Logic Rectification and Synthesis for Engineering Change

Chih-chang Lin  
University of California, Santa Barbara

David Ihsin Cheng  
University of California, Santa Barbara

Kuang-Chien Chen  
Fujitsu Laboratories of America, INC.

Malgorzata Marek-Sadowska  
University of California, Santa Barbara

Abstract—In the process of VLSI design, specifications are often changed. It is desirable that such changes will not lead to a very different design, so that a large part of engineering effort can be preserved. We treat this problem as a combination of multiple-error diagnosis and logic minimization problems. Given a new specification and an existing synthesized logic network, our algorithms modify the existing network minimally such that the new specification can be realized. In this paper, a new algorithm is developed to identify multiple candidate signals simultaneously minimized such that the new specification can be realized. This specification will not lead to a very different design and is usually called the engineering change (EC) problem.

As automatic synthesis becomes popular, the issue of how to handle engineering changes gains even more importance. Since synthesis tools usually perform global transformations (e.g., sharing of modules) to achieve good quality results, small and local changes in the specification could have global effects and produce a very different network. Realizing this fact, designers usually have to manually modify the synthesized network to realize changes in the specification. Such practice not only increases the chance of introducing inconsistencies between the higher-level specification (e.g., VHDL) and the final network, but also is an error-prone process that 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.

Example 1 Figure 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 illustration purpose, we simply take the network in Figure 1.(a) as the initial specification. After applying logic transformation and optimization procedures [1, 2, 3] on the network in Figure 1.(a), the resulting network is shown in Figure 1.(c).

Now suppose the specification in Figure 1.(a) has been slightly modified by changing p3 from an XOR gate to an AND gate, as shown in Figure 1.(b). After applying the same synthesis procedures as before, we obtain a network shown in Figure 1.(d). As we can see, although the change in specification arises from a local modification, general synthesis procedures do not localize such a change and the networks in Figure 1.(c) and Figure 1.(d) are quite different (i.e. the number of gates is changed from 10 to 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 Figure 1.(c) directly, so that we can obtain a network similar to Figure 1.(c), yet realizing the new specification in Figure 1.(b). Such manual EC technique, however, cannot be easily applied in this case. This is because after we have applied transformation and optimization procedures, the signal corresponding to gate p5 in Figure 1.(a) is no longer available in the optimized network Figure 1.(c). Therefore, although we know the change arises from the modification of p5, it is difficult to tell in Figure 1.(c) where and how the modifications should be done.

On the other hand, a good synthesis procedure which considers engineering changes will be able to modify the network in Figure 1.(c) minimally, such that the resulting network is functionally equivalent to the specification in Figure 1.(b). In later sections, we will discuss such EC algorithms. By applying these algorithms on the network in Figure 1.(c), the network in Figure 1.(e) is obtained, differing from the network in Figure 1.(c) in that the gates, k_0, k_1 and k_2, have been removed and a gate k_3

I. Introduction

In a typical VLSI design process, specifications are often changed in order to correct design errors, or to meet certain design constrains such as area, timing and power consumption. Since a lot of engineering effort may already have been invested (e.g., the layout of a chip may have been obtained), it is desirable that such changes in specification will not lead to a very different design and a large part of the engineering effort can be preserved. This is usually called the engineering change (EC) problem.

As automatic synthesis becomes popular, the issue of how to handle engineering changes gains even more importance. Since synthesis tools usually perform global transformations (e.g., sharing of modules) to achieve good quality results, small and local changes in the specification could have global effects and produce a very different network. Realizing this fact, designers usually have to manually modify the synthesized network to realize changes in the specification. Such practice not only increases the chance of introducing inconsistencies between the higher-level specification (e.g., VHDL) and the final network, but also is an error-prone process that 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.

Example 1 Figure 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 illustration purpose, we simply take the network in Figure 1.(a) as the initial specification. After applying logic transformation and optimization procedures [1, 2, 3] on the network in Figure 1.(a), the resulting network is shown in Figure 1.(c).

