CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Texas A&M Repository

A ROBUST WINDOW-BASED MULTI-NODE MINIMIZATION TECHNIQUE USING BOOLEAN RELATIONS

A Thesis

by

JEFFREY COBB

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2007

Major Subject: Computer Engineering

A ROBUST WINDOW-BASED MULTI-NODE MINIMIZATION TECHNIQUE USING BOOLEAN RELATIONS

A Thesis

by

JEFFREY COBB

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Sunil Khatri
Committee Members,	Riccardo Bettati
	Weiping Shi
Head of Department,	Costas Georghiades

December 2007

Major Subject: Computer Engineering

ABSTRACT

A Robust Window-Based Multi-Node Minimization Technique
Using Boolean Relations. (December 2007)
Jeffrey Cobb, B.S., Texas A&M University
Chair of Advisory Committee: Dr. Sunil Khatri

Multi-node optimization using Boolean relations is a powerful approach for network minimization. The approach has been studied in theory, and so far its superiority over single node optimization techniques has only been conjectured for practical designs. This is due to the highly memory intensive computations involved in the calculation of Boolean relations representing the multi-node optimization flexibility. In this thesis, an algorithm to perform Boolean relation-based multi-node optimization using a robust, fast and memory efficient algorithm is presented. In particular, two nodes are simultaneously optimized at a time. Results are reported on large designs, demonstrating the initial power of this multi-node optimization algorithm. The robustness of the approach arises from the use of a window-based technique for computing these Boolean relations. Secondly, aggressive early quantification is performed during the computation, keeping memory utilization low. Finally, smart heuristics are employed for selecting the node pair to be optimized simultaneously. These features allow the approach to scale well and provide good results for large designs. Experiments are performed on a set of large benchmarks and the algorithm's performance is compared to a SAT-based network optimization technique using complete don't cares. On average, the approach presented in this thesis achieves a 12% reduction in literal count across all the large designs compared to the complete don't cares, while maintaining small runtimes and low memory usage.

To my parents and my grandparents

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The optimization of industrial multi-level Boolean networks is traditionally performed using algebraic techniques. The main reason for this is that Boolean techniques such as don't care-based optimization, though more powerful, do not scale well with design size. Don't cares are calculated for a single node, and they specify the flexibility for implementing the node function. These don't cares (for a node) are computed using a combination of Satisfiability Don't Cares (SDCs), Observability Don't Cares (ODCs) or External Don't Cares (XDCs). These are described further in [1].

ODCs [2, 3] of a node are a powerful representation of the node's flexibility. However, the minimization of a node with respect to its ODCs can potentially change the ODCs at other nodes in the circuit, resulting in a need to re-compute ODCs for all circuit nodes. A subset of ODCs, termed as Compatible Observability Don't Cares (CODCs) [2] were formulated to remove this limitation. By definition, if a node n is minimized with respect to its CODCs, then the CODCs of all other circuit nodes are still valid (and therefore do not need to be recomputed). However, in the CODC computation, the order of selecting nodes during the CODC computation becomes important. The maximum flexibility that can be obtained at the fanin node i of a node n is a function of the CODCs of the fanins computed prior to i. In both the ODC and CODC approaches, network optimization is performed on one node at a time.

As significant improvement (in terms of optimization power) over don't care-based techniques can be obtained by considering *multiple nodes at once*. The formulation of such an optimization results in a Boolean relation [4], which implicitly represents the flexibility

---

The journal model is *IEEE Transactions on Automatic Control.*

available in optimizing the nodes *simultaneously*. The flexibility inherent in multi-node optimization cannot be expressed using functions. Table I.1 represents a Boolean relation,

Table I.1. Example of a Boolean Relation

Inputs	Outputs
00	00
01	01
10	{00,11}
11	10

which, for a single input vector $\{10\}$, can express more than one *allowed* output vector, $\{00,11\}$. On the other hand, no Boolean function can represent the fact that both vectors $\{00,11\}$ are allowed at the outputs, for the output $\{10\}$.

The superiority of a multi-node optimization approach (using Boolean relations) over don't cares has been pointed out in [5, 6]. The reason for this superior optimization flexibility is that in the computation of a node's don't cares, the functions of all the other nodes are *not* allowed to change. This restriction does not apply to the multi-node optimization approach (using Boolean relations) since they allow the *simultaneous* modification of all nodes being targeted. However, this superior optimization flexibility has a price. The multi-node optimization approach requires that a Boolean relation be solved, which is typically a highly time and memory intensive operation. As a result, not much attention has been devoted to these approaches, although there have been theoretical works which have suggested the superiority of this technique over don't care-based approaches [6]. However there has been no robust, scalable approach which demonstrates the applicability of multi-node optimization techniques to large designs.

The power of a multi-node optimization approach can be illustrated by way of a small example [7]. Consider the network η shown in Figure I.1(a), where node V_1's output f

implements the Boolean function $x \cdot y$ and node V_2's output g implements $x + y$. Given a network η with primary outputs Z, the ODC of a node y is given by

$$ODC(y) = \prod_{z_i \in PO(\eta)} \overline{(\frac{\partial z_i}{\partial y})}$$

where,

$$\frac{\partial z_i}{\partial y} = z_i|_y \oplus z_i|_{\overline{y}}.$$

Using this equation for the network of Figure I.1, the ODCs are computed to be $ODC(V_1)$ = $ODC(V_2)$ = $\varnothing$. As a result, no optimization is possible using ODCs. However, one can observe that z is equivalent to $x \oplus y$ as shown in Figure I.1(b). This optimization can *only* be obtained when V_1 and V_2 are optimized simultaneously. The Boolean relation resulting from such an optimization can express this flexibility. After minimizing this Boolean relation, nodes V_1 and V_2 can be deleted from the network without compromising the network's functionality.

The following sections discuss the problem definition, previous work, and the organization of this thesis.

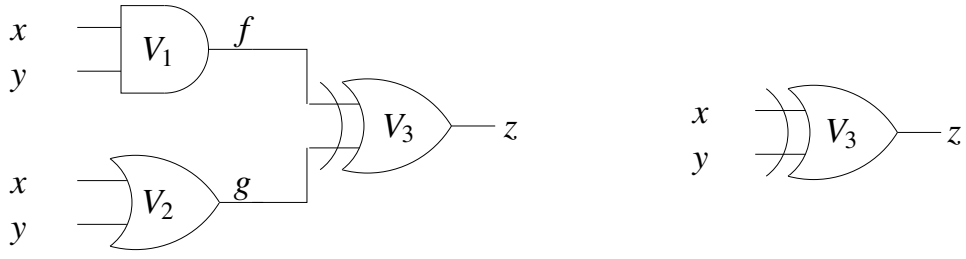

Fig. I.1. Network η before and after optimization

I-A. Problem Definition

This thesis addresses the problem of creating a scalable dual-node optimization technique which can handle the large designs typical of industrial circuits. The approach presented in this thesis uses Reduced Ordered Binary Decision Diagrams (ROBDDs) [8, 9] to represent the Boolean relations expressing the flexibility. ROBDDs provide an efficient mechanism to compute and store the Boolean relation. Many of the steps that occur frequently in the dual-node optimization technique, including quantification and complementation, can be performed extremely efficiently using this data structure.

Large designs can have tens of thousands of nodes in the network, which would make the task of computing the Boolean relation that represents the dual-node flexibility impractical due to the computational time and/or memory required. To address this problem, the relation is not built in terms of the primary inputs and outputs of the network, but it is instead built using a subnetwork η' rooted around the nodes being targeted. Building the relations in terms of the primary inputs and outputs would be applicable to small networks. However, the ROBDDs representing the relation would be intractable for larger networks. Working with the subnetwork allows the resulting Boolean relation to be *significantly* smaller, which enables the approach to work on large networks.

Another feature of this approach, which allows it to scale elegantly, is that it uses *aggressive early quantification* [10] while computing the Boolean relation. During the computation, the size of the ROBDDs can blow up rapidly if the relevant intermediate variables are not quantified out. Performing operations on the large ROBDDs can be very expensive in time as well as memory, especially on large networks. To reduce the size of the ROBDDs, intermediate variables are quantified out during each step of the computation when possible.

Additionally, since the work presented in this thesis optimizes two nodes at a time, the

node pair must be carefully selected. Optimizing all node pairs in a network would result in a quadratic cost. By choosing only those node pairs which have a high likelihood of minimizing the network, the algorithm remains efficient for large designs.

Finally, the dual-node optimization approach in this thesis results in a Boolean relation that encodes the flexibility in implementing the targeted nodes. To re-implement the targeted functions, this relation needs to be minimized. The technique used to do this comes from [11].

I-B. Previous Work

Some of the previous research efforts which are relevant to the technique and objective of this thesis are discussed next. In [12], the authors describe a method to compute don't cares using overlapping subnetworks, computed using a varying window size. Their method does not optimize wires, but only the gates in a design, in contrast to the approach in this thesis (which frequently removes wires in a circuit). Further, the technique of [12] uses [13] to optimize a single subnetwork. In [13], optimization is done by manipulating a cover of the subnetwork explicitly. The authors indicate that this requires large amounts of runtime for small networks. As a consequence, the technique of [12], in many examples, requires run-times which are dramatically larger than MIS [14].

The approach of [15] partitions the circuit into subnetworks, each of which is flattened and optimized using ESPRESSO [16]. This thesis uses a similar approach of circuit partitioning but with a relation-based optimization method in place of ESPRESSO, and achieves a significantly lower literal count.

In [17], the CODC computation of [18] was shown to be dependent on the current implementation of a node, and an implementation-independent computation was proposed. In [19], the authors perform CODC computation on overlapping subnetworks, and demon-

strate a faster technique compared to the full CODC computation. They report achieving a good literal count reduction (within 10% of the *full_simplify* (FS) command of SIS [20]) with a faster runtime (25x faster than FS). The method presented in this thesis improves on these results due to the additional flexibility encoded in the dual-node optimization technique (using Boolean relations).

In [21], the authors present a Boolean Satisfiability (SAT) based methodology for computing the ODC and SDC, termed as complete DC (CDC), for a every node in a network. They also propose a windowing scheme to maintain robustness. This approach provides the best results in terms of both optimization ability and runtime among all the previous single-node approaches mentioned here.

While [21] explores the flexibility of exactly one node at a time, a much greater flexibility can be availed by optimizing multiple nodes of a network simultaneously. This is a relatively unexplored aspect of multi-level optimization. There are research efforts which recognize the power of such a technique [5, 22, 6], but none of these work on even medium sized circuits. The survey described in [5] only points out the advantage of multi-node minimization over don't cares. The approach in [22] describes a BDD-based computation of SPFDs [23], which can encode the flexibility of more than one node, but it is limited to small circuits and shown not to be scalable to large designs.

In [6], an approach for computing the Boolean relation of a single subnetwork of the original network is described. However no approach or intuition for selecting the subnetwork is discussed. The approach in this thesis, in contrast, uses an efficient method to find pairs of nodes to optimize together. This method effectively filters out pairs of nodes for which the expected flexibility is low. Also, the results reported in [6] are for very small circuits, and incur extremely high runtimes. The implementation in this thesis is powerful and robust, resulting in the ability to optimize large networks extremely fast, with a high quality of results. Further, [6] does not use a relation minimizer, but instead it calls ESPRESSO

in order to minimize the Boolean relation that represents the optimization flexibility. The authors do acknowledge this as a possible limitation in their paper. The work in this thesis uses BREL [11] to minimize the Boolean relation which is constructed for each pair of nodes being optimized simultaneously.

There are some earlier research efforts in the context of multi-node optimization (using Boolean relations to express the multi-node optimization flexibility), but the approach in this thesis is very different. A technique which calculates this Boolean relation in terms of primary inputs is presented in [7]. The work in this thesis computes this Boolean relation in terms of the 'primary input' variables *of the extracted subnetwork*, allowing the technique to handle large designs.

A technique to compute the maximal Boolean relation that represents the optimization flexibility for the nodes in an arbitrary subnetwork is presented in [24], which was improved by [25] to additionally compute approximate Boolean relations. However, they do not support their work with experimental results.

