Logic Synthesis for Engineering Change

Chih-chang Lin
University of California, Santa Barbara

Kuang-Chien Chen
Fujitsu Laboratories of America, INC.

Shih-Chieh Chang [1]
Synopsys Inc.

Malgorzata Marek-Sadowska
University of California, Santa Barbara

Kwang-Ting Cheng
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 consider synthesis algorithms for handling such engineering changes. Given a synthesized network, our algorithm modifies it minimally to realize a new specification.

1 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.

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 it 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. After applying logic transformation and optimization procedures [1, 2, 3] on it, the resulting network is shown in Figure 1.(c). Suppose the specification in Figure 1.(a) is modified by changing* p_5 *from an XOR gate to an AND gate (Figure 1.(b)). After applying the same synthesis procedures, we obtain a network shown in Figure 1.(d). 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.*

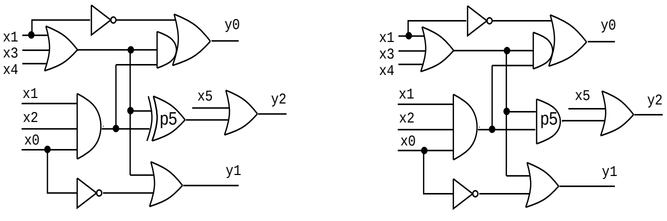
One can modify the network in Figure 1.(c) directly, and obtain a network similar to Figure 1.(c), yet realizing the new specification in Figure 1.(b). Such manual EC technique, however, cannot be easily applied in this case. This is because the signal corresponding to gate p_5 *in Figure 1.(a) is no 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.*

A good synthesis procedure which considers engineering changes will be able to modify the network in Figure 1.(c) minimally, and retain functional equivalence to the specification in Figure 1.(b). By applying our 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 *are removed and a gate* k_3 *is added.* □

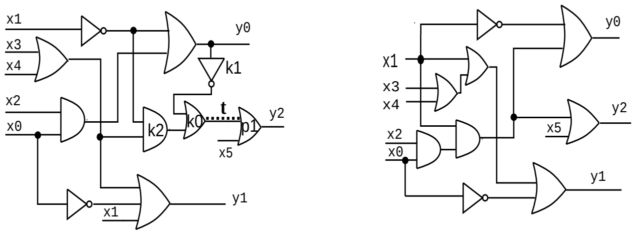
Note that changes made at high levels can potentially introduce large changes in the final design. For example, suppose a design is described in terms of its state transition graph, and modifications were made resulting in changes in the number of states and state transitions. Then, during synthesis, 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

---

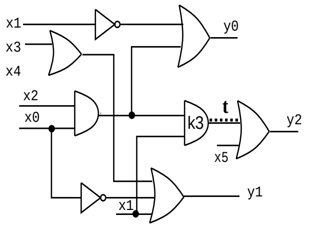
[1] this work was conducted when the author was in University of California, Santa Barbara.



(a) the original specification. (b) a new specification.



(c) a circuit synthesized from the specification in (a). (d) a circuit synthesized from the new specification in (b).



(e) a circuit synthesized by applying EC algorithm.

Figure 1: An example of the EC problem.

on the core problem of the engineering change, i.e., handling functional specification changes for *combinational networks*.

2 Basic Definitions

For simplicity, we assume all the gates in the network realize the AND, OR, NAND or NOR functions. Let $conn_i = (S_i, D_i)$ be a connection, where S_i is the source node and D_i is the destination gate. We use $f_{conn_i}(X)$ to represent the function of $conn_i$ with respect to the primary inputs X.

A connection is called redundant if the function of the network remains unchanged after adding or removing it. A connection $conn_2 = (S_2, D_2)$ is called **substitutable** by another connection $conn_1 = (S_1, D_1)$ if the function of the network remains unchanged after adding $conn_1$ and removing $conn_2$. In the case where D_1 equals to D_2, $conn_2$ is called **directly substitutable** by $conn_1$. For given connections $conn_1$ and $conn_2$, the exact requirement of $conn_2$ being directly substitutable by $conn_1$ is [4]

$$f_{conn_1}(X) \oplus f_{conn_2}(X) \subseteq ODC(conn_2),$$

where $ODC(conn_2)$ represents the observability don't–cares of $conn_2$ [2]. In the case where D_1 is different from D_2, $conn_2$ is called **indirectly substitutable** by $conn_1$. There is no simple way to tell if $conn_1$ can indirectly substitute $conn_2$ except by explicitly checking the equivalence of two networks, one with $conn_1$ and the other with $conn_2$. Moreover, there are $O(N^2)$ such possible substitutions, where N is the number of gates in the network. It would be time–consuming to do an explicit search on the whole network.