Now suppose the specification in Figure 1.(a) has been slightly modified by changing p3 from an XOR gate to an AND gate, as shown in Figure 1.(b). After applying the same synthesis procedures as before, we obtain a network shown in Figure 1.(d). As we can see, although the change in specification arises from a local modification, general synthesis procedures do not localize such a change and the networks in Figure 1.(c) and Figure 1.(d) are quite different (i.e. the number of gates is changed from 10 to 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 Figure 1.(c) directly, so that we can obtain a network similar to Figure 1.(c), yet realizing the new specification in Figure 1.(b). Such manual EC technique, however, cannot be easily applied in this case. This is because after we have applied transformation and optimization procedures, the signal corresponding to gate p5 in Figure 1.(a) is no longer available in the optimized network Figure 1.(c). Therefore, although we know the change arises from the modification of p5, it is difficult to tell in Figure 1.(c) where and how the modifications should be done.

On the other hand, a good synthesis procedure which considers engineering changes will be able to modify the network in Figure 1.(c) minimally, such that the resulting network is functionally equivalent to the specification in Figure 1.(b). In later sections, we will discuss such EC algorithms. By applying these algorithms on the network in Figure 1.(c), the network in Figure 1.(e) is obtained, differing from the network in Figure 1.(c) in that the gates, k_0, k_1 and k_2, have been removed and a gate k_3
II. Terminology and previous work

Let $S^o$ be the original specification and $C^o$ a corresponding synthesized logic network. Suppose $S^o$ is a new specification resulting from engineering changes. The goal of logic synthesis for engineering change is to synthesize a network $C^o$ such that it realizes $S^o$ and the structural differences between $C^o$ and $C^o$ are minimized. In the remainder of the paper, we shall simply refer to $S^o$ ($S^o$) and $C^o$ ($C^o$) as the old (new) specification and network, respectively.

There have been several papers on logic synthesis algorithms for engineering change. In [4], $C^o$ and $C^o$ are synthesized independently from $S^o$ and $S^o$, respectively, and then a post-processing step is performed to identify the correspondence between pins and gates of $C^o$ and $C^o$. This method is effective when $C^o$ and $C^o$ are structurally similar, but this is often not the case with existing logic synthesis algorithms which tend to change substantially the structure of the networks.

In [5], 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. It has been shown that any new specification can be realized in this way if both pre- and post-logic networks are allowed. In [6], 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 when it may be desirable to keep the old design unchanged. For example, it can be used to patch an existing layout for function changes, without going through the whole layout process again. However, the pre- and post-logic added may still be too large to be useful, and it is not suitable in situations where the internal structure of the old network can be modified.

In [7], a novel approach is proposed which explores 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 equivalence, [7] establishes a mapping between the signals in the existing network and the ones in the new specification. Then, this mapping information is used to guide an ATPG–based logic substitution process. 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 much work on the problem of error diagnosis [8, 9, 10, 11, 12]. The error diagnosis problem can be viewed as an engineering change problem if the appropriate networks are interpreted as follows. $C^o$ is supposed to implement the specification $S^o$ and contains an implementation error such that it actually

Fig. 1. An example of the EC problem.

has been added. Note that the removal of gates is probably beneficial for layout and timing, and the change made here is local. □

Note that changes made at high levels 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 were made resulting in changes in the number of states and state transitions. Then, during synthesis, state encoding different from the original encoding 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. In this paper, we will concentrate on the core problem of the engineering change, i.e., handling functional specification changes for combinational networks.

Remainder of this paper is organized as follows. Section II provides appropriate background, definitions and reviews related work. Section III, Section IV and Section V discusses our approach of logic synthesis for engineering change. Section VI summarizes the overall EC algorithm. Section VII shows the experimental results. Finally we give conclusions and discuss the plan for future work.
implements $S^o$ and $S^o \neq S^n$. Therefore, the correct specification $S^n$ is now the new specification, and our goal is to modify $C^o$ into another network $C^n$ which implements $S^n$ correctly. In [8, 9, 10, 11], error correction techniques were proposed based on a single-error model which assumed that the structural difference between $C^o$ and $C^n$ can be characterized as a single gate type change or a single wire mis-connection.

The error diagnosis problem has a strong relationship to the EC problem. However, 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 a new specification. As a result, single-error model does not suffice. An extension of single-error model for the EC problem is shown in [13] which modifies multiple signals sequentially in the following steps:

1) Identify all erroneous outputs, $PO_{err}$.
2) Identify a single candidate signal which can rectify as many erroneous outputs as possible, and also derive the new function of this signal.
3) Synthesize the new function by utilizing existing logic of the old network.
4) Remove the corrected outputs from $PO_{err}$. If $PO_{err}$ is not empty, then loop back to Step 2.

