

THESIS
3
2000



This is to certify that the
dissertation entitled

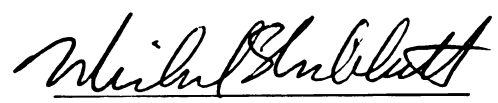
A METHODOLOGY FOR BEHAVIORAL-LEVEL SWITCHING
ACTIVITY ESTIMATION IN CMOS CIRCUITS

presented by

Ronnie Lee Wright

has been accepted towards fulfillment
of the requirements for

Ph.D degree in Electrical Eng


Major professor

Date Sept. 20, 1999

**A Methodology For Behavioral-Level Switching
Activity Estimation in CMOS Circuits**

By

Ronnie Lee Wright

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering

1999

A

The
analysis
major
gence
sign in
schema
based
Such C
a proc
A
a circ
capacit
power
which

ABSTRACT

A Methodology For Behavioral-Level Switching Activity Estimation in CMOS Circuits

By

Ronnie Lee Wright

The demand for computer-aided design (CAD) tools to accurately perform power analysis on high-level design specifications has become increasingly important. Two major factors are responsible for promoting this research and development: the emergence of low power as a key VLSI design parameter and the integrated circuit design industry's increased migration from hardware-design methodologies based on schematics or simple programmable logic device (PLD) languages to methodologies based on high-level design specifications described by hardware description languages. Such CAD tools will enable the development of power efficient digital circuits within a process that offers increased design flexibility, design reuse, and lower cost.

A CAD tool's ability to accurately compute the internal power dissipated by a circuit heavily depends on how well it estimates switching activity and network capacitance. The switching activity is a key factor used in the calculation of dynamic power dissipation for CMOS circuits; it represents the probability or frequency at which power-consuming gate output transitions take place.

The objective of this research is to develop a more accurate and cost-effective technique for computing the switching activity and dynamic power dissipation of behavioral-level design specifications described in VHDL (VHSIC Hardware Description Language). Unlike some gate- or circuit- level *statistical* or *probabilistic* estimation schemes, this new technique operates at the behavioral level of design abstraction without considering technology or statistical circuit characterizations. This new approach accepts a user-specified depth-accuracy parameter that is responsible for controlling the accuracy of the switching activity estimate at the expense of time and memory. The developed techniques and algorithms have been implemented in a program called the Behavioral-Level Activity and Power Estimator (BLAPE). Results and benchmark comparisons with other power analysis CAD tools are given to validate the application and effectiveness of the new approach.

Copyright © by
Ronnie Lee Wright
1999

To my grandmother, Mrs. Lillie Mae Heller

m

p

th

ing

this

Mr.

fact

push

of un

I wou

as an

tion a

Marie

remind

thank

ACKNOWLEDGMENTS

First, giving honor to God, who is the head of my life. If it were not for God and my savior, Jesus Christ, the completion of this research effort would not have been possible. I thank God for giving me the ability and strength to persist and weather the storm.

I would like to thank my hero and grandmother, Mrs. Lillie Mae Heller for inspiring me to become an engineer. She was brave and fought in the struggles that made this and many other opportunities possible. I would like to thank my grandfather, Mr. Willard Ervin Heller (Expired: May 18, 1995) for being a good provider, a great father figure and a very positive influence in my life. As a young man my grandfather pushed for excellence in all of my schoolwork; his insistence initiated the possibility of undergraduate studies and ensured the completion of advanced graduate studies. I would like to thank my uncle and father figure, Dr. Oscar Lee Wright, for serving as an excellent male role model and demonstrating that hard work and determination are the key elements to success. I would like to thank my mother, Ms. Burnice Marie Wright for passing on the ability to be academically successful and constantly reminding me that no problem was too difficult for me to solve. I would like to thank my youngest aunt, Mrs. Marneta Lynn Griffin, for her support in xeroxing

o

h

st

M

in

oc

at

to

ete

I

Br

and

cri

tha

rese

documents/papers, as well as occasional proof-reading of my writings. I would like to thank all my friends and relatives that placed me in their prayers and offered their support. I would even like to thank those who encouraged me to quit when times were difficult. I would like to thank Dr. Christophe Fiorio of the Technische Universität in Berlin, Germany for his development and support of the `algorithm2e.sty` L^AT_EX style file. I would like to give a special thanks to my major dissertation advisor, Dr. Michael A. Shanblatt. Dr. Shanblatt demonstrated excellence in advising and assisted in helping me improve my research skills. He believed in me and the research I was conducting. Also, Dr. Shanblatt was always very professional, timely, and an expert at obtaining funding for my graduate studies and research. Additionally, I would like to thank Dr. Shanblatt's family for being understanding; there were many weekends, evenings, and nights in which I telephoned Dr. Shanblatt at home to discuss research.

I would like to thank my dissertation committee: Dr. Michael A. Shanblatt, Dr. Bruce Kim, Dr. Diane T. Rover, Dr. Anthony S. Wojcik, Dr. Richard J. Enbody, and Dr. Byron C. Drachman for serving on the committee and providing constructive criticisms to improve the quality of my research. I would also like to give a special thanks to Dr. Wojcik for providing papers and other references to assist in the research.

TABLE OF CONTENTS

LIST OF TABLES	xii
LIST OF FIGURES	xiii
1 Introduction	1
1.1 Background	2
1.1.1 Electronic Design Automation Process	2
1.1.2 Design Representations	2
1.1.3 Design Optimization	3
1.1.4 Design Phases	3
1.1.5 Steps Within the Design Phase	5
1.2 Motivation	6
1.3 Low Power Design	7
1.3.1 Sources of Power Dissipation	7
1.3.2 Low Power Optimization Techniques	11
1.3.2.1 Supply Voltage Reduction	11
1.3.2.2 Physical Capacitance Reduction	12
1.3.2.3 Switching Activity Reduction	13
1.3.2.4 Structural Optimization	14
1.3.2.5 System and Architectural Level Optimizations	16
1.4 HDL-Based Design	17
1.5 Problem Statement	18
1.6 Dissertation Overview	19
2 Switching Activity Estimation	21
2.1 Calculation of Switching Activity	21
2.1.1 Input Pattern Dependence	22
2.1.2 Glitch Activity	24
2.1.3 Delay Model	25
2.1.4 Logic Function	26

3

4

5

2.1.5	Circuit Structure	27
2.2	Previous Work	29
2.2.1	Statistical Methods	30
2.2.2	Probabilistic Methods	34
2.2.3	High-Level Methods	42
3	Behavioral Representations of Switching Functions	46
3.1	Switching Algebra Background	46
3.2	Truth Tables	47
3.2.1	Logic Operations	47
3.2.2	Output Enumeration	49
3.3	Boolean Expressions	49
3.4	Binary Decision Diagrams	51
3.5	Behavioral VHDL Specifications	54
4	Structural Representations of Switching Functions	56
4.1	Schematics	57
4.2	Netlists	58
4.3	Structural VHDL Specifications	59
4.4	Connective Binary Decision Diagrams	60
4.4.1	Motivation	61
4.4.2	Overview	61
4.4.3	Minimized Scalable BDDs	62
4.4.3.1	MSBDD Generation Routines	63
4.4.4	CBDD Definitions	65
4.4.5	Analysis of Connective-BDDs	67
4.4.6	Implementation	70
4.4.7	Advantages and Disadvantages	72
4.4.8	Results	73
4.4.9	Summary	74
5	Behavioral-Level Switching Activity Estimation	76
5.1	Methodology Overview	77
5.1.1	Methodology Assumptions	77
5.1.2	Methodology Outline	78
5.2	Methodology Task Decomposition	78
5.2.1	Transformation of VHDL into Boolean Equations	80
5.2.2	Decomposition of Boolean Equations	84

6

7

AP

A

B

5.2.2.1	Implicit Structure Representation	84
5.2.2.2	Mapped Structure Representation	86
5.2.3	Building the CBDD Representation	88
5.2.3.1	CBDD Selection Rationale	88
5.2.3.2	Generation of CBDD Graph	88
5.3	Signal Probability Computation	92
5.3.1	Overview	93
5.3.1.1	Network Levelization	94
5.3.1.2	Disjoint Post-Order Boolean Equation Generation	98
5.3.1.3	IPR Cubeset Generation	104
5.3.1.4	Computation of Signal Probability	109
5.3.1.5	SPCA Component Interconnection	111
5.4	Switching Activity and Power Computation	115
5.5	Experimental Results	119
5.5.1	Experiment 1 : 64-Bit Adder Benchmark	120
5.5.2	Experiment 2 : Arithmetic Benchmarks	123
5.5.3	Experiment 3 : ISCAS-85 Benchmarks	124
5.5.4	Experiment 4 : Nonredundant ISCAS-85 Benchmarks	125
5.5.5	Experiment 5 : MCNC Synth89 Benchmarks	126
5.5.6	Remarks	128
6	Behavioral-Level Visualization of Switching Activity	132
6.1	Activity Viewer Tool	132
6.1.1	Input Transformation Process	135
6.1.2	Activity Viewer Results	135
6.2	Future Work and Summary	137
7	Conclusions	139
7.1	Contributions	139
7.2	Future Work	141
7.3	Impact of Contributions	142
	APPENDICES	143
A	Definitions and Formulas	143
B	Application of BLAPE to 4-bit Booth Multiplier	146
B.1	High-Level VHDL Specification Input	146
B.2	Boolean Equation Generation	148

B.3	Generate Implicit Structural Specification	158
B.4	Network Levelization	164
B.5	SIS Activity and Power Estimation	171
B.6	BLAPE Activity and Power Estimation	177
BIBLIOGRAPHY		184

LIST OF TABLES

2.1	Boolean expression signal probabilities.	22
2.2	Effects of uncorrelated inputs.	23
2.3	Effects of spatially correlated inputs.	23
2.4	Effects of temporal input correlations.	24
2.5	Effects of spatio-temporal input correlations.	24
2.6	Two-input AND, OR, XOR truth table.	26
2.7	Two-input AND, OR, XOR signal probability table.	27
2.8	Switching activities for two-input AND, OR, XOR gates.	27
3.1	Huntington's postulates.	47
3.2	Truth tables for NOT, AND, OR logic operations.	48
3.3	Truth table for full-adder carry-out bit.	49
4.1	Benchmark Results.	74
5.1	Post-order Boolean equations for FA carry-out bit.	103
5.2	Output from Generate_IPR_Cuberset procedure.	108
5.3	Signal probability computation for full-adder carry-out bit.	112
5.4	Symbol definitions.	115
5.5	Procedure running times.	115
5.6	64-bit adder size/power estimates.	120
5.7	Power/Accuracy estimates of arithmetic benchmarks.	123
5.8	Time estimates of arithmetic benchmarks.	123
5.9	ISCAS-85 power/accuracy benchmarks.	124
5.10	ISCAS-85 time benchmarks.	124
5.11	Nonredundant ISCAS-85 power/accuracy benchmarks.	125
5.12	Nonredundant ISCAS-85 time benchmarks.	125
5.13	MCNC Synth89 power/accuracy benchmarks.	127
5.14	MCNC Synth89 time benchmarks.	128
5.15	Benchmarks not available in SIS.	130

LIST OF FIGURES

1.1	EDA process.	4
1.2	Design phase.	6
1.3	Short-circuit current.	8
1.4	Capacitive current.	8
1.5	Standby current.	9
1.6	Leakage current.	9
1.7	Path balancing.	14
1.8	Technology decomposition.	15
2.1	Glitch example.	25
2.2	Reconvergent fanout circuit example.	28
2.3	Sequential circuit model.	31
2.4	Symbolic sequential circuit model.	36
2.5	k-unrolling of next state logic.	41
2.6	m-expanded network with m=2.	42
3.1	BDD of full-adder carry-out bit.	52
3.2	BDD with ordering 1.	53
3.3	BDD with ordering 2.	53
3.4	Behavioral VHDL model of FA carry-out bit.	55
4.1	Schematics of basic logic gates.	57
4.2	Logic diagram for FA carry-out bit.	58
4.3	Gate-level BLIF netlist for FA carry-out bit.	59
4.4	Structural VHDL model of FA carry-out bit.	60
4.5	Minimized-scalable BDDs.	62
4.6	Definition of CBDD.	66
4.7	CBDD of FA carry-out bit.	71
5.1	BLAPE methodology diagram.	79
5.2	Behavioral VHDL specification for full-adder.	83

5.3	Full-adder Boolean equations.	83
5.4	Netlist for implicit full-adder structure.	85
5.5	Logic diagram for implicit full-adder structure.	85
5.6	Netlist for mapped full-adder structure.	87
5.7	Logic diagram for mapped full-adder structure.	87
5.8	Implicit structural CBDD full-adder.	90
5.9	Mapped structural CBDD full-adder.	91
5.10	SPCA structural overview.	94
5.11	DAG of FA carry-out bit network.	97
5.12	Carry-out bit adjacency list and levelization.	97
5.13	Expression tree for full-adder carry-out bit.	100
5.14	BLAPE implicit structural FA activity/power estimates.	117
5.15	BLAPE mapped structural FA activity/power estimates.	117
5.16	SIS implicit structural FA activity/power estimates.	118
5.17	SIS mapped structural FA activity/power estimates.	118
5.18	Power v. k (64-bit adder).	121
5.19	Accuracy v. k (64-bit adder).	122
5.20	Time v. k (64-bit adder).	122
5.21	Average accuracy v. k (All benchmarks).	129
6.1	Activity color mapping.	133
6.2	Bar view.	134
6.3	Level view.	134
6.4	Booth multiplier(4-bit) level view.	136
6.5	Booth multiplier(4-bit) bar view.	137

CHAPTER 1

Introduction

The goal of accurately estimating dynamic power at the behavioral level of design abstraction has recently received increased attention. The demand for accurate behavioral-level computer-aided design (CAD) tools for power analysis is due to the IC industry's large-scale migration to HDL-based hardware design methodologies, and the emergence of power consumption as a key VLSI design parameter

The estimates generated by many behavioral-level power analysis CAD tools contain inaccuracies stemming from a lack of specific information which is determined or specified at deeper levels (*i.e.*, the circuit- and physical- levels) within the electronic design automation (EDA) process. Despite these inaccuracies, HDL-based hardware design methodologies are necessary for efficient hardware design. Currently, the benefits of behavioral-level design outweigh the inaccuracies found in the estimates of behavioral-level power analysis CAD tools.

The focus of this research is to develop enhanced tools which provide additional accuracy for behavioral-level design. The research presented in this dissertation outlines a new technique which provides improved accuracy of switching activity and power estimates using behavioral-level design specifications described in VHDL. Before discussing the details of this research, a review of the EDA process and the motivation for this research topic is given in the following sections.

1.1 Background

1.1.1 Electronic Design Automation Process

Today's complex circuit designs require computer support for virtually all aspects of design. Computer-based design automation (DA) tools make the design of very large or complex circuits feasible. Designs which are too large or complex for manual design use such processes for improved quality (performance and reliability), reduced product cost, and shortened design time. Given a specification of an abstract object, a computer-based DA system generates the physical design automatically and verifies that the design satisfies its requirements specification. The design process followed by DA systems can be viewed as a sequence of transformations on the following design representations: *behavioral*, *structural*, and *physical*, at various levels of abstraction.

1.1.2 Design Representations

- *Behavioral representations* - describe a circuit's function. Behavior can be described in a functional or procedural fashion. Emphasis is placed on what the design does, not how it is built. Designs are viewed as one or more black boxes with sets of inputs and outputs, and a set of functions describing the behavior of each output in terms of the inputs over time.
- *Structural representations* - describe the composition of circuits in terms of cells (abstractions of circuit element definitions) and components (abstractions of instances of circuit elements) and the interconnection among these components. Structural descriptions include block diagrams, schematic drawings, and netlists.
- *Physical representations* - are characterized by information used to manufacture and fabricate the physical system. They are concerned with binding the

structural design space to silicon. Physical information includes geometric layout data such as transistor location and wire routing.

1.1.3 Design Optimization

When designing complex integrated circuits, removing false design paths is as important as executing the correct design procedures. The removal of false design paths is accomplished by analyzing a number of design variations and choosing the one that best meets the system's requirement specifications. The earlier one optimizes a design or removes an unsatisfactory design variation, the less effort one exerts in pursuing an alternative design that does not meet requirements.

1.1.4 Design Phases

The electronic design automation process is a top-down approach to designing large and complex integrated circuits. The EDA process starts with a system specification and continues through a series of abstraction level transformations, resulting in separate design phases (Figure 1.1). Each succeeding level adds more specific or detailed design information. The process is briefly described below.

- *Design Specification* considers the following: 1) application of system, 2) performance requirements, 3) system architecture, 4) external interfaces and protocols, and 5) manufacturing costs.
- *Behavioral Design* is synthesized to meet specifications. The result is a behavioral representation such as Boolean expressions, differential equations, instruction set descriptions, algorithms, or flowcharts. Behavioral simulation is the method of analysis.

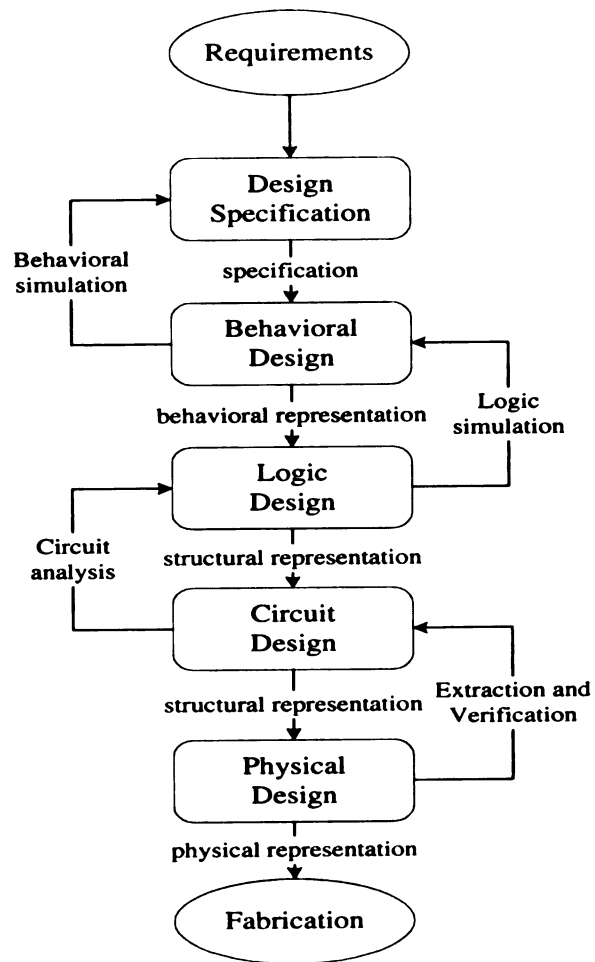


Figure 1.1. EDA process.

- *Logic Design* is concerned with the logic structure that implements the behavioral design. The design representation may be a *register-transfer-level* (RTL) description, schematic description, logic diagram, or netlist of gates. For analysis the representations are simulated at the transistor-, gate-, and register-transfer- levels. Validation takes place by comparing results from the logic-level and behavioral-level simulations.
- *Circuit Design* is concerned with the electrical laws that govern the detailed behavior of the basic elements such as transistors, resistors, capacitors, and inductors. Transistors are sized to meet signal delay requirements. Analysis is performed using circuit and timing simulations.
- *Physical Design* is concerned with the transformation of the structural representation from the previous phase into the geometric shapes (and layout cells) representing details of the fabrication process. Additionally, the placement of cells and routing are important concerns of the physical design stage.

1.1.5 Steps Within the Design Phase

The process of removing a false design path is governed by the following operations: *synthesis*, *analysis*, and *verification*, illustrated in Figure 1.2.

- *Synthesis* derives a new design representation based on the representation of a previous stage.
- *Analysis* evaluates the correctness of a design representation against its requirements.
- *Verification* provides a formal process for demonstrating the equivalence of two design representations under specified conditions.

The

the

of

in

of

an

pr

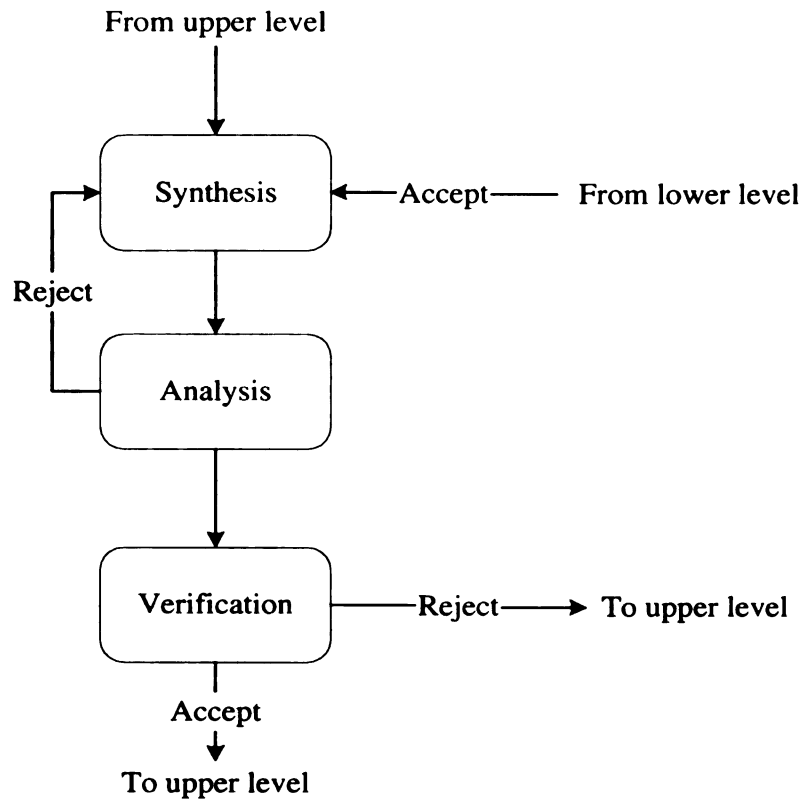


Figure 1.2. Design phase.

1.2 Motivation

The emergence of low power as a critical VLSI design parameter combined with the popularity and large-scale movement to HDL-based hardware design methodologies provide the motivation for this research effort. Additionally, the overwhelming inaccuracies in previous behavioral-level power analysis CAD tools present a significant problem, whose elimination will benefit various aspects of the electronic design automation process. Discussions on low power design and HDL-based design are presented next.

1.3 Low Power Design

Reducing power consumption has emerged as a critical design parameter for digital VLSI systems [1]. The trend has been to develop methodologies and techniques which maintain a circuit's throughput and area constraints while achieving some desired level of power efficiency. The power efficiency revolution was initiated by the introduction of high-throughput portable electronic devices, such as laptop computers, portable televisions, camcorders, and wireless communications systems. For these devices it is desirable to support high-speed computation with complex functionality, while more efficiently using minimal size and minimal weight batteries. Advances in battery technology have also promoted the design of power efficient circuits. It is anticipated that battery lifetimes will increase to about 90 - 110 watt-hours/kilogram over the next five years [1]. If low power design techniques are not considered, then high-throughput portable electronic devices will suffer from short operation times due to limited battery lifetimes or become burdened by heavier battery packs. Other factors which motivate the design of power efficient circuits include chip packaging costs, cooling, and reliability.

1.3.1 Sources of Power Dissipation

Power dissipation in digital CMOS circuits can be classified into two categories: *static* and *dynamic* dissipation. The static and dynamic power dissipations are determined by the way in which the individual MOS transistors circulate current [2]. There are four main currents which are related to power dissipation in CMOS circuits: *short circuit current*, *capacitive current*, *leakage current*, and *standby current*.

The short circuit current arises when both NMOS and PMOS transistors are simultaneously conducting current from the supply to ground during an input transition. The capacitive current is present when charging and discharging of a capacitance takes

ph
co
dr
th
J
V
th
cu
ag

place to switch the output state of the logic device. The short circuit and capacitive currents are illustrated in Figures 1.3 and 1.4 for a CMOS inverter. Short-circuit and capacitive currents are known as dynamic currents because they result from the switching which arises when an input transition takes place. The magnitude of these currents are on the order of microamperes and milliamperes.

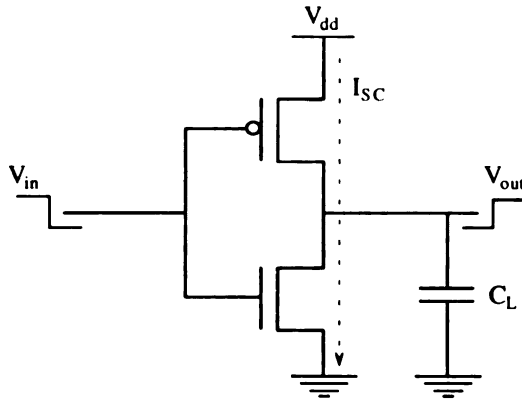


Figure 1.3. Short-circuit current.

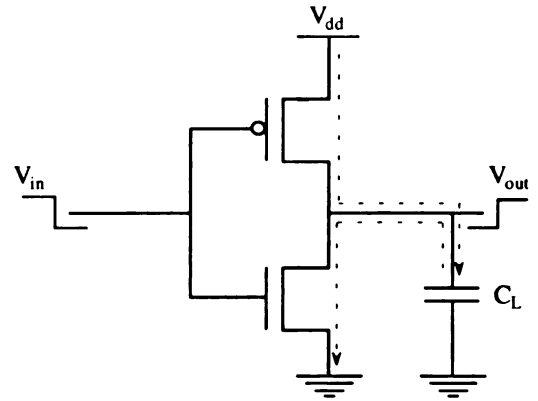


Figure 1.4. Capacitive current.

The standby current represents current which is continuously drawn from the supply to ground. The leakage current, which is a property of the fabrication technology, consists of reverse bias current in the parasitic diodes, which are formed by source-drain diffusions and p-well or n-well diffusions. The leakage current contributed by the reverse biased parasitic diodes can be described by the diode equation: $i_D = I_0[e^{\frac{qV}{kT}} - 1]$, where I_0 is the reverse saturation current, q is the charge of an electron, V is the voltage across the diode's *pn* junction, k is Boltzman's constant, and T is the temperature of the device material in degrees Kelvin.

An additional component of leakage current is due to subthreshold conduction current, which is generated by the inversion charges that are produced for gate voltages below the threshold voltage. The standby and leakage currents, shown in Figures

st
C
le
by

Th
L
en
ta
cir

1.5 and 1.6 for a CMOS inverter, are known as static currents. These currents result only from applying power to the device. The leakage and standby currents are very small, on the order of nanoamperes and microamperes, respectively. The leakage and

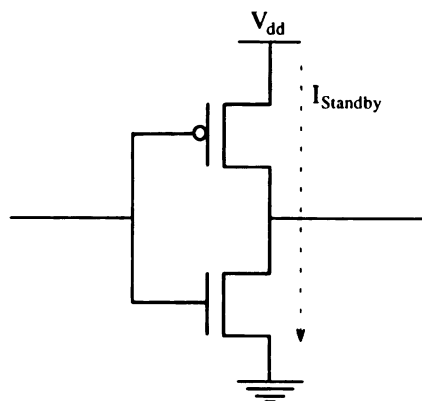


Figure 1.5. Standby current.

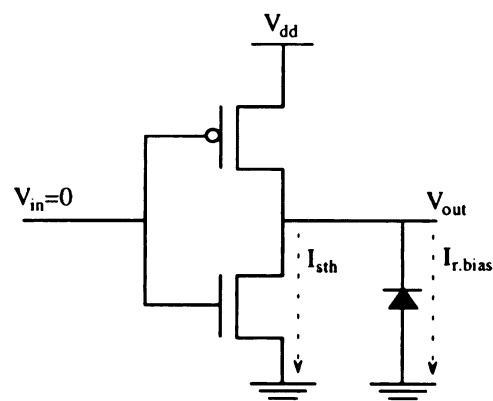


Figure 1.6. Leakage current.

standby currents are responsible for the static power dissipation component in digital CMOS circuits. The static power dissipation quantity is the result of summing the leakage and standby currents to form a static current, and multiplying this current by the supply voltage (V_{dd}). The equations are

$$I_{Static} = I_{Leakage} + I_{Standby}$$

$$P_{Static} = I_{Static}V_{dd}.$$

The dynamic power dissipation due to short circuit current is given by Equation 1.1, where α represents the switching activity factor or the probability that a power-consuming transition occurs, Q_{SC} is defined as the amount of charge transferred per transition, f_{CLK} is the clock frequency, and V_{dd} is the supply voltage. The short circuit power dissipation contributes about 10 percent to the total power dissipation.

$$P_{Short} = \alpha Q_{SC} f_{CLK} V_{dd}. \quad (1.1)$$

The dynamic power dissipation due to charging/discharging of node capacitances is referred to as switching activity power or switched-capacitance power. This power component is given by Equation 1.2, where C_L represents the node capacitance, and the other components are as defined previously. The switched-capacitive power dissipation contributes about 90 percent to the total power dissipation.

$$P_{Switch} = \frac{1}{2} \alpha C_L f_{CLK} V_{dd}^2 \quad (1.2)$$

The switched-capacitance power is much larger than short-circuit power dissipation due to the low rate of occurrence in which both NMOS and PMOS transistors of the CMOS inverter are simultaneously on. The dynamic power dissipation quantity is the result of summing the short-circuit and switched-capacitive power dissipations, given by

$$P_{Dynamic} = P_{Short} + P_{Switch}. \quad (1.3)$$

When all of the power dissipation components are combined, the total power dissipation is

$$P_{Total} = P_{Dynamic} + P_{Static} \quad (1.4)$$

or

$$P_{Total} = \frac{1}{2} \alpha C_L f_{CLK} V_{dd}^2 + \alpha Q_{SC} f_{CLK} V_{dd} + I_{Static} V_{dd}. \quad (1.5)$$

The most dominant component is the switched capacitance power dissipation, which accounts for 90 percent of the power dissipation in CMOS circuits [3]. Most of the research in the area of low power design is concerned with reducing the switched capacitance power dissipation component because of its dominance.

1.3.2 Low Power Optimization Techniques

The majority of low power circuit research targets the dynamic power component as a means of reducing power. The switched-capacitance power, which is the power consumed in CMOS circuits caused by switching currents, is the primary focus because it is responsible for 90 percent of the total power dissipation. The switched-capacitance power for a single gate is given by Equation 1.2. The supply voltage (V_{dd}), physical capacitance (C_L), and switching activity factor (α) are the parameters most targeted for optimization within the switched-capacitance power component. Other methods for switched-capacitance power minimization will be addressed in this discussion. Reducing the supply voltage is most attractive because of its quadratic relationship to power. A factor of two decrease in the supply voltage will yield a factor of four decrease in the switched-capacitance power. However, as supply voltage is lowered, circuit delays increase leading to reduced system performance. For $V_{dd} > V_t$, delays increase linearly with decreasing voltage. This is expressed by

$$Delay \propto \frac{V_{dd}}{(V_{dd} - V_t)^2},$$

where V_t is the threshold voltage, or the voltage level at which the transistor conducts.