To search heuristically for indirectly substitutable connections, a technique used in a multi–level logic optimizer [2] can be employed. Given a $conn_2$, [2] restricts the search to $conn_1$'s, where $conn_1$'s are redundant connections and D_1's are dominators [3] of D_2. After adding $conn_1$ (a redundant connection) to the network, $conn_2$ can be removed if it becomes redundant, therefore $conn_1$ is indirectly substituting $conn_2$. We will give an example in Section 4.2. For more details, please refer to [2, 3].

In the following discussion, let S^o be the original specification and C^o a 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 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.

3 Previous Work

In [5], C^o and C^n are synthesized independently from S^o and S^n, 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 [6, 7], 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. This approach is useful when changes are made at a later stage of the design process. However, the pre– and post–logic added may 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 [8], a novel approach is proposed which explores the *structural* equivalence between the S^o and S^n, and the *functional* equivalence between the S^o and C^o. Using these structural and functional equivalence, [8] establishes a mapping between the signals in C^o and the ones in S^n. 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 specifications and C^o.

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 specifi-

---

[2] In [4], $f_{conn_1}(X)$ is called a permissible function of $conn_2$.

[3] A node A is called a dominator of another node B if every path from B to the primary outputs passes through A.

cation, and our goal is to modify C^o into another network C^n which implements S^n correctly. In [9, 10, 11], error diagnosis and 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.

In the following sections, we will discuss our approach for solving the engineering change problem. Note that since the amount of design modifications needed for achieving 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.

4 Synthesis Algorithms for EC

As discussed in the previous section, the error diagnosis problem has a strong relationship to the EC problem. However, in error diagnosis, usually a simple single–error model is assumed. In EC problems, our experience shows that 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. Therefore, 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 network and new specification, and also derive the target functions of those candidate signals.

2) Synthesis of the target 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. Furthermore, the synthesis algorithms in Step 2 have to be powerful enough so that the target functions can be realized using as few gates as possible.

4.1 Identification of Candidate Signals

To identify candidate signals which can potentially rectify the difference between the old network and new specification, an error location technique discussed in [10, 11] has been extended for use in our approach.

In the old network C^o, suppose a signal t with function $f_t^o(X)$ is selected (X is the set of primary inputs), we want to know if there exists a new function $f_t^n(X)$ such that replacing the function of t by $f_t^n(X)$ will yield a network which implements the new specification S^n. The following lemma [10, 11] states the necessary and sufficient condition.

Lemma 1 *Let $y_i^o(X)$ be the function of the i^{th} output, $1 \le i \le k$, of C^o, and $y_i^o(X,t)$ the same function expressed in terms of X and t. Let $y_i^n(X)$ be the function of the corresponding i^{th} output in the new specification S^n. Then, the difference between C^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:*

$$y_i^n(X) \oplus y_i^o(X, t = f_t^n(X)) = 0, \ \text{for } 1 \le i \le k, \qquad (1)$$

which is equivalent to

$$y_i^n(X) \oplus y_i^o(X, t = 0) \le f_t^n(X) \le \overline{y_i^n(X) \oplus y_i^o(X, t = 1)}, \quad \text{for } 1 \le i \le k. \qquad (2)$$

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

$$f_t^{on}(X) = \bigcup_{i=1}^{k} y_i^n(X) \oplus y_i^o(X, t = 0), \ \text{and}$$

$$f_t^{off}(X) = \bigcup_{i=1}^{k} y_i^n(X) \oplus y_i^o(X, t = 1).$$

Essentially, Lemma 1 states that each output y_i poses a constraint on the t and when all the constrains can be satisfied simultaneously, the signal t is a candidate signal. Note that information of the network's structure (e.g. intersection of fanin cones, dominator sets [11], etc.) can be used to trim down the search space and an OBDD-based [12] method can implement Lemma 1 efficiently. For example, in Figure 1.(c), we can identify the connection $t = (k0, p1)$ as the candidate signal, and its target function is as follows:

$$f_t^{on}(X) = x_0 x_1 x_2 \overline{x}_5, \text{ and } f_t^{off}(X) = (\overline{x}_0 + \overline{x}_1 + \overline{x}_2)\overline{x}_5.$$

Once a candidate signal t is found, we realize its target function by a subcircuit. The size of the subcircuit can be very small (e.g., a single gate) or it could be quite large, depending on both the target function and synthesis algorithms, and this is where the quality of EC solutions is determined.