To realize the new specification with minimal modifications, the above process should identify as few signals as possible, and the synthesis procedures in Step 3 have to be powerful enough so that the new functions can be realized using as few gates as possible.

In this paper, following the framework of [13], we develop a new algorithm which identifies multiple signals simultaneously for Step 2 based on the concept of co-observable domain. As a result, the new algorithm can search a larger solution space and the experimental results are competitive.

III. Observable and co-observable domain

In the EC problem, given an existing network and a new specification, we are interested in modifying the network minimally to realize the new specification. It is the modification of internal signals’ logical functions that allows the new specification can be realized. Therefore, we should have a way to detect and measure the effects of internal signals on the network’s functionality. Based on the concept of observability don’t care [1] and Boolean relation [14, 15, 16], we define observable domain for a single signal and co-observable domain for multiple signals, respectively. They completely characterize the effects of internal signals on the network’s functionality.

For simplicity, in the following Section A and Section B, we assume the existing network $C^o$ has a single output realizing $f(X)$ where $X$ is the set of primary inputs. The extension to multiple outputs is discussed in Section C. $f(X)$ is assumed to be different from the new specification $f^n(X)$ and the error minterm set is defined as

$$E_f(X) = f(X) \oplus f^n(X),$$

where $E_f(X)$ can be viewed as a set of minterms or a Boolean function.

**Definition 1** A set of signals $t_1, \ldots, t_k$ in the network $C^o$ is a candidate location set if there exists a new function $f^n_t(X)$ for each $t_i$, such that after substituting $t_i(X)$ (the original function of $t_i$ with respect to the primary inputs $X$) by $f^n_t(X)$, $f(X)$ is equal to $f^n(X)$.

**A. Observable domain**

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

**Definition 2** Let $f^i_1(X, t_i)$ be a representation of $f(X)$ by treating $t_i$ as an input. The observable domain of $t_i$ with respect to $f$ is a set of minterms, where

$$OD^i_t(X) = \{X_1 \mid f^i_1(X_1, t_i) \neq 0 \text{ or } 1, \text{ for } X_1 \in 2^X \}.$$

For a given minterm $X_1$ in $OD^i_t(X)$, when its applied to $f^i_1(X, t_i)$, it will make $t_i$ observable at the output. In other words, under the input minterm $X_1$, the value $t_i$ controls the value of $f^i_1$. On the other hand, if $X_1$ is not in $OD^i_t(X)$, then the value of $t_i$ has no control on the value of $f^i_1$ (i.e., $X_1$ is an observability don’t care of the signal $t_i$ [1]).

**Example 2** Figure 2 shows a network with the output function $f(a, b, c)$. To compute the observable domain of the signal $t_1$, first of all, $f$ is re-expressed as $f^i_1(a, b, c, t_1) = \bar{a}b + t_1(b + c)$ by considering $t_1$ as an extra input. Then, based on the above definition, we find that among the eight different minterms, the minterms $\{\bar{a}bc, \bar{a}b\}$ make $f^i_1$ become 1, while the minterms $\{a\bar{c}, \bar{a}b\}$ make $f^i_1$ become 0. As a result, $OD_{t_1}$ is $\{abc, a\bar{c}, \bar{a}b, a\bar{b}\}$ or we can express it more conveniently as $ab + ac + \bar{b}c$. $\square$

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

**Lemma 1** A set of signals, $t_1, \ldots, t_k$, is a candidate location set, if

$$E_f(X) \subseteq \bigcup_{i=1}^{k} OD^i_t(X).$$

**Proof.** Given an error minterm in $E_f(X)$, we can find at least a $t_i$ such that if we switch $t_i(X)$’s value for that minterm, then the output is rectified for that error minterm. Since each error minterm can be rectified independently, the lemma follows. □
The observable domain of the signal $t_1$ is $ab + ac + c$ and the co-observable domain of signal $t_1$ and $t_2$ is $a + b + c$.

