INCREMENTAL SYNTHESIS FOR "ENGINEERING CHANGES"

by

Yosinori Watanabe and Robert K. Brayton

Memorandum No. UCB/ERL M90/76

27 August 1990

INCREMENTAL SYNTHESIS FOR "ENGINEERING CHANGES"

by

Yosinori Watanabe and Robert K. Brayton

Memorandum No. UCB/ERL M90/76

27 August 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

INCREMENTAL SYNTHESIS FOR "ENGINEERING CHANGES"

by

Yosinori Watanabe and Robert K. Brayton

Memorandum No. UCB/ERL M90/76

27 August 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Incremental Synthesis for "Engineering Changes"

Yosinori Watanabe and Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA, 94720, USA

August 27, 1990

Abstract

The problem of rectifying design incorrectness due to specification changes as well as design errors of VLSI's is introduced. In the light of the practical significance of developing systems for the automatic rectification, a formulation of the rectification problem and a basic approach using logic synthesis techniques are presented. It is shown that the rectification problem is closely related to topics in sequential logic verification and the synthesis for hierarchical networks. The paper provides a review of both topics which are currently being investigated by Berkeley's Logic Group.

1 Introduction

A goal for computer-aided design (CAD) systems for very large scaled integrated circuits (VLSI) is to realize near-optimal designs that meet specifications set by the designer and are competitive with or better than manual designs [3]. The ability to quickly produce such designs is crucial for the application-specific integrated circuits (ASIC) market.

Historically, CAD started with developing systems for the analysis of designs and has evolved toward automatic synthesis. Both analysis and synthesis systems

for various CAD areas have been developed and made commercially available. In the practical design process of VLSI's, however, what requires a large amount of time is rectification due to design incorrectness. Although the development of automatic synthesis systems may result in reducing the designer's efforts for rectification, one sees that the actual design process consists of analysis, synthesis, and rectification. Nevertheless, few systems for rectification are available.

There are two reasons that rectification is required in designing VLSI's. One is design errors, the main reason for the rectification so far. Design errors arise locally in small portions of the design. Designers try to complete the rectification by slightly modifying the erroneous part. The other reason is specification changes. It is usual to encounter changes of the specification in the design process. The rectification due to such an "engineering change" is generally more complicated than for design errors since unlike design errors, a large part of the design may be affected by the change of the specification and it is difficult to find how to modify the original design. Therefore, in some cases, a change of the specification forces a redesign rather than rectification. Such a redesign is also necessary if automatic synthesis from a behavioral description to silicon is used, because a set of masks are generated automatically and the designer is not allowed to touch the intermediate products of the design. However, in some cases, the behavior of the redesigned masks is close to that of the original masks, so that it may be more practical to use the original design with rectification. An automatic system, which makes the minimal rectification on the original design to meet the correct specification and functions interactively with automatic synthesis systems, produces the following advantages. First, the designer can obtain rectified designs without finding the erroneous part or the means of modification, and thus both the designer's efforts and the design time spent on rectification will be dramatically reduced. Second, automatic rectification will be possible for designs generated by automatic synthesis systems and thus a set of rectified masks will be obtained as

alternatives to redesigned ones. While there exists a strong demand for automatic synthesis with which correct designs are rapidly generated, one cannot ignore the practical significance of the automatic rectification. Automatic rectification will become an increasingly important issue as the ASIC market continues its rapid growth since designing ASIC's typically requires frequent specification changes.

We have focused on the automatic rectification from the logic synthesis point of view. This paper introduces a precise formulation of the rectification problem which is reasonable enough for practical use and leads to related theoretical questions and results.

If one considers the rectification problem in the domain of logic synthesis, the problem essentially requires the comparison of the Boolean functions corresponding to the correct and the incorrect designs. Therefore it is crucial to compute efficiently the image of a subset of the domain for a given Boolean mapping. Coudert *et al.* in [8] have proposed methods for the image computations of Boolean functions to enumerate reachable states of finite state machines. The methods have been applied to verification of finite state machines [24, 8] and have a wide range of applications for problems which require the state enumeration of finite state machines [24]. Some examples stated in [24] are verification of asynchronous circuits for particular protocols [7], testing of sequential faults in non-scan designs [10], and computation of the initial state after retiming [12]. Hence one sees that developing efficient methods for image computation is a key issue in logic synthesis and has been actively investigated in recent years.

The rectification problem requires a technique which synthesizes a Boolean network with the minimal area whose functionality is compatible with a one-to-many Boolean mapping that is obtained based on the comparison of the Boolean functions of the correct and incorrect designs. A one-to-many Boolean mapping is called a Boolean relation [5]. Boolean relations are a generalization of incompletely specified Boolean functions [5, 2]. Although it is well known that don't

cares are a powerful source for optimizing incompletely specified functions [2], not all aspects of incomplete specification of Boolean functions can be captured using don't cares. Minimization of Boolean relations helps in deriving smaller networks which otherwise cannot be obtained. Boolean relations arise in several contexts, e.g. in synthesis for hierarchical networks [5] and in synthesis of finite state machines with sets of equivalent states [13]. An exact procedure for the minimization of Boolean relations has been proposed [23]. Because of its exponential complexity, however, the method cannot be applied to large examples. It is important to develop ways of minimizing Boolean relations in reasonable time with reasonable memory space [11].

Thus, the rectification problem is related to the two fundamental issues of logic synthesis. In the light of the theoretical interest of the rectification problem, we present what the rectification problem is, how the problem is solved, and how the problem is related to other topics in this field. In Section 2, the rectification problem is formulated. Section 3 presents a basic approach for the problem to show how the image computations of Boolean mappings and Boolean relations are used. In Section 4 and Section 5, we describe the problems of the image computation and Boolean relations respectively and discuss basic approaches and applications mainly being carried out by Berkeley's Logic Group. Development of an automatic rectification system is discussed in Section 6, in which we provide an answer to a fundamental question for the rectification problem, i.e. whether rectification succeeds for a given design. Section 7 concludes this paper.