The main reasons why the approach in this thesis is significantly more efficient than these methods are:

- The intelligent selection of node pairs to minimize, in order to maximize the likelihood of gain, without exhaustively trying all possible pairs.
- The use of a window-based computation which allows the approach to scale very elegantly and provide excellent results even for large designs.
- An aggressive use of quantification scheduling to control the ROBDD size at all stages of the Boolean relation computation.

Techniques for minimizing a Boolean relation are reported in [11, 26, 27]. In [26] the authors represent a Boolean function as a multi-valued decision diagram and pro-

pose a heuristic to minimize it. The authors of [27] formulate the problem of minimizing a Boolean relation as a binate-covering problem. The more recent approach used in BREL [11] follows a recursive branch-and-bound heuristic for minimizing a given Boolean relation. This approach demonstrates better results and runtimes as compared to those reported in [26, 27]. Therefore the work in this thesis uses BREL [11] for minimizing the Boolean relation that is computed. The details of the BREL algorithm are described in the next chapter.

I-C. Organization

This thesis is organized as follows. Chapter II provides the necessary background knowledge and terminology used in this thesis, as well as a description of the data structures used. In Chapter III, the algorithms used to select node pairs and to compute the Boolean relation expressing the flexibility of a two-node optimization are given. The windowing method is also described in this chapter, as well as the early quantification technique is employed when computing the relation.

Chapter IV presents the experimental results for the algorithms described in Chapter III. Section IV-C shows the approach used to choose the parameters that control the node selection algorithm. Section IV-D reports the results obtained when the proposed method is compared with the *mfsw* approach of [21]. Section IV-E presents results of many variations on the original approach, as well as a timing and quantification analysis. Section IV-F shows relevant statistics about which node pairs yielded gains and which did not.

Chapter V summarizes the thesis and discusses the implications of the experimental results. Ideas for future improvements are also presented in this chapter.

CHAPTER II

BACKGROUND

II-A. Chapter Overview

This chapter introduces the terminology and data structures used throughout this thesis. Section II-B provides some preliminaries and definitions. Section II-C explains the Reduced Ordered Binary Decision Diagram (ROBDD) data structure and some of the ROBDD functions used in the algorithms in this thesis. Section II-D describes the algorithm of BREL used to minimize Boolean relations. The chapter is summarized in Section II-E.

II-B. Preliminaries and Definitions

The goal of this thesis is to reduce the size and complexity of a Boolean network at the technology independent level. A Boolean network is defined as the following:

Definition 1 *A Boolean network* η *is a directed acyclic graph (DAG)* $G = (V,E)$ *in which every node has a Boolean function* f_i *associated with it. Also,* f_i *has a corresponding Boolean variable* y_i *associated with it, such that* $y_i \equiv f_i$.

There is a directed edge $e_{ij} \in E$ from y_i to y_j if f_j explicitly depends on y_i.

A node y_i is a *fanin* of a node y_j iff there exists a directed edge $e_{ij} \in E$. Node y_i is a *fanout* of y_j iff there exists a directed edge $e_{ji} \in E$. $FI(y)$ and $FO(y)$ represent the set of fanins and the set of fanouts of y respectively. $FI(y)$ and $FO(y)$ are equivalently referred to as *immediate fanins* and *immediate fanouts* respectively.

A node y_i is in the *transitive fanin* of a node y_j if there is a directed path from y_i to y_j. Node y_i is in the *transitive fanout* of node y_j if there is a directed path from y_j to y_i. The transitive fanin of a node y_i up to a k levels, $TFI(y_i,k)$, is the set of nodes $\{y_j\}$ such that

there is a directed path of length less than or equal to k, between y_j and y_i. Similarly, the transitive fanout of a node $TFO(y_i,k)$ is the set of nodes $\{y_j\}$ such that there is a directed path of length less than or equal to k, between y_i and y_j.

The *transitive fanin frontier* of a node y_i at k levels, $TFI_{frontier}(y_i,k)$, is the set of nodes $\{y_j\}$ such that there is a directed path of length exactly equal to k, between y_j and y_i. The *transitive fanout frontier* of a node $TFO_{frontier}$ is the set of nodes $\{y_j\}$ such that there is a directed path of length exactly equal to k, between y_i and y_j.

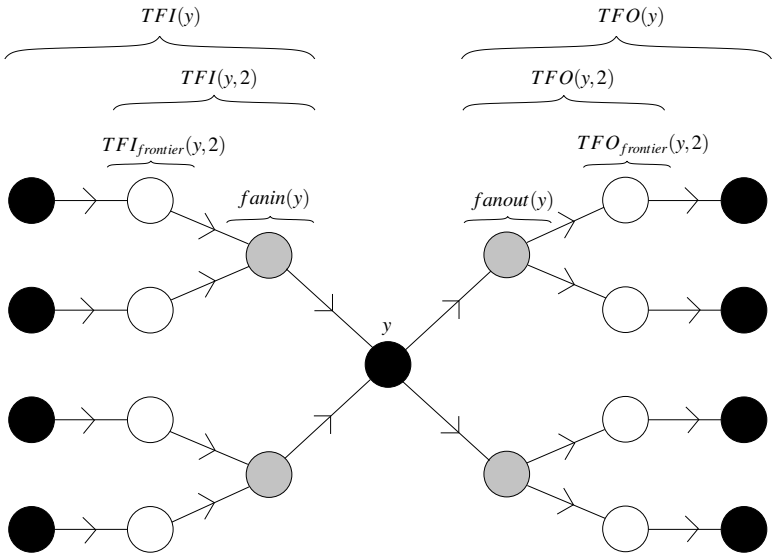

Fig. II.1. Terminology for Nodes in a Boolean Network

These definitions are illustrated in Figure II.1. The gray nodes are the immediate fanins and fanouts of the node y. The white nodes represent nodes in the $TFI_{frontier}$ and $TFO_{frontier}$ of node y. The nodes of $TFI(y,2)$ and $TFO(y,2)$ are also shown, as well as the nodes in $TFI(y)$ and $TFO(y)$. These classifications are used extensively in Chapter III of this thesis.

Definition 2 *The* **consensus operator** *or* **universal quantification** *of a function f with respect to a variable x_i is*

$$\forall_{x_i} f = f_{x_i} \cdot f_{\overline{x_i}}$$

Definition 3 *The* **smoothing operator** *or* **existential quantification** *of a function f with respect to a variable x_i is*

$$\exists_{x_i} f = f_{x_i} + f_{\overline{x_i}}$$

A Boolean relation is used to represent the flexibility available in optimizing multiple nodes simultaneously. Related definitions are given next.

Definition 4 *A* **Boolean relation** $\mathcal{R}$ *is a one-to-many multi-output Boolean mapping, $\mathcal{R}$: $B^n \rightarrow B^m$.*

An output vector $y^l \in B^m$ is *allowed for* an input vector $x^k \in B^n$ iff $(x^k,y^l) \in \mathcal{R}$.

Definition 5 *A* **multi-output Boolean function** *f is a mapping compatible with $\mathcal{R}$ if $f(x) \in \mathcal{R}$, $\forall_x \in B^n$. This is denoted by $f \prec \mathcal{R}$.*

A Boolean relation $\mathcal{R}$ can be represented by its *characteristic function* $\Phi : B^n \times B^m \rightarrow B$ such that $\Phi(x^k, y^l) = 1$ iff $(x^k, y^l) \in \mathcal{R}$.

For a network η which implements the multi-output Boolean function $\boldsymbol{z} = f(\boldsymbol{x})$, the characteristic function is denoted by Φ^η, where

$$\Phi^\eta = \prod_{k=1}^{m} (z_i \overline{\oplus} f_{z_i}(\boldsymbol{x}))$$

where m is the number of outputs of η and $f_{z_i}(\boldsymbol{x})$ is the function of z_i in terms of $\boldsymbol{x}$.

Note that in the sequel a set of variables $\{a\}$ is represented as $\boldsymbol{a}$.

II-C. Reduced Ordered Binary Decision Diagrams (ROBDD)

A binary decision graph (BDD) is a rooted, directed, acyclic graph containing decision nodes and two terminal nodes, 0 and 1. Each node has two children called the low child and high child. The low child corresponds to the negative Shannon cofactor of the node, and the high child corresponds to the positive Shannon cofactor of the node (both with respect to the variable associated with the node). The value of the function for a particular assignment of variables is found by tracing a path from the root of the tree to a terminal node, following the appropriate child from each node, as dictated by the assignment of variables. There is no particular ordering for the variables, so different path may contain variables in a different order. This means that BDDs are not canonical, in that there are multiple ways to represent the same Boolean function. Figure II.2 shows the BDD of the logic function $x_1 + \overline{x_2} \cdot x_3$. The dashed line goes to the low child, and the solid line goes to the high child. Notice that the variable ordering for the high child of x_1 is different from that of the low child. In addition, all nodes from the high child of x_1 reach the 1 terminal node, revealing redundancies in this representation.

A reduced ordered binary decision diagram (ROBDD) is a type of BDD with special properties. First, the ROBDD is obtained by removing redundancies in the BDD structure. Specifically, nodes whose children are identical are removed, and children that have identical subtrees are merged. An example of this is shown in Figure II.3. The ROBDD for the same function $x_1 + \overline{x_2} \cdot x_3$ is shown in this figure, which shows that a fewer nodes are required compared to Figure II.2. Second, an ROBDD is ordered. In other words, a fixed variable ordering is used along any path from root to leaves. These properties ensure that an ROBDD is canonical, in that there is only one way to represent a Boolean function for a particular variable ordering. This allows the checking of equivalence between two ROBDDs to be performed in constant time. In addition, complementation, which is non-trivial

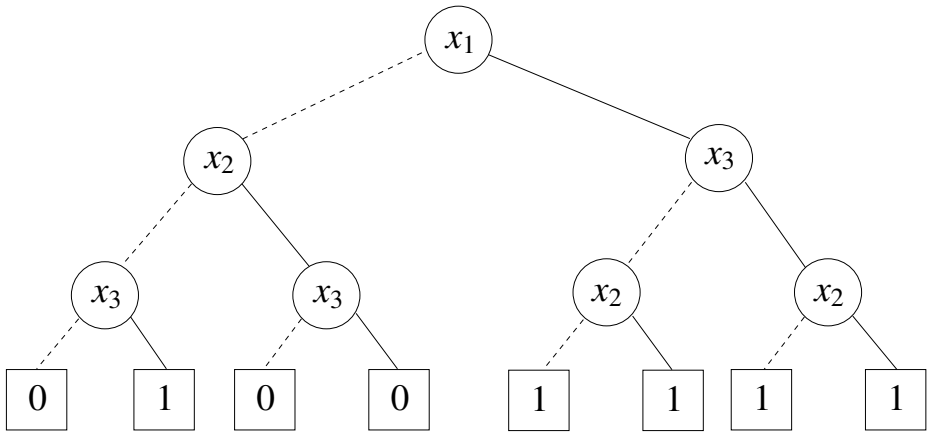

Fig. II.2. BDD of Logic Function $x_1 + \overline{x_2} \cdot x_3$

for sum-of-product representations, take constant time as well, since only the values of the terminal nodes must be complemented. These properties allow many computations to be performed much faster using ROBDDs compared to other representation schemes. However, some functions such a multiplier require an exponential number of ROBDD nodes regardless of the variable ordering used.

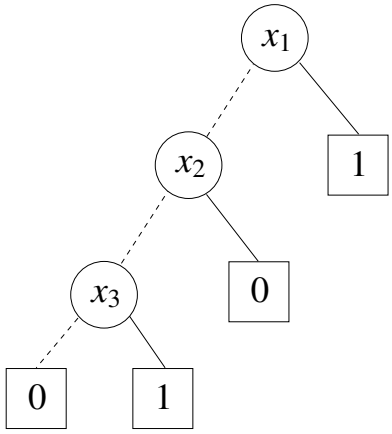

Fig. II.3. ROBDD of Logic Function $x_1 + \overline{x_2} \cdot x_3$

The size of an ROBDD in terms of the number of nodes is heavily dependent on the variable ordering chosen. Figure II.4 shows an alternate variable ordering for an ROBDD of the function $x_1 + \overline{x_2} \cdot x_3$. Compared with the ROBDD in Figure II.3, there is an extra node required. As the logic functions get more complex, the effect of a poor variable ordering becomes even more pronounced. Therefore, a variable ordering must be chosen that minimizes the number of nodes in the ROBDD. While the exact algorithm for computing the ordering of variables is NP-Complete, many heuristics have been developed which work well in practice. For the remainder of this thesis, the terms ROBDD and BDD are used interchangeably.

