thus each node $i$ in the fan-out of node $j$ in $\eta$ is disconnected from node $j$ in $\eta_{v_j}$.

## III. Multilevel Logic Minimization

Given a Boolean network $\eta$, it is of obvious interest to obtain an equivalent prime and irredundant network $\eta'$. One possible procedure to obtain this simplification is to examine each cube as well as each literal in first encounter order, and for each such cube or literal to construct a simplified network $\eta'$, identical to $\eta$ except for the removal of the selected cube or literal. Then Definition 2 may be embodied in a computer program such as [18] to check if $\eta = \eta'$. If so, the cube or literal is redundant, and can be removed from $\eta$. If no cube or literal can be so removed, then the resulting Boolean network is prime and irredundant (Definition 3). This elementary minimization procedure is the one used to obtain the Boolean network of Fig. 1(b) from that of Fig. 1(a). This procedure is intimately related to the way in which the basic D-algorithm (and its variants) [16] is used in test generation algorithms. However, such elementary procedures are not efficient, and do not lead to high-quality minimization results (compare Fig. 1(b) and (c)).

We present here a more efficient procedure which appears to give high-quality multilevel minimization results. The procedure is based on 1) computing, for each intermediate node $j$ in $\eta$, a representation $D_j$ of the don't care set $d_j$ of the incompletely specified function ( $f_j$, $d_j$, $r_j$) associated with node $j$, $j = 1, 2, \cdots, m$; and 2) minimizing the representation $F_j$ of $f_j$ with respect to $D_j$ by calling an efficient 2-level minimizer (we use the ESPRESSO-IIC program [5] for this purpose) to render $F_j$ prime, irredundant, and approximately R-minimal. Note that the properties of primality and irredundancy arise from *both* the representation $F_j$ and the Boolean network, $\eta$, in which it is embedded. This is because, considered in isolation, the $F_j$ are representations of *completely specified* functions, but embedded in the network, they are representations of *incompletely specified* functions, i.e., they have a don't care set. Thus a network of individually prime and irredundant functions, such as that shown in Fig. 1(a), may be neither prime nor irredundant.

To discover such redundancies, we identify the don't care sets generated by the structure of a Boolean network. We illustrate this by identifying a representation, $D_3$, of the don't care set $d_3$ of node 3 of the Boolean network of Fig. 1(a). For reasons discussed later in this section, we can show that the 5-cube set

$$D_3 = \bar{y}_2(x_1\bar{x}_2 + \bar{x}_1x_2) + y_2(x_1x_2 + \bar{x}_1\bar{x}_2) + \bar{x}_1\bar{x}_2$$

is a valid representation of $d_3$, assuming $DX_i = \varnothing$, $i \in PO = \{1, 2\}$, i.e., that the Boolean network $\eta$ has no external don't care set. Since $\bar{x}_1\bar{x}_2 \in D_3$ it is clear that cube $\bar{x}_1\bar{x}_2$ of $F_3$ is redundant and can be deleted. Further, since $y_2x_1x_2 \in D_3$, and $y_2x_1x_2 + \bar{y}_2x_1x_2 = x_1x_2$, literal $\bar{y}_2$ may be dropped from cube $x_1x_2\bar{y}_2$ in $F_3$. After these two typical minimization steps we have derived the prime and irredundant network $\eta'$ (Fig. 1(b)) from $\eta$ and have in fact shown that $\eta' = \eta$.

The problem we faced in our research was how to compute a representation $D_j$ in the general case. To show how this is done, we define two additional don't care sets *DIV* (the *intermediate variable* don't care set, common to all nodes $j = 1, 2, \cdots, m$) and $DT_j$ (the *transitive fan-out* don't care set, which is specific to node $j$ ). These are to be appended to the appropriate *external* don't care set to form $D_j$, as discussed below. We begin by defining *DIV*.

*Definition 5*

The "*overall*" *intermediate variable don't care set*, *DIV*, is defined by

$$DIV = \sum_{j=1}^{m} DIV_j \tag{3.1a}$$

where

$$DIV_j = y_j\bar{F}_j + \bar{y}_jF_j = y_j \oplus F_j. \tag{3.1b}$$

Note by DeMorgan's law, we have

$$\overline{DIV} = \prod_{j=1}^{m} (y_j \equiv F_j) = \prod_{j=1}^{m} (\bar{y}_j\bar{F}_j + y_jF_j). \quad \square \tag{3.1c}$$

It is thus clear that for any vertex in $v \in B^t$ (represented by the overall vector $v$ of $\eta$) which satisfies (2.1) for $j = 1, 2, \cdots, m$, it follows that $v \in \overline{DIV}$. Conversely, if *any* of the equations $y_j = F_j(v)$ is *not* satisfied, we have $v \in DIV$. These observations will be used repeatedly in the sequel. Note that in the above example, the first 4 terms in $D_3$ represent the contribution of $DIV_2$.

We note that each member of the $C_{10}$ and $C_{00}$ "forcing" sets defined in [32] corresponds to two literal implicants of *DIV*. Thus their recurrence relation provides, in linear time, a proper subset of *DIV*.

The origins of the *transitive fan-out* don't care set representation $DT_j$ associated with node $j$ are subtler, so a detailed discussion of these don't care terms is deferred until later in this section. However, in simple cases such as that exemplified in Fig. 1, the transitive fan-out don't care set has a straightforward construction. Suppose that $\forall i \in FO_j$,

$$i \in PO \quad \text{and} \quad FO_i = \varnothing. \tag{3.2a}$$

Then

$$DT_j = \bigcap_{i \in FO_j} E_{ij} \tag{3.2b}$$

where

$$E_{ij} = \left((F_i)_{y_j} \equiv (F_i)_{\bar{y}_j}\right) = (F_i)_{y_j})\,(F_i)_{\bar{y}_j} + \overline{(F_i)_{y_j}})\,\overline{(F_i)_{\bar{y}_j}}. \tag{3.2c}$$

Note that the condition (3.2a) applies in the case of function $F_3$ of the Boolean network of Fig. 1(a) for which $(F_1)_{y_3} = \bar{x}_1\bar{x}_2 + 1 = 1$ and $(F_1)_{\bar{y}_3} = \bar{x}_1\bar{x}_2$. Thus $E_{13} = (1)(\bar{x}_1\bar{x}_2) = \bar{x}_1\bar{x}_2$ and $DT_3 = E_{13} = \bar{x}_1\bar{x}_2$, which can be seen to be the last term in the representation $D_3$ given above.

A physical interpretation of $DT_j$ can be given as follows. Consider a primary input vector, $x$, and a corresponding solution vector $v(x) = (y(x), x)$, for which all the primary outputs of $\eta$ are insensitive to the values $y_j$ takes on under this set of inputs. Note in the example that if $x_1 = 0$, $x_2 = 0$ is applied to $\eta$, the primary outputs are $F_1 = 1$ and $F_2 = 0$, regardless of the value of $y_3$. Thus $DT_j$ can be seen to specify a set of values for the vector $x$ such that the value of each of the primary outputs is insensitive to the value of $y_j$, and, by extension, to the representation $F_j$. As we shall show in Section VI, $DT_j$ is simply the union of primary input vectors which *do not* test for either $y_j$ stuck-at-1 or $y_j$ stuck-at-0. Each such primary input vector represents a cube (*not* necessarily just a vertex) in the overall space $B^t$, which is don't care for all the primary outputs since none of them are affected by $F_j$. The representation $DT_j$ of the transitive fan-out don't care set is the union of all such primary input cubes.

We have now given sufficient background to make a precise definition of $DT_j$ meaningful.

*Definition 6 (Transitive Fan-out Don't Care Set)*

We denote, for each primary output $i \in PO \cap TFO_j$, the "*transitive fan-out*" *don't care set* associated with function $j$ by

$$DT_{ij} = \left\{ x \in B^n \mid (v_i)_{v_j}(x) = (v_i)_{\bar{v}_j}(x) \right\} \quad (3.3)$$

where $(v_i)_{v_j}$ and $(v_i)_{\bar{v}_j}$ are the logic variables associated with the corresponding functions $(F_i)_{v_j}$ and $(F_i)_{\bar{v}_j}$, i.e., in the cofactored Boolean networks, and $PO \cap TFO_j$ identifies the subset of primary outputs contained in the transitive fan-out of $F_j$. □

This definition plays a crucial role in the following definition and theorem, and is formed primarily to facilitate theorem proving. In the algorithm of Fig. 4, we actually employ only the special case of (3.2), which is equivalent to (3.3) when the condition (3.2a) is satisfied. Note that because of Definition 4, $DT_{ii} = \varnothing$, $\forall i \in PO$. The members of the $q_j$ "blocking sets" defined in [32] may be observed to correspond to implicants of $DT_{ij}$. Again, a proper subset of the implicants of $DT_{ij}$ is obtained, in linear time, by the procedure of [32].