1.3.2.1 Supply Voltage Reduction

One approach taken to scale or reduce supply voltage without sacrificing throughput was reported in [1]. In this approach the threshold voltage (V_t) is reduced, allowing the supply voltage to be scaled down without loss of speed. Reducing both supply and threshold voltages by some small amount maintains the operational behavior of the MOS transistors, without decreases in device currents. The limitations in lowering the threshold voltage are due to the requirement to retain adequate noise margins and control of increased subthreshold leakage currents. In [4], a study was conducted

on the effect of reducing the supply voltage for a variety of different logic circuits, containing from 56 to 44,000 transistors. The results of the study indicated that a speed penalty was incurred for all circuits and that delays drastically increased as the supply voltage approached the sum of the threshold voltages of the devices. Shen *et al.* [5] suggested a method of designing lower power circuits, where the initial task is to build the circuit to be fast as possible, regardless of the area and power. Finally, the supply voltage is decreased to lower power dissipation.

1.3.2.2 Physical Capacitance Reduction

A technique used for minimizing switched-capacitance power dissipation involves lowering or reducing the circuit's physical capacitance. Power dissipation is dependent upon the physical capacitance associated with each of the individual gates in the circuit. Methods for reducing physical capacitance include using less logic, smaller transistor sizing, and shorter wirelength. Techniques for optimizing logic include resource sharing and improved logic equation minimization methods. The reduction or scaling of transistor size reduces physical capacitances, but also reduces the current drive capability of the transistor, resulting in slower circuits [3]. Lowering physical capacitance by means of using shorter wiring can be achieved by improved placement and routing strategies, which attempt to optimally locate and position logic blocks such that minimal wiring is used for block interconnections. Optimal wiring strategies result in reducing physical capacitance and I^2R power losses. Improvements in the placement and routing areas are discussed in [6], where a new implementation of the simulated annealing algorithm is presented. The new algorithm makes use of an improved cost function which includes an enhanced overlap penalty function, combined with the inclusion of a timing path penalty function. These improvements allow for the optimized placement and routing of rectilinearly shaped macro cells and have produced modest reductions in chip area and total wire length.

1.3.2.3 Switching Activity Reduction

Minimizing the switching activity factor is another means for optimizing power dissipation in CMOS circuits. The switching activity is dependent upon the logic function implemented and the primary input signals to the circuit. The switching activity α at node x is defined as the fraction of time the node performs a transition within a clock period. Switching activity may be reduced at the algorithmic and architecture levels.

One method for reducing switching activity involves optimizing the number representation [7]. For certain signal processing applications, a change from the two's complement number representation to the sign-magnitude number representation gave modest improvements. It was found that the two's complement representation was subject to higher switching activity when the sign of input signal values changed. This is due to the fact that sign extension causes many of the most significant bits (MSB's) to toggle. In the case of the sign magnitude representation, only a single bit toggles when the input signal changes sign, resulting in a reduction of switching activity for some of the MSB's.

A method to reduce the switching activity factor by means of *path balancing* is illustrated in Figure 1.7. Path balancing reduces glitches or spurious transition activity in combinational logic circuits. Glitching is reduced if paths in the circuit that converge at certain gates all have roughly equal lengths (delays) [5]. Balancing path delays leads to nearly simultaneous switching on the input signals to a gate, and thus eliminates possible hazards at the output of the gate. This equalization of path delays in a circuit results in a reduction of spurious gate output transitions which reduces switching activity, thus lowering power dissipation. Path balancing can take place before technology mapping by selective collapsing and logic decomposition or after technology mapping by delay insertion and pin reordering.

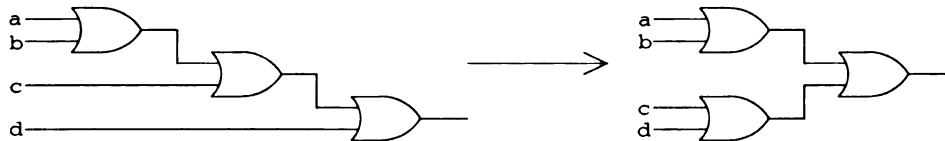


Figure 1.7. Path balancing.

A switching activity reduction method has been proposed for finite state machines (FSM's), where an encoding of states was the focus of attention [8]. This method uses a hypercube embedding technique and generates state encodings such that the sum of bit toggles between each pair of states multiplied by the encoding affinity between states is minimized. By reducing the number of bit toggles in each state transition, the switching activity in the combinational logic which determines the next state and output functions is reduced.

1.3.2.4 Structural Optimization

Techniques which consider a design's structure and component interconnection also offer reductions in switching activity. Various combinations and arrangements of specific logic gates and subcircuits lead to improved power reduction for certain designs.

Sobelman *et al.* [9] proposed a power dissipation reduction technique for multiplier circuits. This method makes use of a self-timed evaluate signal, such that each carry-save or carry-propagate adder within the array triggers only after all of its inputs have stabilized. This technique avoids spurious switching of internal nodes so that the average power dissipation is minimized. Moshnyaga *et al.* [10] proposed another multiplier power dissipation improvement. The goal of this method is to lower the switching activity per operation by reducing the number of active elements in the adding array. This method incorporates the use of 4-2 compressors which lower the number of propagation stages in the adding array. The 4-2 compressors have five inputs and three outputs which can compress four partial products into two. The



compressor has the same logic function as that of a carry-save adder constructed by two serial full adders, but uses fewer transistors and has 25% less propagation delay.

Tsui *et al.* [11] discuss a *technology decomposition* technique which converts a set of Boolean equations, or network, to another network consisting of only AND and INVERTER gates. This technique minimizes the total switching activity in the final two-input AND tree using a zero-delay model. The goal of the procedure is to apply high-activity inputs into the tree composition at the latest possible stage. Consider Figure 1.8 for example, where $P(x_i)$ is the signal probability of primary input or internal signal and $E_{sw}(g)$ represents the switching activity of a gate. In this technology decomposition example, the highest activity signal (d) is applied last in the tree decomposition of configuration A , thus yielding a lower switching activity than the equivalent logic in configuration B . The placement of higher activity signals in later stages of the tree decomposition limits the switching activity experienced by internal gates and lowers the number of transitions taking place at the gate's output.

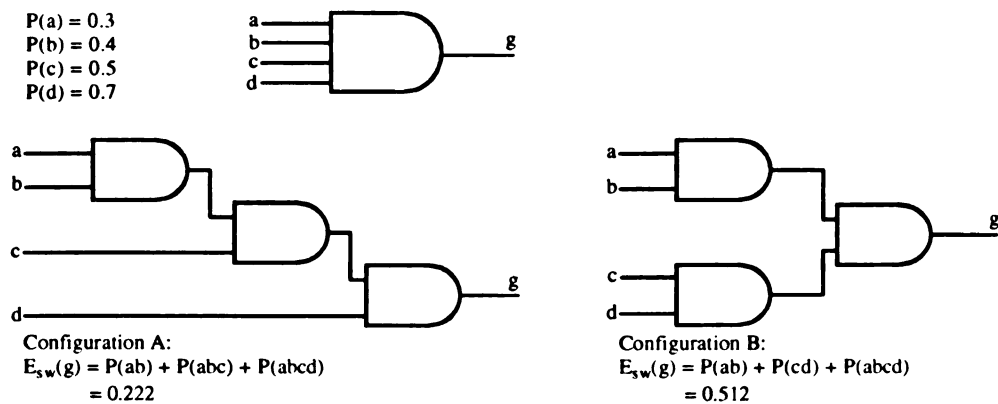


Figure 1.8. Technology decomposition.

1.3.2.5 System and Architectural Level Optimizations

Other techniques for power dissipation minimization which have received attention are *power-down methods*, *precomputation*, *don't care optimization*, *low-power software optimization*, and *adiabatic-switching*. In the case of the power-down approach, blocks of logic not involved in the present computation are automatically turned off to save power. Methods for detecting and disabling unused blocks as well as scheduling algorithms which maximize the "shut-down" period of execution units are discussed in [1, 12]. When certain conditions are satisfied, modules are disabled, thus eliminating any switching activity and power dissipation. In the precomputation technique the idea is to selectively precompute the output logic values of the circuits one clock cycle before they are required, and then use the precomputed values to reduce internal switching activity in succeeding cycles [13]. The technique for *don't care* optimization reported in [14] explores the Boolean space in an effort to identify minterms highly appropriate for influencing switching activity. A partitioning of the *don't care* set into regions strongly and weakly influential upon switching activity is performed. This variance is exploited to bias area optimization towards reduced power dissipation.

Methods for the measurement of power dissipation for the software component of an embedded system have received some attention. For some time, application specific software running on dedicated microprocessor or microcontroller-based systems have been optimized for size and speed. In [15], an approach for estimating the power cost of embedded software is presented. This experimental approach involves the measurement of the amount of current drawn by the microprocessor or microcontroller when instructions are executed. The energy cost of the instruction is the observed average current value multiplied by the number of cycles taken by the instruction. By reordering several sequences of instructions, the average current for the execution of a program was reduced. The goal of this method is to assign energy costs to

each instruction and to generate power-efficient software by selecting and ordering instructions such that the overall program energy cost is minimized.

Athas *et al.* [16] explore a technique for low-power CMOS design by means of constructing combinational and sequential adiabatic switching logic circuits. The concept behind adiabatic switching logic is to recycle signal energies stored in circuit capacitances instead of allowing the energies to dissipate in the form of heat.

1.4 HDL-Based Design

Many IC manufacturers are changing their hardware design methodologies to an HDL-based design approach. The primary benefit of the HDL-based design approach is higher productivity [17]. Increased efficiency is achieved when using an HDL by the use of available cores (*e.g.*, UARTs, bus interfaces, microcontrollers) to incorporate functions into the design. Another example of HDL design efficiency is exhibited in its ability to facilitate reuse of already designed components.

HDLs promote hierarchical design allowing one to build more flexibility into lower level building blocks. The ability to define components at the structural-level or behavioral-level of abstraction is a key feature of an HDL. This results in shorter product design completion times.

Additionally, designers have less difficulty re-targeting HDL-based designs to different programmable-logic families than with schematic-based designs. This advantage allows a developer to quickly compare the device cost, performance, and other factors with product-lines of various vendors.

To summarize, HDL-based hardware design methodologies promote design reuse, component definition, design flexibility, and shortened design time. These advantages translate into higher productivity and reduced cost.

1.5 Problem Statement

The research of this dissertation addresses the power dissipation problem. The objective is to find a more accurate and low-cost method of computing switching activity and power dissipation for behavioral-level designs described in VHDL. The main goal is to achieve an accurate or reasonably approximate estimate of switching activity with minimal computational complexity and memory resources. Additionally, the research is concerned with identifying and locating high activity circuit nodes through the development and implementation of new activity visualization tools.

The use of the proposed estimator and visualization tool will assist circuit designers in the development of power-efficient designs. IC designs will greatly benefit from the use of the proposed techniques in terms of design flexibility, time, cost, and reuse.

The research is centered on the use of VHDL because of its widespread use in industry and academic research environments for development, simulation and testing capability of high-level and implementation-free designs. VHDL also supports behavioral and structural level design. It allows the development of digital systems based on functional descriptions or component interconnections.

A majority of switching activity estimation techniques target gate- and circuit-level design descriptions. Very few techniques exist for accurately computing switching activity at higher levels of design abstraction. On average most high-level switching activity and power estimates contain about 12% error [18]. The goal of this research is to improve the accuracy of high-level switching activity and power estimators and to be competitive in terms of accuracy, CPU time and memory resources with respect to gate- and circuit- level estimators.

1.6 Dissertation Overview

Chapter 2 describes the switching activity estimation problem. It provides background and fundamental information detailing the factors that complicate the accurate calculation of switching activity for CMOS circuits. Additionally, a survey of current and past switching activity estimation techniques, along with a description of their advantages and disadvantages, is given.

Chapter 3 provides a discussion of the topic of behavioral-level design. Specifically, this chapter addresses some implementation-free or behavioral methods for representing switching functions. Methods such as truth tables, Boolean equations, Binary Decision Diagrams (BDDs), and VHDL behavioral specifications are described.

In Chapter 4, the concept of structural methods for representing switching functions is reviewed. The most common structural representation, the gate-level netlist is illustrated, along with an overview of VHDL structural specifications. Additionally, an introduction and overview of the Connective Binary Decision Diagram (CBDD) is presented.

In Chapter 5, an overview of the new methodology for accurately estimating the switching activity of behavioral-level designs described in VHDL is given. The specific assumptions and constraints that make the new technique a success are highlighted. Additionally, the algorithms and the intermediate transformations are discussed. To demonstrate the effectiveness of the new approach, results and benchmark comparisons with other power analysis CAD tools are given.

Chapter 6 presents a new visualization tool that identifies and highlights the power-hungry areas of the circuit design. A description of the tool's input transformation process as well as an overview of the activity views is given. A set of examples are used to highlight the usefulness of the new tools.

Chapter 7 contains conclusions and a discussion of future enhancements that could

improve the behavioral-level activity and power estimator methodology and supporting algorithms.

Appendix A contains definitions and theorems used throughout the dissertation. Finally, Appendix B contains a walk-through of the BLAPE algorithm when applied to a 4-bit Booth multiplier, described in VHDL.

C

S

The
power
when
by ev
speci

2.1

Se
signa
of ca
is th
signa

2.1.1

switc
calen

CHAPTER 2

Switching Activity Estimation

The switching activity factor, α , is a key component of the switched-capacitance power dissipation model, described by Equation 1.2. Dynamic power is dissipated whenever any internal switching occurs. Thus, switching activity can be determined by evaluating and summing the $0 \rightarrow 1$ and $1 \rightarrow 0$ transition probabilities at the specified node.

2.1 Calculation of Switching Activity

Switching Activity, denoted $E_{sw}(x)$ or $\alpha(x)$, is defined as the probability that the logic signal at node x experiences a change in its logic state. The most common method of calculating switching activity involves the use of *signal probability*, $P_s(x)$, which is the probability that the signal at node x is equal to logic 1. The evaluation of signal probabilities for inputs and simple Boolean expressions is described in Table 2.1. Upon determining a node's signal probability, $P_s(x)$, the calculation of the node's switching activity is given by Equation 2.1. A derivation of the switching activity calculation model is provided in Appendix A.

$$E_{sw}(x) = 2 \cdot (1 - P_s(x)) \cdot P_s(x) \quad (2.1)$$

T

stre

unav

depe

follow

on ste

conce

2.1.

The s

abilit

depen

puts

appar

fore,

that (

twice

<i>Function</i>	<i>Signal Probability</i>	<i>Assumptions</i>
A	$P_s(A)$	--
\overline{A}	$1 - P_s(A)$	--
$A \cdot B$	$P_s(A) \cdot P_s(B)$	<i>A Indep B</i>
$A + B$	$P_s(A) + P_s(B) - P_s(A \cdot B)$	<i>A Indep B</i>
$x_1 x_2 \cdots x_n$	$\prod_{i=1}^n P_s(x_i)$	<i>All x_i Indep</i>
$x_1 + x_2 + \dots + x_n$	$1 - \prod_{i=1}^n (1 - P_s(x_i))$	<i>All x_i Indep</i>

Table 2.1. Boolean expression signal probabilities.

The calculation of switching activity is difficult because it depends on circuit input streams, various circuit parameters, and technology-dependent factors which may be unavailable or difficult to characterize. Some of these factors include *input pattern dependence*, *glitch activity*, *delay model*, *logic function*, and *circuit structure*. The following sections discuss details of switching activity computations which are based on statistical or probabilistic concepts. Appendix A contains definitions of terms and concepts used in the following sections.

2.1.1 Input Pattern Dependence

The switching activity, $E_{sw}(g)$, at gate output g , depends on the input signal probabilities, the gate's logic function, as well as spatial and temporal input pattern dependencies. For example, consider a two-input NAND gate with independent inputs (x_1, x_2) , such that their signal probabilities are each $\frac{1}{2}$. From Table 2.2, it is apparent that 6 out of 16 input patterns result in a gate output transition. Therefore, $E_{sw}(g) = \frac{3}{8}$. When spatial correlation conditions are applied to (x_1, x_2) such that $(0, 0)$ and $(1, 1)$ are the only input patterns, the NAND gate output switches twice out of the possible four input arrangements and therefore $E_{sw}(g) = \frac{1}{2}$ (Table

2.3). Given that a temporal correlation governs the inputs, where each 0 applied to input x_1 is followed by a 1, while each 1 applied to input x_2 is followed by a 0, 4 output transitions out of 9 input patterns take place, with $E_{sw}(g) = \frac{4}{9}$ (Table 2.4). If spatial-temporal correlation conditions are governing the input dependence, such that x_2 changes exactly when x_1 changes, then $E_{sw}(g) = \frac{1}{4}$ (Table 2.5).

x_1	x_2	$NAND$
0 → 0	0 → 0	1 → 1
0 → 0	0 → 1	1 → 1
0 → 1	0 → 0	1 → 1
0 → 1	0 → 1	1 → 0
0 → 0	1 → 0	1 → 1
0 → 0	1 → 1	1 → 1
0 → 1	1 → 0	1 → 1
0 → 1	1 → 1	1 → 0
1 → 0	0 → 0	1 → 1
1 → 0	0 → 1	1 → 1
1 → 1	0 → 0	1 → 1
1 → 1	0 → 1	1 → 0
1 → 0	1 → 0	0 → 1
1 → 0	1 → 1	0 → 1
1 → 1	1 → 0	0 → 1
1 → 1	1 → 1	0 → 0

Table 2.2. Effects of uncorrelated inputs.

x_1	x_2	$NAND$
0 → 0	0 → 0	1 → 1
0 → 1	0 → 1	1 → 0
1 → 0	1 → 0	0 → 1
1 → 1	1 → 1	0 → 0

Table 2.3. Effects of spatially correlated inputs.

x_1	x_2	$NAND$
0 → 1	0 → 0	1 → 1
0 → 1	0 → 1	1 → 0
0 → 1	1 → 0	1 → 1
1 → 0	0 → 0	1 → 1
1 → 0	0 → 1	1 → 1
1 → 0	1 → 0	0 → 1
1 → 1	0 → 0	1 → 1
1 → 1	0 → 1	1 → 0
1 → 1	1 → 0	0 → 1

Table 2.4. Effects of temporal input correlations.

x_1	x_2	$NAND$
0 → 0	0 → 0	1 → 1
0 → 0	1 → 1	1 → 1
0 → 1	0 → 1	1 → 0
0 → 1	1 → 0	1 → 1
1 → 0	1 → 0	0 → 1
1 → 0	0 → 1	1 → 1
1 → 1	0 → 0	1 → 1
1 → 1	1 → 1	0 → 0

Table 2.5. Effects of spatio-temporal input correlations.

2.1.2 Glitch Activity

Spurious transitions at any node outputs represent an additional component of the switching activity. The spurious transitions, known as glitches or glitch activity, are unwanted transitions that occur before a node settles to its final steady-state value. The simple circuit and waveform shown in Figure 2.1 demonstrate the effects of glitch activity. The output y of the circuit should always remain at logic one, but due to

if
l.
s.
re-
to

may

typic

acco

such

due t

2.1.

The c

such

tion c

all ch

internal delays injected by the inverter (denoted by signal x^c) a transient drop to logic zero occurs at the output. Glitches are apparent at the architectural level in static designs due to finite propagation delays from one logic block to the next block resulting in a node having multiple transitions in a single clock cycle before settling to the correct logic level [4]. Each of the spurious transitions consumes power and

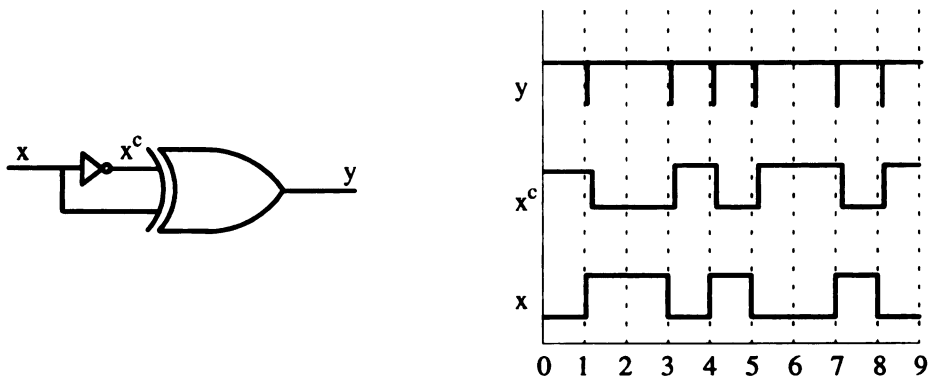


Figure 2.1. Glitch example.

may account for as much as 10 to 40 percent of the switching activity power loss in typical combinational logic circuits [3]. It was reported by Shen *et al.* [5] that glitches accounted for 20 percent of the power dissipation over a range of circuits, and circuits such as combinational adders were subject to a 70 percent or more power dissipation due to glitch activity.

2.1.3 Delay Model

The delay model used for switching activity estimation is an important factor. Models such as zero delay and real delay make different assumptions concerning the propagation of the signals throughout the circuit. In the zero delay model it is assumed that all changes at the circuit inputs reach the internal gates of the circuit instantaneously,

which allows for a glitch-free representation of the circuit. In the case of the real delay model, each gate in the circuit is given a delay. This in turn, may cause internal and output nodes to experience multiple transitions during a single input transition. In many circuits, glitch activity accounts for a large percentage of the switching activity, thus making the calculation of the switching activity more difficult. The difficulties arise in the determination of the glitch locations.

2.1.4 Logic Function

The logic function implemented by a circuit directly influences the switching activity. The logic function of a gate determines the probability that the present value of the gate will differ from the previous value. For example, the truth table (Table 2.6) provides the logic input/output conditions for two-input AND, OR, and XOR gates. To mimic the logic in this truth table from a probability perspective, the signal probability table (Table 2.7) uses p to denote the probability that an input signal equals logic 1 or $P(x_i = 1)$, and uses q , which is $1 - p$, to denote the probability that an input signal equals logic 0 or $P(x_i = 0)$. The signal probability for the logic gate outputs are computed by summing the probability permutation entries, which indicate a logic 1 at the gate's output of the truth table for the corresponding input combination. The resulting switching activities for each two-input gate (Table 2.8) are computed by applying Equation 2.1 to the signal probabilities.

x_1	x_2	<i>AND</i>	<i>OR</i>	<i>XOR</i>
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Table 2.6. Two-input AND, OR, XOR truth table.

$P_s(x_1)$	$P_s(x_2)$	$P_s(AND)$	$P_s(OR)$	$P_s(XOR)$
q	q	q^2	q^2	q^2
q	p	qp	(qp)	(qp)
p	q	pq	(pq)	(pq)
p	p	(p^2)	(p^2)	p^2

Table 2.7. Two-input AND, OR, XOR signal probability table.

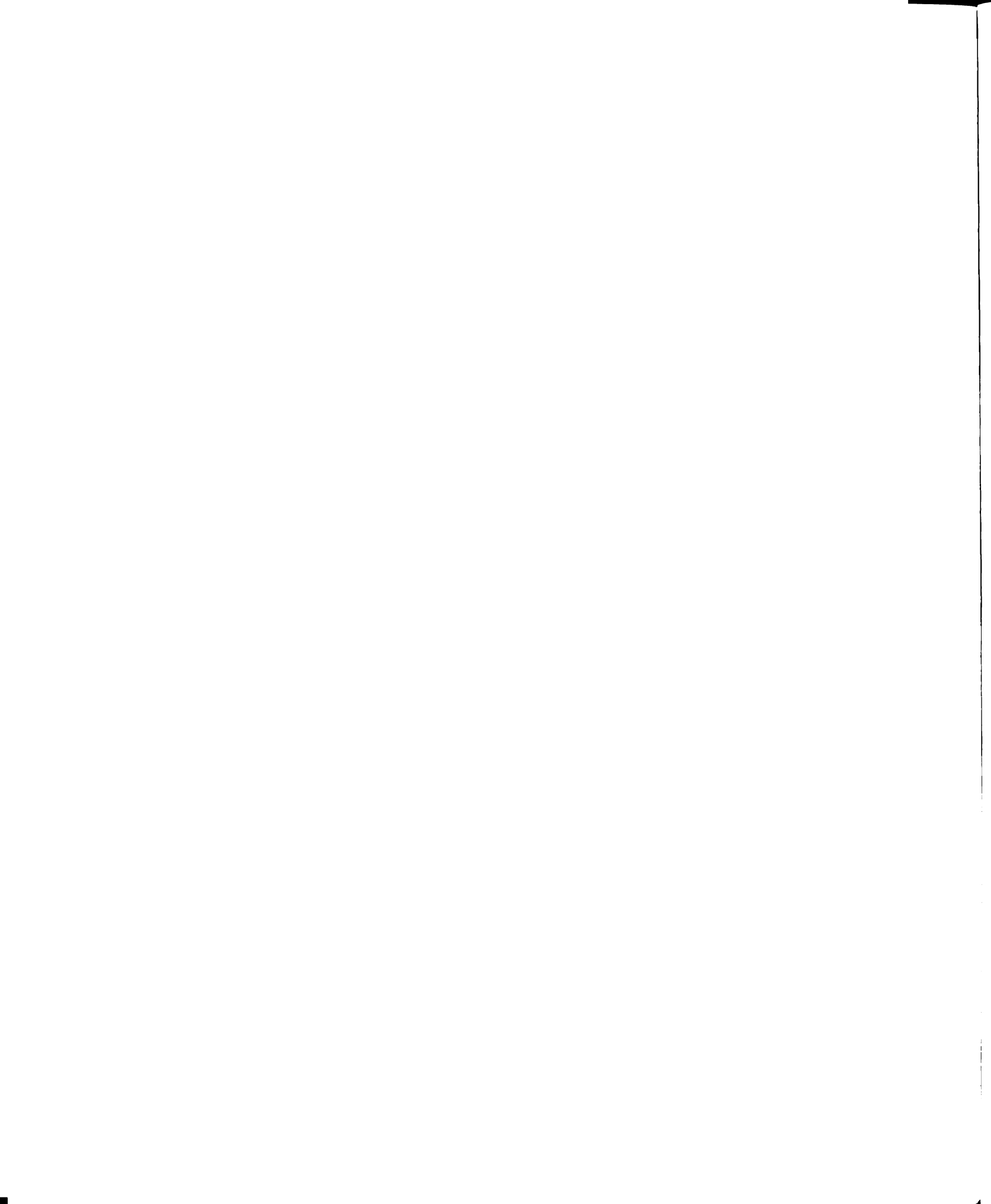
g	$P_s(g)$	$E_{sw}(g)$
<i>AND</i>	p^2	$2p^2(1 - p^2)$
<i>OR</i>	$1 - q^2$	$2q^2(1 - q^2)$
<i>XOR</i>	$2pq$	$4(1 - 2pq)pq$

Table 2.8. Switching activities for two-input AND, OR, XOR gates.

2.1.5 Circuit Structure

The circuit structure may also cause difficulties in computing the switching activity when reconvergent fanout nodes are considered. This problem is more challenging because the internal signals may be correlated, potentially requiring a large amount of computational effort and memory usage. Some power estimation techniques ignore these fanout correlations. Approximations may be used to improve accuracy while shortening the execution time of the simulation.

The simple circuit depicted in Figure 2.2 contains reconvergent fanout. The gate outputs, M_1 and M_2 , are correlated due to their common dependence on input x_2 . The correlation results in a signal probability overestimate (Equation 2.4) at output F , when assuming M_1 and M_2 are independent. This common inaccuracy is attributed to a statistical inequivalence of two Boolean expression forms (Equations 2.2 and 2.3)



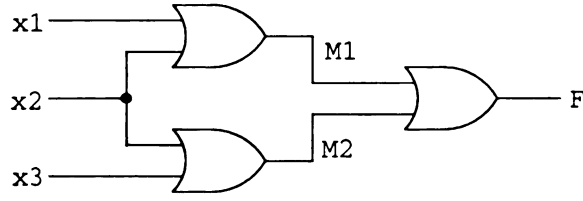


Figure 2.2. Reconvergent fanout circuit example.

which equivalently represent the output F . The correct signal probability (Equation 2.5) for node F is obtained when minimizing node F 's Boolean expression to Equation 2.3, followed by applying the *OR*-signal probability calculation given in Table 2.1. In the reconvergent fanout example below, all input signal probabilities are assumed to be $P_s(x_i) = \frac{1}{2}$.

$$\begin{aligned}
 f_1 &= M_1 + M_2 \\
 &= (x_1 + x_2) + (x_2 + x_3)
 \end{aligned} \tag{2.2}$$

$$f_2 = (x_1 + x_2 + x_3) \tag{2.3}$$

$$\begin{aligned}
 P_s(f_1) &= P_s(M_1 + M_2) \\
 &= 1 - (1 - P_s(M_1))(1 - P_s(M_2)) \\
 &= 1 - (1 - P_s(x_1 + x_2))(1 - P_s(x_2 + x_3)) \\
 &= \frac{15}{16} \text{ (Overestimate)}
 \end{aligned} \tag{2.4}$$

$$\begin{aligned}
 P_s(f_2) &= 1 - (1 - P_s(x_1))(1 - P_s(x_2))(1 - P_s(x_3)) \\
 &= \frac{7}{8} \text{ (Correct)}
 \end{aligned} \tag{2.5}$$

$$\begin{aligned}
 E_{sw}(f_2) &= 2 \cdot P_s(f_2) \cdot (1 - P_s(f_2)) \\
 &= 2 \cdot \frac{7}{8} \cdot \frac{1}{8} \\
 &= \frac{14}{64} \\
 &= 0.219
 \end{aligned}$$

S.

to

ab

on

of

in

M.

ce

w

pe

in

a

s

a

r

r

r

r

r

r

r

r

r

r

r

r

r

2.2 Previous Work

Switching activity estimation techniques have usually been classified as either *statistical* or *probabilistic*, and in most cases are used at the circuit- or gate- level of abstraction. In statistical methods, traditional models are used to simulate the circuit for a set of randomly chosen input vectors while monitoring the switching activity on each circuit node. The input vectors are generated from user-specified probability information. This approach uses statistical mean estimation techniques, such as the Monte Carlo procedure, to determine when to terminate the simulation to obtain a certain user-specified accuracy and confidence level.

In the probabilistic approach, a stochastic model describing the input signals along with special library models for gates are used. The signal probabilities of the circuit's primary inputs are propagated into the circuit to promote switching activity at all internal and output nodes.

Other methods of switching activity estimation are performed at a higher level of abstraction. In general, these methods involve the transformation of a hardware description language (HDL) behavioral specifications into *register-transfer-level* (RTL) architectures followed by a simulation process to determine switching activity estimates.

The following sections discuss previous work in the area of switching activity estimation for both statistical, probabilistic and high-level methods. The discussion summarizes techniques which support combinational and sequential circuits, where factors such as delay model, circuit structure, glitch activity, and input pattern dependence are considered.

