Optimization of Combinational Logic Circuits Based on Compatible Gates

Maurizio Damiani, Jerry Chih-Yuan Yang, *Student Member, IEEE*, and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*— This paper presents a set of new techniques for the optimization of multiple-level combinational Boolean networks. We describe first a technique based upon the selection of appropriate multiple-output subnetworks (consisting of so-called *compatible gates*) whose local functions can be optimized simultaneously. We then generalize the method to optimize larger and more arbitrary subsets of gates, called *unate subsets*. Because simultaneous optimization of local functions can take place, our methods are more powerful and general than Boolean optimization methods using *don't cares*, where only single-gate optimization can be performed. In addition, our methods represent a more efficient alternative to Boolean relations-based optimization procedures because the problem can be modeled by a *unate* covering problem instead of the more difficult *binate* covering problem. The method is implemented in program ACHILLES and compares favorably to SIS.**

I. Introduction

Logic synthesis has traditionally addressed optimization problems for two-level forms and multiple-level networks. Two-level synthesis has been intensely researched from theoretical and engineering perspectives, and efficient algorithms for exact [8], [13], [15], [17] and approximate [3], [10], [18] solutions are available.

Exact optimization algorithms for multiple-level logic networks have also been considered [12]. They are, however, generally impractical even for medium-sized networks. For this reason, many efficient approximation algorithms have been developed over the past decade. In [2], global-flow analysis is used to optimize a logic network. Other popular approaches can be classified according to the algebraic/Boolean type of operations they perform. Algebraic techniques, such as factoring and kerneling, are described in [4].

As algebraic methods do not take full advantage of the properties of Boolean algebra, a spectrum of Boolean optimization techniques has been developed in parallel. Such techniques consist mainly of iteratively refining an initial network by identifying subnetworks to be optimized, deriving their associated degrees of freedom (expressed by so-called *don't care conditions*), and replacing such subnetworks by simpler, optimized ones.

Manuscript received October 31, 1994; revised May 11, 1995. This work was supported in part by ARPA and NSF under contract MIP 9115432, and in part by a Hitachi Graduate Fellowship. This paper was recommended by Associate Editor M. Fujita.

M. Damiani is with the Dipartimento di Elettronica ed Informatica, Universitá di Padova, Via Gradenigo 6/A, Padova, Italy 35131.

J. C.-Y. Yang and G. De Micheli are with the Center for Integrated Systems, Stanford University, Stanford CA 94305 USA.

IEEE Log Number 9413603.

The independent optimization of the local function of a network, called **single-gate optimization**, lies at one end of the spectrum. It has been shown [1], [16] that the degrees of freedom associated to a single gate can be represented by a *don't care set*. Once this set is obtained, two-level synthesis algorithms can be used to optimize the subnetwork [1].

The concurrent optimization of several local functions, called **multiple-gate optimization**, lies at the other end of the spectrum. Multiple-gate optimization has been shown to offer potentially better quality networks as compared to single-gate optimization because of the additional degrees of freedom associated with the redesign of larger blocks of logic.

Exact methods for multiple-gate optimization, first analyzed in [5], have been shown to best exploit these degrees of freedom. Unfortunately these methods suffer from two major disadvantages. First, even for small subnetworks, the number of primes that have to be derived can be remarkably large. Second, given the set of primes, it entails the solution of an often complex *binate covering problem*, for which efficient algorithms are still the subject of investigation. As a result, the overall efficiency of the method is limited, and only relatively small networks can currently be handled.

Approximations to multiple-gate optimization include the use of *compatible don't cares* [16] which allows us to extend *don't care* based optimization to multiple functions by suitably restricting the individual *don't care* sets associated with each function. Although such methods are applicable to large networks, the restriction placed on *don't care* sets reduces the degrees of freedom and hence possibly the quality of the results.

The binate nature of the covering problem arises essentially from the arbitrariness of the subnetwork selected for optimization. In this paper, we develop alternative techniques for the optimization of multiple-output subnetworks. These techniques are based upon an accurate choice of the subnetworks to be optimized. The difficult binate covering step is avoided, and yet an optimization quality superior to *don't care* achieved because multiple local functions can be optimized simultaneously. To this regard, first we introduce the notion of **compatible set of gates** as a subset of gates whose optimization can be solved *exactly* by classical two-level synthesis algorithms. We show that the simultaneous optimization of compatible gates allows us to reach optimal solutions not achievable by conventional *don't care* methods. We then leverage upon these results and present an algorithm for the optimization of more general subnetworks in an internally unate network. The algorithms have been implemented and tested on several benchmark

x_0
x_1
x_{n_I}
F_0
F_1
F_{n_o}

Fig. 1. Example of a logic network.

circuits, and the results in terms of literal savings as well as CPU time are very promising.

II. Terminology

Let $\mathcal{B}$ denote the Boolean set {0, 1}. A k-dimensional Boolean vector $\mathbf{x} = [x_1, \ldots, x_k]^T$ is an element of the set $\mathcal{B}^k$ (bold-facing is hereafter used to denote vector quantities. In particular, the symbol $\mathbf{1}$ denotes a vector whose components are all 1).

A n_i-input, n_o-output Boolean function $\mathbf{F}$ is a mapping $\mathbf{F}: \mathcal{B}^{n_i} \rightarrow \mathcal{B}^{n_o}$. We use $\mathbf{x}$ to denote the set of primary inputs, and $\mathbf{F}$ to denote the set of primary output functions. A **literal function**, or **literal**, is the function expressed by a variable or its complement. A **cube** c is the product of some literals. A logic network is a collection of local single-output functions called **gates**. A set of local functions is denoted by $\mathbf{y}(\mathbf{x})$ where y_i is the variable associated with the output of each gate g_i, and in general can be expressed as a function of primary inputs. Fig. 1 illustrates a logic network.

The **cofactors** (or **residues**) of a function $\mathbf{F}$ with respect to a variable x_i are the functions $\mathbf{F}_{x_i} = \mathbf{F}(x_1, \ldots, x_i = 1, \ldots, x_n)$ and $\mathbf{F}_{x_i'} = \mathbf{F}(x_1, \ldots, x_i = 0, \ldots, x_n)$. The **universal quantification** or **consensus** of a function $\mathbf{F}$ with respect to a variable x_i is the function $\forall_{x_i}\mathbf{F} = \mathbf{F}_{x_i}\mathbf{F}_{x_i'}$. The **existential quantification** or **smoothing** of a function $\mathbf{F}$ with respect to x_i is defined as $\exists_{x_i}\mathbf{F} = \mathbf{F}_{x_i} + \mathbf{F}_{x_i'}$. A scalar function F_1 **contains** F_2 (denoted by $F_1 \geq F_2$) if $F_2 = 1$ implies $F_1 = 1$. The containment relation holds for two vector functions if it holds component-wise.

A function $\mathbf{F}$ is termed **positive unate** in x_i if $\mathbf{F}_{x_i} \geq \mathbf{F}_{x_i'}$, and **negative unate** if $\mathbf{F}_{x_i} \leq \mathbf{F}_{x_i'}$. Otherwise the function is termed **binate** in x_i. A function $\mathbf{F}$ positive (negative) unate in a variable x_i can always be expressed without using the literal $x_i'(x_i)$ [14].

The desired terminal behavior of a combinational network is **specified** by two functions, $\mathbf{ON}(\mathbf{x})$ and $\mathbf{DC}(\mathbf{x})$, the latter in particular representing the input combinations that either do not occur or such that the value of some of the network outputs is regarded as irrelevant [1].

The functions **ON** and **DC** identify the set of possible terminal behaviors for the network: specifications are met by an implementation, realizing a function $\mathbf{F}(\mathbf{x})$ if and only if $\mathbf{F}(\mathbf{x}) = \mathbf{ON}(\mathbf{x})$ for every input $\mathbf{x}$ not in **DC**.

Another, equivalent, description of the set of terminal behaviors is in terms of the functions $\mathbf{F}_{\min} = \mathbf{ON} \cdot \mathbf{DC}'$ and $\mathbf{F}_{\max} = \mathbf{ON} + \mathbf{DC}$. Specifications are met by $\mathbf{F}$ if

$$\mathbf{F}_{\min} \leq \mathbf{F} \leq \mathbf{F}_{\max}. \tag{1}$$

We consider hereafter specifications directly in terms of a pair $\mathbf{F}_{\min}$, $\mathbf{F}_{\max}$.

III. Previous Work

Most Boolean methods for multiple-level logic synthesis rely upon two-level synthesis engines. For this reason and in order to establish some essential terminology, we first review some basic concepts of two-level synthesis.