*Definition 7*

A representation of the don't care set, $D_j$, imposed on node $j$ by the Boolean network is

$$D_j = DI_j + \prod_{i \in PO \cap TFO_j} (DX_i + DT_{ij}) \quad (3.4a)$$

where

$$DI_j = \sum_{k \in (TFI_j - PI)} y_k \bar{F}_k + \bar{y}_k F_k. \quad (3.4b)$$

Corollary 1 below gives us reason to call this the "complete" don't care set. Note $DI_j \subseteq DIV$ derives solely from the transitive fan-in of $F_j$. □

```
Line  Procedure ESPRESSO_MLD (F,PO,DX)
      /*
      Input: Boolean Network η=(F,PO), cf., Definition 1,
      well as the set DX of external don't care sets,
      {DX_1,DX_2,··· DX_p}.
      Output: Minimized Network η'=(F',PO).
      A visitation index, VIS, is employed, such that
      VIS_k=0 (not visited), 1 (visited not changed), -1 (changed)
      If VIS_j=1, ∀j ∈ {1,2,...,m} , η' is prime and irredundant.
      */
      Begin
 1    F'←F                                  Initialize η'.
 2    VIS_k←0, k=1,2,...m                   Initialize visitation index.
 3    For(k=1,2, ··· m)DI_k←(y_k F̄_k+ȳ_k F_k)   Compute components of IV DC set .
 4    J←{j | VIS_j=0 and FO_j⊆PO and
        FO_k=∅, ∀k ∈ FO_j}                  Minimizable set (no reconvergent fan-out).
      While (J≠∅)
        Begin                               While there are functions
                                            non-minimized which fan
 5      j←SELECT1(J)                        out only to Primary outputs,
        VIS_j←1                             select one and mark as visited.
 6      DIA_j←SELECT2(DI ,j)                Select Intermediate don't cares.
 7      If (j∈PO) Then DO_j←∅               Initialize Transitive Fan
          Else                              Out DC set to tautology.
          DO_j←1
          For (i∈FO_j)                      Loop over fanout of F_j.
            Begin
 8          DT_ij←((F_i)_yj≡(F_i)_ȳj)       Equivalence of cofactors.
 9          DO_j←DO_j∩(DT_ij+DX_i)          Update output DC set.
            End For                         for Function F_j.
          End Else
10      DA_j←DIA_j+DO_j                     Acyclic DC sets for F_j.
11      (F_j,VIS_j)←ESPRESSO_IIC(F_j,DA_j)  Minimize F_j w.r.t. DA_j and reset
12      F'_j←F_j                            VIS_j to -1 if changed.
13      (F',F)←SIMPLIFY(F',F)               Update and simplify F' and F.
14      DI_j←y_jF_j+ȳ_jF_j                  Update DI_j.
15      If (FO_j≠∅)F←FLATTEN (j,F)          Flatten F_j into FO_j.
16      If (j∉PO) F←F − F_j                 Delete if not primary output.
17      J←{j |VIS_j=0 and FO_j⊆PO and
          FO_k=∅, ∀k ∈ FO_j}                Update minimizable set.
        End While
18      Return (F',VIS)                     Return minimized Boolean Network.
        End ESPRESSO_MLD                    If (VIS_j=1, ∀j) η' is Prime and Irredundant.
```

Fig. 4. Procedure ESPRESSO_MLD.

It is of interest to observe the possible interrelationships that exist between the transitive fan out and external don't care terms in (3.4). To this end, we present two examples.

*Example 1*

Suppose, for the network of Fig. 1(c), we specify the external don't care sets $DX_1 = x_1$ and $DX_2 = x_2$. Then the don't care representation of (3.4) becomes, for node 2,

$$D_2 = DI_2 + (DX_1 + DT_{12})(DX_2 + DT_{22})$$
$$= DX_1 DX_2$$

in which $DI_2 = \varnothing$, since $F_2''$ has only *primary* inputs, and $DT_{12} = DT_{22} = \varnothing$, since $y_2$ and $y_1$ are *both* primary outputs (note $j \in TFO_j$ by convention). Thus $D_2 = x_1x_2$, which may be used to minimize $F_2''$ further, to

$$F_2''' = x_1 + x_2.$$

Although further minimization of $F_2''$ was made possible by introducing the above external don't care terms, $F_1''$ remains prime, irredundant, and R-minimal, and so $F_1''' = F_1''$. However, it should be noted that the don't care set for $F_1''$ *is* ltered by the minimization of $F_2''$ to $F_2'''$. In fact, *prior* to this minimization, we have

$$D_1'' = y_2(x_1, x_2 + \bar{x}_1\bar{x}_2) + \bar{y}_2(x_1\bar{x}_2 + \bar{x}_1x_2) + x_1$$

whereas after this minimization,

$$D_1''' = y_2(\bar{x}_1\bar{x}_2) + \bar{y}_2(x_1 + x_2) + x_1 \not\equiv D_1'''$$

which proves that the don't care set of function $F_k$ is *not necessarily invariant* with respect to the minimization of $F_j, j \neq k$. □

*Example 2*

Consider some Boolean network (*not* that of Fig. 1) in which $F_i = y_j x_l$ and $F_k = y_j \bar{x}_l$, $FO_j = \{i, k\} \subseteq PO$, and $FO_i = FO_k = \varnothing$. Then the special case assumptions of (3.2) apply to the computation of $DT_{ij}$ and $DT_{kj}$, i.e.,

$$DT_{ij} = \bar{x}_i, \quad DT_{kj} = x_l.$$

Thus from (3.4) we have

$$\begin{aligned} D_j &= DI_j + (DX_i + DT_{ij})(DX_k + DT_{kj}) \\ &= DI_j + DX_iDX_k + DX_iDT_{kj} + DX_kDT_{ij} + DT_j) \\ &= DI_j + DX_iDX_k + DX_iDT_{kj} + DX_kDT_{ij}. \end{aligned}$$

Note that in this case, although

$$DT_j = \prod_{\hat{i} \in PO \cap TFO_j} DT_{\hat{i}j} = DT_{ij}DT_{kj} = \varnothing$$

$DT_{ij}$ and $DT_{ik}$ may, *individually*, still contribute to $D_j$, assuming $DX_iDT_{kj} \neq \varnothing$ or $DX_kDT_{kj} \neq \varnothing$. □

We now give the theorem and corollary that show that (3.4) gives a complete and correct representation of the don't care set.

*Theorem 1*

Let $\eta$, $\eta'$ be two Boolean networks where $\eta$ is identically $\eta'$ except for $F_j$, which has been altered to $F_j'$ in such a way that $TFI_j(\eta') \subseteq TFI_j(\eta)$. Then $\eta = \eta'$ if and only if for all $w \in B^{n+m}$, either

i) $F_j(w) = F_j'(w)$ or
ii) $w \in D_j$.

*Proof (If part):* Suppose $\eta \neq \eta'$. Then by Definition 2 there exists $x \in B^n$ and some output $i \in PO_j$ such that $x \notin DX_i$ and $v_i(x) \neq v_i'(x)$. Define $w \in B^{n+m}$ so that for $k \in TFI_j$, $w_k = v_k(x) = v_k'(x)$, and $w_x = x$ (i.e., the primary input subvector of $w$ is the primary input vector $x$). The other elements $w_l$ of the vector $w$ are chosen arbitrarily. Thus $w \notin DX_i$ (because $x = w_x \notin DX_i$), and $w \notin DI_j$ (cf. discussion of (2.1)). Since $v_i(x) \neq v_i'(x)$, then certainly $v_j(x) \neq v_j'(x)$, since $F_j$ and $F_j'$ are the only functions which differ in $\eta$ and $\eta'$. Thus since $F_j(w) = F_j(v(x)) = v_j(x)$ and similarly for $F_j'(w)$, then $F_j(w) \neq F_j'(w)$. There are now two cases: $v_j(x) = 1$, $v_j'(x) = 0$ and vice versa. For the case $v_j(x) = 1$, we have $(v_i)_{v_j}(x) = v_i(x)$, $(v_i)_{\bar{v}_j}(x) = v_i'(x)$, so $(v_i)_{v_j}(x) \neq (v_i)_{\bar{v}_j}(x)$; hence $x \notin DT_{ij}$. The case $v_j(x) = 0$ yields the same conclusion. Thus $w \notin DT_{ij}$, so $w \notin D_j$, and we have produced $w \in B^{n+m}$ which contradicts i) and ii), which proves the if part.

*(Only If part):* Suppose there exists $w \in B^{n+m}$ such that $F_j(w) \neq F_j'(w)$ and $w \notin D_j$. Then $w \notin DI_j$, which implies that for $x = w_x$, $w_k = v_k(x) = v_k'(x)$, $k \in TFI_j$. This follows from the fact that the two networks $\eta$ and $\eta'$ are the same in $TFI_j$, and hence for the same $x$, $w \notin DI_j$ implies that $w$ satisfies the same defining relations as $v_k$ and $v_k'$, $\forall k \in TFI_j$. Thus $F_j(w) = v_j(x)$ and $F_j'(w) = v_j'(x)$. Also $w \notin D_j$ implies that there exists some $i \in PO_j$ such that $x \notin DX_i$ and $x \notin DT_{ij}$. Since $F_j(w) \neq F_j'(w)$, then $v_j'(x) \neq v_j(x)$. Suppose $v_j(x) = 1$, then $v_i(x) = (v_i)_{v_j}(x)$ and $v_i'(x) = (v_i)_{\bar{v}_j}(x)$ and because $x \notin DT_{ij}$, then $v_i(x) \neq v_i'(x)$. Therefore, since $x \notin DX_i$, then $\eta \neq \eta'$. The case $v_j(x) = 0$ is similar. □

It is important to realize the intent of presenting this theorem, which is to establish a representation for the don't care set $d_j$ of the incompletely specified function associated with node $j$ of $\eta$. Once this is established, we shall have reduced the problem of minimizing $F_j$ in a multilevel environment to a conventional 2-level minimization problem. To this end we offer the following corollary.

*Corollary 1*

$D_j$ is a representation of the don't care set $d_j$ of the incompletely specified function $(f_j, d_j, r_j)$.

*Proof:* $D_j$ represents the set of vertices $v(x) \in B^{m+n}$ such that the IO map $z(x)$ of $\eta$ is insensitive to the specific representation given for $F_j$. But this implies that $D_j$ is a representation of $d_j$. □

Having finally determined a representation $D_j$ of the implied don't care set for node $j$ of Boolean network $\eta$, we are now ready to present two key theorems in the development of our algorithm for multilevel minimization.

*Theorem 2(a)*

Cube $c \in F_j$ in Boolean network $\eta$ is *irredundant* if and only if

$$c \notin (F_j - \{c\}) \cup D_j. \tag{3.5}$$

*Proof. (If part):* Suppose $c \notin (F_j - \{c\}) \cup D_j$. Then, because Corollary 1 has established $D_j$ as a representation of the don't care set $d_j$, there exists $v(x) \in c$ such that $v(x) \in f_j$, the *care on set* of the incompletely specified function associated with node $j$. That is, $v(x)$ is a relatively essential vertex [5], contained in $c$, so $c$ is irredundant.

*(Only If part):* Suppose $c$ is irredundant (Definition 3). Then $c$ contains a relatively essential vertex $v(x) \notin (F_j - \{c\}) \cup D_j$. □

*Theorem 2(b)*

Cube $c$ of function $F_j$ of $\eta$ is *prime* in variable $v_l$ if and only if either a) neither $v_l$ nor $\bar{v}_l$ appears as a literal of $c$, or b)

$$c' \notin F_j \cup D_j \tag{3.6}$$

where $c'$ is the cube obtained by deleting literal $v_l(\bar{v}_l)$ from cube $c$.

