Logic Synthesis for Engineering Change

Chih-Chang Lin, *Member, IEEE*, Kuang-Chien Chen, *Member, IEEE*, and Malgorzata Marek-Sadowska, *Fellow, IEEE*

***Abstract*— During the process of very large scale integration design, specifications are often changed. To preserve as large a portion of the engineering effort as possible, it is desirable that such changes will not lead to a very different design. In this work, we consider logic synthesis algorithms for handling engineering changes. To solve it, we propose a combination of multiple-error diagnosis and logic minimization techniques. Given a new specification and an existing synthesized network, our algorithms first identify the candidate signals in the network, and then synthesize the candidate functions. The synthesis step utilizes the existing network as much as possible so that the new specification can be realized with minimal changes.**

***Index Terms*—Automatic test pattern generation (ATPG), logic synthesis.**

I. Introduction

In a typical very large scale integration (VLSI) design process, specifications are often changed in order to correct design errors and accommodate changes, or to meet certain design constraints such as area, timing, and power consumption. Since a large engineering effort may already have been invested (e.g., the layout of a chip may have been obtained), it is desirable that these changes in specification do not lead to a very different design. This kind of problem is called the *engineering change* (EC) [38], [20], [21] or *incremental synthesis* [33], [6] problem.

As automatic logic synthesis [7], [3], [24], [13], [12] becomes popular, the issue of how to handle engineering changes gains even more importance. This is because synthesis tools usually perform global transformations to achieve good quality results. Small and local changes in the specification may have global effects and produce a very different network. As a result, designers usually have to manually modify the synthesized network to realize changes in the specification. This practice not only increases the chance of introducing inconsistencies between the higher-level specification and the final network, but it also is an error-prone process. It often fails because the correspondence between the specification and synthesized network cannot be easily identified (e.g., a signal in the VHDL specification may not appear as a signal in the synthesized network). Therefore, there is an urgent need for synthesis algorithms which can handle engineering changes effectively.

Manuscript received May 21, 1996; revised October 15, 1996. This work was supported in part by the National Science Foundation (NSF) under Grant MIP 9419119 and by the California MICRO program through Fujitsu, Xilinx, and Actel. This paper was recommended by Associate Editor A. Saldanha.

C.-C. Lin and K.-C. Chen are with VerPlex Systems, Inc., Santa Clara, CA 95054 USA

M. Marek-Sadowska is with the Electrical and Computer Engineering Department, University of California, Santa Barbara, CA 93106 USA.

Publisher Item Identifier S 0278-0070(99)01596-1.

Fig. 1(a) shows a network which represents the original specification. In general, a specification may be given in a high-level description language, but here for illustrating purposes, we simply take the network in Fig. 1(a) as the initial specification. After optimizing the network in Fig. 1(a), the result is the network shown in Fig. 1(c).

Now suppose the specification in Fig. 1(a) has been slightly modified by changing the gate p_5 from an XOR gate to an AND gate, as shown in Fig. 1(b). After applying the same synthesis procedures as before, we obtain the network shown in Fig. 1(d). Although the change in specification arises from a local modification, common logic synthesis procedures do not localize such a change, and the networks in Fig. 1(c) and Fig. 1(d) are quite different (i.e., the number of gates has been changed from 10–9, and five out of the nine gates have different fanins or fanouts).

Another way to handle the specification change is to modify the network in Fig. 1(c) directly. In this way we can obtain a network similar to Fig. 1(c), and still achieve the new specification as shown in Fig. 1(b). A manual EC cannot be easily applied in this case, because the signal corresponding to the gate p_5 in Fig. 1(a) is no longer available in the optimized network Fig. 1(c). Therefore, although we know the change arises from the modification of p_5, it is difficult to tell in Fig. 1(c) where and how the modifications should be done.

On the other hand, a good synthesis procedure which considers engineering changes can modify the network in Fig. 1(c) minimally, so that the resulting network is functionally equivalent to the specification in Fig. 1(b). In later sections, we will discuss such EC algorithms. The result of applying these algorithms on the network in Fig. 1(c) is shown in Fig. 1(e). The two networks differ in that the gates k_0, k_1 and k_2 have been removed and a gate k_3 has been added. Note that the removal of gates may be beneficial for layout and timing, and the change made here is local.

Note that the changes made at high levels of design abstraction can potentially introduce large changes in the final design. For example, suppose a design is described in terms of a finite state machine, and modifications change the number of states and their transitions. Then, during the synthesis, a state encoding different from the original one may be used, potentially leading to a very different network. Therefore, it should not be expected that engineering changes can always be done with very few modifications. Here, we will focus on the core problem of the engineering change, i.e., handling functional specification changes for combinational networks.

The remainder of this paper is organized as follows. Section II provides the appropriate background and definitions. Section III reviews related work. Sections IV–VI discuss our approach to logic synthesis for engineering

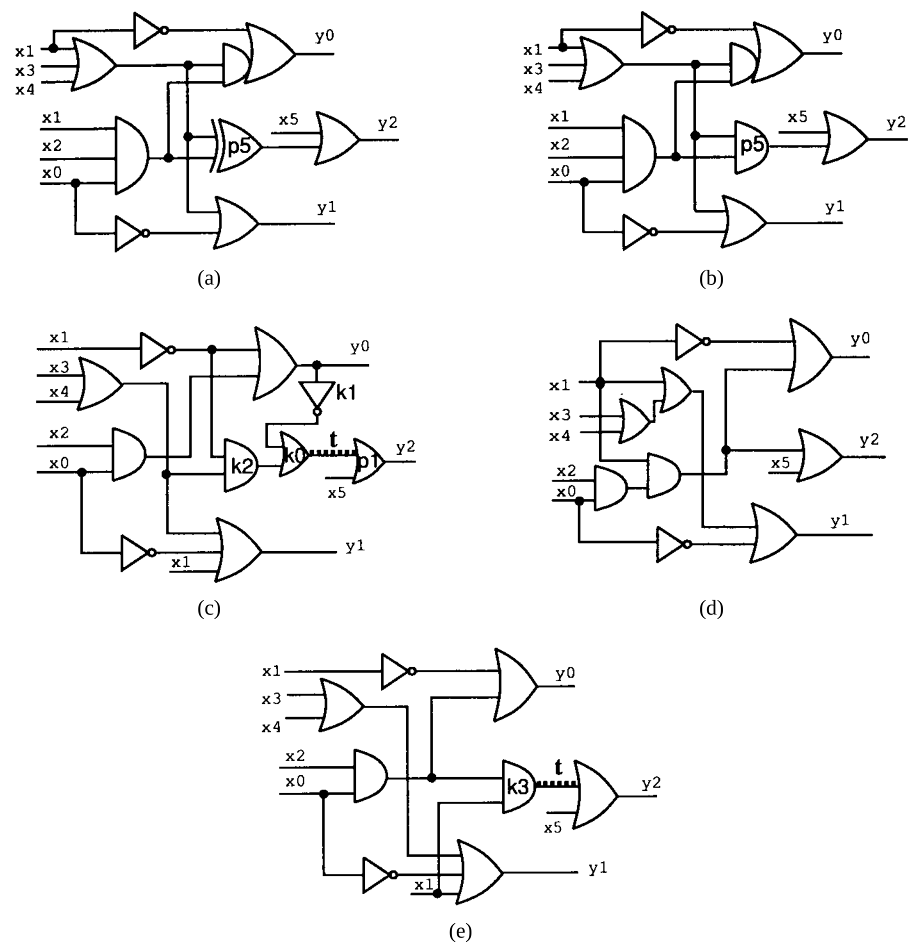


Fig. 1. An example of the EC problem.

change. Section VII summarizes the overall EC algorithm. Section VIII shows the experimental results. Finally, we give conclusions.

II. Basic Definitions

Operations and representation of Boolean functions relevant to our discussion are defined below.

Definition 1: The cofactor f_{x_i} of a Boolean function $f(x_1, \cdots, x_i, \cdots, x_n)$ with respect to x_i is $f(x_1, \cdots, 1, \cdots, x_n)$ and the cofactor $f_{\overline{x}_i}$ with respect to $\overline{x}_i$ is $f(x_1, \cdots, 0, \cdots, x_n)$.

