COMPUTING LEAKAGE CURRENT DISTRIBUTIONS AND DETERMINATION OF MINIMUM LEAKAGE VECTORS FOR COMBINATIONAL DESIGNS

A Thesis

by

KANUPRIYA GULATI

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

May 2006

Major Subject: Computer Engineering

COMPUTING LEAKAGE CURRENT DISTRIBUTIONS AND DETERMINATION OF MINIMUM LEAKAGE VECTORS FOR COMBINATIONAL DESIGNS

A Thesis

by

KANUPRIYA GULATI

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,	Hank Walker
	Peng Li
Head of Department,	Costas Georghiades

May 2006

Major Subject: Computer Engineering

ABSTRACT

Computing Leakage Current Distributions and
Determination of Minimum Leakage Vectors for Combinational Design. (May 2006)
Kanupriya Gulati, B.E., University of Delhi
Chair of Advisory Committee: Dr. Sunil Khatri

Analyzing circuit leakage and minimizing leakage during the standby mode of operation of a circuit are important problems faced during contemporary circuit design. Analysis of the leakage profiles of an implementation would enable a designer to select between several implementations in a leakage optimal way. Once such an implementation is selected, minimizing leakage during standby operation (by finding the minimum leakage state over all input vector states) allows further power reductions. However, both these problems are NP-hard. Since leakage power is currently approaching about half the total circuit power, these two problems are of prime relevance.

This thesis addresses these NP-hard problems. An Algebraic Decision Diagram (ADD) based approach to determine and implicitly represent the leakage value for all input vectors of a combinational circuit is presented. In its exact form, this technique can compute the leakage value of each input vector, by storing these leakage values implicitly in an ADD structure. To broaden the applicability of this technique, an approximate version of the algorithm is presented as well. The approximation is done by limiting the total number of discriminant nodes in any ADD. It is experimentally demonstrated that these approximate techniques produce results with quantifiable errors. In particular, it is shown that limiting the number of discriminants to a

value between 12 and 16 is practical, allowing for good accuracy and lowered memory utilization.

In addition, a heuristic approach to determine the input vector which minimizes leakage for a combinational design is presented. Approximate signal probabilities of internal nodes are used as a guide in finding the minimum leakage vector. Probabilistic heuristics are used to select the next gate to be processed, as well as to select the best state of the selected gate. A fast satisfiability solver is employed to ensure the consistency of the assignments that are made in this process. Experimental results indicate that this method has very low run-times, with excellent accuracy, compared to existing approaches.

To my parents

ACKNOWLEDGMENTS

Any record of work is incomplete without an expression of gratitude towards those who made it possible. I would like to thank, at the outset, the chair of my committee, Dr. Sunil Khatri. I am grateful to him for his academic guidance, continued support and above all being a source of inspiration all along. I thank my committee members, Dr. Hank Walker and Dr. Peng Li, for their valuable suggestions, availability and timely criticism. I greatly value the knowledge they have imparted to me. I also thank the administrative staff of the Department of Electrical and Computer Engineering for their timely help and continued cooperation with paperwork and other administrative matters.

I value the discussions, both academic and non-academic, with Nikhil, Rajesh and the rest of the folks in the computer engineering research group. Nothing would have been possible, of course, without my family, because of whom I am what I am, and where I am, today.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

In CMOS technologies, the sources of power consumption can be classified as dynamic power consumption (switching power), static power consumption (leakage power) and short-circuit power (which, for most circuits is less than 20% of the dynamic power). Traditionally, dynamic (switching) power has dominated the total power consumption of a VLSI IC. However, due to current scaling trends, leakage power has now become a major component of the total power consumption in VLSI circuits. The leakage current for a PMOS or NMOS device corresponds to the I_{ds} of the device when the device is in the *cut-off* or *sub-threshold* region of operation. The expression for this current [1] is:

$$I_{ds} = \frac{W}{L} I_0 e^{(\frac{V_{gs}-V_T-V_{off}}{nv_t})}(1 - e^{(-\frac{V_{ds}}{v_t})}) \tag{1.1}$$

Here I_0 and V_{off}[1] are constants, while v_t is the thermal voltage (26mV at 300°K) and n is the sub-threshold swing parameter. Note that I_{ds} increases exponentially with a decrease in V_T. This is why a reduction in supply voltage (which is accompanied by a reduction in threshold voltage) results in exponential increase in leakage. This is expected to be a major concern for VLSI design in the nanometer realm [2]. Further, the increasing popularity for portable/hand-held electronics has meant that leakage power consumption has received even greater attention. Since these portable devices spend most of their time in a *standby* state (also sometimes called the *sleep* state), reducing the leakage power consumption in this *standby* state is crucial to extending the battery life of these designs.

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

[1]Typically $V_{off} = -0.08V$

One of the natural techniques for reducing the leakage of a circuit is to gate the power supply using power-gating transistors (also called *sleep* transistors). Typically high-V_T power-gating transistors are placed between the power supply and the logic gates (MTCMOS [3, 4]). In some cases these power-gating transistors are embedded in the logic gates itself [5]. In standby, these power-gating transistors are turned off, thus shutting off power to the circuit in question. Such power-gating techniques can reduce circuit leakages by 2 to 3 orders of magnitude. However, the addition of a power-gating transistor causes an increase in the delay of the circuit. Further, the process of waking the circuit up involves a delay (and a power transient), since the supply rails need to reach their stable values before the circuit can operate again.

Increasing V_T via body effect and bulk voltage modulation [6, 7] is another way to reduce leakage power. The leakage current of a transistor decreases with greater Reverse Body Bias (RBB). RBB affects V_T through body effect, and sub-threshold leakage has an exponential dependence on V_T as seen in Equation 1.1. The body effect equation can be written as $V_T = V_T^0 + \gamma\sqrt{V_{sb}}$ where V_T^0 is the threshold voltage at zero V_{sb}. However, with current technology scaling, the body effect factor γ is reducing and body biasing usually only yields a best-case decrease in leakage of about $10\times$.

All the techniques listed above require significant circuit modifications in order to reduce leakage. Another technique, which proclaims up to 2 orders of magnitude leakage reduction, is the technique of *parking* a circuit in its minimum leakage state (or vector). This technique involves little or no circuit modification and does not require any additional power supplies. A combinational circuit is *parked* in a particular state by driving the primary inputs of the circuit to a particular value. This value can be scanned in or forced using MUXes (with the standby/sleep signal used as a select signal for the MUX). The leakage minimization approach presented in this thesis falls under this category.

With leakage power increasing as a fraction of the total power of a design, due to the current design trends, it is no longer sufficient to simply find the input vector that minimizes circuit leakage. It is arguably more important to find the leakage for all input vectors. This is useful when comparing candidate implementations of a design with the same minimum leakage values. An implementation, that has a leakage histogram with larger number of input vectors contributing to lower leakage values, would be preferred over other implementations. This would not only minimize the leakage during the regular operation of the circuit, but also ease the task of finding a vector which results in minimum leakage state.

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

A. Problem Definition

Table A shows the leakage of a NAND3 gate for all possible input vectors to the gate. The leakage values shown are from a SPICE simulation using the 0.1μ BPTM [8] models, with a VDD of 1.2V.

As can be seen from Table A, setting a gate in its minimal leakage state (000 in the case of the NAND3 gate) can reduce leakage by about 2 orders of magnitude. Ideally, it is desirable to set *every* gate in the circuit to its minimal leakage state. However, this may not be possible due to the logical inter-dependencies between the inputs of the gates. Finding this minimum leakage state (or the input vector to park the circuit in its minimum leakage state) is an NP-hard problem. It is worth noting that the reduction in circuit leakage may not be very conspicuous for a large random logic design due to the law of large numbers. In the case of a very large

Table I. Leakage of a NAND3 Gate

Input	Leakage(A)
000	1.37389e-10
001	2.69965e-10
010	2.70326e-10
011	4.96216e-09
100	2.62308e-10
101	2.67509e-09
110	2.51066e-09
111	1.01162e-08

random logic design, the ratio of the maximum leakage to the minimum leakage is approximately close to unity, i.e. the leakage is relatively constant. The results obtained in the experimental section of this thesis are viewed in the context of the law of large numbers as well.

It is important to note that with leakage power increasing as a fraction of the total power of a design, it is no longer sufficient to simply find the input vector that minimizes circuit leakage. It is arguably more important to find the leakage for *all* input vectors (of course, the minimum leakage vector can be found by this exercise). When comparing candidate implementations of a design with the same minimum leakage values, one would prefer the design that has a leakage histogram with the largest number of input vectors contributing lower leakage values. This would not only minimize the leakage during the regular operation of the circuit, but also ease the task of finding a vector which results in minimum leakage. It was reported in [9]

that the maximum leakage value of a design can be as high as 2.4× the minimum value (1.6× on average), again underscoring the importance of computing the leakage of *all* input vectors for implementations and choosing one with a favorable leakage histogram.

This thesis addresses both problems stated above. Clearly, an explicit representation of all leakage values would be infeasible. The problem of computing the leakage of all input vectors for a design is approached as follows. An Algebraic Decision Diagram (ADD) based approach is proposed to represent the leakage values of a circuit. The problem of building an ADD to implicitly represent the exact[2] leakage values of a design has been formulated and solved. In order to expand the applicability of this approach to larger designs, a method to implicitly compute the approximate leakage values of a design is also presented. These approaches can be used to construct the histogram of leakage values for a design. This data is beneficial when comparing two or more candidate implementations (with similar maximum leakage values) of a single circuit. Experimental data indicates that the approximate calculation of leakage values demonstrated a bounded loss of accuracy, with a significant improvement in the efficiency of the technique. Leakage histograms for area-mapped and delay-mapped versions of some benchmark circuits were computed, and their leakage characteristics were compared.

Several research efforts have addressed the problem of determining an input vector that minimizes the leakage of a design. This problem is also called Input Vector Control (IVC). An efficient heuristic to determine the minimum leakage vector (i.e. the input vector which drives the circuit to its lowest leakage state) is proposed in the thesis. This problem can be viewed as one of selecting the state of each gate in

[2]The term *exact* used here and in the sequel refers to an *algorithmic* exact as opposed to an absolute exact.

the circuit such that the total leakage over all gates is minimized, and the state of each gate in the circuit is logically feasible (i.e. is logically compatible with states of all the other gates). The key idea presented here uses signal probabilities as a guide to determine the minimum leakage vector. In other words, the selection of the best candidate gate, as well as the input state to use for that gate, will be performed probabilistically.