2.2.1 Statistical Methods

Burch *et al.* [19] proposed a Monte Carlo simulation approach for the estimation of power dissipation in combinational circuits which alleviates the problem of pattern dependence by properly choosing the input vectors. The approach applies randomly generated input patterns to the circuit while monitoring the power dissipation for T clock cycles. Each measurement gives a power sample which is treated as a random variable. As the sample size T approaches infinity, the sample distribution approaches a normal distribution in accordance with the central limit theorem. Sample sizes greater than 30 ensure normal density for most combinational circuits. To establish a stopping criterion, the normality assumption is required, where total power (P_T) is normally distributed for any T . N different simulations of the circuit of length T are performed, producing the sample average (η_T) and the sample standard deviation (s_T), for N different P_T values. According to [20], there is a $(1 - \alpha) * 100\%$ confidence that $|\eta_T - E[P_T]| < t_{\frac{\alpha}{2}} \cdot \frac{s_T}{\sqrt{n}}$, where $t_{\frac{\alpha}{2}}$ is obtained from the t -distribution with $(N - 1)$ degrees of freedom. As a result, for the desired percentage error ε in the power estimate and for a given confidence level $(1 - \alpha)$, the circuit must be simulated for N iterations, where N is given as

$$N = \left(\frac{t_{\frac{\alpha}{2}} \cdot S_T}{\varepsilon \cdot \eta_T} \right)^2. \quad (2.6)$$

The Monte Carlo simulation method may not converge for circuits which do not have normal power distributions. Non-convergence may also occur when T is too small. Moreover, the Monte Carlo method does not support sequential circuits since it must wait for a setup time T_{MAX} , where T_{MAX} is the longest delay along any path. For sequential circuits $T_{MAX} = \infty$. The setup time is the time required before the beginning of a sample interval to guarantee stationarity of the circuit's internal transition process. A process is called stationary if its distribution functions or certain

expected values are invariant with respect to time [21]. When this technique was applied to ISCAS-85 benchmark circuits, the maximum error for a 5% accuracy and 99% confidence level was greater than 5% for only 1% of the cases [19].

Stamoulis proposed a Monte Carlo approach for estimating switching activity in sequential circuits based on the analysis of paths in the state transition graph (STG) [22]. The simulation is performed at the gate-level and obtains an accuracy within 10% of the actual switching probability value for a particular node with a 95% confidence level. Using the circuit described by Figure 2.3 the approach focuses on accurately determining the switching probabilities at the flip-flop output nodes.

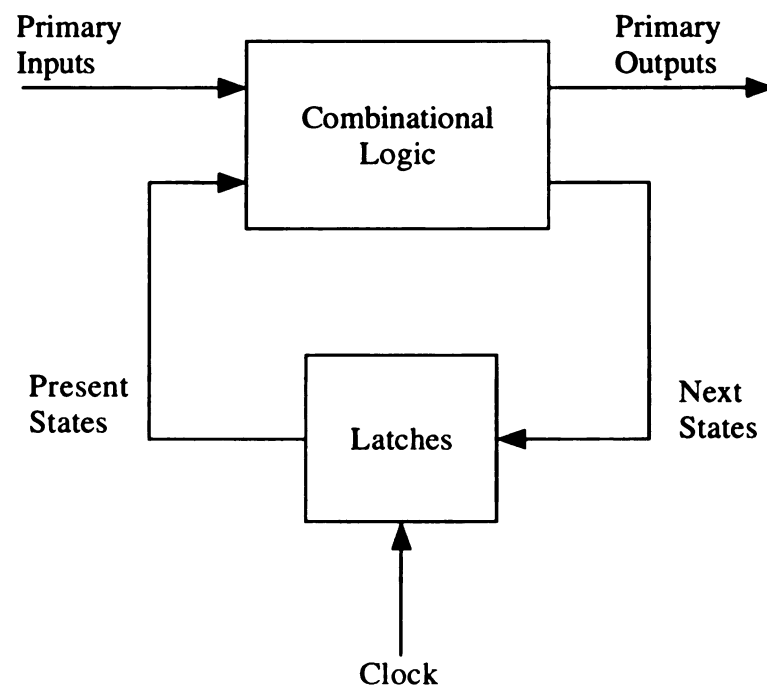


Figure 2.3. Sequential circuit model.

The following assumptions are made: 1) the circuit can be in any of the states with equal probability upon power up, 2) the latch outputs are glitch-free, and 3) all latches reach steady state before the next state enters the combinational logic.

These assumptions allow for the estimation of the power dissipation of the latches separately from the rest of the circuit (*i.e.*, the combinational part), and constrain the number of latch output transitions to at most one per clock cycle. By the definition of initializable circuits, the stochastic processes which describe the state of the flip-flops in the time domain are stationary. States that are N clock ticks apart become independent as $N \rightarrow \infty$. This ensures that the node processes are also mean ergodic. The mean ergodicity property refers to a stationary random process $X(t)$, where the ensemble averages $[\mu_x]_T$ satisfy 1) $\lim_{T \rightarrow \infty} E\{[\mu_x]_T\} = \mu_x$ and 2) $\lim_{T \rightarrow \infty} Var\{[\mu_x]_T\} = 0$ [21]. By using the “path” notion in the STG, long time-consuming simulations are avoided. Pathwise averages are computed using sample simulations with different initial conditions. The estimation process is performed in two steps. First, the pathwise transition probabilities are estimated using Monte Carlo simulation to determine the minimum number of path samples which will contribute to an error ε with confidence $(1 - \alpha) * 100\%$. Second, the average switching probability estimation is computed over all paths, with the switching probability estimate of each path being one measurement. The results of this technique when applied to ISCAS-89 benchmark circuits indicate switching probabilities can be estimated with 5% accuracy at a 95% confidence level [22].

Xakellis *et al.* [23] proposed a switching activity estimation approach which efficiently estimates the transition density at all circuit nodes. This technique improves upon the approach proposed in [19] by eliminating the statistical sampling at single gates, which requires a large number of input patterns for convergence. The convergence problem is overcome by classifying nodes into low-density and regular-density categories and applying absolute error bounds on low-density nodes instead of percentage error bounds. This is done by establishing a threshold to classify low- and regular-density nodes. A node with a transition density value less than the threshold is a low-density node; nodes with a transition density equal to or above the thresh-

old are regular density nodes. The advantage of this method is that it allows the desired accuracy error bounds to be specified by the user. The user also supplies the transition density for every input node, which is the fraction of time the circuit input signal is high. If the circuit input probability is not specified, it is assigned a default value of $\frac{1}{2}$. Next, a random number generator is used to generate the logic input waveforms which drive the simulator. For a given period T , the number of transitions at each node is counted. This process is repeated N times to form an average transition density at each node. This approach, just as in [19], requires a normal distribution of the node densities to establish a stopping criterion with confidence $(1 - \alpha) * 100\%$. Additionally, for synchronous mode simulation, the input signals are assumed to be Markov. The speed of the algorithm is strongly affected by the user-specified threshold which will determine the stopping criterion for the two categories of nodes. Convergence speeds are improved while sacrificing accuracy only at low-density nodes. When tested on a variety of ISCAS-85 benchmark circuits, it was found that over 95% of regular node transition density values have less than 5% error. Low density nodes performed well; over 95% of the low-density node transition values were found to be less than the specified absolute error [23].

Another Monte Carlo simulation technique was proposed by Najm *et al.* [24] in which a method for estimating the switching activity at the latch outputs of sequential circuits is discussed. Similar to [23], this approach makes use of up-front user-specified accuracy information. The algorithm runs until the specified accuracy is achieved. This technique applies a number of randomly generated input vectors to the circuit and collects statistics at the latch output using zero-delay logic simulation. When computing the state line probabilities (signal probability of latch outputs), the Monte Carlo approach is used for estimating N , the number of iterations necessary to achieve the user-specified error-tolerance ϵ and confidence level α [23, 25]. The approach makes two assumptions: 1) the sequential circuit is a non-decomposable FSM, and

2) the state of the machine at time K becomes independent of its initial state as $K \rightarrow \infty$. Assumption 2 implies that the FSM is aperiodic in that it does not cycle through a repetitive pattern of states. The method requires that two simulation runs be performed, with each run starting in a different initial state, say (X_0, X_1) . The signal probabilities $P_K(x_i|X_0)$ and $P_K(x_i|X_1)$ are computed for increasing values of K . Both simulation run estimates should converge to $P(x)$, the signal probability of the input signal. When both measures remain within a window of $\pm\epsilon$ for three consecutive time instants, the node is assumed to have reached convergence. When all nodes have converged, the simulation is complete and the average of the last $P_K(x_i|X_0)$ and $P_K(x_i|X_1)$ value is reported as the signal probability $P(x_i)$, for each x_i . An important feature of this technique is that no assumptions about the FSM behavior (Markov or otherwise) or state line independence is made. Results for this approach on sequential circuits with 1452 flip-flops required 4.6 hours of simulation time (SUN Sparc10) to achieve an error tolerance of 5% with confidence 95% [24].

2.2.2 Probabilistic Methods

Ghosh *et al.* [26] proposed a probabilistic method for the estimation of average switching activity in combinational and sequential circuits. The method considers temporal correlation at the internal nodes and outputs, but requires that the primary inputs be uncorrelated. The method automatically computes switching rates and correlation among flip-flop outputs of FSM's. The behavior of the primary inputs is described in terms of their transition probabilities. The general delay model is used to correctly compute Boolean conditions by compensating for glitch activity. For the signal x , the transition probabilities are p_x^{ij} , which denotes the probability that the signal x will experience a transition from state i to j . The probability of the signal x changing from 1 to 0 is $p_x^{10} = \frac{1}{N}x(k)\overline{x(k+1)}$. The signal probability of node x in terms of transition probabilities is $P_s(x) = P_x^1 = P_x^{10} + P_x^{11}$. These signal prob-

abilities are propagated from the inputs to internal and output nodes by summing the probabilities which disjointly cover the Boolean function in terms of its primary inputs. For example, if the function $g = x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}\bar{z}$, then $P(g = 1)$ or $P_g = p(x)p(\bar{y})p(z) + p(\bar{x})p(y)p(z) + p(\bar{x})p(\bar{y})p(\bar{z})$. The transition probabilities are propagated from the inputs to internal and output nodes by finding the signal probability of the XOR of the Boolean function of each node in two consecutive time frames. For node y , $P_t(y) = P_s(y(t) \oplus y(t+1))$. For a two-input AND gate, where $y = x_1$ AND x_2 , the following equations apply:

$$\begin{aligned}
P_t(y) &= P_s[x_1(t)x_2(t) \oplus x_1(t+1)x_2(t+1)] = P_s[f(y)] \\
f(y) &= \overline{x_1(t+1)}x_1(t)x_2(t) + \\
&\quad \overline{x_2(t+1)}x_1(t)x_2(t) + \overline{x_1(t)}x_1(t+1)x_2(t+1) + \\
&\quad \overline{x_2(t)}x_1(t+1)x_2(t+1)
\end{aligned}$$

and

$$P_s[f(y)] = p_{x_1}^{10} \cdot p_{x_2}^1 + p_{x_2}^{10} \cdot p_{x_1}^1 + p_{x_1}^{01} \cdot p_{x_2}^1 + p_{x_1}^1 \cdot p_{x_2}^{01}.$$

Binary decision diagrams (BDDs) are used for the calculation of the signal probabilities for each of the Boolean functions [27]. This is more simplistic than the enumeration of disjoint covers for each Boolean function. The exact signal probabilities can be computed by performing a linear traversal of the BDD representation of a logic function [28, 29]. To account for gate delays a symbolic simulation method is used to generate a multiple-output function that represents the total switching activity over any possible input vector pair.

The method proposed by Ghosh *et al.* [26] may also be applied to FSM's. This technique accounts for correlation by transforming the conventional FSM structure

F
w
st
T
at
b
n
th
a

de
sto
are
as
no
the
col
pro

(Figure 2.3) to a new structure (Figure 2.4) containing symbolic simulation equations which represent internal and output nodes in the next state logic. For the present state, the gate output's switching activity can only be determined by primary inputs. To compute the transition probabilities, the static probabilities for the present state are used. The next state logic generates Boolean equations which model correlation between the present and next states, thus computing the transition probabilities automatically, which considers the correlation between transitions. This method assumes that present state lines are uncorrelated and that upon power-up the FSM can be in any of the 2^N states, where N is number of flip-flops.

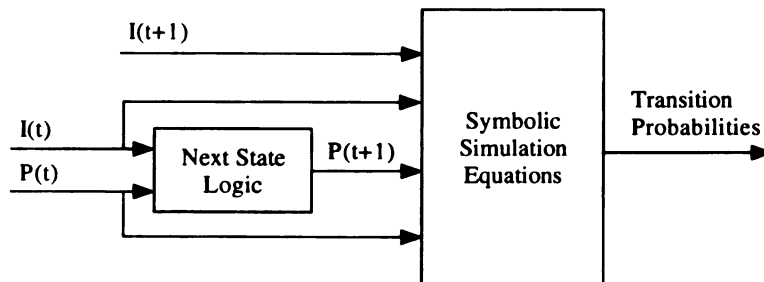


Figure 2.4. Symbolic sequential circuit model.

In [29] a new measure of activity, the transition density is proposed. The transition density may be defined as the average switching rate at a node. This method uses a stochastic model of the logic signals in which the density values of the primary inputs are propagated to internal and output nodes. The transition density at x is defined as $D(x) = \lim_{T \rightarrow \infty} \frac{n_x(T)}{T}$, where $n_x(T)$ is the number of transitions of the signal $x(t)$ at node x in the interval $\left(-\frac{T}{2}, \frac{T}{2}\right]$. The propagation algorithm involves a single pass over the circuit and computes the transition density at all nodes. The primary inputs are considered to be spatially independent and strict sense stationary (SSS). A random process $X(t)$ is called strict sense stationary (SSS) if all of the distribution functions

de-
de-
wh
an
or
an
th
a E
co
co
pe
ab
Ne
on
an
wh
in
cus
dif
ma
ma
circ
nu

describing the process are invariant under a translation of time [21]. The transition density at each node is defined as

$$D(y_i) = \sum_{i=1}^n P \left(\frac{dy_i}{dx_i} \right) \cdot D(x_i), \quad (2.7)$$

where $\frac{dy_i}{dx_i}$ represents the Boolean difference of a function y with respect to signal x_i and $P(\cdot)$ represents the equilibrium probability. Since the input signals are SSS, the output will have the same statistics as its inputs. The zero-delay model is assumed, and the algorithm is limited to combinational circuits only. The algorithm considers the circuit to be an interconnection of logic modules, where each module represents a Boolean function based on the zero-delay model. The drawback to the approach concerns the input independence requirement. If the circuit topology includes re-convergent fanout and feedback, then internal nodes could become correlated, thus possibly destroying the independence property. The propagation of density and probability proceeds on a per module basis from the primary inputs to primary outputs. Next, the equilibrium probability, $P \left(\frac{dy}{dx} \right)$, is evaluated for each node in a module, on a per module basis using the BDD [30]. This approach has a high cost in time and memory usage because the propagation algorithm must interact with the BDD, which may grow exponentially with respect to the number and order of the circuit inputs.

In [31] an improvement in the accuracy of circuit activity measurement is discussed. This technique offers a more efficient mechanism for computing the Boolean difference probabilities at each node of the circuit, which are necessary for the estimation of transition density [24]. In addition, this method allows measurements to be made in a pattern independent manner. The algorithm partitions the combinational circuit, which is modeled as a directed acyclic graph, with the goal of maximizing the number of correlated nodes within each partition.

me
fa
ne
the
pr
the
dir
fu
pr
ne
ate
Th
as
als
For
siz
par
pri
bel
the
res
for
Bo
the
tha

The partitioning aspect of the algorithm prevents overestimation and underestimation inaccuracies in the density simulation which may occur from the correlation of fanout inputs in the circuit. To compute the Boolean difference probabilities for each node in a given circuit, this method constructs an ordered BDD (OBDD), by using the APPLY and RESTRICT procedures at each node [27]. The APPLY procedure provides the basic method for creating the representation of a function according to the operators in a Boolean expression or logic gate network. The RESTRICT procedure transforms the graph representing a function f into a function representing the function $f|_{x_i=b}$ for specified values of i and b . The efficiency of the computation for producing maximally reduced OBDDs is improved according to the gate operation needed. The development of a new operation, the DIFFERENCE operation, generates Boolean difference functions by applying algebraic operations to other functions. The DIFFERENCE operation is implemented as a sequence of APPLY operations.

The partitioning aspect of the technique allows for an improvement in accuracy as the size of the partition grows. However, as the size of the OBDD grows, the algorithm slows down. The OBDD can grow exponentially with the number of inputs. For this reason the number of inputs is chosen as a parameter to determine partition size. Next, a breadth-first search is applied at every primary input such that each partition consists of a single output with k variables, where the variables may be primary outputs or inputs. The circuit partitioning procedure stops when each node belongs to a partition which contains k or fewer input variables. The purpose of the circuit partitioning is to keep each partition small enough to achieve accurate results. The partitions are then placed in a partition set, stored in the order of their formation. The transition density is computed for each partition by computing the Boolean difference probability. The drawback to the technique is the rapid growth of the OBDD, which leads to memory overflow problems. Experimental results report that for 50 combinational circuits, with up to 20 inputs, this technique required 40%

|

v
s
c

T
ite
the
the

$$ns_1 = f_1(i_1..i_M, P_{s_1}..P_{s_N}) \quad (2.8)$$

$$ns_2 = f_2(i_1..i_M, P_{s_1}..P_{s_N})$$

$$\vdots$$

$$ns_N = f_N(i_1..i_M, P_{s_1}..P_{s_N})$$

and

$$P(ns_1) = P(f_1(i_1..i_M, P_{s_1}..P_{s_N})) \quad (2.9)$$

$$P(ns_2) = P(f_2(i_1..i_M, P_{s_1}..P_{s_N}))$$

$$\vdots$$

$$P(ns_N) = P(f_N(i_1..i_M, P_{s_1}..P_{s_N}))$$

where $P(ns_1) = P(ns_1 = 1)$ and $P(p_{s_i}) = P(ns_i) = p_i$ for $1 < i < N$. The present state line probabilities are applied to a nonlinear function g and a nonlinear system of equations is given by

$$y_1 = p_1 - g_1(p_1, p_2, \dots, p_N) = 0 \quad (2.10)$$

$$y_2 = p_2 - g_2(p_1, p_2, \dots, p_N) = 0$$

$$\vdots$$

$$y_N = p_N - g_N(p_1, p_2, \dots, p_N) = 0.$$

The nonlinear system of equations may be denoted as $Y(P) = 0$ or $P = G(P)$. An iterative solution can be obtained by the use of the Newton-Raphson method for the system $Y(P) = 0$ [33]. Given the nonlinear system of equations, $P = G(P)$, the Picard-Peano method can be applied to determine a solution [32]. Since the

nonlinear solutions do not capture correlation between state line probabilities, minor inaccuracies are incurred.

To improve accuracy, an unrolling of the next state logic network is performed (Figure 2.5) [32]. The signal probabilities are approximated by unrolling the next state logic k times, where k is a user-specified parameter. Usually an unrolling of the next state logic improves the accuracy of the results. For $k = 3$ the average error was reported to be only 1.5% [32]. As k increases the time consumption increases, along with a decrease in average error.

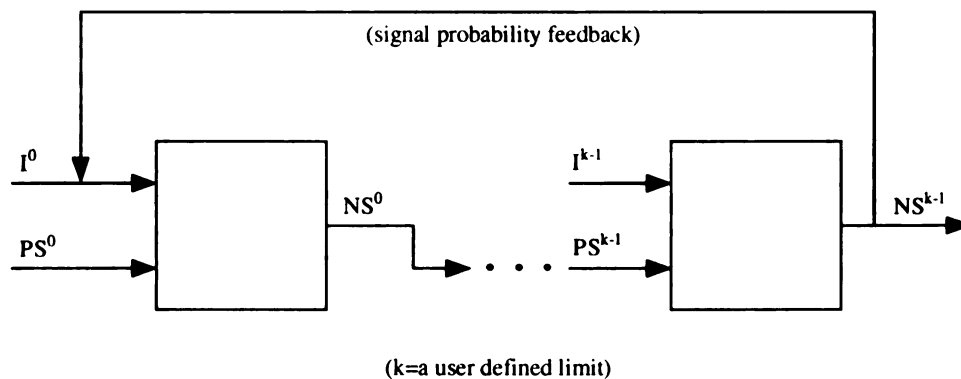


Figure 2.5. k -unrolling of next state logic.

The correlation accuracy improvement (m -expanded network) involves modifying the next state logic by selecting m -tuples of the present state lines, separated by one clock cycle, and computing probabilities for each combination of the m -tuples pairs (Figure 2.6) [33]. These probability values are fed into the combinational logic block. Using the ISCAS-85 benchmark circuits, the m -expanded network method for an accuracy improvement with $m = 2$ obtained an average error less than 4.1%, and for $m = 4$ the reported average error was less than 3.6% [33].

Cheng *et al.* [34] proposed a method to increase the speed of estimation of power dissipation by applying topological analysis to the circuit using the concept of super-

En
w
u
t
re
E
W
ni
mu

2.2

A p
al.
(PD
eter
swi
its a
regis

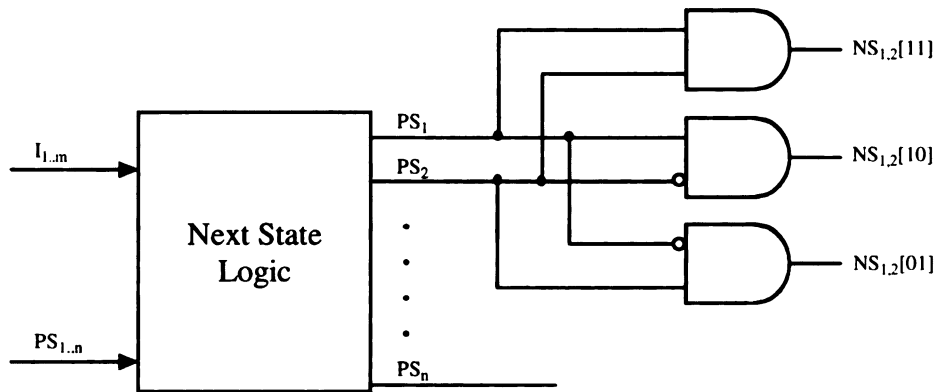


Figure 2.6. m -expanded network with $m=2$.

gates. The supergate of a node in a combinational circuit is the minimal subcircuit whose inputs are logically independent. The algorithm transforms a circuit to an undirected graph and solves the problem of finding supergates by finding articulation points in an undirected graph. An articulation point is defined as a node whose removal disconnects the graph. The supergate concept allows for the partitioning of BDDs, where each partition boundary is defined by its logically independent inputs. When the supergate concept is applied to most BDD-based power estimation techniques, for select benchmark circuits, cpu-time and memory usage are reduced by as much as 86.2 and 94.0 percent, respectively [34].

2.2.3 High-Level Methods

A profile driven approach to low power behavioral synthesis is presented in Katkoori *et al.* [35, 18]. The method presented is known as the Profile Driven Synthesis System (PDSS). Given a behavioral design specification, a set of input vectors, a parameterized library module, and user-specified constraints such as area and speed, the switching activity is estimated for the design. Each library module is characterized by its average switching activity per input vector and consists of units such as adders, registers, muxes, *etc.* The behavioral specification can be written in a hardware

a
o
o
t
f
2
of
a
of
ar
ul
of
ti
sin
of
ph
pa
da
the
t
esti
acti

description language such as VHDL. The synthesized design consists of interacting datapath and controller components. The datapath is composed of modules from the module library and the controller is a finite state machine (FSM) implemented as a PLA/microprogram. The PDSS accepts the behavioral VHDL specification as input and extracts a data flow graph (DFG). The DFG is passed through a profiler, where operations and carriers (edges of the DFG) are collected. In more detail, carriers are defined as data flow edges that cross a control step boundary which denotes a value that needs to be stored in a register. The goal of the profiling phase is to gather the following: 1) the number of times a node is executed for a given profiling stimuli, 2) the number of times each edge is traversed during execution, and 3) the number of times the edge value changes. Upon the completion of the profiling phase, four additional phases are entered: 1) scheduling and performance estimation, 2) register optimization, 3) interconnect optimization, and 4) controller generation.

During the scheduling and performance estimation phase, operations in the DFG are assigned to control steps and various operation nodes are bound to specific modules selected from the module library. The schedule is acceptable when the estimates of the area and clock period satisfy the user-specified constraints. The register optimization phase groups carriers such that no two carriers in the same group are simultaneously active. The interconnect optimization phase involves the assignment of interconnect paths to each value transfer in the DFG. The controller generation phase produces a finite state machine description. The FSM accepts as input datapath status flags and produces control signals which enable register transfers in the datapath. DFG edges that cross control step boundaries denote state transitions in the FSM. Each control step corresponds to at least one state in the FSM.

Using the data collected in the profiling phase, the PDSS system determines an estimate for aggregate switching activity (ASA). The ASA is the sum of the switching activities in the datapath and the controller. The switching activity associated with

f

2

1

s

B

ca

di

to

li

ti

lin

pr

co

Ka

ties

π c

ma

ste

con

it b

exa

to a

rep

Fig

fewer nodes than the straight-forward way of computing Boolean difference equations.

In two simultaneous publications by Tsui *et al.* [32] and Monteiro *et al.* [33], exact and approximate methods for estimating switching activity in FSM's are discussed. These techniques also closely model the work presented in [26] and share the same sequential circuit and correlation decomposition structures (Figures 2.3 and 2.4). Both techniques are almost identical with a few minor differences. To begin with, the calculation of switching activity for sequential circuits must consider the probability of the circuit being in any of its 2^N states, where N is the number of flip-flops. Further, to arrive at an accurate estimate of switching activity, the values for the present state line probabilities, steady-state probabilities, and signal probabilities must be known.

One method of exactly computing the steady-state probabilities is to find a solution to the 2^N linear system of Chapman-Kolmogorov equations. Next, present state line probabilities are computed from the steady-state probabilities and the signal probability is computed by generating the Boolean function's disjoint covering. The computational cost of computing the steady-state probabilities from the Chapman-Kolmogorov system of equations is very high, but produces the exact state probabilities. The exact solution involves the system $\pi = P\pi$ or $(I - P)\pi = 0$, where the vector π contains the steady-state probabilities. P is referred to as the transition probability matrix, which is derived from the state transition graph (STG). To determine π , the steady-state probability vector, the null-space of the system $(I - P)\pi = 0$ must be computed. As the number of states grows exponentially with the number of flip-flops it becomes impossible to build an STG for large sequential machines and thus the exact method cannot be applied.

A more time-efficient but less accurate approach searches for an iterative solution to a nonlinear system of N equations [32, 33]. This system of nonlinear equations represents the next state logic preceding the symbolic combinational logic block in Figure 2.4. The next and present state line probabilities have the following form:

at

wi

sta

of

The

ite

the

the

$$ns_1 = f_1(i_1..i_M, P_{s_1}..P_{s_N}) \quad (2.8)$$

$$ns_2 = f_2(i_1..i_M, P_{s_1}..P_{s_N})$$

⋮

$$ns_N = f_N(i_1..i_M, P_{s_1}..P_{s_N})$$

and

$$P(ns_1) = P(f_1(i_1..i_M, P_{s_1}..P_{s_N})) \quad (2.9)$$

$$P(ns_2) = P(f_2(i_1..i_M, P_{s_1}..P_{s_N}))$$

⋮

$$P(ns_N) = P(f_N(i_1..i_M, P_{s_1}..P_{s_N}))$$

where $P(ns_1) = P(ns_1 = 1)$ and $P(ps_i) = P(ns_i) = p_i$ for $1 < i < N$. The present state line probabilities are applied to a nonlinear function g and a nonlinear system of equations is given by

$$y_1 = p_1 - g_1(p_1, p_2, \dots, p_N) = 0 \quad (2.10)$$

$$y_2 = p_2 - g_2(p_1, p_2, \dots, p_N) = 0$$

⋮

$$y_N = p_N - g_N(p_1, p_2, \dots, p_N) = 0.$$

The nonlinear system of equations may be denoted as $Y(P) = 0$ or $P = G(P)$. An iterative solution can be obtained by the use of the Newton-Raphson method for the system $Y(P) = 0$ [33]. Given the nonlinear system of equations, $P = G(P)$, the Picard-Peano method can be applied to determine a solution [32]. Since the

nonlinear solutions do not capture correlation between state line probabilities, minor inaccuracies are incurred.

To improve accuracy, an unrolling of the next state logic network is performed (Figure 2.5) [32]. The signal probabilities are approximated by unrolling the next state logic k times, where k is a user-specified parameter. Usually an unrolling of the next state logic improves the accuracy of the results. For $k = 3$ the average error was reported to be only 1.5% [32]. As k increases the time consumption increases, along with a decrease in average error.

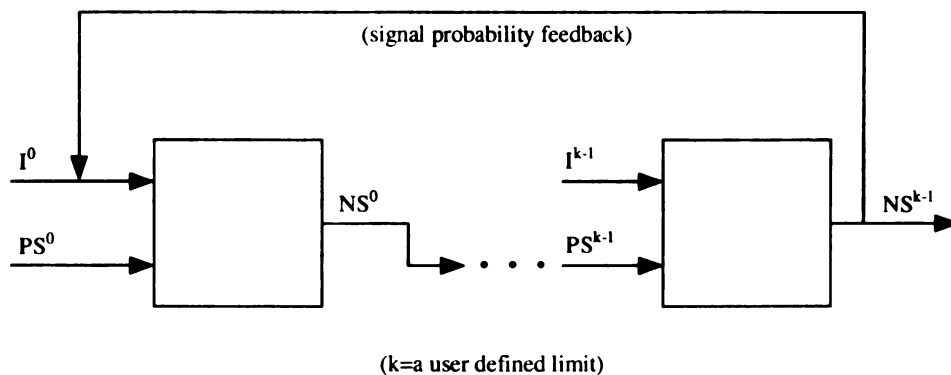


Figure 2.5. k -unrolling of next state logic.

The correlation accuracy improvement (m -expanded network) involves modifying the next state logic by selecting m -tuples of the present state lines, separated by one clock cycle, and computing probabilities for each combination of the m -tuples pairs (Figure 2.6) [33]. These probability values are fed into the combinational logic block. Using the ISCAS-85 benchmark circuits, the m -expanded network method for an accuracy improvement with $m = 2$ obtained an average error less than 4.1%, and for $m = 4$ the reported average error was less than 3.6% [33].