2 The Rectification Problem

A problem of rectifying design incorrectness is formulated. A typical situation where rectification is considered is when the designer finds that the behavior of the design needs to be changed after a set of layout masks has been already obtained

in the final stage of the design process. Therefore we suppose that rectification is made for a combinational logic circuit represented at a gate level corresponding to a set of layout masks which have already been obtained.

2.1 Definitions

We begin with some terminology required for the problem formulation.

Definition: Boolean Network

> A Boolean network is a directed acyclic graph where each node is a Boolean variable y_i and a representation f_i of a Boolean function. A directed arc from node i to node j exists in the graph if node j uses the variable y_i explicitly in the representation f_j.

A Boolean network is a multiple level implementation of a set of Boolean functions. For the purpose of rectification, we consider a Boolean network as one corresponding to a combinational logic circuit represented at a gate level. In the rest of the paper, we do not distinguish a Boolean network from a combinational logic circuit or a gate level logic circuit.

Definition: Functional Error

> A combinational logic circuit has a functional error if for some assignment of signals at the primary inputs, the combination of signals desired by the designer does not appear at the primary outputs after an infinite interval of time.

We do not consider temporal design incorrectness in this paper. Namely, an intended behavior of a given circuit is defined as follows.

Definition: Intended Behavior

> A combinational logic circuit has an intended behavior if it has no functional errors.

An intended behavior is generally given by the designer as a design specification. Recall that functional errors are detected at the mask level and are due to either design errors or engineering changes.

2.2 The Rectification Problem

Our objective is to obtain a gate level circuit with no functional error by means of the minimum rectification. We consider that the rectification made for a given gate level circuit, if possible, consists of (1) inserting additional circuitry between the gate pins, before the primary input pins or after the primary output pins, and (2) connecting input pins either to a power supply or ground. We exclude any other rectification such as changing the number of input pins of gates or changing types of gates. Here we define a gate pin or a pin as a terminal of a cell corresponding to the gate that is facing the routing area and an input pin (respectively output pin) as a pin that is used as an input (output) for the cell. The operations necessary to realize the rectification we consider above are (1) creating circuitry, (2) cutting wires inside the circuit, and (3) connecting a wire from a pin to power supply, ground, or newly created circuitry. In general, the possibility of rectifications for a given circuit depends upon its physical design. A chip obtained at the final stage of the design process is typically so dense that little room is available for placing additional circuitry or for wiring inside the circuit while one may expect some room external to the circuit in the chip. Therefore, we assume that creating circuitry and/or wiring are allowed only in the area surrounding the circuit inside the chip. With this assumption, the rectification considered above is restricted to (1) inserting additional circuitry before the primary input pins or after the primary output pins, and (2) connecting input pins either to power supply or ground. Based on the assumption, the rectification problem is formulated as follows.

Problem:

Given a Boolean network η and an intended behavior, find a least cost Boolean network η' with no functional error by

1. creating a Boolean network, η_I, driven by the primary inputs of η such that an input of a node in η, which is driven by some primary output of η_I, was originally connected to a primary input of η,
2. creating a Boolean network, η_O, driven by the primary outputs of η, and
3. sticking the minimum number of the inputs of the nodes in η to 1 or 0.

We use as the cost either the number of cubes or the number of literals [3], depending upon the method of the implementation.

2.3 Related Works

Several methods for the problem of rectifying functional errors of a given Boolean network have been proposed [15, 21]. A method provided in [15] performs a backward traversal for one primary output at a time to detect one gate responsible for functional errors, followed by computing a set of functions to which the function of the gate must be changed so that the rectification is completed. This method works under a single fault assumption. For a multiple fault case, however, stringent constraints are required for the method to succeed in the rectification. In [21], a backward rectification method based on the transduction method [20] has been proposed for single output NOR gate networks. Our approach, provided in the following section, is novel in the sense that rectification is applied for all the primary

outputs at a time and functional errors are rectified without locating erroneous gates.

3 Basic Approach

We present a basic approach for part of the rectification problem which does not consider sticking inputs to either 1 or 0. The main reason for this exclusion is that it is not clear how the stuck inputs affect the functionality of the entire network. At present, we have no way to estimate how much additional incorrectness is introduced by sticking a single input. In addition, we consider attaching a network after the primary outputs only if rectification is not completed by means of a network created before the primary inputs. In Section 6, we introduce a necessary and sufficient condition under which the rectification can be completed by creating a network before the primary inputs only. Creating a network after the primary outputs is straightforward if also the primary inputs are available, as described in Section 6. An alternative method which uses only the primary outputs has not been found. Therefore, our current approach is to find a network with the least cost which is attached before the primary inputs to complete the rectification as much as possible, and then to complete the rectification if necessary using an output network.

Given a multiple output Boolean network, η, which requires rectification, let n be the number of the primary inputs and m the number of the primary outputs. Let t be the number of input lines that are fed by the primary inputs. These lines are called *primary input lines*. The introduction of the primary input lines is based on the observation that typically in a chip or module, all the primary input signals that corresponds to the same primary input variable are not driven from the exterior through the same pin and thus it is possible to assign different signals to pins that were originally used as the same primary input.

Let $f^{(i)} : B^n \rightarrow B$ be a Boolean function for the i-th primary output specified by the intended behavior and represented in terms of the primary inputs. Namely, $f^{(i)}$ is a function which must be satisfied by the i-th primary output in the correct design. Let $g^{(i)} : B^t \rightarrow B$ be a Boolean function for the i-th primary output of the original network η and represented in terms of the primary input lines. Recall that each primary input line is treated as an independent Boolean variable and there are $t \geq n$ of these variables. Finally, for each vertex (or minterm)[1] x of the n dimensional Boolean space B^n, let $r(\mathrm{x})$ be the set of minterms y of B^t with the property that $f^{(i)}(\mathrm{x}) = g^{(i)}(\mathrm{y})$ for $i = 1, \cdots, m$. In general, $r(\mathrm{x})$ is a one-to-many mapping and is called a Boolean relation [5]. Consider a Boolean network η_I with n inputs and t outputs which is driven by the primary inputs and feeds the primary input lines. If for each minterm of the primary inputs, x, the minterm y obtained at the primary output stage of η_I is a member of $r(\mathrm{x})$, then the minterm y is applied to the original network η to realize the intended functionality $f^{(i)}(\mathrm{x})$ for $i = 1, \cdots, m$. Specifically, let $h : B^n \rightarrow B^t$ be a multiple output Boolean function. The function h is *compatible* with r if $h(\mathrm{x}) \in r(\mathrm{x})$ for each minterm x in B^n. Then our objective is to find a Boolean network η_I with a compatible function h with the least cost. A structure of the rectified network is shown in Figure 1.