Some of the previous work done in these two areas is discussed in the following section.

B. Previous Work

The problem of finding the minimum leakage sleep vector for a combinational CMOS gate-level circuit has received some attention recently. Researchers have used models and algorithms to estimate the nominal leakage current of a circuit [10, 11, 12]. In [13], the authors find a minimal leakage vector using random search with the number of vectors used for the random search selected so as to achieve a specified statistical confidence and tolerance. In [11], the authors reported a genetic algorithm based approach to solve the problem. The authors of [14] introduce a concept called leakage observability, and based on this idea, describe a greedy approach as well as an exact branch and bound search to find the maximum and minimum leakage bounds. The work of [9] is based on an ILP formulation. It makes use of pseudo-Boolean functions which are incorporated into an optimal ILP model and a heuristic mixed integer linear programming method as well. In contrast to these approaches to solve IVC, the approach described in this thesis is a heuristic that uses signal probabilities and leakage values of the gates to help assign values to the nodes in a combinational

circuit.

In [15], the authors present a greedy search based heuristic, guided by node controllabilities and functional dependencies. The algorithm used in [15] involves finding the controllability and the controllability lists of all nodes in a circuit and then using this information as a guide to choose the gates to set to a low leakage state. The controllability of a node is defined as the minimum number of inputs that have to be assigned to specific states in order to force the node to a particular state (based on concepts used in automatic test pattern generation). Controllability lists are defined as the minimum constraints necessary on the input vector to force a node to particular state. The time complexity of their algorithm is reported to $O(n^2)$ where n is the number of cells (gates) in the circuit. However, in estimating the complexity of their algorithm, it is not clear if the authors include the time taken to generate the controllabilities and controllability lists of each node in the circuit. While finding the controllabilities can be done fairly easily [16], generating the controllability lists can be more involved. In the approach followed in this thesis, node controllabilities or their controllability lists are not computed. Instead signal probabilities are computed. This can be done in O(n) time. The algorithm for this is detailed in Section C in Chapter III. In [17], the authors describe several methods to set pass/fail limits for IDDQ testing, among which is a probabilistic method. For each cell in a design (each cell is assumed to have a single output, implemented in static CMOS), the authors compute the maximum IDDQ when the output is ON (OFF), assuming 4σ process variation limits. Additionally, the cell probabilities are determined for the input vectors that result in the maximum IDDQ of the cell for both the ON and OFF state. In contrast to [17], the approach in this thesis takes into account probabilities of all input vectors of a cell implicitly, and not just those of two outputs that result in a worst-case IDDQ value. Further, the signal probabilities, in the heuristic presented

in this thesis, are adjusted for reconvergence, unlike [17].

In [18, 19], the authors formulate the problem of finding a minimum leakage vector as a satisfiability problem. In [18] the authors use an incremental SAT solver to find the minimum and maximum leakage current. While their approach worked well for small circuits, the authors report very large runtimes for large circuits. The authors therefore suggest using their algorithm as a checker for the random search suggested in [13]. In [19], the authors introduced a method for controlling the internal nodes by modifying some gates, without using extra multiplexers. In addition, the delay constraints are explicitly accounted for and the optimal subset of internal nodes of the circuit to be controlled is determined by the SAT formulation. The approach described for IVC in this thesis requires a SAT solver as well, but does not involve internal node modifications, which makes it computationally tractable. Moreover, larger designs are handled more easily, since the SAT solver is invoked only to verify the state assignments of individual gates, after every k iterations. The frequency with which the SAT solver is invoked is decided using experimental data, in order to run large circuits with low run-times and good accuracy.

Once the minimum leakage input pattern is found, this vector is used to drive the circuit in its standby mode. This may require the addition of a number of multiplexers at the primary inputs of the circuit. The multiplexers are controlled using a *sleep* signal. (In a scan based design, these multiplexers are not required). Since the power reduction using these techniques can be achieved only for sleep durations that are sufficiently long, the sleep signal is activated only if the sleep duration is long enough.

In [20], the authors use ADDs to find the leakage of a channel-connected region (CCR) as a function of its inputs. The focus in [20] was on full-custom circuitry and the authors used their technique to find functional failures in CCRs due to excessive leakage (input vectors that caused leakage to go above a certain value). Exclusivity

constraints were added to constrain the ADD of a CCR to legal input vectors.

All of the techniques cited above attempt to compute a *single* vector which results in a minimum (or maximum) leakage state. An approach to compute the leakage values for *all possible input combinations* is also presented in this thesis. Using Algebraic Decision Diagrams (ADDs) [21, 22], the leakage of the circuit for *all* input vectors are *implicitly* represented in a single structure. The inherent sharing of nodes in such a structure allows for a compact representation of the leakage of the design. In order to improve the efficiency of the leakage ADD construction, the values of the leaf nodes are binned so as to reduce the number of leaf nodes of the ADD. This reduces the number of discriminants[3] (as well as the number of nodes) in the leakage ADD of the design. The histogram of leakage values (constructed from the leakage ADD) is used for comparing candidate implementations of a circuit. In [23], the authors also present an ADD based algorithm to determine the lowest leakage state of a circuit. They partition a circuit η into subcircuits and determine the minimum leakage value and IVC of each subcircuit. These leakage values are then summed in order to generate the minimum leakage value of η and the IVC for η is generated by concatenation of the IVCs of the subcircuits. This may, clearly, not lead to a global minima since the search space is greatly pruned by propagating only the minimum leakage (and the corresponding IVC) for each subcircuit. This is not the case with the approach described in this thesis since here the entire range of leakage values are binned as opposed to pruning of all the leakage values except the minimum (or maximum) for the individual subcircuits.

[3]The number of discriminants of an ADD is the number of unique leaves of the ADD

C. Organization

This section serves as an introduction to the content of the thesis. Chapter II provides a background on the data structures and certain functions based on them, which are used extensively in this thesis. In Chapter III, the algorithms for computing the exact and approximate leakage values for all input vectors are presented. The algorithm for determining the minimum leakage vector is also presented in another section of the same chapter. The approximations and steps used in this algorithm are discussed as well.

Chapter IV contains the experimental results for the algorithms described in Chapter III. Section B reports the results obtained for the approximate leakage ADD computed with varying numbers of discriminants. In addition, this section shows the comparison of two different implementations of a few designs with respect to their leakage histograms. Section C compares the leakage values obtained by the probabilistic IVC algorithm with the exact minimum leakage for small designs. For the large designs, the comparison is made against the approximate minimum leakage generated after an appropriately large number of random simulations. The thesis is summarized in Chapter V, with a discussion on the key conclusions that can be drawn from the results.

CHAPTER II

BACKGROUND

A. Chapter Overview

This chapter serves as an introduction to the various functions and data structures which are used in Chapter III.Section B explains the Reduced Ordered Binary Decision Diagram (ROBDD) data structure and the functions on ROBDDs used in the algorithms in this thesis. Section C discusses the properties of Algebraic Decision Diagrams (ADDs) and the ADD functions used in this thesis. The chapter is concluded in Section D.

B. Reduced Ordered Binary Decision Diagrams (ROBDD)

An ROBDD is a graphical representation of a Boolean function. It can represent many logic functions compactly as compared to a sum-of product (SOP) or a truth table representation. Moreover, several logic operations like tautology checking and complementation can be performed on ROBDDs in constant time. For a particular variable ordering, an ROBDD is a canonical form of representing a Boolean function. However, it is more efficient in memory utilization as compared to a truth table which is another canonical representation of a Boolean function. As the name suggests, ROBDDs are a *reduced* form of BDDs with a particular *variable ordering*. The structure of the BDD and the reduction rules followed are described in the sequel.

A BDD represents a Boolean function as a directed acyclic graph (DAG), with each nonterminal node assigned to a variable of the function. It is also referred to as a Shannon co-factoring tree. Each node performs the Shannon co-factoring of the

Boolean function represented by that node, with respect to the variable assigned to it. Figure 1 illustrates the BDD for the function $(x_1+x_2)\cdot x_3$. Each node has two outgoing edges, corresponding to the positive cofactor of the node function with respect to the node variable (shown as a solid line) or the negative cofactor of the node function with respect to the node variable (shown as a dashed line). The terminal nodes (shown as boxes) are labeled with 0 or 1, corresponding to the possible function values. For any assignment to the function variables, the function value is determined by tracing a path from the root of the BDD to a terminal node following the appropriate positive or negative branch from each node. The number of vertices in the BDD is exponential in terms of number of variables in the logic function. Therefore, for functions with a large number of variables, BDDs may not be a good choice for representing the function. In general, the variable ordering along different paths in the BDD can be different.

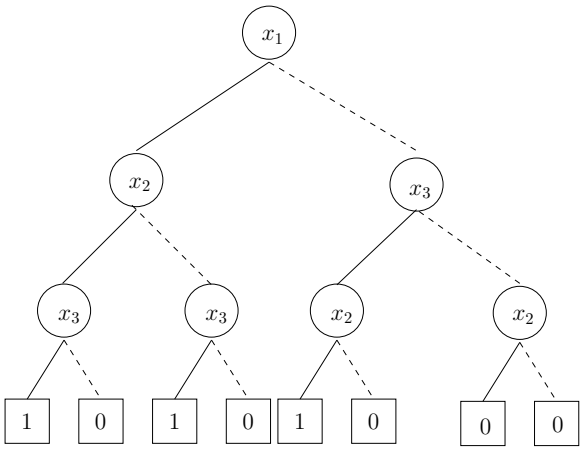


Fig. 1. Shannon Cofactoring Tree of Logic Function $(x_1 + x_2) \cdot x_3$

The graph in Figure 1 is transformed into an ordered BDDs (OBDDs) if we use a fixed variable ordering along any path from root to leaves. Consider the variable

to be order $x_1 < x_2 < x_3$. That is, every path from the root to a leaf encounters variables in the order $x_1 < x_2 < x_3$. The resulting OBDD is shown in Figure 2. In addition, on application of the following reduction rules on the OBDD, an ROBDD for the function is obtained.