Cheng *et al.* [34] proposed a method to increase the speed of estimation of power dissipation by applying topological analysis to the circuit using the concept of super-

So
w
un
ti
re
B
W
ni
m
2.2
A
al
(P
ete
sw
its
reg

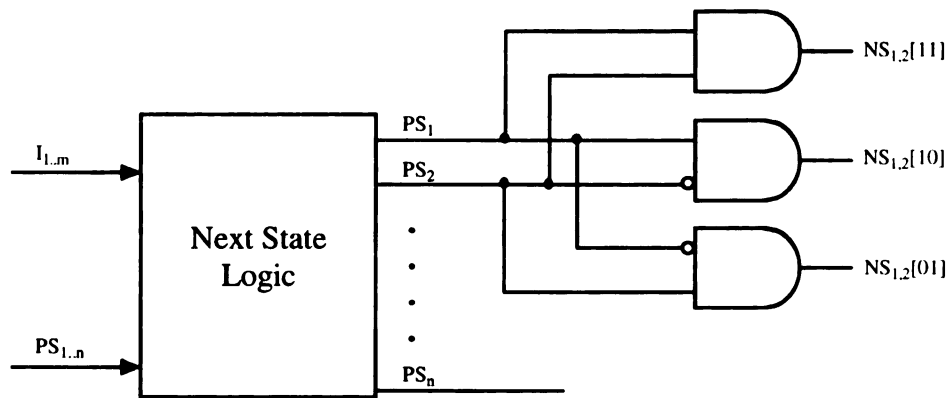


Figure 2.6. m -expanded network with $m=2$.

gates. The supergate of a node in a combinational circuit is the minimal subcircuit whose inputs are logically independent. The algorithm transforms a circuit to an undirected graph and solves the problem of finding supergates by finding articulation points in an undirected graph. An articulation point is defined as a node whose removal disconnects the graph. The supergate concept allows for the partitioning of BDDs, where each partition boundary is defined by its logically independent inputs. When the supergate concept is applied to most BDD-based power estimation techniques, for select benchmark circuits, cpu-time and memory usage are reduced by as much as 86.2 and 94.0 percent, respectively [34].

2.2.3 High-Level Methods

A profile driven approach to low power behavioral synthesis is presented in Katkooi *et al.* [35, 18]. The method presented is known as the Profile Driven Synthesis System (PDSS). Given a behavioral design specification, a set of input vectors, a parameterized library module, and user-specified constraints such as area and speed, the switching activity is estimated for the design. Each library module is characterized by its average switching activity per input vector and consists of units such as adders, registers, muxes, *etc.* The behavioral specification can be written in a hardware

6
d
n
E
a
o
d
ti
fo
2
of
a
o
ar
ul
of
tin
sin
of
ph
pa
da
th
est
act

description language such as VHDL. The synthesized design consists of interacting datapath and controller components. The datapath is composed of modules from the module library and the controller is a finite state machine (FSM) implemented as a PLA/microprogram. The PDSS accepts the behavioral VHDL specification as input and extracts a data flow graph (DFG). The DFG is passed through a profiler, where operations and carriers (edges of the DFG) are collected. In more detail, carriers are defined as data flow edges that cross a control step boundary which denotes a value that needs to be stored in a register. The goal of the profiling phase is to gather the following: 1) the number of times a node is executed for a given profiling stimuli, 2) the number of times each edge is traversed during execution, and 3) the number of times the edge value changes. Upon the completion of the profiling phase, four additional phases are entered: 1) scheduling and performance estimation, 2) register optimization, 3) interconnect optimization, and 4) controller generation.

During the scheduling and performance estimation phase, operations in the DFG are assigned to control steps and various operation nodes are bound to specific modules selected from the module library. The schedule is acceptable when the estimates of the area and clock period satisfy the user-specified constraints. The register optimization phase groups carriers such that no two carriers in the same group are simultaneously active. The interconnect optimization phase involves the assignment of interconnect paths to each value transfer in the DFG. The controller generation phase produces a finite state machine description. The FSM accepts as input datapath status flags and produces control signals which enable register transfers in the datapath. DFG edges that cross control step boundaries denote state transitions in the FSM. Each control step corresponds to at least one state in the FSM.

Using the data collected in the profiling phase, the PDSS system determines an estimate for aggregate switching activity (ASA). The ASA is the sum of the switching activities in the datapath and the controller. The switching activity associated with

the datapath is computed by summing the switching activities determined for the following: combinational operators, registers, and interconnection elements such as buses, multiplexers, and wires. The switching activity contributed by the controller is determined by analyzing the PLA structure used to implement the controller's FSM. The activity concerning the PLA structure is the sum of the PLA's input and output plane switching activities. The experimental results of the method indicate that the estimated switching activity deviates by less than 10% [35].

Landman *et al.* [36] present techniques for accurately estimating power consumption based on a high-level description of the system architecture. This approach, based on stochastic modeling of bus statistics, achieves the accuracy associated with gate-level estimation tools. Algorithmic and architectural estimation techniques based on high-level statistics such as mean, variance, and autocorrelation are developed using concepts from the gate-level techniques. While gate-level techniques focus on power consumed by individual Boolean logic gates as a function of their input probabilities; this method considers module (*adder, register, multiplier*) power consumption in regards to input-word statistics. For a variety of input distributions, the techniques perform very well on real-world signals such as speech, music and image, typically found in digital signal processing (DSP) applications. The results obtained from these techniques exhibit an estimation accuracy within 9.4% of the gate-level simulations [36].

In [37, 38] Nemani *et al.* and Najm, in similar publications, presented a digital IC power estimation technique that operates at the *register-transfer-level* (RTL). The estimator is based on using entropy as a measure of the average activity to be expected in the final circuit-level implementation. Entropy is a characterization of a random variable or random process. If x is a random Boolean variable with signal probability p , then the entropy of x is defined as: $H(x) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}$. The entropy can be expressed for discrete systems in terms of inputs and outputs. The high-level

power estimation methodology for a combinational circuit block that is part of a synchronous sequential circuit involves the following steps:

- 1) Run a structural RTL simulation of the sequential circuit to measure the input/output entropies of the combinational block.
- 2) From the input/output entropies, estimate average node density, circuit area, and average power.
- 3) Combine with latch and clock power to compute total average power.

The area and average power estimates are computed via input/output entropy estimates where $A \propto \frac{2^n}{n} H(Y)$, $P_{avg} \propto A \times H$, such that $H(Y)$ is the output entropy and H is the average input entropy. When this method was tested against a zero-delay model for 56 different ISCAS-85 benchmark circuits with sizes ranging from 100 to 22,000 gates, the error was reported to be less than 9% with a 90% confidence level [37].

In summary, the estimates generated by current high-level power and switching activity computation methods experience between 9% to 12% error on average. For some circuits containing large amounts of reconvergent fanout the error is as high as 80%. The error experienced by current high-level power and switching activity estimation techniques is too high. This dissertation presents a new methodology for improved switching activity estimation of behavioral-level designs described in VHDL.

0

R

S

Th

ch

ter

of

pr

in

3.

Th

{0.

two

185

bin

call

(Ta

CHAPTER 3

Behavioral Representations of Switching Functions

The fundamental building blocks of digital systems are switching functions. This chapter specifically addresses the use of Boolean switching functions. A review of terminology and notation used to describe the functional and mathematical models of Boolean switching functions when applied to behavioral design of digital systems is provided. Additionally, a discussion of various behavioral representations for switching functions is provided as well.

3.1 Switching Algebra Background

The mathematics which define rules and operations for processing the binary set $\{0, 1\}$ in a combinational digital system is called *Switching Algebra*, also known as two-valued Boolean algebra, developed by English mathematician George Boole in 1854. The most primitive operations applied to variables (*i.e.*, $\forall x_i \in \{0, 1\}$) of the binary set are “.”, “+”, and “–”, which denote switching algebra operations, also called logic operations. A switching algebra must satisfy Huntington’s postulates (Table 3.1), which define the basis of switching algebra theory [39].

T
in
e
e
in
v
c
b

T
th
op
is
sh

<i>Property</i>	<i>Meaning</i>
<i>Closure</i>	$x, y \in \{0, 1\} \rightarrow (x + y) \in \{0, 1\}$ $x, y \in \{0, 1\} \rightarrow (x \cdot y) \in \{0, 1\}$
<i>Identity</i>	$x + 0 = x$ $x \cdot 0 = 0$ $x + 1 = 1$ $x \cdot 1 = x$
<i>Commutative</i>	$x + y = y + x$ $x \cdot y = y \cdot x$
<i>Distributive</i>	$x + (y \cdot z) = (x + y)(x + z)$ $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
<i>Complement</i>	$x + \bar{x} = 1$ $x \cdot \bar{x} = 0$

Table 3.1. Huntington's postulates.

3.2 Truth Tables

The definition of the three logic operators (“+”, “·”, “−”) can be deduced from Huntington's postulates using truth tables. A *truth table* is a mechanism for systematically enumerating the output or result of a logic operation for every possible Boolean input combination. The truth table can be extended to enumerate the outputs for switching functions consisting of sets of logic operations and Boolean operands. A double vertical bar is used to separate truth table inputs from truth table outputs. The commonly used truth table convention places inputs on the left of the double vertical bar, with outputs placed to the right.

3.2.1 Logic Operations

The “−” logic operation is called the *complement*, *NOT*, or *invert* operation, of which the term NOT is the most commonly used. The NOT operation is a unary logic operation, meaning that it operates on a single operand. When the NOT operation is applied to the Boolean operand x , the result is \bar{x} . The truth table in Table 3.2(a) shows that the NOT operation changes 0 to 1 and changes 1 to 0.

x	\bar{x}
0	1
1	0

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

(a) NOT (b) AND (c) OR

Table 3.2. Truth tables for NOT, AND, OR logic operations.

The “ \cdot ” logic operation is called the *AND* or *conjunction* operation. The term AND is most commonly used. It is a binary operator, meaning that it takes two operands and generates 2^2 input combinations. The result after applying the AND operation to binary operands x and y is 1, only when $x = y = 1$, otherwise the result is 0. The truth table in Table 3.2(b) depicts the AND logic operation.

The “ $+$ ” logic operation is called the *OR* or *disjunction* operation. The term OR is most commonly used and is classified as a binary operator. The result after applying the OR operation to binary operands x and y is 0, only when $x = y = 0$, otherwise the result is 1. The truth table in Table 3.2(c) depicts the OR logic operation.

The truth table technique is an implementation-independent method of representing switching functions. Truth tables are considered behavioral representations because no reference to technology, structure, or implementation is made. The truth table completely specifies a switching function’s output for all possible (2^N) input combinations, where N is the number of Boolean inputs. Consider the 3-input switching function which represents the carry-out bit for a full-adder (FA). The FA carry-out bit input/output behavior is specified for all 2^3 inputs sequences (Table 3.3).

3

7

0

t:

to

f:

r

va

fo

3.

Th

3.3

in

su

a	b	c_{in}	c_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.3. Truth table for full-adder carry-out bit.

3.2.2 Output Enumeration

The outputs of switching functions are easily enumerated with truth tables. Each row of a fully specified truth table is referred to as a *term* or *product*. The term represents the input combination that determines the output of a switching function. Each term or product is logically an AND-product of input variable literals. A switching function consisting of N input variables has 2^N terms. The enumeration of terms utilizes the base-2 number system because switching algebra is defined against a two-valued domain of $\{0, 1\}$. The expression for representing a term's decimal equivalent for an ordered n -bit sequence $(x_{n-1}, x_{n-2}, \dots, x_1, x_0)$ is

$$X = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0 = \sum_{i=0}^{n-1} x_i2^i \quad (3.1)$$

3.3 Boolean Expressions

The full-adder's carry-out bit switching function described by the truth table in Table 3.3 has 2^3 terms, numbered from 0 to 7 using Equation 3.1. The terms which result in switching function's output of logic 1 are known as *minterms*. The carry-out bit switching function has four minterms and can be expressed in the following forms:

$$f(a, b, c_{in}) = 011 + 101 + 110 + 111 \quad (3.2)$$

$$= \bar{a}bc_{in} + a\bar{b}c_{in} + ab\bar{c}_{in} + abc_{in} \quad (3.3)$$

$$= m_3 + m_5 + m_6 + m_7 \quad (3.4)$$

$$= \sum m(3, 5, 6, 7) \quad (3.5)$$

Each of the forms represent what is known as a Boolean expression. The Boolean expression is an n -variable function of Boolean inputs whose output is Boolean, or $f(x) : \{0, 1\}^n \rightarrow \{0, 1\}$. Equations 3.2 and 3.3 represent the minterms of the carry-out bit switching function in binary and literal form, respectively. Each switching variable is expressed as un/complemented (1/0) for the specified input combination. Equation 3.4 uses a shorthand notation for minterms, m_x . The subscript x denotes the associated row of the truth table or binary value of the input combination. Equation 3.5 is the most commonly used form for representing minterm lists, it is a more compact representation of the form in Equation 3.4. The switching function representations shown in Equations 3.2, 3.4, and 3.5 are known as *canonical minterm* or *sum-of-products* (SOP) expressions. After reducing Equation 3.3, using Huntington's postulates, the resulting Boolean expression is

$$f(a, b, c_{in}) = ab + c_{in}(a + b). \quad (3.6)$$

The reduced Boolean expression (Equation 3.6) is in SOP form, but it is not a canonical minterm expression because the product terms are not minterms. The canonical minterm expressions, as well as Boolean expressions, are implementation-independent methods for representing the input/output behavior of switching functions. Boolean expressions do not consider technology, circuit structure, or imple-

mentation of a digital design. The advantage of the canonical minterm expression representation over the truth table representation is size. Canonical minterm expressions are more compact than truth tables. Truth tables represent all 2^N input combinations, whereas canonical minterm expression representations list the input combinations for which the switching function's output is logic 1 only.

3.4 Binary Decision Diagrams

Probability-based power analysis tools depend heavily on Binary Decision Diagrams (BDDs) to determine signal activity. Historic uses of BDDs have been in the digital circuit design areas of synthesis, verification, and testing. The BDD is not a new concept. As early as 1959, Lee [40] introduced the concept of Binary Decision Programs and a set of rules to transform these programs into switching circuits. Later, in 1978, Akers [41] revisited the concept of BDDs by using the diagram as a means to define, analyze, and test large digital functions from an implementation-free perspective. It was in 1986 that Bryant [27] demonstrated the advantages of BDDs as a canonical representation. Bryant demonstrated that BDDs have two very useful properties. First, BDDs are canonical: given two circuits, they are equivalent if their BDDs are identical. Second, BDDs are effective at representing combinatorially large sets, which is useful in FSM equivalence checking and logic minimization.

BDDs represent a switching function as a directed acyclic graph (DAG). A graph consists of an interconnection of nodes (vertices) and edges (arcs). There are two node types: decision nodes or terminal nodes. *Terminal nodes* are characterized by not having outgoing edges which lead to children nodes and contain fixed values which possibly correspond to the output of a function. *Decision nodes* are characterized by having outgoing edges which lead to other decision nodes and terminal nodes. The decision node is labeled with a variable identifier and has one outgoing edge for each

value this variable can assume.

Since Boolean decisions are being made, the decision node variable identifiers can only assume the values of 0 and 1. The terminal node values will be fixed at either 0 or 1, and the possibilities for edges will be either the 0-edge or 1-edge. The 0-edge will be chosen when the decision node variable assumes the value 0 and the 1-edge is chosen when the decision node variable takes on the value 1. Figure 3.1 is a BDD representing the full-adder (FA) carry-out bit switching function described by Equation 3.6 and Table 3.3. Typically in BDD graphs dotted arcs represent the 0-edges while the solid arcs represent 1-edges.

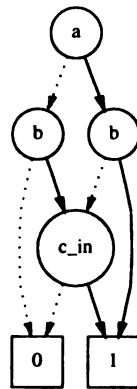


Figure 3.1. BDD of full-adder carry-out bit.

The BDD size is directly related to the number of nodes in the BDD's graph, which is controlled by the number of input variables and their ordering. One such BDD type, the Ordered Binary Decision Diagram (OBDD), addresses the BDD size issue by considering input variable ordering. The ordering of input variables determines the level at which each input will appear in the BDD's graph. In the OBDD the ordering will remain the same for each path taken from the root (lowest order) node to a terminal node. Different input variable orderings lead to different BDDs, with

each potentially having a different size.

Consider the switching function given by Equation 3.7. An input variable ordering of $a < d < b < c$ leads to the BDD displayed in Figure 3.2, while an input variable ordering of $b < c < a < d$ leads to the BDD representation displayed in Figure 3.3.

$$f(a, b, c, d) = abc + \bar{b}d + \bar{c}d \quad (3.7)$$

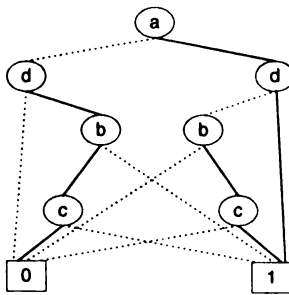


Figure 3.2. BDD with ordering 1.

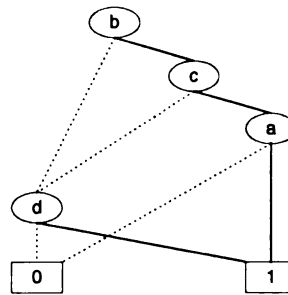


Figure 3.3. BDD with ordering 2.

Clearly the second ordering provides the more compact BDD representation; it has a smaller node count. Hence, a good input variable ordering will yield a more compact BDD representation with reasonable memory usage [42]. A modification to the OBDD is the Reduced-Ordered BDD (ROBDD). An initial ordering is given in the ROBDD, and the iterative identification and removal of isomorphic subgraphs and redundant nodes takes place [39]. The removal results in a BDD which is minimal for the given input variable ordering and canonical in form.

Once such ROBDD implementation was developed by Brace *et al.* [30]. This ROBDD implementation made improvements in the *if-then-else* (ITE) operator, hashing technique, and memory garbage collection. The results reported that an amortized memory cost of 22 bytes per node was achieved. Additionally, it was reported that the improvements yielded a faster, more memory-efficient ROBDD implementation

than the original implementation presented in [27].

Shen *et al.* [43], proposed a data structure called the Free Boolean Diagram (FBD), which improved the ROBDD representation by trading off canonicity. One distinction between the ROBDD and the FBD is that the FBD allows different input variable orderings along different paths from the root node to a terminal node. Additionally, the nodes in the graph of the FBD may be of type *function* (XOR or AND nodes), which is further discussed in [43]. It was reported that the FBD implementation resulted in an amortized memory cost of 32 bytes per node and for certain cases the FBD size was significantly reduced [43]. The FBD implementation is an improvement over the ROBDD implementation for certain circuits because its total memory usage for behavioral representation is less.

3.5 Behavioral VHDL Specifications

VHDL, short for VHSIC (Very High Speed IC) Hardware Description Language, is a formal notation for hardware description, standardized by the IEEE in 1987, that allows for, among other descriptions, the behavioral-level design of a digital system [44]. The VHDL behavioral specification supports the modeling of digital hardware using sequential statements similar to programming languages such as Ada, C, or Pascal. VHDL, just like the previously mentioned programming languages, contains loop, *if-then-else*, and assignment constructs.

A VHDL model is comprised of *entities* and *architectures*. An entity defines the interface between a system and its environment, such as signals that flow into and out of the system or component. The architecture describes the functional nature of the digital system; it defines how the outputs respond to the inputs.

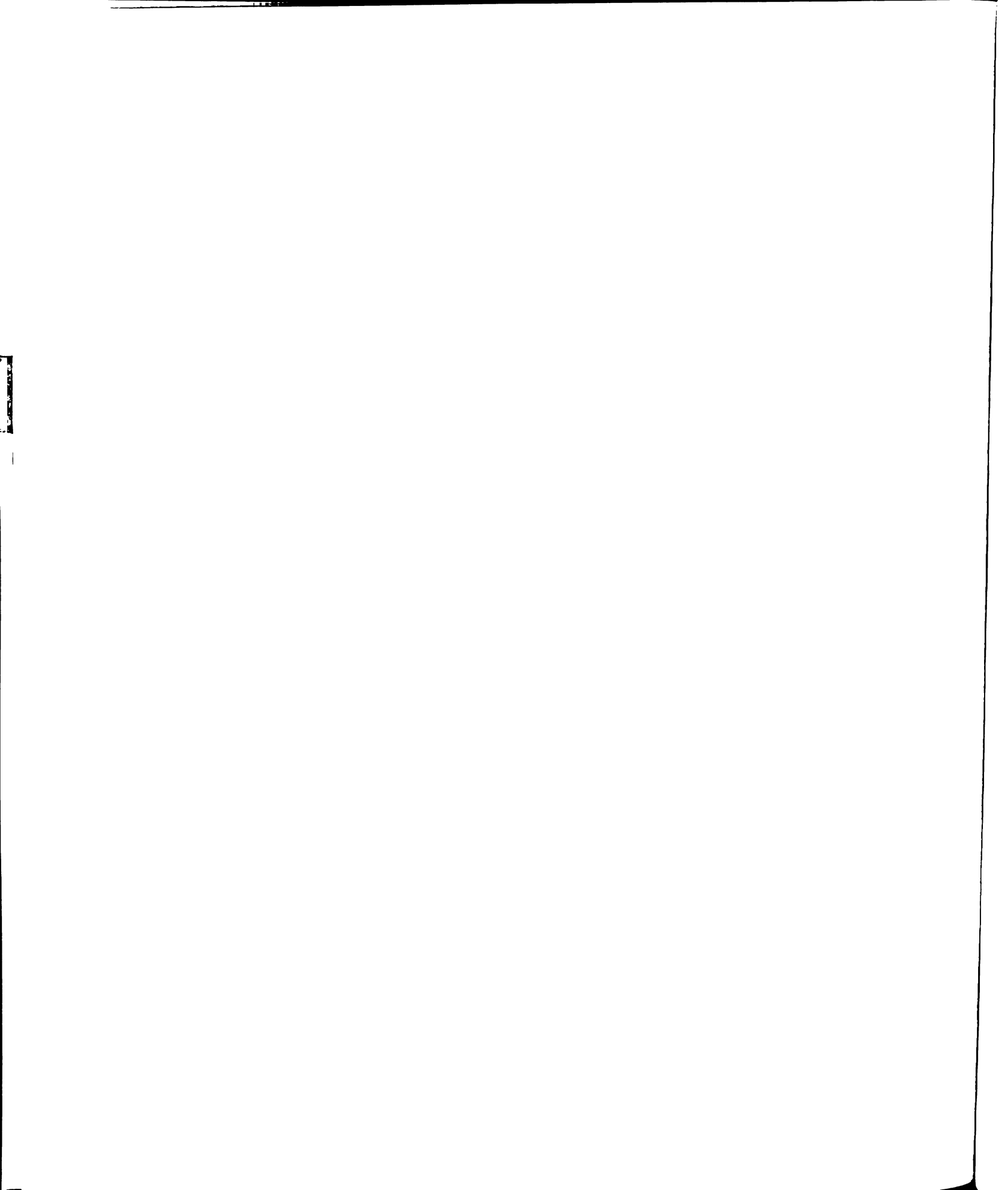
A behavioral VHDL specification describing the carry-out bit of a full-adder (FA) is given in Figure 3.4, where the architecture closely resembles the Boolean

function defined by Equation 3.6. Behavioral VHDL specifications are high-level implementation-independent representations which describe the functional nature of a digital system without any reference to circuit structure or technology.

```
ENTITY CARRY is
  port (A, B, Cin : IN BIT;
        Cout      : OUT BIT);
end CARRY;

ARCHITECTURE Behavioral of CARRY is
BEGIN -- Behavioral FA carry-out bit
  Cout <= ((A and B) or (Cin and (A or B))) after 10ns;
END Behavioral;
```

Figure 3.4. Behavioral VHDL model of FA carry-out bit.



CHAPTER 4

Structural Representations of Switching Functions

The previously discussed behavioral representations describe only a circuit's functionality in terms of truth tables, Boolean expressions, or BDDs. When detailed information, such as *timing*, *speed*, or *area* are needed, these representations are of little use. To effectively describe the input/output relationships of a circuit, a lower level of design abstraction is necessary. The logic- and circuit- design levels of abstraction within the EDA process (Figure 1.1) provide information which enable designers to obtain or calculate critical timing, speed, or area information. The logic- and circuit- design levels of abstraction yield *structural* representations. Given a logic- or circuit- level description, structural-level representations effectively describe the design's input/output relationships, as well as the internal connectivity of circuit elements and logic gates. This chapter deals with logic-level structural representations, including the modeling of switching functions in terms of *schematics*, *netlists*, *structural VHDL specifications*, and the *Connective Binary Decision Diagram (CBDD)*.

4.1 Schematics

Schematics or logic diagrams are a visual means of representing the structure of a digital circuit design. *Schematics* illustrate connective relationships between a circuit's primary inputs/outputs, internal signals, and logic elements. Schematics display the logic elements and the wires used to interconnect these elements. The most basic logic elements used to perform the operations necessary to implement switching functions are the *NOT*-, *AND*-, and *OR*- gates. The schematics of the basic logic gates are illustrated in Figures 4.1, and they implement the behavior given in the truth tables of Table 3.2.

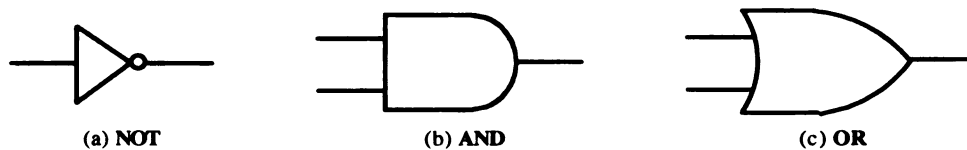


Figure 4.1. Schematics of basic logic gates.

The interconnection and recombination of these basic logic gates support the design of more complex switching functions such as adders, ALUs, muxes, *etc.* By interconnecting an arrangement of *OR*- and *AND*- gates, a structural representation for the carry-out bit of the full-adder can be realized, Figure 4.2.

Vendors such as ViewLogic, Xilinx, and Mentor Graphics develop *schematic entry* software systems which support the design of digital hardware. Schematic entry software systems provide a pre-characterized library of logic elements with the ability to place and wire these components to a design's connective specification. A limitation of the schematic design approach is its difficulty supporting low-level and detailed views of large and complex designs. Large and complex designs are limited to system-level or hierarchical views.

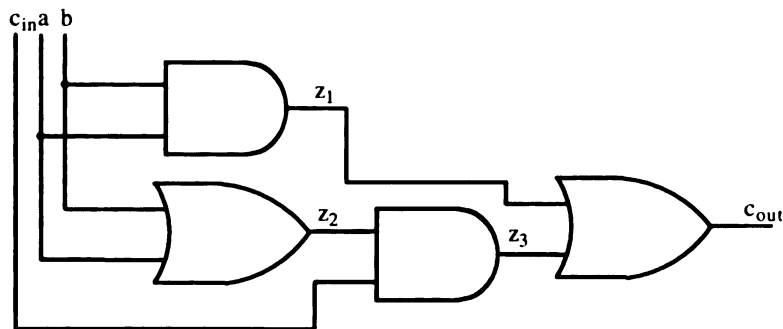


Figure 4.2. Logic diagram for FA carry-out bit.

4.2 Netlists

The structure of a circuit can be represented using a netlist. A *netlist* is a data structure that describes all components connected to each *net* or to each internally produced signal within a circuit. Industry standard formats exist for describing a netlist enabling the easy transfer of designs between various design tools and vendors.

Design tools may use netlist formats which represent circuit structure at various levels of design abstraction. For example, the SPICE tool considers the circuit-level of design abstraction. SPICE netlists model a circuit as an interconnection of analog elements such as *capacitors*, *resistors*, and *transistors*. The Berkeley SIS tool considers the logic-level of design abstraction. The widely used SIS netlist format, known as Berkeley Logic Interchange Format (BLIF), models a digital circuit as an interconnection of predefined or user-defined logic elements, which includes but is not limited to, *N/AND*, *N/OR*, *NOT*, or *XOR* gates.

Consider Figure 4.3 which is a gate-level netlist that describes the structure of a full-adder carry-out bit. The gate-level netlist example directly corresponds to the logic diagram shown in Figure 4.2 and the Boolean expression given in Equation 3.6. In general, netlist representations describe the structure and connective relationships visualized by schematic entry systems and may serve as their input. Similar


```

.inputs c_in a b
.outputs c_out

.gate and2 a=a b=b 0=z1
.gate or2 a=a b=b 0=z2
.gate and2 a=z2 b=c_in 0=z3
.gate or2 a=z1 b=z3 0=c_out
.end

```

Figure 4.3. Gate-level BLIF netlist for FA carry-out bit.

to schematic-based design, netlist-based design is limited by circuit size (number of nets). As the circuit size grows, the difficulty of debugging and isolating problems increases. Isolating circuit design problems given lower level circuit component granularity is a complex task when visually analyzing the netlist. Tools, such as *analyzers* or *simulators* are needed to locate and assess low level problems.

4.3 Structural VHDL Specifications

In VHDL, design architectures may be developed using a structural specification. *Structural specifications* express an architecture as a hierarchical arrangement of interconnected components. The interconnected components may be pre-defined library units or user-defined units. Structural-based VHDL design is powerful because it simplifies the development of complex systems. Larger or more complex systems are easier to design because the high-level system architecture can be viewed as an interconnection of less complex black boxes or components, which can be modeled behaviorally or structurally. The interconnection of components is made possible through the use of *signals*, which are analogous to wires.

A structural VHDL specification describing the carry-out bit of a full-adder (FA) is depicted in Figure 4.4 where the architecture closely resembles the logic diagram in Figure 4.2 and the gate-level netlist shown in Figure 4.3. Structural VHDL specifica-

```

ENTITY CARRY is
    port (A, B, Cin : IN BIT;
          Cout      : OUT BIT);
END CARRY;

ARCHITECTURE Structural of CARRY is

    -- declare internal components
    COMPONENT or_gate
        port (X, Y : IN BIT;
              Z   : OUT BIT);
    END COMPONENT;

    COMPONENT and_gate
        port (X, Y : IN BIT;
              Z   : OUT BIT);
    END COMPONENT;

    signal Z1, Z2, Z3 : BIT;

BEGIN -- Structural FA carry-out bit
    -- instantiate declared components to form structure
    AND1 : and_gate port map (X=>a, Y=>b, Z=>Z1);
    OR1  : or_gate  port map (X=>a, Y=>b, Z=>Z2);
    AND2 : and_gate port map (X=>Z2, Y=>Cin, Z=>Z3);
    OR2  : or_gate  port map (X=>Z1, Y=>Z3, Z=>Cout);
END Structural;

```

Figure 4.4. Structural VHDL model of FA carry-out bit.

tions are very useful when a design is decomposed into a set of components allowing a system to be viewed as an interconnection of library or user-defined black boxes. The structural VHDL design approach supports the interchanging of various off-the-shelf cores and other pre-defined components to help a designer meet system requirements.

4.4 Connective Binary Decision Diagrams

The Connective Binary Decision Diagram (CBDD) is a variant of the standard BDD. The CBDD is a major contribution of this research. It is entirely new and has some advantages over the traditional BDD implementations. CBDDs model the structure

present in a digital circuit's netlist description as well as the circuit's connective relationships. The combination of the CBDD's graph along with a set of graph traversal algorithms provide the necessary support to construct compact structural representations which maintain circuit behavior.

4.4.1 Motivation

The traditional BDD implementation is attractive because it is a canonical representation. The canonicity feature is very useful for circuit verification applications. But, the advantages of the BDD quickly diminish when the circuit size becomes large. Traditional BDDs may result in multiple subgraphs, one for each internal and primary output. Efficient manipulation of conventional BDDs is limited to modest-sized circuits. As the number of circuit inputs and logic elements increase, conventional BDD graphs may experience exponential ($\approx 2^N$) growth. Exponentially-sized graphs result in CPU-intensive applications and may exhaust the available memory of a typical workstation. Given the rapid advancement in VLSI design, the efficient representation of larger and more complex circuits by probabilistic power analysis tools is a crucial requirement. To meet this challenge a new decision diagram, whose size does not grow exponentially, is needed.