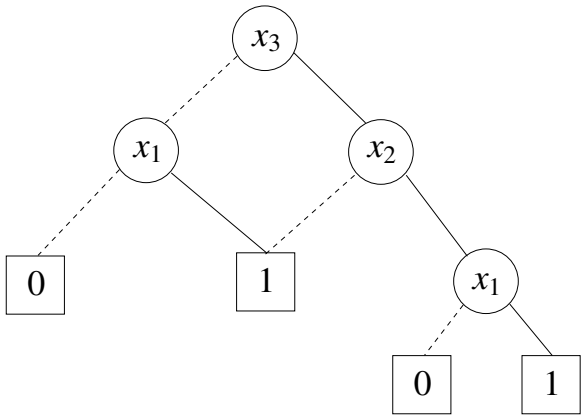

Fig. II.4. ROBDD of Logic Function $x_1 + \overline{x_2} \cdot x_3$ with Non-Optimal Variable Ordering

The following BDD operations are used in the work presented in the thesis:

- *bdd_smooth(f, smoothing_vars)*: This function returns the BDD formula of f existentially quantified with respect to the variables in the array *smoothing_vars*. For example, if $f = ab + \overline{a}c$, and $smoothing\_vars = [a]$, then the function returns $b + c$.

- *bdd_consensus(f, quantifying_vars)*: This function returns the BDD formula of f universally quantified with respect to the variables in the array $quantifying\_vars$. For

example, if $f = ab + \overline{a}c$, and $quantifying\_vars = [a]$, then the function returns bc.

- *bdd_node_to_bdd(node,leaves)*: This function builds the BDD for *node* in terms of the variables given in the table *leaves*. The BDD is built recursively from the BDD's of its immediate fanins. If a visited node already has a BDD, then this BDD will be reused; if it does not, then a new BDD will be constructed for the node.

II-D. BREL Boolean Relation Minimizer

Finding a set of multi-output functions that are compatible with a Boolean relation is a trivial task. For the relation in Table I.1, arbitrarily choosing either $\{00\}$ or $\{11\}$ as outputs for the input vector $\{10\}$ would yield a compatible solution. However, this solution may not be minimal in terms of the literal count of the resulting functions. BREL uses a recursive algorithm to explore a wide range of solutions, and chooses the best result based on a given cost function.

First, a quick initial solution is found. This is done by projecting the relation onto each output, and then minimizing the resulting incompletely specified function using the maximum flexibility provided by the relation. The constraints of the solution are passed on to the rest of the outputs to ensure that the final solution is compatible with the relation. Once this is done for all outputs, an initial cost for the solution is determined. However, this initial solution depends on the order that the outputs are minimized. In addition, it favors outputs minimized first, since they have the most flexibility, while the last outputs inherit little flexibility.

Next, a recursive algorithm is used to find an optimal solution. Each output is first minimized independently. If the resulting solution is compatible with the relation and has the lowest cost explored so far, then the solution is returned to the calling function. If the resulting solution is incompatible with the relation, then the relation R is split into two

relations R_1 and R_2, which are compatible with R. This is done by selecting an incompatible input vertex x and an output y_i and defining R_1 and R_2 as:

$$R_1 = R \cdot (\overline{x} + \overline{y_i})$$

$$R_2 = R \cdot (\overline{x} + y_i)$$

The algorithm is recursively called on R_1 and R_2, until either the cost is greater than the best cost previously explored, or if the terminal case is reached where R is a function. In the end, the output of BREL is the minimum-cost set of functions that are compatible with R.

II-E. Chapter Summary

This chapter presented the definitions and terminology that will be used for the remainder of this thesis. The structure and properties of BDDs, which are used extensively in this thesis, were also described. In addition, the Boolean relation minimizer BREL used in this thesis was explained.

CHAPTER III

APPROACH

III-A. Chapter Overview

This chapter is divided into the following sections. Section III-B describes the overall structure of the algorithm presented in this thesis. Section III-C explains each step of the algorithm in detail. Section III-D concludes the chapter.

III-B. Algorithm Overview

In general, the exact computation of the Boolean relation expressing the optimization flexibility of multiple nodes is extremely memory intensive, even for small networks. This is one of the reasons why past research efforts in this area have been mostly theoretical in nature. The approach for simultaneous multi-node minimization of a multi-level network presented in this thesis has several salient features.

- The flexibility is computed for simultaneously optimizing a pair of nodes of the network at a time, using an ROBDD-based approach.
- Memory explosion is avoided by a *windowing* technique which first creates a subnetwork around the two nodes being optimized. This subnetwork has a user-controllable topological depth. The Boolean relation representing the flexibility for simultaneously optimizing the two nodes is built in terms of the primary inputs of the *subnetwork*. This keeps the sizes of the ROBDDs under control, and effectively allows the approach to scale robustly for large networks, with very good result quality.
- During the computation of the ROBDD of the characteristic function of the Boolean relation, memory utilization is aggressively controlled by performing careful early

quantification.

- Further, instead of running this algorithm on all pairs of nodes, it is run on only those node pairs that are likely to yield good optimization opportunities. This is done *without* enumerating all node pairs.

Algorithm 1 describes the flow of the multi-level optimization methodology. The input is is a Boolean network η, and the output is an optimized Boolean network $\eta^{'}$, which is functionally equivalent to η.

The algorithm begins by efficiently selecting pairs of nodes to optimize from the original multilevel network η. Given a pair of nodes (n_i, n_j) to optimize simultaneously, the algorithm then finds a subnetwork $\eta_{i,j}$ which is rooted around these nodes. The Boolean relation $\mathcal{R}$ representing the simultaneous flexibility of these 2 nodes is computed in terms of the primary inputs of the subnetwork $\eta_{i,j}$. Finally, the Boolean relation $\mathcal{R}$ is minimized using a relation minimizer (BREL [11] in this thesis). The relation minimizer returns a multi-output function (in particular a 2 output function) f, such that f is compatible with $\mathcal{R}$ ($f \prec \mathcal{R}$). The optimized pair of nodes are then grafted back into η. At the end of the for loop, a *minimized* multi-level network $\eta^{'}$ is obtained.

The details of the steps of the algorithm are described in the next subsection.

III-C. Algorithm Details

III-C.1. Selecting Node Pairs

When selecting node pairs, it is important to find nodes that share common fanins and fanouts when the subnetwork is created. Not only will this make the subnetwork smaller, but it will also increase the likelihood that more flexibility will be found from the resulting relation.

Algorithm 1 Boolean Relation-based Multi-Node Optimization

$L = select\_nodes(thresh, k_1, k_2, \alpha)$

for all $(n_i, n_j) \in L$ **do**

$\eta_{i,j} = extract\_subnetwork(n_i, n_j, k_1)$

$\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}) = build\_relation\_bdd(\eta_{i,j}, X, Z, S, Y)$

$(n_i^{'}, n_j^{'}) = BREL(\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}))$

Graft $(n_i^{'}, n_j^{'})$ *in* η

Delete n_i *and* n_j from η

end for

Return $\eta^{'} = network\_sweep(\eta)$

To generate a list of all node pairs to minimize, $select\_nodes(thresh, k_1, k_2, \alpha)$ is called. This algorithm is shown in pseudocode in Algorithm 2 and graphically in Figure III.1.

This function starts by selecting a node n_i in the network. To find a potential partner n_j for this node, $TFI_{frontier}(n_i, k_1)$ is called, which returns only the nodes $\boldsymbol{m}$ in the transitive fanin frontier of n_i which have a backward depth of exactly k_1 levels from n_i. This step is shown in Figure III.1(a). For each of these nodes $m_l \in \boldsymbol{m}$, $TFO(m_l, k_2)$ is called, which returns nodes $\boldsymbol{n}$ in the transitive fanout of m_l that have a forward depth of up to k_2 levels from m_l. This gives all potential partners $\boldsymbol{n}$ and ensures that they will later share at least one common primary input in the subnetwork with n_i.

Note that nodes in the transitive fanin or fanout of n_i are not included in $\boldsymbol{n}$. The reason for this is explained in Section III-C.3. Figure III.1(b) shows the nodes (circles) that are included in $\boldsymbol{n}$, and the darkened edges show paths in the TFI of n_i where nodes will not be included in $\boldsymbol{n}$.

Next, each node $n_j \in \boldsymbol{n}$ is tested against n_i to measure their compatibility. The TFI and TFO (up to k_1 levels) of both nodes are considered when determining compatibility;

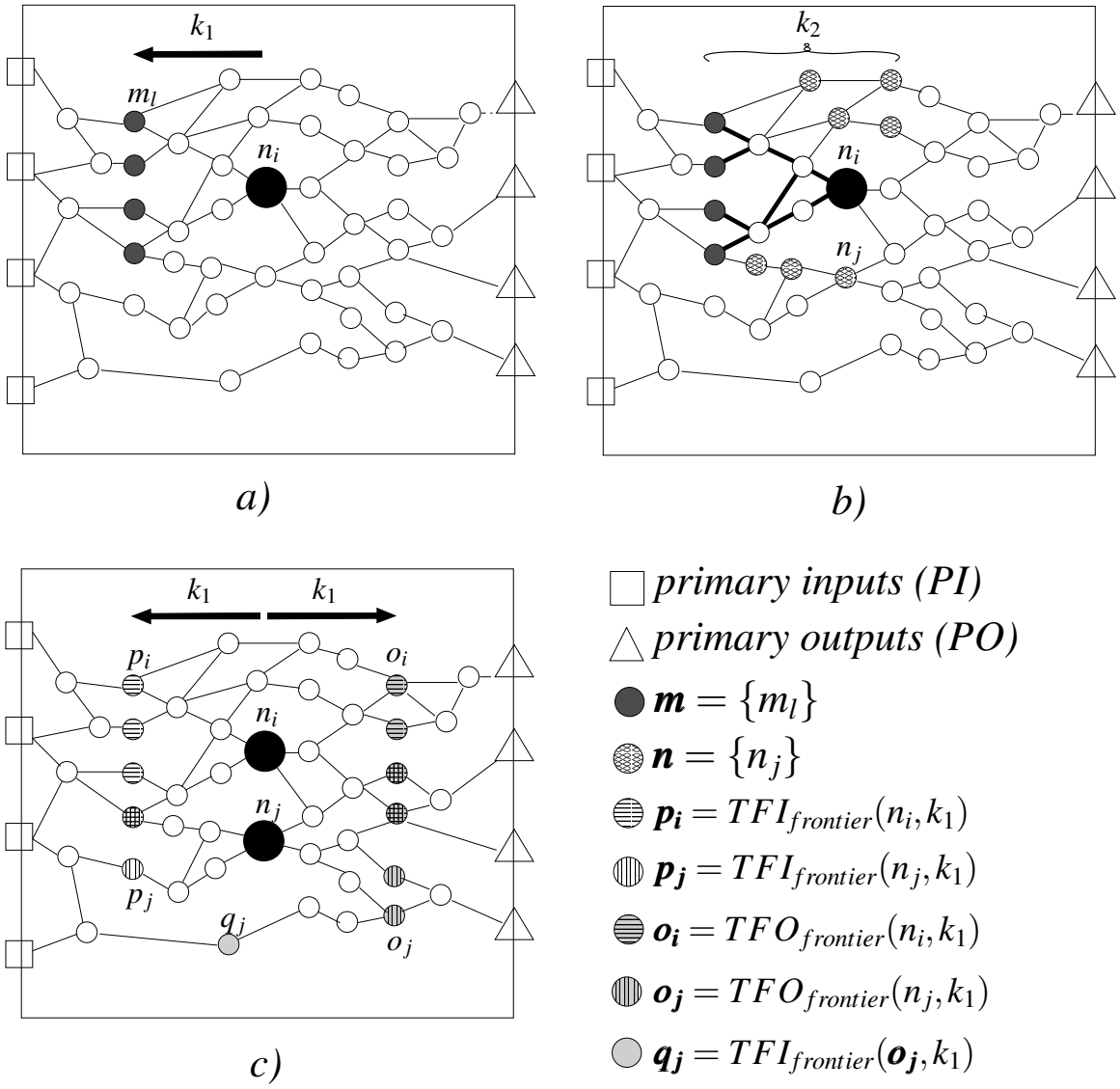

Fig. III.1. Selection of Node Pairs

Algorithm 2 Pseudocode of Node Selection Algorithm

```
for all (n_i) ∈ η do
  m ← TFI_frontier(n_i, k_1)
  for all m_l ∈ m do
    n ← TFO(m_l, k_2)
    for all n_j ∈ n do
      if n_j ∉ fanin or fanout of n_i then
        δ ← common_pi(n_i, n_j, k_1)
        ε ← common_po(n_i, n_j, k_1)
        if α·δ + (1 − α)·ε ≥ thresh then
          L ← (n_i, n_j)
        end if
      end if
    end for
  end for
end for
Return L
```