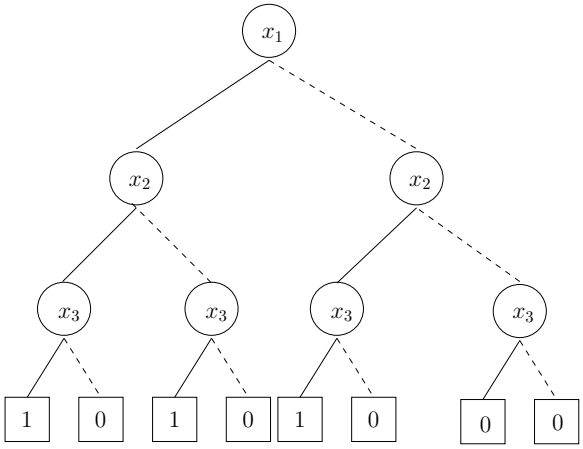


Fig. 2. OBDD of Logic Function $(x_1 + x_2) \cdot x_3$

- Remove nodes which have identical children.
- Merge nodes which have isomorphic BDDs.

ROBDDs are a canonical representation of a logic function for a given variable ordering. Figure 3 shows the resulting ROBDD when the above mentioned reduction rules are applied to the OBDD shown in Figure 2. Note that even in an ROBDD, the number of nodes can be exponential in terms of the number of variables. However, the size of ROBDDs (i.e. number of nodes) depends upon the variable ordering. Therefore, variables must be ordered in a manner that minimizes the size of the ROBDD. Computing an optimum variable ordering is an NP-Complete problem. However there are efficient heuristics available that can choose an appropriate ordering of variables

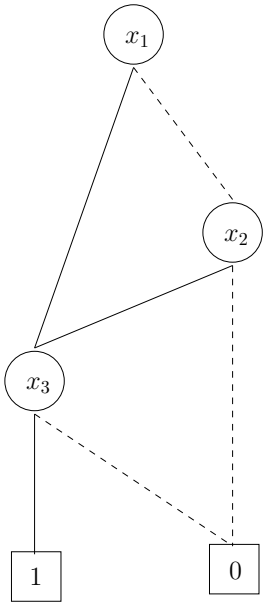


Fig. 3. ROBDD for Logic Function $(x_1 + x_2) \cdot x_3$

which results in the ROBDD of reasonable size. However, there are functions that have polynomial sized multi-level representations while their ROBDDs are exponential for all input orderings. A multiplier is an example of such a function. The terms ROBDD and BDD are used interchangeably in the rest of this document.

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

- *bdd_find_minterm(f)*: This function returns one cube or minterm from all the existing cubes or paths to terminal node '1' of the BDD for f. This path is basically a cube in the onset of the Boolean function represented by f.

- *bdd_count_onset(f,var_array)*: This function counts the number of minterms in the onset of the function f, over the variables in var_array (single variable BDD formulas). var_array must contain the variables in the support of f. For example, if f=$b \cdot d$, and var_array=[a,b,c,d], then this function returns 4.

- *bdd_substitute(f, old_array, new_array)*: This function substitutes all variables from the array *old_array* with the corresponding variables from the array *new_array* in the BDD of 'f'. *old_array* and *new_array* are arrays of BDDs with equal cardinality. Given two arrays of variable BDDs a and b consisting of member values (a_1 .. a_n) and (b_1 .. b_n), this function replaces all occurrences of a_i by b_i in f. This operation linear in the number of nodes in the BDD representation of f.

C. Algebraic Decision Diagrams (ADD)

BDDs with multiple terminal nodes are called Multi-terminal BDDs (MTBDD). Due to their applicability to different algebras (including Boolean algebra) the term Algebraic BDD was coined in [21]. A BDD can be viewed as an ADD with of terminal values from the set {0,1}. An ADD with n terminals has terminal values selected from the set { a_1, a_2, $\cdots$, a_n}, where a_i are algebraic or symbolic values. The values are also called *discriminants* of the ADD.

Some general properties of ADDs are as follows.

- ADDs are canonical. When dealing with ADDs with a large number of discriminants the usefulness of this property may decrease.
- Edge attributes such as complementation flags may be of limited utility, because complementation in Boolean algebra may not have a meaningful counterpart in the ADD context.
- These factors leads to a recombination efficiency (which arises due to sharing of isomorphic subgraphs) which is relatively small in comparison to BDDs.
- In comparison to other sparse data structures, ADDs provide a uniform $log(N)$

access time where N is the number of real numbers being stored in the ADD.

- ADDs cannot beat sparse matrix data structures in terms of worst case space complexity. However, recombinations of isomorphic subgraphs may give considerable practical advantage to ADDs over other data structures.

An example of an ADD on three variables x_1, x_2 and x_3 is shown in Figure 4. The discriminants here are not restricted to {0,1}. Also, note that the sharing mechanism is similar to that in a BDD, but since the terminal nodes can be any numeric (or symbolic) value, the number of nodes shared could be fewer as compared to those in a BDD.

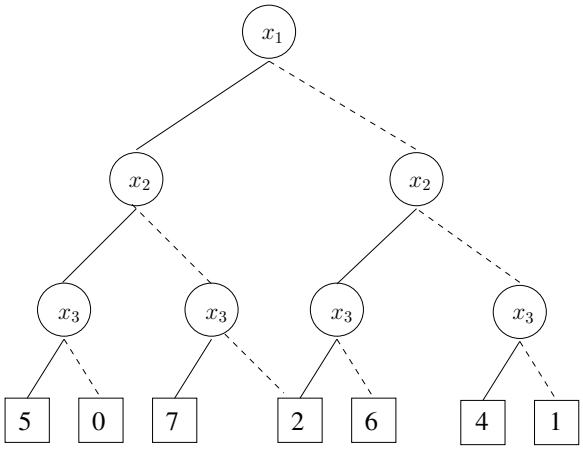


Fig. 4. An Example ADD on Three Variables x_1, x_2 and x_3

.

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

- *ITE(f,g,h)*: The If-Then-Else(ITE) function takes three arguments. The first is an ADD restricted to have only 0 or 1 as terminal values. The second and third arguments are generic ADDs. *ITE* is defined as

$ITE(f, g, h) = f \cdot g + f^{'} \cdot h$

ITE can be applied as a recursive procedure for traversing through an entire ADD structure.

- *ADD_threshold(f,g)*: This function thresholds the discriminants of ADD f against a constant g. If the value of a terminal node is greater that or equal to g, it keeps the terminal node value as it is, else assigns the terminal node to a value 0 or *FALSE*.

- *ADD_to_BDD(f,t)*: This function is identical to *ADD_threshold(f,t)* except that when the value of a a terminal node is greater than or equal to t, the terminal node is assigned the value 1 or logical *TRUE*. In effect, the decision diagram is left with terminal nodes belonging to the set {0,1} and hence is now a *BDD*.

- *cofactor(f,g)*: This function returns Shannon cofactor of an ADD f with respect to ADD g. g must be an ADD or a BDD of a cube.

D. Chapter Summary

This chapter explains the BDD and ADD date structures, which are used extensively in this thesis. The general properties of these data structures are discussed along with some functions based on these data structures, that are used by the algorithms presented in this thesis.

CHAPTER III

APPROACH

A. Chapter Overview

This chapter is divided into the following sections. Section B describes the approach for computing the exact and approximate leakage values for all input vectors for a circuit. The algorithms are explained in detail (along with pseudocode), followed by a discussion on the expandability and applications of the approach. Section C presents a probabilistic heuristic (which is guided by signal probabilities) to determine the input vector which drives the circuit into its lowest leakage state. Again, the pseudocode is explained in detail after a brief outline of the heuristic. Section D concludes the chapter.

B. Computing Leakage Values Across All Input Vectors

The approach described in this thesis is based on an ADD [21, 22] based computation, which enables the determination of the leakage values for all possible input vectors in the design. The approach described in this section is termed as AL^{all}. The exact version of AL^{all} is called AL^{all}_{ex}, while the approximate version is called AL^{all}_{app}. The determination the leakage values for all input vectors is useful in several contexts, such as

- It allows the computation of the average, minimum and maximum leakage for the design in an accurate manner.
- It allows the construction the histogram of leakage values for a design. This can be of use when comparing two or more candidate implementations (with

similar minimum or maximum leakage values) of a single circuit. The design with a leakage histogram that is skewed towards the lower leakage values would be preferred, since it would reduce dynamic power under normal operation. For example, during dynamic operation, the circuit may switch between repeatedly between a set of vectors. In this case, the implementation which has a leakage histogram skewed towards lower leakage values would be preferred. Figure 5 illustrates this idea. The leakage histograms of two designs (with similar maximum leakage values) is shown. The histogram to the right is preferred, since it has a large number of vectors with low leakage values.

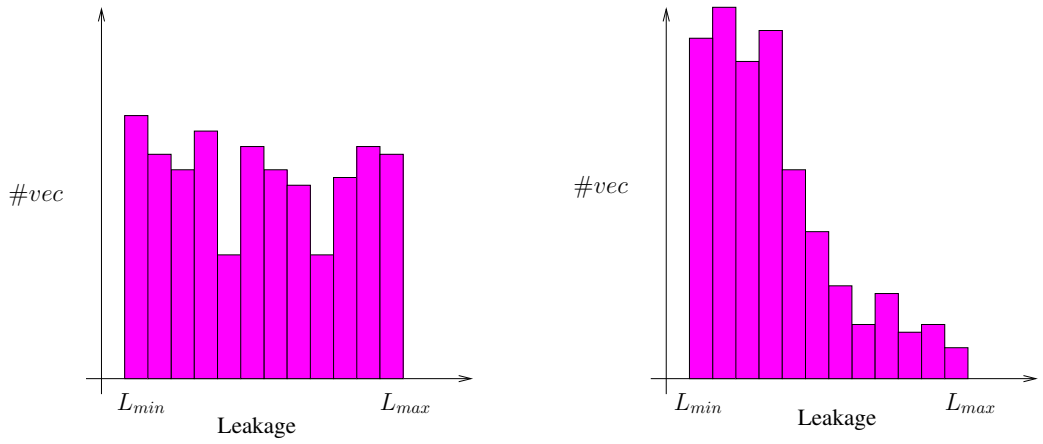


Fig. 5. Leakage Histograms for Two Implementations of a Design

1. Exact Computation of the Leakages of all Vectors

In order to compute the exact leakages of all vectors, the approach, called AL_{ex}^{all}, is described below. Consider a combinational logic network η, consisting of logic gates G_j selected from some library P. The ROBDD of G_j is referred to as g_j, and the

leakage ADD of G_j as $\mathcal{G}_j$. This ADD represents the leakage value of each primary input minterm m of g_j (obtained by following the path from the root, indicated by the literals of m, until a terminal vertex is reached). The value of this vertex is the leakage of G_j under the input m. Note that the support of $\mathcal{G}_j$ is the primary inputs of the circuit.