4.4.2 Overview

CBDD's, unlike conventional BDD implementations, maintain a circuit's structural input/output relationships and internal connectivity. The CBDD results in one directed acyclic graph which completely represents the structure of an entire circuit. The CBDD's graph grows linearly. Its size is proportional to the number of logic elements in the circuit description. As a result, for most circuits, CBDDs yield a more compact graph representation when compared to conventional BDD implementations.

4.4.3 Minimized Scalable BDDs

A significant component used to construct the CBDD's graph is the *minimized-scalable BDD* (MSBDD). MSBDDs are subgraphs which represent the behavior of a circuit's structural building blocks such as gates, muxes, adders, *etc.* The MSBDD concept is a contribution made by this research effort. Fundamentally, MSBDDs are Binary Decision Diagrams as discussed in [27], but are minimal in the number of nodes and edges used to construct the decision graph.

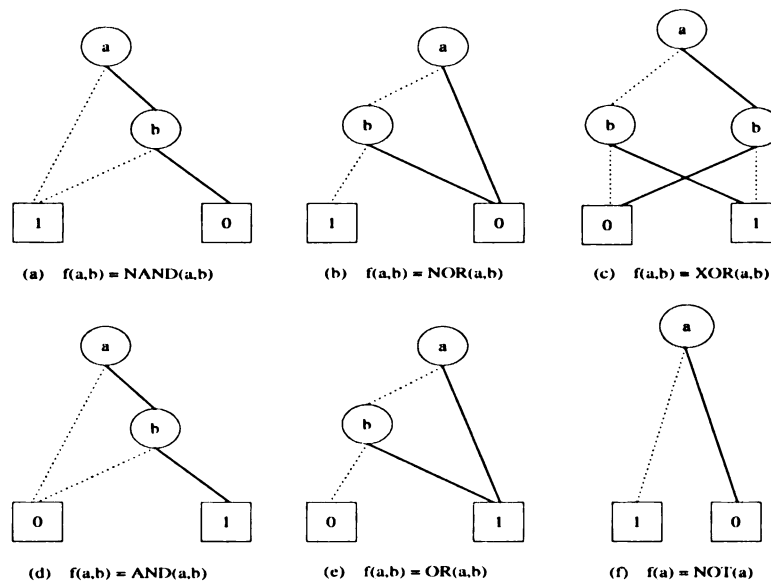


Figure 4.5. Minimized-scalable BDDs.

A revealing feature concerning the MSBDD is its ability to represent the behavior of a variable-input logic block with minimal size (node count). The MSBDD is designed to have a small number of decision nodes and only two terminal nodes for each logic block representation. The minimal size is achieved by utilizing unique *MSBDD Generation Routines* which recognize and exploit the *output drive* of the logic block being represented by the MSBDD. The output drive is a simplified set of decision

node interconnections which serves to minimized the logic block's output path depth. For example, an AND gate is 0-driven. The presence of a logic 0 at any AND input, regardless of fanin size, results in a logic 0 gate output. Likewise, a logic 1 at any OR gate input results in a logic 1 output. Therefore, OR gates are 1-driven. The output drive feature considered by the MSBDD supports the construction of minimized decision diagrams for certain logic elements such as NOT, NAND, AND, NOR, and OR gates, Figure 4.5. It is accomplished by increasing the number of direct connections or edges between decision nodes and terminal nodes within the MSBDD's graph with the goal of minimizing the depth.

4.4.3.1 MSBDD Generation Routines

The MSBDD generation routines are unique for each represented logic element. The main goal of each MSBDD generation routine is to produce an interconnection of nodes and edges which yield a decision diagram of limited size and depth. The CBDD implementation utilizes MSBDD generation routines which mimic the behavior of basic logic elements such as NOT, NAND, AND, NOR, and OR gates. Additionally, the CBDD implementation supports the inclusion of more complex logic structures. This is easily accomplished by adding a routine that generates an interconnection of nodes and edges which mimic the intended behavior.

The `Mk_NAND` procedure illustrated in Algorithm 1 generates an MSBDD graph that represents the behavior of an n -input NAND gate. This procedure interconnects the MSBDD graph with a CBDD graph in a manner that mimics the structure found in the input specification. The procedure utilizes three input parameters: a CBDD, an argument list, and fanin size. The CBDD's graph models the functionality of the network described by an input netlist. The argument list contains the fanin names, and fanin size is the number of inputs to the NAND gate.

Lines 2-4 of the `Mk_NAND` procedure build an array of decision nodes for each

```

Procedure: Mk_NAND
input      :  $G$  - CBDD of network
input      :  $n$  - Fanin size
input      :  $argv$  - Argument names
output     :  $M'$  - MSBDD of n-NAND logic gate

1 begin
2   for  $i \in 0..n$  do
3      $vlist[i] \leftarrow Mk\_CBDD\_Node\_Set(G, argv[i])$ 
4   end
5   for  $i \in 0..n$  do
6      $Add\_Edge(G, vlist[i], ZERO\_TERMINAL, ZERO\_EDGE)$ 
7   end
8   for  $i \in 0..n - 1$  do
9      $Add\_Edge(G, vlist[i], vlist[i + 1], ONE\_EDGE)$ 
10  end
11   $Add\_Edge(G, vlist[n-1], ONE\_TERMINAL, ONE\_EDGE)$ 
12   $Add\_Edge(G, ZERO\_TERMINAL, OUT\_NODE, VALUE\_EDGE)$ 
13   $Add\_Edge(G, ONE\_TERMINAL, OUT\_NODE, VALUE\_EDGE)$ 
14   $M' \leftarrow Connect\_Nodes(G, n, vlist, OUTPUT\_NODE)$ 
15 end

```

Algorithm 1: Mk_NAND_msbdd

argument. Lines 5-7 generate 0-edge connections between the decision nodes and a zero-terminal node. Lines 8-10 perform 1-edge connections between the decision nodes. Line 11 connects the last decision node to a one-terminal node. Using a *value*-edge, lines 12 and 13 connect the zero- and one- terminals to the output node. Line 14 connects the newly created output node to the CBDD's graph and returns an MSBDD reference.

The `Mk_CBDD_Node_Set`, `Add_Edge`, and `Connect_Nodes` operations invoked by the `Mk_NAND` procedure, run in $O(1)$ time. These operations are invoked a maximum of $3n + 4$ times, yielding a time complexity of $O(3n + 4)$, where n is the number of inputs. Since a NAND gate is being modeled, n is usually small, resulting in $n + 3$ nodes and $2n + 2$ edges. A similar routine exists for the remaining basic logic gates.

The CBDD package supports logic elements other than gates. The behavior of higher level logic elements such as *muxes*, *adders* and *encoders* can be modeled by the construction of unique MSBDD structures.

4.4.4 CBDD Definitions

An advantage of the CBDD is its ability to maintain the circuit's structural input/output relationships and internal connectivity. The definition of the CBDD is based on the definitions of a DAG and conventional BDD. Structural and connective relationships are achieved by altering the conventional BDD's definition to support additional node (vertex) and edge (arc) types and properties. The CBDD's formal definition is given and described by Definition 4.1 and Figure 4.6.

Definition 4.1 *A Connective Binary Decision Diagram (CBDD) is a directed acyclic graph which is composed of an MSBDD set M , vertex set V and edge set E .*

There are differences between the conventional BDD implementation and the CBDD. First, the conventional BDD graph contains only two node types, the decision and terminal node types. The CBDD's graph supports four node types, with each node serving a unique purpose. Second, the CBDD's graph supports an additional edge, the *value-edge*. Third, the CBDD's graph is an interconnection of subgraphs, known as Minimized-Scalable BDDs (MSBDDs).

CBDDs utilize the 0- and 1- edges in the same manner as a conventional BDD. The traversal of the 0- and 1- edges indicates that the predecessor decision node variable is equal to 0 or 1, respectively. The *value-edge* is new to the CBDD's design; *value-edges* are used to model the propagation of internally produced signals.

The CBDD's graph contains *Output*, *Input*, *Internal* and *Terminal* node types. The concept of an Output node is new; it represents or contains the value of a circuit's primary output. Output nodes are preceded by fixed-valued nodes or Terminal nodes

1. $\forall v \in V$ may be of type:
 - 1.1. *Input*
 - 1.1.1. followed by children via 0- and 1- *edges*
 - 1.1.2. assumes variable primary input value
 - 1.1.3. $value(v) \in \{0, 1\}$
 - 1.2. *Internal*
 - 1.2.1. followed by children via 0-, 1- and *value-edges*
 - 1.2.2. assumes variable internal output value
 - 1.2.3. $value(v) \in \{0, 1\}$
 - 1.3. *Terminal*
 - 1.3.1. value termination for internal and output nodes
 - 1.3.2. preceded by children via *value-edges*
 - 1.3.3. value is fixed, $value(v) \in \{0, 1\}$
 - 1.4. *Output*
 - 1.4.1. absolute termination, not followed by children
 - 1.4.2. assumes variable primary output value
 - 1.4.3. value is fixed, $value(v) \in \{0, 1\}$
2. $\forall e \in E$ may be of type:
 - 2.1. 0-edge (1-edge)
 - 2.1.1. traversed when $value(v) = 0(1)$
 - 2.1.2. outgoing to Input, Internal, and Terminal vertices
 - 2.2. *value-edge*
 - 2.2.1. propagates $value(v)$ to Internal or Output vertices
 - 2.2.2. incoming only to terminal vertices
3. MSBDD set M elements:
 - 3.1. $\forall m \in M, vertex(m) \in V$
 - 3.2. MSBDD terminals are connected to Internal or Output nodes via *value-edges*
 - 3.3. \forall MSBDDs represent a function on vertex v , f_v
 - 3.4. $f_v = \bar{x}_i \cdot f_{low(v)} + x_i \cdot f_{high(v)}$, where x_i is a decision variable

Figure 4.6. Definition of CBDD.

connected by *value-edges*, and are not succeeded by descendent nodes. The Input node is essentially a decision node, as in the conventional BDD; it represents the state of an input variable and may be followed by descendent nodes connected by 0-edges and 1-edges. The Internal node is a special decision node whose binary value results from the evaluation of a logic element represented by an MSBDD. Internal nodes are preceded by fixed-valued nodes or Terminals via *value-edges* and are succeeded by descendent Input nodes. The Internal node's value is propagated to the Input nodes of other logic element MSBDD representations by the use of *value-edges*. The

Terminal nodes used in the CBDD's graph differ from those used by the conventional BDD. CBDD Terminal nodes carry a fixed binary value but have descendents; they are succeeded by Internal or Output nodes and connected by *value*-edges.

4.4.5 Analysis of Connective-BDDs

Circuit size is a limiting factor for applications utilizing traditional BDD implementations. For large circuits BDD graphs may become exponentially-sized, resulting in CPU-intensive applications. The CBDD is an alternative graph structure which is useful for certain applications. This section provides background information concerning the CBDD's linear growth and functional rationalization. An explanation detailing the CBDD's improved circuit representation when compared to the traditional BDD is given.

Theorem 4.1 *The CBDD's graph grows linearly with respect to the number of logic elements within the circuit.*

Proof: Let M be the number of logic elements present in a circuit. Let n be the fanin size of a logic element. MSBDD graphs are constructed to grow linearly, with the largest MSBDD graph (n -XOR gate) having $2n + 3$ nodes. The CBDD's graph is an interconnection of M MSBDD graphs, resulting in linear growth and a maximum size of $M \times (2n + 3)$ nodes.

Theorem 4.2 *The minimized-scalable BDD (MSBDD) maintains the behavior of an individual logic element.*

Proof: The MSBDD's graph contains a single 0-Terminal and a single 1-Terminal. If a post-order traversal from the 1-Terminal to the root node of the MSBDD's graph is performed then a set of disjoint decision paths is generated. The

AND-product of the decision nodes along the disjoint paths are minterms in the *on-set* of the logic element's Boolean switching function. The OR-sum of these disjoint minterms equals the *sum-of-products* that describes the switching behavior of the given logic element.

Theorem 4.3 *The CBDD, an interconnection of MSBDDs, maintains the behavior of a circuit.*

Proof: From Theorem 4.2, it was established that MSBDDs maintain the behavior of individual logic elements. The post-order traversal of a MSBDD's graph results in a Boolean switching function whose switching variables may be primary inputs or internally produced signals. The circuit's behavior, a collection of 2-level Boolean switching functions for all circuit nodes, can be produced by performing recursive replacement of non-primary input variables by their corresponding MSBDD switching function until all non-primary input variables have been replaced by primary input variables.

Theorem 4.4 *The minimal MSBDD representation for n -input N/AND and N/OR logic functions contain $n + 3$ nodes and $2n + 2$ edges at most.*

Proof: The output of all n -input N/AND and N/OT logic functions is directly controlled by the logic function's *output drive*. The output drive of a logic function is a single input which directly controls the function's output. The n -input N/AND logic functions are 0-driven because a single logic zero applied to any of the n inputs leads to immediate output values of 1/0. Likewise, the n -input N/OR logic functions are 1-driven because a single logic one applied to any of the n inputs leads to immediate output values of 0/1. Assuming that a MSBDD graph representation contains a

maximum of two terminal vertices, the disjoint terms of the MSBDD's logic function are constructed from the *AND*-product of decision nodes within the MSBDD's graph. The terms have the following form: $\widehat{x}_N \cdot \prod_{i=0}^{N-1} x_i$, where x_i are decision nodes given that $0 \leq N \leq n - 1$, and \widehat{x}_N is an output driven decision node that is directly connected to a terminal vertex. Let L equal the number of literals for each disjoint term of the logic function represented by the MSBDD, such that $1 \leq L \leq n$. The number of edges necessary to construct a disjoint term and connect the term's output driving decision node to one of the terminal vertices within the MSBDD's graph is $1 + (L - 1)$ edges. An additional edge is needed to connect the term's non-output driving decision node combinations to a second terminal vertex. Additionally, two *value*-edges are used to connect the terminal vertices to the MSBDD output node. Given that K is the number of disjoint terms within the logic function, the total number of edges to minimally represent the MSBDD of a N/AND or N/OR logic function is given by

$$\begin{aligned}
\text{Edge Count} &= 2 + \sum_{i=0}^{K-1} (1 + (L_i - 1)) \quad \text{for } 1 \leq L_i \leq n \quad 1 \leq K \leq n \\
&= 2 + \sum_{i=0}^{K-1} L_i \quad \text{for } L_i = i + 1 \\
&= 2 + 2K \\
&\approx 2 + 2n
\end{aligned}$$

The minimal number of MSBDD nodes required to represent a N/AND or N/OR logic function is n plus the number of terminal vertices (usually 2), plus a single output node, for a minimum total of $n + 3$ nodes.

Assertion 4.1 *For large circuits CBDDs generally result in smaller and more compact representations when compared to traditional BDDs.*

Explanation: Applications using traditional BDDs try to make logical sense of the behavior obtained from a given input specification for purpose the of producing an optimized set of Boolean equations and limiting the BDD's size (node count). The BDD's size is controlled by the number of circuit inputs along with the input variable ordering.

Circuits containing many internal logic elements and a large number of inputs (N), may result in BDD graphs comprised of 2^N nodes along with $N!$ input variable orderings. For many applications, the time necessary to explore all $N!$ input variable orderings is not available, so a non-optimal input variable ordering is generally chosen. Benchmark comparisons demonstrate that non-optimal input variable ordering results in a BDD graph which is larger than a CBDD graph for the same circuit.

The CBDD does not analyze behavior or consider input variable ordering. For the logic elements encountered during the input of the design specification, the CBDD application interconnects the MSBDD subgraphs of the corresponding logic elements. According to Theorems 4.2 and 4.4, MSBDD subgraphs are unique and minimal in size for the given logic elements. From Theorem 4.3, the interconnection of MSBDD subgraphs results in a compact CBDD graph representing a circuit's behavior and connective relationships. For the reasons stated, the CBDD's graph maintains behavior and for large circuits it is smaller than the graph of traditional BDDs.

4.4.6 Implementation

The application program that generates a CBDD reads a BLIF-formatted netlist file as input. The basic logic gates such as N/AND, N/OR, XOR, and NOT, when encountered during the input phase, are converted to minimized-scalable BDDs (MSBDDs). These MSBDDs represent the most reduced BDDs, in terms of total node count, for the given functional unit (logic gate) and size.

Figure 4.5 displays a small selection of the MSBDDs. Once the MSBDDs are generated, their Internal node outputs are interconnected with the Input nodes of other MSBDDs according to the structure present in the netlist. The full-adder carry-out bit, structurally represented in Figures 4.2, 4.3, and 4.4, is represented by the CBDD in Figure 4.7.

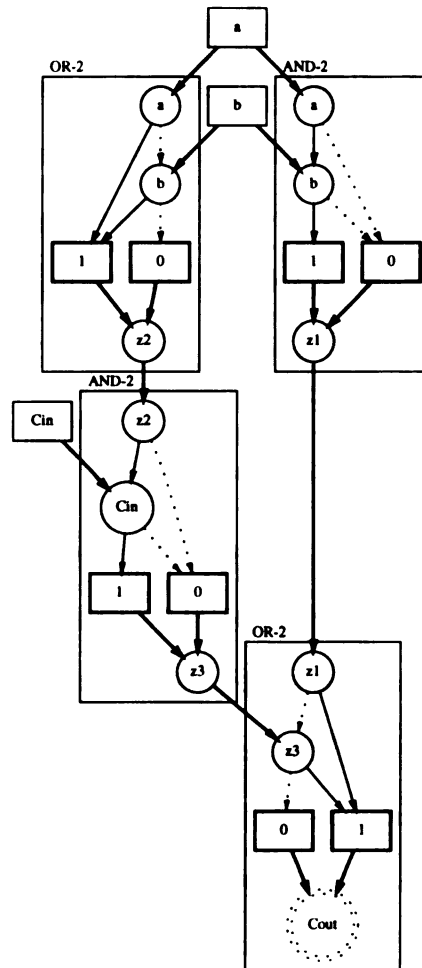


Figure 4.7. CBDD of FA carry-out bit.

The CBDD of Figure 4.7 models the structure of the full-adder carry-out bit. The CBDD of the carry-out bit is an interconnection of MSBDDs which model the behavior and structural connectivity of two 2-input AND gates along with two 2-

input OR gates, in accordance with the logic diagram of Figure 4.2. The MSBDDs for all logic gates are enclosed by labeled boxes featuring the gate name and size. The circuit's primary inputs, located outside of the MSBDD enclosures, are represented by boxes containing the input name. Primary inputs are connected to the MSBDD graphs by *value*-edges, depicted by heavy solid lines. Within the MSBDD enclosures 0- and 1- edges are represented by dotted lines and solid lines, respectively. Input nodes are represented by circles enclosing the input name and interconnected by 0- and 1- edges. Terminal nodes are represented by boxes containing a 0 or 1 value. Internal nodes are represented by circles containing an internally produced signal name; they are preceded by Terminal nodes connected by *value*-edges. Primary outputs are represented by double-dotted circles enclosing the output name; they are preceded by Terminal nodes.

4.4.7 Advantages and Disadvantages

In spite of the differences between the conventional BDD implementation and the CBDD, there are some very positive benefits of using the CBDD. First, the CBDD is not affected by input variable ordering because the internal logic elements of a digital circuit can be mapped to pre-defined MSBDDs, which are already minimal in size. Second, CBDD size (number of nodes) is directly related to the number of logic elements present in the circuit, so exponential growth will not occur. Third, the CBDD's graph represents the entire circuit design, whereas the traditional BDD utilizes multiple graphs, one for each primary output and internally produced signal. Fourth, the CBDD's size grows linearly because its graph utilizes Internal decision nodes representing internally produced signals, instead of expressing the specified node in terms of primary inputs which could result in exponential growth. The use of Internal nodes increases the sparseness of the CBDD's graph and provides a more compact representation of the circuit's behavior and structure.

The drawbacks of the CBDD include loss of canonicity. However, if needed, equivalence of two circuits can be determined by performing functional simulations using their CBDDs, followed by a comparison of their results. Additionally, unlike traditional BDDs, the CBDD's graph does not have a single path from the root node to a terminal node. This is not good because several paths must be considered when determining a primary output's value. However, in some cases CBDDs benefit from this drawback; they yield more compact graphs that are represented by far fewer nodes and edges than conventional BDDs. Thus, traversing additional paths in the CBDD is not costly because the maximum CBDD graph depth is less than the BDD graph depth. Lastly, in comparison to traditional BDD implementations, CBDDs generate a more compact representation for large circuits, but may produce larger than normal or poor representations for small circuits.

4.4.8 Results

The ISCAS-85 benchmark circuits were chosen for an experiment to compare the CBDD size to the size of traditional BDD implementations. The results after applying the CBDD implementation to the benchmark circuits are summarized in Table 4.1.

Additionally, as a means of comparison, Table 4.1 provides the results of implementations used by Brace and Shen in [30, 43] for the same benchmarks. The main entity used in comparing the BDD implementations was the BDD node count. The BDD node count is used as a measure of performance for both CPU and memory utilization. The term, **Unable**, used in Table 4.1, indicates that the corresponding decision diagram package was unable to provide a measurement for the given circuit due to memory limitations. In comparison to the BDD implementations by Brace and Shen, for almost every circuit of the benchmark suite with the exception of Shen's FBD node measurement for circuit **c7552**, the CBDD resulted in a significant node savings. Given the size of other circuits within the benchmark suite and their cor-

<i>Circuit</i>	<i>#Inputs</i>	<i>#Outputs</i>	<i>ROBDD</i> [30] <i>#Nodes</i>	<i>FBD</i> [43] <i>#Nodes</i>	<i>CBDD</i> <i>#Nodes</i>
c432	36	7	30200	31195	2017
c499	41	32	49786	33214	2899
c880	60	26	7655	7761	4276
c1355	41	32	39858	33214	6411
c1908	33	25	12463	12734	9387
c2670	233	140	Unable	57767	12886
c3540	50	22	208947	88652	17874
c5315	178	123	32193	26129	26078
c6288	32	32	Unable	115607	29361
c7552	207	108	Unable	19187	37774

Table 4.1. Benchmark Results.

responding FBD size, it is very likely that Shen’s report of the FBD node count for circuit **c7552** is an order of magnitude smaller than it should be.

The savings in node count exhibited by the CBDD implementation is due to usage of Internal nodes which represent the value of internally produced signals. The Internal nodes along with *value*-edges propagate the binary output of internal logic elements to subsequent MSBDD inputs nodes at deeper levels within the circuit structure. The small size of the CBDD is due to the fact that CBDDs grow with respect to the number of functional units or logic elements present in the circuit’s structure, not the number of inputs or input variable ordering.

4.4.9 Summary

This section has described the Connective BDD (CBDD), defined as a DAG interconnection of Minimized-Scalable BDDs. CBDDs are very economical in representing large circuits and maintain the circuit’s structural and connective relationships. CBDDs represent circuits with far fewer nodes than previous BDD implementations. CBDDs have a few drawbacks including loss of canonicity, multiple paths from the

root to terminal nodes, and poor representation of small circuits. The main advantage of the CBDD is that it does not suffer from exponential growth when the numbers of inputs and interconnections grow.

CHAPTER 5

Behavioral-Level Switching Activity Estimation

The accurate estimation of switching activity is an important and necessary procedure for probabilistic power analysis tools. Behavioral-level power analysis tools have the disadvantage of not having detailed information concerning structure and technology. This disadvantage is combined with the challenging task of computing dynamic power and switching activity with improved accuracy from an implementation-independent perspective.

There are very few design tools which provide power estimation for behavioral VHDL specifications. The average minimum error found in their power estimates is around 10%. Switching activity is a critical parameter needed to compute dynamic power dissipation (Equation 1.2) in CMOS circuits. A large portion of the error experienced by power analysis tools can be easily attributed to inaccurate switching activity (α) estimates due to factors such as *reconvergent fanout* and *input/internal correlations*.

This chapter presents a new technique which accurately performs high-level activity and power analysis. This approach operates at the behavioral level of design abstraction and uses behavioral VHDL specifications as input. The developed tech-

nique, and associated algorithms, have been implemented in a program called the Behavioral-Level Activity and Power Estimator (BLAPE). The details concerning the design and implementation are discussed, along with benchmark comparisons to demonstrate the effectiveness and capability of the new approach.

5.1 Methodology Overview

The focus of this section is to provide an overview of the assumptions, system architecture, and components used to implement the Behavioral-Level Activity and Power Estimator (BLAPE) system. A description of all major tasks which contribute to the process of accurately estimating switching activity is presented as well.

5.1.1 Methodology Assumptions

- *Circuit designs must be synthesizable.*

Input to the BLAPE system is in the form of behavioral VHDL specifications. The VHDL language supports many constructs such as **for-** and **while-** loops which are nice for simulation but may not result in a synthesizable design. The BLAPE system requires implementation-free and platform-unspecific design descriptions which represent real circuits that are mappable to some technology. This assumption supports the easy transfer of various benchmark VHDL specifications between BLAPE and other high-level power analysis tools.

- *Circuit types are combinational.*

The circuits targeted for analysis must be combinational, where all circuit outputs are functions of their primary inputs. This assumption simplifies the conversion of the input behavioral VHDL specification to a set of Boolean equations.

- *Zero delay model.*

All gate or logic element operation times are zero. Upon a change to the input stream, it is assumed that steady-state logic results appear at their gate outputs instantaneously. This assumption simplifies the computation of signal probability, but does not support the computation of glitch activity.

- *Uncorrelated primary inputs.*

Primary inputs are assumed to be spatially and temporally independent. This assumption allows the signal probability computation for each node to be a sum of disjoint input/internal signal probability products.

5.1.2 Methodology Outline

The BLAPE system is capable of operating on two input sources. It supports the use of behavioral/structural VHDL specifications as high-level input, along with a BLIF-formatted gate-level netlist as low-level input. The output generated by BLAPE consists of the switching activity (α) and node capacitance (C_L) for each net, along with a power estimate for the entire circuit. The system-level flow of the BLAPE implementation is given in Figure 5.1.

5.2 Methodology Task Decomposition

The Behavioral-Level and Activity Power Estimator system is composed of many individual tasks and sub-tasks. In this section a task decomposition of the methodology together with a brief description of the major tasks is given.

- *Task 1 - Transform behavioral VHDL specification to Boolean equations.*

This task performs syntax checking and determines if the VHDL input specification is synthesizable. A data structure containing primary inputs, primary

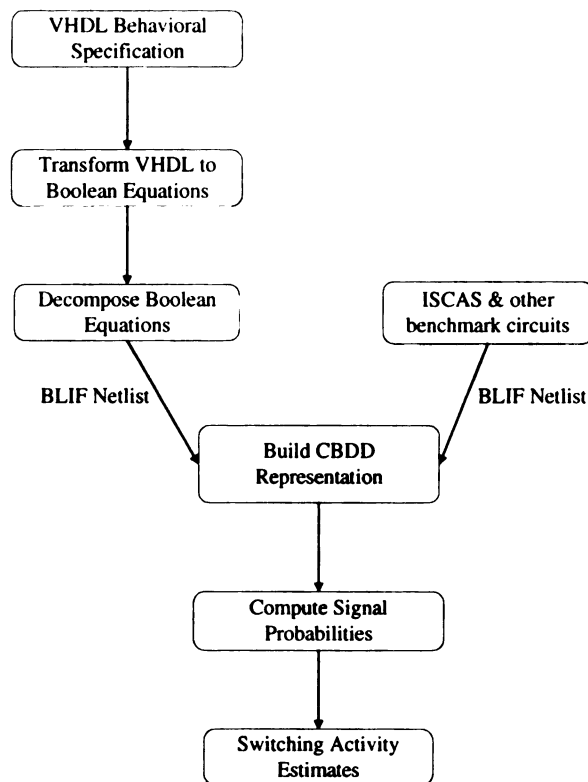


Figure 5.1. BLAPE methodology diagram.

outputs, and a list of Boolean equations in *sum-of-products* form is generated when the VHDL input specification is synthesizable.

- Task 2 - *Decompose Boolean equations.*

This task applies structure to the Boolean equations generated in Task 1 in two possible forms. First, *implicit structure*, which directly models a Boolean equation in *sum-of-products* form using the basic logic operations (NOT, AND, OR) only is generated. Second, *mapped structure*, where the Boolean equations are transformed to a structural specification consisting of gates or logic elements defined by a user-specified library is produced.

- Task 3 - *Generate Connective BDD (CBDD).*

This task converts the structural specification generated in Task 2 to a CBDD.

The CBDD maintains the design's original behavior and maps the input circuit's individual logic elements to nodes within the CBDD's graph.

- Task 4 - *Determine signal probabilities.*

This task traverses the CBDD generated in Task 3 in an effort to collapse a node's Boolean expression, represented by a multi-level Boolean function, to a Boolean expression with a user-specified depth, not less than 2. The final Boolean expression contains disjoint product terms, whose indices are primary input and internal signal probabilities. The result is a node signal probability that is almost independent or free of reconvergent fanout.

- Task 5 - *Estimate switching activities.*

This task computes the switching activity for a specified node given the signal probability provided in Task 4.

- Task 6 - *Compute dynamic power.*

This task determines the dynamic power of the entire circuit, utilizing the switching activity estimate (α) computed in Task 5, a computed estimate for node capacitance (C_L), and default values for supply voltage (V_{dd}), and clock frequency (f_{CLK}).

5.2.1 Transformation of VHDL into Boolean Equations

The initial stage of the BLAPE methodology requires the use of a VHDL specification as input. The VHDL modeling style is assumed to be behavioral, although a structural VHDL specification is permitted. The focus of this transformation is to provide a systematic and efficient mechanism for transforming a behavioral VHDL specification to a set of Boolean equations.

After careful consideration, it was decided that a VHDL compiler was the best solution to this problem. Like Ada, C, and Pascal programming language compilers, the VHDL compiler would be responsible for syntax analysis, bounds checking, and other normal compiler tasks. The **Altera MAX+PLUS II** compiler (Version 9.01) was selected. Useful and necessary features supported by the selected compiler include determination of design synthesizability and generation of a data structure containing the circuit's primary inputs and outputs, combined with a list of Boolean equations describing the input/output relationships of all primary outputs and internal nets.

Similar to the compilers of other programming languages, the selected VHDL compiler supports directives and optimizations to control the outcome of the synthesized design. One such optimization is logic reduction, whereby the desired compact logic representation is generated by means of technology mapping and by the removal of redundant or unused logic. The invocation of this logic reduction option will consistently generate the same reduced set of Boolean equations for various behaviorally equivalent designs. For the purpose of this research, the logic reduction feature was disabled. The rationale for making this decision was to preserve the design's inherent logical structure, with the understanding that structural modifications to the logic represented in the VHDL specification directly affect the switching activity and the power dissipated by the design. In making this decision, the BLAPE system yields unique power and activity estimates for various designs that are behaviorally equivalent. The goal of disabling the logic minimization option is accomplished by setting the synthesis style to *WYSIWYG* and setting the *MINIMIZATION* primitive to **OFF**.

