Logic Rectification and Synthesis for Engineering Change

Chih-chang Lin
University of California, Santa Barbara

Kuang-Chien Chen
Fujitsu Laboratories of America, INC.

David Ihsin Cheng
University of California, Santa Barbara

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 from the existing network, such that appropriate modifications of these signals can rectify the specification change.

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 p_5 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 p_5 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 p_5, 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

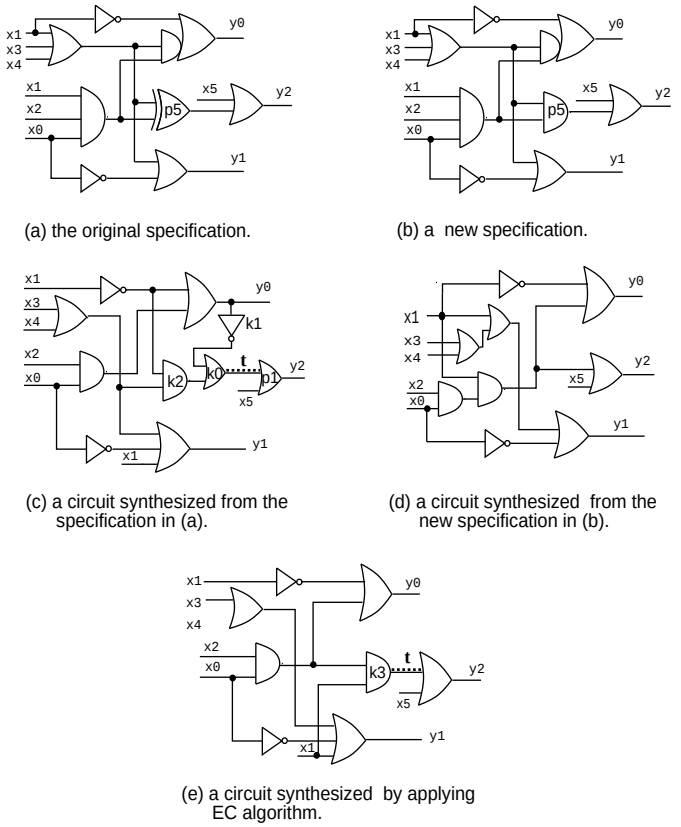

Fig. 1. An example of the EC problem.

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.

II. Terminology and previous work

Let S^o be the original specification and C^o a corresponding synthesized logic network. Suppose S^n is a new specification resulting from engineering changes. The goal of logic synthesis for engineering change is to synthesize a network C^n such that it realizes S^n and the structural differences between C^o and C^n are minimized. In the remainder of the paper, we shall simply refer to S^o (S^n) and C^o (C^n) 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^n are synthesized *independently* from S^o and S^n, respectively, and then a post–processing step is 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 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^n and contains an implementation error such that it actually

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^{error}.
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^{error}. If PO^{error} 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 $f^s(X)$ and the **error minterm set** is defined as

$$E_f(X) \;=\; f(X) \oplus f^s(X),$$

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

Definition 1 *A set of signals $t_1, \cdots, t_k$ in the network C^o is a* **candidate location set** *if there exists a new function $t_i^n(X)$ 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 $t_i^n(X)$, $f(X)$ is equal to $f^s(X)$.*

A. Observable domain

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

Definition 2 *Let $f^{t_i}(X, t_i)$ 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*

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

For a given minterm X_1 in $OD^f_{t_i}(X)$, when its applied to $f^{t_i}(X, t_i)$, 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^{t_i}. On the other hand, if X_1 is not in $OD^f_{t_i}(X)$, then the value of t_i has no control on the value of f^{t_i} (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 $f^{t_1}(a, b, c, t_1) = \bar{a}b + t_1(b + c)$ by considering t_1 as an extra input. Then, based on the above definition, we can find that among all the 8 different minterms, the minterms $\{\bar{a}bc, \bar{a}b\bar{c}\}$ make f^{t_1} become 1, while the minterms $\{a\bar{b}\bar{c}, \bar{a}\bar{b}\bar{c}\}$ make f^{t_1} become 0. As a result, OD_{t_1} is $\{abc, ab\bar{c}, \bar{a}\bar{b}c, a\bar{b}c\}$ or we can express it more conveniently as $ab + ac + \bar{b}c$.* □

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, $t_1, \cdots, t_k$, is a candidate location set, if*

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

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. ∎

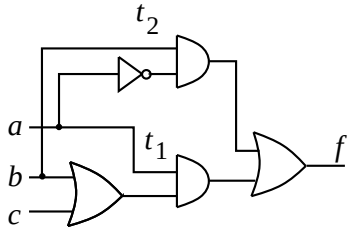

Fig. 2. Example of the observable domain and co–observable domain. The observable domain of the signal t_1 is $ab + ac + \bar{b}c$ and the co–observable domain of signal t_1 and t_2 is $\bar{a} + b + c$.