**B. Co-observable domain**

The condition of Lemma 1 is sufficient but not necessary because it does not consider the interaction of multiple signals. We introduce the concept of co-observable domain to handle such a case. Given two signals $t_i$ and $t_j$, their co-observable domain (COD) is defined as follows:

**Definition 3** Let $f^{t_i,t_j}(X, t_i, t_j)$ be a representation of $f(X)$ by treating $t_i$ and $t_j$ as inputs. The co-observable domain of $t_i$ and $t_j$ with respect to $f$, is

$$\text{COD}_{t_i,t_j}(X) = \{ X_1 | f^{t_i,t_j}(X_1, t_i, t_j) \neq 0 \text{ or } 1, \text{ for } X_1 \in 2^X \}.$$  

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

**Lemma 2** 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.,

$$OD_i(X) \cup OD_j(X) \subseteq \text{COD}_{t_i,t_j}(X).$$

**Example 3** We use Figure 2 again to show the concept of co-observable domain. To compute the co-observable domain of the signals $t_1$ and $t_2$, first of all, $f$ is re-expressed as $f^{t_1,t_2}(a, b, c, t_1, t_2) = t_2 \overline{a} + t_1(b + c)$ by considering $t_1$ and $t_2$ as two extra inputs. Then, we find that among all the 8 different minterms, no minterms can make $f^{t_1,t_2}$ become 1, while the minterm in $\{\overline{a} b \}$ makes $f^{t_1,t_2}$ become 0. As a result, $\text{COD}_{t_1,t_2}$ is $\overline{a} + b + c$. □

We can also extend the definition of co-observable domain 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 location set.

**Theorem 1** A set of signals $t_1, \ldots, t_k$ is a candidate location set, iff

$$E_f(X) \subseteq \text{COD}_{t_1, \ldots, t_k}(X).$$

**C. Handling multiple outputs**

Here, we show how to extend the definition of OD and COD to multiple-output networks. Assume the network has outputs $f_1, \ldots, f_m$, and the new specifications are $f_{1}^*, \ldots, f_{m}^*$. We construct a single output network $Z(X)$ as follows:

$$Z(X) = \bigvee_{i=1}^{m} f_i(X) \oplus f_i^*(X),$$

where $Z(X)$ is not equal to the zero function (otherwise the network has already implemented the new specification). Then, the problem of rectifying multiple outputs becomes the problem of rectifying a single output network $Z(X)$, where its new specification $Z^*(X)$ is the zero function. Note that here we are only allowed to modify the signals in the fan-in cones of $f_1, \ldots, f_m$.

**IV. COD Computation and search of candidate location set**

In this section, given a signal $t_i$ or a set of signals $t_1, \ldots, t_k$, we show how to compute $OD_i$ and $COD_{t_1, \ldots, t_k}$ efficiently based on BDD manipulations [17]. Then, we show how to use $OD_i$ effectively to guide the search for a candidate location set while using Theorem 1 to verify its candidacy.

**A. COD Computation**

Since $OD$ is a special case of COD, we only discuss how to compute $COD$. Given a set of signals $t_1, \ldots, t_k$, we first construct $BDD_f$, which is a BDD for $f$ in term of $X$ and $t_1, \ldots, t_k$. Then we apply the consensus operator and smoothing operator on $BDD_f$ with respect to the BDD variables $t_1, \ldots, t_k$ to obtain $BDD_f^*$ and $BDD_f^\perp$ separately. $BDD_f$ contains all the minterms which make $BDD_f$ 1 and $BDD_f^\perp$ contains all the minterms which make $BDD_f^\perp$ not equivalent to 0. Since COD is the set of minterms which make $BDD_f^\perp$ not equivalent to 1 or 0, the difference between $BDD_f^*$ and $BDD_f^\perp$ is COD. The pseudo code for computing co-observable domain is as follows:

**COD($f, X, t_1, \ldots, t_k$)**

$$\{ \text{BDD}_f = \text{buildBdd}(f, X, t_1, \ldots, t_k) \}
\text{BDD}_{\text{cod}} = \text{smooth}_{t_1, \ldots, t_k}(\text{BDD}_f) - \text{consensus}_{t_1, \ldots, t_k}(\text{BDD}_f)$$