A. Two-Level Synthesis

Consider the synthesis of a (single-output) network whose output y is to satisfy (1), imposing a realization of y as a sum of cubes c_k

$$F_{\min} \leq y = \sum_{k=1}^{N} c_k \leq F_{\max}. \tag{2}$$

The upper bound in (2) holds *if and only if* each cube c_k satisfies the inequality

$$c_k \leq F_{\max}. \tag{3}$$

Any such cube is termed an **implicant**. An implicant is termed **prime** if no literal can be removed from it without violating the inequality (3). For the purpose of logic optimization, only prime implicants need be considered [14, 18]. Each implicant c_k has an associated **cost** w_k, which depends on the technology under consideration. For example, in PLA minimization all implicants take the same area, and therefore have identical cost; in a multiple-level context, the number of literals can be taken as cost measure [4]. The cost of a sum of implicants is usually taken as the sum of the individual costs.

Once the list of primes has been built, a minimum-cost cover of $F_{\min}$ is determined by solving

$$\textbf{minimize:} \sum_{k=1}^{N} \alpha_k w_k; \quad \textbf{subject to:} \quad F_{\min} \leq \sum_{k=1}^{N} \alpha_k c_k \tag{4}$$

where the Boolean parameters α_k are used in this context to **parameterize** the search space: they are set to 1 if c_k appears in the cover, and to 0 otherwise. The approach is extended easily to the synthesis of multiple-output circuits by defining **multiple-output primes** [14], [18]. A multiple-output prime is a prime of the product of some components of $\mathbf{F}_{\max}$. These components are termed the **influence set** of the prime.

Branch-and-bound methods can be used to solve exactly the covering problem. Engineering solutions have been thoroughly analyzed, for example in [18], and have made two-level synthesis feasible for very large problems [8], [15].

The constraint part of (4) can be rewritten as

$$\forall_{x_1,\ldots,x_n}\left(\sum_{k=1}^{N} \alpha_k c_k(\mathbf{x}) + F_{\min}'(\mathbf{x})\right) = 1. \tag{5}$$

The left-hand side of (5) represents a Boolean function F_α of the parameters α_i only; the constraint (4) is therefore equivalent to

$$F_\alpha = 1. \tag{6}$$

The conversion of (4) into (6) is known in the literature as *Petrick's method* [14].

Two properties of two-level synthesis are worth remarking in the context of this paper. First, once the list of primes has been built, we are guaranteed that no solution will violate the upper bound in 1), so that only the lower bound needs to be considered [as explicited by (4)]. Similarly, only the upper bound needs to be considered during the extraction of primes. Second, the effect of adding/removing a cube from a partial cover of $\mathbf{F}_{\min}$ is always predictable: that partial cover is increased/decreased. This property eases the problem of sifting the primes during the covering step, and it is reflected by the unateness of F_α: intuitively, by switching any parameter α_i from 0 to 1, we cannot decrease our chances of satisfying (6). These are important attributes of the problem that need to be preserved in its generalizations.

B. Don't Care-Based Multiple-Level Optimization

Two-level optimization is the basic engine in *don't care*-based multiple-level logic optimization, where it is used to iteratively optimize single-output gates in the network.

Consider a single-output subnetwork, with local output y, to be resynthesized. The primary output $\mathbf{F}$ of the overall network can be expressed in terms of the signal y

$$\begin{aligned}\mathbf{F} &= \mathbf{F}(\mathbf{x}, y) = y'\mathbf{F}_{y'} + y\mathbf{F}_y \\ &= (y\mathbf{1} + \mathbf{F}_{y'})(y'\mathbf{1} + \mathbf{F}_y).\end{aligned} \tag{7}$$

By replacing (7) in (1), it follows that y must satisfy

$$\mathbf{F}_{\min} \le y'\mathbf{F}_{y'} + y\mathbf{F}_y \le \mathbf{F}_{\max}. \tag{8}$$

A constraint on y similar to (1) can be obtained from (8) as follows: The upper bound in (8) holds if and only if $y'\mathbf{F}_{y'} \le \mathbf{F}_{\max}$ and $y\mathbf{F}_y \le \mathbf{F}_{\max}$, i.e.

$$y' \le \mathbf{F}_{\max} + \mathbf{F}'_{y'}; \quad y \le \mathbf{F}_{\max} + \mathbf{F}'_y. \tag{9}$$

Equation (9) can be rewritten as

$$\mathbf{F}'_{\max}\mathbf{F}_{y'} \le y\mathbf{1} \le \mathbf{F}_{\max} + \mathbf{F}'_y. \tag{10}$$

Similarly, the lower bound holds if and only if $\mathbf{F}_{y'} + y\mathbf{1} \ge \mathbf{F}_{\min}$ and $\mathbf{F}_y + y'\mathbf{1} \ge \mathbf{F}_{\min}$, i.e.

$$\mathbf{F}_{\min}\mathbf{F}_{y'} \le y\mathbf{1} \le \mathbf{F}'_{\min} + \mathbf{F}_y. \tag{11}$$

Equations (10) and (11) can be merged together, to obtain

$$\begin{aligned}&\mathbf{F}_{\min}\mathbf{F}'_{y'} + \mathbf{F}'_{\max}\mathbf{F}_{y'} \\ &\le y\mathbf{1} \le (\mathbf{F}_{\max} + \mathbf{F}'_y)(\mathbf{F}'_{\min} + \mathbf{F}_y).\end{aligned} \tag{12}$$

Equation (12) represents the exact degrees of freedom available in the synthesis of the signal y, and is formally identical to (1): the value of y is undetermined corresponding to those points for which the lower bound differs from the upper bound. Such points are the local *don't cares* for y, and are denoted by $\mathbf{DC}_y(\mathbf{x})$. Once the bounds (or, equivalently, the *don't cares*) for y are computed, ordinary two-level synthesis algorithms can be applied.[1]

C. Boolean Relations-Based Multiple-Level Optimization

Don't care-based methods allow us to optimize only one single-output subnetwork at a time. It has been shown in [5], [9] that this strategy may potentially produce lower-quality results with respect to a more general approach attempting the simultaneous optimization of multiple-output subnetworks.

Let $\mathbf{y} = [y_1, y_2, \ldots, y_m]$ denote the outputs of a subnetwork, to be resynthesized, and let $\mathbf{F}(\mathbf{x}, \mathbf{y})$ denote the network outputs, expressed in terms of the variables y_i. From inequality (1), the functional constraints on $\mathbf{y}$ are expressed by

$$\mathbf{F}_{\min}(\mathbf{x}) \le \mathbf{F}(\mathbf{x}, \mathbf{y}) \le \mathbf{F}_{\max}(\mathbf{x}). \tag{13}$$

An inequality like (13) describes a **Boolean Relation.**[2] The synthesis problem consists of finding a minimum-cost realization of $y_1, \ldots, y_m$ such that (13) holds. An exact solution algorithm, targeting two-level realizations, is presented in [5]. We illustrate the additional difficulties of the covering step with respect to the ordinary two-level synthesis process by means of the following example.

Example 1: Consider the optimization of gates g_1 and g_2, with outputs y_1 and y_2, in the circuit of Fig. 2. Assuming no external *don't care* conditions, $F_{\min} = F_{\max} = a'b' + (ac + bd) \oplus (a'c' + a'b')$, while $F = y_1 \oplus y_2 + a'b'$. Equation (13) then takes the form

$$\begin{aligned}&a'b' + (ac + bd) \oplus (a'c' + a'b') \\ &\le y_1 \oplus y_2 + a'b' \le a'b' + (ac + bd) \oplus (a'c' + a'b').\end{aligned}$$

By the symmetry of the network with respect to y_1 and y_2, cubes $a'c'$, ac, bd, $a'b'$ would be listed as implicants for both y_1 and y_2. Consider constructing now a cover for y_1 and y_2 from such implicants. An initial partial cover, for example obtained by requiring the cover of the minterm $abcd$ of $\mathbf{F}_{\min}$, may consist of the cube ac assigned to y_1. Consider now adding bd to y_2, in order to cover the minterm $abc'd$ of $F_{\min}$. Corresponding to the minterm $abcd$, now $y_1 \oplus y_2 = 0$, while $F_{\min} = 1$; that is, the lower bound of (13) is violated. Similarly, with the input assignment $a = 0$, $b = 1$, $c = 0$, and $d = 1$, the network output changed from the correct value 0 to 1, while $F_{\max} = 0$. Thus, also the upper bound is violated.

Contrary to the case of unate covering problems, where the addition of an implicant to a partial cover can never cause the violation of any functional constraints, here the addition of a single cube has caused the violation of **both** bounds in (13).□

[1] In practice, y is resynthesized by taking advantage also of the other internal signals available in the network. Implicants and primes are in this context expressed in terms of primary inputs and other network variables.

[2] An alternative formulation of a Boolean relation is by means of a **characteristic equation**: $R(\mathbf{x}, \mathbf{y}) = 1$, where R is a Boolean function. It could be shown that these two formulations are equivalent [6].

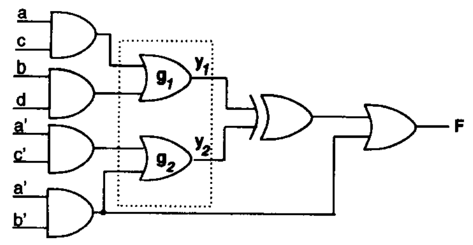


Fig. 2. Boolean relations optimization example.