Definition 2: The consensus of $f(x_1, \cdots, x_n)$ with respect to x_i is $\mathcal{C}_{x_i}(f) = f_{x_i} \wedge f_{\overline{x}_i}$, and the smoothing of $f(x_1, \cdots, x_n)$ with respect to x_i is $\mathcal{S}_{x_i}(f) = f_{x_i} \vee f_{\overline{x}_i}$.

The consensus and smoothing operations are logic implementations of the universal and existential quantifiers. The consensus (smoothing) operation can be extended to a set of variables, by iterative application of the consensus (smoothing) on each variable in the set.

Definition 3 [35]: For any two sets S and T, the Cartesian product of S and T, denoted as $S \times T$, is the set of all ordered pairs of the form (s, t) where $s \in S$ and $t \in T$. A relation of two sets S and T is a subset of $S \times T$.

Definition 4 [8], [38], [18]: A Boolean relation $\mathcal{R}$ is a relation of two sets B^m and B^n, where $B = \{0, 1\}$ (i.e., $\mathcal{R} \subseteq B^m \times B^n$). A Boolean relation $\mathcal{R}$ is well-defined if for every $x \in B^m$, there exists a $y \in B^n$ such that $(x, y) \in \mathcal{R}$.

A Boolean relation can be considered as a one-to-many mapping and transformed to an equivalent one-to-one mapping.

Definition 5 [8], [38], [18]: The characteristic function of a Boolean relation $\mathcal{R}$ is a single output Boolean function $R : B^m \times B^n \rightarrow B$ such that $R(x, y) = 1$ if and only if $(x, y) \in \mathcal{R}$.

A Boolean function can be represented in a compact form by means of reduced ordered binary decision diagram (ROBDD) [9]. We summarize this representation as follows. A Boolean decision diagram (BDD) is a directed acyclic graph with nonterminal and terminal vertices. The two terminal vertices, labeled **1** and **0**, represent constant functions 1 and 0, respectively. A nonterminal vertex v is associated with a variable x_i and has two outgoing edges, e_{high} and e_{low}. The edge e_{high} connects v to v_{high} and e_{low} connects v to v_{low}. The function $f(v)$ of the vertex v is expressed as $f(v) = x_i f(v_{\text{high}}) + \overline{x}_i f(v_{\text{low}})$, where $f(v_{\text{high}})$ and $f(v_{\text{low}})$ are defined recursively. Given a vertex, if v_{high} equals to v_{low}, the vertex v is called redundant and can be eliminated by redirecting its incoming edges to v_{low}. When variables associated with vertices follow the same order along any path of BDD, all the redundant vertices are eliminated, and all isomorphic subgraphs are merged, then such a functional graph is called an ROBDD. It is a compact and canonical representation of a Boolean function. A label 1 (0) along an edge denotes e_{high} (e_{low}). An efficient ROBDD package for Boolean function manipulations was developed in [4] and [22], and is used in our implementation.

A combinational network can be described as a direct acyclic graph (DAG) where the vertices correspond to the primary inputs, the primary outputs and the gates of the network. For simplicity, a gate is assumed to have either NOT, AND, OR, NAND or NOR type. The network's inputs $x_1, \cdots, x_n$ are denoted as X, while the outputs $y_1, \cdots, y_m$ as Y. A connection is a pair $\langle s, d\rangle$, where s is the source gate and d is the destination gate. Given a connection $c = \langle s, d\rangle, f_c(X)$ represents a Boolean function of the gate s with respect to the primary inputs X. Given a k-fanin gate d and a gate s which is not a fanin of d, a connection $\langle s, d\rangle$ is referred to as *added* to the gate d, if the resulting gate d' has $k+1$ fanins and the same type as d. The added fanin is s. Similarly, if a gate s is a fanin of d, a connection $\langle s, d\rangle$ is referred to as *removed* from the gate d, if the resulting gate has $k-1$ fanins and the same type as d. The removed fanin is s. A connection in a network is *redundant* if the function of the outputs Y remains unchanged after adding or removing it. To identify a connection's redundancy, the automatic test pattern generation (ATPG) techniques [2], [16], [17], [25], [30], [27], [31], for stuck-at fault testing can be employed. In a DAG, a vertex d_1 is called a *dominator* of another vertex d_2 if every path from d_2 to the primary outputs passes through d_1. Similarly, a vertex d is dominated by a set of vertices T, if every path from d to the primary outputs passes through at least a vertex in T.

Definition 6: Given a connection c, an input pattern is referred to as the observability don't-care (ODC) of c if when it is applied at the primary inputs, no matter what the value of c is, the values at the primary outputs remain the same.

The ODC's of a connection c can be computed as the on-set of the following function:

$$\text{ODC}_c(X) = \prod_{y_i \in Y} \left(y_{i\,c=1}(X) \equiv y_{i\,c=0}(X)\right)$$

where the subscript $c = 1(0)$ is used to denote the operation of setting the value of the connection c to 1 (0) when computing the function y_i.

Definition 7: A connection $c_2 = \langle s_2, d_2\rangle$ is called *substitutable* by another connection $c_1 = \langle s_1, d_1\rangle$ if the function of the primary outputs remain unchanged after adding c_1 and removing c_2. When d_1 and d_2 represent the same gate, c_2 is called *directly substitutable* by c_1, when d_1 and d_2 are two different gates, c_2 is called *indirectly substitutable* by c_1.

Given two connections $c_1 = \langle s_1, d_1\rangle$ and $c_2 = \langle s_2, d_1\rangle$, the requirement of c_2 being directly substitutable by c_1 is given by the condition [24], [28], [29]

$$f_{c_1}(X) \oplus f_{c_2}(X) \subseteq \text{ODC}_{c_2}(X). \tag{1}$$

In [24], $f_{c_1}(X)$ is also called a permissible function of c_2; see Fig. 2 for illustrations.

In general there is no simple way to tell if c_1 can indirectly substitute c_2 except by explicitly checking the equivalence of two networks, one with c_1 and the other with c_2. Given a

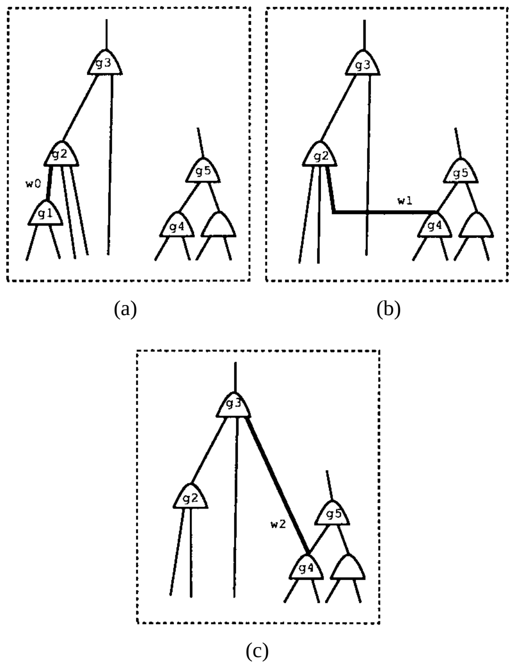


Fig. 2. Illustration of a direct substitution and an indirect substitution. (a) represents the network before substitution. (b) shows the result after the connection $w0$ is directly substituted by $w1$. (c) shows the result after the connection $w2$ indirectly substitutes $w0$.

connection c_2, there are $O(N^2)$ such possible candidates for c_1, where N is the total number of nodes in the network. It would be time-consuming to do an explicit search. To search heuristically for indirectly substitutable connections, an ATPG-based technique used for multilevel logic optimization [13], [12] can be employed. It restricts the search to such a connection which is redundant and whose destination gate is a dominator of the substituted connection's destination gate. After adding a redundant connection c_1 to the network, c_2 can be removed if it becomes redundant. In such a case, c_1 is indirectly substituting c_2. We will discuss an example of indirect substitution in Section VI. For more details, please refer to [13] and [12].

In the following, let S^o be the original specification and C^o the corresponding synthesized network. Suppose S^n is a new specification resulting from engineering changes. Then, the goal of logic synthesis for engineering change is to synthesize the network C^n which realizes S^n and the structural differences between C^o and C^n are minimized by reusing the logic in C^o as much as possible. We refer to S^o (S^n) and C^o (C^n) as the old (new) specification and network, respectively.