**B. Search of a candidate location set**

Since there are huge combinations of signals, we have to develop heuristics to guide the search for a candidate...
search_candidate_location_set( \( E_f, k \) )
{
    compute_OD_foreach_signal()
    \( T = \emptyset \)
    \( C = \) sorted_candidate_signals_by_OD_coverage
    loop
        \( t = \) first_candidate(\( C \))
        if (\( t \) is not dominated by signals in \( T \))
            \( T = \) minimal_dominating_set(\( T + \{ t \} \))
            \( COD = \) compute_COD(\( T \))
        end
        \( C = C - \{ t \} \)
        if (\( E_f \subseteq COD \))
            return(\( T \))
        \}
    until (size_of(\( T \)) > \( k \)) or (\( C = \emptyset \))
    return(\( \emptyset \))
}

Fig. 3. The pseudo code for searching a candidate location set guided by observable domain. The parameter \( k \) is the size constraint of \( T \).

location set. This can also be considered as a multiple-error diagnosis problem [12]. In our approach, we utilize the information extracted from OD to guide the search. Moreover, to avoid redundant computation, topological information can be utilized. For example, given a set of signals \( t_1, \ldots, t_k \), if \( COD_t_1,\ldots,t_k \) is not a candidate location set, then any combinations of signals in the fan-in cones of \( t_1, \ldots, t_k \) are not either.

The overall searching procedure is shown in Figure 3. First, the observable domain of each signal \( t_i \) \((OD_{t_i})\) is computed and stored. Then the signals are sorted according to their coverage of \( E_f \), i.e., the number of minterms in \( OD_{t_i} \land E_f \). After that, the algorithm sequentially adds one signal to the set \( T \) until its \( COD \) satisfies Theorem 1. Theoretically, without size constraint, given enough time, the algorithm should be able to find a candidate location set \( T \) which satisfied Theorem 1, but for practical purpose, we set a size constraint \( k \) as shown in Figure 3 to avoid the memory explosion due to BDD operations. Note that, by setting the size constraint to 1, the algorithm proposed in [13] becomes a special case of this new approach.

V. SYNTHESIZING FUNCTIONS FOR SIGNALS IN A CANDIDATE LOCATION SET

Given a set of signals \( t_1, \ldots, t_k \), if its \( COD \) covers all the error minterms, there exists a new function \( t_i^f(X) \) for each \( t_i \) such that by replacing each \( t_i \) by \( t_i(X) \) 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 of these signals. After the new functions of these signals are decided, we apply the substitution methods [13] to realize these new functions by utilizing the existing gates of the network.

Like many problems in logic minimization, it’s difficult to formulate an exact cost function for deciding what new functions the signals \( t_i \)'s should have. Here, the ultimate goal is to make the new functions easier to be synthesized by utilizing the existing gates of the network. We developed two heuristic procedures to decide the on-set and off-set of \( t_i^f(X) \). Both of them are designed such that the synthesis procedures (Step 3 in Section II) can take advantage of the freedom of Boolean relation. These two heuristics are

1) minimize the sum of the numbers of minterms changed between \( t_i(X) \)'s and \( t_i^f(X) \)'s.

2) minimize the number of \( t_i^f(X) \)'s BDD nodes.

A. Freedom of choosing new functions for a candidate location set

As discussed in Section C, we assume the multiple-output network and the new specification have been merged into a single-output network \( Z(X) \). The on-set of \( Z(X) \) denotes the error minterms, while \( Z^t(X) \) is the zero function.

The freedom for determining the on-set and off-set of \( t_i^f(X) \) is completely characterized by the following characteristic function which represents a Boolean relation among \( t_i^f(X) \)’s:

\[ K = Z^{t_1,\ldots,t_k}(X, t_1^f, \ldots, t_k^f) = 0. \]

In other words, any combinations of \( t_i^f(X) \)'s are legal if after substituting \( t_i(X) \)'s by \( t_i^f(X) \)'s, the resulting \( Z^{t_1,\ldots,t_k}(X) \) is the zero function. In the following, for simplicity, we use \( Z \) to denote \( Z^{t_1,\ldots,t_k} \).