Assume that for each gate G_j, there is an array called ($lkg\_array(G_j)$) describing its leakage values for all possible values of its immediate fanins. For example, if the G_j was a 2-input gate, then its leakage array would consist of 4 values, corresponding to all 4 possible input combinations for the gate. Let the 2 fanins be called H_1 and H_2. For ease of the exposition, assume that these are sorted in a numerical order, so that the leakage value of the input combination 00 appears first, followed by that of the input values 01, and so on. Suppose that under some primary input minterm m, the ROBDDs h_1 and h_2 evaluate to $h_{1_{val}}$ and $h_{2_{val}}$ respectively. The corresponding leakage value for the gate G_j is found by indexing the $(h_{1_{val}} : h_{2_{val}})^{th}$ value of $lkg\_array(G_j)$. For example, if $h_{1_{val}} = 1$ and $h_{2_{val}} = 0$, the second value of $lkg\_array(G_j)$ is indexed to obtain the appropriate leakage value.

The algorithm AL_{ex}^{all} proceeds as follows. It first finds the ROBDDs of all network nodes. Next, it finds the (global) leakage ADDs of each of the nodes in the network using Algorithm 1. Suppose the leakage ADD of H is computed. Assume that it has 2 fanins F and G. The leakage ADD of H is found by the subroutine $node\_compute\_lkg\_ADD(f, g, lkg\_array(H))$. In this routine, if the ROBDDs f and g are constant (f_{val} and g_{val} respectively), then the leakage value for this condition is simply found by indexing the $(f_{val} : g_{val})^{th}$ value of $lkg\_array(H)$ and returning an ADD node of this value. If either of f or g are non-constant, then the top variable v among these ROBDDs is returned. The computation recursively computes $\mathcal{H}_v$ and $\mathcal{H}_{\overline{v}}$, and finally returns $\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$.

Algorithm 1 The node_compute_lkg_ADD Algorithm

$node\_compute\_lkg\_ADD(f, g, lkg\_array(H)$

// terminal case below

if $f_{val} = is\_constant(f)$ && $g_{val} = is\_constant(f)$ **then**

$\mathcal{H} = create\_ADD\_node(f_{val} : g_{val})$

return $\mathcal{H}$

end if

$v = topvar(f, g)$

$f_v = cofactor(f, v)$

$f_{\overline{v}} = cofactor(f, \overline{v})$

$g_v = cofactor(g, v)$

$g_{\overline{v}} = cofactor(g, \overline{v})$

$\mathcal{H}_v = node\_compute\_lkg\_ADD(f_v, g_v, lkg\_array(H))$

$\mathcal{H}_{\overline{v}} = node\_compute\_lkg\_ADD(f_{\overline{v}}, g_{\overline{v}}, lkg\_array(H))$

$\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$

return $\mathcal{H}$

Algorithm 1 is applicable for gates G_j with two inputs. The technology library usually consists of at most 4-input gates. As a result, two additional routines similar to Algorithm 1, for 3 and 4 input gates are required.

Note that leakage ADDs of the mapped gates of the network need not be computed in any particular order. After the leakage ADDs of each gate have been computed, the leakage ADD of the entire circuit (this is referred to as $\mathcal{H}_{total}$), is found by adding each gate's leakage ADD. The routine to add two ADDs is shown in Algorithm 2. If the circuit has n gates, then this operation requires $n-1$ ADD addition operations, since the addition of ADDs is performed in a pair-wise manner.

Algorithm 2 first tests if the ADDs $\mathcal{F}$ and $\mathcal{G}$ to be added are both constants. If this is the case (call the constants $\mathcal{F}_{val}$ and $\mathcal{G}_{val}$) it creates and returns an ADD node with value $\mathcal{F}_{val} + \mathcal{G}_{val}$. If at least one of $\mathcal{F}$ or $\mathcal{G}$ are non-constant, then the top variable v is found among them. $\mathcal{H}_v = add\_ADD(\mathcal{F}_v, \mathcal{G}_v)$ and $\mathcal{H}_{\overline{v}} = add\_ADD(\mathcal{F}_{\overline{v}}, \mathcal{G}_{\overline{v}})$ is recursively computed, and $\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$ is returned.

Once $\mathcal{H}_{total}$ (the sum of all the leakage ADDs of the gates in the design) is computed, the minimum valued leaf L_{min} (which is the minimum discriminant of $\mathcal{H}_{total}$) of the final ADD is found. This discriminant corresponds to the lowest leakage state of the design. A primary input vector that results in this leakage value is found by using Algorithm 3. A similar exercise can be conducted for any discriminant, which enables the construction of a leakage histogram for the design.

Thresholding an ADD consists of the task of converting it into an ADD with fewer discriminants. $ADD\_threshold(\mathcal{H}, val)$ makes all discriminants with values greater than or equal to val point to the 0 discriminant. All discriminants with values less than val are retained in the result.

Algorithm 3 first thresholds $\mathcal{H}_{total}$ with the value $L_{min} + \delta$. The value δ is such that there is no leakage value for the design in the closed interval $[L_{min}, L_{min} + \delta]$. In

Algorithm 2 The add_ADD Algorithm

$add\_ADD(\mathcal{F}, \mathcal{G})$

// terminal case below

if $f_{val} = is\_constant(\mathcal{F})$ && $g_{val} = is\_constant(\mathcal{G})$ **then**

$\mathcal{H} = create\_ADD\_node(\mathcal{F}_{val} + \mathcal{G}_{val})$

return $\mathcal{H}$

end if

$v = topvar(\mathcal{F}, \mathcal{G})$

$\mathcal{F}_v = cofactor(\mathcal{F}, v)$

$\mathcal{F}_{\overline{v}} = cofactor(\mathcal{F}, \overline{v})$

$\mathcal{G}_v = cofactor(\mathcal{G}, v)$

$\mathcal{G}_{\overline{v}} = cofactor(\mathcal{G}, \overline{v})$

$\mathcal{H}_v = add\_ADD(\mathcal{F}_v, \mathcal{G}_v)$

$\mathcal{H}_{\overline{v}} = add\_ADD(\mathcal{F}_{\overline{v}}, \mathcal{G}_{\overline{v}})$

$\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$

return $\mathcal{H}$

Algorithm 3 Finding an Input Vector with Minimum Leakage L_{min}

$find\_a\_minterm\_with\_min\_leakage(\mathcal{H}_{total})$

$\mathcal{H}_{thresholded} = ADD\_threshold(\mathcal{H}_{total}, L_{min} + \delta)$

$h_{thresholded} = ADD\_to\_BDD(\mathcal{H}_{thresholded})$

return $BDD\_find\_minterm(h_{thresholded})$

other words, there is no discriminant in the leakage ADD $\mathcal{H}_{total}$ in the above closed interval. Therefore the resulting leakage ADD after thresholding ($\mathcal{H}_{thresholded}$) consists of exactly two discriminants (L_{min} and 0). Next, $\mathcal{H}_{thresholded}$ is converted into a BDD, by replacing the L_{min} discriminant by the 1 discriminant. A path to the 1 terminal node in this BDD is now found by using the well known linear-time BDD algorithm to find a single minterm.

In a similar manner, the BDD for any specific leakage value (i.e. any specific discriminant of the leakage ADD) can be found. For a general leakage value L other than the maximum or minimum, the thresholding with threshold values $L+\delta$ as well as $L-\delta$ needs to be done, where δ is such that there is no other discriminant of the leakage ADD in the interval $[L+\delta, L-\delta]$. From the resulting BDD of the result, the standard linear-time BDD algorithms can be used to find the number of minterms for the discriminant of value L. From this, the leakage histogram for the circuit is computed.

The CUDD [24] package is used for all the ADD operations in this paper. This package has routines to perform the operations described in the algorithms described in this thesis.

2. Approximate Computation of Leakages of all Vectors

The algorithm AL_{ex}^{all} of Section 1 produces the exact leakage values for the circuit being considered. Also, the BDD representation of all minterms with any specific leakage value L can be computed as described in Section 1. From this BDD, the number of input vectors (or a single vector) with leakage L can be computed in linear time. However, in an exact ADD representation of circuit leakage, the number of discriminants can be quite large. As a consequence, it is important to compute the

circuit leakage ADDs in an *approximate* manner. This results in a reduction in the memory utilization and thereby allows the method to to handle larger designs.

The algorithm AL^{all}_{app}, computes the approximate leakage ADD of the circuit. In this approach the discriminant values are discretized during the *add_ADD* operation, such that the total number of discriminants of the added result are bounded by a user-specified constant m. The following subsection elaborates upon the discretization approach.

a. Binning of Leakage ADD Values

Since the library used consists of gates with up to 4 inputs, the maximum number of discriminants for the leakage ADDs of any gate is limited to 16. However, the resulting ADD after the *add_ADD* operation on two ADDs with D_1 and D_2 discriminants respectively), may have as many as $D_1 \cdot D_2$ discriminants. To control the size of the resulting ADD after addition, discretization of the discriminants of the result is performed. The discretization is driven by a user-specified constraint m, which represents the maximum number of discriminants in any ADD constructed (intermediate or final).

Consider the addition of two ADDs $\mathcal{F}$ and $\mathcal{G}$, using the *add_ADD* routine. Let the minimum and maximum discriminant values of $\mathcal{F}$ ($\mathcal{G}$) be $L^{\mathcal{F}}_{min}$ and $L^{\mathcal{F}}_{max}$ ($L^{\mathcal{G}}_{min}$ and $L^{\mathcal{G}}_{max}$) respectively. As a consequence, the minimum and maximum discriminant values of the result will be $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min})$ and $(L^{\mathcal{F}}_{max} + L^{\mathcal{G}}_{max})$ respectively. Let the interval between these two values be R. Next discretize the interval into m values $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min})$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{R}{m-1})$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{2R}{m-1})$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{3R}{m-1})$, $\cdots$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{(m-2)R}{m-1})$, $(L^{\mathcal{F}}_{max} + L^{\mathcal{G}}_{max})$.

Next, during the terminal case computation of Algorithm 2, compute $v = \mathcal{F}_{val} +$

$\mathcal{G}_{val}$ and adjust its value to the nearest of the m discretized discriminant values described in the previous paragraph. Let the adjusted value be v_{adj}. Then, the value returned by Algorithm 2 in the terminal case is v_{adj}.