III. Previous Work

There have been several papers describing logic synthesis algorithms for engineering change. In [33], C^o and C^n were synthesized independently from S^o and S^n, respectively, and then a post-processing step was performed to identify the correspondence between pins and gates of C^o and C^n. This method is effective when C^o and C^n are structurally similar, but this often is not the case with existing logic synthesis algorithms, which tend to change substantially the network structure.

In [38], the idea was to leave the old network C^o totally unchanged, and to rectify the specification changes by attaching pre-logic and post-logic networks to the primary inputs and outputs of C^o. Boolean relation-based algorithms were developed to derive the functions of the pre- and post-logic. In [15], the application of Boolean unification techniques to solve the same problem (i.e., derive the functions of the pre- and post- logic) were discussed. This approach is useful when changes are made at a later stage of the design process. It can be used to patch an existing layout for functional changes, without going through the whole layout process again. However, the added pre- and post-logic may be too large to be useful. It is not suitable in situations where the internal structure of the old network can be modified.

In [6], a novel approach has been proposed. It explored the structural equivalence between the old and new specifications, and the functional equivalence between the old specification and the existing synthesized network. Using these structural and functional equivalences, [6] established a mapping between signals in the existing network and the ones in the new specification. Then this mapping information guided an ATPG-based direct substitution process to replace signals in the new specification by the signals in the existing network. This method is computationally efficient. However, its effectiveness depends on the amount of the functional equivalence between the old specification and the existing synthesized network.

In the last few years, there have been many papers published on the problem of error diagnosis [14], [19], [26], [23], [36], [37]. The error diagnosis problem can be viewed as an engineering change problem if the appropriate networks are interpreted as follows: C^o which should implement the specification S^n due to error actually implements S^o, where $S^o \neq S^n$. Therefore, the correct specification S^n is now the new specification, and the goal is to modify C^o into another network C^n which implements S^n correctly. Most error diagnosis techniques which have been proposed are based on a single error model implying that the structural difference between C^o and C^n can be rectified by a single gate type change. An extension of [36] to multiple error has been proposed in [37] with limitation of three or four errors.

In this paper, we explore a different domain of the EC problem. We assume that the EC algorithms are allowed to modify the internal logic of the existing network. The amount of modifications has no limitation, as opposed to the one proposed in [37]. Also, we do not rely on the existence of functional equivalence between the signals in the old specification and the existing synthesized network, as used in [6]. In the following sections, we will discuss our approach for solving the engineering change problem. Note that since the amount of design modifications needed to achieve specification changes is really unpredictable, a good approach needs to be able to complete the task even when a large modification is needed. At the same time, it should be able to keep the modifications as small as possible.

IV. Synthesis Algorithms for EC

As discussed in the previous section, the error diagnosis problem has a strong relationship to the EC problem. In error diagnosis, usually a simple single error model is assumed. In EC problems, changes in specification can potentially result in diverse changes in a network, and it is often necessary to make multiple changes in the old network in order to realize the new specification. As a result, a single error model does not suffice. To develop a robust algorithm for EC, we do not assume any error model and we do not limit the number of gates and connections that can be changed. The overall process of our EC algorithm can be divided into two major steps.

1) Identification of candidate signals which can rectify the difference between the old and new specifications, and also derive the new functions of these candidate signals.
2) Synthesis of the new functions by utilizing existing logic of the old network.

To realize the new specification with minimal modifications, Step 1 should identify as few signals as possible. We explore two approaches. The first approach [20] identifies one signal at a time while the second approach [21] identifies multiple signals simultaneously. We will detail them in Section V. After identifying candidate signals, we need to have synthesis algorithms in Step 2 to realize the new functions using as few gates as possible. Several synthesis algorithms are employed to cope with specification changes that require different degrees of modifications in the synthesized network C^o. We will detail them in Section VI.

V. Identification of Candidate Signals

Given an existing network and a new specification, our goal is to modify the network minimally to realize the new specification. To achieve this, the internal signals' logical functions have to be changed appropriately. Therefore, we should have a way to detect and measure the effects of internal signals on the network's functionality.

A. Single Candidate Signal Identification

To identify candidate signals which can rectify the difference between the old and new specifications by changing some internal signals' Boolean functions, an error location technique discussed in [14] and [19] has been extended for our purposes.

In the old network C^o, suppose a signal t carrying a function $f_t^o(X)$ has been selected. We ask does there exist a new function $f_t^n(X)$, which when replacing the function of t yields a network implementing the new specification S^n. The following lemma states the necessary and sufficient condition.

Lemma 1 [14] and [19]: Let $y_i^o(X)$ be the function of the ith output of C^o, and $y_i^o(X,t)$ the same function expressed in terms of X and the internal signal t. Let $y_i^n(X)$ be the function of the corresponding ith output in the new specification S^n. The difference between S^o and S^n can be rectified by replacing $f_t^o(X)$ with $f_t^n(X)$ if and only if the following condition holds:

$$\forall i\, y_i^n(X) \oplus y_i^o\big(X, t = f_t^n(X)\big) = 0 \tag{2}$$

which is equivalent to

$$\forall i\, y_i^n(X) \oplus y_i^o(X, t = 0) \leq f_t^n(X) \leq \overline{y_i^n(X) \oplus y_i^o(X, t = 1)}. \tag{3}$$

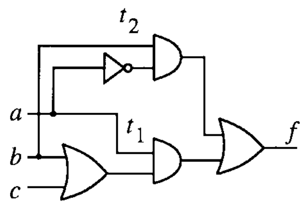


Fig. 3. Example of the observable domain and co-observable domain (COD). The observable domain of the signal t_1 is $ab + ac + \bar{b}c$, and the COD of signals t_1 and t_2 is $\bar{a} + b + c$.

The function $f_t^n(X)$ is an incompletely specified function whose on-set and off-set are as follows:

$$f_t^{\mathrm{on}}(X) = \cup\ y_i^n(X) \oplus y_i^o(X, t = 0), \quad \text{and}$$
$$f_t^{\mathrm{off}}(X) = \cup\ y_i^n(X) \oplus y_i^o(X, t = 1).$$

Equation (2) in Lemma 1 states that to achieve equivalence between $y_i^n(X)$ and $y_i^o(X, t = f_t^n(X))$, their Boolean difference must be zero. To derive (3), we transform (2) applying Shannon expansion and use t to denote $f_t^n(X)$

$$t\big(y_i^n(X) \oplus y_i^o(X, t = 1)\big) + \bar{t}\big(y_i^n(X) \oplus y_i^o(X, t = 0)\big) = 0.$$

This new equation states that $(y_i^n(X) \oplus y_i^o(X, t = 1))$ cannot overlap with t's on-set and $(y_i^n(X) \oplus y_i^o(X, t = 0))$ cannot overlap with t's off-set. Thus, $(y_i^n(X) \oplus y_i^o(X, t = 1))$ must be in the off-set of t while $(y_i^n(X) \oplus y_i^o(X, t = 0))$ must be in the on-set of t. For a function to be realizable, its on-set must not overlap with off-set, thus we have (3). Note that for a given output which is not in the fanout cone of the signal t, (3) reduces to $0 \leq 1$ and it is always true.

Essentially, Lemma 1 states that each output y_i imposes a constraint on t and when all the constraints can be satisfied simultaneously, the signal t is a candidate signal whose function, if changed appropriately to $f_t^n(X)$, can rectify the difference between S^o and S^n. The information about the network's structure can be used to trim down the search space of the candidate signals. For example, the candidate signals must be contained in the intersection of the error outputs' fanin cones (otherwise, it cannot modify the functionality of the error outputs). Also, given two signals t_1 and t_2, if t_1 is dominated by t_2, and t_2 is not a candidate signal, then t_1 cannot be a candidate signal [14]. An ROBDD-based [9] Boolean representation method can implement Lemma 1 efficiently. For example, in the Fig. 1(c), we can identify the connection $t = (k0, p1)$ as the candidate signal, whose new function is

$$f_t^{\mathrm{on}}(X) = x_0x_1x_2\bar{x}_5, \text{ and } f_t^{\mathrm{off}}(X) = (\bar{x}_0 + \bar{x}_1 + \bar{x}_2)\bar{x}_5.$$

Once a candidate signal t has been found, we try to realize its new function. The size of its realization can be very small (e.g., a single gate) or it can be quite large, depending on both the new function and synthesis algorithms. Algorithms for synthesizing the new function are discussed in Section VI.