We performed experiments for several examples. Hardware description of a correct and incorrect designs were specified and translated into Boolean functions using BDSYN [22]. Both designs were optimized to multiple level networks by misII [4]. The incorrect network was mapped to a gate level network by specifying a library. Then for each minterm x of B^n for which rectification was required, $r(\mathrm{x})$ was computed and the relation r was minimized using a Boolean relation minimizer [23] to find a compatible function h. A more detailed description of

[1] In this paper, we make no distinction between a vertex, a minterm and an element of a Boolean space.

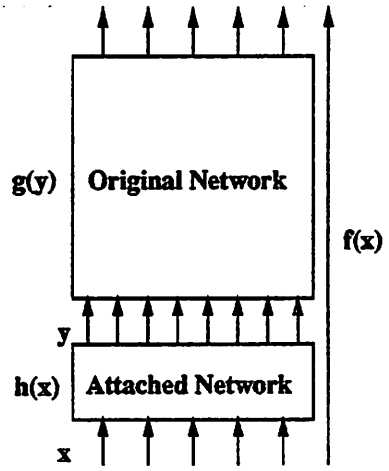

Figure 1: Structure of the Rectified Network

this approach is found in Section 6. According to the experiments, $r(\mathrm{x})$ is usually a very large set and thus it is important to compute and store $r(\mathrm{x})$ efficiently. This problem is generalized to the problem of the image computation for Boolean relations. We also need a method which minimizes Boolean relations in reasonable time. We now present how these fundamental problems, namely the image computations and the minimization of Boolean relations, are handled. As previously mentioned, both problems are recognized as important issues in the field of the logic synthesis because of their wide range of applications and possible future impact. Thus they have been actively investigated in the community including Berkeley's Logic Group. We describe the respective problems in the next sections and focus on solution methods mainly proposed by Berkeley's Logic Group. Image computations are discussed in Section 4. Section 5 describes Boolean relations.

4 Image Computations

The efficient computation of the images for Boolean mappings is important for many algorithms in a wide variety of applications. One such example is the problem of enumerating the set of states reachable from the initial states of a finite state machine. This arises in equivalency checking between the sequential machines [8], verification of asynchronous circuits for particular protocols [7], and fault propagation for highly sequential circuits [10]. As we will see in Section 5, a finite state machine contains a number of equivalent states in general, so that it is represented in terms of Boolean relations. Therefore, it is desirable to develop an efficient way to compute the images for Boolean relations as well as for Boolean functions.

We first describe the image and the inverse image computations of Boolean functions, followed by an extension for Boolean relations.

4.1 Image Computations of Boolean Functions

The formal definition of the image or the inverse image of a set by a Boolean function is given as follows.

Definition: Image of a Boolean Function

Given a Boolean function $f : B^n \rightarrow B^m$ and a subset A of B^n, the image of A by f, denoted $f(A)$, is the set of minterms $\mathrm{y} \in B^m$ for which there exists a minterm $\mathrm{x} \in A$ such that $\mathrm{y} = f(\mathrm{x})$.

Definition: Inverse Image of a Boolean Function

Given a Boolean function $f : B^n \rightarrow B^m$ and a subset A of B^m, the inverse image of A by f, denoted $f^{-1}(A)$, is the set of minterms $\mathrm{x} \in B^n$ for which there exist a minterm $\mathrm{y} \in A$ such that $\mathrm{y} = f(\mathrm{x})$.

We are interested in computing $f(A)$ for given f and $A \subseteq B^n$, or conversely, computing $f^{-1}(A)$ for $A \subseteq B^m$. Coudert *et al* originally suggested methods for this problem [8] and an improved version of the techniques have been implemented [24]. The methods employ binary decision diagrams (BDD) [6] to represent Boolean functions. Boolean functions can be manipulated efficiently with the use of BDD's. Several techniques of efficient implementation of BDD's have been proposed [1, 19]. The BDD based methods are able to perform the image computations efficiently. We provide a review of the methods based on the reference [24].

4.1.1 The Transition Relation Method

The transition relation method requires a BDD operation called the *smoothing operator* [18] defined as follows.

Definition: Smoothing Operator

Given a Boolean function $f : B^n \rightarrow B$ and a set or a subset of the input variables for f, $\mathrm{x} = \{x_1, \cdots, x_k\}$, the smoothing operator of f by x is defined as

$$S_{\mathrm{x}}(f) \;=\; S_{x_1}(S_{x_2}(\cdots S_{x_{k-1}}(S_{x_k}(f))\cdots)),$$

where $S_{x_i}(g) = g_{x_i} + g_{x_i'}$, and g_a designates the cofactor of the function g with respect to the literal a.

Note that the smoothing operator is independent of the order of the input variables by which a Boolean function is cofactored. The smoothing of a function is realized very efficiently on BDD's.

We are ready to present the transition relation method, which computes the images of Boolean functions.

Definition: Consistency Function

Given a Boolean function $f : B^n \rightarrow B^m$, the consistency function of f, $F(\mathrm{x}, \mathrm{y})$, is the characteristic function defined for a subset of $B^n \times B^m$ which consists of pairs of the minterms (x, y) such that $\mathrm{y} = f(\mathrm{x})$.

If we denote $f_i(\mathrm{x}), i = 1, \cdots, m$ as a Boolean representation of the i-th function of $f : B^n \rightarrow B^m$ in terms of the input variables $\{x_1, \cdots, x_n\}$, then the consistency function of f is represented as $F(\mathrm{x}, \mathrm{y}) = \prod_{i=1}^{m} (y_i \equiv f_i(\mathrm{x}))$, where y_i is the i-th output variable defined in B^m and $f \equiv g$ designates the equivalency between f and g which is realized by an XNOR operation. The consistency functions are represented in terms of BDD's. In this paper, we make no distinction between a set and its characteristic function.