This limits the total number of discriminants in the result of add_ADD to m, instead of $D_1 \cdot D_2$, resulting in significantly reduced memory utilization in general. Also, the maximum error introduced by a single step of this addition is $\frac{1}{2(m-1)}$, allowing the user to trade off the memory utilization and maximum tolerable error.

b. Extensions to the Approach

In its current form, this algorithm computes the leakage ADDs for upto medium sized circuits. To improve this further, a partitioned [25] construction of leakage ADDs may prove beneficial. In this approach, a k-way min-cut partitioning of the circuit is first performed, and the leakage ADDs of each partition is computed separately (on the space of the local inputs for that partition), before finally computing the image of these ADDs on the space of the primary inputs of the design.

Another application of this approach would be to compute the leakage ADD $\mathcal{G}$ for an arithmetic unit, from the leakage ADD $\mathcal{G}_s$ of a bit-slice of the unit. Suppose that the i^{th} bit slice depends on free variables[1] v_f^i and bound variables[2] v_b^i. Let the leakage ADD of the i^{th} bit slice be $\mathcal{G}_s^i(v_b^i, v_f^i)$[3], and the leakage ADD of the logic driving variables v_b^i be called g_b^i. The leakage ADD $\mathcal{G}$ can be computed by Algorithm 4.

---

[1]Free variables are variables that are primary inputs of $\mathcal{G}$.

[2]Bound variables are variables of $\mathcal{G}_s$ that are the outputs of other bit slices in the design.

[3]$\mathcal{G}_s^i(v_b^i, v_f^i)$ is computed from the leakage ADD of a generic slice ($\mathcal{G}_s$) by a simple variable substitution.

Algorithm 4 Finding $\mathcal{G}$ from $\mathcal{G}_s^i$

$\mathcal{G} \leftarrow \mathcal{G}_s^1$

for $(i = 2; i <= n; i++)$ **do**

$\mathcal{G} \leftarrow \mathcal{G} + BDD\_substitute(\mathcal{G}_s^i, v_b^i, g_b^i)$

end for

return $\mathcal{G}$

In this manner, the total leakage of the arithmetic unit is computed iteratively, using the computed leakage ADD of a single slice. In the i^{th} iteration, each bound variables is substituted in the leakage ADD of the i^{th} slice with the leakage ADD of the driving logic for that variable. The resulting leakage ADD of the slice is then added to the leakage ADD of the entire design. Hence, the computation of the leakage ADD of any slice i includes the constraints imposed by the leakage values for slices j whose outputs are inputs to the slice i.

C. Determining an Input Vector Resulting in the Lowest Leakage

If the requirement is only to determine the minimum leakage vector, and the knowledge of the leakage distribution is not required, constructing the exact or approximate leakage ADD may not be the most relevant approach. Therefore, a heuristic to compute only the minimum leakage vector is presented. This heuristic is called PL^{min}. The brief outline of the methodology for selecting the input vector that minimizes circuit leakage is as follows:

- First, signal probabilities are computed for all nodes in the design, assuming that

all inputs have a signal probability of 0.5[4]. These probabilities are heuristically adjusted for inaccuracies arising from reconvergent fanouts.

- Next, the best candidate gate is selected, whose leakage would be set in a given iteration. This is performed by selecting the gate that is probabilistically most likely to result in the largest leakage reduction.

- For the gate thus selected, its best state is assigned next, such that the leakage of the selected gate is probabilistically minimized. All other gates in the circuit which are newly implied by the state just selected are accounted for while making this decision.

- To test if the logic values that were set to 1 or 0 during this iteration are satisfiable, a Boolean Satisfiability solver is invoked. The SAT solver is called every p iterations to reduce the runtime overhead. If the circuit is unsatisfiable, the assignments of the last p iterations are discarded, and the iteration that caused the circuit to become unsatisfied is determined. After making a different selection for that iteration, the method proceeds as before.

- After any iteration, gate probabilities are adjusted to account for the nodes that were newly assigned fixed logic values.

- A fixed number of passes are made for the circuit, with the above steps being applied successively. Each pass is more "lenient" in setting a node to a logic value v when its signal probability deviates from v. The last pass is most lenient, allowing any deviation from v to be accepted.

[4]In case of sequential circuits, utilize the probabilities of the signals at the outputs of memory elements instead. Also, for combinational designs, if it is known that input signal probabilities are skewed away from 0.5, then these skewed values can be used.

Algorithm 5 describes the pseudocode for the approach (PL^{min}) for computing the minimum leakage vector for a combinational network η.

Algorithm 5 Pseudocode of Minimum Leakage Vector Algorithm: PL^{min}

```
compute_minimum_leakage_vector(η, p){
compute_signal_probabilities(η)
platinumvalues ← Φ
for i = 1; i ≤ k; i++ do
  goldvalues ← Φ
  iteration = 1
  (G = find_best_gate(η)
  if (G is not marked visited) then
    mark G as visited
    (S = find_best_leakage_state(G, η)
    if S satisfies m_i then
      goldvalues ← goldvalues ∪ S ∪ get_implications(S)
      propagate probabilities in TFO of goldvalues nodes
    end if
    if iteration is a multiple of p OR all inputs assigned/implied then
      if goldvalues are satisfiable then
        if all inputs assigned then
          exit
        end if
        platinumvalues ← platinumvalues ∪ goldvalues
      else
        goldvalues ← platinumvalues
      end if
    end if
  end if
  iteration += 1
  mark all gates as unvisited
end for
}
```

1. Computing Signal Probabilities

The algorithm *compute_minimum_leakage_vector*(η, p) begins by computing signal probabilities for all nodes in the network η. The inputs are assumed to have probabil-

ities of 0.5, and these probabilities are propagated throughout the circuit[5]. After the initial pass of probability propagation, a heuristic adjustment is performed to account for errors due to reconvergent fanouts. The heuristic for probability adjustment in the presence of reconvergence is illustrated in Figure 6. The input parameter p is used to determine how frequently a boolean satisfiability solver is invoked.

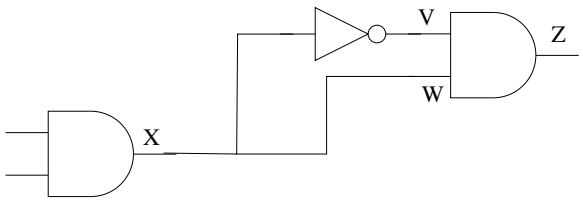


Fig. 6. Adjusting Probabilities for Reconverging Nodes

Suppose a node X, with a statically computed probability of P_X reconverges at Z. Setting the probability of X to 1 and 0, the probabilities of the inputs to the reconvergent gate (V and W) are found. Suppose the probabilities of V (W) are V_1 (W_1) and V_0 (W_0) respectively, when X is set to 1(0). In this case, the new probability of Z is $P_Z^{new} = \frac{V_0 \cdot W_0 + V_1 \cdot W_1}{2}$.

From this the adjustment factor for the probability of Z is computed as follows:

$Adjustment(Z) = \frac{(P_Z^{new} - P_Z)}{P_Z}$

In future updates of the probability of the node Z, suppose the statically computed probability of node Z is $P_Z^{modified}$. In that case, the final adjusted value of the probability of node Z is

$P_Z^{adj} = (P_Z^{modified}) \cdot (1 + Adjustment(Z))$.

In other words, $Adjustment(Z)$ is computed once, and utilized to adjust the

[5]If the input i of an n-input AND gate has probability p_i, then the output has probability $\Pi_i p_i$. Likewise, for an OR gate, the output has probability $1 - \Pi_i(1 - p_i)$. The probabilities of other gates can be found in a similar fashion

statically computed values of the probability of node Z, each time it is modified due to other assignments in the circuit.

In the example of Figure 6, $Adjustment(Z) = -1$. Therefore, $P_Z^{adj} = 0$ each time the probability of Z is modified. This is reasonable, given that the output Z is logically 0.

If an adjusted probability of a node results in its probability becoming higher than a user specified value P_{adj} (lower than $1 - P_{adj}$), then the probability of the node is capped at P_{adj} $(1 - P_{adj})$ respectively.

2. Finding the Best Leakage Candidate

Once signal probabilities are computed, the best candidate gate whose input state is to be finalized during the current iteration is selected. Gates are ranked by the probabilistic criterion described below:

$C = \frac{\sum(p_i \cdot l_i)}{\sum(p_i)}(l_i^{max} - l_i^{min})$

Here, p_i is the probability that the gate is in state i. Here "state", means a complete assignment of the inputs (and outputs) of the gate. The quantity l_i is the leakage of the state i. The value l_i^{max} (l_i^{min}) is the maximum (minimum) leakage value of this gate. The gate with the maximum value of C is selected. In other words, this criterion selects gates that have a high probability of being in a high-leakage state. The last term in the expression for C ensures that gates with large leakage ranges are favored, since they offer potentially greater optimization flexibility. The gate that maximizes C is selected preferentially over others.

3. Finding the Best Leakage State for a Selected Gate

Suppose a gate G was selected by the previous step. Next it is assigned it a state such that its leakage is minimized. This is done by applying the probabilistic criterion L below. Note that all gates other than G whose states become fully assigned[6] on account by implication propagation of the current state of G, are included in the computation of L. Let the number of such gates be n. The value of probabilistic leakage in the numerator of L is normalized with respect to the number of such gates, and is computed as follows:

$$L = \frac{\sum_j (d_j \cdot l_j)}{n}$$

The state of gate G that minimizes L is preferentially selected over others. Here d_j is the deviation of the values assigned to the gate inputs from their probabilistic values. For example, consider an AND gate with inputs a and b with probabilities 0.1 and 0.7 respectively. If inputs a and b are logic 1 and logic 0 respectively, then the deviation is $(|1-0.1|)(|0.7-0|)$. In other words, while attempting to select a state for gate G, we try to minimize the resulting leakage and also maximize the probabilistic 'ease' with which this state can be assigned.