If many outputs of the old and new specifications are different, the chance of finding a single candidate signal which can rectify all of them becomes small, and thus, we need to consider the possibility of changing the functions of multiple signals. We developed two approaches. The first one is a *divide-and-conquer* strategy. The second approach changes multiple signals *simultaneously*. We will detail them in Sections V-B and V-C.

B. Divide and Conquer

Let PO$^{\mathrm{error}}$ (PO$^{\mathrm{correct}}$) denote the set of outputs which are different (equivalent) in the old and new specifications. When a single candidate signal cannot be identified by Lemma 1, the set PO$^{\mathrm{error}}$ is partitioned into two subgroups (PO$_1^{\mathrm{error}}$ and PO$_2^{\mathrm{error}}$) and each sub-subgroup is rectified sequentially. When searching for a candidate signal to rectify the erroneous outputs in PO$_i^{\mathrm{error}}$, we check Lemma 1 only on the outputs in PO$_i^{\mathrm{error}}$ and PO$^{\mathrm{correct}}$. Thus, the possibility of finding such signal increases. After PO$_i^{\mathrm{error}}$ being rectified, it is appended to PO$^{\mathrm{correct}}$. We have developed a simulated annealing-based approach to find a partition such that the intersection of fanin cones in each subgroup is maximized to increase the chances of finding candidate signals.

A more accurate cost function for partition utilizing the logical information derived from Lemma 1 is possible. For example, given a signal t, we represent the constraint (3) from each output as an incompletely specified function (ISF) in the Boolean space B^X and construct a graph whose vertices represent the ISF's and the edges represent the existence of overlapping between two corresponding ISF's. If the signal t is a candidate signal, then the graph is complete. When it is not, the size of the maximal clique in the graph indicates what is the maximal number of outputs in PO$^{\mathrm{error}}$ that can be rectified by the signal t. Thus, using the logical information derived from Lemma 1, we can perform a more accurate partition. However, in our current implementation, we use structural information for partitioning PO$^{\mathrm{error}}$. Using this divide-and-conquer scheme, we can always finish the search of candidate signals. In the worst case, we have to rectify each output in PO$^{\mathrm{error}}$ individually.

C. Multiple Candidate Signals Identification

To consider changing multiple signals simultaneously for finding candidate signals, Lemma 1 has to be extended using Boolean relations. Based on the concept of ODC's [28], [29] and Boolean relations [8], [10], [18], [38], below we define *observable domain* for a single signal and *co-observable domain* for multiple signals. They completely characterize the effects of internal signals on the network's functionality.

1) Observable Domain and Co-observable Domain: For simplicity, first we assume that the existing network C^o has only one output realizing $f(X)$ where X is the set of primary inputs. The extension to multiple outputs is discussed at the end of this section. The function $f(X)$ is assumed to be different from the new specification $f^s(X)$. The **error minterm set** is defined as a Boolean function $E_f(X) = f(X) \oplus f^s(X)$, where $E_f(X)$ can also be viewed as a set of its on-set minterms.

Definition 8: A set of signals $t_1, \cdots, t_k$ in the network C^o is a *candidate correction set* if there exists a new function $t_i^n(X)$ for each t_i, such that after substituting $t_i^o(X)$ (the original function of t_i with respect to the primary inputs X) by $t_i^n(X), f(X)$ is equal to $f^s(X)$.

Given a signal t_i, the observable domain (OD) of t_i is defined as follows:

Definition 9: Let $f(X, t_i)$ be a representation of $f(X)$ when t_i is treated as an input. The *observable domain* of t_i with respect to f is a set of minterms, $\mathrm{OD}_{t_i}^f(X) = \{X_1 | f(X_1, t_i) \neq 0$ and $f(X_1, t_i) \neq 1, X_1 \in B^X\}$.

Each minterm X_1 in $\mathrm{OD}_{t_i}^f(X)$ when applied to $f(X, t_i)$ makes t_i observable at the output. In other words, under the input minterm X_1, the value t_i controls the value of f. On the other hand, if X_1 is not in $\mathrm{OD}_{t_i}^f(X)$, then the value of t_i has no control on the value of f, and we say that X_1 is an ODC [28], [29] of the signal t_i. Fig. 3 shows a network realizing a function $f(a, b, c)$. To compute the observable domain of the signal t_1, f is re-expressed as $f^{t_1}(a, b, c, t_1) = \overline{a}b + t_1(b + c)$. Based on Definition 8, we find that among all the eight different input patterns, the patterns $\{\overline{a}bc, \overline{a}b\overline{c}\}$ set f^{t_1} to one, and the patterns $\{a\overline{b}\overline{c}, \overline{a}\overline{b}\overline{c}\}$ set f^{t_1} to zero. As a result, on-set of $\mathrm{OD}_{t_1}^f$ is composed of the remaining patterns $\{abc, ab\overline{c}, \overline{a}\overline{b}c, a\overline{b}c\}$ and expressed more conveniently as $ab + ac + \overline{b}c$.

Based on the notion of observable domain, we have the following sufficient condition for a set of signals to be a candidate correction set.

Lemma 2: A set of signals, $t_1, \cdots, t_k$, is a candidate correction set, if $E_f(X) \subseteq \bigcup_{i=1}^k \mathrm{OD}_{t_i}^f(X)$.

Proof: Given an error minterm X_1 in $E_f(X)$, there exists a t_i in the candidate correction set such that $X_1 \in \mathrm{OD}_{t_i}^f(X)$. Since t_i's value controls the output's value under the minterm X_1, if we switch t_i's value for the minterm X_1, while keeping the values of other t_js unchanged, the output is rectified. Each error minterm can be rectified independently. Thus, the lemma follows.

The condition of Lemma 2 is sufficient but not necessary because it does not consider the interaction of multiple signals. We introduce the concept of COD to handle such a case. Given two signals t_i and t_j, their COD is defined as follows.

Definition 10: Let $f(X, t_i, t_j)$ be a representation of $f(X)$ when t_i and t_j are treated as inputs. The *co-observable domain* of t_i and t_j with respect to f is a set of minterms, $\mathrm{COD}_{t_i, t_j}^f(X) = \{X_1 | f(X_1, t_i, t_j) \neq 0$ and $1, X_1 \in B^X\}$.

Since COD considers the interaction of signals, we have the following lemma.

Lemma 3: Given two signals t_i and t_j, their COD is a super set of the sum of OD_i and OD_j, i.e.,

$$\mathrm{OD}_{t_i}^f(X) \cup \mathrm{OD}_{t_j}^f(X) \subseteq \mathrm{COD}_{t_i, t_j}^f(X).$$

To compute the COD of the signals t_1 and t_2 in Fig. 3, first of all, f is re-expressed as $f^{t_1, t_2}(a, b, c, t_1, t_2) = t_2\overline{a} + t_1(b + c)$. We find that among all the eight different input patterns, no patterns set f^{t_1, t_2} to 1, and the pattern in $\{a\overline{b}\overline{c}\}$ sets f^{t_1, t_2} to 0. As a result, $\mathrm{COD}_{t_1, t_2}^f$ is $\overline{a} + b + c$.

We can also extend the definition of COD to more than two signals. Using the concept of COD, we have the following necessary and sufficient condition for a set of signals to be a candidate correction set.

Theorem 1: A set of signals $t_1, \cdots, t_k$ is a candidate correction set, if and only if

$$E_f(X) \subseteq \mathrm{COD}_{t_1, \cdots, t_k}^f(X).$$

Now, we show how to extend the definition of OD and COD to multiple-output networks. Assume the network has outputs $f_1, \cdots, f_m$, and the new specifications are $f_1^s, \cdots, f_m^s$. We construct a single output network $f(X)$ as follows:

$$f(X) = \bigvee_{i=1}^{m} f_i(X) \oplus f_i^s(X).$$

The function $f(X)$ is not equal to the zero function; otherwise the network already implements the new specification. The problem of rectifying multiple outputs becomes the problem of rectifying a single output network $f(X)$, where its new specification $f^s(X)$ is the zero function. Note that we are only allowed to modify the signals in the fanin cones of $f_1, \cdots, f_m$.