The image of a subset A of B^n by f, $f(A)$, is a projection of a set $F(\mathrm{x}, \mathrm{y}) \cap A \times B^m$ on the space B^m. This is realized by means of the smoothing operator as follows since the projection of $F(\mathrm{x}, \mathrm{y})$ on the space B^m is obtained by smoothing the BDD for $F(\mathrm{x}, \mathrm{y})$ by the set of the input variables $\mathrm{x} = \{x_1, \cdots, x_n\}$.

$$f(A) = S_{\mathrm{x}}(F(\mathrm{x}, \mathrm{y}) \cap A(\mathrm{x})),$$

where $A(\mathrm{x})$ is the characteristic function of the set A. Hence the image computations of Boolean functions are completed with two BDD operations: an *and* and a *smooth*.

Similarly, the inverse image of a subset A of B^m by f, $f^{-1}(A)$, is obtained as

$$f^{-1}(A) = S_{\mathrm{y}}(F(\mathrm{x}, \mathrm{y}) \cap A(\mathrm{y})),$$

where $A(\mathrm{y})$ is the characteristic function of the set A and $\mathrm{y} = \{y_1, \cdots, y_m\}$ are the output variables of f.

4.1.2 The Recursive Image Computation Method

The other method for the image computations of Boolean functions employs another operator, originally introduced by Coudert *et al* [8], called the constraint operator. The operator is a generalization of the cofactor operation [2] and thus is also called the generalized cofactor [24].

The generalized cofactor extends the classical definition of a cofactor of a Boolean function by a cube [2] to a cofactor by an arbitrary non-null Boolean function.

Definition: The Generalized Cofactor

Given a Boolean function $f : B^n \rightarrow B$, a non-null Boolean function $c : B^n \rightarrow B$, and an oredering of the input variables $(x_1, x_2, \cdots, x_n)$, the cofactor of f with respect to c, f_c, is the function $f_c = f(\pi_c)$, where π_c is a mapping from B^n to B^n defined as follows:

$$\pi_c(\mathrm{x}) = \begin{cases} \mathrm{x} & \text{if } c(\mathrm{x}) = 1 \\ \arg \min_{\forall \mathrm{y} \in B^n, c(\mathrm{y})=1} d(\mathrm{x},\mathrm{y}) & \text{if } c(\mathrm{x}) = 0, \end{cases}$$

where

$$d(\mathrm{x},\mathrm{y}) = \sum_{i=1}^{n} |x_i - y_i| 2^{n-i}.$$

The generalized cofactor is defined depending upon a given ordering of the input variables, which is not a major restriction for an operation on BDD's. The mapping π_c projects a minterm x to a minterm y in the onset of c which has the closest distance to x according to the distance d. It is shown that y is defined uniquely given an ordering of the input variables and hence distance d. If f is a multiple output Boolean function, $f = \{f_1, \cdots, f_m\}$, then the cofactor of f with respect to c is defined as $f_c = \{(f_1)_c, \cdots, (f_m)_c\}$.

The generalized cofactor is reduced to the classical cube cofactor if c is a cube, since $\mathrm{y} = \pi_c(\mathrm{x})$ is defined regardless of the ordering of the input variables as

```
function cofactor(f,c)
begin
        if (c = 0)      return error;
        if (f = 1)      return 1;
        if (f = 0)      return 0;
        if (c = 1)      return f;
        if (c_{x̄1} = 0)  return cofactor(f_{x1}, c_{x1});
        if (c_{x1} = 0)  return cofactor(f_{x̄1}, c_{x̄1});
        return x1 cofactor(f_{x1}, c_{x1}) + x̄1 cofactor(f_{x̄1}, c_{x̄1});
end
```

Figure 2: The Generalized Cofactor Algorithm

follows:

$$y_i = \begin{cases} 1 & \text{if } c_i = 1 \text{ and } x_i = 0 \\ 0 & \text{if } c_i = 0 \text{ and } x_i = 1 \\ x_i & \text{if } c_i = 2 \text{ or } c_i = x_i. \end{cases}$$

Here x_i and y_i is the value of the i-th variable of the minterm x and y respectively while c_i is the value of the i-th variable of the cube c; $c_i = 0$ if the i-th variable appears complemented in c, $c_i = 1$ if the i-th variable appears not complemented in c and $c_i = 2$ if the i-th variable does not appear in c.

An algorithm for the generalized cofactor with BDD's is shown in Figure 2. The algorithm takes as input a BDD for f and a BDD for c, where the ordering of the input variables is identical, and returns a BDD of the cofactor of f with respect to c. The properties of the classical cube cofactor which the generalized cofactor also preserves are provided in the reference [24].

Using the generalized cofactor, the image of a Boolean function is computed recursively. The method is called the recursive image computation method. For

a given multiple output Boolean function $f : B^n \rightarrow B^m$ and a subset of B^n, A, we want to compute the image of A by f, $f(A)$ (also denoted $image(f, A)$). Let $c : B^n \rightarrow B$ be a Boolean function such that $c(A) = 1$ i.e. the characteristic function for A. Then the range of the cofactor of f with respect to c, $f_c(B^n)$ is identical with $f(A)$ since $f_c(B^n) = f(\pi_c(B^n))$ and $\pi_c(B^n)$ is the onset of c which is A. Therefore, the problem of computing the image of A by f is reduced to the problem of computing the range of f_c.

Let us denote $g = \{g_1, \cdots, g_m\}$ as the cofactor of f with respect to c. Our problem is to compute the range of g. Let $\{y_1, \cdots, y_m\}$ be the set of the output variables of g. Then due to the fact that

$$range(g) = y_1 image(\{f_2, \cdots, f_m\}, f_1^{-1}(1)) + y_1' image(\{f_2, \cdots, f_m\}, f_1^{-1}(0)),$$

we obtain the range of g defined recursively as

$$range(g) = y_1 range(\{(f_2)_{f_1}, \cdots, (f_m)_{f_1}\}) + y_1' range(\{(f_2)_{f_1'}, \cdots, (f_m)_{f_1'}\}).$$

Hence, the method performs cofactor operations recursively on the BDD's to complete the image computation.