In order to bias the state selection towards assignments with lower leakage, the deviation is incremented by a value β. Likewise, in order to bias the state selection towards those with lower deviation, l_j is incremented by a fixed value γ. Therefore, the modified value of L that is used is

$$L = \frac{\sum_j (d_j+\beta) \cdot (l_j+\gamma)}{n}$$

[6] A gate is said to be *fully assigned* if all its inputs are assigned to specific logic values

4. Accepting Leakage States and Endgame

The state selected from the previous section is now implied throughout the circuit. The resulting implied values are referred to as golden values. The deviations of the resulting implications are now checked against a margin value m_i. If any deviation is greater than m_i, then the assignment to gate G is discarded. Initially, m_i is set to a small value, and with increasing iteration i, it is relaxed. This is in an attempt to get closer to a global minima, by a more careful selection of states in early iterations. The number of iterations is the variable k, which in the experiments is set to 3.

Once the new implications are computed, the implied nodes' probabilities are adjusted to reflect the freshly computed implications. If a node is set to a logic 1, then its probability is set to (1-α), while a node which is set to logic 0 has its probability updated to α. Here, α is a user-specified constant close to 0.

For every p gates selected (or if all primary inputs have been assigned or implied), the golden values are tested for satisfiability (this test is done by invoking the BerkMin [26] satisfiability solver). If these golden values are consistent, then they are designated as new platinum values, never to be modified in the future. If the golden values are satisfiable, and all inputs are assigned, then the algorithm exits. If the golden values are not satisfiable satisfied, then the golden values are rolled back and discarded, by copying the last set of platinum values into the set of golden values. For up to the next p iterations, the satisfiability solver is called after *each* new state assignment. This is in an attempt to locate which of the last p assignments caused the unsatisfiability condition to occur. Once this assignment has been identified, the invocation of the satisfiability solver is again performed after every p state assignments. If the satisfiability solver returns an unsatisfiable condition for a certain state s assigned at a particular gate g, then s is never assigned to g again.

D. Chapter Summary

This chapter described the algorithms used for computing the exact and approximate leakage values for all input vectors for a circuit and also a probabilistic heuristic to determine the input vector which drives the circuit to its lowest leakage state (or to any required leakage state.) The intuition behind these algorithms was explained, along with an exposition of the details. In addition, some extensions for future work were discussed. The pseudocode was provided for a peruse explanation of the algorithms. The experimental results for both the approaches and the conclusions drawn from them are discussed in the following chapters.

CHAPTER IV

EXPERIMENTAL RESULTS

A. Chapter Overview

This chapter presents the experimental results for the algorithms described in the previous chapter, namely AL_{app}^{all} and PL^{min}. Section B discusses results obtained for the approximate leakage ADD computed with varying numbers of discriminants. In addition, two different implementations of few design were compared with respect to their leakage histograms. Section C compares the leakage values obtained by the minimum leakage vector algorithm (PL^{min}) with the exact minimum leakage for small designs, and with the approximate minimum leakage (generated after 10,000 random simulations) for large designs. These designs were such that the computation of the exact minimum leakage was not feasible. The chapter is concluded in Section D.

B. Results for AL_{app}^{all}

The technique AL_{app}^{all} was applied on a series of MCNC91 benchmark designs, using a 0.1μm technology library with 13 gates, with between 1 and 4 inputs. After running technology independent logic optimizations (*script_rugged* in SIS [27]), these designs were mapped for area and delay (again in SIS).

The AL_{ex}^{all} and AL_{app}^{all} leakage computation techniques were implemented in SIS, and implemented using the CUDD [24] package. Applying the approximate technique AL_{app}^{all} with discretized discriminants, enabled the computation of leakage ADDs for larger designs.

Tables B and B describe the maximum and minimum leakages (in pA) of four

designs, as a function of the value of m (the number of discretized discriminants used during ADD construction). Each design was mapped for minimum area as well as minimum delay. The row labeled "exact" represents the leakages with no discretization of leakage values (effectively $m = \infty$). Note that a good choice of the values of m is between 12 and 16 for most cases.

Table II. Accuracy Versus Bin Size I

	9symml				cc			
	Delay map		Area map		Delay map		Area map	
	min	max	min	max	min	max	min	max
exact	622.9	734.6	474.1	611.8	193.2	272.5	127.2	227
20 bins	540.8	772.3	429.1	633.2	209.6	267.8	131.5	221.1
16 bins	396.7	955.8	402.9	600.8	197.2	261.5	122	209.8
12 bins	285	1064.5	284	821.6	197.5	270.4	117.3	253.5
8 bins	212.4	1206.6	199.3	964.4	91	360.1	76.4	278
4 bins	212.4	1206.6	199.3	964.4	91	360.1	76.4	278

Figure 7 describes the range of leakage values for the minterms mapped to the lowest discriminant of the ADD, compared against the normalized value of the range of the exact leakage. Ideally, this should be a point, with leakage L_{min}. It was observed that for most designs, this range is small, indicating that the method is accurate. The approximate experiments for this figure were performed with $m = 20$.

Tables B and B indicate the maximum and minimum leakage (represented in 10's of pA) for several designs, mapped both for minimum area as well as delay. It

Table III. Accuracy Versus Bin Size II

	decod				alu2			
	Delay map		Area map		Delay map		Area map	
	min	max	min	max	min	max	min	max
exact	187.8	238.6	30.6	79.9	1241.9	1382.9	872.8	1060.7
20 bins	200.8	239.1	31	83	905.5	1771.4	645.1	1348.5
16 bins	208	241.9	27.6	90.8	700.5	2005.2	576	1563.3
12 bins	212.6	235.5	23.6	74.9	536.7	2193.2	484.8	1753.4
8 bins	89.3	314.5	33	92.3	511.9	2251.2	382	1856.5
4 bins	89.3	314.5	33	92.3	511.9	2251.2	382	1856.5

was observed that mapping for minimum area results on average in a 20% reduction in both the maximum and minimum leakage value, compared to delay mapping. The experiments in these tables were performed with m=12.

The leakage histograms associated with the leakage ADDs were computed for some designs. For this experiment, m=20 was used. The comparison between the area mapped and delay mapped histograms suggests that the area-mapped histograms are typically "better", with a larger number of minterms which have smaller leakage values. Figure 8 illustrates the results of this experiment.

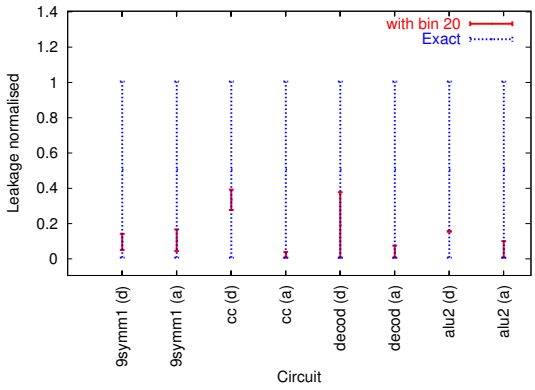


Fig. 7. Error of ADD Based Leakage Computation

C. Minimum Leakage Vector Determination Using PL^{min}

Extensive experiments were conducted to validate the probabilisitic leakage minimizing heuristic. Results were compared with the exact minimum circuit leakage values. When it was not possible to find the exact minimum circuit leakage values, the minimum leakage value over a large number of input vector samples was found. In all experiments, the value of k, (i.e. the number of iterations) was 3. The 3 sets of parameter sets (M1, M2 and M3) that were utilized for the experiments are described in Table C. These are referred to as *methods* in the sequel. The value of p used was 1, but it can be increased for less accurate but faster invocations of the algorithm. The parameter values reported in Table C were found after extensive experimentation with many circuits.

Methods M1 and M2 utilize a value of m_1 of 0.6. As a consequence, these methods can be expected to set more gates to platinum values in the first iteration. These methods are designed to reduce the number of gates discarded due to margin violations. Among these methods, M1 has a higher γ value, and therefore biases the state selection towards states which have smaller deviations. On the other hand, M2

has a higher β value, and as a consequence, state selection favors states with lower leakage. Method M3 has a smaller m_1 value, and therefore tends to reject gates due to margin violations. It is biased towards state selections which have smaller deviations.

For these three methods, the minimum leakage values were compared with those obtained by an exhaustive evaluation of leakages. This was performed for small examples, and results are reported in Table C. The minimum leakage value returned by the PL^{min} method (Column 4), along with the exact maximum (Column 2) and minimum (Column 3) leakages are shown in this table. Further, a figure of merit R is reported in Column 5.

$$R = \frac{PL^{min}\ min\ leakage - Exact\ min\ leakage}{Exact\ max\ leakage - Exact\ min\ leakage}$$

The values of the maximum and minimum range of leakages are computed based on an exhaustive simulation of the circuit. Ideally, R should be 0. Runtimes for the PL^{min} method are reported in Column 7, while the method utilized is reported in Column 6.

Note that the figure of merit R is a more rigorous metric for comparing the effectiveness of any MLV determination technique. In the prior approaches to the MLV determination problem, the figure of merit utilized was

$$R_{old} = \frac{Heuristic\ min\ leakage - Exact\ min\ leakage}{Exact\ min\ leakage}$$

The R metric is a more rigorous method for evaluating the results of an MLV technique compared to the R_{old} metric. This is because R_{old} may give optimistic results when the minimum leakage value is high, and comparable to the maximum leakage value. Such behavior is exhibited by larger circuits. The R metric avoids this artificial optimism.

From Table C, the average value of R for PL^{min} method was about 0.125. For the PL^{min} method, the average value of the previously utilized figure of merit R_{old} is about 0.053.

Table C shows that runtimes for the PL^{min} method are very small, with a good figure of merit for the method. Given that the run-times are very small, *we can afford to apply all three methods (M1, M2 and M3), and choose the best result among the three. In general, we may try several methods and select one that yields the vector with the smallest leakage.*

The algorithm PL^{min} was tested on larger circuits as well. The results of this experiment are shown in Tables C and C. The columns in these tables are as in Table C, with the exception that exact leakage values could not be computed for the examples in this table. Instead, the minimum and maximum leakage values (found over 10,000 random vectors) are reported in Tables C and C. According to [13], this results in a greater than 99% confidence that the resulting leakage is within 0.5% from the minimum leakage value.

Tables C and C show that PL^{min} method produces MLVs with very low errors, with small runtimes. For the previously reported methods of [9], [28] and [15], the errors were respectively 5.3%, 3.7% and 10.4% (using the R_{old}[1] metric, for which PL^{min} results in an error of 3.7%). Further, the runtimes for PL^{min} are significantly smaller than those of [28].