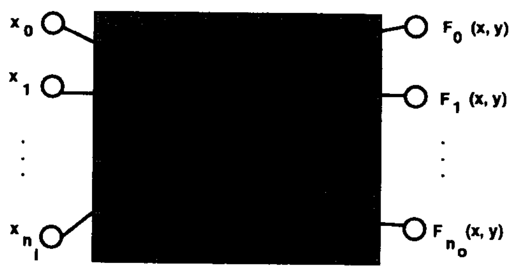


Fig. 3. Network with selected gates.

The difficulties of multiple-gate optimization are two-fold: first, the interplay of the various variables y_i that makes it impossible to isolate individual bounds for each function. Secondly, when trying to express (13) in a form similar to (2), that is, representing individual bounds on the signals y_i, each bound may depend on other variables y_j. In turn, it could be shown that this results in a **binate** covering step. Fast binate covering solvers are the subject of ongoing research [11]; nevertheless, the binate nature of the problem reflects an intrinsic complexity which is not found in the unate case. In particular, as shown in the previous example, the effect of adding/removing a prime to a partial solution is no longer trivially predictable, and both bounds in (13) may be violated by the addition of a single cube. As a consequence, branch-and-bound solvers may (and usually do) undergo much more backtracking than with a unate problem of comparable size, resulting in a substantially increased CPU time.

IV. Compatible Gates

The analysis of Boolean relations points out that binate problems arise because of the generally binate dependence of $\mathbf{F}$ on the variables y_i. In order to better understand the reasons for this type of dependency, we assume that the vertices of the logic network actually represent individual elementary **gates** (AND's, NAND's, OR's, NOR's, inverters).

We introduce the notion of **compatible gates** in order to perform multiple-gate optimization while avoiding the binate covering problem. In the rest of the paper, given a network output expression $\mathbf{F}(\mathbf{x}, \mathbf{y})$, $\mathbf{x}$ is the set of input variables and $\mathbf{y}$ is the set of gate outputs to be optimized. This relationship is shown in Fig. 3.

Definition 4.1: In a logic network, let $\mathbf{p}_j = p_j(x_1, \dots, x_n)$ and $\mathbf{q} = q(x_1, \dots, x_n)$, where $j = 1, 2, \dots, m$, represent functions that do not depend on $y_1, \dots, y_m$. A subset of gates $\mathcal{S} = \{g_1, \dots, g_m\}$ with outputs $y_1 \cdots y_m$ and functions $\mathbf{p}_j$ and $\mathbf{q}$ is said to be **compatible** if the network input-output behavior $\mathbf{F}$ can be expressed as

$$\mathbf{F} = \sum_{j=1}^{m} y_j \mathbf{p}_j + \mathbf{q} \tag{14}$$

modulo a phase change in the variables y_j or $\mathbf{F}$.

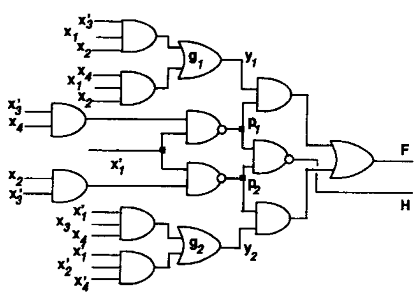


Fig. 4. Gates g_1 and g_2 are compatible.

As shown below, compatible gates can be optimized jointly without solving binate covering problems. Intuitively, compatible gates are selected such that their optimization can only affect the outputs in a monotonic or unate way, and thereby forcing the covering problem to be unate.

Example 2: Consider the two-output circuit in Fig. 4. Gates g_1 and g_2 are compatible because F and H can be written as

$$F = (x_1 + x_3 + x_4')y_1 + (x_1 + x_2' + x_3)y_2$$
$$H = 0y_1 + 0y_2 + ((x_1 + x_3 + x_4')(x_1 + x_2' + x_3))'.$$

□

The compatibility of a set $\mathcal{S}$ of gates is a Boolean property. In order to ascertain it, one would have to verify that all network outputs can indeed be expressed as in Definition 4.1. This task is potentially very CPU-intensive. In the next section, we describe the optimization algorithm. In Section VI, we present algorithms for constructing subsets of compatible gates from the network topology only.

V. Optimizing Compatible Gates

The functional constraints for a set of compatible gates can be obtained by replacing (14) into (13). From (14) we obtain

$$\mathbf{F}_{\min} \le \sum_{j=1}^{m} y_j \mathbf{p}_j + \mathbf{q} \le \mathbf{F}_{\max}. \tag{15}$$

Equation (15) can be solved using steps similar to that of two-level optimization. In particular, the optimization steps consist of *implicant extraction* and *covering*.

A. Implicant Extraction

Assuming that $\mathbf{q} \le \mathbf{F}_{\max}$, the upper bound of (15) holds *if and only if* for each product $y_j \mathbf{p}_j$ the inequality

$$y_j \mathbf{p}_j \le \mathbf{F}_{\max}$$

is verified, i.e., if and only if

$$y_j \mathbf{1} \le \mathbf{F}_{\max} + \mathbf{p}_j'; \qquad j = 1, \dots, m \tag{16}$$

TABLE I
MULTIPLE-OUTPUT PRIMES FOR EXAMPLE 5.3

	Primes	Influence sets
c_1	$x_1'x_2'x_3$	y_1, y_2
c_2	$x_1'x_2'x_4'$	y_1, y_2
c_3	$x_1'x_3x_4$	y_1, y_2
c_4	x_2x_4	y_1, y_2
c_5	$x_1x_2x_3'$	y_1, y_2
c_6	x_2x_3'	y_2
c_7	$x_1'x_3x_4'$	y_2
c_8	$x_1'x_2'$	y_1
c_9	$x_1'x_4$	y_1

or, equivalently

$$y_j \le F_{\max,j}; \qquad j = 1, \ldots, m \tag{17}$$

where $F_{\max,j}$ is the product of all the components of $\mathbf{F}_{\max} + \mathbf{p}_j'$. A cube c can thus appear in a two-level expression of y_j if and only if $c \le F_{\max,j}$. As this constraint is identical to (3), the prime-extraction strategies [14], [18] of ordinary two-level synthesis can be used.

Example 3: Consider the optimization problem for gates g_1 and g_2 in Fig. (4). From Example 2

$$p_1 = (x_1 + x_3 + x_4')'$$
$$p_2 = (x_1 + x_2' + x_3)'.$$

We assume no external *con't care* set. Consequently, $F_{\min} = F_{\max} = x_1x_2x_3' + x_2x_3x_4 + x_1'x_2'(x_3 + x_4')$. The Karnaugh maps of $F_{\min}$ and $F_{\max}$ are shown in Fig. 5(a), along with those of p_1 and p_2. Fig. 5(b) shows the maps of $F_{\max,1} = F_{\max} + p_1'$ and $F_{\max,2} = F_{\max} + p_2'$, used for the extraction of the primes of y_1 and y_2, respectively. The list of all multiple-output primes is given in Table I. Note that primes 1 through 5 can be used by both y_1 and y_2. □

B. Covering Step

Let N indicate the number of primes. For example, in the problem of Example (3), $N = 9$. We then impose a sum-of-products representation associated with each variable y_j

$$y_j = \sum_{k=1}^{N} \alpha_{jk} c_k \tag{18}$$

with the only restriction that $\alpha_{jk} = 0$ if y_j is not in the influence set of c_k. Since the upper bound of (15) is now satisfied by construction (i.e., by implicant computation), the minimization of $y_1, \ldots, y_m$ can be formulated as a minimum-cost covering problem

$$\mathbf{F}_{\min} \le \mathbf{q} + \sum_{j=1}^{m} \sum_{k=1}^{N} \alpha_{jk} c_k \mathbf{p}_j \tag{19}$$

whose similarity with (4) is evident, the products $c_k\mathbf{p}_j$ now playing the role of the primes of two-level synthesis.

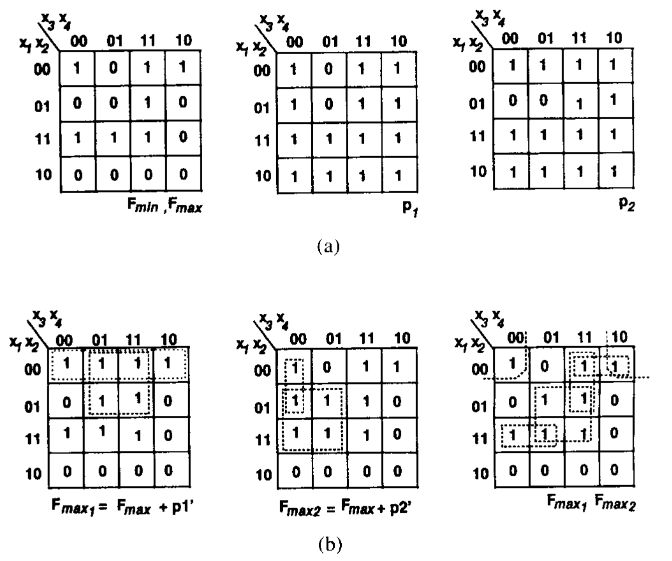


Fig. 5. (a) Maps of $F_{\min}, F_{\max}, p_1, p_2$. (b) Maps of $F_{\max,1}$, $F_{\max,2}$ and of the product $F_{\max,1}$ $F_{\max,2}$. Primes of y_1 and y_2 are shown in the maps of $F_{\max,1}$ and $F_{\max,2}$, respectively. The map of $F_{\max,1}F_{\max,2}$ shows the primes common to y_1 and y_2.