*Proof:* Similar to that of Theorem 2(a). □

These two theorems result from the fact that $D_j$ is a "complete" representation of the don't care component

$d_j$ of the incompletely specified function $(f_j, d_j, r_j)$ associated with node $j$ of the Boolean network. This fact is the basis for the proposed approach to multilevel logic minimization, and it will be shown in Section 4 below that a prime and irredundant Boolean network can be obtained by applying a 2-level logic minimizer to each of the representations $F_j$ in sequence. This full sequence is then iterated until, on one complete pass through it, no representation changes from what it has been on the previous pass.

Equations (3.5) and (3.6) show how to make a Boolean network prime and irredundant. But as discussed in Section I, to make $F_j$ R-minimal we also need to apply the following REDUCE operation. As we shall see below, this operation takes on added significance in the multilevel case, and in fact, accounts for an entirely new aspect of logic minimization.

*Definition 8 (REDUCE Operation)*

Cube $c' \subset c \in F_j$ is the *reduction* of $c$ if a) $\eta = \eta'$, where $\eta'$ is defined by replacing $F_j$ with $F_j' = c' \cup (F_j - c)$, and b) for all $c'' \subset c'$, $\eta \neq \eta''$, where $F_j'' = c'' \cup (F_j - c)$. □

*Proposition 1*

The reduction, $c'$, of cube $c$ is unique.

*Proof:* Note $c'$ contains all relatively essential minterms of the representation $F_j$ of the single-output function $f_j$. If $c'$ were not unique then there would be another reduction $c'' \neq c$ which would also contain these minterms for which $\eta = \eta''$. Hence $c'$ and $c''$ can both be replaced by cube $c' \cap c''$, which also contains all these minterms. Since $c' \neq c''$, then either $c' \cap c'' \subset c'$ or $c' \cap c'' \subset c''$, contradicting the hypothesis that both $c'$ and $c''$ were reductions of $c$. □

Since the reduction of cube $c$ is unique, the overall REDUCE operation for $c$ is composed of a sequence of "atomic" REDUCE operations, carried out in any order. Each of these atomic operations determines whether equivalence at the Boolean network level is maintained if $c$ is replaced by $c^*$, where $c^*$ is obtained from $c$ by adding literal $v_k$, and where $v_k$ is not originally present in $c$. If the answer to this question is positive, then $c$ can be replaced by $c^*$. The process repeats until we have attempted the addition of the positive and negative phase of every literal not originally present in $c$. Note that if both $c_k$ and $\bar{c}_k$ can be individually added while maintaining equivalence, then $c$ is redundant and can be deleted from $F_j$.

It is tempting to conjecture that as in the case for primality and irredundancy, the don't care set $D_j$ is sufficient to determine the reduction of cube $c \in F_j$. Unfortunately this is not quite the case, although the following proposition can be proved about a single atomic REDUCE operation.

*Theorem 2(c) (REDUCE Don't Care Set)*

The minimal and sufficient don't care set for the atomic REDUCE operation of adding literal $v_k$ to cube $c \in F_j$ is

$$DR_{j,k} = DI_j + DI_k + \prod_{i \in PO \cap TFO_j} (DX_i \cup DT_{ij}) \quad (3.7)$$

where $DI_j$ and $DI_k$ are defined by (3.4b).

*Proof:* The proof of Theorem 1 can be applied, *mutatis mutandis*, noting that adding literal $v_k$ to cube $c$ potentially augments the transitive fan-in of $F_j$. □

Note that unlike $D_j$, $DR_{j,k}$ includes intermediate variable don't cares from the transitive fan-in of both $F_j$ and $F_k$. Even though $DR_{jk}$ is sufficient for the single atomic REDUCE operation associated with literal $v_k$, a larger don't care may be required by the next atomic operation. This is because the successful addition of literal $v_k$ adds an edge to the graph $(N, E)$ of the Boolean network $\eta$, and, therefore, may alter $TFI_j$. Thus if the entire REDUCE operation is to be performed with a single don't care set, and if applicability to arbitrary Boolean networks is desired, then the entirety of the overall don't care set *DIV* (cf. Definition 5) must be employed. This conclusion is mitigated, however, by the following remarks.

*Remarks*

a) Note that $c'$ is a minimal (smallest) cube containing all the relatively essential vertices of $c$ [5]. Hence $c'$ has more literals than $c$, hence $c'$ is reexpressed in a larger support than $c$ had. The remarkable fact is that $c'$ can, in principle, now depend on any variable in the transitive fan-out of the transitive fan-in of $F_j$. As shown for $F_1$ in the Example of Fig. 1, this can lead to significant simplifications in the $F_j$. The REDUCE operation appears to be one of the most significant parts of multilevel minimization.

b) Although the overall intermediate variable don't care set *DIV* is necessary to obtain the true reduction of $c$, in practice we use an approximation $DA_j$, where

$$DA_j = DIA_j + \prod_{i \in PO \cap TFO_j} (DX_i \cup DT_{ij})) \quad (3.8a)$$

$$DIA_j = \left( \sum_{IV - TFO_j} v_k \overline{F_k} + \overline{v_k} F_k \right) \quad (3.8b)$$

is obtained from *DIV* by deleting the intermediate variable don't care contributions of the transitive fan-out of $F_j$. If this deletion were not done, *cyclic dependencies* might (will) occur, i.e., some cube $c$ in $F_j$ can be reduced until it contains literal $v_j$ itself, and, on a subsequent EXPAND step, $c$ might grow to $c = v^j$, leading ultimately to the correct, but trivial, conclusion that $v_j = F_j$. We shall call the reduction of cube $c$ with respect to $DA_j$ the *acyclic reduction* of $c$.

c) Note that the operations of primality and cube redundancy testing (cf. Theorems 2(a) and 2(b)) do not alter the transitive fan-in of $F_j$.

d) Note that (3.8) implies that $D_j \subseteq DA_j$, so that $DA_j$ is sufficient for establishing the primality and irredundancy of cube $c$ as well as for finding its acyclic reduction. □

As pointed out in [5, sec. 4.7], reduction is an important mechanism for minimizing the representation $F_j = \{c_i\}$. In fact, by reducing some prime cube $c_k \in F_j$ to its reduction $c_k'$ (Definition 8), it is possible that after reex-

panding $c_k'$ to a different prime $c_k^+ \neq c_k$, a second, formerly irredundant prime cube, $c_l$, may now become redundant, $l \neq k$. This remark gives us, at last, sufficient background to define R-minimality.

*Definition 9*

A *prime and irredundant* Boolean network $\eta$ is *R-minimal* if there exists no cube $c_k \in F_j$ of $\eta$ whose *acyclic* reduction $c_k'$ (Definition 8) can be reexpanded (i.e., raised back to primality) into cube $c_k^+$ such that for $k \neq l$,

$$\{c_k, c_l\} \subseteq (F_j - \{c_k, c_l\}) \cup c_k^+ \cup D_j. \quad (3.9)$$

That is, the introduction of the reduced and reexpanded cube causes both the originally prime and irredundant cubes $c_k$ and $c_l$ to become *redundant*. □

Note that the example Boolean network of Fig. 1(c) is R-minimal, because none of the cubes of either $F_1$ or $F_2$ can be reduced.

It is expensive in practice to absolutely *guarantee* R-minimality, but it is certainly possible in principle. ESPRESSO-IIC executes a routine for reduction and reexpansion called LAST_GASP which guarantees only an approximate form of R-minimality. However, it has been observed in all but a very few cases to date that the actual results of LAST_GASP were, in fact, R-minimal. The REDUCE operation and its variants (this was called "RESHAPE" in MINI [20]) enable logic minimizers to "climb out" of the local minima usually associated with the current prime and irredundant representation. This is, in many cases, the key to the high-quality results obtainable by heuristic minimizers.

## IV. The ESPRESSO_MLD Procedure for Multilevel Logic Minimization

Theorems 2 establish don't care methods for applying the EXPAND, IRREDUNDANT_COVER, and REDUCE operations to the cubes of the function representations $F_j$ of $\eta$. We can now present an algorithm which calls the 2-level logic minimizer ESPRESSO_IIC to carry out these operations on each $F_j$ in turn. On exit from ESPRESSO-IIC, $F_j$ is prime and irredundant. However, ESPRESSO_IIC has the property that $F_j$ is left unchanged if the first pass through the REDUCE, EXPAND, and IRREDUNDANT_COVER sequence fails to decrease a given cost function measuring the number of terms and literals of the result. Any other valid 2-level minimizer which has this property will also produce a prime and irredundant Boolean network. However, as discussed above, ESPRESSO-IIC guarantees a weak form of R-minimality as well. The algorithm uses the representation of the don't care set $DA_j$, defined by (3.8) for function $F_j$ of Boolean network $\eta$, to render all the cubes of $F_j$ *prime* and *irredundant*, $\forall j \in IV$, according to Definition 3 and Theorems 2(a) and 2(b).

This algorithm is presented in Fig. 4 as Procedure ESPRESSO_MLD. ESPRESSO_MLD calls ESPRESSO_IIC to minimize the functions $F_j$ in a certain order. In most cases, it first minimizes the primary output functions; since conditions (3.2a) are usually satisfied for primary outputs, they are in the first $J$ constructed in line 4. Clearly for any $j \in PO$, $DT_{ji} = \varnothing$, and hence $DT_j = \varnothing$. Then the algorithm selects for the next function (cf. lines 4 and 17 in Fig. 4) some unminimized function which has only nonreconvergent fan-out to primary outputs, i.e., the next function $F_j$ satisfies $FO_j \subseteq PO$ and $FO_i = \varnothing$, $\forall i \in FO_j$. After minimizing this function, it is stored away for future reference in the minimized function set $F_j'$ (line 12), and then "flattened," i.e., substituted, into its fan-out (line 13). If it is not a primary output it is then deleted from $\eta$. Because of this deletion, and because $\eta$ is assumed to be combinational, such a next function always exists (but is not unique). Another such function is selected next. This is repeated until all functions have been minimized.

Note that ESPRESSO_MLD employs the device of carrying two separate versions, $\eta$ and $\eta'$, of the minimized Boolean network. Here $\eta'$ is the version of the original network $\eta$, in which each function is replaced by its minimized version, i.e., the version which is prime, irredundant, and, with high probability, R-minimal. This is the version returned (line 18) by ESPRESSO_MLD. The second version starts out the same as the original Boolean network, but is modified on each pass through the while loop (lines 5–17) by flattening the most recently minimized function into its fan-out, and then deleting it unless it is a primary output function. As discussed at the end of this section, this device permits us to use the construction of (3.2) in computing the transitive fan-out don't care sets $DT_{ij}$, $i \in PO$.