4.2 Image Computations of Boolean Relations

We now introduce a method for the image computations of Boolean relations. We consider the situation provided in Section 3. Namely, for a given minterm $\mathrm{x} \in B^n$, let $r(\mathrm{x})$ be a set of minterms y such that $f^{(i)}(\mathrm{x}) = g^{(i)}(\mathrm{y})$ for $i = 1, \cdots, m$. The mapping r is a Boolean relation from B^n to B^t.

Definition: Image of a Boolean Relation

> Given a Boolean relation $r : B^n \rightarrow B^t$ and a subset A of B^n, the image of A by r, $r(A)$, is the set of minterms $\mathrm{y} \in B^t$ for which there exists a minterm $\mathrm{x} \in A$ such that $\mathrm{y} \in r(\mathrm{x})$.

Definition: Inverse Image of a Boolean Relation

Given a Boolean relation $r : B^n \rightarrow B^t$ and a subset A of B^t, the inverse image of A by r, $r^{-1}(A)$, is the set of minterms $\mathbf{x} \in B^n$ for which there exists a minterm $\mathbf{y} \in A$ such that $\mathbf{y} \in r(\mathbf{x})$.

The image or the inverse image computations for Boolean relations are completed by extending the transition relation method. We need to define a consistency function of a Boolean relation.

Definition: Consistency Function

Given a Boolean relation $r : B^n \rightarrow B^t$, the consistency function of r, $R(\mathbf{x}, \mathbf{y})$, is the characteristic function defined for a subset of $B^n \times B^t$ which consists of pairs of the minterms $(\mathbf{x}, \mathbf{y})$ such that $\mathbf{y} \in r(\mathbf{x})$.

The consistency function is represented in terms of a BDD as $R(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{m} (f^{(i)}(\mathbf{x}) \equiv g^{(i)}(\mathbf{y}))$. The image of a subset A of B^n by r is a projection of a set $R(\mathbf{x}, \mathbf{y}) \cap A \times B^t$ on the space B^t. Therefore, we obtain

$$r(A) = S_{\mathbf{x}}(R(\mathbf{x}, \mathbf{y}) \cap A(\mathbf{x})),$$

where $\mathbf{x}$ is a set of the input variables and $A(\mathbf{x})$ is the characteristic function of A. Note that if in particular A is a minterm in B^n, c, then $r(A)$ is computed by taking the cofactor of $R(\mathbf{x}, \mathbf{y})$ with respect to c: $r(A) = (R(\mathbf{x}, \mathbf{y}))_c$.

Similarly, the inverse image of a subset A of B^m by r, $r^{-1}(A)$, is obtained as

$$r^{-1}(A) = S_{\mathbf{y}}(R(\mathbf{x}, \mathbf{y}) \cap A(\mathbf{y})),$$

where $\mathbf{y}$ is a set of the output variables of r.

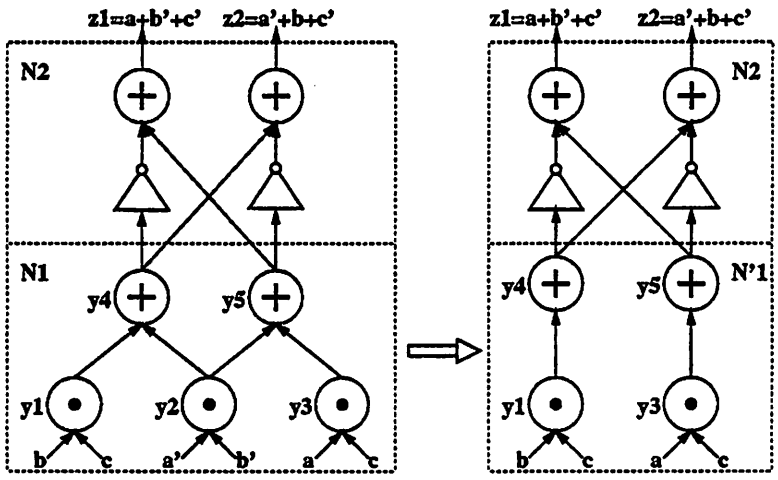

Figure 3: Example of the Insufficiency of Don't Care Sets

5 Boolean Relations

In this section, a review of Boolean relations is provided. We describe insufficiency of don't cares to represent incompletely specified Boolean functions and claim that Boolean relations are a generalization of incompletely specified Boolean functions. Both exact and heuristic algorithms for minimizing Boolean relations are presented.

5.1 Boolean Relations

Figure 3 is an example to show that don't cares do not provide enough information for optimally simplifying an incompletely specified Boolean function. The subnetwork N_1 can be simplified to N_1', as shown in the figure, without affecting the functional behavior of the primary outputs z_1 or z_2.

If don't cares are sufficient to represent incompletely specified Boolean func-

tions, then N_1 must be simplified to N_1' using don't cares associated with y_4 and y_5, denoted DC_4 and DC_5 respectively. In the simplification which eliminates y_2, the observability don't care [3] for y_2 needs to contain $a'b'$. The don't care for y_2, DC_2, is obtained as

$$\begin{aligned} DC_2 &= (((F_4)_{y_2} \equiv (F_4)_{y_2'}) + DC_4)(((F_5)_{y_2} \equiv (F_5)_{y_2'}) + DC_5) \\ &= (bc + DC_4)(ac + DC_5), \end{aligned}$$

where F_4 and F_5 are representations of Boolean functions for y_4 and y_5, respectively. Therefore, the only way that DC_2 contains $a'b'$ is that both DC_4 and DC_5 contain $a'b'$. This implies that y_4 can be simplified independently of y_5 and thus $y_4 = y_1 + y_2$ is simplified to $y_4 = y_1 = bc$. However, this simplification leaves $y_5 = ac + a'b'$ and the functionality of the primary output z_2 would be incorrect. Hence, the simplification from N_1 to N_1' cannot be completed only with don't cares associated with the N_1.