B. Minimize the difference between \( t_i(X) \) and \( t_i^f(X) \)

This heuristic finds a realization such that the difference between \( t_i(X) \) and \( t_i^f(X) \) is minimized. We use the number of minterms in \( t_i(X) \oplus t_i^f(X) \) to measure the difference. The problem can be formally stated as follows: given a Boolean relation of \( t_1, \ldots, t_k, Z(X, t_1, \ldots, t_k) = 0 \), find a realization \( t_i^f(X) \) for each \( t_i \) such that the total number of different minterms, \( \sum_{i=1}^{k} ||t_i(X) \oplus t_i^f(X)|| \), is minimized.

We can derive an optimal solution by dynamic programming to traverse the BDD graph once. In this formulation, given a minterm, we need to know what the old values of \( t_i \)'s are. The following characteristic function,

\[ P_i = \prod_{i=1}^{k}(t_i^f \equiv t_i(X)), \]
MINIMIZE_BDD_SIZE($K$)
\[
\{ \\
\text{for } i = 1 \text{ to } k \\
\quad K_{t_i} = \text{cofactor}_{t_i}(K) \\
\quad K_r_i = \text{cofactor}_{t_i}(K) \\
\quad \text{on} = C_{t_i+1} \ldots t_i(K_r_i) \\
\quad \text{off} = C_{t_i+1} \ldots t_i(K_r_i) \\
\quad t_i^o(X) = \text{bdd_minimize}(\text{on}, \text{off}) \\
\quad K = \text{bdd_compose}(K, t_i, t_i^o(X)) \\
\end{align*}
\]

VI. The overall EC algorithm

In Section IV, we showed the algorithm to find a candidate location set $T$, where there exists a new function $t_i^o(X)$ for each signal $t_i$ in $T$, such that the result of substituting $t_i$ by $t_i^o(X)$ in the existing network will realize the new specification. In Section V, we discussed two different heuristics to decide the new functions of $t_i$'s. Thus, the remaining work is to realize these new functions by utilizing the existing gates of the network as much as possible. We apply the direct substitution and indirect substitution methods proposed in [13] for this purpose.

We briefly describe these substitution methods here. For more details, please refer to [20, 2, 3, 13]. A connection $\text{conn}_i = (S_i, D_i)$ is a signal, where $S_i$ and $D_i$ are its source and destination gates. A connection $\text{conn}_2 = (S_2, D_2)$ is called substitutable by another connection $\text{conn}_1 = (S_1, D_1)$ if the functionality of the network remains unchanged after adding $\text{conn}_1$ and removing $\text{conn}_2$. In the case where $D_i$ is equal to $D_2$, $\text{conn}_2$ is called directly substitutable by $\text{conn}_1$. The exact requirement of $\text{conn}_2$ being directly substitutable by $\text{conn}_1$ are shown in [20, 21]. In the case where $D_i$ is different from $D_2$, $\text{conn}_2$ is called indirectly substitutable by $\text{conn}_1$. Based on the concept of indirect substitution, an ATPG-based approach is shown in [2, 3] for logic optimization.

To apply these substitution methods for our purpose, we first synthesize a network $C^T$ for the new functions $t_i^o(X)$'s based on their BDD representations (independent from the existing network). Then, the indirect and direct substitution algorithms are used iteratively to utilize the signals (or gates) in the existing network $C^o$ to replace the signals in $C^T$.

The overall EC algorithm is shown in Figure 5, where $PO^{\text{error}}(PO^{\text{correct}})$ denotes the set of outputs which are different (equivalent) in the old and new specifications. Before calling the EC algorithm, an BDD-based verification tool [1] is used to find $PO^{\text{error}}$ and $PO^{\text{correct}}$. Due to
EC($C^o$, $S^o$, $PO^{error}$, $PO^{correct}$, $k$) 
{
    $E$ = compute_error_mintermset($C^o$, $S^o$, $PO^{error}$)
    $T$ = search_candidate_location_set($E$, $k$)
    if $T$ ≠ ∅
        EC_COD_synthesize($C^o$, $T$)
        append $PO^{error}$ to $PO^{correct}$
    else
        $\{PO^{error}_1, PO^{error}_2\}$ ← partition($PO^{error}$)
        $\text{EC}(C^o, S^o, PO^{error}_1, PO^{correct})$
        $\text{EC}(C^o, S^o, PO^{error}_2, PO^{correct})$
    end
}

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

Fig. 5. The pseudo code of EC algorithm.

the imposed size constraint $k$, the algorithm might fail to
find a candidate location set to rectify all the outputs in
$PO^{error}$. If this happens, $PO^{error}$ is split into two sub-
set $PO^{error}_1$ and $PO^{error}_2$, and the EC algorithm rectifies
them separately.