2) COD Computation and Search of a Candidate Correction Set: Given a signal t_i or a set of signals $t_1, \cdots, t_k$, we show how to compute $\mathrm{OD}_{t_i}^f$ and $\mathrm{COD}_{t_1, \cdots, t_k}^f$ efficiently using ROBDD representation [9]. Then, we show how to use OD_{t_i} to guide the search for a candidate correction set while using Theorem 1 to verify its candidacy.

Since OD is a special case of COD, we only discuss how to compute the COD. Given a set of signals $t_1, \cdots, t_k$, we first construct ROBDD_f, which is an ROBDD for f in term of X and $t_1, \cdots, t_k$. Then we apply the consensus operator and smoothing operator on ROBDD_f with respect to the variables $t_1, \cdots, t_k$ to obtain ROBDD_f^c and ROBDD_f^s. ROBDD_f^c contains all the minterms which make ROBDD_f evaluated to one and ROBDD_f^s contains all the minterms which make ROBDD_f not zero (i.e., any Boolean function of $t_1, \cdots, t_k$ but not zero function). Since $\mathrm{COD}_{t_1, \cdots, t_k}^f$ is the set of minterms which make ROBDD_f not equivalent to one and zero, the difference between ROBDD_f^s and ROBDD_f^c is the $\mathrm{COD}_{t_1, \cdots, t_k}^f$, (i.e., $\mathrm{COD}_{t_1, \cdots, t_k}^f = \mathrm{ROBDD}_f^s \wedge \neg\, \mathrm{ROBDD}_f^c$).

Since there are many combinations of signals, we have to develop heuristics to guide the search for a candidate correction set. In our approach, we utilize the information extracted from OD to guide the search. The overall searching procedure is shown in Fig. 4. First, the observable domain $\mathrm{OD}_{t_i}^f$ of each signal t_i is computed. Then the signals are sorted according to their coverage of the error minterms E_f, i.e., the number of minterms in $\mathrm{OD}_{t_i}^f \wedge E_f$. Then, the algorithm sequentially adds one signal to the set T until its COD satisfies Theorem 1. Without the size constraint on T, the procedure should be able to find a candidate correction set T which satisfies Theorem 1, but for practical purpose, we set a size constraint k to avoid the memory explosion during ROBDD operations.

3) Determining New Functions for Signals in a Candidate Correction Set: Given a candidate correction set $t_1, \cdots, t_k$, we know there exists a new function $t_i^n(X)$ for each t_i, such that by replacing $t_i^o(X)$ (the original functions of t_i) by $t_i^n(X)$ simultaneously, all the error minterms can be corrected. In this section, we discuss the freedom we have and the methods for deciding the new functions $t_i^n(X)$ of these signals t_i. The

```
search_candidate_correction_set($E_f, k$)
{
    compute_OD_for_each_signal()
    $C$ = sort_signals_by_OD_coverage()
    $T = \emptyset$
    loop {
        $t$ = first_candidate($C$)
        if ($t$ is not dominated by sigals in $T$)
            $T$ = minimal_dominating_set($T \cup \{t\}$)
            $COD$ = compute_COD($T$)
        end
        $C = C - \{t\}$
        if ($E_f \subseteq COD$)
            return($T$)
    } until ( size($T$) > $k$) or ($C$ is $\emptyset$)
    return($\emptyset$)
}
```

Fig. 4. The pseudocode for searching a candidate correction set guided by observable domain. The parameter k is the size constraint of T.

goal is to synthesize the new functions utilizing the existing gates of the network. We developed two heuristic procedures to decide the on-set and off-set of $t_i^n(X)$, for $1 \le i \le k$. Both of them are designed such that the synthesis procedures (shown in Section VI) takes advantage of the freedom of a Boolean relation (shown below). These heuristics are as follows:

a) minimizing the difference between $t_i^o(X)$'s and $t_i^n(X)$'s, or
b) minimizing the number of $t_i^n(X)$'s ROBDD nodes.

As discussed at the end of Section V-C.1, we assume that the multioutput network and its new specification have been merged into a single output network $f(X)$, so that the new specification $f^s(X)$ of the merged network is the zero function. The freedom in determining the on-set and off-set of $t_i^n(X)$ is completely characterized by the function $K = (f(X, t_1, \cdots, t_k) \equiv 0)$ which represents a Boolean relation among $t_1, \cdots, t_k$. In other words, any combinations of $t_i^n(X)$'s are legal if after substituting t_i's by $t_i^n(X)$'s, the resulting function of f becomes a zero function.

Minimizing the difference between $t_i^o(X)$'s and $t_i^n(X)$'s

When minimizing the difference between $t_i^o(X)$ and $t_i^n(X)$, we use the number of minterms in $t_i^o(X) \oplus t_i^n(X)$ as a measure. The problem can be formally stated as follows: given a Boolean relation $f(X, t_1, \cdots, t_k) \equiv 0$, find a realization $t_i^n(X)$ for each t_i such that the total number of different minterms, $\Sigma_{X_1 \in B^X} |\{i | t_i^o(X_1) \ne t_i^n(X_1)\}|$, is minimized. To do this, given an input pattern, we need to know what the old values t_i's are. The characteristic function $\Pi_{i=1}^k (t_i^o \equiv t_i^o(X))$ captures this information, where t_i^o's represent the old values of t_i's. We then construct the following characteristic function:

$$P = \big(f(X, t_1^n, \cdots, t_k^n) \equiv 0\big) \wedge \prod_{i=1}^{k} \big(t_i^o \equiv t_i^o(X)\big).$$

For an input pattern X, the characteristic function P contains the information what new values t_i's can be (i.e., $f(X, t_1^n, \cdots, t_k^n) \equiv 0$) and what the old values t_i's are (i.e., $\Pi_{i=1}^k (t_i^o \equiv t_i^o(X))$). More specifically, given an on-set minterm $(X, t_1^n, t_1^o, \cdots, t_k^n, t_k^o)$ of P, t_i^o's denote the old values of t_i's and t_i^n's denote the new values of t_i's for the input pattern X.

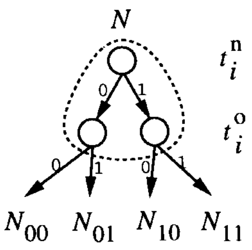


Fig. 5. Example of combining two variables t_i^n and t_i^o into one super variable.

When constructing the ROBDD of P, we order the ROBDD variable t_i^n and the corresponding t_i^o to be next to each other, such that we can implicitly combine two variables t_i^n and t_i^o into one super variable (with four children) as shown in Fig. 5. Then we traverse the ROBDD of P in a depth-first manner and compute the following cost function for each node:

$$\text{Cost}(N) = \text{MIN}\{\text{Cost}(N_{00}),\ \text{Cost}(N_{01}) + 1,\ \text{Cost}(N_{10}) + 1,\ \text{Cost}(N_{11})\}$$

where N is a super node and $N_{00}, N_{01}, N_{10}, N_{11}$ are its four children. The cost of the **1**-terminal and **0**-terminal nodes are set to zero and infinity, respectively.

Using the value Cost(N), we determine which edge among 00, 01, 10, or 11 should be used and which edges should be discarded. For example, given a super node N, if Cost(N_{10})+1 has the smallest values among $\{\text{Cost}(N_{00}),\ \text{Cost}(N_{01}) + 1,\ \text{Cost}(N_{10}) + 1,\ \text{Cost}(N_{11})\}$, then we know that the value of the corresponding t_i^o is zero and by assigning one to t_i^n, $\Sigma_{X_1 \in B^X} |\{i | t_i^o(X_1) \ne t_i^n(X_1)\}|$ for the ROBDD subgraph rooted at N is minimized. To discard the useless edges, we simply redirect them to **0**-terminal node. In the above example, we redirect 00, 01, and 11 edges to **0**-terminal node. The computational complexity of the ROBDD traversal is linear in term of the size of ROBDD. Upon the completion of traversing ROBDD P, we obtain a new ROBDD P'. Using P', we can decide the new function of each $t_i^n(X)$ as follows: $t_i^n(X) = \mathcal{S}_{t_1^o, \ldots, t_k^o, t_1^n, \ldots, t_k^n}(P'_{t_i^n})$.