however, only nodes at the frontier of these sets are used, shown in Figure III.1(c). Node sets $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ are the $TFI_{frontier}$ sets for n_i and n_j respectively. Node sets $\boldsymbol{o_i}$ and $\boldsymbol{o_j}$ are the $TFO_{frontier}$ sets for n_i and n_j respectively. In addition, as will be explained in Section III-C.2, the sets $\boldsymbol{q_i}$ and $\boldsymbol{q_j}$ are the $TFI_{frontier}$ sets for $\boldsymbol{o_i}$ and $\boldsymbol{o_j}$ respectively. In Figure III.1(c), the set of nodes $\boldsymbol{q_i}$ is empty because all nodes in $\boldsymbol{o_i}$ can be expressed completely in terms of nodes already in $\boldsymbol{p_i}$.

These frontier nodes will later be the PIs and POs of the $k_1 \times k_1$ window around the nodes. The more of these nodes that n_i and n_j share in common, the fewer the number of

nodes required in the subnetwork.

The PI and PO compatibility of both nodes are calculated. The more these sets overlap, the more likely they are to be selected as a pair. The PI factor δ is defined as

$$\delta = \frac{|\boldsymbol{p_i} \cap \boldsymbol{p_j} \cap \boldsymbol{q_i} \cap \boldsymbol{q_j}|}{|\boldsymbol{p_i} \cup \boldsymbol{p_j} \cup \boldsymbol{q_i} \cup \boldsymbol{q_j}|}$$

and the PO factor ε is defined as

$$\varepsilon = \frac{|\boldsymbol{o_i} \cap \boldsymbol{o_j}|}{|\boldsymbol{o_i} \cup \boldsymbol{o_j}|}$$

To determine whether or not n_i and n_j will be selected as a pair to be optimized, δ and ε are scaled by α and 1-α, respectively, and tested if their sum is higher than a user-defined threshold *thresh*:

$$\alpha \cdot \delta + (1 - \alpha) \cdot \varepsilon \geq thresh$$

All nodes n_j for which the above test evaluates to be *true* are placed in the node pair list L, along with n_i.

These steps are performed for all $n_i \in \eta$, visited in topological order from the POs to the PIs, until every node has been tested for potential partners. A list L of all node pairs to optimize is returned. Additionally, care is taken to ensure that no pairs appear twice in L,

Next, a subnetwork $\eta_{i,j}$ of η, rooted at nodes (n_i, n_j), is extracted. The technique for this extraction is explained in the following subsection.

III-C.2. Building the Subnetwork

For each pair of nodes (n_i, n_j) found, subnetworks of η rooted at the nodes n_i and n_j are extracted by calling $extract\_subnetwork(n_i, n_j, k_1)$. This function constructs a subnet-

work $\eta_{i,j}$ such that if node $m \in \{TFO(n_i,k_1) \cup TFO(n_j,k_1)\}$, then $m \in \eta_{i,j}$ and if node $p \in \{TFI(n_i,k_1) \cup TFI(n_j,k_1)\}$, then $p \in \eta_{i,j}$. Here k_1 is the same value used when calling *select_nodes*. The result of this step is illustrated in Figure III.2(a) as the shaded subnetwork.

Node $m \in \eta_{i,j}$ is designated as a primary input of $\eta_{i,j}$ if $\exists_{n \in FI(m)},\ n \notin \eta_{i,j}$. Similarly, a node m is designated as a primary output of $\eta_{i,j}$ if $\exists_{n \in FO(m)},\ n \notin \eta_{i,j}$. The set of primary inputs (outputs) of $\eta_{i,j}$ is referred to as X (Z).

Next the set of all nodes $m \in TFI(v,k_1)$ is collected, where v is a primary output of the subnetwork $\eta_{i,j}$. This step is illustrated in Figure III.2(b). Let this set be called D. The nodes in the dotted and shaded region of Figure III.2(b) constitute the set D. These nodes are included in the subnetwork as well, by setting $\eta_{i,j} \leftarrow \eta_{i,j} \cup D$. Figure III.2(c) zooms into the region of interest for the subsequent discussion.

Next, for each $d \in D$ a check is done to see if $FI(d)$ can be expressed completely in terms of the current nodes in $\eta_{i,j}$. This check is performed by recursively traversing the network topologically from d towards the primary inputs X^{global} of η. If this traversal visits a node in $\eta_{i,j}$, the traversal terminates and all nodes visited in this traversal are added to $\eta_{i,j}$. If the traversal visits a node in X^{global} instead, then the set of primary inputs of $\eta_{i,j}$ is augmented with d, i.e. X is updated as $X \leftarrow X \cup d$. This step is illustrated in Figure III.2(d).

Nodes w and $r \in D$ could be considered as primary inputs to the subnetwork; however, all of their fanins can be expressed completely in terms of X. Thus, the fanin of the node $w \in D$ and the fanin u of $r \in D$ are added to $\eta_{i,j}$. However, the fanin of node $t \in D$ cannot be expressed in terms of nodes in $\eta_{i,j}$, and so t is added to X. This check avoids the addition of unnecessary primary inputs for representing the subnetwork $\eta_{i,j}$. A larger number of primary input variables typically results in larger intermediate ROBDDs in the computation of the Boolean relation $\mathcal{R}$, and consequently more time needed for the computations.

Note that the size of each subcircuit $\eta_{i,j}$ is determined by the depth parameter k_1.

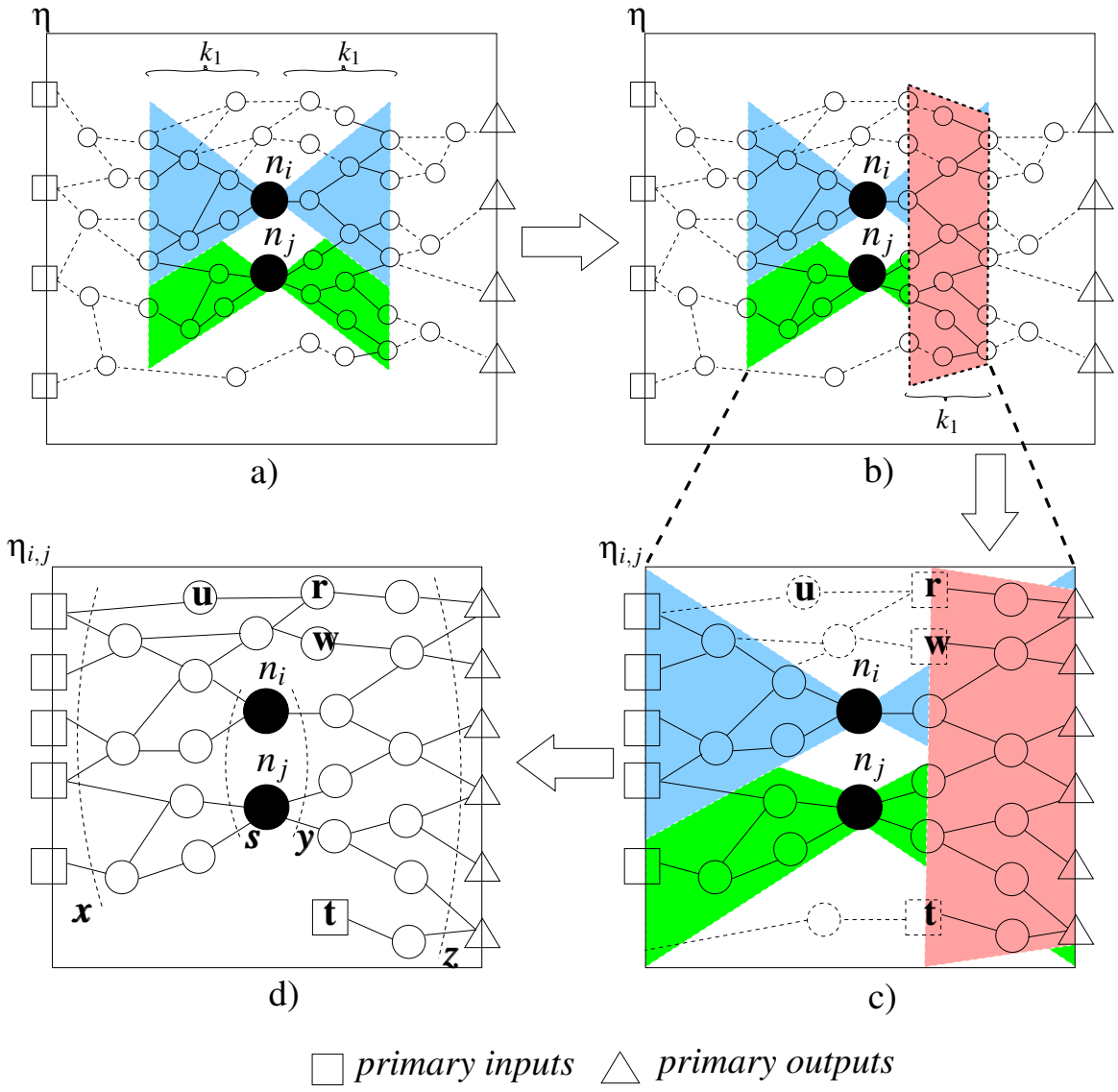

Fig. III.2. Extraction of Subnetwork

Hence, by suitably choosing k_1, it can be guaranteed that the subcircuits are never too large, and the Boolean relation can be computed with low memory utilization, even for an extremely large network η. The final subnetwork $\eta_{i,j}$ is shown in Figure III.2(d). This subnetwork is then used to create a Boolean relation which inherently represents the simultaneous flexibility of both n_i and n_j, as discussed in the following subsection.

III-C.3. Computing the Boolean Relation $\mathcal{R}^Y$

As mentioned previously, the exact computation of a Boolean relation expressing the flexibility in a medium to large design could be extremely memory intensive. Additionally, ROBDD-based computations are used for this relation. ROBDDs can, by nature, exhibit very irregular memory requirements, especially for medium to large designs. A goal of this thesis is to develop a robust methodology for computing the Boolean relation. This is achieved by keeping a tight control on the sizes of the BDDs of the Boolean relation. Not only is this relation computed for a node pair (n_i, n_j) using a windowed subnetwork $\eta_{i,j}$ (thus ensuring that the ROBDDs are small) but also careful early quantification is performed to ensure that the ROBDD sizes stay tractable during the relation computation.

Consider a subnetwork $\eta_{i,j}$, its set of primary inputs X and its set of primary outputs Z. Let the set of nodes being simultaneously optimized be referred to as Y and their combined support be S. Note that S, Y, X and Z correspond to a set of nodes of $\eta_{i,j}$. Let the variables for these be $\boldsymbol{s}$, $\boldsymbol{y}$, $\boldsymbol{x}$ and $\boldsymbol{z}$ respectively as shown in Figure III.2(d). The characteristic function of the Boolean relation $\mathcal{R}$ is a mapping $B^{|S|} \times B^{|Y|} \rightarrow B$ s.t.