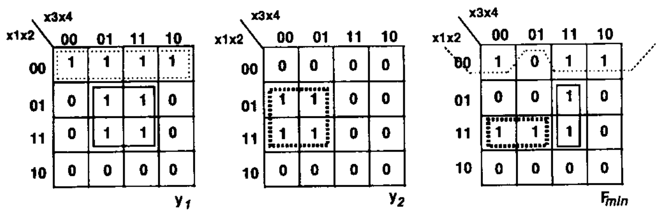


Fig. 6. A minimum-cost solution for the covering of $\mathbf{F}_{\min}$.

Example 4: In the optimization problem of Example 3, we are to solve the covering problem

$$F_{\min} \le p_1y_1 + p_2y_2.$$

Using the set of primes found in Example 3, y_1 and y_2 are expressed by

$$\begin{aligned} y_1 &= \alpha_{1,1}c_1 + \alpha_{1,2}c_2 + \alpha_{1,3}c_3 + \alpha_{1,4}c_4 \\ &\quad + \alpha_{1,5}c_5 + \alpha_{1,8}c_8 + \alpha_{1,9}c_9 \\ y_2 &= \alpha_{2,1}c_1 + \alpha_{2,2}c_2 + \alpha_{2,3}c_3 + \alpha_{2,4}c_4 \\ &\quad + \alpha_{2,5}c_5 + \alpha_{2,6}c_6 + \alpha_{2,7}c_7. \end{aligned}$$

The optimum solution has cost 6 and is given by $y_1 = x_1'x_2' + x_2x_4$; $y_2 = x_2x_3'$, corresponding to the assignments

$$\alpha_{1,1} = \alpha_{1,2} = \alpha_{1,3} = \alpha_{1,5} = \alpha_{1,9} = 0; \quad \alpha_{1,4} = \alpha_{1,8} = 1$$
$$\alpha_{2,1} = \alpha_{2,2} = \alpha_{2,3} = \alpha_{2,4} = \alpha_{2,5} = \alpha_{2,7} = 0; \quad \alpha_{2,6} = 1.$$

The initial cost, in terms of literals, was 12. The solution corresponds to the cover shown in Fig. 6, and resulting in the circuit of Fig. 7. □

It is worth contrasting, in the above example, the role of y_1 and y_2 in covering $F_{\min}$. Before optimization, p_1y_1 covered the minterms $x_1x_2x_3'x_4'$, $x_1x_2x_3'x_4$, $x_1x_2x_3x_4$ of $F_{\min}$, while p_2y_2 covered $x_1'x_2'x_3'x_4'$, $x_1'x_2'x_3x_4'$, $x_1'x_2x_3x_4$, $x_1'x_2'x_3x_4$. After optimization, y_1 and y_2 essentially "switched role" in

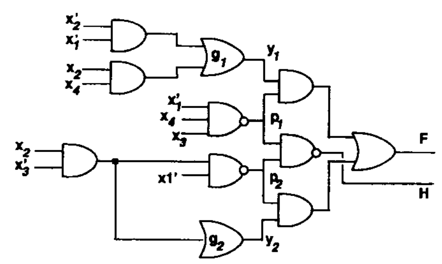


Fig. 7. Network resulting from the simultaneous optimization of compatible gates g_1 and g_2.

x1x2 \ x3x4	00	01	11	10
00	–	–	–	–
01	0	–	–	0
11	1	1	1	0
10	0	0	0	0

y_1

x1x2 \ x3x4	00	01	11	10
00	1	0	1	1
01	–	–	1	0
11	–	–	0	0
10	0	0	0	0

y_2

Fig. 8. *don't care* conditions associated with y_1 and y_2: only 1 literal can be removed.

the cover: p_2y_2 is now used for covering $x_1x_2x_3'x_4'$, $x_1x_2x_3'x_4$, while p_1y_1 covers all other minterms.

In the general case, the possibility for any of $y_1, \ldots, y_m$ to cover a minterm of $F_{\min}$ is evident from (15). Standard single-gate optimization methods based on *don't cares* [1] regard the optimization of each gate $g_1, \ldots, g_m$ as separate problems, and therefore this degree of freedom is not used. For example, in the circuit of Fig. (4), the optimization of g_1 is distinct from that of g_2. The *don't care* conditions associated with (say) y_1 are those minterms for which either $p_1 = 0$ or such that $p_2y_2 = 1$, and are shown in the map of Fig. 8, along with the initial cover. It can immediately be verified that y_1 can only be optimized into $x_1x_2x_3' + x_2x_4$, saving only one literal.

The *don't cares* for y_2 are also shown in Fig. 8. No optimization is possible in this case. Note also that the optimization result is (in this particular example) independent from the order in which g_1 and g_2 are optimized. Unlike the compatible gates case, it is impossible for the covers of y_1 and y_2 to "switch" role in covering $F_{\min}$.

VI. Finding Compatible Gates

In this section, we describe an algorithm for finding compatible gates based on network topology.

Definition 6.1: A network is termed **unate** with respect to a gate g if all reconvergent paths from g have the same parity of inversions. We call gate g a **u-gate**. A network is **internally unate** if it is unate with respect to each of its gates. All paths from g to a primary output z_i in an internally unate network have parity π_i, which is defined to be the **parity of** g with respect to z_i.

In the subsequent analysis, we make the assumption that the network is first transformed into its equivalent NOR-only form. In this case, the parity of a path is simply the parity of the path length.

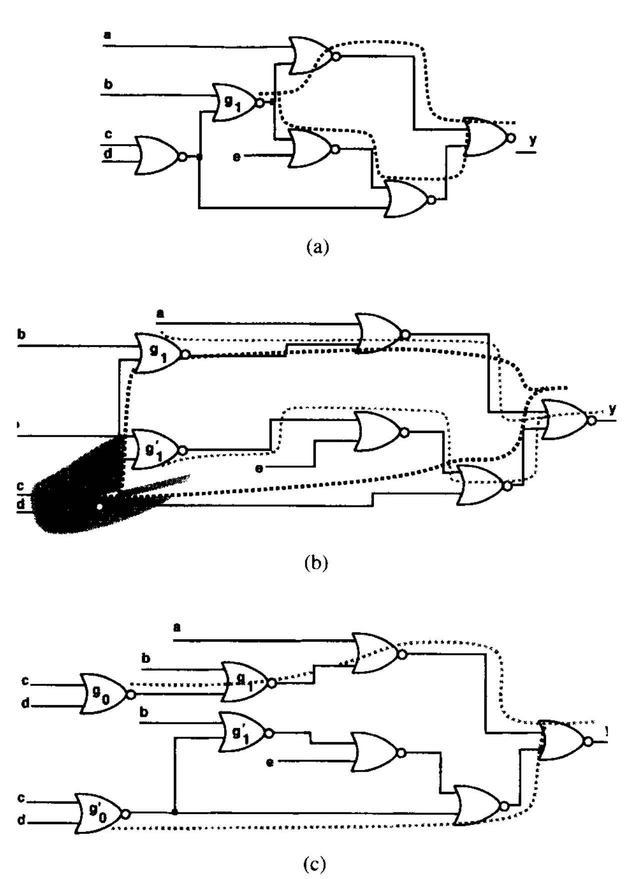


Fig. 9. Internally Unate Example: (a) Network not internally unate due to gate g_i; (b) Internally unate network after duplication (Duplicated gates are shaded).

In defining (14) for compatible gates, it is evident that the dependency of **F** on $y_1, \ldots, y_m$ must be unate. In order to increase the chances of finding sets of compatible gates, it is thus convenient to transform a network into an internally unate one. This is done by duplicating those gates whose fanouts contain reconvergent paths with different inversion parity.

Example 5: Consider the logic network shown in Fig. 9(a). The network is not internally unate because the reconvergent paths from gate g_1 to the output y do not have the same parity of inversions. We duplicate gate g_1 and its fan-in cone into g_1', shown by the shaded gates in Fig. 9(b). Now gates g_1 and g_1' are u-gates since there are no reconvergent paths of different parity from these gates. The same procedure is then repeated on g_0, which is split into g_0 and g_0'. The network is eventually internally unate (noting that the two reconvergent paths out of g_0' are of the same (even) parity). The increase in size is two gates. □

Note that the unfolding process is not recursive; rather, starting from the primary outputs, gates with incorrect fanout are duplicated. Therefore each gate is duplicated at most once. The resulting network is therefore at most twice the size of the original one. In practice, the increase is smaller.