Figures 5.2 and 5.3 depict a behavioral VHDL specification for a full-adder (FA) and the corresponding Boolean expressions, generated by the Altera compiler. The Altera compiler generates a report file containing the Boolean equations shown in Figure 5.3. The Altera Boolean equation notation makes use of the basic (AND/OR)

logic operations only. Each equation is represented by a list of product terms, where an & separates the literals of each term and each term appears on a single line separated by a # sign. All complemented literals are preceded by the ! character. These Boolean equations represent the behavior of the VHDL input specification and are eventually used to compute switching activity in later stages of the BLAPE implementation. Following the compilation of the VHDL specification, the generated Boolean equations are extracted from a report file. Next, the BLAPE implementation converts the Boolean equations to an *implicit* or *mapped* structural representation. The result is a gate-level netlist, discussed in the next section.


```

ENTITY FA is
    port (x, y, c_in : IN BIT;
          sum, c_out : OUT BIT);
END FA;

ARCHITECTURE behav of FA is
BEGIN -- behav
    sum  <= (x xor y xor c_in);
    c_out <= (c_in and (x or y)) or (x and y);
END behav;

```

Figure 5.2. Behavioral VHDL specification for full-adder.

```

c_in    : INPUT;
x       : INPUT;
y       : INPUT;

c_out   = _LC2_A1;
sum     = _LC1_A1;

_LC1_A1 = LCELL( _EQ001);
_EQ001 = c_in & x & y
        # c_in & !x & !y
        # !c_in & !x & y
        # !c_in & x & !y;

_LC2_A1 = LCELL( _EQ002);
_EQ002 = x & y
        # c_in & y
        # c_in & x;

```

Figure 5.3. Full-adder Boolean equations.

5.2.2 Decomposition of Boolean Equations

Applying structure to a design which originates as an algorithm or set of Boolean expressions binds the behavioral representation to a selected set and arrangement of logic elements. Decomposing the circuit specification to various structures enables a designer to select a design which best satisfies the system-level power requirements. The decomposition stage of the BLAPE implementation results in an *implicit* or *mapped* structural representation, which is then transformed to a BLIF-formatted gate-level netlist. The BLIF format was selected because of its simplicity and compatibility with the Berkeley SIS design tool.

5.2.2.1 Implicit Structure Representation

The implicit structure representation mimics the two-level *sum-of-products* form of the Boolean expressions. This representation places connective restrictions on the circuit behavior, using the basic logic gates (NOT, AND, OR) only. Given the full-adder Boolean equations (Figure 5.3), the resulting implicit structure netlist and corresponding logic diagram are illustrated in Figures 5.4 and 5.5.

In many instances, the resulting implicit structure representation does not yield the optimal structure because of unrealistic, although logically correct, logic elements which exceed input number restrictions. For this reason, the implicit structure should be used as an initial or starting structural design solution, which can be improved iteratively by the use of technology mapping procedures as described in the next section [45].

```

.inputs c_in x y
.outputs c_out sum

.gate not a=c_in 0=NOTc_in
.gate not a=y 0=NOTy
.gate not a=x 0=NOTx
.gate buf1 a=_LC2_B1 0=c_out
.gate buf1 a=_LC1_B1 0=sum
.gate and3 a=NOTc_in b=x c=NOTy 0=t0
.gate and3 a=NOTc_in b=NOTx c=y 0=t1
.gate and3 a=c_in b=x c=y 0=t2
.gate and3 a=c_in b=NOTx c=NOTy 0=t3
.gate or4 a=t0 b=t1 c=t2 d=t3 0=_LC1_B1
.gate and2 a=c_in b=x 0=t4
.gate and2 a=c_in b=y 0=t5
.gate and2 a=x b=y 0=t6
.gate or3 a=t4 b=t5 c=t6 0=_LC2_B1
.end

```

Figure 5.4. Netlist for implicit full-adder structure.

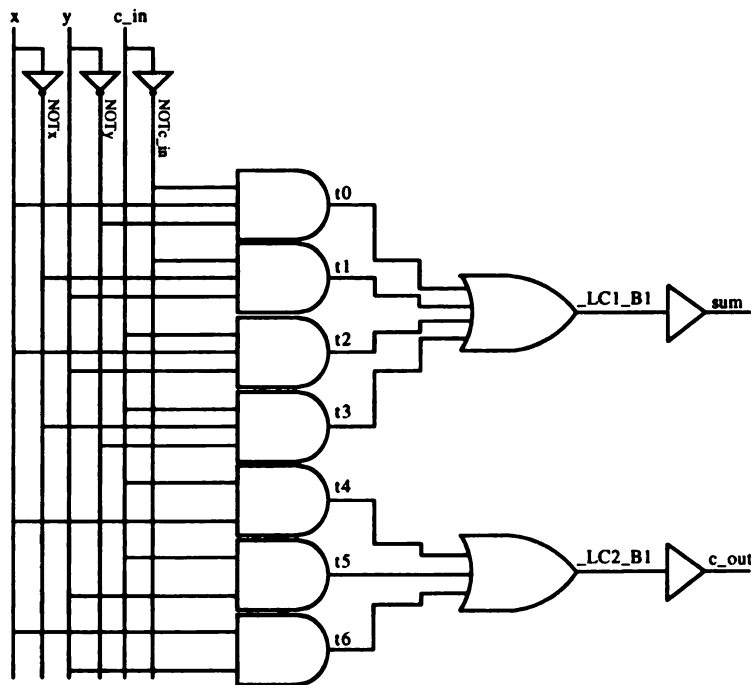


Figure 5.5. Logic diagram for implicit full-adder structure.

5.2.2.2 Mapped Structure Representation

The mapped structure representation is a structural circuit specification composed of user-specified gates or logic elements. This specification is generated by an automated process, known as *technology mapping*. The technology mapping process transforms an optimized set of technology independent logic equations into a feasible circuit which satisfies certain area, delay, and power constraints. The role of technology mapping is neither to radically change the circuit's structure nor to reduce the circuit's depth along the critical path, but to perform the best gate selection for implementing the logic equations, given the system-level area, delay, and power criteria [39].

For the purposes of this research structural mapping is used as a means to generate a realistic circuit specification. As illustrated by Figure 1.8, various circuit structures or decompositions yield different switching activities, and hence different power dissipations. This stage of the BLAPE process supports the computation of improved switching activity by producing a gate-level structural specification which closely resembles the desired circuit. As a result, the activity and power estimates are more meaningful and consistent with eventual circuit- or transistor- level switching activity and power approximations.

The BLAPE implementation utilizes the Berkeley SIS tool's technology mapping procedure. During the mapping stage of the BLAPE process, an implicit structural circuit specification is used as input to the SIS tool, along with a user-specified gate library. The *map* operator is applied, resulting in a behaviorally-equivalent circuit that can be optimized for area and delay. For example, given a gate library composed of **NOT**- and **NAND**- gates only, a technology mapping of the full-adder implicit structural representation, shown in Figure 5.4, results in the mapped structural representation displayed in Figure 5.6. Additionally, the corresponding logic diagram for the full-adder's mapped structure representation is given in Figure 5.7.

```

.inputs c_in x y
.outputs c_out sum

.gate nand2 a=c_in b=y 0=[236]
.gate nand2 a=x b=y 0=[238]
.gate nand2 a=c_in b=x 0=[234]
.gate nand3 a=[236] b=[238] c=[234] 0=c_out
.gate not a=y 0=[182]
.gate not a=c_in 0=[181]
.gate nand3 a=[182] b=x c=[181] 0=[226]
.gate not a=x 0=[183]
.gate nand3 a=y b=[181] c=[183] 0=[228]
.gate nand3 a=c_in b=x c=y 0=[230]
.gate nand3 a=[182] b=c_in c=[183] 0=[232]
.gate nand4 a=[226] b=[228] c=[230] d=[232] 0=sum
.end

```

Figure 5.6. Netlist for mapped full-adder structure.

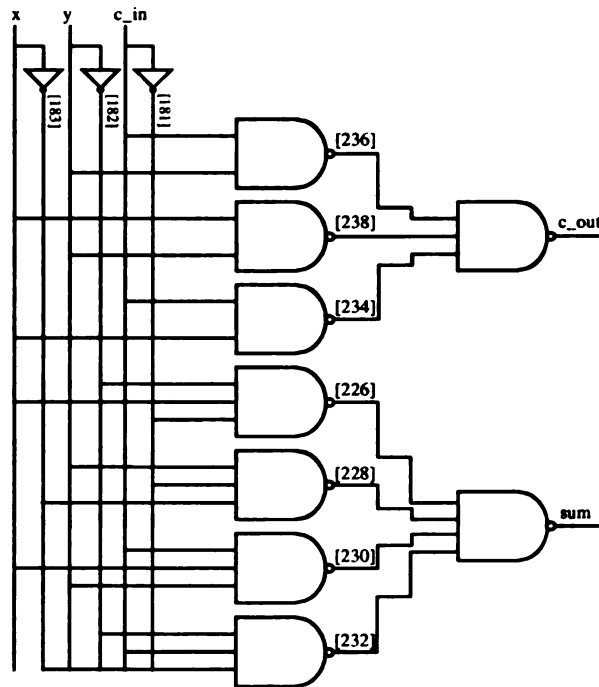


Figure 5.7. Logic diagram for mapped full-adder structure.

5.2.3 Building the CBDD Representation

5.2.3.1 CBDD Selection Rationale

The Connective Binary Decision Diagram (CBDD) is a directed acyclic graph (DAG) which serves as a graph-based behavioral representation capable of modeling a circuit's input/output relationships and internal connectivity. The CBDD was selected due to its linear growth and preservation of circuit structure. Because it uses the CBDD, the BLAPE implementation is capable of processing larger circuits in comparison to the Berkeley SIS tool. Many CAD applications that use traditional BDDs, specifically probabilistic power analysis tools, are limited to modest-sized circuits.

CAD applications generally encompass a class of problems, known as *nondeterministic polynomial* or *NP-complete* problems [39]. For this class of problems there are no known feasible or polynomial-time solutions. The solutions to such problems, which use conventional BDDs, are computationally expensive and easily capable of exhausting a typical workstation's memory system. The time complexity of algorithms which use decision diagrams is $O(f(|V|, |E|))$, where f is an operation performed on a graph such that $|V|$ and $|E|$ represent the size of the graph's vertex and edge sets, respectively. A reduction in both $|V|$ and $|E|$ is appealing to CAD applications as it reduces time complexity. An additional reason for selecting the CBDD for this research is due to its improved representation of large circuits. When compared to other conventional BDD implementations, using ISCAS-85 benchmark circuits, the CBDD's graphs were significantly smaller. An average size reduction of one order of magnitude was achieved [46].

5.2.3.2 Generation of CBDD Graph

The CBDD's graph is produced during the structural specification input phase. The structural specification is a BLIF-formatted netlist which may be generated by

BLAPE or selected from the ISCAS-85 or other benchmark suites. Each logic gate, along with its inputs and output is transformed into a minimized-scalable BDD (MS-BDD). For the basic logic gates (N/AND, N/OR, XOR, NOT), there are routines (Algorithm 1) which construct the behavior of the specified gate. These routines build MSBDDs with the least number of nodes and edges for the specified fanin size. The node and edge data structures are placed into a lookup table of lists for the purposes of maintaining connectivity.

During the input phase, an *adjacency list* is built. The adjacency list contains connection (wire) information, which describes node to node reachability and is used for network levelization and other tasks. The CBDD is an edge-driven DAG rather than vertex or node driven. Traversals and graph inquiries rely on routines which search for and manipulate sets of edges. The precise modeling of signal flow is made possible by the CBDD's *value-edge*. Additionally, the *value-edge* is responsible for the linear growth of the CBDD's graph. The CBDD representations corresponding to the implicit and mapped full-adder structural representations shown in Figures 5.4 and 5.6 are illustrated in Figures 5.8 and 5.9, respectively. The mapped structural CBDD has fewer nodes and edges. It uses an interconnection of NOT and NAND gates only. Later sections of this chapter will demonstrate that these two full-adder structures representations yield different switching activities and different power dissipations.

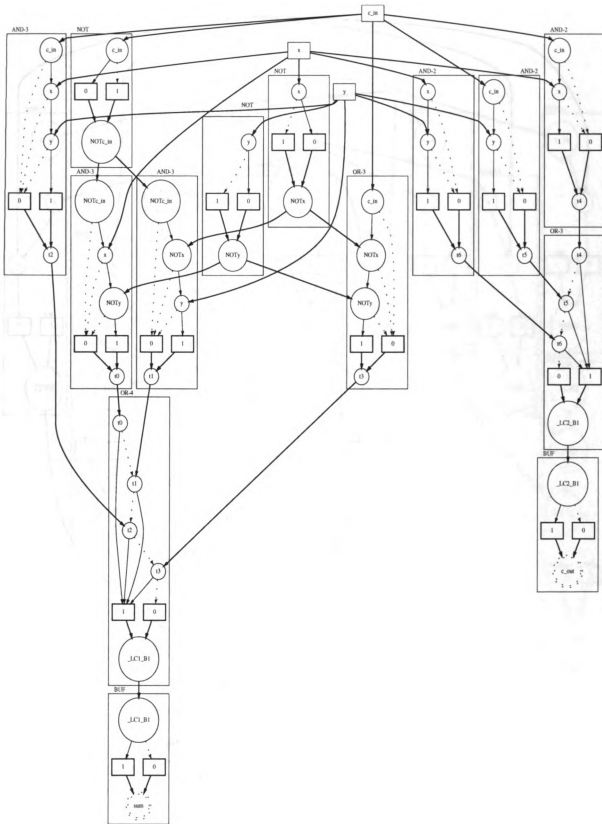


Figure 5.8. Implicit structural CBDD full-adder.

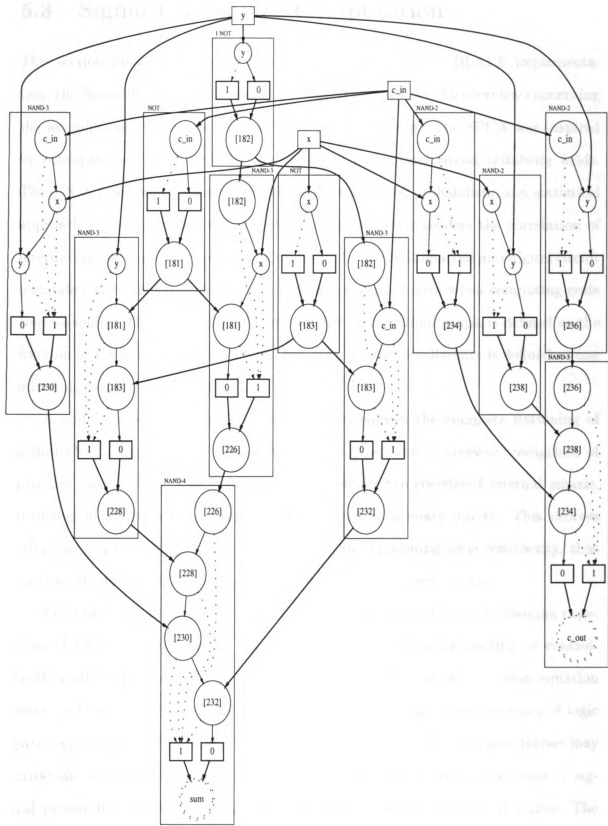


Figure 5.9. Mapped structural CBDD full-adder.

5.3 Signal Probability Computation

This section presents the most important component of the BLAPE implementation, the *Signal Probability Computation Algorithm* (SPCA). An overview concerning the assumptions, origin and details of the SPCA is given. The SPCA was inspired by concepts associated with the *fault analysis* and *digital circuit testability* fields. The SPCA is an improvement over previously mentioned probabilistic and statistical approaches for computing signal probability because it considers the correlation of internal signals caused by reconvergent fanout. In the fault analysis and digital circuit testability areas, reconvergent fanout is a large source of error when computing node *controllability*. The 1-controllability of an edge in the circuit graph is defined as the fraction of 1's in the truth table of its function. Its 0-controllability is defined as one minus its 1-controllability [47].

A node's exact controllability computation requires the complete flattening of a multi-level expression to a two-level or *sum-of-products* expression, comprised of primary inputs only. The flattening aspect uncoils the correlated internal signals, resulting in an expression composed of independent primary inputs. This process offers an exact solution, but is hampered by an exponential time complexity, thus limiting the technique's usefulness to small- or modest- sized circuits.

The exact computation of signal probability encounters the same dilemma experienced when computing exact controllability. Exact signal probability or controllability offers the best or most appropriate solution for a node's Boolean equation since each has zero error. When addressing a network as an interconnection of logic gates, error due to reconvergent fanout arises (Figure 2.2). Reconvergent fanout may cause an overestimate (Equation 2.3) or underestimate in the computation of signal probability, due to the statistical dependence of signals arriving at a gate. The SPCA approximates signal probability, which is less accurate than exact computation.

The approximation scheme assumes that the dependence resulting from reconvergent fanout is reduced the further the distance between a gate's fanout and the resulting reconvergent fanin gate. This assumption is used by fault analysis heuristics that compute node controllabilities of large circuits [47]. The assumption allows SPCA to assume that after a certain depth (distance), reconvergent fanins may be considered to be *almost-independent* or very close to statistically independent. As a result, the signal probability approximation approach is much faster and less complex than the exact approach.

5.3.1 Overview

The SPCA is responsible for cost-effectively computing the signal probability at each node in the network. The algorithm has four inputs: 1) node identifier, 2) CBDD, 3) input signal probabilities, and 4) depth-accuracy parameter (k). The input signal probabilities are a user-specified set of real numbers which represent the probability that inputs are logic one or $P(x_i = 1)$. The depth-accuracy parameter (k) is the depth or level to which the SPCA collapses or flattens the network. A larger depth-accuracy yields a more accurate signal probability estimate at the expense of time and memory resources. A structural overview of the SPCA is illustrated in Figure 5.10.

The SPCA involves the coordination of four main components: 1) Network Levelization, 2) Disjoint Post-Order Boolean Equation Generation, 3) IPR Cubeset Generation, and 4) Signal Probability Computation. The *Network Levelization* component is responsible for ordering the CBDD nodes according to their degree of logical independence. The *Disjoint Post-Order Boolean Equation Generation* component is responsible for traversing the CBDD from a given node with the intent of producing a disjoint post-ordered Boolean equation that is flattened to k (depth-accuracy) levels. The *IPR Cubeset Generation* component makes sense of the post-ordered Boolean

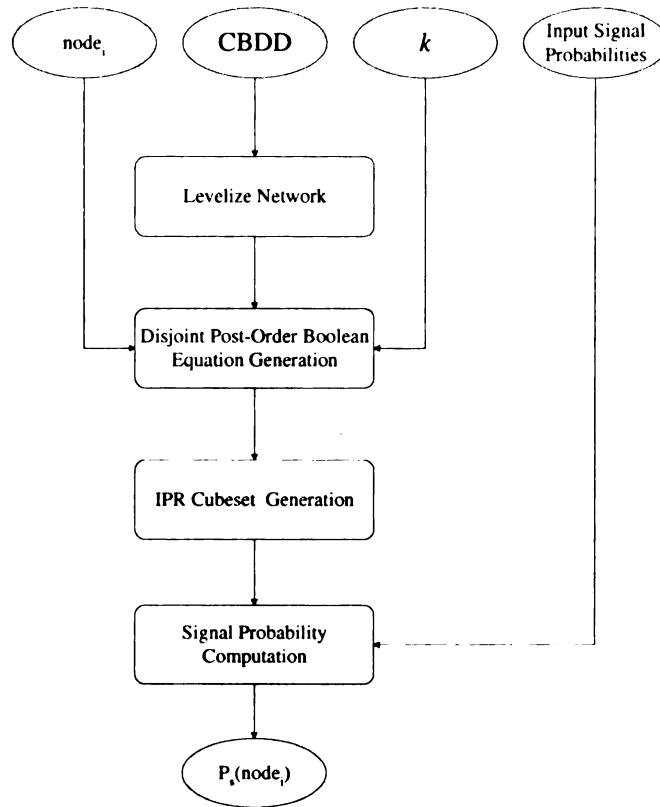


Figure 5.10. SPCA structural overview.

equations; it converts the equations to a set of Integer Pair Representation (IPR) cubes, in *infix* form. The *Signal Probability Computation* component reads the IPR cubeset and approximates the given node's signal probability.

5.3.1.1 Network Levelization

Before the generation of Boolean equations takes place, each node in the network is leveled. Levelization is the process of ordering the internal/output nodes of the CBDD's graph in terms of their depth or distance from the primary inputs. The nodes at lowest level of depth, level zero, are the primary input variables. The levelization process is managed by the `Network.Levelization` procedure (Algorithm 2); this process is necessary because the approximated signal probability of internal nodes is repeatedly used for future signal probability computations involving deeper

nodes. The `Network_Levelization` procedure relies on a simple recursive algorithm which performs a depth-first search of the CBDD's graph. The search runs in $O(n + e)$ time, where n and e are the number of nodes and edges of the CBDD's graph, respectively. The depth-first search is a popular and well known search algorithm used for traversing graphs. Detailed analysis concerning the complexity of the depth-first search algorithm is reported in [48].

```

Procedure: Network_Levelization
input      :  $v$  - CBDD node
input      :  $lev$  - Level of node

1 if  $lev > v.lev$  then
2    $v.lev \leftarrow lev$ 
3 end
4  $a \leftarrow v.adj$ 
5 while  $a \neq NULL$  do
6   Network_Levelization( $a.v, lev + 1$ )
7    $a \leftarrow a.next$ 
8 end

```

Algorithm 2: Levelize network procedure.

Correctness of Network_Levelization procedure

Claim 5.1 *Network_Levelization(v, lev) produces a dependency ordering of the nodes in a directed acyclic graph.*

Explanation: Assume that v is initially the DAG root node, and lev is initially set to zero. Let $a \in DAG$. Upon the initial invocation of the Network_Levelization procedure all node levels are zero. Each invocation compares the node level, $v.lev$, to the stack argument, lev , at line 1. Line 2 updates $v.lev$ (node v 's level) with the larger level value from the stack when the comparison condition of line 1 is satisfied. For every invocation of Network_Levelization, node variable a is assigned to the adjacency list of node variable v on line 4. The adjacency list assigned to variable a contains all descendents of node variable v . The nodes adjacent to node v are visited sequentially by the execution of lines 5-7. Each node having descendents experiences a recursive invocation of the Network_Levelization procedure at line 6. Each recursive invocation traverses to a deeper level within the CBDD's graph. The traversal finds a deeper generation of node v 's lineage and results in an increment of the stack argument lev . If some node y experiences multiple recursive invocations, $y.lev$ is only updated when

a larger level is encountered causing each node to be updated with the appropriate dependency level value. Recursive invocations terminate when all paths from the root node to all primary output nodes have been traversed.

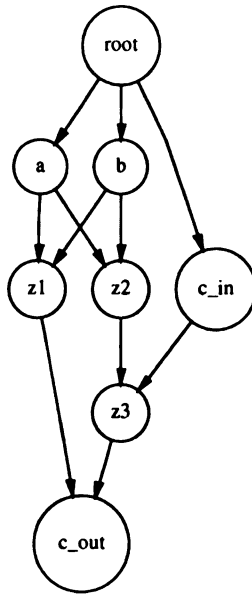


Figure 5.11. DAG of FA carry-out bit network.

<i>Node</i>	<i>Adjacency List</i>	<i>Level</i>
<i>root</i>	<i>a, b, c_in</i>	-1
<i>a</i>	<i>z1, z2</i>	0
<i>b</i>	<i>z1, z2</i>	0
<i>c_in</i>	<i>z3</i>	0
<i>z1</i>	<i>c_out</i>	1
<i>z2</i>	<i>z3</i>	1
<i>z3</i>	<i>c_out</i>	2
<i>c_out</i>		3

Figure 5.12. Carry-out bit adjacency list and levelization.

The inputs to the Network_Levelization procedure include: 1) CBDD node and 2) node level. The procedure is initially invoked with the root node of the given CBDD,

with the node level parameter set to zero. The procedure's output is the level ordering for each of the CBDD's internal and output nodes. Figure 5.12 shows an example levelization for the mapped full-adder carry-out bit network. The DAG modeling just the primary input, gate and primary output interconnections maintained by the CBDD is given in Figure 5.11. The network node adjacency list is illustrated in Figure 5.12. The results of the network levelization example demonstrate that primary inputs are the most independent nodes; they appear at level zero. Due to their dependence on other signals, internal signals and primary outputs appear at deeper levels of the network.

5.3.1.2 Disjoint Post-Order Boolean Equation Generation

The generation of disjoint Boolean equations is a key task necessary to the computation of signal probability. The procedure described in this section produces a disjoint Boolean equation, where all terms are mutually exclusive similar to the minterms of a truth table. The `Post_Order` procedure, Algorithm 3, is a simple recursive routine that ascends from some node (v) to a user-selected depth-accuracy (k) within a CBDD's graph. The goal of the `Post_Order` procedure is to construct a disjoint post-ordered Boolean equation composed only of literals that appear a maximum of k levels above node v 's level within the CBDD's graph.


```

Procedure: Post_Order
input      : CBDD - CBDD of network
input      : v - Vertex
input      : d - Vertex depth
input      : lcnt - Level counter
output     : POE_Stack - Post ordered expression stack

1 begin
2   Edge_Set  $\leftarrow$  Get_In_Edges(CBDD, v)
3   for  $\forall E \in$  Edge_set do
4     if E.edge.type = ZERO_EDGE or E.edge.type = ONE_EDGE
5       then
6         tcnt  $\leftarrow$  tcnt + 1
7         eV  $\leftarrow$  Get_Source_Vertex(CBDD, E.vertex)
8         Sign  $\leftarrow$  (E.edge.type = ZERO_EDGE ? FALSE : TRUE)
9         Val  $\leftarrow$  (E.edge.type = ZERO_EDGE ? - eV.idx : eV.idx)
10        if eV.vertex.type = INPUT then
11          Push(POE_Stack, Val)
12        else
13          if d < depth_accuracy(k) then
14            T  $\leftarrow$  Get_Terminal_Vertex(CBDD, eV, Sign)
15            Post_Order(CBDD, T, d + 1, 0)
16          else
17            Push(POE_Stack, Val)
18          endif
19        endif
20        Post_Order(CBDD, eV, d, lcnt + 1)
21        if lcnt > 0 then
22          Push(POE_Stack, *)
23        end
24        if tcnt > 0 then
25          Push(POE_Stack, +)
26        end
27      end
28 end

```

Algorithm 3: Post-order CBDD traversal.

The design of the `Post_Order` procedure is the result of modifications to the traditional post-order binary tree traversal. The post-order traversal was selected because it's a very effective means for circumnavigating a graph to produce meaningful expressions. The most common implementation of the post-order traversal algorithm is recursive, where all un-traversed node edges are visited first, followed last by a node visit(output). The result is an expression in *postfix* notation. Given the expression tree, Figure 5.13, describing the full-adder carry-out bit Boolean equation (Equation 3.6), the resulting *postfix* expression is $ab * cab + **$.

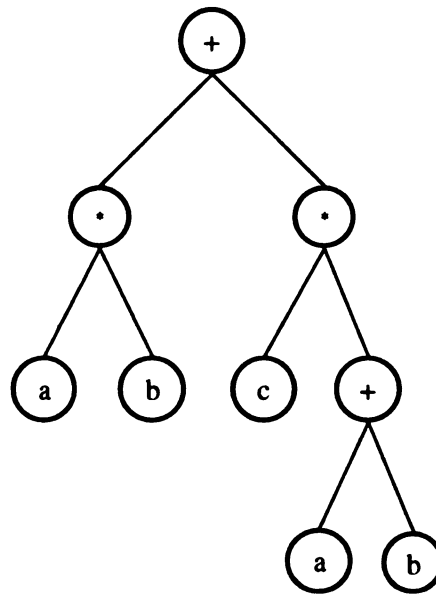


Figure 5.13. Expression tree for full-adder carry-out bit.

Correctness of `Post_Order` procedure

The `Post_Order` procedure arguments include 1) a CBDD, 2) vertex v , 3) invocation depth d and 4) node counter $lcnt$. Argument v is a node within the CBDD's graph, the start point of the traversal. Argument d is the procedure's invocation depth given node v . The $lcnt$ argument tracks the number of Output nodes encountered

during the procedure's recursive invocations for a given vertex v .

Claim 5.2 *The Post_Order procedure generates a valid post-ordered Boolean equation.*

Explanation: The Post_Order procedure generates a post-order Boolean expression composed of Boolean AND/OR operations accompanied by literals that represent the value of individual decision nodes. The procedure assumes that all logic operations have two inputs which alleviates the need for delimiters. The Post_Order procedure is invoked with the vertex depth (d) and the level counter ($lcnt$) arguments. Initially, both arguments are set to zero. The vertex, v , is initially set to the 1-Terminal of the specified MSBDD within the CBDD's graph.

The procedure begins by collecting the incoming edges of node v on line 2. The incoming edge set of node v is traversed by lines 3-27. For each iteration of the edge set, line 4 checks if the given edge is a 0-edge or 1-edge. If the line 4 condition is satisfied, then lines 5-23 are encountered, otherwise lines 23-26 are encountered. Lines 24-26 **Push** the Boolean OR symbol to the output stack if at least one pair of post-order arguments have been previously added to the stack. The block between lines 5 and 23 allow the Post_Order procedure to ascend from a given node to its predecessor node. A counter tracking the number of post-order argument pairs per invocation is maintained on line 5.

To assist with the ascending traversal from node v to its predecessor, line 6 locates the head vertex of the current edge, by invoking the Get_Source_Vertex routine. Based on the current edge type, the *sign* variable is updated in line 7. The *sign* variable contains the Boolean phase of the head source vertex. The index value of head source vertex is determined on line 8. A negative index value indicates that the head vertex literal is complemented. If the head vertex models a primary input, then lines 9-10 add the head vertex index value of line 8 to the stack. Otherwise, the head

source vertex models an internal signal and is dealt with in lines 12-17. If the depth of the search is less than the depth-accuracy (k) parameter a recursive invocation takes place on line 14; this allows higher exploration into the CBDD's graph. Line 13, invokes the `Get_Terminal_Vertex` routine which directs the upward search from the edge's source vertex to a Terminal vertex in the preceding MSBDD. The Terminal is chosen based on the literal *sign* value. Lines 15-17 place the source node's literal value on the output stack. Line 19 performs a second recursive invocation and increments the level counter argument. The second invocation allows for the upward traversal within the subgraph of the new MSBDD resulting from Line 13. Lines 20-22 build the Boolean product terms by pushing a Boolean AND symbol to output stack when the second literal (decision node) is encountered during the recursive invocation. Lines 24-26 push the Boolean OR operator symbol to the output stack just after the pairs of product terms encountered during the upward traversal from the Terminal node. The repeated interaction of lines 2-27 result in an output stack that contains product terms whose literals are positioned k levels or less above the specified input vertex argument within the CBDD's graph. All terms are described using 2-input Boolean AND operators and each post-order pair of terms is followed by a Boolean OR symbol on the output stack.