D. Chapter Summary

This chapter presents the experimental results for the algorithms described in this thesis, namely ALL^{all}, AL^{all}_{Ab} and PL^{min}. Results obtained for the approximate leakage ADD (computed with varying number of discriminants) are compared with exact values. In addition, two different implementations, mapped for *area* and *delay*, for some

[1]Previous papers have reported results using the R_{old} metric alone.

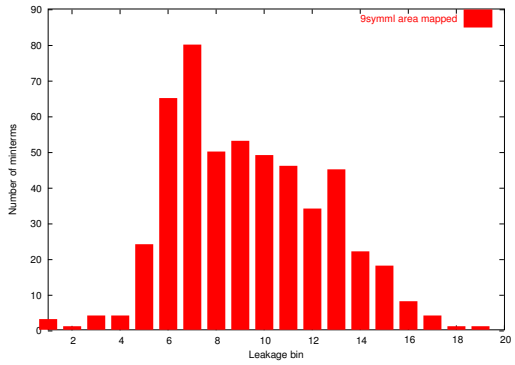
design are compared. The comparison is made on the different leakage histograms obtained for the above two common mapping criteria. In Section C, the leakage values obtained by the minimum leakage vector determination algorithm PL^{min}, are compared against the exact minimum leakage values obtained for small designs and against the approximate minimum leakage values generated after 10,000 random simulations for large designs. From Table C, for most designs the ratio of maximum leakage versus minimum leakage found from experimental results averaged around 2.6. The conclusions based on the experimental data obtained are discussed in the following chapter.

Table IV. Leakage Min/max Values for Area and Delay Mapped Designs I

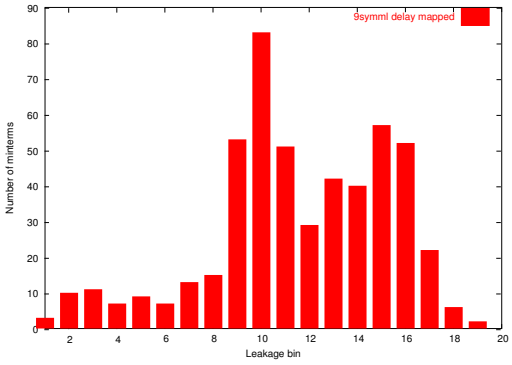
	Delay mapped		Area mapped	
	max	min	max	min
9symml	10645.4	2850.7	8216.3	2840.0
b9	6385.3	1385.7	5573.1	1333.5
c8	6542.8	2564.5	6572.5	2337.5
cc	2704.7	1975.5	2535.9	1173.6
cht	9589.2	3248.4	9077.8	3100.9
cm138a	1179.0	885.5	618.4	291.6
cm150a	2332.8	1078.2	2109.1	956.3
cm151a	1153.0	653.5	1153.0	653.5
cm152a	974.6	613.2	974.6	613.2
cm162a	2167.7	1213.4	2131.3	958.1
cm163a	1976.9	1189.4	2218.4	1067.8
cm42a	993.4	777.0	672.0	417.9
cm82a	1062.0	855.8	929.8	712.6
cm85a	2147.4	1245.9	1658.8	1084.7
count	7740.8	2354.9	6427.2	892.0
cu	2316.0	1328.5	1912.7	1091.8
f51m	3331.7	2562.7	3224.4	2255.6
frg1	7814.1	1723.1	7515.6	1298.4

Table V. Leakage Min/max Values for Area and Delay Mapped Designs II

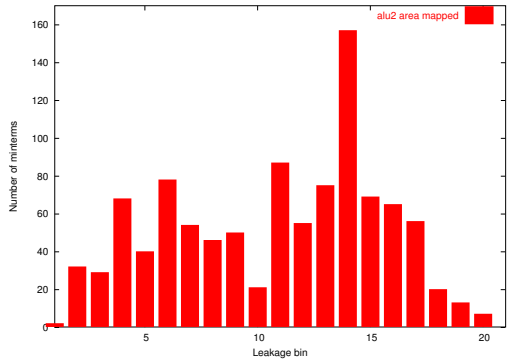
	Delay mapped		Area mapped	
	max	min	max	min
i1	2453.0	785.8	1950.9	558.3
lal	5406.9	1400.9	4584.6	1004.8
majority	429.8	269.1	350.7	192.1
mux	2541.8	1672.0	2064.6	1088.3
parity	3031.0	1884.9	3031.0	1884.9
pcle	3982.1	1453.9	3578.4	1397.3
pcler8	5485.5	1527.7	4849.5	1352.9
pm1	2043.5	856.1	1763.3	504.1
sct	3730.8	1729.9	3136.4	1618.2
t	321.8	179.7	321.8	179.7
tcon	1465.8	1052.5	1070.2	656.9
unreg	5199.0	2893.4	5083.3	1966.5
x2	1557.5	704.9	1340.0	587.2
z4ml	1715.6	1389.2	1482.5	1051.8
decod	2355.1	2126.8	749.7	236.9
alu2	21932.5	5367.9	17534.8	4848.5
alu4	43888.3	10457.2	33218.0	7870.9
t481	48647.5	9664.6	38554.8	5936.5
vda	34696.6	11041.7	25198.8	7223.9
apex7	14949.1	3320.9	12413.3	1802.8
AVERAGE	7286.6	2323.3	5942.1	1711.7



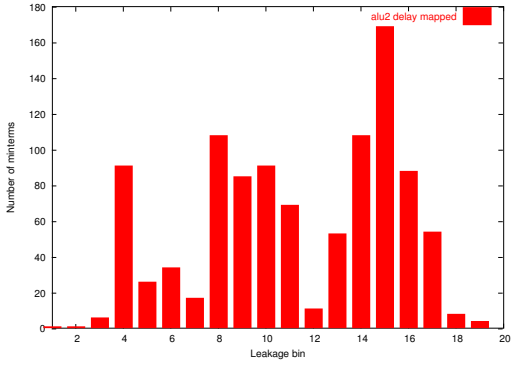
(a) 9symml-a



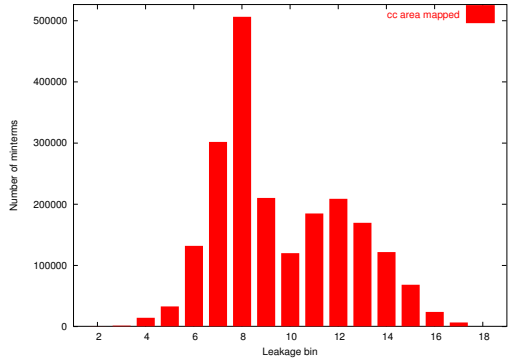
(b) 9symml-d



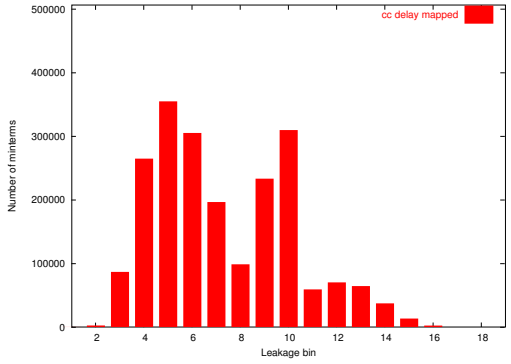
(c) alu2-a



(d) alu2-d



(e) cc-a



(f) cc-d

Fig. 8. Leakage Histograms for Delay and Area Mapped Circuits

Table VI. Parameters Used in the Experiments

Method	m_1	m_2	m_3	β	γ	P_{adj}	α
M1	0.6	0.96	1	0.5	50	0.95	0.95
M2	0.6	0.96	1	5	10	0.95	0.95
M3	0.4	0.96	1	0.1	100	0.9	0.9

Table VII. Exhaustive and Estimated Leakages for Small Circuits

Circuit	Low	High	PL^{min} Low	R	R_{old}	Meth.	Time (s)
C17	12.47	30.15	12.47	0.000	0.000	M2	0
cm138a	70.80	109.86	71.63	0.021	0.012	M1	0.01
cm151a	129.55	167.57	135.75	0.163	0.048	M2	0.02
cm152a	65.95	99.07	75.54	0.290	0.145	M3	0.01
cm42a	90.19	111.23	90.73	0.026	0.006	M1	0
cm82a	87.22	101.16	91.37	0.298	0.048	M2	0.02
cm85a	142.25	217.08	162.08	0.265	0.139	M1	0.06
decod	185.97	237.43	185.97	0.000	0.000	M3	0.08
majority	32.41	44.55	32.41	0.000	0.000	M2	0.01
t	17.48	33.72	20.7	0.198	0.184	M2	0
AVG				0.1261	0.053		

Table VIII. Leakages for Large Circuits I

Circuit	Low	High	PL^{min} Low	R	R_{old}	Meth.	Time (s)
apex6	2738.57	3085.62	2609.550	-0.372	-0.047	M3	81.86
b9	328.16	554.04	362.11	0.150	0.103	M2	0.47
C1908	2353.45	2833.71	2473.570	0.250	0.051	M2	42.73
C2670	3430.12	3880.88	3459.29	0.065	0.009	M2	141.31
C3540	5268.64	5918.25	5402.009	0.205	0.025	M1	296.19
C432	841.13	1074.19	893.27	0.224	0.062	M2	1.78
C499	1748.97	1942.81	1775.1	0.135	0.015	M2	20.18
c8	507.41	829.42	552.24	0.139	0.088	M3	1.46
cc	190.44	402.85	203.87	0.063	0.071	M1	0.12
cht	735.31	1066.45	777.480	0.127	0.057	M3	3.39
cm150a	257.87	324.84	264.81	0.104	0.027	M3	0.21
cm162a	157.88	228.34	173.2	0.217	0.097	M1	0.04
cm163a	151.72	213.44	165.62	0.225	0.092	M1	0.05
cmb	119.87	203.63	130.61	0.128	0.090	M1	0.03
comp	523.47	691.71	563.31	0.237	0.076	M3	1.03
count	483.50	624.68	488.97	0.039	0.011	M3	0.81
cu	159.71	276.39	184.16	0.210	0.153	M3	0.14
example2	1079.71	1396.58	998.44	-0.256	-0.075	M3	9.49