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*

$$COD^f_{t_i,t_j}(X) \;=\; \{\; X_1 \mid f^{t_i,t_j}(X_1, t_i, t_j) \neq 0 \;\; or \;\; 1,$$

$$for \; X_1 \in 2^X\}.$$

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.,*

$$OD^f_{t_i}(X) \bigcup OD^f_{t_j}(X) \;\subseteq\; COD^f_{t_i,t_j}(X).$$

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 $f^{t_1,t_2}(a, b, c, t_1, t_2) = t_2\bar{a} + t_1(b + c)$ by considering t_1, 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 $\{a\bar{b}\bar{c}\}$ makes f^{t_1,t_2} become 0. As a result, COD_{t_1,t_2} is $\bar{a} + b + c$.* □

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 $t_1, \cdots, t_k$ is a candidate location set, iff*

$$E_f(X) \;\subseteq\; COD^f_{t_1,\cdots,t_k}(X).$$

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 $f_1, \cdots, f_m$, and the new specifications are $f^s_1, \cdots, f^s_m$. We construct a single output network Z(X) as follows:

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

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 $Z^s(X)$ is the zero function. Note that here we are only allowed to modify the signals in the fanin cones of $f_1, \cdots, f_m$.

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

In this section, given a signal t_i or a set of signals $t_1, \cdots, t_k$, we show how to compute OD_{t_i} and $COD_{t_1,\cdots,t_k}$ efficiently based on BDD manipulations [17]. Then, we show how to use OD_{t_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 $t_1, \cdots, t_k$, we first construct BDD_f, which is a BDD for f in term of X and $t_1, \cdots, t_k$. Then we apply the **consensus** operator and **smoothing** operator on BDD_f with respect to the BDD variables $t_1, \cdots, t_k$ to obtain BDD^c_f and BDD^s_f separately. BDD^c_f contains all the minterms which make BDD_f 1 and BDD^s_f contains all the minterms which make BDD_f not equivalent to 0. Since COD is the set of minterms which make BDD_f not equivalent to 1 or 0, the difference between BDD^s_f and BDD^c_f is COD. The pseudo code for computing co-observable domain is as follows:

```
COD(f, X, t_1, ..., t_k)
{
  BDD_f   = build_bdd(f, X, t_1, ..., t_k)
  BDD_cod = Smooth_{t_1,...,t_k}(BDD_f) -
                Consensus_{t_1,...,t_k}(BDD_f)
  return(BDD_cod)
}.
```