If many outputs of the old network and new specification are different, the chance of finding a candidate signal which can rectify all of them becomes small, and thus, we need to consider the possibility of changing the functions of many signals. In our approach, when a single candidate signal cannot be identified, a divide–and–conquer strategy is applied to overcome the problem. Let $PO^{error}(PO^{correct})$ denote the set of outputs which are different (equivalent) in the old network and new specification. During divide–and–conquer step, the set PO^{error} is heuristically partitioned into two sub–groups (PO_1^{error} and PO_2^{error}) and then each sub–group is rectified sequentially. When searching for a candidate signal to rectify the erroneous outputs in PO_i^{error}, we have only to check Lemma 1 on the outputs in PO_i^{error} and $PO^{correct}$. Thus, the possibility of finding such a signal increases. We developed a simulated annealing based approach to find a partition such that the intersection of fanin cones in each sub–group is maximized to increase the chances of finding candidate signals. 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^{error} individually.

4.2 Synthesizing Target Functions for Candidate Signals

As discussed in the previous subsection, an incompletely specified Boolean function represented by $[f_t^{on}(X), \overline{f_t^{off}(X)}]$ can be determined as a target function for a candidate signal t. Our goal is to synthesize a function $f_t^n(X) \in [f_t^{on}(X), \overline{f_t^{off}(X)}]$ using as few gates as possible, by fully utilizing the existing logic in C^o. We examine several such algorithms in this subsection.

The first algorithm adopted in our approach is the RENO algorithm [13], which tries to synthesize a single complex gate to realize the target function. First, in RENO, a list of *candidate cubes* are generated using outputs of internal gates in C^o. Each candidate cube when expressed in terms of primary inputs X covers a portion of $f_t^{on}(X)$ but none of $f_t^{off}(X)$. After generating a set of candidate cubes which can cover $f_t^{on}(X)$, the problem is transformed into the so–called **sum–to–one subset** problem which can be viewed as a generalized 2–level minimization problem. The solution obtained will be a SOP form in terms of selected candidate cubes. Also, the quality of the synthesis result depends on the total number and choice of the candidate cubes. In our experiments, computation time is reasonable when the number of candidate cubes is less than 30. However, this number is usually not large enough to fully explore the synthesis possibilities.

To enhance the gate synthesis algorithm, we developed another approach for single complex gate synthesis. In this approach, instead of generating candidate cubes explicitly, we used an image computation technique [14] to examine implicitly the candidacy of all the 3^k cubes formed by a set of k internal signals $g_1, \cdots, g_k$. The idea is as follows: a cube (formed by $g_1, \cdots, g_k$) is a candidate cube if and only if its function does not intersect the off–set $f_t^{off}(X)$. Since the image of the off–set $f_t^{off}(X)$, with respect to $g_1, \cdots, g_k$, represents cubes which intersect the off–set, the complement of the image contains all the candidate cubes formed by the k signals and they can be used to synthesize the on–set $f_t^{on}(X)$.

Lemma 2 *The following formula computes all the candidate cubes formed by a set of k signals, $G = \{g_1, \cdots, g_k\}$:*

$$\{candidate\ cubes\} = \overline{image_G(f_t^{off}(X))},$$

where $image_G()$ is a mapping from 2^X to 2^G.

Compared with RENO, the image computation based method can examine more candidate cubes and thus obtain better results. In comparison to [10, 11], the image based method is more efficient since all the minterms in terms of G are considered simultaneously and an OBDD–based technique [14] can be used to achieve efficient computation. However, it is still a very difficult problem to choose an optimal set of k signals. We have developed heuristics in which the signals are incrementally added to the set G and conditionally rejected if the coverage of $f_t^{on}(X)$ does not increase.

The previous two approaches both try to realize the target function by a single complex gate. It was found experimentally that very often, although a candidate signal can be found, it cannot be realized by a single gate using the gate synthesis algorithms discussed above. It suggests that trying to realize the target function by just one gate is probably too restrictive, and a more general synthesis approach is still needed.

The third approach, which is also the most general one, synthesizes a subcircuit K_t to realize $f_t^{on}(X)$ in terms of the primary inputs first, and then this subcircuit is minimized by using existing logic of the old network C^o. Two minimization techniques, one based on permissible functions [4], and one based on ATPG techniques [2, 3] were employed.