Let us consider a general case to find an expression which provides all the conditions under which a Boolean function is simplified. Given Boolean functions $f : B^p \rightarrow B^n$ and $g : B^q \rightarrow B^m$ with $n \leq q$, consider a function $h \stackrel{\text{def}}{=} g(f(\mathrm{x}), x_{p+1}, \cdots, x_r)$, where $r = p + q - n$ and g is a completely specified function. Then, h is a Boolean function from B^r to B^m.

For a given minterm of B^r, $\mathrm{x} = \{x_1, \cdots, x_p, x_{p+1}, \cdots, x_r\}$, let $r(\mathrm{x})$ be a set of minterms $\mathrm{y} \in B^n$ such that $g(\mathrm{y}, x_{p+1}, \cdots, x_r) = h(\mathrm{x})$. In general, r is a one-to-many mapping from B^r to B^n and is a *Boolean relation*. It is shown in [5] that the Boolean relation r provides all the conditions under which $f : B^p \rightarrow B^n$ is altered to another function $f^* : B^p \rightarrow B^n$ without affecting the functionality of h. Note that a Boolean relation r is reduced to an expression using don't cares only if $r(\mathrm{x})$ can be expressed as a single cube for each minterm $\mathrm{x} \in B^r$. In this sense, we see that don't care expressions are a special case of Boolean relations.

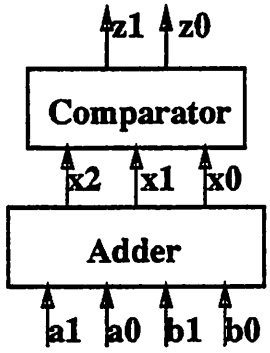

Figure 4: Hierarchical Network

5.2 Applications of Boolean Relations

As we have seen in the previous section, Boolean relations are a generalization of incompletely specified Boolean functions. Boolean relations arise in any context which manipulates incompletely specified Boolean functions.

5.2.1 Hierarchical Networks

A direct application of Boolean relations is hierarchical networks where a network with its function g is driven by another network f and other inputs $\{x_{p+1}, \cdots, x_r\}$. A simple example is a comparator driven by an adder as shown in Figure 4 [3, 23].

In Figure 4, the function of the adder, $f : B^4 \rightarrow B^3$, is $f = a + b$, while the function of the comparator, $g : B^3 \rightarrow B^2$, is given by:

$$z = \begin{cases} 01 & \text{if } a+b < 3 \\ 00 & \text{if } a+b = 3 \text{ or } a+b = 4 \\ 10 & \text{if } a+b > 4. \end{cases}$$

Therefore, $x_2x_1x_0 = \{000, 001, 010\}$ are not distinguished by the comparator. A Boolean relation associated with the adder is shown in Table 1.

$a_1a_0b_1b_0$	$x_2x_1x_0$
0000	{000, 001, 010}
0001	{000, 001, 010}
0010	{000, 001, 010}
0100	{000, 001, 010}
1000	{000, 001, 010}
0011	{011, 100}
0101	{000, 001. 010}
0110	{011, 100}
1001	{011, 100}
1010	{011, 100}
1100	{011, 100}
0111	{011, 100}
1011	{000, 001, 010}
1101	{011, 100}
1110	{000, 001, 010}
1111	{000, 001. 010}

Table 1: Boolean Relation of the Adder

The minimized representation of f by using only don't cares is shown in Table 2, while the minimization of the Boolean relation results in the smaller representation as shown in Table 3.

5.2.2 Finite State Machines

Another application of Boolean relations is finite state machines. Finite state machines may contain a number of equivalent states. Namely, for a given current state and a set of inputs, multiple states are implied by the machine. Usually equivalent states are removed using state minimization techniques. However, this can lead to suboptimal results. Equivalent states can be represented in terms of Boolean relations and the minimization or the restructuring of the machine based on Boolean

$a_1a_0b_1b_0$	$x_2x_1x_0$
$11-0$	011
-110	011
$10-1$	011
-011	011
-111	100
$11-1$	100
$111-$	010
$1-1-$	100

Table 2: The Minimized Representation using Don't Cares

$a_1a_0b_1b_0$	$x_2x_1x_0$
$0-1-$	010
$1-0-$	010
$1-1-$	100
$---1$	001
$-1--$	001

Table 3: The Minimized Representation using Boolean Relation

relations may result in a smaller machine which otherwise could not be achieved [14]. In implementing a finite state machine, it is crucial to remove redundancies to obtain a highly testable circuit. A fault that causes interchanges and/or creates equivalent states is called an equivalent sequentially redundant fault (SRF). It is shown that equivalent-SRF's are removed through logic minimizations based on Boolean relations and an algorithm for this problem has been proposed [9].

5.3 Minimizing Boolean Relations

The problem of minimizing Boolean relations is to find a Boolean function, f, compatible with a given Boolean relation r such that the minimum representation of f is minimum among all the functions compatible with r. Recall that a Boolean function $f : B^r \rightarrow B^n$ is compatible with a Boolean relation r if $f(\mathrm{x}) \in r(\mathrm{x})$ for each minterm $\mathrm{x} \in B^r$. An exact and a heuristic approach for the problem have been proposed [11, 23]. Both approaches are based on the two level minimization. We provide a brief review of both methods.

5.3.1 Exact Minimization of Boolean Relations

The observation that the minimum implementation of a Boolean relation in disjunctive form consists of prime implicants of some compatible function leads to the exact minimizer [23], which is based on an extension of the Quine-McCluskey method [17]. For a given Boolean relation from B^r to B^n, a prime implicant of a Boolean function compatible with the relation r is called a *c-prime* of r. The first step is to generate all the c-primes. Instead of finding all the compatible functions and generating primes for each function, all the c-primes of r can be derived by means of a type of iterated consensus operation starting with the set of all the implicants whose input part is a minterm $\mathrm{x} \in B^r$ and the output part is $r(\mathrm{x})$. After the set of the c-primes $\{c_1, \cdots, c_t\}$ is obtained, a compatible function with the least cost is chosen. This is formulated as a $0-1$ integer linear programming problem: minimize $C = \sum_{j=1}^{t} w_j \alpha_j$ subject to $\forall \mathrm{x} \in B^r : f(\mathrm{x}) \stackrel{\text{def}}{=} \sum_{j=1}^{t} \alpha_j c_j \in r(\mathrm{x})$, where $\alpha_j \in B$ and w_j is a cost of c_j.