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \forall_{\boldsymbol{z}}[(\boldsymbol{z} = Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow \Phi(\boldsymbol{x},\boldsymbol{z})]]$$

In this expression, $\Phi(\boldsymbol{x},\boldsymbol{z})$ is the characteristic function of the circuit outputs $\boldsymbol{z} = f(\boldsymbol{x})$. The subexpression $Z^M(\boldsymbol{x},\boldsymbol{y})$ represents the characteristic function of the circuit outputs expressed in terms of $\boldsymbol{x}$ and $\boldsymbol{y}$. Also, $g_S(\boldsymbol{x})$ is the characteristic function of the $\boldsymbol{s}$ variables in

terms of $\boldsymbol{x}$. The computation of $\mathcal{R}^Y$ is explained intuitively as follows. For all primary input minterms $\boldsymbol{x}$, let $\boldsymbol{s}$ take on values dictated by $\boldsymbol{x}$ (i.e. $\boldsymbol{s} = g_S(\boldsymbol{x})$). If this is the case, then if $\boldsymbol{z}$ takes on the values dictated by $\boldsymbol{x}$ and the node values of $\boldsymbol{y}$, the values of $\boldsymbol{x}$ and $\boldsymbol{z}$ should be related by the original network functionality (i.e. $\Phi(\boldsymbol{x},\boldsymbol{z})$).

One caveat of this computation is that the two nodes (n_1, n_2) for which the relation is calculated cannot be in each others' TFI or TFO. The reason for this is explained as follows. Suppose node n_1 is a fanin of node n_2. If that is true, then $n_1 \in \boldsymbol{s}$ and $n_1 \in \boldsymbol{y}$ simultaneously. If the relation is then minimized, BREL produces functions $\boldsymbol{y} = f(\boldsymbol{s})$ that are cyclic, with variables being on both sides of the equation. This could lead to feedback in the optimized circuit. For this reason, a node in the other node's TFI or TFO is not chosen in the node selection algorithm.

III-C.4. Quantification Scheduling

In the approach presented in this thesis, the Boolean relation $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$ is computed using ROBDDs. In order to avoid a possible memory explosion problem, early quantification is performed as explained next.

The computation for $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$ is rewritten as

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \forall_{\boldsymbol{z}}[\prod_i (z_i \overline{\oplus} Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow \prod_i (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]$$

This expression can be re-written as:

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \forall_{\boldsymbol{z}}[\prod_i [(z_i \overline{\oplus} Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]]$$

The first observation is that the quantification over $\boldsymbol{z}$ ($\forall_{\boldsymbol{z}}$) and the product term over i ($\prod_i$) can be swapped to obtain a new expression for $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$:

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \prod_i [\forall_{\boldsymbol{z}}[(z_i \overline{\oplus} Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]]$$

This is correct because in general,

$$\forall_{\omega}(f \cdot g) = \forall_{\omega}(f) \cdot \forall_{\omega}(g)$$

Quantifying out the $\boldsymbol{z}$ variables earlier results in smaller intermediate ROBDDs for the expression to the right of the first implication. The computation can therefore be expressed as:

$$\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow P(\boldsymbol{x})] = \forall_{\boldsymbol{x}}[\overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + P(\boldsymbol{x})]$$

where P($\mathbf{x}$) is is the ROBDD obtained after applying the first observation.

$$P(\boldsymbol{x}) = \prod_{i}[\forall_{\boldsymbol{z}}[(z_i \overline{\oplus} Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]$$

In general, however,

$$\forall_{\omega}(f+g) \neq \forall_{\omega}(f) + \forall_{\omega}(g)$$

Let the common variables between f and g be ω^*. Let $\omega^{'} = \omega \cap \omega^*$. Then,

$$\forall_{\omega}(f+g) = \forall_{\omega'}(\forall_{\omega \setminus \omega'}(f) + \forall_{\omega \setminus \omega'}(g))$$

The second observation is that $g_S(\boldsymbol{x})$ depends on a smaller subset ($\boldsymbol{x'}$) of the primary inputs ($\boldsymbol{x}$) of the network. Hence, $\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y})$ can be computed as

$$\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x'}}[\forall_{\boldsymbol{x} \setminus \boldsymbol{x'}} \overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + \forall_{\boldsymbol{x} \setminus \boldsymbol{x'}}(P(\boldsymbol{x}))]$$

which reduces to:

$$\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x'}}[\overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + \forall_{\boldsymbol{x} \setminus \boldsymbol{x'}}(P(\boldsymbol{x}))]$$

In practice, both observations are applied in tandem. First $g_S(\boldsymbol{x})$ is found, as well as

the set $\boldsymbol{x}'$. Then, while computing P($\mathbf{x}$), $\boldsymbol{x}\backslash\boldsymbol{x}'$ is quantified out. The final computing step is

$$\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}'}[\overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + P'(\boldsymbol{x}')]$$

where $P'(\boldsymbol{x}') = \forall_{\boldsymbol{x}\backslash\boldsymbol{x}'}(P(\boldsymbol{x}))$. By implementing both these techniques, intermediate ROBDD never blows up in size. Without using the early quantification ideas, the ROBDD size is dramatically larger, hence the early quantification is key to the robustness and scalability of the approach of this thesis. The final ROBDD representing $\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y})$ is returned to the calling function.

III-C.5. Endgame

Next, BREL is called to minimize $\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y})$. The output of BREL is a pair of completely specified functions for the nodes n_i' and n_j' such that these functions are compatible with $\mathcal{R}^{Y}(\boldsymbol{s},\boldsymbol{y})$ and the total cost of n_i' and n_j' is minimal. The new nodes n_i' and n_j' are grafted back into η and the original nodes n_i and n_j are deleted from η.

At the end of the *for* loop in Algorithm 1, when all node pairs have been processed by the relation-based minimization procedure, the *network_sweep* command of SIS [20] is run. This command quickly eliminates any constant-valued nodes in the network that may have been created during the minimization process. Finally, the *network_verify* command of SIS is run to check if the resulting network η' is functionally equivalent to the original network η.

III-D. Chapter Summary

This chapter described the algorithm used for selecting the node pairs to be minimized, as well as the methods for generating the subnetwork and computing the Boolean relation. Details of the early quantification scheduling were given, along with the intuition behind

all of the steps involved. The experimental results for this approach and the conclusions drawn from them are discussed in the following chapters.

CHAPTER IV

EXPERIMENTAL RESULTS

IV-A. Chapter Overview

This chapter presents the experimental results for the algorithm described in the previous chapter. Section IV-B describes the preprocessing steps used in all the experimental results. Section IV-C shows the methodology used to determine the the parameters that control the node selection algorithm. Section IV-D reports the results obtained when comparing with the *mfsw* approach of [21], which is the most powerful technique among single-node optimized approaches (in terms of runtime and quality of results). Section IV-E discusses some variations on the original algorithm and their results. Section IV-F shows the properties of the node pairs that either did or did not produce gains when minimized. The chapter is concluded in Section IV-G.

The metric for quality that is used throughout this chapter is literal count. This is the sum of the number of literals for each node in the network. The fewer the number of literals in the network, the better the optimization technique. The literal counts shown in the results are all relative to another approach used for comparison. Runtimes are also reported in these results. For some experiments, absolute runtime is reported, and for others, runtime relative to another approach is reported.

The approach in this thesis was implemented in SIS [20], a logic synthesis package written in C. The ROBDD package used was the CUDD package [9]. A sample of 15 medium and large circuits from the *mcnc91* and *itc99* benchmarks were used in the experiments. The experiments were performed on a Linux-based Dell Optiplex with a 2.6GHz Core 2 Quad CPU with 4 GB of RAM.

IV-B. Preprocessing Steps

Before any minimization is performed on the original network, two preprocessing steps are performed. The first is the *network_sweep* command of SIS, which eliminates constant-valued nodes as well as nodes which do not fanout anywhere.

The second step is running *sat_sweep* [28] on the network. This command uses a Boolean Satisfiability (SAT) checker to determine if two nodes u and v are functionally identical by calling SAT_CHECK($u \oplus v$). This checks if there is any input vector to u and v for which the outputs of u and v differ. If there is, then the nodes are functionally different and cannot be merged, and a new pair is selected. Otherwise, then the nodes are functionally equivalent and can be merged together. This algorithm quickly reduces the literal count of a circuit by removing redundancies. The results in this chapter are obtained over and above what *sat_sweep* achieves. The results reported for the competing technique *mfsw* [21] were also preceded by a *sat_sweep* command. In other words, *sat_sweep* is run first. Then the additional improvements obtained by the proposed method are compared to those obtained by *mfsw*.

IV-C. Parameter Selection

As described in Section III-C, the node selection algorithm is based on four user-defined parameters, namely $thresh$, k_1, k_2, and α. Tuning these parameters can customize the trade-off between quality of results and runtime. In general, the longer the runtime, the better the quality of results. However, the runtime is heavily dependent on the number of nodes chosen. If changing a parameter increases the runtime, this is because either more node pairs were selected, or the processing time of a node pair is increased. Depending on the 'quality' of the additional pairs, the literal reduction could change as well. Because of this, optimal values need to be determined for all parameters as a first step.

The experiments in this section are conducted to find a 'golden' set of parameter values for the proposed approach. In these experiments, the ranges of values for each parameter are listed in Table IV-C. The nominal values of these parameters are also listed in this table.

Table IV.1. Initial Values, Final Values, Increments, and Nominal Values of the Node Selection Parameters

Parameter	Low	High	Increment	Nominal
α	0	1.0	0.1	0.5
k_1	2	3	1	2
k_2	2	4	1	3
thresh	0	1.0	0.1	0.5

IV-C.1. Selecting α

The first parameter to determine is α. This parameter determines the weight that PIs and POs of the subnetwork are given when selecting a node pair. The parameter α can range from 0, which considers only POs, to 1.0, which considers only PIs. The reason for determining α first is because it is the parameter least dependent on the others. Since *thresh*, k_1, and k_2 affect only the *number* of pairs selected and the window size, α can be chosen first.

Figure IV.1 shows α being swept from 0 to 1.0, while the other three parameters are held constant. The nominal values for *thresh*, k_1, and k_2 were chosen in the middle of their ranges at 0.5, 2, and 3 respectively. The left axis represents the ratio of literals obtained compared to that obtained after running *sat_sweep*, and the right axis represents the average runtime of the method used here. For each value of α, the average literal ratio and runtime is presented in Figure IV.1, across all the benchmark examples.

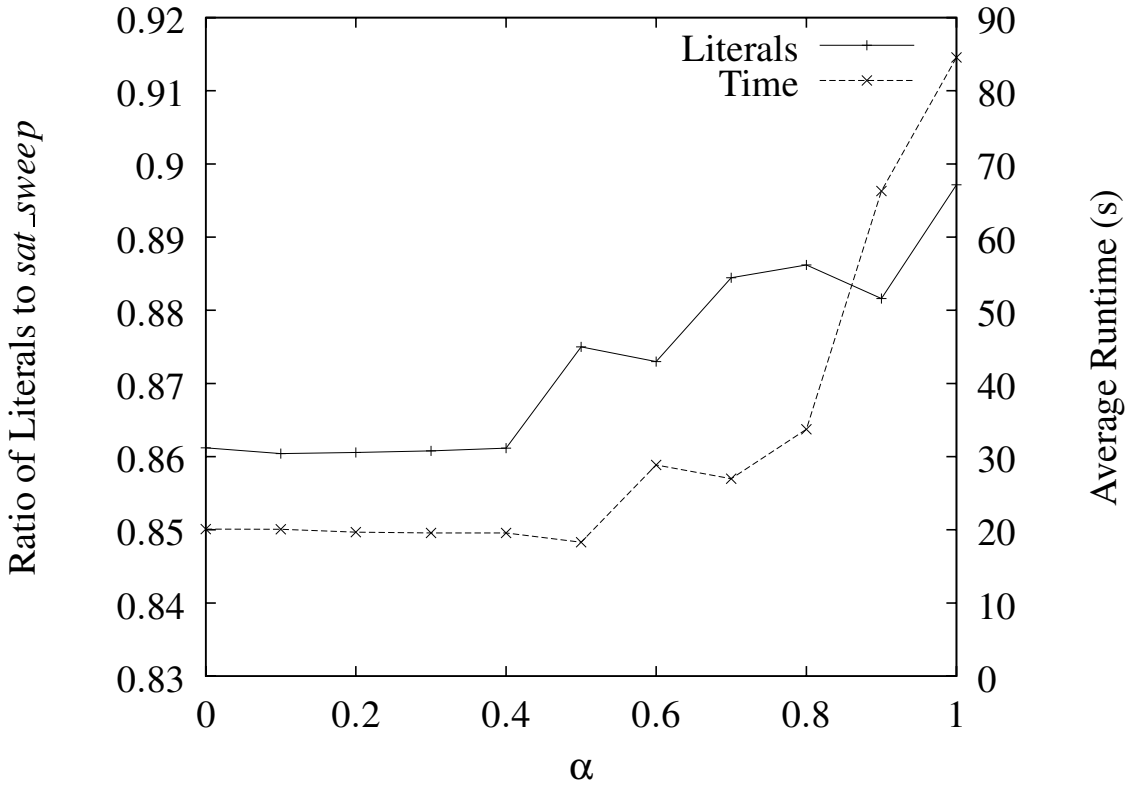

Fig. IV.1. Sweeping α with $thresh = 0.5$, $k_1 = 2$, $k_2 = 3$

The main observation is that in general, lower values of α yield lower runtimes as well as fewer literals. This means that PO compatibility is more important when choosing node pairs than PI compatibility. At the extremes, it is seen than an α value of 0 provides 40% more literal reduction in a quarter of the runtime than with an α value of 1.0. Since both runtime and literal count increase with α, one can infer that with the higher α values, more node pairs were chosen, but the pairs gave less literal count reduction. This shows that when two nodes are minimized together, more flexibility is obtained if they reconverge quickly than if they share a common variable support.