4) Minimizing the Number of ROBDD Nodes of $t_i^n(X)$'s: The second approach is to find a realization of $t_i^n(X)$'s, such that the number of ROBDD nodes is minimized. As a result, the obtained functions might be easier to implement. We adopt the greedy approach (shown below) reported in [18] and sequentially compute the maximal freedom allowed to implement each $t_i^n(X)$'s

```
minimize_ROBDD_size($f, t_1, \cdots, t_k$)
{
    $K = (f(X, t_1, \cdots, t_k) \equiv 0)$
    for $i$ =1 to $k$
        $K_i$ = smooth$_{t_{i+1}, \cdots, t_k}(K)$
        $K_{i,t_i}$ = cofactor$_{t_i}(K_i)$
        $K_{i,\overline{t_i}}$ = cofactor$_{\overline{t_i}}(K_i)$
        $on = K_{i,t_i} \overline{K_{i,\overline{t_i}}}$
        $off = \overline{K_{i,t_i}} K_{i,\overline{t_i}}$
        $t_i^n(X)$ = bdd_minimize($on$, $off$)
        $K$ = bdd_compose($K, t_i, t_i^n(X)$)
    end
}.
```

For each t_i, by applying the smoothing operator on K with the variables $t_{i+1}, \cdots, t_k$, we have the resulting Boolean relation K_i as a relation of t_i and X. Then, we transformed K_i to an incompletely specified function with on-set as $K_{i,t_i}\overline{K_{i,\overline{t_i}}}$ and off-set $\overline{K_{i,t_i}}K_{i,\overline{t_i}}$. Using this extracted maximal freedom, we apply the *bdd_minimize* algorithm [11], [34] to minimize the number of ROBDD nodes on this incompletely specified function. We then modify the characteristic function K to reflect the fact that the new function $t_i^n(X)$ for t_i has been decided. This is done by applying *bdd_compose* operator which substitutes the ROBDD variable t_i in K by the function $t_i^n(X)$.

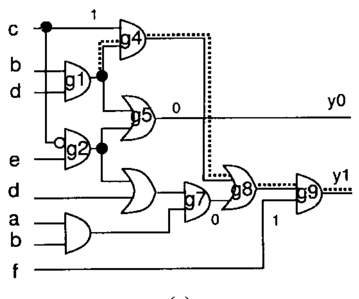
VI. Synthesizing New Functions for Candidate Signals

As discussed in the previous section, an incompletely specified Boolean function can be determined as a new function for a candidate signal t, or a set of Boolean functions $t_1^n(X), \cdots, t_k^n(X)$ can be determined as new functions for a candidate correction set $t_1, \cdots, t_k$ in order to rectify the difference between the old and new specifications. In this section, we discuss algorithms to synthesize these new functions using as few gates as possible, by fully utilizing the existing logic in C^o. We first synthesize the subcircuit $\mathcal{K}$ realizing $t_1^n(X), \cdots, t_k^n(X)$ in terms of the primary inputs. Next this subcircuit is minimized by using existing logic of the old network C^o. Two minimization techniques are employed: one based on permissible functions [24], [28], [29], [5] and the other based on redundancy addition and removal techniques [13], [12].

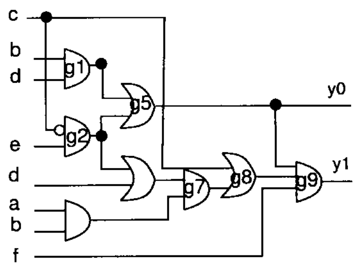
In the permissible functions-based method, for each connection c_2 in the subcircuit $\mathcal{K}$, we search for a connection c_1 in C^o which can directly substitute c_2. If this is possible, the connection c_2 (and frequently also a large portion of the fanin cone of c_2 in $\mathcal{K}$) can be removed. To implement this substitution method, we use an ROBDD-based approach to compute the ODC's of c_2 and then use (1) to check if a connection c_1 in C^o can directly substitute c_2. The ROBDD-based technique has disadvantages for large circuits, while the ATPG-based technique [5], [16], [30] behaves better in terms of performance degradation, because we can set up a time limit for the ATPG process. By doing so, we avoid spending large amounts of time on some hard-to-detect faults.

In the redundancy addition and removal method [13], [12], the effort is to find a connection c_1 which can indirectly substitute a connection c_2 in $\mathcal{K}$. Indirect substitution is more powerful than direct substitution, since the search space is larger. However, if not implemented efficiently, it could be impractical. In [12] and [13], given a connection c_2, the search space of c_1 is restricted to redundant connections and the destination node of c_1 has to be a dominator of c_2. The goal here is to find c_1, which when connected to c_2's dominators, make the c_2 redundant and therefore removable.

As an example, consider the irredundant network in Fig. 6(a). Suppose we are searching for indirectly substitutable connections of $c_2 = \langle g_1, g_4 \rangle$. The idea is to find a connection c_1 whose addition to the network will make the connection c_2 redundant and thus removable. To preserve functionality of the network, we also have to make sure that the connection c_1 is redundant in the original network. We explain the procedures step-by-step as follows.



(a)



(b)

Fig. 6. The example of indirect substitution by redundancy addition and removal.

For $c_2 = \langle g_1, g_4 \rangle$ to be irredundant (i.e., stuck-at-1 testable), the assignments $\{g_1 = 0, c = 1, g_7 = 0, f = 1\}$ must be realizable by at least one input pattern (test pattern). The constraint $\{g_1 = 0\}$ is called a justification constraint, while $\{c = 1, g_7 = 0, f = 1\}$ are propagation constraints. We have the propagation constraint, say $g_7 = 0$, because the gate g_8 is a dominator of c_2 and thus its input g_7 is required to be zero (since g_8 is an OR gate) to allow the value of c_2 propagate through the dotted line in Fig. 6(a). Using these constraints, we conclude that g_5 must be zero (because $c = 1$ implies g_2 zero). Now if we connect g_5 to g_9, it will block the propagation of the value coming from c_2 through the dotted line and thus make the connection c_2 redundant. Next, before removing c_2, we check if the connection $c_1 = \langle g_5, g_9 \rangle$ is redundant. The answer is yes, thus we can add c_1 and remove c_2. The result is shown in Fig. 6(b).

These two kinds of substitution methods are complementary because they explore different substitution domains. Note that indirect substitution may theoretically subsume direct substitution. However, the implications carried out by the ATPG-based redundancy addition and removal method are not meant to be complete in order to avoid spending a lot of time on hard-to-detect faults. As a result, our indirect substitution does not cover completely the search domain of direct substitution. But since the ATPG-based indirect substitution method can handle large networks more efficiently, it is applied before applying the permissible functions-based method.

VII. Implementation of the EC Algorithm

We have implemented the above EC algorithms based on SIS-1.2 [1], [32] and CMU ROBDD packages [22]. The