Theorem 6.1 below provides a sufficient conditions for a set S of gates to be compatible. The following auxiliary definitions are required:

Definition 6.2: The **fanout gate set** and **fanout edge set** of a gate g, indicated by *FO*(g) and *FOE*(g), respectively, are the set of gates and interconnections contained in at least one path from g to the primary output. The **fanout gate set** and **fanout edge set** of a set of gates $\mathcal{S} = \{g_1, \ldots, g_m\}$, indicated by *FO*($\mathcal{S}$) and *FOE*($\mathcal{S}$), respectively, are

$$FO(\mathcal{S}) = \bigcup_{i=1}^{m} FO(g_i); \qquad FOE(\mathcal{S}) = \bigcup_{i=1}^{m} FOE(g_i). \tag{20}$$

Theorem 6.1 In a *NOR*-only network, let $\mathcal{S} = \{g_1, \ldots, g_m\}$ be a set of gates all of **even** (odd) parity and not in each others' fanout. Let $y_1, \ldots, y_m$ denote their respective outputs. Suppose that for all gates $g \in FO(\mathcal{S})$ has at most one input in $FOE(\mathcal{S})$.
The outputs of all gates in the **even** (odd) network can then be expressed by equations in the form of (14)

$$\mathbf{F} = \sum_{j=1}^{m} y_j \mathbf{p}_j + \mathbf{q}$$

Moreover, the output of each gate g in the network can be expressed by one of the following two equations: For gates of even parity

$$g = q_g + \sum_{j=1}^{m} p_{jg} y_j. \tag{21}$$

For gates of odd parity.

$$g = \left(q_g + \sum_{j=1}^{m} p_{jg} y_j\right)'. \tag{22}$$

Consequently, $\mathcal{S}$ is a set of compatible gates.

Proof: Assume the network gates to be sorted topologically, so that each gate precedes its fanout gates in the ordered list. Let *NGATES* denote the total number of gates. We prove the above proposition inductively, by showing that if it holds for the first $r - 1$ gates, then it must hold for the rth gate, $r = 1, \ldots,$ *NGATES*.

Consider the first gate, g_1. If $g_1 \in \mathcal{S}$, its output is simply y_1, which can be obtained from (21), by setting $q_{g_1} = 0$, $p_{1g_1} = 1$, $p_{jg_1} = 0$; $j = 2, \ldots, m$. If g_1 does not belong to $\mathcal{S}$, by the properties of topological ordering, its inputs can only be among the primary inputs, and consequently its output is still expressed by (21), by setting $p_{jg_1} = 0$.

Consider now the rth gate, g_r. Again, if $g_r \in \mathcal{S}$, the output is expressed by a single variable in $\{y_1, \ldots, y_m\}$, and therefore it satisfies the proposition. If g_r does not belong to $\mathcal{S}$, all its inputs are either primary inputs or gates $g_{r'}, r' < r$, for which the proposition is true by the inductive assumption. We distinguish two cases:

1) g_r is of even (odd) parity. Consequently, all its inputs have odd (even) parity, and only one of them is a function of the internal variables y_i. For simplicity, let g_0 denote the input that (possibly) depends on $y_1, \ldots, y_m$. The output of g_r is then expressed by

$$\left(\left(q_{g_0} + \sum_{j=1}^{m} p_{jg_0} y_j\right)' + \sum_{g_i \in FI(g_r)} q_{g_i}\right)' = q_{g_r} + \sum_{j=1}^{m} p_{jg_r} y_j$$

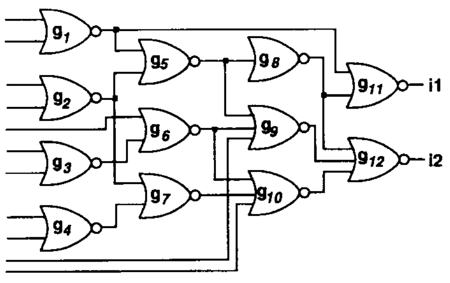


Fig. 10. Example of compatible gates.

where

$$q_{g_r} = q_{g_0} \prod_{g_i \in FI(g_r)} q'_{g_i}; \qquad p_{jg_r} = p_{jg_0} \prod_{g_i \in FI(g_r)} q'_{g_i}.$$

2) g_r is of odd (even) parity, and consequently all its inputs are from gates of even (odd) parity and are expressed by (21); therefore the output of g_r is expressed by

$$\left(\sum_{g_i \in FI(g_r)} \left(q_{g_i} + \sum_{j=1}^{m} p_{jg_i} y_j\right)\right)' = \left(q_{g_r} + \sum_{j=1}^{m} p_{jg_r} y_j\right)'$$

where

$$q_{g_r} = \sum_{g_i \in FI(g_r)} q_{g_i}; \qquad p_{jg_r} = \sum_{g_i \in FI(g_r)} p_{jg_i}.$$

By induction, the output of each gate (in particular, each primary output) is expressed by (21) or (22); therefore, the gates in $\mathcal{S}$ are compatible. □

Theorem 6.1 states that the gates of a subset $\mathcal{S}$ are compatible if all paths from the gates of $\mathcal{S}$ to the primary outputs meet **only** corresponding gates of **opposite** parity. The following theorem states a symmetric result, namely, that gates of $\mathcal{S}$ are compatible if their paths to the primary outputs meet **only** corresponding gates of **the same** parity. The proof is essentially the same as for Theorem 6.1, and it is therefore omitted.

Theorem 6.2: In a NOR-only network let $\mathcal{S} = \{g_1, \ldots, g_m\}$ denote a set of gates of **even** (odd) parity, and not in each other's fanout. Suppose that for all gates $g \in FO(\mathcal{S})$ of **odd** (even) parity, g has at most one input interconnection from $FOE(\mathcal{S})$. The gates of $\mathcal{S}$ are then compatible.

It can also be verified that in a multiple-output network, the gates of a set $\mathcal{S}$ are compatible if and only if Theorems 6.1 and 6.2 hold with respect to each output.

Example 6: In the internally unate, NOR-only network of Fig. 10, consider the set $\mathcal{S} = \{g_1, g_2, g_4\}$. All gates of $\mathcal{S}$ are of **odd** parity and not in each other's fanout. Moreover, $FO(\mathcal{S}) = \{g_5, g_7, g_8, g_9, g_{10}, g_{11}, g_{12}\}$ and for all gates in $FO(\mathcal{S})$ of **odd** parity (namely, g_8, g_9, g_{10}), there is only one input interconnection that belongs to $FOE(\mathcal{S})$. $\mathcal{S}$ then represents a compatible set by Theorem 6.1.

Similarly, the set $\mathcal{S} = \{g_3, g_4\}$ is compatible by Theorem 6.2, as in this case $FO(\mathcal{S}) = \{g_6, g_7, g_9, g_{10}, g_{12}\}$, and the gates of $FO(\mathcal{S})$ with even parity (namely, g_6 and g_7) have only one input interconnection in $FOE(\mathcal{S})$.

Other compatible sets are, for example, $\{g_1, g_{10}\}$ (by Theorem 6.1) and $\{g_5, g_7\}$ (Theorem 6.2).

It is worth noting that some gates (in this case, g_4) can appear in more than one compatible sets. □

Theorem 6.1 also provides a technique for constructing a set of compatible gates directly from the network topology, starting from a "seed" gate g and a parameter *rule* that specifies the desired criterion Theorem 6.1 or 6.2 to be checked during the construction. The algorithm is as follows:

```
COMPATIBLES($g$, rule)
label_fanout($g$, FO);
$\mathcal{S} = \{g\}$;
for($i = 1; i \le NGATES; i++$) {
    if ((is_labeled($g_i$) == FALSE) &
    (parity($g_i$) == parity($g$))) {
    label_fanout($g_i$, TMP);
    compatible = dfs_check($g_i$, parity($g$), rule);
    if(compatible) {
       label_fanout($g_i$, FO);
       $\mathcal{S} = \mathcal{S} \cup \{g_i\}$;
    }
    }
}
return ($\mathcal{S}$);
dfs_check(gate, parity, rule)
    if (incorrect_fanin(gate, parity, rule)) {
        return (FALSE);
    }
    if (gate → label == TMP) {
        result = TRUE;
        fanout = gate → fanout_list;
        while ((result == TRUE) & (fanout ≠ NULL)) {
            result = dfs_check(fanout, parity, rule);
            fanout = fanout → next;
        };
        return result;
    }
else {
    return (TRUE);
}
```

COMPATIBLES starts by labeling "*FO*" the fanout cone of g, as no gates in that cone can belong to a compatible set containing g. Labeled gates represent elements of the set $FO(\mathcal{S})$. All gates g_i that are not yet labeled and have the correct parity are then examined for insertion in $\mathcal{S}$. To this purpose, the fanout of g_i that is not already in $FO(\mathcal{S})$ is temporarily labeled "*TMP*," and then visited by *dfs_check* in order to check the satisfaction of *rule*. The procedure *dfs_check* performs a depth-first traversal on gate g_i. The traversal returns 0 whenever a violation of *rule* is detected. Otherwise, if the traversal reaches the primary outputs, or gates in $FO(\mathcal{S})$, then 1 is returned indicating that g_i is compatible. If g_i is compatible, it becomes part of $\mathcal{S}$ and its fanout is merged with $FO(\mathcal{S})$.