Analysis of Post_Order procedure

Traditional post-order BDD traversals explore only two edges per node because of the binary nature of the BDD's implementation. The Post_Order procedure for the CBDD must consider two edges per Input node and β edges per Internal node, where β is the maximum fanout of an internally produced signal. Let G be the number of logic elements in the circuit. According to the analysis of the `Mk_NAND` procedure, a β -input MSBDD NAND representation has $\beta + 3$ nodes and $2\beta + 2$ edges. The

fanout of the NAND MSBDD's Output node results in a maximum of β value-edges, yielding a maximum of $3\beta + 2$ edges per logic element. The Post_Order procedure is recursively applied on a per node basis. The traversal encounters a maximum of V_{max} nodes and E_{max} edges which are $G \times (\beta + 3)$ and $G \times (3\beta + 2)$, respectively.

For each recursive iteration of the Post_Order procedure, assume that the operations at lines 2-8 are always performed. These operations run in constant time, with a total constant time of $O(7)$. The *if-then-else* block at lines 9-18 runs in $O(4)$ time, by assuming that lines (9, 12, 13, 14) always run, yielding the longest possible thread through that block. The recursive invocation on line 19 runs in $O(1)$ time, and the *if-then* blocks at lines 20-22 and lines 24-26 both run in $O(2)$ time for a total constant time of $O(4)$. The block of lines from 3-27 encloses a total constant time of $O(7) + O(4) + O(5) = O(16)$. The search can be viewed as a traversal of all edges with visitation to all nodes in the ascending ancestry of the specified start node. The largest number of invocations of the Post_Order procedure for a given node is $V_{max} + E_{max}$, with each invocation costing a maximum of $O(16)$ time. Thus, the worst case running time of the Post_Order procedure is $O(16 \times (V_{max} + E_{max}))$. Given the user-selected depth-accuracy parameter, k , and the maximum logic element fanout, β , the overall worst case running time of the Post_Order procedure can be expressed as $O(16 \times (4\beta + 5) \times G)$.

		Post - Order Boolean Equations		
Idx	Node	k=1	k=2	k=3
1	c_in	-	-	-
2	a	-	-	-
3	b	-	-	-
4	z1	3 2*	3 2*	3 2*
5	z2	2 3 (-2) * +	2 3 (-2) * +	2 3 (-2) * +
6	z3	1 5*	1 2 3 (-2) * + *	1 2 3 (-2) * + *
7	c_out	4 6 (-4) * +	3 2 * 1 5 * (-2) (-3) 2 * * +	3 2 * 1 2 3 (-2) * * * (-2)(-3) 2 * * +

Table 5.1. Post-order Boolean equations for FA carry-out bit.

After invoking the `Post_Order` procedure with the full-adder carry-out bit CBDD, Figure 4.7, the resulting post-ordered Boolean equations are shown in Table 5.1 for $k = 1, 2,$ and 3 . From inspection, as the depth-accuracy parameter increases the length of the post-order equations increase as expected. At the request of a deep network flattening (large k), nodes having large fanouts (many *value*-edges) contribute to a dramatic increase in the run-time of the procedure. Thus, a deeper equation flattening requires longer execution times and obviously a greater demand on memory resources.

5.3.1.3 IPR Cubeset Generation

The *IPR Cubeset Generation* component is responsible for converting the post-ordered Boolean equations, generated by the `Post_Order` procedure, to a collection of IPR (Integer Pair Representation) cubes in *infix* form. IPR is a very compact Boolean equation representation system, developed by Diaz and Jimenez in [49]. A *cube* is similar to a product term. It may represent one or more minterms of a switching function. It consists of a pair of integers: the *position* and *expansion*. The position integer represents the literals of the individual switching variables and the expansion integer indicates whether a literal is in use or if it is a *don't care* variable [49]. A *cubeset* is a list or collection of cubes. For the purposes of the SPCA, cubesets are used to efficiently represent the behavior of switching functions.

The *Generate_IPR_Cubeset* procedure, Algorithm 4, is responsible for generating an *infix* Boolean expression in terms of IPR cubes. The procedure's input argument is a post-ordered Boolean expression stack that is composed of Boolean operators (AND/OR) and literals only. It is assumed that all operators are binary, requiring two inputs. This assumption simplifies the conversion of *postfix* to *infix* notation because delimiters are not necessary.

```

Procedure: Generate_IPR_Cubese
input      : IN_Stack - Input expression stack
output     : IPR_Cubese - IPR cubese

1 begin
2   while IN_Stack Not Empty do
3     repeat
4       item ← Pop(IN_Stack)
5       if item.type = OPERATOR then
6         Operator ← item
7       else
8         Push(Eval_Expr_Stack, item)
9       endif
10      until item = OPERATOR
11      item1 ← Pop(Eval_Expr_Stack)
12      item2 ← Pop(Eval_Expr_Stack)
13      result ← Perform_IPR_Operation(Operator, item1, item2)
14      Push(Eval_Expr_Stack, result)
15    end
16    IPR_Cubese ← Pop(Eval_Expr_Stack)
17 end

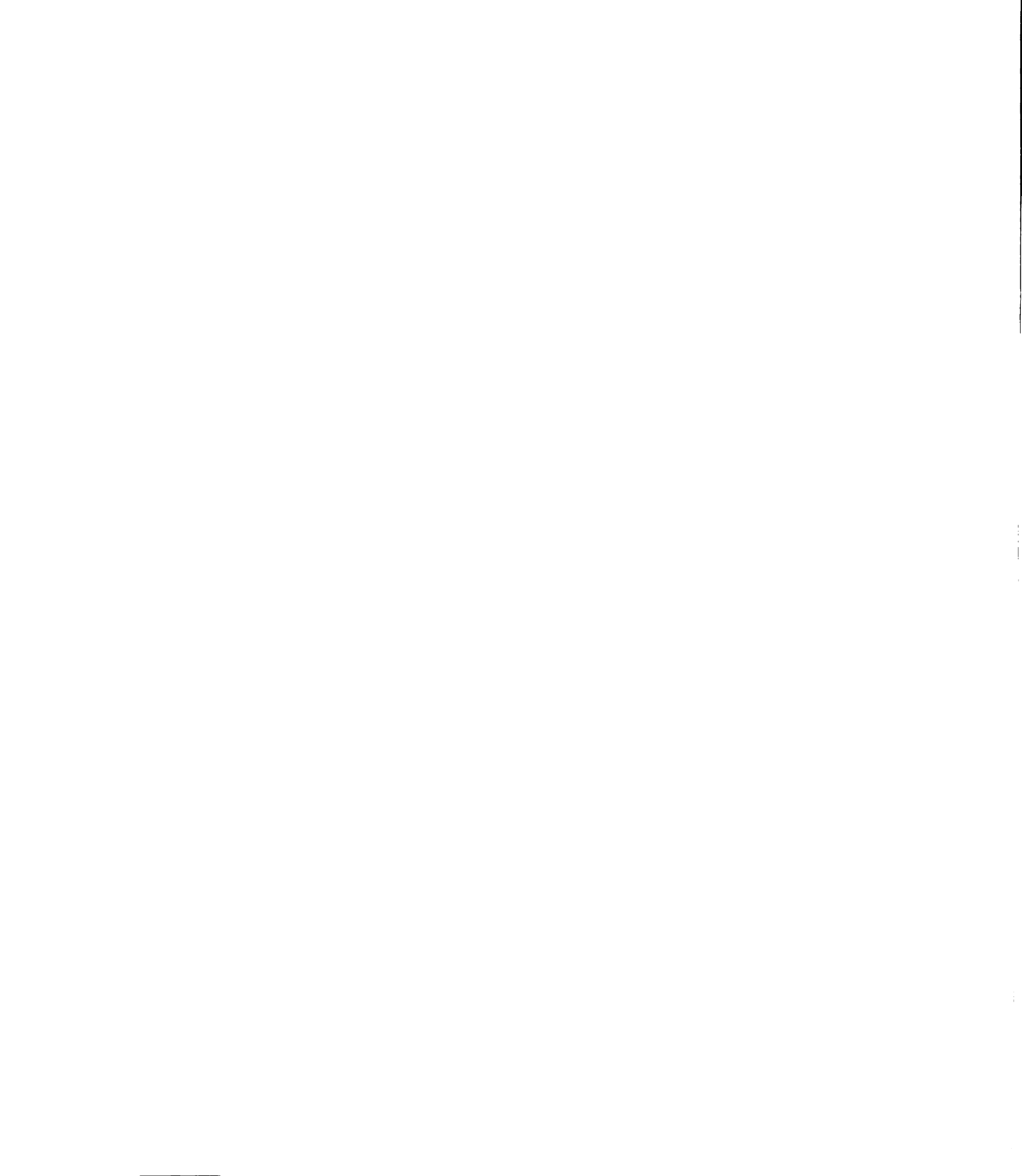
```

Algorithm 4: IPR cubese generation.

Correctness of Generate_IPR_Cubese procedure

Claim 5.3 *The Generate_IPR_Cubese procedure generates a Boolean expression in infix notation.*

Explanation: The Generate_IPR_Cubese is invoked with a post-ordered Boolean expression input stack. Line 2 checks the length of the input stack. While the stack is not empty the procedure enters lines 3-10, where it continuously removes items from the stack until an operator (AND/OR) is encountered. Line 4 performs the **Pop** command which removes the next item from the stack. If the item is not an operator then it must be a literal. The literal is **Pushed** onto the *evaluation expression stack* at line 8. If an operator is encountered by the check at line 5, then the item variable is updated with the operator value, line 6. Following the update, the **repeat-until**



loop is broken at line 10, with processing continuing to line 11. The repeat-until loop, lines 3-10, is iterated two times and loads two literals on the evaluation expression stack, due to post-order pairing of literals in the Post_Order procedure.

Lines 11 and 12 remove the top two items from the evaluation expression stack. Line 13 invokes the Perform_IPR_Operation routine with the two items from the evaluation expression stack and the operator acquired from line 6. The Perform_IPR_Operation routine converts the item arguments to IPR cubes and performs a Boolean (AND/OR) IPR operation on the two item cubes. The result of the operation is Pushed onto the evaluation expression stack (line 14). The procedure returns to line 2 and tests the size of the input stack. While the input stack size remains nonzero processing continues as previously described. When the input stack becomes empty, processing continues to line 16, where the final IPR cube *infix* Boolean expression result is retrieved from the evaluation expression stack and returned to the invoking procedure.

Analysis of Generate_IPR_Cubaset procedure

The complexity of the Generate_IPR_Cubaset is more difficult to compute than it appears. The duration of the outer **while-loop**, lines 2-15, must be determined; this duration is actually the size of the post-order input expression stack (*in_size*). The exact computation of such a value is circuit specific. However, the derivation for the worst case *in_size* approximation is presented in the following test.

Let β equal the maximum number of logic element inputs. Given the structure of a β -input MSBDD, the post-order expression size is determined by summing the number of decision node literals with the number of Boolean (AND/OR) operations encountered during the post-order traversals. For a single level ($k=1$) flattening of a typical MSBDD (N/AND, N/OR, XOR), a worst case approximation for *in_size* (input stack size) given β inputs, is computed as:

$$\begin{aligned}
in_size_{k=1} &= \underbrace{\sum_{i=1}^{\beta} i}_{\#literals} + \underbrace{\sum_{i=0}^{\beta-1} i}_{\#ANDs} + \underbrace{\beta - 1}_{\#ORs} \\
&= \frac{1}{2}\beta(1 + \beta) + \frac{1}{2}\beta(\beta - 1) + \beta - 1 \\
&= \underbrace{\frac{\beta^2 + \beta}{2}}_{\#literals} + \underbrace{\frac{\beta^2 + \beta}{2} - 1}_{\#operations} \\
&= \beta^2 + \beta - 1.
\end{aligned}$$

For multi-level circuits, the Post_Order procedure will expand the CBDD into an input stack expression whose size depends on k , the depth-accuracy parameter. Let L , the literal count for a β -input MSBDD, equal $\frac{\beta^2 + \beta}{2}$. Let C , the number of Boolean operations for a β -input MSBDD post-order expression, equal $\frac{\beta^2 + \beta}{2} - 1$. Let F , equal $L + C$ or $\beta^2 + \beta - 1$. The resulting input stack expression size can be modeled using the following recurrence relation: $in_size_k = L \times in_size_{k-1} + C$. After a few iterations of the recurrence relation the solution was determined to be

$$\begin{aligned}
in_size_k &= F && \text{for } k = 1 \\
&= L \cdot F + C && \text{for } k = 2 \\
&= L \cdot (L \cdot F + C)^{k-2} + C && \text{for } k > 2 \\
&= \frac{\beta^2 + \beta}{2} \left(\left(\frac{\beta^4 + 2\beta^3 + \beta^2}{2} - 1 \right)^{k-2} + 1 \right) - 1 && \text{for } k > 2
\end{aligned}$$

The condition on line 2 runs in constant time, $O(1)$. The Push/Pop operations at lines 4, 8, 11, 12, 14, and 16 run in constant time, $O(1)$. Lines 5-9 run in a maximum of $O(2)$ time. The Perform_IPR_Operation routine runs in $O(G)$ time, where G is the number of circuit inputs plus the number of circuit logic elements. The while-loop, lines 2-15, executes a maximum of $\frac{in_size}{2}$ times. The **repeat-until** block, lines 3-10, always runs twice, with a total time cost of $O(8)$. Lines 11-14, result in a running time of $O(1) + O(2) + O(G) + O(1) \approx O(G + 4)$. The total running time for each iteration of the while-loop, lines 2-15, is $O(1) + O(8) + O(G +$

4) $\approx O(G + 13)$. Given that the while-loop iterates $\frac{in_size_k}{2}$ times, the worst case running time of the Generate_IPR_Cubaset procedure is $O\left(\frac{in_size_k}{2} \times (G + 13)\right)$ or $O\left(\frac{1}{2} \left[\frac{\beta^2 + \beta}{2} \left(\left(\frac{\beta^4 + 2\beta^3 + \beta^2}{2} - 1 \right)^{k-2} + 1 \right) - 1 \right] \times (G + 13)\right)$, for $\beta > 1$ and $k > 2$.

Step	Input Expression Stack	Evaluation Expression Stack
0	3 2 * 1 2 3 (-2) * + * (-2)(-3)2 * + * +	-
1	2 * 1 2 3 (-2) * + * (-2)(-3)2 * + * +	3
2	* 1 2 3 (-2) * + * (-2)(-3)2 * + * +	3 2
3	1 2 3 (-2) * + * (-2)(-3)2 * + * +	(3 * 2)
4	2 3 (-2) * + * (-2)(-3)2 * + * +	(3 * 2) 1
5	3 (-2) * + * (-2)(-3)2 * + * +	(3 * 2) 1 2
6	(-2) * + * (-2)(-3)2 * + * +	(3 * 2) 1 2 3
7	* + * (-2)(-3)2 * + * +	(3 * 2) 1 2 3 (-2)
8	+ * (-2)(-3)2 * + * +	(3 * 2) 1 2 (3 * (-2))
9	* (-2)(-3)2 * + * +	(3 * 2) 1 (2 + (3 * (-2)))
10	(-2)(-3)2 * + * +	(3 * 2) ((2 * 1) + (3 * (-2) * 1))
11	(-3)2 * + * +	(3 * 2) ((2 * 1) + (3 * (-2) * 1)) (-2)
12	2 * + * +	(3 * 2) ((2 * 1) + (3 * (-2) * 1)) (-2) (-3)
13	* + * +	(3 * 2) ((2 * 1) + (3 * (-2) * 1)) (-2) (-3) 2
14	+ * +	(3 * 2) ((2 * 1) + (3 * (-2) * 1)) (-2) ((-3) * 2)
15	* +	(3 * 2) ((2 * 1) + (3 * (-2) * 1)) ((-2) + ((-3) * 2))
16	+	(3 * 2) (((-3) * 2 * 1) + (3 * (-2) * 1))
17	-	(3 * 2) + ((-3) * 2 * 1) + (3 * (-2) * 1)

Table 5.2. Output from Generate_IPR_Cubaset procedure.

The Generate_IPR_Cubaset procedure returns an evaluation stack which contains the terms of a disjoint Boolean equation that is in *sum-of-products* form. Each term is represented by an IPR cube. The full-adder carry-out bit primary output, *c_out*, produced by the Post_Order procedure, Table 5.1, is used in an example to demonstrate the conversion of post-order Boolean equations to a *sum-of-products* equation in *infix* form. Table 5.2 shows the stages of the input expression stack (*c_out*) and the evaluation expression stack as their contents are modified by the Generate_IPR_Cubaset procedure's *postfix* to *infix* conversion. If a node index to literal mapping, Table 5.1, is performed on the final evaluation expression stack output, the resulting *infix* expression for *c_out* is $ab + \bar{a}bc + \bar{a}\bar{b}c$. The final expression is composed of disjoint terms only.

5.3.1.4 Computation of Signal Probability

The *Signal Probability Computation* component is responsible for computing the signal probability for a specified node within the CBDD's graph. This task is implemented by the *Compute_Signal_Probability* procedure (Algorithm 5). Using the disjoint Boolean expression IPR cubes produced by the *Generate_IPR_Cubese*t procedure, the *Compute_Signal_Probability* procedure determines the signal probability of each cube within the IPR cubese

t and sums all IPR cube signal probabilities. The result is the signal probability for the node represented by the disjoint Boolean expression input.

```
Procedure: Compute_Signal_Probability
input      : IPR_Cubeset - IPR Cubeset
input      : SP - Signal probability array
output     : Expr_SP - Expression signal probability

1 begin
2   for  $\forall C \in IPR\_Cubese$  do
3     T_prob  $\leftarrow$  1.00
4     for idx  $\in$  1..IPR_Cube_Length do
5       EXP  $\leftarrow$  (C.exp[idx] AND mask[idx])
6       POS  $\leftarrow$  (C.pos[idx] AND mask[idx])
7       if EXP = 0 then
8         I_prob  $\leftarrow$  (POS > 0 ? SP[idx].p : SP[idx].q)
9         T_prob  $\leftarrow$  T_prob * I_prob
10      end
11    end
12    Expr_SP  $\leftarrow$  Expr_SP + I_prob
13  end
14 end
```

Algorithm 5: Signal probability computation procedure.

Correctness of Compute_Signal_Probability procedure

Claim 5.4 *The Compute_Signal_Probability procedure computes the signal probability for a disjoint Boolean expression.*

Explanation: The Compute_Signal_Probability procedure advances through the list of IPR cubes (terms) within the outer **for-loop**, lines 2-13. Line 3 initializes the cube signal probability to 1. The inner **for-loop**, lines 4-11, iterates over the entire signal variable space for the current IPR cube. The goal at this point is to determine the variables used by the cube. Variables in use have a 0 in their corresponding bit position of the cube's expansion integer. Line 5, using bitwise AND operations, determines if the variable is in use. Line 6 determines the phase of the variable using bitwise AND operations. The **if-block**, lines 7-10, is entered if the variable is in use, otherwise processing continues within the inner for-loop for the next variable. Line 8 reads the signal probability for the corresponding variable from signal probability array based on the Boolean variable's phase. Line 9 performs a multiplication of the variable's signal probability with the current cube's intermediate signal probability product. Upon termination of the inner for-loop, line 9 computes the signal probability of a cube in the following manner: $P_s(cube_k) = \prod_{v \in V_k} P_s(v)$, where V_k is the set of signals in use by $cube_k$. Line 12 performs a summation of the cube signal probabilities. Upon termination of the outer for-loop, line 12 computes the signal probability for the IPR cubeset as

$$P_s(cubeset) = \sum_{i=1}^{\# \text{ cubes}} P_s(cube_i).$$

Analysis of Compute_Signal_Probability procedure

The running time of the Compute_Signal_Probability procedure heavily depends on the size of the IPR cubeset (number of terms). In the worst case the maximum number of terms is 2^N , where N is the number of circuit inputs. The system supports a user-specified depth-accuracy (k) parameter which limits the maximum number of terms when k is less than the depth of the input circuit. Given that the circuit is composed of β -input logic elements, the worst case number of terms is approximately equal to β^{k+1} , where β is the fanin size of the logic element.

The outer for-loop, lines 2-13, will iterate a maximum of β^{k+1} times. Line 3 runs in constant time, $O(1)$. Lines 5-6 run in constant time, $O(2)$. The if-block, lines 7-10, in the worst case runs in $O(3)$ time. The duration of the inner for-loop is equal to number of logic elements plus the number of primary inputs within the circuit, designated as R . Therefore, the running time of the inner for-loop is $O(5R)$. Line 12 runs in constant time, $O(1)$. The worst case running time of the procedure is $O((5R+2) \times \beta^{k+1})$. From the analysis it's apparent that an increase in k dramatically affects the running time of the procedure.

The more difficult tasks of computing signal probability were accomplished by the Post_Order and Generate_IPR_Cubeset procedures. The tasks performed by the Compute_Signal_Probability procedure are much easier to execute. An example which demonstrates the work performed by the Compute_Signal_Probability procedure is illustrated in Table 5.3. The IPR cubeset for the carry-out bit is used as input in this example and the signal probabilities for primary inputs a, b and c , are $\frac{1}{3}, \frac{1}{4}$ and $\frac{1}{5}$, respectively.

5.3.1.5 SPCA Component Interconnection

The SPCA is an interconnection of procedures which flatten the input circuit's network to a user-selected depth and efficiently approximates signal probability for all

<i>Term</i>	<i>Term Product</i>	$P_s(\textit{Term})$
<i>ab</i>	$\frac{1}{3} \cdot \frac{1}{4}$	$\frac{1}{12}$
$\bar{a}bc$	$\frac{2}{3} \cdot \frac{1}{4} \cdot \frac{1}{5}$	$\frac{1}{30}$
$a\bar{b}c$	$\frac{1}{3} \cdot \frac{3}{4} \cdot \frac{1}{5}$	$\frac{1}{20}$
$P_s(\textit{cuberset})$		$\frac{1}{6}$

Table 5.3. Signal probability computation for full-adder carry-out bit.

internal and output nodes. Previous sections have discussed the operational and time complexity details of the individual SPCA components. The *SPCA_Interconnect* procedure, Algorithm 6, is responsible for the invocation and coordination of all SPCA component procedures. When invoked, the *SPCA_interconnect* procedure coordinates the signal probability computation for an entire CBDD with user-selectable Boolean expression flattening.

```

Procedure: SPCA_Interconnect
input      : CBDD - DAG of input circuit
input      : PI_SP - Primary input signal probabilities
input      : k - Depth_Accuracy
output     : SP_Array - Signal probability array

1 begin
2   Network_Levelization(CBDD.root, 0)
3   for  $\forall$  MSBDD  $\in$  CBDD do
4     Expr  $\leftarrow$  Post_Order(CBDD, MSBDD.Output_vertex, k, 0)
5     IPR_Cuberset  $\leftarrow$  Generate_IPR_Cuberset(Expr)
6     Sp  $\leftarrow$  Compute_Signal_Probability(IPR_Cuberset, PI_SP)
7     SP_Array[MSBDD.Output_vertex.idx]  $\leftarrow$  Sp
8   end
9 end

```

Algorithm 6: SPCA interconnection procedure

Correctness of SPCA_Interconnect procedure

Claim 5.5 *The SPCA_Interconnect procedure generates the signal probabilities for all nodes within a CBDD.*

Explanation: Conventional methods for estimating signal probability of a circuit utilize traditional BDDs. The computation involves a post-order traversal of the BDDs for each net within the circuit, resulting in a Boolean expression. Assuming *uncorrelated primary inputs* and the *zero-delay* model, the conventional methods for estimating signal probability of a net modeled by a BDD utilize Equations 5.1, 5.2 and 5.3.

$$P_s(term_j) = \prod_{i=1}^{m_j} p(x_i) \quad (5.1)$$

$$term_x \cap term_y = 0 \quad \forall x \neq y \quad (5.2)$$

$$P_s(node_k) = \sum_{j=1}^{n_k} P_s(term_j) \quad (5.3)$$

- Equation 5.1 specifies that the term signal probability is the product of the term's input signal probabilities.
- Equation 5.2 specifies that all terms are disjoint or independent of one another.
- Equation 5.3 specifies that the node signal probability is the sum of disjoint term signal probabilities.

The signal probability estimation method presented in this dissertation is correct and satisfies Equations 5.1, 5.2 and 5.3. The proposed method utilizes a CBDD with user-selectable Boolean expression flattening as opposed to a BDD. Theorem 4.3 validates the fact that a CBDD maintains a circuit's behavior. Equation 5.2 is satisfied by the application of Theorem 4.2 which validates that all paths from any decision

node x_j to any decision node x_k are unique for $j \neq k$, within an MSBDD. The generation of Boolean equations as a collection of disjoint IPR cubes is performed by the Post_Order and Generate_IPR_Cubeseq procedures. Signal probability computation for each node within the CBDD is performed by the Compute_Signal_Probability procedure which satisfies Equations 5.1 and 5.3.

The SPCA_Interconnect procedure correctly generates the signal probability for all internal and primary output nodes within a CBDD. The procedures invoked by the SPCA_Interconnect procedure are well defined and produce predictable outputs for their given inputs. The more important procedures which perform searches/traversals are based on well known search algorithms which have been proven to be consistent and correct. The overall interconnection of these procedures is logical and has been tested as a system. The system level testing results are predictable and consistent with the results of conventional BDDs; this verifies that the SPCA_Interconnect procedure correctly computes signal probability for a circuit represented by a CBDD.

Analysis of SPCA_Interconnect procedure

The running time for the SPCA_Interconnect depends on the worst case running times of all procedure invocations. Line 2, the invocation of Network_Levelization is called once. The for-loop, lines 3-7, accounts for a majority of the SPCA_Interconnect running time. The duration of the for-loop is G . Line 7 runs in constant time, $O(1)$, but is dominated by the time of the procedure invocations of lines 4-6. Using Tables 5.4 and 5.5, for $\beta > 1$ and $k > 2$ the worst case running time for the SPCA_Interconnect procedure is $O(P1 + G \times (P2 + P3 + P4))$. From observation it's apparent that the worst case running times of the SPCA_Interconnect procedures have the following ranking, $P1 \ll P2 \ll P4 \ll P3$. The ranking indicates that the SPCA_Interconnect procedure can potentially spend most of its time in the invo-

cation of the `Generate_IPR_Cubese`t procedure. This procedure is computationally expensive because the input expression stack could become very large and the literal to IPR cube conversion has a cost of $(G + 13)$ per cube. The resulting worst case running time for the `SPCA_Interconnect` procedure is approximately $O(G \times P3)$ or $O\left(\frac{G}{2} \left[\frac{\beta^2 + \beta}{2} \left(\left(\frac{\beta^4 + 2\beta^3 + \beta^2}{2} - 1 \right)^{k-2} + 1 \right) - 1 \right] \times (G + 13) \right)$

<i>Symbol</i>	<i>Meaning</i>
N	Number of circuit inputs
G	Number of circuit logic elements
R	$N + G$
β	Maximum logic element fanin
$ V $	CBDD vertex count
$ E $	CBDD edge count
k	Depth-accuracy parameter

Table 5.4. Symbol definitions.

<i>Symbol</i>	<i>Procedure</i>	<i>WorstCaseRunningTime</i>
$P1$	<code>Network_Levelization</code>	$O(V + E)$
$P2$	<code>Post_Order</code>	$O(16 \times (4\beta + 5) \times G)$
$P3$	<code>Generate_IPR_Cubese</code> t	$O\left(\frac{1}{2} \left[\frac{\beta^2 + \beta}{2} \left(\left(\frac{\beta^4 + 2\beta^3 + \beta^2}{2} - 1 \right)^{k-2} + 1 \right) - 1 \right] \times (G + 13) \right)$
$P4$	<code>Compute_Signal_Probability</code>	$O(5R + 2) \times \beta^{k+1}$

Table 5.5. Procedure running times.

5.4 Switching Activity and Power Computation

The final component of the BLAPE implementation involves the computation of switching activity and dynamic power. The complicated task of computing signal probability was performed by *Signal Probability Computation Algorithm* (SPCA).

Upon completion of the SPCA, a signal probability array is generated. The signal probability array is passed to a routine which computes switching activity using $\alpha(x) = 2 \cdot P_s(x) \cdot (1 - P_s(x))$. Switching activity is computed for each node in the network. After this computation, dynamic power is computed for each node using $P_{Switch}(x) = \frac{1}{2}\alpha(x) \cdot C_L(x) \cdot f_{CLK} \cdot V_{dd}^2$. Next, the BLAPE implementation provides a power estimate for the entire circuit using $\sum_{i=1}^N P_{Switch}(x_i)$, where N is the number of nets in the circuit.

The following examples (Figures 5.14 and 5.15) provide BLAPE generated activity and power output for both *implicit* and *mapped* structural representations (Figures 5.4 and 5.6) of the full-adder. As a means of comparison, SIS generated activity and power analysis is given in Figures 5.16 and 5.17 for the same structural specifications. SIS switching activity roundoff to two decimals places results in a 0.26% power estimate difference, when compared to BLAPE.

Node	Activity	Capacitance
c_in	0.5000	cap=11
x	0.5000	cap=11
y	0.5000	cap=11
c_out	0.5000	cap=0
sum	0.5000	cap=0
NOTc_in	0.5000	cap=6
NOTy	0.5000	cap=6
NOTx	0.5000	cap=6
t0	0.2188	cap=5
t1	0.2188	cap=5
t2	0.2188	cap=5
t3	0.2188	cap=5
_LC1_B1	0.5000	cap=2
t4	0.3750	cap=4
t5	0.3750	cap=4
t6	0.3750	cap=4
_LC2_B1	0.5000	cap=2

Power = 91.94 uW, assuming 20 MHz CLK, Vdd = 5V

Figure 5.14. BLAPE implicit structural FA activity/power estimates.