TABLE I
EXPERIMENTAL RESULTS OF THE EC ALGORITHMS. THE COLUMNS LABELED "%" DENOTE THE AMOUNT OF LOGIC REUSE

					Results of EC algorithms											
					k=1				k=5							
				S^n	M0				M1				M2			
	I/O	S^o	C^o	changes,PO^{error}	P	A,R	%	time	P	A,R	%	time	P	A,R	%	time
z4ml	7 / 4	4	28	1, 1	1	4, 3	85.7	7	1	4, 3	85.7	4	1	4, 3	85.7	4
				2, 2	1,1	6, 3	78.6	10	2	5, 1	82.1	12	2	10, 5	64.3	5
				3, 3	1,1,1	7, 3	75.0	30	1,2	6, 1	78.6	23	1,2	9, 1	67.9	23
				4, 4	1,1,1,1	6, 0	78.6	24	1,1,2	5, 0	78.6	16	1,1,2	5, 0	82.1	17
b9	41 / 21	117	89	1, 2	1, 1	6, 2	93.3	4	2	3, 2	96.6	1	2	4, 2	95.5	1.7
				2, 3	1, 1, 1	6, 3	93.3	14	3	4, 3	95.5	5	3	4, 3	95.5	2
				3, 4	1,1,1,1	7, 3	92.1	33	3,1	4, 3	95.5	20	3,1	7, 3	92.1	20
				4, 6	1,1,1,1,2	10, 7	88.8	123	3,1,2	8, 7	91.0	71	3,1,2	7, 7	92.1	70
frg1	28 / 3	3	115	1, 1	1	3, 0	97.4	67	1	3, 0	97.4	120	1	3, 0	97.4	104
				2, 2	1,1	4, 1	96.5	74	2	6, 2	94.8	149	2	6, 1	94.8	151
				3, 3	1,1,1	7, 2	93.9	71	1,2	6, 2	94.8	170	1,2	7, 3	93.6	165
				4, 3	1,1,1	7, 5	93.9	74	1,2	7, 3	93.9	161	1,2	6, 5	94.8	161
count	35 / 16	47	79	1, 1	1	2, 0	97.5	4	1	2, 0	97.5	3	1	2, 0	97.5	3
				2, 2	1,1	4, 1	94.9	8	2	4, 2	94.9	5	2	4, 2	94.9	3
				3, 2	1,1	7, 2	91.1	10	2	6, 4	92.4	8	2	5, 3	93.7	4
				4, 3	1,1,1	8, 4	89.9	34	1,2	7, 5	91.1	23	1,2	6, 4	92.4	23
x1	51 / 35	28	207	1, 1	1	4, 1	98.1	25	1	4, 1	98.1	17	1	4, 1	98.1	16
				2, 2	1,1	5, 2	97.6	31	1,1	5, 2	97.6	22	1,1	5, 2	97.6	23
				3, 3	1,1,1	7, 3	96.6	79	1,1,1	7, 3	96.6	60	1,1,1	7, 3	96.6	63
				4, 4	1,1,1,1	8, 4	96.1	104	1,1,1,1	8, 4	96.1	77	1,1,1,1	8, 4	96.1	79
x2	10 / 7	12	25	1, 1	1	1, 1	96.0	1	1	1, 1	96.0	0.7	1	1, 1	96.0	0.7
				2, 2	1,1	1, 3	96.0	2	2	3, 5	88.0	2	2	2, 5	92.0	1
				3, 3	1,1,1	5, 4	80.0	13	1,1,1	12, 7	52.0	15	1,1,1	12, 7	52.0	16
				4, 4	1,1,1,1	3, 8	88.0	7	2,1,1	3, 8	88.0	7	2,1,1	2, 8	92.0	6
C880	60 / 26	357	261	1, 1	1	1, 1	99.6	69	1	1, 1	99.6	102	1	1, 1	99.6	106
				2, 2	1,1	1,12	99.6	72	1,1	1,12	99.6	120	1,1	1,12	99.6	124
				3, 3	1,1,1	2,13	99.2	285	1,1,1	2,13	99.2	426	1,1,1	2,13	99.2	442
				4, 4	1,1,1,1	2,12	99.2	251	1,1,1,1	2,12	99.2	437	1,1,1,1	2,12	99.2	447
SrCr	85 / 82	272	339	2, 1	1	20, 5	94.1	1471	1	20, 5	94.1	1471	1	20, 5	94.1	1471

```
$EC(C^o, S^n, PO^{error}, PO^{correct}, k)$
{
    $E_f$ = compute_error_minterm_set($C^o, S^n, PO^{error}$)
    $T$ = search_candidate_correction_set($E_f, k$)
    if $T \neq \emptyset$
        EC_synthesize($C^o, T$)
        append $PO^{error}$ to $PO^{correct}$
    else
        $\{PO_1^{error}, PO_2^{error}\} \longleftarrow$ partition($PO^{error}$)
        $EC(C^o, S^n, PO_1^{error}, PO^{correct})$
        $EC(C^o, S^n, PO_2^{error}, PO^{correct})$
    end
}

EC_synthesize($C^o, T$)
{
    $t_i^n$'s = decide_functions_from_Boolean_relation($C^o, T$)
    replace $t_i$ by $t_i^n$ in $C^o$
    loop {
        perform_indirect_substitution($C^o$)
        perform_direct_substitution($C^o$)
    } until (no further improvement)
```

Fig. 7. The pseudocode of EC algorithm.

overall EC algorithms are shown in Fig. 7. Before calling the EC algorithm, an ROBDD-based verification tool [1] is used to find PO^{error} and $\text{PO}^{\text{correct}}$.

The procedure `search_candidate_correction_set` performs the algorithm shown in Fig. 4 to find the candidate correction set T, such that there exists a new function $t_i^n(X)$ for each signal t_i in T, and the result of substituting t_i by $t_i^n(X)$ in the existing network will realize the new specification.

Due to the imposed size constraint k, the algorithm `search_candidate_correction_set` might fail to find the appropriate set. If this happens, PO^{error} is split into two subsets $\text{PO}_1^{\text{error}}$ and $\text{PO}_2^{\text{error}}$, and the EC algorithm rectifies them separately. On the other hand, if T is not empty, the procedure `EC_synthesize` (described in Section VI), synthesizes the functions $t_i^n(X)$'s by utilizing the existing gates of the network as much as possible. We apply the indirect and direct substitution methods iteratively to achieve a better utilization of the existing logic.

VIII. THE EXPERIMENT

In this section, we show the results of applying the EC algorithms described in the previous sections. Several combinational benchmark circuits from MCNC91 and one example ($SrCr$) from Fujitsu Laboratories Ltd. are included in our test suite. The circuit $SrCr$ (part of an ATM router chip) was originally given in VHDL by the designer, and later on, the specification was modified by creating a new internal signal. This $SrCr$ example was a hierarchical design and contained flip-flops. For our experiment, we flattened the design and extracted the combinational portion of the circuit.

For MCNC91 benchmarks, it was assumed that each of them represents the original specification S^o. To obtain C^o, we optimized S^o by running the `script.rugged` script and then performed technology decomposition (`tech_decomp -a 4 -o 4`) in SIS [1]. The resulting number of gates are shown in the fourth column of Table I. Beside the difference in the number of gates, the networks' topology between S^o and C^o are quite different also. The reason we used technology decomposition instead of technology mapping is because our current implementation allows only primitive gates (NOT, AND, OR, NAND, or NOR) in the network. Furthermore, to obtain S^n, we randomly modified S^o by changing the functions of the internal gates. For a complex gate (represented as a SOP form in BLIF format [1]), we arbitrarily modified its cubes. For a simple gate, i.e., an AND gate, we changed it to OR gate, etc. The fifth column of Table I shows the number of such changes and the number of primary outputs affected.

Then, given C^o and S^n, we applied the EC algorithms to generate C^n on a SUN SPARC 5 machine with 128-megabytes memory. We tested both algorithms described in Section V with the constraints $k = 1$ and 5 on the size of the candidate correction set T. For $k = 5$, we also tested the two different heuristics for determining new functions as described in Section V-C3. The result of using first heuristic (i.e., minimizing the total number of minterms changed) is shown in the column labeled **M1**, while the other (i.e., minimizing the number of ROBDD nodes) is shown in the column labeled **M2**.

We report the number of added gates (A), removed gates (R), and computation time (in seconds). They measure quality of the EC algorithms. The columns labeled P shows the results of recursively partitioning erroneous outputs. For example, (3, 1, 2) means that during the search for the candidate correction set, the six erroneous outputs in $\mathrm{PO}^{\mathrm{error}}$ were partitioned into three subgroups. The first subgroup has three erroneous outputs, while the other two have one and two, respectively. Each of them was rectified separately. As expected, **M1** and **M2** partition $\mathrm{PO}^{\mathrm{error}}$ into fewer number of subgroups. This is because they explore the chances of modifying multiple signals simultaneously. Also, the more we change the specification, the more added and removed gates are required. The execution time also increases.

In terms of the EC quality, for the example b9, **M1** and **M2** perform better than **M0**, while for x2, the results from **M0** are better. On the average, the results for three different heuristics are quite competitive. This is probably because of the inaccuracy of the cost function used in the identification process. In other words, the final results of EC algorithms depend on the synthesis methods used to synthesize the new functions. During the process of determining those new functions, it is difficult to use a cost function which accurately represents the final synthesis results.

IX. Conclusions

We proposed synthesis algorithms to automate the engineering changes. To realize the changes of the specifications, we developed algorithms which modify the existing synthesized network such that a substantial portion of the earlier engineering effort can be preserved.

Compared to the existing techniques [38], [15], we explore a different domain of the EC problem. We assume that the EC algorithms are allowed to modify the internal logic of the existing network. Besides, we do not limit the amount of modifications of the existing synthesized network, as opposed to the one proposed in [37]. Also, we do not rely on the existence of functional equivalence between the signals in the old specification and in the synthesized network as it is in [6].