Example 7: Refer again to Fig. (10) for this example. Consider constructing a set of compatible gates around g_1, using rule (1). Gates g_5, g_8, g_9, g_{11}, and g_{12} are labeled first, because they belong to $FO(g_1)$. The first unlabeled gate is therefore g_2. The depth-first scan of its fanout reaches g_5 first, which has parity opposite to g_1. The check of the fanin of g_5 is therefore not needed. Gates g_7 and g_{10} are then reached. In particular, since g_{10} has the same parity as g_1, its fanin is checked to verify that there is indeed only one interconnection [in this case, (g_7, g_{10})] to gates in $\mathcal{S}$. Procedure *dfs_check* returns in this case a value *TRUE* for the compatibility of g_2 to g_1. □

VII. Unate Optimization

In the previous section, we showed that in the case of compatible gates, the functional constraints expressed by (13) can be reduced **exactly** to an upper bound [expressed by (17)] on the individual variables y_i and by a global covering constraint, expressed by (19). These could be solved by a two-step procedure similar to that of two-level optimization. We now sacrifice exactness and generalize this result to the optimization of more general, appropriate subsets $\mathcal{S}$ of u-gates. These subsets are hereafter named **unate subsets.**

A. Optimizing Unate Subsets

Let $\mathcal{S} = \{g_1, \ldots, g_m\}$ denote a set of gates of the same parity and not in each other's fanout. Assume that $\mathbf{F}$ is, say, positive unate with respect to $\{y_1, \ldots, y_m\}$. We can perform optimization on the subset of u-gates in a style that is totally analogous to compatible gates by dividing it into *implicant extraction* and *covering* steps.

B. Implicant Extraction

In this step, for each y_i to be optimized, a set of *maximal functions* is extracted. In particular, the maximal functions of each each y_i can be expressed as (23), which is similar to (17)

$$y_i \le G_{\max,j}; \quad j = 1, \ldots, m. \tag{23}$$

Appropriate implicants can then be extracted from (23).

Intuitively, the maximal functions are the largest functions that can be used while satisfying the bound $\mathbf{F} \le \mathbf{F}_{\max}$. Therefore, they represent the upper bounds on y_i. We introduce the following definition

Definition 7.1: A set of local functions

$$\{G_{\max,1}(\mathbf{x}), G_{\max,2}(\mathbf{x}), \ldots, G_{\max,m}(\mathbf{x})\}$$

is said to be **maximal** if

$$\mathbf{F}(\mathbf{x}, G_{\max,1}(\mathbf{x}), G_{\max,2}(\mathbf{x}), \ldots, G_{\max,m}(\mathbf{x})) \le \mathbf{F}_{\max}(\mathbf{x}) \qquad \forall \mathbf{x} \in \mathcal{B}^{n_i} \tag{24}$$

and the inequality (24) is violated when any $G_{\max,j}$ is replaced by a larger function $\tilde{F} > G_{\max,j}$.

The idea behind the notion of maximal functions is that by substituting each y_j by any function $\phi_j \le G_{\max,j}$, we are guaranteed that the upper bound

$$\mathbf{F}(\mathbf{x}, \phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})) \le \mathbf{F}_{\max}(\mathbf{x}) \tag{25}$$

will not be violated. The conditions

$$y_i \leq G_{\max,i}$$

therefore represent *sufficient* conditions for this bound to hold.

The following theorem provides means for finding a set of maximal functions. It also shows that computing such functions has complexity comparable with computing ordinary *don't care* sets.

Theorem 7.1: Let us consider a network with a totally ordered set of gates. Let $S = \{g_1, \ldots, g_m\}$ be a set of u-gates. The set of maximal functions, as defined by (24), with respect to a set of u-gates S can be obtained by

$$G_{\max,j} = f^{y_j} + DC_j \tag{26}$$

where f^{y_j} denotes the output function of g_j in the unoptimized network. DC_j represents the *don't care* set associated with g_j calculated with the following rule: the output functions for gates $g_1, \ldots, g_{j-1}$ are set to $G_{\max_1}, \ldots, G_{\max_{j-1}}$, respectively, and the output functions for gates $g_j, \ldots, g_m$, are set to f^{y_k}; $k = j, \ldots, m$.

Proof: The proof is divided into two parts. First, it is shown that the bounds $G_{\max,j} = f^{y_j} + DC_j$ satisfy (24). It is then shown, by contradiction, that these bounds are indeed maximal.

To prove the first part, suppose that maximal functions for the variables $y_1, \ldots, y_{j-1}$ have already been computed. They are such that

$$\mathbf{F}(\mathbf{x}, G_{\max,1}, \ldots, G_{\max,j-1}, f^{y_j}, f^{y_{j+1}}, \ldots, f^{y_m}) \leq \mathbf{F}_{\max}.$$

The constraint inequality on y_j can then be expressed by

$$\mathbf{F}_{\min} \leq \mathbf{F}(\mathbf{x}, G_{\max,1}, \ldots, G_{\max,j-1}, y_j, f^{y_{j+1}}, \ldots, f^{y_m}) \leq \mathbf{F}_{\max}$$

and is satisfied as long as y_j satisfies

$$f^{y_j} \cdot DC_j' \leq y_j \leq f^{y_j} + DC_j$$

where DC_j is the *don't care* set associated to y_j, under the theorem's assumptions. It is then guaranteed that

$$\mathbf{F}(\mathbf{x}, G_{\max,1}, \ldots, G_{\max,j-1}, G_{\max,j}, f^{y_{j+1}}, \ldots, f^{y_m}) \leq \mathbf{F}_{\max}$$

for $j = 1, \ldots, m$.

To prove maximality, it is sufficient to show that $G_{\max,j}$ cannot be replaced by any larger function. Suppose, by contradiction, that a different bound $\tilde{F}$ can be used, such that for some input combination $\mathbf{x}_0$ we have $G_{\max,j}(\mathbf{x}_0) = 0$ but $\tilde{F}(\mathbf{x}_0) = 1$. Notice that $G_{\max,j}(\mathbf{x}_0) = 0$ implies that $f^{y_j}(\mathbf{x}_0) = 0$ and $DC_j(\mathbf{x}_0) = 0$. Corresponding to $\mathbf{x}_0$, it must then be

$$\mathbf{F}(\mathbf{x}_0, G_{\max,1}(\mathbf{x}_0), \ldots, G_{\max,j-1}(\mathbf{x}_0), 0, \ldots, G_{\max,m}(\mathbf{x}_0)) = 0$$
$$\mathbf{F}(\mathbf{x}_0, G_{\max,1}(\mathbf{x}_0), \ldots, G_{\max,j-1}(\mathbf{x}_0), 1, \ldots, G_{\max,m}(\mathbf{x}_0)) = 1$$

and

$$\mathbf{F}_{\max}(\mathbf{x}_0) = 0$$

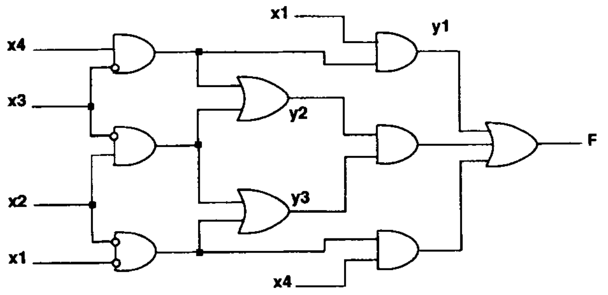


Fig. 11. Network for Example 8.

(or otherwise a change in the value of y_j could not affect the F and result in $DC_j(\mathbf{x}_0) = 0$.) If a larger bound $\tilde{F}_j$ could be used, this would mean that we could replace y_j by a function ϕ_j such that, in particular, $\phi_j(\mathbf{x}_0) = 1$, and replace all other functions $y_1, \ldots, y_m$ by $G_{\max,1}, \ldots, G_{\max,m}$. But in this case, we would have

$$\mathbf{F}(\mathbf{x}_0, G_{\max,1}(\mathbf{x}_0), \ldots, G_{\max,j-1}(\mathbf{x}_0), 1, \ldots G_{\max,m}(\mathbf{x}_0)) = 1$$

while $\mathbf{F}(\mathbf{x}_0) = 0$, violating the specifications. □

Note that the computation of each maximal function corresponds to finding the local *don't care* for the associated gate. Therefore, the maximal functions computation has the same complexity as computing the *don't care* conditions for each gate.