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 = ∅
  C = sorted_candidate_signals_by_OD_coverage
  loop {
      t   = first_candidate(C)
      if (t is not dominated by sigals in T)
          T     = minimal_dominating_set(T + {t})
          COD   = compute_COD(T)
      end
      C   = C − {t}
      if (E_f ⊆ COD)
          return(T)
  } until ( size_of(T) > k) or (C is ∅)
  return(∅)
}
```

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 $t_1, \cdots, t_k$, if $COD_{t_1,\cdots,t_k}$ is not a candidate location set, then any combinations of signals in the fanin cones of $t_1, \cdots, t_k$ 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 $OD_{t_i} \wedge E_f$. 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 $t_1, \cdots, t_k$, if its COD covers all the error minterms, there exists a new function $t_i^n(X)$ for each t_i, such that by replacing each 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 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 $t_i^n(X)$. 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 $t_i^n(X)$'s.

2) minimize the number of $t_i^n(X)$'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 $Z^s(X)$ is the zero function.

The freedom for determining the on–set and off–set of t_i^n is completely characterized by the following characteristic function which represents a Boolean relation among $t_1^n, \cdots, t_k^n$:

$$K = Z^{t_1,\cdots,t_k}(X, t_1^n, \cdots, t_k^n) = 0.$$

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

B. Minimize the difference between $t_i(X)$ and $t_i^n(X)$

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

$$P_1 = \prod_{i=1}^{k} (t_i^o \equiv t_i(X)),$$

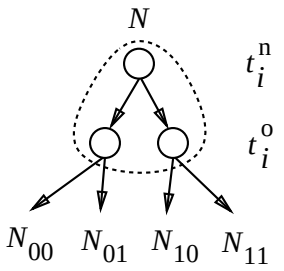

Fig. 4. Example of combining two BDD variables t_i^n and t_i^o into one super BDD variable with 4 sons.

captures these information in a BDD formula, where t_i^o's represent the old value of t_i's. We then construct the following BDD,

$$P = \overline{Z(X, t_1^n, \cdots, t_k^n)} \wedge P_1(X, t_1^o, \cdots, t_k^o),$$

with BDD variable ordering as follows:

$$x_1 < \cdots < x_n < \overbrace{t_1^n < t_1^o} < \overbrace{t_2^n < t_2^o} < \cdots < \overbrace{t_k^n < t_k^o}.$$

There are two BDD variables t_i^n and t_i^o associated with each t_i. In the BDD P, a path $X_1, t_1^n, t_1^o, \cdots, t_k^n, t_k^o$ which leads to **1**–terminal node of BDD, represents what the old value (t_i^o) of t_i is and what legal value (t_i^n) for t_i can be for the given minterm X_1.

To find an optimal solution, we virtually combine two BDD variables t_i^n and t_i^o into one super BDD variable (with 4 sons) as shown in Figure 4 and traverse the BDD P in a depth–first manner for all the BDD nodes of P with indices greater than or equal to t_1^n. During the BDD traversal, the cost function for a BDD node is computed as as follows:

$$\begin{aligned} Cost(N) \quad = \text{MIN}\{ \quad & Cost(N_{00}), Cost(N_{01}) + 1, \\ & Cost(N_{10}) + 1, Cost(N_{11}) \quad \}, \end{aligned}$$

where N is a BDD node and $N_{00}, N_{01}, N_{10}, N_{11}$ are the four sons of the BDD node N. The cost of the **1**–terminal and **0**–terminal nodes are zero and infinite, respectively. Using this cost function in the BDD traversal, we can optimally decide the values of t_i^n's for each minterm *implicitly*.

C. Minimize the number of BDD nodes of $t_i^n(X)$'s

The second heuristic is to find a realization of $t_i^n(X)$'s, such that the number of BDD nodes is minimized. As a result, the obtained functions might be easier to implement. Since it is difficult to find a global minimum solution, we sequentially extract the maximal freedom allowed to implement each $t_i^n(X)$'s and then apply the *bdd_minimize* algorithm [18, 19] to minimize the number of BDD nodes. After the new function $t_i^n(X)$ for t_i has been decided, we have to modify the characteristic function K by applying *bdd_compose* operator [1] which substitutes the BDD variable t_i of K by function $t_i^n(X)$. The algorithm is as follows:

```
MINIMIZE_BDD_SIZE(K)
{
  for i = 1 to k
    K_{t_i}     = cofactor_{t_i}(K)
    K_{t̄_i}     = cofactor_{t̄_i}(K)
    on          = C_{t_{i+1},...,t_k}(K_{t̄_i})
    off         = C_{t_{i+1},...,t_k}(K_{t_i})
    t_i^n(X)    = bdd_minimize(on, off)
    K           = bdd_compose(K, t_i, t_i^n(X))
  end
}.
```

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 $t_i^n(X)$ for each signal t_i in T, such that the result of substituting t_i by $t_i^n(X)$ 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 $conn_i = (S_i, D_i)$ is a signal, where S_i and D_i are its source and destination gates. A connection $conn_2 = (S_2, D_2)$ is called **substitutable** by another connection $conn_1 = (S_1, D_1)$ if the functionality of the network remains unchanged after adding $conn_1$ and removing $conn_2$. In the case where D_1 is equal to D_2, $conn_2$ is called **directly** substitutable by $conn_1$. The exact requirement of $conn_2$ being directly substitutable by $conn_1$ are shown in [20, 21]. In the case where D_1 is different from D_2, $conn_2$ is called **indirectly** substitutable by $conn_1$. 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^n'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 $PO^{error}(PO^{correct})$ 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^{error} and $PO^{correct}$. Due to

```
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_location_set(E_f, k)
  if T ≠ ∅
     EC_COD_synthesize(C^o, T)
     append PO^error to PO^correct
  else
     {PO_1^error, PO_2^error} ⟵ 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_COD_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. 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 subset PO_1^{error} and PO_2^{error}, and the EC algorithm rectifies them separately.