Let T be a matrix representation of a Boolean function of $\{\alpha_1, \cdots, \alpha_t\}$, which is the product of the 2^r constraints shown above. Then the problem is reduced to the following problem:

Find a subset of columns of T, S, with the minimum cost C such that

for each row T_i, either

1. $\exists j : t_{ij} = 1$ and $T_j \in S$ or
2. $\exists j : t_{ij} = 0$ and $T_j \notin S$,

where T_j is the j-th column of T.

This is a binate covering problem [23] and the minimum representation of the Boolean relation is obtained by solving the problem. This procedure is exponential both in terms of CPU time and memory space required to store c-primes.

5.3.2 Heuristic Minimization of Boolean Relations

The heuristic Boolean relation minimizer [11] is based on an extension of Espresso [2]. The idea is that starting from a Boolean function compatible with r, the expand, irredundant, and reduce procedures are iteratively applied to find a compatible function with less cost until no improvement is observed. The method makes use of test pattern generation techniques in each procedure. The first step is to create a two level AND-OR network, called the interconnected network, in which a subnetwork for an initial function compatible with r, driver PLA, drives another network, driven PLA, so that the given relation is realized. The problem is to alter the driver PLA without affecting the functional behavior of the interconnected network. Then the expand procedure is applied to the driver PLA in which a redundant literal is removed for each cube. Unlike Espresso, however, the function of the driver PLA does not have to be preserved as long as the new function is still compatible with r. Therefore, in the domain of Boolean relations, a literal is redundant if the removal of the literal does not affect the functionality of the interconnected network. Thus, the expand procedure removes a literal if a stuck-at-1 fault for the literal is undetectable in the interconnected network. Followed by the expand procedure is the irredundant procedure which removes redundant cubes.

Similarly to expand, a cube in the driver PLA is redundant if the removal of the cube does not affect the functionality of the interconnected network. Therefore, from the point of view of the test pattern generation, an AND gate in the driver PLA can be removed if a stuck-at-0 fault at the output of the gate is undetectable in the interconnected network. The reduce procedure, which follows the irredundant procedure, tries to make each prime cube non-prime, hoping that the resulting representation may lead to a better prime and irredundant representation after re-applying the expand and irredundant procedures. Specifically, for each AND gate in the driver PLA, a literal which is not connected to the gate is added one by one as input of the gate to see if a stuck-at-1 fault on the literal is detectable. Then the literal is retained only if it is undetectable in the interconnected network. These procedures are iterated as long as the cost of the representation for the driver PLA is decreasing. The experimental results show that this method works in reasonable time even for large examples.

6 Development of an Automatic Rectification System

A preliminary version of an automatic rectification system has been implemented as a package of misII [4]. In this section, we discuss the overall flow of the rectification process. We also discuss some heuristics used to keep the memory requirements small, and some results which make the procedure more efficient. The system takes as input two Boolean networks corresponding to the correct and incorrect designs. A modified logic verification algorithm [16] is used to find a set of minterms of B^n, denoted M, for which rectification is necessary. M is represented in terms of a BDD. Note that if a minterm $x \in B^n$ is not in M, then the original network completes the correct result. In order to keep the Boolean relation small for representation and use in the Boolean relation minimizer, we construct a Boolean relation, r, which is a restricted form of the one introduced in

Section 3. Specifically, r is defined as follows:

$$r(\mathbf{x}) = \begin{cases} \{\mathbf{y} \in B^t \mid f^{(i)}(\mathbf{x}) = g^{(i)}(\mathbf{y}) i = 1, \cdots, m\} & \text{if } \mathbf{x} \in M \\ \mathbf{y} \text{ such that the value of } y_j \text{ is equal to the value of } x_i & \text{if } \mathbf{x} \notin M, \end{cases}$$

where x_i is a primary input originally connected to the primary input line y_j.

We now formalize this construction and give a procedure for creating r. The consistency function, $\tilde{R}(\mathbf{x}, \mathbf{y})$, of the restricted Boolean relation r is computed as follows. Let $w(x_i)$, for each primary input x_i, be a set of the variables of the primary input lines that were originally connected to x_i. Let W be a set of minterms of B^t such that for each x_i, all the variables in $w(x_i)$ take the same value. In other words, W is represented as $W = \prod_{i=1}^{n} (\prod_{y_j, y_k \in w(x_i)} y_j \equiv y_k)$. Then $\tilde{R}$ is given by the union of $R \cap M \times B^t$ and $R \cap M' \times W$, where M' is the complement of M and R is a consistency function defined as $R = \prod_{i=1}^{m} (f^{(i)}(\mathbf{x}) \equiv g^{(i)}(\mathbf{y}))$. The consistency function $\tilde{R}$ is represented in terms of a BDD as $\tilde{R} = RM + RW$. In practice, instead of computing $r(\mathbf{x})$ for each $\mathbf{x} \in B^n$, we first compute $\tilde{R}(\mathbf{x}, \mathbf{y})$ and then obtain $r(\mathbf{x})$ by taking the cofactor of $\tilde{R}$ with respect to each minterm $\mathbf{x}$. Note that the fundamental Boolean operations such as AND or OR, as well as the cofactor operation are completed very efficiently on BDD's. Once $r(\mathbf{x})$ is determined for all $\mathbf{x}$, it is minimized by externally calling the exact Boolean relation minimizer [23] to obtain an attached network before the primary inputs.

Note that if $r(\mathbf{x})$ is empty for some $\mathbf{x} \in M$, then there is no hope to complete rectification by attaching a network before the primary inputs only. The possibility of the rectification is checked by the following:

Theorem 6.1 *Given a Boolean relation $r : B^n \rightarrow B^t$ and a set $A \subseteq B^n$, there exists a minterm $\mathbf{x} \in A$ such that $r(\mathbf{x}) = \phi$ if and only if $A \not\subseteq r^{-1}(r(A))$.*

Corollary 6.1 *A set of minterms $\mathbf{x}$ of A such that $r(\mathbf{x}) = \phi$ is given by $A \cap (\neg r^{-1}(r(A)))$.*

Hence, the rectification is completed by means of the network attached before the primary inputs if and only if M is contained in $r^{-1}(r(M))$. Note that this check is done very efficiently by the transition relation method in the early stage of the rectification process.