This theorem states that the maximal function for gate i depends on the maximal functions already calculated $(j < i)$. This means that unlike the case of compatible gates, the maximal function for a given gate may be not unique.

Example 8: For the network of Fig. 11, assuming no external *don't care* conditions, we find the maximal functions for y_1, y_2, and y_3. The DC_{y_j} terms correspond to the observability *don't care* at y_j, computed using the $F_{\max}$ of the previous gates

$$y_1 = x_1 x_3' x_4; \quad y_2 = x_3'(x_4 + x_2); \quad y_3 = x_3' x_2 + x_1' x_2'.$$

Maximal functions derived by Theorem 7.1 are

$$\begin{aligned} G_{\max,1} &= x_1 x_3' x_4 + DC_{y_1} = x_3' x_4 + (x_3' + x_4) x_1' x_2' \\ G_{\max,2} &= x_3'(x_4 + x_2) + DC_{y_2}(y_1 = G_{\max,1}) \\ &= x_4 + x_3' x_2' + x_1 x_2' + x_3 x_2 \\ G_{\max,3} &= x_3' x_2' + x_1' x_2 + DC_{y_3}(y_1 = G_{\max,1}, y_2 = G_{\max,2}) \\ &= x_3' x_2 + x_1' x_2' + x_4 x_3'. \end{aligned}$$

□

C. Covering Step

Equation (23) allows us to find a set of multiple-output primes for $y_1, \ldots, y_m$. The covering step then consists of finding a minimum-cost sum such that the lower bound of (13) holds.

We now present a reduction for transforming the covering step to the one presented for compatible gates. We first illustrate the reduction by means of an example.

Example 9: In Fig. (11), consider the combination of inputs $\mathbf{x}$ resulting in $\mathbf{F}_{\min}(\mathbf{x}) = 1$. With each such combination we can associate the set of values of y_1, y_2, y_3 such that $\mathbf{F}(\mathbf{x}, \mathbf{y}) = 1$. For instance, for the entry $x_1x_2x_3x_4 = 1001$, it must be $F_{x_1x_2'x_3'x_4}(\mathbf{y}) = y_1 + y_2y_3 = 1$. Let us now denote with $G(\mathbf{y})$ the left-hand side of this constraint, i.e., $G(\mathbf{y}) = y_1 + y_2y_3$. Notice that $G(\mathbf{y})$ is unate in each y_j and changes depending on the combination of values currently selected for x_1, x_2, x_3, x_4.

Any constraint $G(\mathbf{y}) = 1$ can be represented in a canonical form

$$G(\mathbf{y}) = (G_{y_1'y_2'y_3'} + y_1 + y_2 + y_3)(G_{y_1'y_2'y_3} + y_1 + y_2) \\ \cdots (G_{y_1y_2y_3'} + y_3)G_{y_1y_2y_3} = 1$$

which, in turn, is equivalent to the eight constraints

$$\begin{aligned} G_{y_1'y_2'y_3'} + y_1 + y_2 + y_3 &= 1 \\ G_{y_1'y_2'y_3} + y_1 + y_2 \quad &= 1 \\ \cdots \\ G_{y_1y_2y_3'} + y_3 \quad &= 1 \\ G_{y_1y_2y_3} \quad &= 1. \end{aligned} \tag{27}$$

By introducing an auxiliary variable z_j for each y_j, we can rewrite (27) as

$$G(\mathbf{z}) + z_1'y_1 + z_2'y_2 + z_3'y_3 = 1 \qquad \forall z_1, z_2, z_3$$

or, equivalently

$$G'(\mathbf{z}) \le z_1'y_1 + z_2'y_2 + z_3'y_3.$$

In this particular example, we get

$$(z_1 + z_2z_3)' \le z_1'y_1 + z_2'y_2 + z_3'y_3.$$

□

Example 9 shows a transformation that converts the covering problem of arbitrary u-gates into a form that is similar to optimization of compatible gates, i.e., (15).

More generally, corresponding to each combination $\mathbf{x}$ such that $\mathbf{F}_{\min}(\mathbf{x}) = 1$, the constraint $\mathbf{F}(\mathbf{x}, \mathbf{y}) = 1$ can be reexpressed as

$$\mathbf{F}(\mathbf{x}, \mathbf{z}) + z_1'y_1 + z_2'y_2 + \cdots + z_m'y_m = 1.$$

The resulting covering problem to find the minimum-cost solution is analogous to the compatible gates case. The transformation is formalized in the following theorem.

Theorem 7.2: Given $\mathbf{F}(\mathbf{x})$, let $\mathbf{y}$ be a set of u-gates with respect to $\mathbf{F}$. Let $\mathbf{z} = [z_1, \ldots, z_m]$ denote m auxiliary Boolean variables. The lower bound of (13) holds if and only if

$$\mathbf{F}_{\min} \le \mathbf{F}(\mathbf{x}, \mathbf{z}) + \sum_{j=1}^{m} y_j(z_j'\mathbf{1}) \quad \forall \mathbf{z}. \tag{28}$$

Proof: We first show by contradiction that

$$\mathbf{F}(\mathbf{x}, \mathbf{y}) \le \mathbf{F}(\mathbf{x}, \mathbf{z}) + \sum_{j=1}^{m} y_j(z_j'\mathbf{1}), \quad \forall x, y, z. \tag{29}$$

Equation (29) can be violated only by a combination $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ such that one component of $\mathbf{F}(\mathbf{x}_0, \mathbf{y}_0)$ takes value 1, the same component of $\mathbf{F}(\mathbf{x}_0, \mathbf{z}_0)$ takes value 0, and the rightmost term of (29) takes value 0. In any such combination, there must be at least one value $y_{i,0} = 1$ and $z_{i,0} = 0$ (or otherwise, by the unateness of $\mathbf{F}$, we would have $\mathbf{F}(\mathbf{x}_0, \mathbf{y}_0) \le \mathbf{F}(\mathbf{x}_0, \mathbf{z}_0)$). But if there exists an index i such that $y_{i,0} = 1, z_{i,0} = 0$, then the rightmost term of (29) takes value 1, and the right-hand side of the inequality holds, a contradiction.

Therefore, $\mathbf{F}_{\min}(\mathbf{x}) \le \mathbf{F}(\mathbf{x}, \mathbf{y})$ together with (29) implies

$$\mathbf{F}_{\min}(\mathbf{x}) \le (\mathbf{x}, \mathbf{z}) + \sum_{j=1}^{m} y_j(z_j'\mathbf{1}).$$

To complete the proof, it must now be shown that $\mathbf{F}_{\min}(\mathbf{x}) \le \mathbf{F}(\mathbf{x}, \mathbf{z}) + \sum_{j=1}^{m} y_j(z_j'\mathbf{1})$, $\forall \mathbf{z}$ implies $\mathbf{F}_{\min}(\mathbf{x}) \le \mathbf{F}(\mathbf{x}, \mathbf{y})$. Suppose, by contradiction, that this is not true. There exists then a value $\mathbf{x}_0$, $\mathbf{y}_0$ such that some component of $\mathbf{F}_{\min}(\mathbf{x}_0)$ takes value 1, $\mathbf{F}(\mathbf{x}_0, \mathbf{y}_0)$ takes value 0, but $\mathbf{F}_{\min}(\mathbf{x}_0) \le \mathbf{F}(\mathbf{x}_0, \mathbf{z}) + \sum_{j=1}^{m} y_{j,0}(z_j'\mathbf{1})$, $\forall \mathbf{z}$. In this case, it must be $\mathbf{F}(\mathbf{x}_0, \mathbf{z}) + \sum_{j=1}^{m} y_{j,0}(z_j'\mathbf{1}) = 1$, regardless of $\mathbf{z}$. But this implies that, for $\mathbf{z} = \mathbf{y}_0$, $\mathbf{F}(\mathbf{x}_0, \mathbf{z}) = 1$, i.e.,$\mathbf{F}(\mathbf{x}_0, \mathbf{y}_0) = 1$, a contradiction. □

Equation (28) has the same format of (15), with $\mathbf{q}$ and $\mathbf{p}_j$ being replaced by $\mathbf{F}(\mathbf{x}, \mathbf{Z})$ and $z_j'\mathbf{1}$, respectively. Theorem 7.2 thus allows us to reduce the covering step to the one used for compatible gates. Theorems 7.1 and 7.2 show that the algorithms presented in Section V can be used to optimize arbitrary sets of gates with the same parity, without being restricted to sets of compatible gates only.

VIII. Implementation and Results

The implementation of the algorithms presented in Sections V and VII is as follows: The original networks are first transformed into a unate, NOR-only description. All internal functions are represented using BDD's [7]. For each unoptimized gate g_i, the following heuristic is used. First, we try to find a set of compatible gates for g_i, called $\mathcal{S}_c$. In the case where not enough compatible gates can be found, we find a set of gates that are unate with respect to g_i, called $\mathcal{S}_a$.