In the permissible function based method, for each connection $conn_2$ in the subcircuit K_t, we search for a connection $conn_1$ in C^o which can **directly substitute** $conn_2$. If this happens, the connection $conn_2$ and frequently also a large portion of its fanin cone in K_t can be removed.

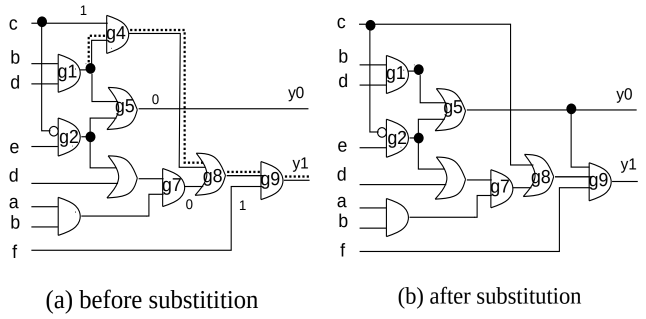
In the ATPG–based method [2, 3], the effort is on finding connections which can **indirectly substitute** a chosen connection in K_t. The goal here is to find $conn_1$, which when connected to $conn_2$'s dominators make the $conn_2$ redundant and therefore removable.

Example 2 *Consider the irredundant network in Figure 2.(a). Suppose we are searching for indirectly substitutable connections of $conn_2 = (g_1, g_4)$. The idea is to find a connection $conn_1$ such that its addition to the network will make the connection $conn_2$ redundant. To preserve the functionality of the network, we also have to make sure that the connection $conn_1$ is redundant in the original network.*

For $conn_2$ to be irredundant, the assignments $\{g_1 = 0, c = 1, g_7 = 0, f = 1\}$ on the gates must be realizable by at least one input 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 $conn_2$ and its input g_7 is required to be zero to allow the value of $conn_2$ propagate through the dotted line. 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 $conn_2$ through the dotted line and thus make the connection $conn_2$ redundant. Before removing $conn_2$, we check if the connection $conn_1 = (g_5, g_9)$ is redundant. The answer is yes, thus, we can add $conn_1$ and remove $conn_2$. The result is shown in Figure 2.(b). □

Note that the justification and propagation constraints described above are only the necessary constraints. The example shows how to add a redundant connection to the network to violate the propagation constraints of a target connection. We can also apply the same technique to violate the justification constraints.

These two kinds of substitution (direct and indirect) methods are complementary because they explore different substitution domains. Indirect substitution may the-



(a) before substitition

(b) after substitution

Figure 2: Indirect substitution.

oretically 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 huge portion of time on hard–to–detect faults. As a result, the indirect substitution does not completely cover the search domain of direct substitution.

4.3 Implementation of the EC algorithm

We have implemented the above EC algorithm based on SIS 1.2 [1] and CMU BDD package [15]. Before calling the EC algorithm, an OBDD–based verification tool [1] is used to find PO^{error} and $PO^{correct}$. The overall EC algorithm is shown in Figure 3.

Given an incompletely specified Boolean function, the procedure perform_bdd_minimize() finds an OBDD representation with minimal number of OBDD nodes and returns its OBDD size [16, 17]. The procedure choose_best_candidate() finds a signal t among A with minimal number of OBDD nodes.

5 The Experiment

In this section, we show the results on several combinational benchmark circuits from MCNC91 and one industrial example ($SrCr$) from Fujitsu.

The circuit $SrCr$ (part of an ATM router chip) originally was given in VHDL. Later on, the specification was modified by creating a new signal. It was a hierarchical design and contained flip–flops. For the experiment, we flattened the design and extracted its combinational portion.

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 resulting number of gates is shown in the third column of Figure 4. 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 AND gate, we changed it to OR gate, etc. The fourth column shows the number of such changes and the number of primary outputs affected.

```
$EC(C^o, S^n, PO^{error}, PO^{correct})$
{
    $A$ = search_candidate_signals($C^o, S^n, PO^{error}, PO^{correct}$)
    if $A \neq \emptyset$
      foreach $t$ in $A$
        $size_t \longleftarrow$ perform_bdd_minimize($f_t^{on}, f_t^{off}$)
      end
      $t \longleftarrow$ choose_best_candidate($A$)
      EC_synthesize($C^o, t, f_t^{on}, f_t^{off}$)
      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($C^o, t, f_t^{on}, f_t^{off}$)
{
    $t_{new} \longleftarrow$ RENO_image_synthesize($C^o, t, f_t^{on}, f_t^{off}$)
    if successful
      replace $t$ by $t_{new}$ in $C^o$
    else
      $K_t \longleftarrow$ construct_subcircuit_from_bdd($f_t, X$)
      replace $t$ by $K_t$ in $C^o$ and mark the gates in $K_t$
      loop {
        perform_indirect_substitution($C^o$)
        perform_direct_substitution($C^o$)
      } until (no further improvement)
    end
}
```

Figure 3: The pseudo code of EC algorithm.