Note further that a subprocedure, SIMPLIFY, is called (line 13) after replacing $F_j$ and $F_j'$ by the minimized version of $F_j$ in the respective Boolean networks $\eta$ and $\eta'$. SIMPLIFY checks for either of the conditions a) $F_j \cup D_j \equiv 1$, or b) $F_j \subseteq D_j$. In the former case, ESPRESSO_IIC will return a representation consisting of a single cube with no literals, and in the latter case, one consisting of an empty set of cubes. In case a) the care off set of the incompletely specified function $(f_j, d_j, r_j)$ is empty, i.e., $r_j = \varnothing$. It follows that $\eta = \eta_{v_j}$, so SIMPLIFY substitutes $v_j = 1$ into the functions in the fan-out of $F_j$. Case b), for which we have $f_j \equiv \varnothing$, is similar, except $v_j = 0$ is substituted. In either case the simplification is propagated recursively toward the primary outputs and primary inputs. In propagating toward the primary outputs, functions in the fan-out of $F_j$ are tested in turn for simplification by SIMPLIFY. In propagating toward the primary inputs, we note that after simplifying, $F_j$ no longer depends on any of its inputs. Thus edges in $\eta$ associated with these inputs may be deleted from the graph associated with $\eta$. SIMPLIFY thus checks to see if $F_j$ was the last fan-out of any function $F_i$ such that $j \in FO_i$. If so, $F_i$ is deleted from $\eta$ and $\eta'$. Note that if as a result of such simplification, some function $F_k$ has no remaining fan-out, i.e., $FO_k = \varnothing$, then it may be seen from Definition 6 that $DT_{ik} \equiv 1$, $\forall i \in PO$. Consequently, in this case, when $j$ is later equal to $k$, the for loop (lines 8 and 9), which computes $DO_k =$

$\Pi_{i \in PO \cap TFO_k} (DX_i + DT_{ik})$, will initialize $DO_k$ to 1, and $DO_k$ will remain at that value unless $DX_i \neq 0$ for some $i$. Such functions thus will fall into category a) above. This process continues recursively until the two networks stabilize.

Even though $F_j$ is prime and irredundant on exit from ESPRESSO_IIC, a function $F_k$ previously minimized by ESPRESSO_IIC may no longer be prime or irredundant. This is because the don't care set $d_k$ is not invariant with respect to the minimization of $F_j$. It is quite easy to construct examples which demonstrate this fact. Thus in order to verify primality and irredundancy of the returned network $\eta'$, procedure ESPRESSO_MLD uses a visitation index $VIS_j$ to mark which functions were actually alerted by the call to ESPRESSO-IIC. As stated above, $F_j$ is left unchanged unless ESPRESSO-IIC can obtain a finite *decrease* in the cost of $F_j$. If on exit (line 18), $VIS_j = 1, \forall j \in \{1, 2, \cdots, m\}$, then it is true that no cube of any function representation was altered by the calls to ESPRESSO_IIC, which proves, as shown below, that the given Boolean network is prime, irredundant, and, with high probability, R-minimal. Conversely, if on exit $VIS_j = -1$ for any $j$, then, although we are sure that $F_j'$ is prime and irredundant, we can no longer be sure that another function $F_k'$ is still prime and irredundant, where the representation $F_k'$ was returned by a previous call to ESPRESSO_IIC. Thus the whole procedure ESPRESSO_MLD should be called again. Since we are guaranteed that the overall cost function has a finite decrease if any function is altered, we know that the sequence of calls to ESPRESSO_MLD must ultimately converge, and, on the last call, return an unchanged Boolean network. This latter network is prime and irredundant and very likely R-minimal.

It is of interest to discuss how the structure of the flattened Boolean network $\eta$ is exploited in computing the transitive fan-out portion of the "acyclic" don't care set $DA_j$ prior to each call to ESPRESSO-IIC. By construction, $\eta$ is just $\eta'$ with all intermediate variables in the set $TFO_j - PO$ flattened (line 15) and deleted (line 16), and those in the set $TFO_j \cap PO$ flattened but not deleted. The key observation is that flattening an intermediate variable does not alter the IO map of a Boolean network. Hence $\eta$ and $\eta'$ have identical IO maps, i.e., $z(x) \equiv z'(x)$. Because of the flattening of intermediate variables in the transitive fan-out of $F_j$, we are able to use the construction of (3.2) in computing $DT_{ij}$ for each primary output $i \in PO$ in Boolean network $\eta$. Note that by construction of $\eta$, $FO_j \subseteq PO$ and $FO_i = \varnothing$, $\forall i \in FO_j$. Thus the computation $DT_{ij}$ is restricted to the case where $F_i$ has only a direct dependence on $v_j$, $\forall i \in PO \cap FO_j$. That is, for functions that fan-out only to primary outputs which have no fan out, $DT_{ij}$ is equivalent to $E_{ij}$, where (cf. lines 8 and 9)

$$E_{ij} = (F_i)_{y_j} (F_i)_{\bar{y}_j} + (\bar{F}_i)_{y_j} (\bar{F}_i)_{\bar{y}_j}. \tag{4.1}$$

Finally, by the following lemma we are able to show that this don't care set is identical to the transitive fan out don't care set $DT_{ij}'$ of the unflattened but minimized network $\eta'$ returned by ESPRESSO_MLD.

*Lemma 1*

For each primary output $i \in PO$, let $DT_{ij}$ be the transitive fan-out don't care set associated with the minimized and flattened Boolean network $\eta$ computed by ESPRESSO_MLD, and let $DT_{ij}'$ be that associated with the minimized but not flattened network $\eta'$ returned by ESPRESSO_MLD. Then

$$DT_{ij} \equiv DT_{ij}', \quad \forall i \in PO. \tag{4.2}$$

*Proof:* By construction, $\eta'$ is the minimized but not flattened Boolean network with primary inputs $x$ and IO map $z'(x)$. Similarly, $\eta_{v_j}'$ is the cofactor of this network with respect to the variable, $v_j$, of the function to be minimized, which has the same primary inputs, $x$, but has IO map $z_{v_j}'(x)$. Note that $z_{v_j}'(x)$ may be regarded as a Boolean network with one "extra" primary input, namely $v_j$, which has been set permanently to 1. By construction $\eta_{v_j}$ is just a flattened version of $\eta_{v_j}'$ which has the same primary inputs, $x$, but has IO map $z_{v_j}(x)$. But since $\eta_{v_j}$ can be obtained from $\eta_{v_j}'$ by flattening, it follows that these two Boolean networks have identical IO maps, i.e.,

$$z_{v_j}(x) \equiv z_{v_j}'(x) \tag{4.3a}$$

and, similarly,

$$z_{\bar{v}_j}(x) = z_{\bar{v}_j}'(x). \tag{4.3b}$$

Since $DT_{ij}$ and $DT_{ij}'$ are specified by Definition 6 in terms of the IO maps $z_{v_j}(x)$, $z_{\bar{v}_j}(x)$, $z_{\bar{v}_j}'(x)$, and $z_{v_j}'(x)$, it follows that $DT_{ij} \equiv DT_{ij}'$, so (4.2) is proved. □

Now reconsider procedure ESPRESSO_MLD, which calls ESPRESSO_IIC only for functions $F_j$ which satisfy $FO_j \subseteq PO$ and $FO_k \equiv \varnothing$, $\forall i \in FO_j$. Thus the transitive fan-out don't care set of function $F_j$ of Boolean network $\eta$ can be computed according to the construction of (3.2) (line 8) for each primary output in $TFO_j$. By Lemma 1, this is identical to $DT_{ij}'$ in the Boolean network $\eta'$ which we are minimizing. Thus $DT_{ij}$ can be added to the external don't care set for each of the aforementioned primary outputs and intersected together (line 9 of the inner for loop) to form the rightmost don't care term in (3.4). This interim result is stored in the variable $DO_j$ and is added to the appropriate intermediate variable don't care sets (cf. (3.4) and the remark following (3.8)), to form $DA_j$ (line 10). This construction of the relevant don't care sets permits us to state the following key theorem, which, in essence, proves the correctness of ESPRESSO_MLD.

*Theorem 3*

Suppose that when ESPRESSO_MLD terminates, all $m$ functions $F_j$ have had 2-level minimization applied to them, *without* any changes to any of the $F_j$. Then the returned Boolean network, $\eta'$, is prime and irredundant.

*Proof:* The procedure uses the acyclic superset, $DA_j$ of $D_j$ for minimizing each of the $F_j$, where $D_j$ is a representation of the don't care set $d_j$ of the incompletely specified function $(f_j, d_j, r_j)$. Thus after each pass, by Theo-

rem 2, Lemma 1, and Corollary 1, the current $F_j$ is prime and irredundant. Now if all $m$ functions have failed to change on one complete pass through ESPRESSO_MLD, each has been shown to prime and irredundant *given* the final state of $\eta'$. Thus by Definition 3, $\eta'$ is prime and irredundant. □

## V. Experimental Results

Tables I and II illustrate the results of running ESPRESSO_MLD on some multilevel examples generated using Weak Division [1]. These computational results were obtained using an approximate implementation of ESPRESSO_MLD. The approximation made was the following. In practice $DIA_j$ can be quite large, so only a subset of the complete $DIA_j$ was used in obtaining the results of Table I. The subset used was the don't care terms associated with the set of all intermediate variables which are in the transitive fan-out of the transitive fan-in of $F_j$, but not in the transitive fan-out of $F_j$. Thus the REDUCE operation of ESPRESSO_IIC is limited to the introduction of either primary input variables or intermediate variables which are in this set. This approximation is returned (line 5) by the subprocedure call to SELECT2, and is also an "acyclic" approximation for the REDUCE operation in ESPRESSO_IIC. This acyclicity constraint prevents, as discussed in the above Remark, trivial reductions of $F_j$ by the REDUCE-EXPAND-IRREDUNDANT_COVER sequence. However, note that this requires EXPAND and IRREDUNDANT_COVER to operate with a *fixed* intermediate variable portion of the don't care set, despite the fact that the transitive fan-in of $F_j$ is being altered by REDUCE. Note that this means the approximately implemented version of ESPRESSO_MLD does not guarantee primality, although ancillary experiments have indicated that the computationally minimized networks very probably are prime. Further, the experimental results appear to have high minimization quality, an observation based on attempts at further minimization, which used alternative computational techniques.