In the case where $\mathcal{S}_c$ is optimized, we use (14) to extract the functions $\mathbf{p}_j$ and $\mathbf{q}$. In particular, $\mathbf{q}$ is computed by setting y_j to 0. The functions $\mathbf{p}_j$ are then computed by setting y_j to 1, with $y_i; i \ne j$ stuck-at-0.

In the case of optimizing arbitrary unate network $\mathcal{S}_a$, Theorem 7.1 is used to determine the maximal functions for each y_j. Note that optimizing $\mathcal{S}_c$ is preferable because for a set of m compatible gates, $m + 1$ computations for $\mathbf{p}_j$ and $\mathbf{q}$ are needed to obtain all the required *don't cares*. For $\mathcal{S}_a$, two computations (with y_j stuck-at-0 and stuck-at-1) are required for the extraction of the *don't care* set of each variable y_j, resulting in a total of $2m$ computations.

TABLE II
OPTIMIZATION RESULTS. RUNTIMES ARE IN SECONDS ON DEC5000/240

	Initial Stat.		Interconn.		Literals(fac)		Gates		*CPU time*	
Circuit	Int.	Gates	Achilles	SIS	Achilles	SIS	Achilles	SIS	Achilles	SIS
cm85a	108	63	67	77	42	46	31	34	1.5	1.2
cml62a	113	60	99	102	47	49	41	52	1.8	1.3
pml	130	60	67	78	47	52	31	36	1.6	1.3
9symml	375	152	288	325	163	186	88	101	108.4	64.2
alu2	924	262	366	570	303	362	215	231	309.7	403.0
alu4	1682	521	902	1128	612	703	420	487	1612.6	1718.5
apex6	1141	745	1009	1315	687	743	589	639	115.1	30.3
C499	945	530	913	945	505	552	498	530	202.1	133.6
C880	797	458	643	731	355	409	295	342	340.6	30.7
C1908	936	489	828	891	518	542	445	482	422.1	138.8

A set of primes for the gate outputs is then constructed. Because of the large possible set of primes, we limit the number of primes considered to n. As a greedy heuristic, we first choose single-literal primes to fill n. The limit n is decided experimentally, and set to 300 in the results presented here. The BDD of $\mathbf{F}(\mathbf{x}, \mathbf{z})$ is then built, and the covering problem solved. Networks are then iteratively optimized until no improvement occurs, and eventually folded back to a binate form. The algorithms presented in this paper were implemented in C program called ACHILLES, and tested against a set of MCNC synthesis benchmarks.

Table II provides a comparison of ACHILLES with SIS using *script.rugged*. The column *Initial Stat.* lists the network statistics before optimization, where *Int.* is number of internal interconnections and *gates* is the gate count. The column *Interconn.* shows number of interconnections after optimization. The *gates* column compares final gate counts. *Literal* column shows the final literals in factored form. The results in the table show that ACHILLES performs better than SIS for all figures of merit. In particular, ACHILLES does 11% better than SIS in factored literals.

Note that *script.rugged* was chosen because it is the most robust script of the SIS script suite, and it matches closely to our type of optimization. Our objective was to compare optimization results based only on Boolean operations, namely compatible gates versus *don't cares*. The *script.rugged* calls *full_simplify* [19], which computes observability *don't cares* to optimize the network.

The table shows that the ACHILLES runtimes are competitive with that of SIS. In this implementation, we are more interested in the quality of the optimization than the efficiency of the algorithms, therefore an *exact* covering solver is used. We can improve the runtime in the future by substituting a faster heuristic or approximate solvers (such as used in ESPRESSO [18]).

IX. CONCLUSION

In this paper, we presented a comparative analysis of approaches to multilevel logic optimization, and described new algorithms for simultaneous multiple-gate optimization. The algorithms are based on the notion of **compatible gates** and unate networks. We identify the main advantage of the present approach over previous solutions in its capability of exact minimization of suitable multiple-output networks, by means of traditional two-level optimization algorithms. Experimental results show an improvement of 11% over existing methods.

REFERENCES

[1] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, *et al.*, "Multilevel logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 723–740, June 1988.

[2] C. L. Berman and L. H. Trevillyan, "Global flow optimization in automatic logic design," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 557-564, May 1991.

[3] R. K. Brayton, C. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* New York: Kluwer Academic, 1984.

[4] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062-1081, Nov. 1987.

[5] R. K. Brayton and F. Somenzi, "An exact minimizer for boolean relations," in *Proc. Int. Conf. Computer-Aided Design*, pp. 316–319, Nov. 1989.

[6] F. M. Brown, *Boolean Reasoning.* New York: Kluwer Academic, 1990.

[7] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.

[8] O. Coudert, J. C. Madre, and H. Fraisse, "A new viewpoint on two-level logic minimization," in *Proc. Design Automat. Conf.*, pp. 625–630, June 1993.

[9] M. Fujita, Y. Tamiya, Y. Matsunaga, and K. C. Chen, "Muti-level logic synthesis for boolean relations," *VLSI '91*, 1991.

[10] S. J. Hong, R. G. Cain, and D. L. Ostapko, "Mini: A heuristic approach to logic minimization," *IBM J. Res. Develop.*, 1974.

[11] S. W. Jeong and F. Somenzi, "A new algorithm for the binate covering problem and its application to the minimization of boolean relations," in *Proc. Int. Conf. Computer-Aided Design*, pp. 417–420, 1992.

[12] E. L. Lawler, "An approach to multilevel boolean minimization," *ACM J.*, vol. 11, pp. 283–295, July 1964.

[13] E. J. McCluskey, "Minimization of boolean functions," *Bell Syst. Tech J.*, vol. 35, pp. 1417-1444, Nov. 1956.

[14] ——, *Logic Design Principles With Emphasis on Testable Semicustom Circuits.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[15] P. McGeer, J. Sanghavi, and R. K. Brayton, "Espresso-signature: A new exact minimizer for logic functions," in *Proc. Design Automat. Conf.*, pp. 618–624, June 1993.

[16] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method—design of logic networks based on permissible functions," *IEEE Trans. Comput.*, vol. 38, pp. 1404–1424, Oct. 1989.

[17] W. Quine, "The problem of simplifying truth functions," *Amer. Mathemat. Monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[18] R. L. Rudell and A. L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD–6, pp. 727–750, Sept. 1987.

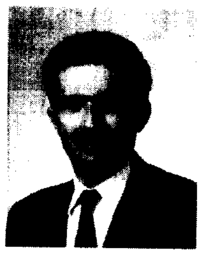
[19] H. Savoj, R. K. Brayton and H. Touati, "Extracting local don't cares and network optimization," in *Proc. Int. Conf. Computer-Aided Design*, pp. 514–517, Nov. 1991.

Maurizio Damiani graduated in electrical engineering at the University in Bologna in 1987. He received the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA.

Since 1992, he is Associate Professor at the University of Padua. His current research interests include formal modeling, synthesis and testing of sequential finite-state circuits and systems.

Jerry Chih-Yuan Yang (S'90) received the B.S.E.E. and M.S.E.E. degrees from Stanford University, Stanford, CA. He is pursuing the Ph.D. degree at the same university.

His research interests include logic synthesis, system-level modeling, verification, and synthesis; and finite-state based synthesis methods.



Giovanni De Micheli (S'82–M'83–SM'89–F'94) received the Dr. Eng. degree, in nuclear engineering from the Politecnico di Milano, Italy, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley in 1979, 1980 and 1983 respectively.

From 1984 to 1986, he worked at the IBM T.J. Watson Research Center, Yorktown Heights, NY, where he was project leader of the Design Automation Workstation group. Previously he held positions at the Department of Electronics of the Politecnico di Milano, Italy and at Harris Semiconductor, Melbourne, Florida. He is currently an Associate Professor of Electrical Engineering and Computer Science at Stanford University, Stanford, CA. His research interests include several aspects of the computer-aided design of integrated circuits with particular emphasis on automated synthesis, optimization and verification of VLSI circuits. He is author of *Synthesis and Optimization of Digital Circuits*, (McGraw-Hill, 1994), coauthor of *High-level Synthesis of ASIC's under Timing and Synchronization Constraints*, (Kluwer, 1992) and coeditor of *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, (Martinus Nijhoff Publishers). He was also codirector of the Advanced Study Institute on Logic Synthesis and Silicon Compilation, held in L'Aquila, Italy, under the sponsorship of NATO in 1986 and in 1987.

Dr. De Micheli was granted a Presidential Young Investigator award in 1988. He received the 1987 Best Paper Award for the best paper published on the IEEE Transactions on Computer-Aided Design and two *Best Paper Awards* at the Design Automation Conference, in 1983 and in 1993. He is an Associate Editor of the IEEE TRANSACTIONS ON VLSI SYSTEMS and of Integration: The VLSI Journal. He was technical and general chairman of the International Conference on Computer Design in 1988 and 1989 respectively. He has served as member of the technical committee of the ICCD, ICCAD, and DAC Conferences.