Our EC algorithm can be divided into two steps. The first step identifies candidate signals and the corresponding new functions. Having the functions of these candidate signals replaced by the new functions, the differences between the old and new specifications are rectified. The next step synthesizes these new functions by utilizing gates of the existing synthesized network. Identifying which set of signals to change is a difficult problem. The search is guided by the observable and COD's of the signals. Currently, identification and synthesis processes are performed independently. Future improvements will consider new algorithms integrating these two steps closely. As a result, we will be able to obtain a more accurate estimate of the final changes in the EC algorithm.

References

[1] SIS: A system for sequential circuit synthesis. University of California, Berkeley, CA, Rep. M92/41, 1992.
[2] D. M. Armstrong, "On finding a nearly minimal set of fault detection tests for combinational logic nets," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 66–73, Feb. 1966.
[3] K. A. Bartlett *et al.*, "Multilevel logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 723–740, 1988.
[4] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 40–45.
[5] B. Brand, "Verification of large synthesized designs," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1993, pp. 534–537.
[6] D. Brand, A. Drumm, S. Kundu, and P. Narain, "Incremental synthesis," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1994, pp. 14–18.
[7] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. Wang, "A multiple-level logic optimization system," *IEEE Trans. Computer-aided Design*, vol. CAD-6, pp. 1062–1081, 1987.
[8] R. K. Brayton and F. Somenzi, "An exact minimizer for Boolean relations," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 316–319.
[9] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 667–691, 1986.
[10] E. Cerny and M. A. Marin, "An approach to unified methodology of combinational switching circuits," *IEEE Trans. Computers*, vol. 27, pp. 745–756, 1977.
[11] S.-C. Chang, D. I. Cheng, and M. Marek-Sadowska, "BDD representation of incompletely specified functions," in *Proc. Eur. Conf. Design Automation*, 1994, pp. 620–624.
[12] S.-C. Chang and M. Marek-Sadowska, "Perturb and simplify: Multi-level Boolean network optimizer," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1994, pp. 2–5.
[13] K.-T. Cheng and L. A. Entrena, "Multi-level logic optimization by redundancy addition and removal," in *Proc. Eur. Conf. Design Automation*, 1993, pp. 373–377.
[14] P.-Y. Chung, Y.-M. Wang, and I. N. Hajj, "Diagnosis and correction of logic design errors in digital circuits," in *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 503–508.
[15] M. Fujita, Y. Tamiya, Y. Kukimoto, and K.-C. Chen, "Application of Boolean unification to combinational logic synthesis," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 510–513.
[16] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Computers*, vol. C-30, pp. 215–222, Mar. 1981.
[17] W. H. Kautz, "Fault testing and diagnosis in combinational digital circuits," *IEEE Trans. Computers*, vol. C-17, pp. 352–366, Apr. 1968.

[18] Y. Kukimoto and M. Fujita, "Rectification method for lookup-table type FPGA's, in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1992, pp. 54–61.

[19] H.-T. Liaw, J.-H. Tsaih, and C.-S. Lin, "Efficient automatic diagnosis of digital circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 464–467.

[20] C.-C. Lin, K.-C. Chen, S.-H. Chang, M. Marek-Sadowska, and K.-T. Cheng, "Logic synthesis for engineering change," in *Proc. ACM-IEEE Design Automation Conf.*, 1995, pp. 647–652.

[21] C.-C. Lin, K.-C. Chen, D. I. Cheng, and M. Marek-Sadowska, "Logic synthesis and rectification for engineering change," in *Proc. Asian South Pacific Design Automation Conf.*, 1995, pp. 301–309.

[22] D. Long, "BDD—A binary decision diagram package," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep,, 1993.

[23] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of design errors with PRIAM," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989.

[24] S. Muroga, Y. Kambayashi, H.-C. Lai, and J. N. Culliney, "The transduction method—Design of logic networks based on permissible functions," *IEEE Trans. Computers*, 1989, pp. 1404–1424.

[25] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. Computers*, vol. C-25, pp. 630–636, June 1976.

[26] I. Pomeranz and S. M. Reddy, "On error correction in macro-based circuits," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1994, pp. 568–575.

[27] J. P. Roth *et al.*, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Trans. Computers*, vol. C-16, pp. 567–579, Oct. 1967.

[28] A. Saldanha, A. Wang, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-level logic simplification using don't care and filters," in *Proc. ACM/IEEE Design Automation Conf.*, 1989, pp. 277–282.

[29] H. Savoj and R. K. Brayton, "The use of observability and external don't cares for the simplification of multi-level networks," in *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 297–301.

[30] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," in *Proc. Int. Test Conf.*, 1987, pp. 1016–1026.

[31] F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson, "Analyzing errors with the Boolean difference," *IEEE Trans. Computers*, vol. C-17, pp. 676–683, 1968.

[32] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proc. IEEE Int. Conf. Computer Design*, 1992, pp. 328–333.

[33] T. Shinsha, T. Kubo, Y. Sakataya, and K. Ishihara, "Incremental logic synthesis through gate logic structure identification," in *Proc. ACM/IEEE Design Automation Conf.*, 1986, pp. 391–397.

[34] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic minimization of BDD's using don't cares," in *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 225–231.

[35] H. S. Stone, *Discrete Mathematical Structures and Their Applications*, SRA Computer Science Series. Chicago, IL: Science Res. Associates, 1973.

[36] M. Tomita, H.-H. Jiang, T. Yamamoto, and Y. Hayashi, "An algorithm for locating logic design errors," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 468–471.

[37] M. Tomita, T. Yamamoto, F. Sumikawa, and K. Hirano, "Rectification of multiple logic design errors in multiple output circuits," in *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 212–217.

[38] Y. Watanabe and R. K. Brayton, "Incremental synthesis for engineering changes," in *Proc. IEEE Int. Conf. Computer Design*, 1991, pp. 40–43.

Chih-Chang Lin (M'96) received the B.S. degree in computer science and information engineering from National Taiwan University, Taiwan, in 1987, the M.S. degree in computer science from University of California at Santa Barbara in 1991, and the Ph.D degree in electrical and computer engineering, University of California at Santa Barbara in 1996.

He was a Software Engineer of Mentor Graphics Corp, San Jose, CA, developing logic synthesis and test synthesis tool. He is currently the Engineer Director of Verplex Systems, Inc., Santa Clara. CA, developing formal verification tools. His research interest includes logic synthesis and circuit testing.

Kuang-Chien Chen (S'87–M'87) received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1985, and the M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign in 1987 and 1990, respectively.

He is currently with Verplex Systems, Inc., Santa Clara. CA, where he is engaged in the research and development of advanced verification tools. Prior to joining Verplex, he held research and management positions at Fujitsu Laboratories of America, Inc., and has been working in the area of formal verification, synthesis, and logic optimization. His main interest now is in building automatic verification tools based on equivalence checking, model checking, BDD, and ATPG-related techniques.

Dr. Chen is a member of ACM.

Malgorzata Marek-Sadowska (M'87–SM'95–F'97)received the M.S. degree in applied mathematics (1971) and the Ph.D. degree in electrical engineering (1976) from Politechnika Warszawska (Technical University of Warsaw), Warsaw, Poland.

From 1976 to 1982, she was Assistant Professor at the Institute of Electron Technology at the Technical University of Warsaw. She was invited to be a Visiting Professor in the Electrical Engineering Department of the University of California at Berkeley from 1979 to 1980. She became a Research Engineer at the Electronics Research Laboratory in 1979 and continued there until 1990, when she joined the Department of Electrical and Computer Engineering at the University of California, Santa Barbara, as a Professor. Her research interests are in the area of computer-aided design, with an emphasis on layout and logic synthesis of VLSI circuits and systems.

Professor Marek-Sadowska has been a member of numerous technical committees, including the Technical Committee of the International Conference on Computer Aided Design, the Technical Committee of the Design Automation Conference, and the Technical Committee International Workshop on Placement and Routing. From 1989 to 1993, she was Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and Editor-In-Chief from 1993 to 1995. She has been Associate Editor for the *Journal of Circuits, Systems and Computers* since 1990 and serves as a reviewer for numerous technical journals.