In Table I "initial literals" refers to the number of literals in the original multilevel network. The next two columns refer to the number of literals saved when using just the intermediate don't cares and when using both the intermediate and output don't care sets. For example when plab was minimized using just the approximation by SELECT2 of $DIA_j$, the resulting network had 9 fewer literals, but when both $DIA_j$ and $DT_j$ were used (line 10) the resulting network had 20 fewer literals than the initial network. This illustrates the significance of the transitive fanout don't care set, since in all the examples of Table I we assumed $DX_i \equiv \varnothing$, $\forall i \in PO$. No table entry indicates that no function $F_j$ of the given network could be reduced in cost by the implemented minimization procedure.

Runtimes in Table I are in CPU seconds on a Pyramid 90X, which is about twice as fast as a VAX 11/780. The CPU time requirements ranged from minutes on the medium size jobs to hours on the larger ones. Use of the "output" don't care set $DO_j$ (cf. lines 9 and 10 of ESPRESSO_MLD) typically incurs a factor of 2–4 increase in CPU time.

TABLE I
ESPRESSO_MLD Multilevel Minimization Results

| Name | Initial Literals | Literals Saved | | Runtime | |
|---|---|---|---|---|---|
| | | $DIM_j$ | $DIM_j \cup DO_j$ | $DIM_j$ | $DIM_j \cup DO_j$ |
| mark | 8 | 1 | | 1 | |
| f0 | 16 | 1 | | 1 | |
| f1 | 14 | 1 | | 1 | |
| f2 | 28 | 4 | | 9 | |
| f3 | 73 | 1 | 2 | 28 | 57 |
| f4 | 75 | 2 | 2 | 87 | 130 |
| f5 | 75 | 4 | 5 | 44 | 118 |
| gerf | 17 | 1 | 1 | 1 | 3 |
| dec1 | 52 | 1 | 1 | 21 | 34 |
| fadd2 | 29 | 3 | 3 | 2 | 3 |
| clpl | 19 | 2 | | 5 | |
| insdex | 79 | 5 | 5 | 40 | 57 |
| plac | 191 | 7 | 28 | 1746 | 3733 |
| 8fun | 83 | 3 | 3 | 70 | 152 |
| exam2 | 73 | 3 | 3 | 45 | 73 |
| rd53 | 62 | 14 | 24 | 12 | 26 |
| adder | 48 | 4 | 4 | 4 | 87 |
| dec2 | 149 | 3 | 4 | 2041 | 7852 |
| plab | 119 | 9 | 20 | 434 | 1850 |
| z4 | 58 | 14 | 20 | 8 | 77 |

Table II contains the results of experiments run on the subset of the Table I examples for which the "SOCRATES" expert system was used to further optimize the output of the implemented version of ESPRESSO_MLD [1]. The purpose of this set of experiments was to see if the technology-independent gains made by ESPRESSO_MLD were of value when its output was postprocessed by a technology-specific optimized mapping into a standard cell library. We used the SOCRATES expert system for this purpose [1]. In the headers of Table I, *AA* corresponds to running Weak Division in area-specific mode and then running SOCRATES in the area-specific mode [1]. Similarly, *DD* corresponds to running Weak Division in delay-specific mode and then running SOCRATES in delay-specific mode. The last 4 columns show the effect of inserting ESPRESSO_MLD into the synthesis loop. *AA** corresponds to running ESPRESSO_MLD on the output of Weak Division running in area-specific mode and then running SOCRATES in the area-specific mode, and *DD** corresponds to running ESPRESSO_MLD after running Weak Division in delay-specific mode and then running SOCRATES in delay-specific mode.

It can be observed that when technology-independent multilevel minimization was used as a preprocessor to SOCRATES, the *AA** area numbers were better than the *AA* results in 8 of the 12 cases. In 3 of the other cases, better area delay tradeoffs were exhibited. In the 6*DD** examples the delay was reduced (relative to *DD*) in all but one case (exam). These numbers indicate that technology-independent multilevel minimization is often a valuable step to take in the synthesis and optimization process, even when the final result is postprocessed by a technology-specific, optimizing expert system.

TABLE II
WEAK_DIVISION—ESPRESSO_MLD RESULTS

| | *AA* Area | *AA* Delay | *DD* Area | *DD* Delay | *AA** Area | *AA** Delay | *DD** Area | *DD** Delay |
|---|---|---|---|---|---|---|---|---|
| fadd | 39 | 12 | 47 | 9 | 32 | 9 | | |
| adde | 56 | 12 | 57 | 18 | 59 | 14 | | |
| dec1 | 73 | 12 | 84 | 6 | 69 | 10 | | |
| z4 | 76 | 13 | 117 | 16 | 58 | 16 | 61 | 14 |
| rd53 | 89 | 22 | 90 | 11 | 82 | 15 | | |
| f5 | 97 | 14 | 124 | 11 | 95 | 14 | 109 | 8 |
| exam | 98 | 13 | 127 | 9 | 94 | 11 | 116 | 10 |
| f4 | 103 | 13 | 118 | 10 | 103 | 9 | 124 | 8 |
| 8fun | 107 | 14 | 141 | 13 | 110 | 13 | 128 | 10 |
| plab | 158 | 18 | 192 | 14 | 176 | 15 | | |
| dec2 | 203 | 22 | 243 | 16 | 201 | 20 | | |
| plac | 249 | 22 | 337 | 16 | 256 | 26 | 336 | 15 |

In addition to assuring primality and irredundancy the don't care set may be used to alter the adjacency relations of the Boolean network, as shown in the example of Fig. 1(b). It is of interest to observe that when ESPRESSO_MLD is run on this example, the starting representation (bottom left) is prime and irredundant. Thus the first EXPAND and IRREDUNDANT_COVER operations in ESPRESSO_IIC will have no effect. We have observed that this also occurred in each of the examples of Table I, each of which was output from the "weak division" process of algebraic decomposition [8]. We conjecture that this will always be the case for multilevel examples produced by Weak Division. However, in this example, after the initial REDUCE operation is performed, the prime, irredundant, and, with high probability, R-minimal result will be obtained in the second or third EXPAND step. This again occurred on all the examples of Table I for which minimization was successful. We observe, in fact, that REDUCE is performing a major part of the role of the minimization process referred to as Boolean substitution in [8].

## VI. Test Generation as Logic Minimization (or Vice Versa)

We now state some basic results on testability, with the intent of

1) establishing, in greater detail, the intimate relationship between logic minimization and test generation;

2) demonstrating that after multilevel logic minimization, a prime and irredundant Boolean network is obtained for which there is no need whatsoever for either test generation or testability analysis.

### *A. Test Generation as a By-Product of Logic Minimization*

Our derivation of the complete don't care set (cf. Theorem 1) reduces the testing question for function $F_j$ of a multilevel Boolean network $\eta$ to, in effect, the 2-level case. In fact, we shall show that any stuck fault test $x^*$ is simply the primary input part of a solution vector $v^*(x^*) \in \overline{D}_j \subseteq B^{m+n}$, where

$$\overline{D}_j = \overline{DI}_j \cap \left( \sum_{i \in PO \cap TFO_j} \overline{DT}_{ij} \overline{DX}_i \right). \tag{6.1}$$

First note that because $v^*(x^*) \in \overline{DI}_j$, the *local* inputs to $F_j$ will have the same (cf. (2.1)) values they will have under test, i.e., when $x^*$ is applied to the *primary* inputs. Further, because $v^* \in \overline{D}_j$, there will exist at least one primary output node, $i \in PO \cap TFO_j$, such that $v^* \in \overline{DX}_i \cap \overline{DT}_{ij}$. Because $v^* \in \overline{DX}_i$, we are assured that $x^*$ represents an external care condition for primary output $i$. Finally, note that because $v^* \in \overline{DT}_{ij}$, we may conclude that not only does the test produce a difference $v_j(x) \neq v_j'(x)$ between the good ($\eta$) and fault ($\eta'$) machines, but that this change is propagated to output $i$ as well (i.e., $v_i(x^*) \neq v_i'(x^*)$). It may be observed that the condition $v^*(x^*) \in \overline{DI}_j$ plays the role of the "implication" phase of the D-algorithm [25], and $v^*(x^*) \in \overline{DT}_{ij}$ plays the role of the "propagation" phase.

We begin our treatment of the interrelationship between testing and logic minimization by showing that the transitive fan-out don't care set of Definition 6 can be directly related to the set of output stuck fault tests. This relationship is made precise by the following theorem.

*Theorem 4*

Assuming that there are no external don't care conditions, don't care set

$$DT_j = \coprod_{i \in PO \cap TFO_j} DT_{ij}$$

is the set of primary input vectors which do *not* test the Boolean network $\eta$ for either of the output stuck faults $y_j$ stuck-at-1 or $y_j$ stuck-at-0.

*Proof:* It was shown in [19] that a test $x^*$ exists for the output fault $y_j$ stuck-at-1(0) if and only if $\eta \not\equiv \eta_{v_j(\bar{v}_j)}$. Let $T1_j(T0_j)$ be the set of all such tests. Hence, by Definition 2,

$$T1_j(T0_j) = \left\{ x \mid (v_i)_{v_j(\bar{v}_j)}(x) \neq v_i(x), \text{ some } i \in PO \cap TFO_j \right\}.$$

Since $v_j(x) = 1(0)$ implies that $v_i(x) = (v_i)_{v_j}((v_i)_{\bar{v}_j}(x))$, $\forall i \in PO$, it then follows that for each such test, $x$, there exists some $i \in PO \cap TFO_j$ which has the property $(v_i)_{v_j}(x) \neq (v_i)_{\bar{v}_j}(x)$. It follows from the Definition 6 that $DT_j = \overline{(T1_j + T0_j)}$. □

This theorem shows that if *no* test exists for either $y_j$ stuck-at-1 or $y_j$ stuck-at-0, then $DT_j$ is tautologous, i.e., $DT_j \equiv 1$. It is well known that in this case $F_j$ can be deleted from the Boolean network (such deletions actually occur frequently in practical multilevel logic minimization). On the other hand, $DT_j = \varnothing$ would imply that *all primary input* vectors would be tests for either $y_j$ stuck-at-1 or $y_j$ stuck-at-0. Since this is unlikely to occur in practice, we conclude that the typical case is $DT_j \neq \varnothing$, hence $DT_j$ can be expected to be helpful in minimizing $F_j$. However, note, as shown by Example 2 of Section III, that $DT_j$ can be empty, meaning that all primary input vectors test for $y_j$ stuck-at-1, yet some of the $DT_{ij}$ *can still* contribute to $D_j$, due to the interrelationship with the $DX_i$.