If a minterm x such that $r(\mathrm{x}) = \phi$ exists in M, the system sets $r(\mathrm{x})$ in the same way as the case where $\mathrm{x} \notin M$. Namely, according to the above corollary, M is replaced by $M \cap r^{-1}(r(M))$. It then creates the input network η_I as above and creates a network η_O after the primary outputs of the incorrect network η as follows. For each primary output z_i of η, $i = 1, \cdots, m$, let E_i (respectively I_i) be a set of minterms of the primary inputs, x, such that $r(\mathrm{x}) = \phi$ and must be excluded from (respectively included in) the on-set of the function of z_i to complete the rectification. E_i and I_i are obtained as BDD's when M is computed. Then η_O is an m output network where the function of the i-th output is given by $(z_i + I_i)E_i'$. Finally, the system produces the two networks η_I and η_O as output. In practice, our limited examples have not led to a case where a network after the primary outputs was necessary.

Currently this method is exponential both in CPU time and memory space due to the exponential complexity of the exact minimizer of Boolean relations. For example, the Boolean relation minimizer requires that $r(\mathrm{x})$ is explicitly given for each minterm x in a table format. Thus to be truly practical, it is desirable to obtain a BDD based heuristic minimizer for Boolean relations. We are presently evaluating the method in [11].

7 Conclusion

We have introduced the problem of rectifying design incorrectness caused by specification changes and/or design errors. A problem formulation and a basic approach for the problem have been presented. A necessary and sufficient condition

has been derived under which the rectification can be completed by means of a network attached before the primary inputs. We have discussed the close relationship between the rectification problem and the image computation of Boolean mappings as well as the minimization of Boolean relations. Current results for both issues show that the methods proposed so far require various improvement. Specifically, in the image computations, the BDD sizes for the consistency functions must be kept small during the operation. In the minimization of Boolean relations, we need a heuristic method which results in small representations comparable to the exact procedure. The development of computer-aided-rectification (CAR) of VLSI's will depend upon the continuing development of these related and important problems.

8 Acknowledgements

The authors wish to thank F. Somenzi for providing a program for Boolean relation minimization. Helpful discussions with A. Ghosh and H. Touati are gratefully acknowledged. This research is supported in part by the National Science Foundation and the Defense Advanced Research Projects Agency under contract number NSF/DARPA-MIP-871-9546.

References

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, Orlando, June 1990.

[2] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, 1984.

[3] R. K. Brayton, G. D. Hachtel, and A. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of The IEEE*, Vol. 78(No. 2), February 1990.

[4] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: Multiple-Level Logic Optimization System. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-6(No. 6):1062 – 1081, November 1987.

[5] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *International Conference on Very Large Scale Integration*, Munich, August 1989.

[6] R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, Vol. C-35(No. 8):677–691, August 1986.

[7] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L Dill. Sequential Circuit Verification using Symbolic Model Checking. In *27th ACM/IEEE Design Automation Conference*, Orlando, June 1990.

[8] O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines Based on Symbolic Execution. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.

[9] S. Devadas, H. K. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Irredundant Sequential Machines via Optimal Logic Synthesis. *IEEE Transactions on Computer Aided Design*, pages 8–18, January 1990.

[10] A. Ghosh, S. Devadas, and A. R. Newton. Test Generation for Highly Sequential Circuits. In *IEEE International Conference on Computer-Aided Design*, pages 362–365, November 1989.

[11] A. Ghosh, S. Devadas, and A. R. Newton. Heuristic Minimization of Boolean Relations using Testing Techniques. In *IEEE International Conference on Computer Design*, Cambridge, September 1990.

[12] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In R. Bryant, editor, *3rd Caltech Conference on Very Large Scale Integration*, pages 87–116, 1983.

[13] B. Lin and A. R. Newton. Restructuring State Machines and State Assignment: Relationship to Minimizing Logic Across Latch Boundaries. In *International Workshop on Logic Synthesis*, May 1989.

[14] B. Lin and F. Somenzi. Minimization of Symbolic Relations. In *IEEE International Conference on Computer-Aided Design*, November 1990.

[15] J. C. Madre, O. Coudert, and J. P. Billon. Automating the Diagnosis and the Rectification of Design Errors with PRIAM. In *IEEE International Conference on Computer-Aided Design*, November 1989.

[16] S. Malik, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification using Binary Decision Diagrams in a Logic System Environment. In *IEEE International Conference on Computer-Aided Design*, pages 6–9, November 1988.

[17] E. J. McCluskey Jr. Minimization of Boolean Functions. *Bell System Technical Journal*, Vol. 35:1417–1444, November 1956.

[18] R. McGeer. *On the Interaction of Functional and Timing Behavior of Combinational Logic Circuits*. PhD thesis, U.C. Berkeley, November 1989.

[19] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. In *27th*

ACM/IEEE Design Automation Conference, pages 52–57, Orlando, June 1990.

[20] S. Muroga and T. Ibaraki. Synthesis of Networks with a Minimum Number of Negative Gates. *IEEE Transactions of Computers*, Vol. C-20:49–58, January 1971.

[21] S. Muroga, Y. Kambayashi, C. H. Lai, and J. N. Culliney. The Transduction Method - Design of Logic Networks based on Permissible Functions. *IEEE Transactions of Computers*, 1989.

[22] R. B. Segal. Bdsyn:logic description translator bdsim:switch-level simulator. Technical Report UCB/ERL M87/33, University of California, Berkeley, May 1987.

[23] F. Somenzi and R. K. Brayton. An Exact Minimizer for Boolean Relations. In *IEEE International Conference on Computer-Aided Design*, November 1989.

[24] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *IEEE International Conference on Computer-Aided Design*, November 1990.