For $0 \leq \alpha < 0.5$, both runtime and literal count are relatively flat. Therefore, the value of 0.25 is chosen for α for the rest of the experiments in this thesis. Although there is no empirical data to guide the selection of α for values less than 0.5, a value in the middle of

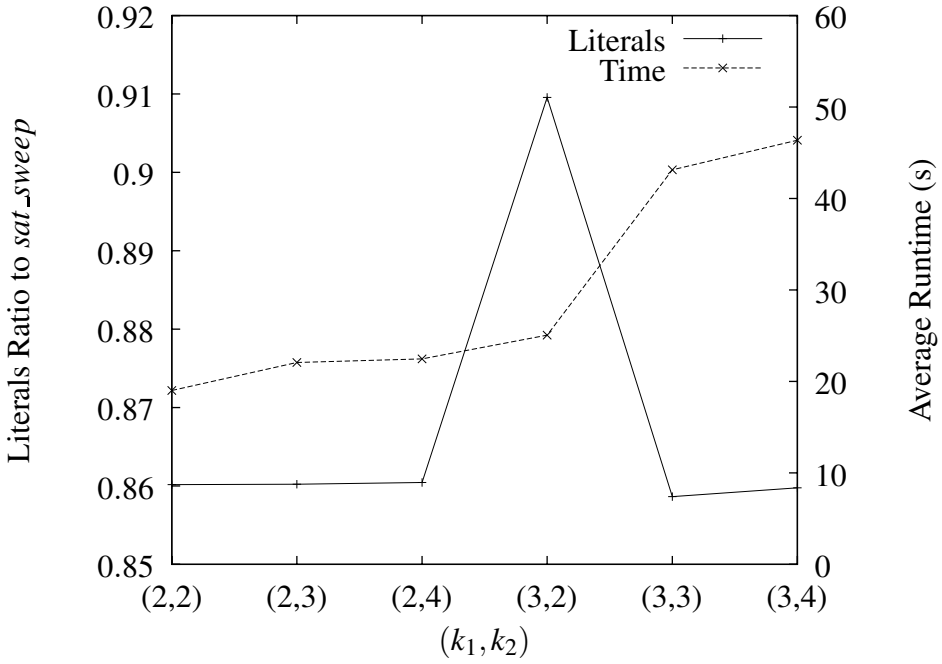

Fig. IV.2. Sweeping k_1 and k_2 with $thresh = 0.5$ and $\alpha = 0.25$

the range is chosen. This is so that PI compatibility still contributes to the node selection, but not enough to degrade the results.

IV-C.2. Selecting k_1 and k_2

The parameters k_1 and k_2 are determined next. The size of the window is determined by k_1, because the subnetwork created includes nodes k_1 levels back and k_1 levels forward from the nodes to be minimized, as shown in Figure III.2(a). Therefore a larger value of k_1 means more nodes are included in the subnetwork. The parameter k_2 affects the number of nodes in $\boldsymbol{n}$, shown in Figure III.1(b), which are tested against the first node n_i for compatibility. A larger value of k_2 means that more partners are tested for n_i.

Figure IV.2 shows the literal ratio and average runtime for different values of (k_1, k_2). The first observation is that the point $(3,2)$ has a distinctly higher literal ratio than the other points. This is because node n_j is selected by going back 3 topological levels from n_i but

then forward only 2 levels from there. This precludes any node n_j that is on the same level as n_i from being selected. By comparing points $(3,2)$ and $(3,3)$ in Figure IV.2, is is clear that these nodes account for a large portion of the gains in literal ratio.

From the other points, it can be seen that increasing k_1 or k_2 has little effect on the literal ratio but causes a much higher increase in runtime. Therefore the values $k_1 = 2$ and $k_2 = 2$ are chosen.

IV-C.3. Selecting *thresh*

The final parameter to determine is *thresh*. This parameter controls how 'compatible' two nodes must be for them to be selected as a pair. A high value of *thresh* means that only node pairs with a high percentage of outputs and inputs in common are chosen for minimization. A low value of *thresh* allows the nodes with fewer inputs and outputs in common to be minimized as well.

Figure IV.3 shows *thresh* being swept with $k_1 = 2, k_2 = 2$, and $\alpha = 0.25$. The left axis again shows the ratio of the number of literals using this technique to the literals from the original network after calling *sat_sweep*. The right axis shows the average runtime in seconds for the minimization to complete, plotted on a log scale. This shows that for $thresh \leq 0.2$, the runtime increases exponentially as *thresh* decreases, and the literal ratio decreases linearly. For large values of *thresh*, the runtime decreases, but the literal ratio increases drastically. This is because very few nodes are selected for minimization when the threshold is high.

While either extreme can be chosen if speed or literal ratio alone were desired, selecting an intermediate value of *thresh* can achieve a balance between the two. Therefore $thresh = 0.4$ is selected for the remaining experiments in this thesis.

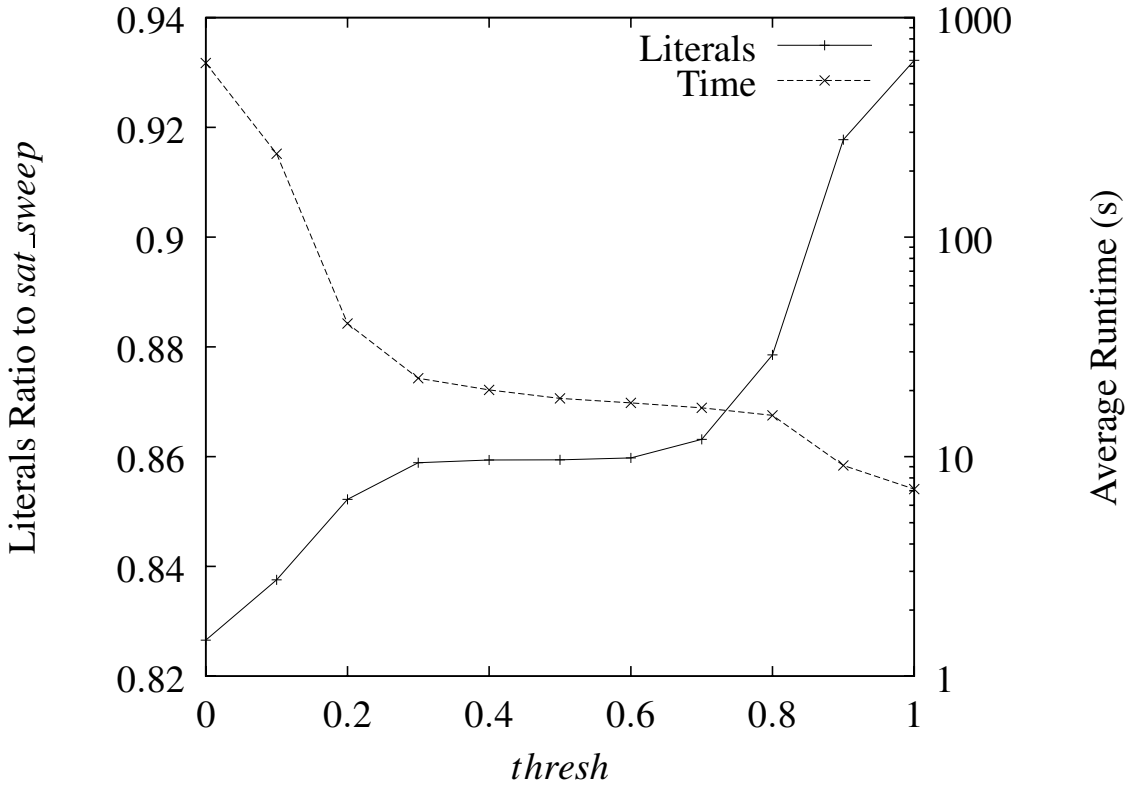

Fig. IV.3. Sweeping *thresh* with $k_1 = 2, k_2 = 2$, and $\alpha = 0.25$

IV-D. Comparison of the Proposed Technique with *mfsw*

As described in Section I-B, many single node techniques have been developed for minimizing Boolean networks using don't cares. Of these techniques, the method of [21], called *mfsw*, has the best results and scales well. In this section, the results from the method presented in this thesis are labeled as *relation* and are compared with the results from *mfsw*. The *mfsw* technique uses a SAT-based CDC method and a 2x2 window for creating subnetworks.

For the remaining experiments, the 'golden' values of the parameters as described in Section IV-C are used. In particular, $thresh = 0.4$, $k_1 = 2$, $k_2 = 2$ and $\alpha = 0.25$.

For the results shown in Table IV.2, *sat_sweep* is run first, providing the starting point

for both the *mfsw* technique and the method described in this thesis. The literal count after *sat_sweep* is shown in Column 2 of Table IV.2. The literal count and runtime after running *mfsw* is reported in Column 3 and Column 4, respectively. The literal count and runtime after running *relation* is reported in Column 5 and Column 6, respectively. For these columns, the average literal count relative to *sat_sweep* is shown in the last row. Column 7 shows the ratio of literals in Column 5 to Column 3, and Column 8 shows the ratio of runtimes in Column 6 to Column 4. Column 9 reports the peak number of ROBDD nodes for *relation*, and Column 10 shows the percentage of node pairs selected by *relation* that actually reduce the number of literals in the network.

From Table IV.2 it is seen that after *sat_sweep*, the *relation* method reduces the literal count by approximately 12% over what *mfsw* achieves. This shows that minimizing two nodes simultaneously has significant benefits in terms of literal count over the don't care approach of *mfsw*. The memory requirements are also very low regardless of the size of the circuit, due to the aggressive quantification scheduling performed. This supports the claim that *relation* scales well, and is a robust technique. Column 7 shows that the node selection method is quite efficient. On average, 45% of the node pairs chosen resulted in a reduction in the number of literals in the network.

In terms of runtime, *mfsw* is clearly more efficient than *relation*, which requires nearly 40x more runtime on average. However, the absolute time values must be taken into account. Column 6 shows that for most circuits runtimes are under one minute, and the peak runtime is still under four minutes for the largest circuit. Compared to the time scale of the entire design process, which is measured in months or years, these times are therefore quite small. In addition, as discussed in Section IV-C, the parameters for selecting node pairs can be altered to decrease the runtime.

It should also be noted that increasing the window size of *mfsw* to a 10x10 window greatly increases the runtime of that method but reduces the literal count by less than 1%.