One interpretation of the typical case $DT_j \not\equiv 1$, $DT_j \not\equiv \varnothing$, is that $F_j$ may be *partially redundant*, in the sense that some of its otherwise "care" on set minterms may be covered by $DT_j$. This type of partial redundancy must be exploited in the minimization of $F_j$ if it is to be made prime and/or irredundant.

Having established how the computation of the don't care set provides a direct and constructive link between logic minimization and test generation, we now turn our attention to the *testability* of a prime and irredundant Boolean network. The usual measure of testability for a Boolean network $\eta$ is how many of its individual input or output stuck faults are testable. One of the most significant aspects of the relation between logic minimization and testing is that making $\eta$ prime and irredundant implies much more than merely making it 100-percent testable for the usual input and output single stuck faults. This distinction is further emphasized when the nodes of the Boolean network are represented by complex gates (e.g., CMOS pluricells, domino logic, etc.) rather than simple primitives like NAND's and NOR's. To show this, we need to define a stuck fault model which is more fine grained than conventional input or output stuck faults.

*Definition 10*

An *internal* stuck fault is a fault in which literal $v_k$ (or $\bar{v}_k$) of cube $c$ of representation $F_j$ of Boolean network $\eta$ is stuck at either its existing value $v_k$ (or $\bar{v}_k$) or its opposite value $\bar{v}_k$ (or $v_k$). □

These faults are called internal, since they correspond directly to transistor level faults in which the transistor representing the specified literal in the implemented logic is stuck on or off. Their definition enables us to prove our main testability result.

*Theorem 5*

A Boolean network is prime and irredundant if and only if it is 100-percent testable for internal stuck faults.

*Proof (If part):* Suppose Boolean network $\eta$ is prime and irredundant, and suppose cube $c$ of function $F_j$, is being raised to prime. Suppose $c$ contains literal $v_k(\bar{v}_k)$ and a logic minimizer is checking to see if $c^* \subseteq F_j \cup D_j$, where $c^*$ is just $c$ with literal $v_k(\bar{v}_k)$ replaced by $\bar{v}_k(v_k)$. A negative answer implies that there exists a vertex $x^* \in B^n$ such that $v^*(x^*) \in c^*R_j$, where $R_j = \overline{(F_j \cup D_j)}$. In fact, the minimizer *must* discover such a vertex $v^*(x^*) \in B^{m+n}$ *before* it can declare variable $k$ of cube $c$ of function $F_j$ to be prime. Given $v^*(x^*)$, we simply take the primary input part $x^* = v^*(x^*)_{PI}$ to obtain a test for an input fault. The fault tested is the internal stuck fault "variable $v_k$ of cube $c$ stuck at $b$," where $b = 1$ if $v_k \in c$, and $b = 0$ if $\bar{v}_k \in c$. Thus when $x^*$ is applied to $\eta$, the value $v_k^*(x^*) = \bar{b}$ will appear as an input to $F_j$ in the good machine, for which $F_j$ is off, i.e., $v_j = 0$, since $v^* \in \bar{F}_j \bar{D}_j$. But with $v_k$ stuck at $b$, cube $c$ in $F_j$ will be turned on for input $v^*(x)$, so that $v_j = 1$ in the "fault machine" (a Boolean network which we call $\eta'$). Because $v^*$ is a minterm of the don't care complement $\bar{D}_j$, (cf. (6.1)), we have $v^* \notin DT_{kj}$, for some $i \in PO \cap TFO_j$, which by Theorem 1 implies that primary output $v_i(x^*)$ will have a different value in the good and fault machines, i.e., $v_i(x^*) \neq v_i'(x^*)$, so that $\eta \neq \eta'$.

This $x$ is a test for input variable $v_k$ of cube $c$ of $F_j$ "stuck at $b$" faults, where $v_k$ appears as $b$ in $c$. To show that tests are implied for "stuck at $\bar{b}$" internal faults, consider the action of ESPRESSO_IIC (in line 11 of ESPRESSO_MLD) in testing if cube $c$ of function $F_j$ is *irredundant*. It is clear that if cube $c$ *has* been declared irredundant, then $c \subseteq (F_j - \{c\}) \cup D_j$. If this is the case, then there exists a test vector $x^*$ and a corresponding relatively essential vertex $v^*(x^*) \in c$ such that $v^*(x^*) \in \overline{(F_j - \{c\}) \cup D_j}$. Such a $v^*(x^*)$ *must* be found by ESPRESSO_IIC before it can declare cube $c$ irredundant. The corresponding $x^*$ is a test for $v_k$ stuck at $\bar{b}$, where $v_k$ is *any* literal which appears as $b$ in cube $c$. The test is for $v_k$ stuck at $\bar{b}$, because for $c_k = b$, $v^*(x^*)$ is such that $v_k^*(x^*) = b$ so that cube $c$ is on for the good machine and thus $v_j^*(x^*) = 1$. However, if $v_k$ is stuck at $\bar{b}$, then $c$ is off and $v^*(x^*)$ is such that all the other cubes of $F_j$ are off, so $v_j = 0$ for the fault machine. Note here that the fault assertion "$v_k$ stuck-at-$\bar{b}$" has *limited scope*. That is, the assertion applies only to cube $c$, and *not* to other cubes of $F_j$. Hence all these other cubes, which are *off* in the good machine, remain *off* in the fault machine. The argument then concludes as it did for the stuck-at-$b$ case, which proves the if part. Note that $x^*$ tests any or all of the variables in cube $c$ stuck at their "opposite" value, which constitutes a *multiple* rather than single stuck fault. Of course, any of the single stuck faults are also tested.

*(Only If part):* Assume $\eta$ is 100-percent testable for internal stuck faults. That is, for each cube $c$ and literal $v_k \in c$ (we assume without loss of generality that $v_k$ appears positively in $c$), there exists a test $x$ for variable $v_k$ stuck at 1. Since $x$ is a test, cube $c$ and literal $v_k$ will be "off" in the good machine, i.e., $v_k(x) = 0$, and $v(x) \notin c$, which implies $v_j(x) = 0$ in the good machine. But

with $v_k$ stuck at 1, cube c will be "on" in the fault machine. Thus $v_j'(x) = 1$, and, because $x$ is a test, $\eta \neq \eta'$. But $v_j(x) \neq v_j'(x)$ implies $c^* \not\subseteq F_j \cup D_j$, where $c^*$ is just $c$ with literal $v_k$ replaced by $\bar{v}_k$. This implies cube $c$ is prime in variable $v_k$. The proof that cube $c$ is irredundant follows similarly from the assumed existence of a test for variable $v_k$ stuck at 0 in cube $c$. This proves the only if part. □

It remains to demonstrate that a prime and irredundant network is testable for all the conventional input and output stuck faults. To see that input stuck faults are all testable, note that in almost every case an internal stuck fault is also an input stuck fault. The essence of the argument is that since we have internal stuck faults for all variables of all cubes if the Boolean network is prime and irredundant, and since all the inputs of $F_j$ are contained in one or more of these cubes, the internal stuck fault tests cover all input stuck faults. This is made precise by the following result.

*Corollary 3*

The internal stuck fault tests of a prime and irredundant representation $F_j$ also test all input stuck faults.

*Proof:* First assume that the representation $F_j$ is *binate* in variable $v_k$, i.e., that it contains (prime) cubes $c^1$ and $c^0$ with literal $v_k$ appearing positively and negatively, respectively. Now the argument of the proof of Theorem 5 shows that the test that showed $c^1$ is prime in literal $v_k$ gives a test, $x^*$, for input $v_k$ stuck-at-1. For this test, the good machine, $\eta$, has $v_j(x^*) = F_j(y^*, x^*) = 0$, but with $v_k$ stuck-at-1, cube $c^1$ turns on, so $v_j'(x) = 1$ in the fault machine. The same argument applied to cube $c^0$ yields a test for the input fault $v_k$ stuck-at-0.

Next assume that $F_j$ is *unate* [5] in $v_k$, i.e., that $F_j$ contains either positive or negative (in variable $v_k$) cubes like $c^1$ or $c^0$, but *not both*. Suppose, without loss of generality, that there exists cube $c^1 \in F_j$ which contains literal $v_k$, and is *irredundant*. Then, as in the proof of Theorem 5, there exists a test $x^*$ for which $c^1$ (and $F_j$) are turned on, but all other cubes in $F_j$ are turned off. Now if input $v_k$ to $F_j$ is stuck-at-0, then $c^1$ is turned off. Because $F_j$ is unate in $v_k$ no other cubes in $F_j$ are turned on by the $v_k$ stuck-at-0 fault, so we have $v_j = 1$ in the fault machine and $v_j' = 0$ in the fault machine.

Note that in this case, the argument involving the primality of cube $c^1$ still provides a test for $v_k$ stuck-at-1.

Now in either of the above two cases we have $v(x^*) \in \bar{D}_j$, else $v(x^*)$ would not contradict the nonprimality of $c^1$ (or $c^0$) in the binate case or the redundancy of $c^1$ in the unate case. Hence the differences between $v_j(x^*)$ and $v_j'(x^*)$ propagate to some primary output (because $v(x^*) \in \overline{DT}_{ij}$ for some $i$). Thus we have input stuck fault tests for both $v_k$ stuck-at-1 and $v_k$ stuck-at-0, and since this is true for any $k \in FI_j$, $F_j$ is 100-percent testable for input stuck faults if $F_j$ is prime and irredundant. □

At this point we have established that a prime and irredundant network is 100-percent testable for both internal and input stuck faults. But there are also multiple stuck faults which are guaranteed testable for a prime and irredundant Boolean network. To see this, observe that the internal stuck fault test, $x^*$, for the primality of variable $v_k$ in cube $c' \in F_j$ also tests for the primality of $v_k$ in all cubes $c^i$ such that $v(x^*) \in c^i$. So $x^*$ is a test for the internal faults $v_k$ stuck-at-1 in all of the cubes $c^i$, and is also a test for the *multiple internal stuck faults* for which $v_k$ is stuck-at-1 in *any subset* of the cube set $\{c^i\}$.