VII. The Experiment

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

The circuit $SrCr$ (part of an ATM router chip) originally 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 specification 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. Beside the difference in the number of gates, the networks' topology between S^o and C^o are quite different also.

To obtain S^n, we randomly modified S^o by changing the function of internal gates. For a complex gate (represented 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 Figure 6 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. We test both heuristics described in Section 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 report 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 existing synthesized network minimally such that substantial 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 functions can rectify the difference between the old and new specifications. The next step synthesizes these new functions by utilizing gates of the existing network.

Deciding which signals to change is a major problem in all minimization algorithms which try to change multiple signals concurrently. In our approach, this problem is solved by using the concepts of observable and coobservable domains to guide the search. Currently, mul-

tiple 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

[1] "SIS: A system for sequential circuit synthesis," Report M92/41, University of California, Berkeley, 1992.

[2] K.T. Cheng and L.A. Entrena, "Multi-level logic optimization by redundancy addition and removal," *Proc. European Conference on Design Automation*, pp. 373–377, 1993.

[3] S.C. Chang and M. Marek-Sadowska, "Perturb and simplify: multi-level boolean network optimizer," *ICCAD*, 1994.

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

[5] Y. Watanabe and R.K. Brayton, "Incremental synthesis for engineering changes," *ICCAD*, pp. 40–43, 1991.

[6] M. Fujita, Y. Tamiya, Y. Kukimoto and K.C. Chen, "Application of boolean unification to combinational logic synthesis," *ICCAD*, pp. 510–513, 1991.

[7] D. Brand, A. Drumm, S. Kundu and P. Narain, "Incremental synthesis," *ICCAD*, 1994.

[8] J. C. Madre, O. Coudert, J.P. Billon, "Automating the diagnosis and the rectification of design errors with PRIAM," *ICCAD*, 1989.

[9] H.T. Liaw, J.H. Tsaih and C.S. Lin, "Efficient automatic diagnosis of digital circuits," *ICCAD*, pp. 464–467, 1990.

[10] P.Y. Chung, Y.M Wang and I.N. Hajj, "Diagnosis and correction of logic design errors in digital circuits," *ACM/IEEE Design Automation Conference*, pp. 503–508, 1993.

[11] I. Pomeranz and S. M. Reddy, "On error correction in macro-based circuits," *ICCAD*, pp. 568–575, 1994.

[12] A. Kuehlmann, D.I. Cheng, A. Srinivasan and D.P. LaPotin, "Error diagnosis for transistor-level verification," *ACM/IEEE Design Automation Conference*, pp. 218–224, 1994.

[13] C.C. Lin, K.C. Chen, S.H. Chang, M. Marek-Sadowska and K.T. Cheng, "Logic synthesis for engineering change," *ACM/IEEE Design Automation Conference*, 1995.

[14] E. Cerny and M. A. Marin, "An approach to unified methodology of combinational switching circuits," *IEEE Trans. on Computers*, pp. 745–756, 1977.

[15] R.K. Brayton and F. Somenzi, "An exact minimizer for boolean relations," *ICCAD*, pp. 316–319, 1989.

[16] Y. Kukimoto and M. Fujita, "Rectification method for lookup–table type FPGA's," *ICCAD*, pp. 54–61, 1992.

[17] R. E. Bryant, "Graph–based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 667–691, 1986.

[18] S.C. Chang, D.I. Cheng and M. Marek-Sadowska, "BDD representation of incompletely specified functions," *EDAC*, pp. 620–624, 1994.

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

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

[21] D. Brand, "Verification of large synthesized designs," *ICCAD*, pp. 534–537, 1993.

				EC	Results of EC algorithms								
				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	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, 2	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, 12	72	1, 1	1, 12	120	1, 1	1, 12	124
				3, 3	1, 1, 1	2, 13	285	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