VII. The Experiment

In this section, we show the experimental results of ap-
plying the EC algorithm described in the previous section.
Several combinational benchmark circuits from MCNC91 and
one industrial example (SrCr) from Fujitsu are in-
duced in our test suite.

The circuit SrCr (part of an ATM router chip) origi-
nally was given in VHDL by the designer and, later on,
the specification was modified by creating a new signal.
It 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 spec-
ification $S^o$. To obtain $C^o$, we optimized $S^o$ by running script.rugged script and then performed
technology decomposition (tech_decomp -a 4 -o 4) in SIS [1]. The numbers of gates in $S^o$ and $C^o$ are shown in
the third and fourth column of Figure 6 respectively. Bes-
side the difference in the number of gates, the networks' topology between $S^o$ and $C^o$ are quite different also.

To obtain $S^o$, we randomly modified $S^o$ by changing the
function of internal gates. For a complex gate (repre-
sented as a SOP form in BLIF format [1]), we arbitrarily
modified its cubes. For a simple gate, say an AND gate, we
changed it to an OR gate, etc. The fifth column of Fig-
ure 6 shows the number of such changes and the number
of primary outputs affected.

Then, given $C^o$ and $S^o$, we applied the EC algorithms
to generate $C^o$. We test both heuristics described in Sec-
ton V with the constraint 5 on the size of the candidate
location set. The results for the algorithms in Section
V.B and Section V.C are shown in the columns labeled
M1 and M2, respectively. For comparison, the results
from [13] are listed in the column labeled M0. We re-
port the number of added gates ($A$), removed gates ($R$)
and computation time (seconds). They could be used to
measure the 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 searching for a candidate location set,
the 6 erroneous outputs in $PO^{error}$ were partitioned into
3 sub-groups. Each of them was rectified separately. As
expected, M1 and M2 partition $PO^{error}$ into fewer number
of sub-groups. This is because they explore the chances
of modifying multiple signals simultaneously.

In terms of the EC quality, for the example b9, M1 and
M2 perform better than M0, while for x2, the results
from M0 is better. Although M0 is a special case of M1
and M2, on the average, the results for three different
heuristics are quite competitive. This is probably because
of the inaccuracy of the cost function. In other words, the
final results of EC algorithms also 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.

VIII. Conclusions and Future Work

In this paper, synthesis algorithms for the engineering change
problem are described. To realize changes of the
specification, we developed algorithms to modify the ex-
isting synthesized network minimally such that sub-
stantial portion of engineering effort can be preserved.

Our EC algorithm can be divided into two steps. The
first step identifies multiple candidate signals, such that
replacing them simultaneously with appropriate new func-
tions can rectify the difference between the old and new
specifications. The next step synthesizes these new func-
tions by utilizing gates of the existing network.

Deciding which signals to change is a major problem
in all minimization algorithms which try to change mul-
tiple signals concurrently. In our approach, this prob-
lem is solved by using the concepts of observable and co-
observable domains to guide the search. Currently, mul-
Multiple signals identification and synthesis of new functions are performed independently. Future improvement will consider new algorithms which integrate these two steps closely such that we can obtain more accurate estimate of the final changes in the EC algorithm.

Acknowledgements
This work was supported in part by the National Science Foundation under Grant MIP 9419119 and in part by the California MICRO program.