Then, we applied the EC algorithm on a SUN SPARC 5 machine (128M memory) to generate C^n. The fifth column shows the result of recursively partitioning erroneous outputs. For example, $(1, 1, 1, 1, 2)$ shows that the number of different outputs in PO^{error} is 6, and during searching for candidate signals, the outputs in PO^{error} were partitioned into 5 sub–groups. The sixth column shows the amount of modifications on C^o to generate C^n. We report the number of added gates (A) and removed gates (R). From our experiments, on the average, the amount of modification on C^o to obtain C^n is less then 5% except for the circuit $z4ml$. Also, the more we change the specification, the more added and removed gates are required.

6 Conclusions

In this paper, synthesis algorithms for the *engineering change* problem are described. To realize the changes of the specifications, 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 candidate signals, such that replacing them with the target functions can rectify the difference between the old network and new specification. The next step synthesizes these target functions by utilizing gates of the existing synthesized network. The experimental results show that our approach is effective for combinational circuits.

Acknowledgments

This work was supported in part by NSF grant MIP91–17328 and in part by Xilinx through 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*, pp. 2–5, 1994.

[4] 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.

[5] 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.

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

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

[8] D. Brand, A. Drumm, S. Kundu and P. Narain, "Incremental synthesis," *ICCAD*, pp. 14–18, 1994.

[9] J. C. Madre, O. Coudert, J.P. Billon, "Automating the diagnosis and the rectification of design errors with PRIAM," *ICCAD*, pp. 30–33, 1989.

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

[11] 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.

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

[13] K.C. Chen, Y. Matsunaga, M. Fujita and S. Muroga, "A resynthesis approach for network optimization," *ACM/IEEE Design Automation Conference*, pp. 458–463, 1991.

			EC	Result of EC algorithm		
	S^o	C^o	changes, PO^{error}	partitions	A, R	sec
$z4ml$	4	28	1, 1	(1)	4, 3	7
			2, 2	(1, 1)	6, 3	10
			3, 3	(1, 1, 1)	7, 3	30
			4, 4	(1, 1, 1, 1)	6, 0	24
$b9$	117	89	1, 2	(1, 1)	6, 2	4
			2, 3	(1, 1, 1)	6, 3	14
			3, 4	(1, 1, 1, 1)	7, 3	33
			4, 6	(1, 1, 1, 1, 2)	10, 7	123
$frg1$	3	115	1, 1	(1)	3, 0	67
			2, 2	(1, 1)	4, 1	74
			3, 3	(1, 1, 1)	7, 2	71
			4, 3	(1, 1, 1)	7, 5	74
$count$	47	79	1, 1	(1)	2, 0	4
			2, 2	(1, 1)	4, 1	8
			3, 2	(1, 1)	7, 2	10
			4, 3	(1, 1, 1)	7, 5	34
$x1$	28	207	1, 1	(1)	4, 1	25
			2, 2	(1, 1)	5, 2	31
			3, 3	(1, 1, 1)	7, 3	79
			4, 4	(1, 1, 1, 1)	8, 4	104
$x2$	12	25	1, 1	(1)	1, 1	1
			2, 2	(1, 1)	1, 3	2
			3, 3	(1, 1, 1)	5, 4	13
			4, 4	(1, 1, 1, 1)	3, 8	8
$C880$	357	261	1, 1	(1)	1, 1	69
			2, 2	(1, 1)	1, 12	72
			3, 3	(1, 1, 1)	2, 13	258
			4, 4	(1, 1, 1, 1)	2, 12	251
$SrCr$	272	339	2, 1	(1)	20, 4	762

Figure 4: Experimental results of EC algorithm.

[14] O. Coudert, C. Berthet and J. C. Madre, "Verification of sequential machines based on symbolic execution," *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, 1989.

[15] K.S. Brace, R.L. Rudell and R.E. Bryant, "Efficienct implementation of a BDD package," *ACM/IEEE Design Automation Conference*, pp. 40–45, 1989.

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

[17] T.R. Shiple, R. Hojati, A.L. Sangiovanni-Vincentelli and