Table IV.2. Results after *sat_sweep*

	orig	*mfsw*		*relation*		ratio			
circuit	lits	lits	time	lits	time	lits	time	mem	% gain
c1355	992	992	0.09	598	1.53	0.603	16.96	339	1
c1908	759	748	0.09	595	6.74	0.795	74.88	54939	0.41
c2670	1252	1197	0.11	901	4.04	0.753	36.76	1025	0.78
c5315	3062	2935	0.29	2372	12.37	0.808	42.65	2683	0.74
c7552	3796	3549	0.43	2990	14.30	0.842	33.25	3314	0.68
b15	15084	14894	1.78	14654	49.31	0.984	27.70	5594	0.64
b17	49096	48595	5.74	48047	228.57	0.989	39.82	6578	0.60
b20	22037	21816	2.56	21501	91.40	0.986	35.70	3489	0.26
b21	22552	22306	2.59	21933	92.08	0.983	35.55	3489	0.32
b22	33330	33001	3.97	32321	203.11	0.979	51.16	3519	0.28
s1494	1239	1177	0.13	1195	3.68	1.015	28.31	594	0.20
s5378	2327	2283	0.27	1993	7.36	0.873	27.26	3306	0.38
s13207	5052	4833	0.38	4259	27.35	0.881	71.96	1430	0.30
s15850	6624	6342	0.52	5519	26.10	0.870	50.19	1234	0.51
s38417	17531	17314	1.43	17158	68.68	0.991	48.02	2598	0.41
average	1	0.974	-	0.859	-	0.882	38.62	-	0.45

This means that while *relation* does require more runtime, the minimization it performs cannot be matched by *mfsw* regardless of the time it is allowed to run.

IV-E. Additional Experiments

Section IV-D presented the gains of the relation-based minimization approach after running *sat_sweep*. In this section, a variety of other experiments are performed to further explore the relation-based technique.

IV-E.1. Running *relation* After *mfsw*

For this experiment, *relation* is run on networks that have *already* been reduced by *sat_sweep* and *mfsw*. The purpose is to test how much *relation* can improve upon the minimization results of *mfsw*. Table IV.3 shows the results of this experiment. Column 2 (3) reports the literal count (runtime) of running *sat_sweep* followed *mfsw*. Columns 4 and 5 show the literal count and runtime (respectively) of running *relation* on the netlist obtained by *sat_sweep* followed by *mfsw*. The literal and runtime ratios are shown in Column 6 and Column 7, respectively.

It is seen from Column 6 that running *relation* after *mfsw* can further reduce the literals by about 13%. Since *the window sizes of both methods were identical*, this improvement represents the benefits of two node minimization over single node minimization. In the specific case of circuit c1355, nearly 40% of the literals can be removed only through the node pair technique. Columns 6 and 7 demonstrate again that the memory utilization is very low, and the node selection method is effective.

Table IV.3. Results after *sat_sweep* and *mfsw*

	mfsw		$mfsw + relation$		ratio		
circuit	lits	time	lits	time	lits	time	mem
c1355	992	0.09	600	1.53	0.605	16.994	336
c1908	748	0.09	588	2.86	0.786	31.759	12026
c2670	1197	0.11	906	4.21	0.757	38.307	742
c5315	2935	0.29	2298	11.57	0.783	39.912	1452
c7552	3549	0.43	2795	13.17	0.788	30.635	1842
b15	14894	1.78	14558	44.18	0.977	24.822	1262
b17	48595	5.74	47639	213.90	0.980	37.264	5648
b20	21816	2.56	21293	91.65	0.976	35.802	3490
b21	22306	2.59	21711	91.97	0.973	35.509	3489
b22	33001	3.97	32050	202.72	0.971	51.063	3511
s1494	1177	0.13	1142	3.29	0.970	25.319	673
s5378	2283	0.27	1972	6.95	0.864	25.738	5666
s13207	4833	0.38	4256	27.15	0.881	71.442	13121
s15850	6342	0.52	5331	23.92	0.841	46.004	1212
s38417	17314	1.43	16968	55.62	0.980	38.897	2009
average	-	-	-	-	0.868	34.684	-

IV-E.2. Running *relation* Twice

In this experiment, after running *sat_sweep*, *relation* is run twice in succession on the same network. The purpose of this experiment is to determine if there are improvements that can be had by minimizing a network multiple times. Three separate experiments are tried. In the first, only nodes pairs that did not give any literal count reduction during the first run are minimized again. In the second, only node pairs that *did* give a reduction in literal count are minimized again. And finally, all node pairs were rerun regardless of whether they yielded a literal reduction or not during the first run.

The results for all three experiments showed a less than 1% improvement in literal count compared to the first run of *relation*. These experiments show that even though the network has changed significantly after the first run of *relation*, these changes have almost no impact on the ability of other nodes to be further minimized in a subsequent iteration. The same conclusion can be drawn for the *mfsw* method as well, which also yields almost no further reductions when run more than once on a network.

IV-E.3. Minimizing Single Nodes

In this experiment, *sat_sweep* is run first, followed by *relation*. During the *relation* algorithm, some nodes get minimized while others do not, either because they were not selected in a node pair or because the algorithm did not reduce their literal count. For such nodes, an additional step of minimization was performed after running *relation*. After *relation*, these nodes are again minimized individually using ODCs. This is implemented using the same steps in Chapter III, by creating a relation corresponding to the subnetwork, and then using BREL to minimize it. The only difference is that only one node is used. This experiment ensures that some type of minimization is attempted for *each* node in the network.

However, experimental results showed that this idea does not further reduce the lit-

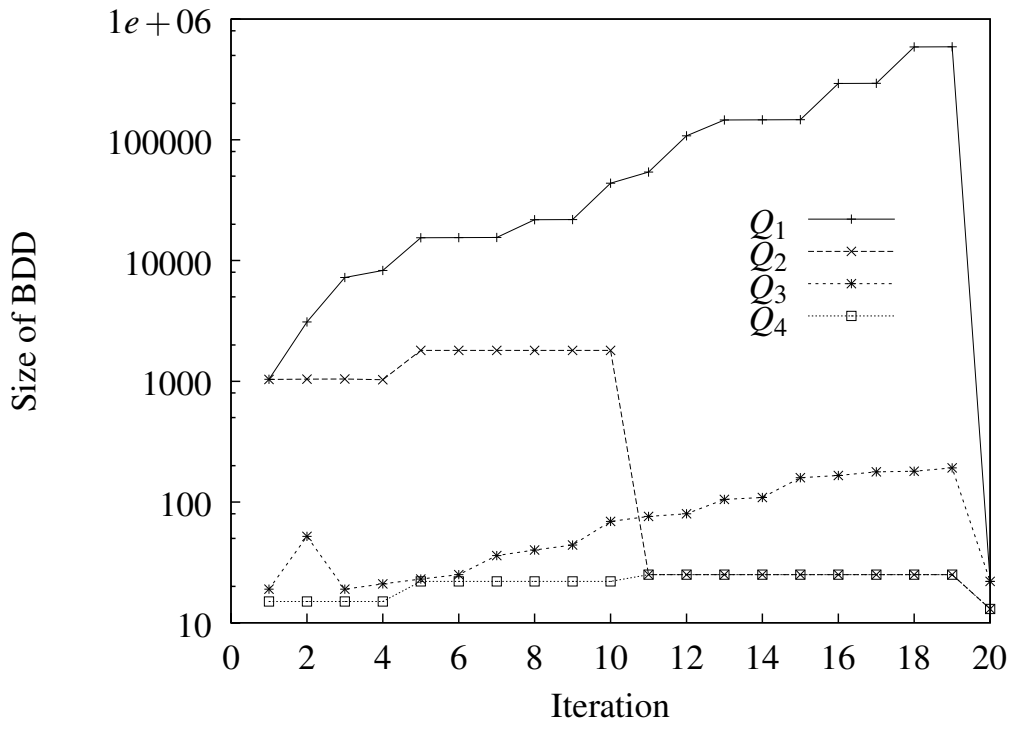

Fig. IV.4. Effects of Quantification Scheduling on BDD Size

eral count by more than 1%, across all the circuits. Almost all of the nodes which were subjected to single node optimization techniques were those that were originally selected but did not reduce their literal count when minimized with other nodes in a pair. The conclusion that can be drawn is that if a node cannot be minimized with another node, then minimizing it alone does not yield any gains either.

IV-E.4. Effects of Early Quantification

Section III-C.4 discusses the methods for early quantification used for the approach presented in this thesis. Figure IV.4 shows the effects of quantifying during different stages of the computation of $P(\boldsymbol{x})$ from $\mathcal{R}(Y)$. One node pair of the network $c432$ is being minimized in this example, and the number of nodes in the BDD during each iteration of the $P(\boldsymbol{x})$ computation is reported.

The plot Q_1 represents the incremental size of the relation BDD without any early quantification. Note that the BDD size is reported on a logarithmic scale. After only 20 BDD calculations, the size of the BDD is almost one million nodes. Only at the end of the computation, when the $\boldsymbol{z}$ and $\boldsymbol{x}$ variables are quantified out, does the size of the BDD drop.

The plot Q_2 shows the BDD size when the output variables $\boldsymbol{z}$ are quantified out after each iteration of the computation. The number of nodes stays near 1000 until a particular output in $\boldsymbol{z}$ is quantified out, and then the BDD size drops to 25.

The plot Q_3 is the case when only the $\boldsymbol{x}\backslash\boldsymbol{x}'$ variables are quantified out after each iteration. The BDD size steadily climbs to over 100, until the final iteration, when the $\boldsymbol{x}$ variables are quantified out.

Q_4 shows the results when the quantification techniques Q_2 and Q_3 are applied in tandem. The size of the BDD never grows past 25 during *the entire computation*.

This example demonstrates that for a single node pair, even when a windowing technique is employed, the BDD of $\mathcal{R}^{(}Y)$can blow up in size unless both the early quantification techniques of Section III-C.4 are employed.

IV-E.5. Code Profiling

Table IV.4 shows the tasks of the minimization algorithm that contribute most to the total runtime. The results for networks with less than 5000 nodes are shown in Column 2, and the results for networks with more than 5000 nodes are shown in Column 3.

For the small networks, relation minimization using BREL takes the vast majority of runtime. Selecting node pairs and creating the relation do not take much time comparatively because they are based on network size.

For the large networks, the node selection and relation building take longer because the number of nodes that are processed is much higher. The runtime for BREL is not necessarily smaller but is smaller in proportion to the total runtime.

Table IV.4. Contribution of Algorithm Components to Total Runtime

	Percentage of Runtime	
Section of Algorithm	Small Networks	Large Networks
BREL Relation Minimization	78%	21%
Node Pair Selection	13%	36%
Building the Relation	9%	35%

IV-F. Node Pair Properties

The first step of the Boolean relation-based minimization technique presented in this thesis is to select node pairs that are likely to provide reductions in literal count when minimized together. The efficacy of this selection algorithm can be seen in Table IV.2, where 45% of the pairs resulted in a literal count reduction. However, the other 55% of the pairs did not give any gain. To find out the characteristics of these pairs, Table IV.5 presents 15 properties of all the node pairs, selected across every circuit. Column 2 shows the average value of any property (per node) for the pairs that did result in a literal count reduction, and Column 3 shows the average values for the nodes that did not. Column 4 takes the ratio of Column 2 to Column 3. It helps determine if there is any difference between the pairs that did result in a literal count reduction and pairs that did not (for that property). A value significantly different from 1 indicates that there is a strong correlation between the property and the minimization ability.