References
| I/O | S
| C
| $S^c$ | Changes, $PO^{error}$ | Results of EC algorithms |
|-----|-----|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
|     |     |                   | M0 A,R | time | M1 A,R | time | M2 A,R | time |
| zml 7/4 | 4 28 | 1,1 | 1 | 4,3 | 7 | 1 | 4,3 | 4 | 1 | 4,3 | 4 |
|       | 2,2 | 1,1 | 6,3 | 10 | 2 | 5,1 | 12 | 2 | 10,5 | 5 |
|       | 3,3 | 1,1,1 | 7,3 | 30 | 1,2 | 6,1 | 23 | 1,2 | 9,1 | 23 |
|       | 4,4 | 1,1,1,1 | 6,0 | 24 | 1,1,2 | 5,0 | 16 | 1,1,2 | 5,0 | 17 |
| b9 41/21 | 117 89 | 1,2 | 1,1 | 6,2 | 4 | 2 | 3,2 | 1 | 2 | 3,2 | 1,7 |
|       | 2,3 | 1,1,1 | 6,3 | 14 | 3 | 4,3 | 5 | 3 | 3,3 | 2 |
|       | 3,4 | 1,1,1,1 | 7,3 | 33 | 3,1 | 4,3 | 20 | 3,1 | 4,3 | 20 |
|       | 4,6 | 1,1,1,1,1 | 10,7 | 123 | 3,1,2 | 8,7 | 71 | 3,1,2 | 7,7 | 70 |
| frg1 28/3 | 3 115 | 1,1 | 1 | 3,0 | 67 | 1 | 3,0 | 120 | 1 | 3,0 | 104 |
|       | 2,2 | 1,1 | 4,1 | 74 | 2 | 6,2 | 149 | 2 | 6,1 | 151 |
|       | 3,3 | 1,1,1 | 7,2 | 71 | 1,2 | 6,2 | 170 | 1,2 | 7,3 | 165 |
|       | 4,3 | 1,1,1 | 7,5 | 74 | 1,2 | 7,3 | 161 | 1,2 | 6,5 | 161 |
| count 35/16 | 47 79 | 1,1 | 1 | 2,0 | 4 | 1 | 2,0 | 3 | 1 | 2,0 | 3 |
|       | 2,2 | 1,1 | 4,1 | 8 | 2 | 4,2 | 5 | 2 | 4,2 | 3 |
|       | 3,2 | 1,1 | 7,2 | 10 | 2 | 6,4 | 8 | 2 | 5,3 | 4 |
|       | 4,3 | 1,1,1 | 8,4 | 34 | 1,2 | 7,5 | 23 | 1,2 | 6,4 | 23 |
| x1 51/35 | 28 207 | 1,1 | 1 | 4,1 | 25 | 1 | 4,1 | 17 | 1 | 4,1 | 16 |
|       | 2,2 | 1,1 | 5,2 | 31 | 1,1 | 5,2 | 22 | 1,1 | 5,2 | 23 |
|       | 3,3 | 1,1,1 | 7,3 | 79 | 1,1,1 | 7,3 | 60 | 1,1,1 | 7,3 | 63 |
|       | 4,4 | 1,1,1,1 | 8,4 | 104 | 1,1,1,1 | 8,4 | 77 | 1,1,1,1 | 8,4 | 79 |
| x2 10/7 | 12 25 | 1,1 | 1 | 1,1 | 1 | 1 | 1,1 | 0,7 | 1 | 1,1 | 0,7 |
|       | 2,2 | 1,1 | 1,3 | 2 | 2 | 3,5 | 2 | 2 | 2,5 | 1 |
|       | 3,3 | 1,1,1 | 5,4 | 13 | 1,1,1 | 12,7 | 15 | 1,1,1 | 12,7 | 16 |
|       | 4,4 | 1,1,1,1 | 3,8 | 7 | 2,1,1 | 3,8 | 7 | 2,1,1 | 2,8 | 6 |
| C880 60/26 | 357 261 | 1,1 | 1 | 1,1 | 69 | 1 | 1,1 | 102 | 1 | 1,1 | 106 |
|       | 2,2 | 1,1 | 1,1,1 | 72 | 1,1 | 1,1,1 | 120 | 1,1 | 1,1,1 | 124 |
|       | 3,3 | 1,1,1 | 2,13 | 128 | 1,1,1 | 2,13 | 426 | 1,1,1 | 2,13 | 442 |
|       | 4,4 | 1,1,1,1 | 2,12 | 251 | 1,1,1,1 | 2,12 | 437 | 1,1,1,1 | 2,12 | 447 |
| SrCr 85/82 | 272 339 | 2,1 | 1 | 20,5 | 1471 | 1 | 20,5 | 1471 | 1 | 20,5 | 1471 |

Fig. 6. The experimental results of the EC algorithms.