Similarly, the tests derived from the redundancy test are also, typically, tests for multiple stuck faults. To see this, note that the input vector, $x^*$, which contradicts the redundancy of cube $c' \in F_j$, gives a stuck fault test for any literal ($v_l$ or $\bar{v}_l$). In fact, any multiple internal stuck fault, comprised of any combination of the literals of cube $c'$ stuck at their opposite values, will also be tested by $x^*$. These latter multiple internal stuck fault tests are also multiple input stuck fault tests for any subset of the set of variables which have literals in any cube $c'$ such that $F_j$ is *unate* in these variables.

The principle at work in all these stuck fault test arguments appears to be the following. Suppose there exists vertex $v(x^*) \in \bar{F}_j \bar{D}_j$ (the care off set of $F_j$) which is distance one in variable $v_k$ from a vertex $\bar{v}(x^*) \in F_j \cup D_j$. Then $x^*$ tests for $v_k$ stuck-at-1. Conversely, if $v(x^*) \in F_j \bar{D}_j$ (the care on set), and is distance one in variable $v_k$ from $\bar{v}(x^*) \in \bar{F}_j \cup D_j$, then $x^*$ tests for $v_k$ stuck-at-0. It is precisely because the tests for primality and irredundancy tests are inherently obligated to isolate such vertex pairs that prime and irredundant networks are 100-percent testable for all input and internal single stuck faults. On this view, the process of automatic test generation is one in which one identifies care on set or care off set vertices which are located in the distance one "shells" surrounding the off and on sets, respectively. In books, e.g. [16], which take the traditional "simulation" (as opposed to don't care) viewpoint, this concept is expressed in terms of the so-called Boolean differences.

To see that output stuck faults are also included in this set of tests, note that $F_j \cup D_j \equiv 1$ implies that $\bar{F}_j \bar{D}_j = \varnothing$, i.e., $F_j$ has no "care" offset. In this case it is clear that no test exists for $v_j = y_j$ stuck at 1. But in this case every literal variable $v_k$ of every cube $c$ of $F_j$ can be deleted in the minimized version of $F_j$. This argument leads us to the interesting conclusion that if there exists any literal $v_k$ (or $\bar{v}_k$) of any cube $c \in F_j$ which is prime, then the test which has been shown to exist for the input fault "variable $v_k$ of cube $c \in F_j$ stuck at $b$" (where literal $v_j$ occurs as $b$ in cube $c$) is also a test for the output fault *output* "$v_j = y_j$ stuck at 1."

A similar argument applies to the case where $F_j \subseteq D_j$, which implies that $F_j \bar{D}_j = \varnothing$, i.e., $F_j$ has no care on set. This argument leads to the conclusion that any *input* $v_k$ stuck at $\bar{b}$ test (for cube $c \in F_j$) is also a test for the fault "*output* $v_j = y_j$ stuck at 0." Thus, in terms of the don't care sets, we can express the basis for the traditional "checkpoint" theorem, (cf. [10, theorem 2.3]).

We conclude that if $\eta$ is prime and irredundant, it is

testable for all output stuck faults as well, i.e., $\eta$ is 100-percent testable for input and output stuck faults.

Tests are also implicitly generated for an entirely different class of faults, the so called "extra device" faults. In the AND plane of a PLA, for example, an extra device fault could perhaps occur because of a discontinuity in an isolation mask, and thus polysilicon is erroneously laid over diffusion. This adds, in effect, an extra literal to some cube of the AND plane. We claim that this type of fault is also completely tested for by the operation of ESPRESSO_IIC when called at line 11 of Fig. 4. These tests are implicitly generated by the REDUCE operation. Applied to cube $c$, this operation attempts to add to $c$ all literals not originally present in $c$. If a literal cannot be added to $c$ without intersecting the care off set, ESPRESSO_IIC will generate a test, as discussed previously, for the corresponding extra-device fault. Of course, if such a literal can be added to $c$, the corresponding extra-device "fault" is not testable. However, this type of extra-device fault will cause no error in the IO behavior of the function.

The principal conclusion to be drawn from the above discussion is that prime and irredundant Boolean networks are far more than merely 100-percent testable for conventional input and output single stuck faults. In addition, they are testable for all the internal single stuck faults as well as for many multiple internal and input stuck faults as well as extra-device faults.

*Remark*

The tests for the stuck faults of Theorem 5 and its corollary are, in principle, supplied as a by-product of the 2-level minimization step (line 11 in ESPRESSO_MLD). In fact if, as in the proof of the above theorem, cube $c^*$ is being intersected with the representation $R_j = \overline{(F_j \cup D_j)}$ of the care off set, then if $v^* \in c^* R_j$, then $v^*_{PI} = x^*$ is in internal stuck fault test. Providing the tests as a by-product of the minimization is simply a matter of outputting or otherwise recording such vectors $x^*$ as they are encountered in the minimization. □

Thus the relationship between testing and logic minimization is quite profound. In fact, it follows that once all *internal* stuck fault tests have been identified and any discovered logical redundancies removed, the Boolean network is prime and irredundant. In brief, Boolean networks are prime and irredundant if and only if they are 100-percent testable (i.e., for conventional input or output faults *and* internal single stuck faults). Many *multiple* stack faults will usually be testable, and the tests for all of these various stack fault tests and supplies as a by-product of the minimization. No separate test generation phase is necessary.

As a final comment, we observe that one cannot decrease the testability of any *single* function, $F_j$, of a given Boolean network by making that function prime and irredundant. In fact, every single (input *and* internal) stuck fault which was testable prior to calling EXPAND and IRREDUNDANT_COVER to make $F_j$ prime and irredundant is still testable afterwards. Further, if $F_j$ was not previously prime and irredundant, there will now exist tests for input and/or internal stuck faults which were *not previously testable*. For example, the prime and irredundant Boolean network of Fig. 1(b) has three testable input faults which were not testable in the given network of Fig. 1(a).

### B. Logic Minimization as a By-Product of Test Generation

It is clear that all Boolean networks satisfying Definition 1 may be and-or decomposed into a "refined" Boolean network in which each node is either an OR gate or an AND gate. We assert that if a test generation tool is used to generate tests for all input and output single stuck faults, then the resulting Boolean network is prime and irredundant. This presupposes, of course, that if any "untestable" faults are discovered, the offending node or edge is deleted and the effect of this simplification is propagated to the rest of the network. In this way, logic minimization can be viewed as a by-product of test generation. However, such a procedure would not take advantage of the EXPAND IRREDUNDANT_COVER REDUCE cycle, which is responsible for ESPRESSO_MLD's ability to quickly reduce a given Boolean network into a prime, irredundant and R-minimal form. It is in comparison to this hypothetical procedure that we call ESPRESSO_MLD "efficient."

## VII. Conclusions

We have presented an approach to multilevel minimization based on don't care sets implied by embedding completely specified functions in a Boolean network. The presentation has including the following:

1) Definitions of prime and irredundant networks have been given, which are straightforward extensions of those for the 2-level case, and which are based on the notion of equivalence of two Boolean networks.
2) We have presented an algorithm, ESPRESSO_MLD, for multilevel minimization which transforms Boolean networks into prime, irredundant, and, with high probability, R-minimal form.
3) We have proven the physically plausible statement that prime and irredundant networks are 100-percent testable for conventional single stuck faults, and that the converse is also true if the internal stuck faults of Definition 10, which include multiple faults, are also testable.
4) We have further shown how the stuck fault tests derive straightforwardly from the minimization process.
5) We have defined the transitive fan-out don't care sets both in terms of network equivalence and in terms of the set of output stuck fault test vectors.
6) We have provided a proven construction of the representation (3.4) of the don't care set $d_j$ of the incompletely specified function $(f_j, d_j, r_j)$. We have observed that the representation $D_j$ is *not* invariant with respect to the minimization of another func-

tion, say $F_k$ (cf. discussion of Example 2, Section III).

It is a well-known fact that actual failure modes of fabricated chips do not always correspond to the fault model of single stuck faults. Nevertheless, it is also a fact that complete testability of the single stuck faults usually leads to a high percentage of working chips. We conjecture that the "extra" testability associated with prime and irredundant networks is at least partially responsible for this fact. The conjecture is based on the hypothesis that designers "naturally" attempt to design prime and irredundant networks, without consciously seeking to do so. As discussed above, if this occurs, many "extra" faults are tested.

Some readers may object to referring the transitive fan-out don't care set all the way back to the primary inputs, thus creating something of a misnomer. Note that $DT_{ij}$ could have been premultiplied by $\overline{DI}_j$, in Definition 6, and then $DT_{ij}$ really would have depended *solely* on the transitive fan-out of $F_j$. We believe that if Definition 6 were so altered, the remainder of the theory of Section III would remain valid (although we have not carried this exercise through rigorously). It is not clear whether or not $(\overline{DI}_j\, DT_{ij})$ has a more compact representation than $DT_{ij}$, since although the intersection with $\overline{DI}_j$ decreases the number of minterms, this operation also "fractures" the representation into smaller cubes. We prefer the form given for Definition 6, because of the direct connection to testability established by Theorem 4.

We have also given an exposition of the role of the ESPRESSO "REDUCE" operation in "reshaping" prime and irredundant Boolean networks into more efficient representations and in achieving the important property of R-minimality. It has been observed that this part of the minimization process is critical in breaking out of the local minima associated with merely prime and irredundant representations. ESPRESSO_MLD achieves high minimization quality by calling ESPRESSO_IIC, which loops through the EXPAND-IRREDUNDANT_COVER-REDUCE sequence. We have noted that while prime and irredundant status is achieved in one pass in the 2-level case, an iteration is required in the multilevel case, because of the interdependence of the individual 2-level functions embedded in the Boolean network.

Computational results obtained using an approximate "C" implementation of ESPRESSO_MLD were presented. We believe that the minimized Boolean networks $\eta'$ obtained for the test problems are prime and irredundant even though an approximated don't care set was used.