Table IX. Leakages for Large Circuits II

Circuit	Low	High	PL^{min} Low	R	R_{old}	Meth.	Time (s)
frg1	362.57	499.46	377.66	0.110	0.042	M3	0.6
i1	124.17	234.66	127.47	0.030	0.027	M1	0.05
i3	705.68	823.34	672.36	-0.283	-0.047	M1	2.94
i4	624.79	951.79	539.9	-0.260	-0.136	M3	4.29
i5	1090.40	1417.71	909.120	-0.554	-0.166	M2	9.44
lal	357.52	687.27	389.02	0.096	0.088	M3	0.94
mux	287.66	400.43	304.31	0.148	0.058	M1	0.22
parity	213.38	271.10	221.56	0.142	0.038	M2	0.08
pcle	219.48	316.58	229.22	0.100	0.044	M1	0.07
pcler8	307.10	397.61	310.9	0.042	0.012	M3	0.18
pm1	92.63	285.76	95.66	0.016	0.033	M3	0.07
rot	2294.48	2658.33	2321.8	0.075	0.012	M3	63.35
tcon	139.77	190.54	139.77	0.000	0.000	M1	0.04
ttt2	857.95	1222.36	919.51	0.169	0.072	M2	3.43
unreg	473.89	615.77	497.68	0.168	0.050	M1	0.51
x1	1156.81	1708.28	1273.85	0.212	0.101	M1	10.65
x3	3219.92	4274.29	3528.949	0.293	0.096	M1	167.64
x4	1690.86	2471.53	1919.870	0.293	0.135	M3	30.27
AVG				0.0746	0.037		

Table X. Ratio of Maximum to Minimum Leakage (Delay Mapped)

Circuit	No. Of Gates	Ratio
i1	59	3.1
lal	165	3.86
majority	12	1.6
mux	103	1.52
parity	75	1.61
pcle	83	2.74
pcler8	107	3.59
pm1	56	2.39
sct	128	2.16
t	7	1.79
tcon	49	1.39
unreg	148	1.8
x2	56	2.21
z4ml	67	1.23
decod	76	1.11
alu2	532	4.09
alu4	1053	4.2
t481	5612	5.03
vda	1111	3.14
apex7	313	4.5
AVG		2.653

CHAPTER V

CONCLUSIONS

In recent times, heuristic as well as exact approaches have been developed to compute the input vector which minimizes the leakage of a circuit. In this thesis, an approach to compute an ADD based implicit approach to find the leakage of all vectors in a circuit has been described. The knowledge of the leakage of a circuit over all vectors can be used in several ways, one of which is to select between competing implementations of a circuit.

The algorithm AL_{ex}^{all} computes the leakage ADDs of each circuit node, and then adds these ADDs to compute the leakage ADD of the circuit in terms of its primary inputs. Also, an approximate version of this algorithm (AL_{app}^{all}), is implemented, which discretizes the number of discriminants of an ADD to a user-specified limit m.

It is experimentally demonstrated that these approximate techniques produce results which have reasonable errors. It is shown that limiting the number of discriminant nodes to a value between 12 and 16 is practical, allowing for good accuracy and lowered memory utilization. Also, area-mapped designs typically have better leakage characteristics than their delay-mapped counterparts. Viewed more generally, it is important to note that for large designs the ratio of maximum leakage versus minimum leakage found from experimental results averaged at 2.5. AL_{app}^{all} readily generates this information even for large designs.

Additionally in this thesis, a probabilistic method to perform input vector assignment for leakage minimization in a combinational circuit is presented. The corresponding algorithm (PL^{min}), begins by computing signal probabilities throughout the circuit. These probabilities are used to guide the selection of the next gate to which fixed values should be assigned. The selected gate is the one with the proba-

bilistic highest leakage value. Once this gate is selected, it is assigned a state, again in a manner which probabilistically minimizes its leakage. The implications induced by such a state selection are computed. A satisfiability solver is invoked, to validate the state selection before our algorithm commits to this assignment. The algorithm terminates when all inputs have been assigned or are implied.

The method is fast, flexible and provides accurate results. On average, for small examples, the PL^{min} method found minimum leakage values which were 5.3% from the minimum circuit leakage. For larger examples, it was impractical to compute the minimum circuit leakage exactly. The statistics for purposes of comparison were computed on the basis of running 10,000 random vectors. For these examples, the PL^{min} method produces MLVs with leakage values within 3.7% from the minimum. The runtimes of the PL^{min} method are much lower than existing techniques which produce results of similar quality. Some of the future work includes modifying the algorithms for a multi-V_T circuit by targeting the high V_T gates earlier, modifying the algorithm where the leakage of a gate for a particular input is not a fixed value (is instead a range), and determining the gates whose change in leakage is not observable in any of the consistent assignments. Such modifications might lead to better runtimes and higher applicability.

REFERENCES

[1] "BSIM3 Homepage," Accessed April 2006, [Online]. Available: http://eecs.berkeley.edu/∼bsim3.html.

[2] "The International Technology Roadmap for Semiconductors," Accessed April 2006, [Online]. Available: http://public.itrs.net/home.htm.

[3] J T Kao and A P Chandrakasan, "Dual-threshold voltage techniques for low-power digital circuits," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 7, pp. 1009–1018, Jul 2000.

[4] S Mutoh, T Douseki, Y Matsuya, T Aoki, S Shigematsu, and J Yamada, "1-v power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 8, pp. 847–854, Aug 1995.

[5] N Jayakumar and S Khatri, "An ASIC design methodology with predictably low leakage, using leakage-immune standard cells," in *Proc., International Symposium on Low Power Electronics and Design*, Aug 2003, pp. 128–133.

[6] H Kawaguchi, K Nose, and T Sakurai, "A super cut-off CMOS (SCCMOS) scheme for 0.5-v supply voltage with picoampere stand-by current," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 10, pp. 1498–1501, Oct 2000.

[7] F Assaderaghi, D Sinitsky, S A Parke, J Bokor, P K Ko, and C Hu, "Dynamic threshold-voltage MOSFET (DTMOS) for ultra-low voltage VLSI," *IEEE Transactions on Electron Devices*, vol. 44, no. 3, pp. 414–422, Mar 1997.

[8] Y Cao, T Sato, D Sylvester, M Orshansky, and C Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc., IEEE Custom Integrated Circuit Conference*, Jun 2000, pp. 201–204.

[9] F Gao and J Hayes, "Exact and heuristic approaches to input vector control for leakage power reduction," in *Proc., International Conference on Computer-aided Design*, Nov 2004, pp. 527–532.

[10] A. Ferre and J. Figueras, "Characterization of leakage power in CMOS technologies," in *Proc., IEEE International Conference on Electronics Circuits and Systems*, Aug 1998, pp. 85–188.

[11] Z Chen, M Johnson, L Wei, and W Roy, "Estimation of standby leakage power in CMOS circuit considering accurate modeling of transistor stacks," in *International Symposium on Low Power Electronics and Design*, 1998, pp. 239–244.

[12] D. Duarte, Y. Tsai, N. Vijaykrishnan, and M. J. Irwin, "Evaluating run-time techniques for leakage power reduction," in *7th ASPDAC/15th International Conference on VLSI Design*, Jan 2002.

[13] J Halter and F Najm, "A gate-level leakage power reduction method for ultra low power cmos circuits," in *Proc., CICC*, 1997, pp. 475–478.

[14] M Johnson, D Somasekhar, and K Roy, "Models and algorithms for bounds on leakage in CMOS circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 714–725, June 1999.

[15] R Rao, F Liu, J Burns, and R Brown, "A heuristic to determine low leakage sleep state vectors for CMOS combinational circuits," in *Proc., International Conference on Computer-aided Design*, Nov 2003, pp. 689–692.

[16] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, New York: Computer Science Press, 1990.

[17] T. A. Unni and D. M. H. Walker, "Model-based i_{DDQ} pass/fail limit setting," in *IEEE International Workshop on I_{dqq} Testing*, 1998, pp. 43–47.

[18] F Aloul, S Hassoun, K Sakallah, and D Blauuw, "Robust SAT-based search algorithm for leakage power reduction," in *Proc., Power and Timing Models and Simulation (PATMOS)*, 2002.

[19] A. Abdollahi, F. Fallah, and M. Pedram, "Runtime mechanisms for leakage current reduction in CMOS VLSI circuits," in *Proc., Symposium on Low Power Electronics and Design*, Aug 2002, pp. 213–218.

[20] H Y Song, S Bohidar, R I Bahar, and J Grodstein, "Symbolic failure analysis of custom circuits due to excessive leakage current," in *Proc. of the Intl. Conf. on Computer Design*, 2003, pp. 70–75.

[21] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *Formal Methods in Systems Design*, vol. 10, no. 2/3, pp. 171–206, 1997.

[22] E. M. Clarke, K. L. McMillan, X Zhao, M. Fujita, and J. Yang, "Spectral transforms for large boolean functions with applications to technology mapping," in *Proc., 30th International Conference on Design Automation*. 1993, pp. 54–60, ACM Press.

[23] K Chopra and S Vrudhula, "Implicit pseudo Boolean enumeration algorithms for input vector control," in *Proc., Design Automation Conference*, San Diego, June 2004, pp. 767–772.

[24] "CUDD: CU decision diagram package," [Online]. Available: http://vlsi.colorado.edu/ fabio/CUDD/cudd.html.

[25] A Narayan, J Jain, M Fujita, and A Sangiovanni-Vincetelli, "Partitioned ROBDDs-a compact, canonical and efficiently manipulable representation for Boolean functions," in *Proc., IEEE/ACM International Conference on Computer-Aided Design*, Nov 1996, pp. 547–554.

[26] E Goldberg and Y Novikov, "BerkMin: A fast and robust SAT-solve," in *Proc., Design Automation and Test in Europe (DATE) Conference*, 2002, pp. 142–149.

[27] 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 Laboratory, Univ. of California, Berkeley, May 1992.

[28] S Naidu and E Jacobs, "Minimizing stand-by leakage power in static CMOS circuits," in *Proc., Design Automation and Test in Europe (DATE) Conference*, March 2001, pp. 370–376.

VITA

Kanupriya Gulati was born in Jullundhar, India in May, 1982. She received her bachelors degree in computer engineering, with honors, from Delhi College of Engineering, University of Delhi, India. Her areas of interest includes logic synthesis techniques for leakage computation and toggle equivalent circuits, hierarchical don't care computation and BDD-based synthesis of structured ASICs. After her master's degree she will continue doctoral studies in computer engineering at Texas A&M University.

She can be reached at the Electrical and Computer Engineering Department, Texas A&M University, College Station, TX 77843.

The typist for this thesis was Kanupriya Gulati.