The first four properties in Table IV.5 show that pairs that minimize well have a higher percentage of fanins and fanouts in common than those that do not. At first it may seem contradictory that common fanins are more correlated to gains than common fanouts, since Figure IV.1 shows otherwise. However, the statistics shown here are for nodes already

Table IV.5. Properties of Node Pairs that Can Be Minimized

Property	Gain	No Gain	Ratio
% of fanin in common	34	14	2.42
% of fanout in common	46	25	1.84
% of subnetwork PI in common	38	27	1.40
% of subnetwork PO in common	49	42	1.16
Difference in level	1.06	1.35	0.79
Average level in network	11.16	9.59	1.16
Depth in circuit (0=PI, 1=PO)	0.26	0.24	1.10
Combined literals	4.50	4.62	0.97
Combined cubes	3.50	4.22	0.83
Combined immediate fanin	4.44	4.56	0.97
Combined immediate fanout	2.17	3.51	0.62
Network PI in subnetwork	0.31	0.53	0.59
Nodes in subnetwork	17.34	33.32	0.52
Literals in subnetwork	32.96	69.39	0.48
Size of Relation	7.87	16.28	0.48

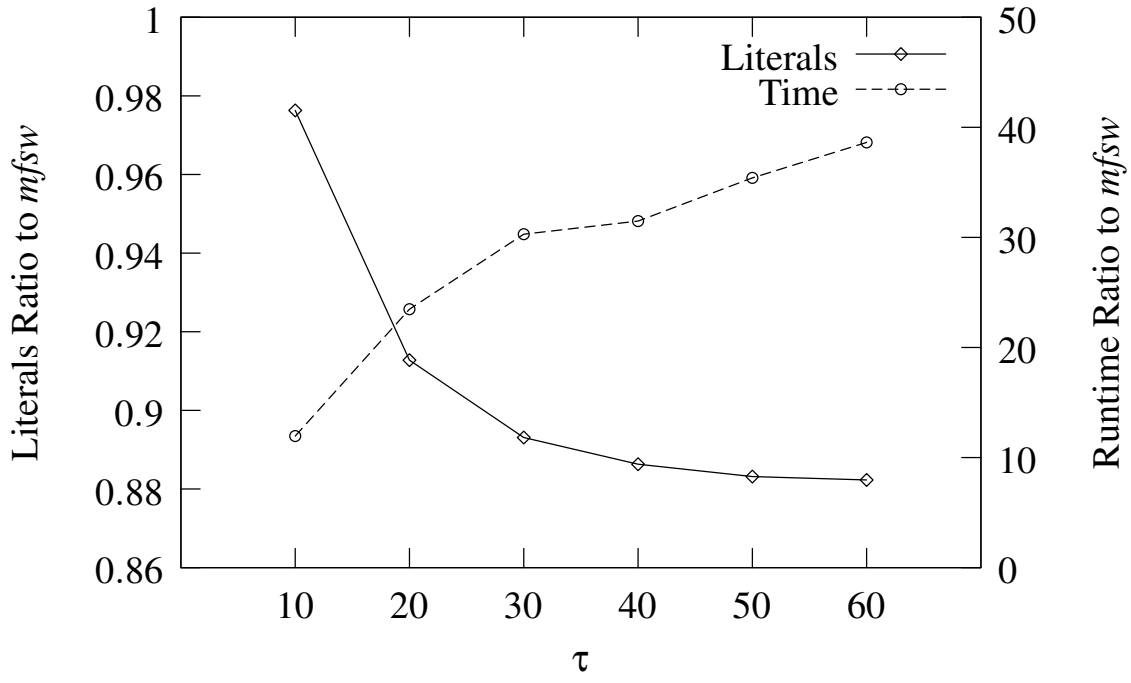

Fig. IV.5. Sweeping τ to Limit Subnetwork Size

selected with a low α. So while *all* pairs selected have many common fanouts, this table shows that sharing common fanins as well allows more flexibility, as expected.

The next three properties deal with the level of the two nodes in the circuit. Nodes that are closer in level to each other are also more likely to minimize well. However, there is little correlation between absolute levels of the nodes and the minimization ability.

The next two properties show that while the number of literals of the node pair is not important, the number of cubes does have a weak correlation with minimization ability. When nodes have more cubes in them, it is harder for the relation to find the flexibilities that can lead to minimization.

From the rest of the properties, a major trend emerges. The larger the size of the subnetwork, which yields a larger relation, the less minimization can be done. Again, this defies intuition at first, because an optimal solution would include all nodes in the network.

However, when the subnetwork is created, node pairs with fewer fanins and fanouts in common require more nodes to build the relation on.

The high correlation between subnetwork size and ineffective node pairs warrants an additional experiment. After the size of the subnetwork is determined for each node pair processed by the *relation* algorithm, if the number of nodes in the subnetwork is above a certain value τ, then the subnetwork is destroyed and the next pair is fetched. This effectively puts a limit on how big the subnetwork can be and saves runtime by not minimizing these node pairs.

Figure IV.5 shows the literal and runtime ratios of *relation* to *mfsw*, as the maximum allowable number of nodes in the subnetwork τ increases. The data shows that by omitting node pairs that create a subnetwork larger than $\tau = 40$ nodes, the runtime is reduced significantly with only a slight increase in literal ratio. This is consistent with the data in Table IV.5, which found that node pairs with large subnetworks are not good candidates for minimization.

IV-G. Chapter Summary

This chapter presents the experimental results for the algorithm described in the previous chapter. Section IV-B describes the preprocessing steps used in all the experimental results. Section IV-C shows the methodology used to determine the the parameters that control the node selection algorithm. Section IV-D reports the results obtained when comparing with the *mfsw* approach. Section IV-E reports the results of other experiments performed, with modifications of the the original *relation* algorithm. Section IV-F shows the properties of those node pairs that either did or did not produce gains when minimized, and reports the results of the additional experiment using these properties. Conclusions based on the experimental data obtained are discussed in the following chapter.

CHAPTER V

CONCLUSIONS

In this thesis, a scalable dual-node technology independent logic optimization technique was presented. This technique scales well and can minimize both small designs and large designs typical of industrial circuits.

The algorithm presented in this thesis first selects which node pairs will be minimized. Rather than minimizing all possible node pairs, only those that are likely to give gains are selected. The selection algorithm looks at how many fanins and fanouts the two nodes have in common to decide whether or not they will be selected. The optimal parameters to the node selection algorithm were chosen through experimentation, and it was found that favoring common fanouts provided the best results.

For each node pair, a subnetwork is created around the nodes. This windowing is done in order to make this approach feasible for large industrial circuits. Once the subnetwork is created, the Boolean relation, which represents the flexibility of the nodes, is computed. During this process, early quantification is performed. Experimental results show that without early quantification, the BDD size can blow up exponentially during the intermediate calculations.

BREL is used to minimize the Boolean relation, and the new nodes replace the original nodes in the original circuit. This is done for all node pairs that were selected.

It is experimentally demonstrated that this technique produces minimized technology independent networks that are on average 12% smaller than networks produced by a single-node minimization technique called *mfsw*. Although the runtimes of *mfsw* are significantly smaller than this approach, the runtime for any given circuit using this approach is never more than four minutes. In addition, the memory usage is very low and is independent of the circuit size.

Additionally, the approach in this thesis can *further* reduce the literal count of networks that have already been minimized by *mfsw*, by 13%. This result shows how the increased flexibility from two-node minimization can simplify networks better than single-node techniques.

Some of the future work involves using a SAT-based approach for constructing the Boolean relation. An alternative SAT-based replacement for BREL can be implemented as well. Both of these have the potential to reduce runtimes of the technique. In addition, modifications to minimize three or more nodes simultaneously can be made to gain even more flexibility using the Boolean relation-based multi-output optimization technique.

One final observation is that processing node pairs in parallel rather than serially would greatly reduce runtimes on multi-core systems. To do this, a method for generating compatible relations would have to be formulated. This means that if a node pair is minimized with respect to its Boolean relation, the Boolean relations of all other node pairs are still valid and do not need to be recomputed. Doing so will allow more node pairs to be processed in the same amount of runtime.

REFERENCES

[1] Soha Hassoun, Ed., *Logic Synthesis and Verification*, Kluwer Academic Publishers, Norwell, MA, Nov 2001.

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

[3] Y. Jiang and R. K. Brayton, "Don't cares and multi-valued logic network minimization," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov 2000.

[4] R. K. Brayton and F. Somenzi, "Boolean relations and the incomplete specification of logic networks," in *Proc. Intl. Conf. on VLSI*, Aug 1989.

[5] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," in *Proc. IEEE*, Feb 1990, pp. 264–300.

[6] B. Wurth and N. Wehn, "Efficient calculation of boolean relations for multi-level logic optimization," in *Proc. European Design and Test Conference*, Feb 1994, pp. 630 – 634.

[7] K. C. Chen and M. Fujita, "Efficient sum-to-one subsets algorithm for logic optimization," in *Proc. Design Automation Conference*, 1992, pp. 443–448.

[8] R. E. Bryant, "Graph based algorithms for Boolean function representation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–690, August 1986.

[9] F. Somenzi, "CUDD: CU decision diagram package," Accessed November 2007, [Online]. Available: http://vlsi.colorado.edu/ fabio/CUDD/cudd.html.

[10] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic model checking with partitioned transition relations," in *Proc. Intl. Conf. on VLSI*, August 1991.

[11] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve Boolean relations," in *Proc. Design Automation Conf.*, 2004, pp. 416–421.

[12] J C Limqueco and S Muroga, "Optimizing large networks by repeated local optimization using windowing scheme," in *IEEE International Symposium on Circuits and Systems, ISCAS*, May 1992, vol. 4, pp. 1993–1996.

[13] J C Limqueco and S Muroga, "SYLON-REDUCE: An MOS network optimization algorithms using permissible functions," in *Proc. Intl. Conf. on Computer-Aided Design*, Sept 1990, pp. 282–285.

[14] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. on CAD/ICAS*, vol. CAD-6(6), pp. 1062–1082, Nov 1987.

[15] S Dey, F Brglez, and G Kedem, "Circuit partitioning and resynthesis," in *Proc. Custom Integrated Circuits Conf.*, May 1990, pp. 29.4/1 –29.4/5.

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

[17] R Brayton, "Compatible output don't cares revisited," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov 2001, pp. 618–623.

[18] H Savoj, R Brayton, and H Touati, "Extracting local don't cares for network optimization," in *Proc. IEEE Transactions on Computer-Aided Design*, Nov 1991, pp. 514–517.

[19] N. Saluja and S. P. Khatri, "A robust algorithm for approximate compatible observability don't care (CODC) computation," in *Proc. Design Automation Conf.*, 2004, pp. 422–427.

[20] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[21] A. Mishchenko and R. K. Brayton, "SAT-based complete don't care computation for network optimization," in *Proc. Design, Automation and Test in Europe*, 2005, pp. 412–417.

[22] S. Sinha and R. K. Brayton, "Implementation and use of SPFDs in optimizing Boolean networks," in *Proc. Intl. Conf. on Computer-Aided Design*, 1998, pp. 103–110.

[23] S. Yamashita, H. Sawada, and A. Nagoya, "A new method to express functional permissibilities for LUT based FPGAs and its applications," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov 1996, pp. 254–261.

[24] E. Cerny and M. A. Marin, "An approach to unified methodology of combinational switching circuits," in *Proc. IEEE Transactions on Computers*, Aug. 1977, vol. 26, pp. 745–756.

[25] H. Savoj and R. K. Brayton, "Observability relations for multi-output nodes," in *Proc. International Workshop on Logic Synthesis*, May 1993.

[26] Y. Watanabe and R. Brayton, "Heuristic minimization of multi-valued relations," in *Proc. IEEE Transactions on Computer-Aided Design*, Oct 1993, pp. 1458–1472.

[27] F. Somenzi and R. K. Brayton, "An exact minimizer for Boolean relations," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov 1989, pp. 316–319.

[28] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *Proc. Intl. Conf. on Computer-Aided Design*, 2004, pp. 50–57.

VITA

Jeffrey Cobb was born in Ft. Worth, Texas in 1982. In 1993, he moved with his family to Sugar Land, Texas and attended Clements High School where he played trumpet in the marching and symphonic bands. He graduated from Clements in 2000 and enrolled that year in the electrical engineering program at Texas A&M University in College Station, Texas. During the summers of his undergraduate career, he worked at Halliburton and Dell as an engineering intern. In his senior year, he participated in the Undergraduate Research Fellows Program where he studied automated test pattern generation methods under Dr. M. Ray Mercer. He received his bachelors degree in 2004 and enrolled in the computer engineering department at Texas A&M University the same year to earn his masters degree. His area of interest includes logic synthesis and logic minimization. He graduated with a Master of Science in computer engineering from Texas A&M in December of 2007. He can be contacted at the Electrical and Computer Engineering Department, c/o Dr. Sunil P. Khatri, MS 3259, Texas A&M University, College Station, TX 77843.

The typist for this thesis was Jeffrey Cobb.