We note that further research into multilevel minimization as a test generation method might be worthwhile, especially in cases where 100-percent coverage is desired. The basic D-algorithm [25] and its variants [16] typically operate on Boolean Networks for which each function $F_j$ is a *primitive* gate (NAND, NOR, XOR, etc.). In contrast, ESPRESSO_MLD operates on a general Boolean Network, where each of the $F_j$ represents an *arbitrary* 2-level function. Thus ESPRESSO_MLD is applicable to alternative technologies such as domino logic, NMOS, and CMOS pluricells. Another contrast is that although some modern D-algorithm variant, e.g., FAN [16], might, because of its restricted applicability, be much faster in finding a single stuck fault, ESPRESSO_MLD might be faster in finding all such faults. This is because ESPRESSO_MLD can use $D_j$, once it is constructed, to repeatedly find all the internal and input stuck faults for the inputs (and output) of $F_j$. ESPRESSO_MLD does, in this sense, offer an interesting alternative to any D-algorithm variant in finding all stuck fault tests, especially in Boolean networks from such technologies as domino logic or complex CMOS, where individual nodes have "large" Boolean functions.

Of course we must keep in mind that the minimization process described in this paper applies to a technology independent level of representation. This is no problem for complex CMOS cells, but when standard cells are required, care must be taken to use a technology mapper (such a mapper is described in [2]) which preserves the properties of primality and irredundancy and, hence, 100-percent testability. It seems reasonable to conclude, therefore, that a Boolean network with one single function, $F_j$, which is *not* prime and irredundant should be minimized *if we can afford the computational expense*, else we will be putting "fat" into silicon. Future work must be done to characterize the domain of applicability of the reported minimization procedure. There certainly exist some practical Boolean networks which can be handled, and some which cannot.

## ACKNOWLEDGMENT

The authors acknowledge the benefits of helpful discussion and commentary from M. Lightner, A. R. Newton, T. Sasao, and T. Williams. A. DeGeus and D. Gregory contributed valuable technical input as well as making GE's "SOCRATES" expert system available to us for testing. Finally, we note that this work would not have been possible without the continued support of B. Chern of the National Science Foundation.

## REFERENCES

[1] K. A. Bartlett, W. Cohen, A. DeGeus, and G. D. Hachtel, "Synthesis and optimization of multilevel logic under timing constraints," *IEEE Trans. Computer-Aided Design*, pp. 582–596, Oct. 1986.

[2] K. A. Bartlett, D. G. Bostick, G. D. Hachtel, R. M. Jacoby, M. R. Lightner, P. H. Moceyunas, C. R. Morrison, and D. Ravenscroft, "BOLD: A multiple-level logic optimization system," *ICCAD87*.

[3] D. Brand, "Redundancy and don't cares in logic synthesis," *IEEE Trans. Computers*, vol. C-32, pp. 947–952, Oct. 1983.

[4] R. K. Brayton and C. T. McMullen, "The decomposition and factorization of Boolean expressions," in *Proc. Int. Symp. on Circuits and Systems*, Rome, pp. 49–54, 1982.

[5] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston: Kluwer Academic, 1984.

[6] R. K. Brayton and C. T. McMullen, "The Yorktown logic editor users manual," *IBM Technical Report*, Yorktown Heights, New York, 1984.

[7] R. K. Brayton, G. D. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli, "ESPRESSO-II: A new logic minimizer for program-

mable logic arrays," *IEEE 1984 Custom Integrated Circuits Conf.*, Rochester, NY, May 21-23, 1984.
[8] R. K. Brayton and C. T. McMullen, "Synthesis and optimization of multi-stage logic," in *Proc. IEEE Int. Conf. Computer Design*, Rye, NY, Oct. 1984.
[9] M. A. Breuer, "Generation of optimal code for expressions via factorization," *Commun. ACM*, vol. 12, no. 6, June 1969.
[10] M. Breuer and A. Friedman, *Diagnosis and Reliable Design of Digital Systems*. Potomac, MD: Computer Science Press, 1976.
[11] M. R. Dagenais, V. K. Agarwal and N. C. Rumin, "MCBOOLE: A new procedure for exact logic minimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, Jan. 1986.
[12] J. Darringer, D. Brand, J. Gerbi, W. Joyner and L. Trevillyan, "LSS: A system for production logic synthesis," *IBM J. Res. Develop.*, Sept. 1984.
[13] G. DeMicheli, "Symbolic minimization of logic functions," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 293-295, Nov. 1985.
[14] J. Dussault, C. Liaw, and M. Tong, "A high level synthesis tool for MOS chip design," in *Proc. 22nd Design Automation Conf.*, Albuquerque, NM, June 1985.
[15] A. D. Freidman, Logical Design of Digital Systems, 1975.
[16] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT Press, 1985.
[17] D. Gregory, K. Bartlett, A. J. deGeus and G. Hachtel, "SOCRATES: A system for automatically synthesizing and optimizing combinational logic," *23rd ACM/IEEE Design Automation Conf.*, June 1986.
[18] G. D. Hachtel and R. M. Jacoby, "Algorithms for multi-level tautology and equivalence," in *Proc. IEEE Int. Symp. on Circuits and Systems*, Kyoto, Japan, June 1985.
[19] G. D. Hachtel and R. M. Jacoby, "Verification algorithms for VLSI synthesis," in *Proc. NATO ASI on Logic Synthesis and Silicon Compilation for VLSI*, The Netherlands, 1987.
[20] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, pp. 443-458, Sept. 1974.
[21] T. Hoshino, M. Endo, and O. Karatsu, "An automatic logic synthesizer for integrated VLSI design systems," *IEEE 1984 Custom Integrated Circuits Conf. Proc.*, Rochester, NY, May 21-23, 1984.
[22] E. L. Lawler, "An approach to multilevel Boolean minimization," *J. ACM*, 1964.
[23] S. Muroga, *VLSI System Design When and How to Design Very-Large-Scale Integrated Circuits*. New York: Wiley, 1982.
[24] B. C. Rosales and P. Goel, "Results from application of a commercial ATG system to large-scale combinational circuits," in *1985 Int. Symp. on Circuits and Systems Proc.*, Kyoto, Japan. pp. 667-670.
[25] J. P. Roth, *Logic Synthesis and Verification*. Potomac, MD: Computer Science Press, 1980.
[26] J. P. Roth, "Minimization by the D-algorithm," *IEEE Trans. Computers*, vol. 35. May 1986.
[27] ——, private communication.
[28] R. Rudell, A. Sangiovanni-Vincentelli and G. DeMicheli, "A finite-state machine synthesis system," in *Int. Symposium on Circuits and Systems*, Kyoto, June 1985, pp. 647-650.
[29] R. Rudell and A. Sangiovanni, "ESPRESSO MV: Algorithms for multiple-valued logic minimization, in *Proc. Cust. Int. Circ. Conf.*, Portland, OR, pp. 230-234, May 1985.
[30] G. L. Smith, R. J. Bahnsen, and H. Halliwell, "Boolean comparison of hardware and flowcharts," *IBM J. Res. Develop.*, vol. 26, no. 1, Jan. 1982.
[31] T. W. Williams, private communication.
[32] L. Trevillyan, W. Joyner, and L. Berman, "Global flow analysis in automated logic design," *IEEE Trans. Comput.*, vol. C-35, pp. 77-81, Jan. 1986.

*

**Karen A. Bartlett** received the B.S. degree in computer science from Washington University, St. Louis, MO, in 1980 and the M.S. degree in electrical engineering from the University of Colorado, Boulder, in 1986.

From 1980 to 1983 she was employed by GTE Laboratories in Waltham, MA, where her projects included symbolic layout and VLSI database design and evaluation. Since June 1983 she has been active in the area of logic synthesis, first at General Electric's Microelectronic Center and then at the University of Colorado. She is currently employed by Seattle Silicon. Her research interests include logic synthesis, design optimization, and silicon compilation.

*

**Robert K. Brayton** (M'75-SM'78-F'81), for a photograph and a biography, please see page 437 of the April 1988 issue of this TRANSACTIONS.

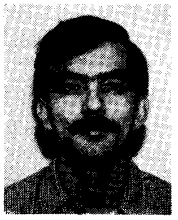
*

**Gary D. Hachtel** (S'62-M'65-SM'74-F'80), for a photograph and a biography, please see page 640 of the May 1988 issue of this TRANSACTIONS.

*

**Reily M. Jacoby,** for a photograph and a biography, please see page 640 of the May 1988 issue of this TRANSACTIONS.

*



**Christopher R. Morrison** received the B.S. degree in electrical engineering from Yale University, New Haven, CT, in 1977.

From 1977 to 1982, he was with IBM at the East Fishkill Facility in Hopewell Junction, NY. His last assignment at IBM was as a Senior Associate Engineer in the Computer Aided Circuit Design Department working on the circuit simulation program ASTAP. Since 1982, he has been on educational leave from IBM at the University of Colorado at Boulder, studying for the Ph.D. degree in electrical engineering. He received an IBM Fellowship award for academic years 1984-85 and 1985-86. He has done research work in routing algorithms, in particular, channel routing, and is currently working on optimization algorithms for multilevel combinational logic circuits. He has cowritten the program ESPRESSO_MLT (multilevel logic minimizer) and has written the program TECHMAP (technology mapper). In addition, he assembled the BOLD (Boulder Optimal Logic Design) system.

*

**Richard L. Rudell** received the B.S. degree in electrical engineering from the University of Minnesota in 1983 and the M.S. degree in electrical engineering from the University of California in 1986. He is currently working towards the Ph.D. degree in electrical engineering at the University of California, Berkeley.

From 1980 to 1983 he worked part-time at the Honeywell Corporate Computer Science Center in Minneapolis in the area of computer-aided design. This work was in the areas of the test pattern generation, high-level synthesis tools, and floor planning algorithms for VLSI. More recently, he has spent the summers of 1984 and 1985 working at the IBM T. J. Watson Research Center in the area of multiple-level logic synthesis. His current interests are in the area of multiple-level logic optimization, including design specification, factoring of Boolean equations, multiple-level minimization, and optimal technology mapping.

*

**Alberto Sangiovanni-Vincentelli** (M'74-SM'81-F'83), for a photograph and a biography, please see page 519 of the April 1988 issue of this TRANSACTIONS.

*

**Albert R. Wang** received the B.S. degree in computer science and applied mathematics from the University of California, San Diego, in 1984. He is currently working towards the Ph.D. degree in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley.

His current research interests include multiple-level logic synthesis and optimization, multiple-level logic verification, and sequential logic synthesis and optimization.