Node	Activity	Capacitance
c_in	0.5000	cap=11
x	0.5000	cap=11
y	0.5000	cap=11
c_out	0.4878	cap=1
sum	0.4851	cap=2
[236]	0.3750	cap=4
[238]	0.3750	cap=4
[234]	0.3750	cap=4
[182]	0.5000	cap=6
[181]	0.5000	cap=6
[226]	0.2188	cap=5
[183]	0.5000	cap=6
[228]	0.2188	cap=5
[230]	0.2188	cap=5
[232]	0.2188	cap=5

Power = 89.58 uW, assuming 20 MHz CLK, Vdd = 5V

Figure 5.15. BLAPE mapped structural FA activity/power estimates.

```

Node c_in  Cap. = 11 Switch Prob. = 0.50 Power = 13.8
Node x     Cap. = 11 Switch Prob. = 0.50 Power = 13.8
Node y     Cap. = 11 Switch Prob. = 0.50 Power = 13.8
Node [3019] Cap. = 0 Switch Prob. = 0.50 Power = 0.0
Node [3020] Cap. = 0 Switch Prob. = 0.50 Power = 0.0
Node NOTc_in Cap. = 6 Switch Prob. = 0.50 Power = 7.5
Node NOTy   Cap. = 6 Switch Prob. = 0.50 Power = 7.5
Node NOTx   Cap. = 6 Switch Prob. = 0.50 Power = 7.5
Node _LC2_B1 Cap. = 2 Switch Prob. = 0.50 Power = 2.5
Node _LC1_B1 Cap. = 3 Switch Prob. = 0.50 Power = 3.8
Node t0     Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t1     Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t2     Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t3     Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t4     Cap. = 4 Switch Prob. = 0.38 Power = 3.8
Node t5     Cap. = 4 Switch Prob. = 0.38 Power = 3.8
Node t6     Cap. = 4 Switch Prob. = 0.38 Power = 3.8
Total Power: 92.187500

```

Figure 5.16. SIS implicit structural FA activity/power estimates.

```

Node c_in  Cap. = 11 Switch Prob. = 0.50 Power = 13.8
Node x     Cap. = 11 Switch Prob. = 0.50 Power = 13.8
Node y     Cap. = 11 Switch Prob. = 0.50 Power = 13.8
Node [3021] Cap. = 1 Switch Prob. = 0.50 Power = 1.2
Node [3022] Cap. = 2 Switch Prob. = 0.50 Power = 2.5
Node [236]  Cap. = 4 Switch Prob. = 0.38 Power = 3.8
Node [238]  Cap. = 4 Switch Prob. = 0.38 Power = 3.8
Node [234]  Cap. = 4 Switch Prob. = 0.38 Power = 3.8
Node [182]  Cap. = 6 Switch Prob. = 0.50 Power = 7.5
Node [181]  Cap. = 6 Switch Prob. = 0.50 Power = 7.5
Node [226]  Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node [183]  Cap. = 6 Switch Prob. = 0.50 Power = 7.5
Node [228]  Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node [230]  Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node [232]  Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Total Power: 89.687500

```

Figure 5.17. SIS mapped structural FA activity/power estimates.

The BLAPE implementation uses a behavioral VHDL specification as its initial input. In an effort to provide realistic activity and power estimates for a given design, structural specifications at the gate-level are considered. Various structural specifications yield various power estimates, as shown above. BLAPE's ability to consider various structural specifications, given an initial VHDL behavioral specification, complements an HDL-based hardware design methodology by providing improved high-level power estimates.

5.5 Experimental Results

This section presents experimental results generated by the BLAPE program. BLAPE results are compared to the results generated by the Berkeley SIS power estimator tool. The Berkeley SIS tool was chosen for comparison because it is a mature VLSI development system that is considered to be the standard for switching activity and dynamic power measurement. The Berkeley SIS tool computes exact signal probability by traversing a traditional BDD with the intent of generating a 2-level disjoint Boolean equation for each net of the input circuit.

Quantities such as power (μw) and time (s) are used as a means of comparison between the two tools. The circuits involved in the tool comparison are a selection of ISCAS-85 benchmarks, MCNC Synth89 benchmarks, and other arithmetic circuits. To simplify the exchange of benchmark circuits between tools, the input circuits are BLIF-formatted gate-level specifications. For BLAPE accuracy comparisons, it is assumed that the Berkeley SIS power estimator generates exact power estimates. Benchmarking was performed on a workstation with the following configuration: Intel Pentium II 350MHz (processor), 64MB (RAM), and Linux (operating system) with 330MB swap partition.

Five experiments were performed in which power estimates of the BLAPE pro-

gram were compared to the estimates of the Berkeley SIS tool. Each experiment includes one or more benchmark circuits. Tables containing the SIS power estimate and computation time, along with the corresponding BLAPE measurements for varying depth-accuracy are given.

5.5.1 Experiment 1 : 64-Bit Adder Benchmark

In the first comparison, the BLAPE and SIS tools were used to perform power estimates for a 64-bit adder. The outcome of the comparison is given in Table 5.6. The 64-bit adder was selected because of its depth and high level of isolated reconvergent fanout.

<i>#Inputs</i>	<i>#Outputs</i>	<i>#Gates</i>	<i>#Levels</i>	<i>#Nodes</i>
128	65	315	128	4944

<i>Tool</i>	μw	<i>CPU</i>	<i>k</i>	<i>%Err</i>
SIS	3030.8	1.20s	—	—
BLAPE	3030.9	1.13s	12	0.003%

Table 5.6. 64-bit adder size/power estimates.

Subsequent runs of the BLAPE program were performed using the 64-bit adder as input. For each run, the depth accuracy parameter (k) was increased. The results of the subsequent runs are illustrated in Figures 5.18, 5.19, and 5.20. The results for the 64-bit adder indicate that for a small depth-accuracy ($k < 3$), execution times are short and BLAPE underestimates power by just 0.019%. For $3 < k < 5$, execution times are a little longer and BLAPE overestimates power by 0.089%. As k increases ($k > 5$), BLAPE starts to converge to the exact power estimate produced by SIS.

For $k = 12$, a very reasonable power estimate is achieved in about the same time as SIS takes. The time execution (Figure 5.20) reveals an exponential growth pattern, attributed to increased depth-accuracy.

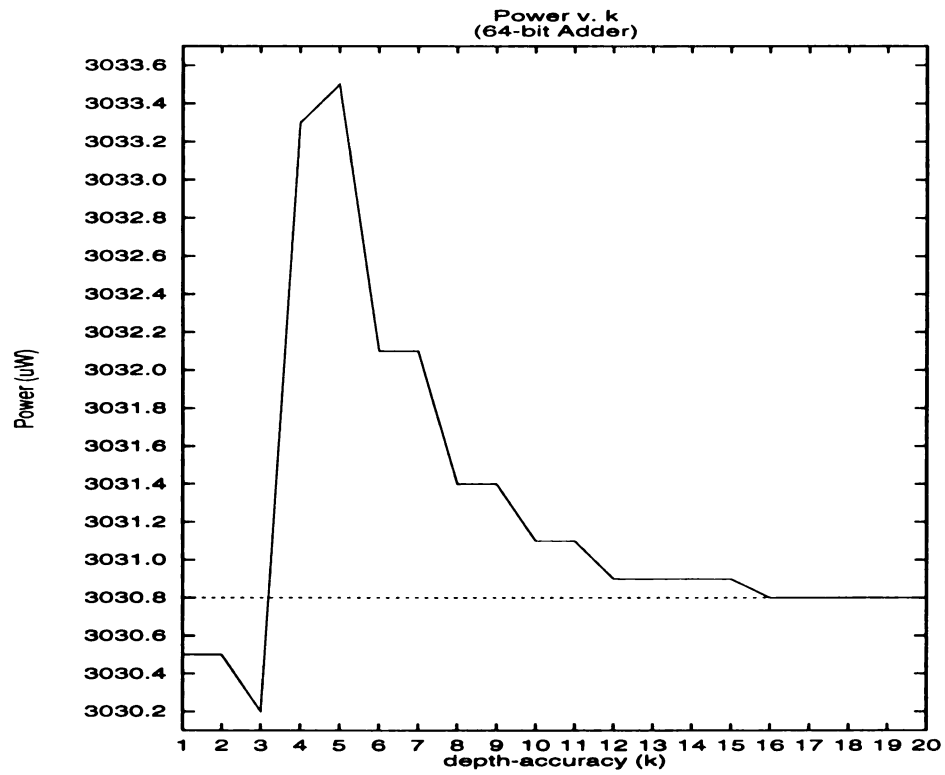


Figure 5.18. Power v. k (64-bit adder).

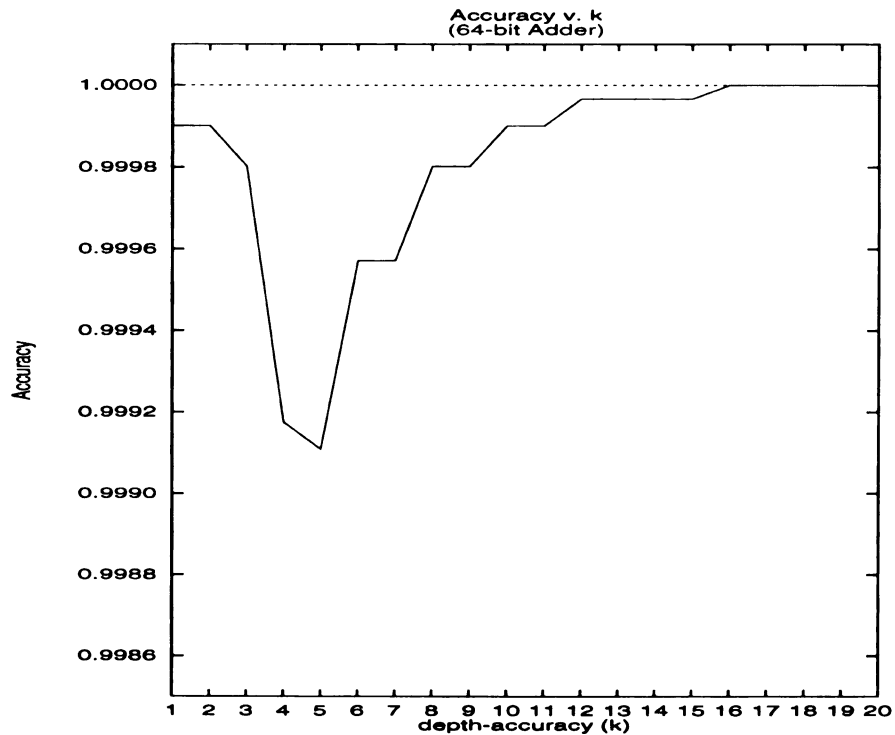


Figure 5.19. Accuracy v. k (64-bit adder).

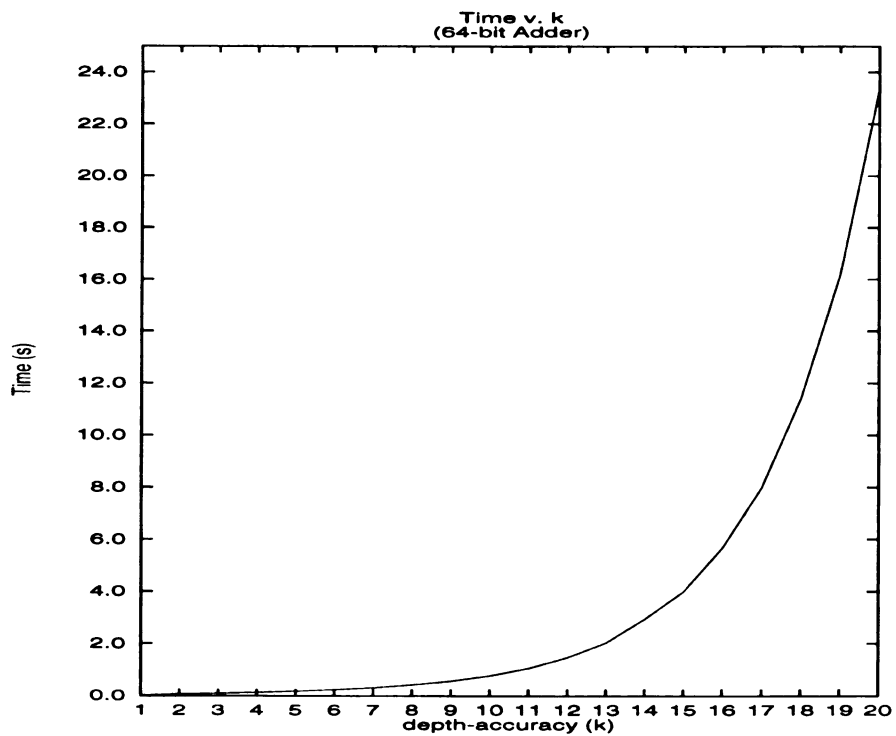


Figure 5.20. Time v. k (64-bit adder).

5.5.2 Experiment 2 : Arithmetic Benchmarks

The arithmetic benchmark suite contains adder and multiplier circuits of various sizes. The arithmetic benchmark comparisons demonstrated the same power/accuracy and performance trends observed with the 64-bit adder (Tables 5.7 and 5.8). Certain circuits, due to their depth and large internal fanouts, were not tested for $k > 5$. For the arithmetic class of circuits a depth-accuracy of $k = 1$ provided an average error of 0.19% along with an average runtime of 0.012 seconds. As k was increased the average power estimate error approached zero. For $k = 3$ there was a small increase in the average error to about 0.32%, which can be attributed to reconvergent fanout. This error is so small that it can be considered negligible.

Circuit	SIS	BLP	k = 1		k = 2		k = 3		k = 5		k = 7	
	μw	μw	μw	%Err	μw	%Err	μw	%Err	μw	%Err	μw	%Err
add8	349	349	0.20	0.20	349	0.20	349	0.17	349	0.06	349	0.00
add16	732	732	0.08	0.08	732	0.08	733	0.18	732	0.07	732	0.03
add32	1498	1498	0.02	0.02	1498	0.02	1501	0.18	1499	0.09	1498	0.04
mult4	410	410	0.02	0.02	410	0.02	411	0.32	410	0.00	410	0.02
mult8	2274	2289	0.65	0.65	2289	0.65	2292	0.78	2287	0.58		
Average			0.19		0.19		0.32		0.16		0.02	

Table 5.7. Power/Accuracy estimates of arithmetic benchmarks.

Circuit	SIS	BLAPE				
	CPU	k = 1	k = 2	k = 3	k = 5	k = 7
add8	0.01	0.01	0.01	0.01	0.01	0.01
add16	0.10	0.00	0.00	0.01	0.01	0.02
add32	0.10	0.02	0.03	0.03	0.06	0.09
mult4	0.01	0.00	0.00	0.01	0.16	3.80
mult8	1.70	0.03	0.10	0.31	10.46	

Table 5.8. Time estimates of arithmetic benchmarks.

5.5.3 Experiment 3 : ISCAS-85 Benchmarks

The ISCAS-85 benchmark suite contains a variety of ALU, control, and decoder circuits. Due to circuit depth and large internal fanouts, these circuits were not tested beyond a depth-accuracy of three. A few circuits exhibited increased error for an increase in the depth-accuracy parameter. This is attributed to reconvergent fanout occurring below the depth-accuracy level of a node and the propagation of this error to subsequent signal probability computations. The ISCAS-85 benchmark comparisons demonstrated similar accuracy/performance trends observed by the arithmetic benchmarks, but with greater average error. Given a depth-accuracy of $k = 1$ the ISCAS-85 benchmark comparison yields an average error of 3.13% along with an average runtime of 0.22 seconds (Tables 5.9 and 5.10). As k is increased the average power estimate error slowly approaches zero.

<i>Circuit</i>	<i>SIS</i>	<i>BLP</i>	$k = 1$		$k = 2$		$k = 3$	
	μw	μw	μw	%Err	μw	%Err	μw	%Err
c1355	2411	2482	2.96		2375	1.49	2442	1.29
c17	34	34	0.00		34	0.00	34	0.00
c1908	2762	2736	0.94		2732	1.08	2740	0.81
c432	1062	1009	4.94		985	7.21		
c499	1820	1811	0.46		1810	0.53	1809	0.60
c5315	12136	11915	1.82		11880	2.11	11871	2.19
c880	1655	1765	6.66		1651	0.25	1608	2.82
c2670	4459	4132	7.33		4402	1.26		
<i>Average</i>			3.13		1.74		1.28	

Table 5.9. ISCAS-85 power/accuracy benchmarks.

<i>Circuit</i>	<i>SIS</i>	<i>BLAPE</i>		
		$k = 1$	$k = 2$	$k = 3$
	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>
c1355	6.10	0.07	0.17	0.56
c17	0.00	0.00	0.00	0.00
c1908	1.80	0.05	0.23	0.99
c432	2.80	0.01	5.23	
c499	2.60	0.03	0.08	0.29
c5315	17.30	1.36	3.71	13.07
c880	1.50	0.03	0.09	0.69
c2670	1.80	0.23	0.61	

Table 5.10. ISCAS-85 time benchmarks.

5.5.4 Experiment 4 : Nonredundant ISCAS-85 Benchmarks

The nonredundant ISCAS-85 benchmark suite contains a variety of ALU, control, and decoder circuits. These circuits are functionally equivalent to the regular ISCAS-85 benchmark circuit suite, except all redundant logic has been removed from the designs. A few circuits exhibited increased error for a larger depth-accuracy due to reconvergent fanout. Because of circuit depth and large internal fanouts the circuits were not tested beyond a depth-accuracy of three. The nonredundant ISCAS-85 benchmark comparisons demonstrated the same power/accuracy and performance trends observed by the ISCAS-85 benchmarks, but the average error for $k = 1$ was smaller due to less logic and less internal reconvergence. Given a depth-accuracy of $k = 1$ the nonredundant ISCAS-85 benchmark comparison yields an average error of 2.29% along with an average runtime of 0.43 seconds (Tables 5.11 and 5.12). As the depth-accuracy is increased the average power estimate error slowly approaches zero.

<i>Circuit</i>	<i>SIS</i>		<i>BLAPE</i> $k = 1$		$k = 2$		$k = 3$	
	μw	μw	μw	%Err	μw	%Err	μw	%Err
c1355nr	2384.4	2462.7	3.00		2357	1.00	2398	1.00
c1908nr	3400.6	3365.9	1.00		3363.5	1.00	3369.6	1.00
c2670nr	3814.3	3772.4	1.00		3759.8	1.00	3757.1	1.00
c432nr	911.8	857.8	6.00		857.8	6.00	875.4	4.00
c499nr	1628.8	1639.8	1.00		1639.8	1.00	1635.1	0.00
c5315nr	11745.4	11510.5	2.00		11484	2.00	11471.3	2.00
c7552nr	13975	13690.2	2.00		13682.8	2.00	13674.6	2.00
<i>Average</i>			2.29		2.00		1.57	

Table 5.11. Nonredundant ISCAS-85 power/accuracy benchmarks.

<i>Circuit</i>	<i>SIS</i>	<i>BLAPE</i>		
		$k = 1$	$k = 2$	$k = 3$
	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>
c1355nr	5.60	0.06	0.14	0.40
c1908nr	2.30	0.12	0.57	13.71
c2670nr	2.30	0.16	0.37	0.83
c432nr	2.40	0.01	0.54	1.87
c499nr	2.40	0.01	0.03	1.58
c5315nr	26.30	0.81	2.41	7.02
c7552nr	2.20	1.88	5.39	20.51

Table 5.12. Nonredundant ISCAS-85 time benchmarks.

5.5.5 Experiment 5 : MCNC Synth89 Benchmarks

The Synth89 benchmark suite contains various multi-level circuits which control, compare and decode. The Synth89 benchmark comparisons demonstrated the same power/accuracy and performance trends observed by the previous benchmarks, but the average error and average runtime were smaller due to less internal reconvergence and smaller circuit size. Tables 5.13 and 5.14 show the power/accuracy and performance comparisons with the Berkeley SIS tool.

Circuit	SIS		BLAPE		k = 1		k = 2		k = 3		k = 5		k = 7	
	μw	%Err	μw	%Err	μw	%Err	μw	%Err	μw	%Err	μw	%Err	μw	%Err
adr4	300.5	0.20	301.1	0.20	301.2	0.23	301.2	0.23	301.2	0.23	300.6	0.03	300.5	0.00
alu1	172.7	0.06	172.6	0.06	172.5	0.12	172.5	0.12	172.7	0.00	172.7	0.00	172.7	0.00
alu2	580.0	0.26	578.5	0.26	578.3	0.29	578.3	0.29	578.6	0.24	578.8	0.21	578.8	0.21
alu3	519.6	0.27	518.2	0.27	518.4	0.23	518.4	0.23	518.6	0.19	518.2	0.27	518.3	0.25
9symml	1286.4	1.80	1263.3	1.80	1263.3	1.80	1263.3	1.80	1266.8	1.52	1274.1	0.96		
alu2-2	1673.3	0.57	1663.7	0.57	1654.5	1.12	1654.5	1.12	1643.3	1.79	1647.0	1.57		
alu4	2881.1	2.79	2800.7	2.79	2800.5	2.80	2800.5	2.80	2800.0	2.81				
b1	76.7	0.26	76.5	0.26	76.5	0.26	76.5	0.26	76.5	0.26	76.5	0.26	76.5	0.26
c8	1095.9	0.39	1091.6	0.39	1091.6	0.39	1091.6	0.39	1091.5	0.40	1091.7	0.38	1091.8	0.37
cc	412.0	0.41	410.3	0.41	410.3	0.41	410.3	0.41	410.3	0.41	410.3	0.41	410.3	0.41
cht	1216.2	0.06	1216.9	0.06	1216.9	0.06	1216.9	0.06	1216.3	0.01	1216.2	0.00	1216.2	0.00
cm138a	156.7	0.38	156.1	0.38	156.1	0.38	156.1	0.38	156.1	0.38	156.1	0.38	156.1	0.38
cm150a	364.2	0.14	364.7	0.14	364.7	0.14	364.7	0.14	361.2	0.82	363.8	0.11	363.9	0.08
cm162a	192.4	0.26	191.9	0.26	191.9	0.26	191.9	0.26	191.9	0.26	191.9	0.26	192.0	0.21
cm163a	202.8	0.49	201.8	0.49	201.8	0.49	201.8	0.49	201.9	0.44	202.2	0.30	202.2	0.30
comp	785.6	2.51	805.3	2.51	805.3	2.51	805.3	2.51	804.1	2.35	805.4	2.52		
lal	891.2	0.52	886.6	0.52	886.7	0.50	886.7	0.50	886.7	0.50	886.1	0.57	886.1	0.57
pair	8058.0	1.60	8186.9	1.60	8001.3	0.70	8001.3	0.70						
term1	2663.2	0.76	2642.9	0.76	2642.7	0.77	2642.7	0.77	2642.6	0.77				
tlarge	4980.4	0.23	4968.7	0.23	4968.7	0.23	4968.7	0.23	4969	0.23				
ttt2	1542.5	0.62	1532.9	0.62	1532.9	0.62	1532.9	0.62	1534.9	0.49	1535.4	0.46	1535.4	0.46
x1	2353.5	0.47	2342.5	0.47	2340.2	0.57	2340.2	0.57	2337.8	0.67	2340.1	0.57	2340.3	0.56
x4	3008.0	1.09	2975.3	1.09	2975.3	1.09	2975.3	1.09	2976.6	1.04	2976.4	1.05	2981.6	0.88
z4ml	355.4	0.00	355.4	0.00	355.4	0.00	355.4	0.00	355.5	0.03	355.5	0.03	355.4	0.00
duke2	4236.7	0.65	4209.3	0.65	4209.0	0.65	4209.0	0.65	4210.2	0.63	4213.8	0.54	4215.9	0.49
e64	10800.7	0.11	10788.8	0.11	10788.8	0.11	10788.8	0.11	10788.8	0.11	10788.8	0.11	10788.8	0.11
o64	644.8	0.88	639.1	0.88	639.1	0.88	639.1	0.88	642.9	0.29				
vg2	1157.1	0.30	1153.6	0.30	1155.5	0.14	1155.5	0.14	1156.4	0.06	1156.7	0.03	1156.7	0.03
Average		0.66		0.66		0.63		0.63		0.62		0.47		0.27

Table 5.13. MCNC Synth89 power/accuracy benchmarks.

	SIS	BLAPE				
		$k = 1$	$k = 2$	$k = 3$	$k = 5$	$k = 7$
<i>Circuit</i>	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>	<i>CPU</i>
adr4	0.90	0.00	0.00	0.00	0.01	0.01
alu1	0.10	0.00	0.00	0.00	0.00	0.00
alu2	0.10	0.00	0.00	0.01	0.01	0.01
alu3	0.10	0.00	0.00	0.01	0.01	0.02
9symml	0.10	0.01	0.02	0.06	7.00	
alu2-2	0.10	0.03	0.08	0.21	13.69	
alu4	0.50	0.09	0.25	1.24		
b1	0.00	0.00	0.00	0.00	0.00	0.00
c8	0.10	0.01	0.02	0.02	0.03	0.03
cc	0.00	0.00	0.00	0.00	0.00	0.00
cht	0.10	0.01	0.02	0.03	0.03	0.03
cm138a	0.00	0.00	0.00	0.00	0.00	0.00
cm150a	0.10	0.00	0.01	0.01	0.03	0.08
cm162a	0.00	0.00	0.00	0.00	0.00	0.00
cm163a	0.00	0.00	0.00	0.00	0.00	0.00
comp	0.00	0.01	0.02	0.03	0.35	
lal	0.10	0.01	0.01	0.02	0.03	0.03
pair	2.30	0.46	1.18			
term1	0.10	0.03	0.09	0.20		
tlarge	0.50	0.10	0.38	0.71		
ttt2	0.10	0.01	0.03	0.04	0.07	0.08
x1	0.10	0.03	0.06	0.10	0.24	0.33
x4	1.10	0.05	0.10	0.18	0.57	0.89
z4ml	0.10	0.00	0.00	0.01	0.02	0.02
duke2	0.10	0.06	0.15	0.29	0.91	1.74
e64	0.20	0.08	0.10	0.11	0.11	0.11
o64	0.10	0.01	0.06	8.47		
vg2	0.01	0.01	0.02	0.04	0.06	0.06

Table 5.14. MCNC Synth89 time benchmarks.

5.5.6 Remarks

BLAPE supports a selective or user-specified power/accuracy. The results of the five experiments indicate that BLAPE provides a very reasonable power estimate for a depth-accuracy of $k = 1$. As the depth-accuracy is increased the average error in the power estimates decreases. The trends indicate that for large k the average error will approach zero (Equation 5.4), with an increase in runtime for deep circuits containing a large number of internal fanouts. The power/accuracy trend for all benchmarks is illustrated in Figure 5.21.

The BLAPE system provides an approximation of signal probability. BLAPE supports user-selectable accuracy via the depth-accuracy parameter. The behavior of

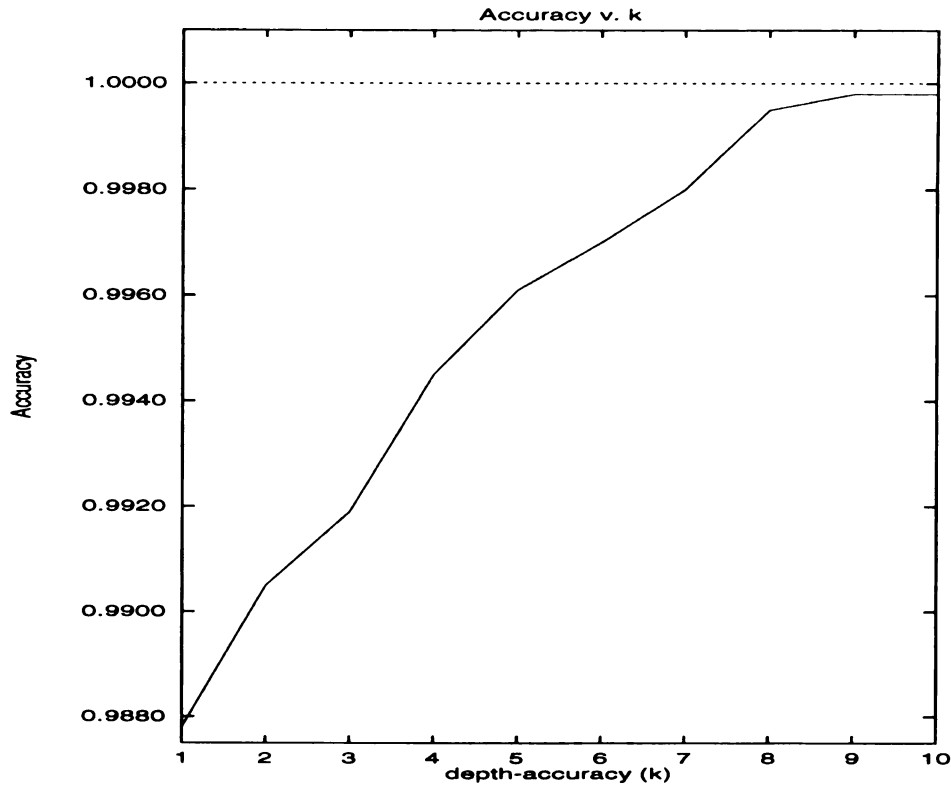


Figure 5.21. Average accuracy v. k (All benchmarks).

BLAPE's accuracy is rationalized by the facts given below and demonstrated in the previously illustrated power/accuracy and time benchmark graphs.

BLAPE's Signal Probability Computation Accuracy

- The signal probability of nodes represented by Boolean equations (*sum-of-products*) equals the sum of their disjoint term signal probabilities, given by Equation 5.3.
- Nodes represented by 2-level Boolean expressions, where $x_i \in \{PI\}$ such that PI is the set of primary inputs, yield exact signal probability.
- Nodes represented by n -level Boolean expressions, where $x_i \in \{PI, IS\}$ such that IS is the set of internal signals, yield approximate signal probability due to reconvergent fanout.

- As nodal equations are reduced in depth, approaching a 2-level expression, the error in their signal probability approximation approaches zero, described by equation 5.4.

$$\lim_{k \rightarrow L_{max}} \frac{Error}{Exact} = \frac{Approx - Exact}{Exact} \rightarrow 0 \quad (5.4)$$

The parameters in Equation 5.4 include 1) L_{max} , the circuit depth, 2) k , the depth-accuracy parameter, 3) $Approx$, BLAPE's signal probability approximation, and 4) $Exact$, the exact (SIS) signal probability. An additional feature of the BLAPE implementation is its support of large circuits. Circuits which SIS was not able to process, due to memory limitations, were handled by BLAPE in a reasonable time (Table 5.15)

BLAPE				$k = 1$		$k = 2$		$k = 3$	
<i>Circuit</i>	<i>#Inputs</i>	<i>#Outputs</i>	<i>#Gates</i>	μw	<i>CPU</i>	μw	<i>CPU</i>	μw	<i>CPU</i>
c3540	50	22	1669	5445	0.19	5448	0.96	5403	13.92
c3540nr	50	22	1594	5713	0.36	5696	1.28	5685	3.34
c7552	207	108	3512	13397	1.02	13462	3.46	13581	16.56

Table 5.15. Benchmarks not available in SIS.

BLAPE's ability to handle larger circuits is due to the CBDD's compact behavior/structure representation. Reasonable accuracy is achieved at $k = 3$. When analyzing all circuits, BLAPE's average error is just 1.22% to 0.21%, for $k = 1$ to 7. BLAPE achieves a dramatic improvement when compared to behavioral-level activity and power estimation tools. The error found in most behavioral-level power estimators about 10 – 12 percent on average and about 80% in the worst reported case. The approach taken by BLAPE is to analyze the high-level structural components from a logic perspective, by generating the disjoint Boolean equation for each of the circuit's internal and output signals. BLAPE allows the user to perform a trade-off of time

versus accuracy. A less accurate estimate can be computed fairly quick but an exact estimate can take longer, depending on the circuit size and the number of inputs.

CHAPTER 6

Behavioral-Level Visualization of Switching Activity

This chapter introduces a new tool for the visualization of switching activity in CMOS circuits. The approach presented consists of analyzing post-mortem data collected by the simulation of switching functions. An estimation of the switching activity for each circuit partition is captured in one of two views. This tool serves as a switching activity profiler, which illuminates a circuit's activity hot-spots and provides the locations where power dissipation minimization techniques can best be applied. The new tool allows the designer of CMOS circuits to visualize the switching activity on a partitioned basis. The illuminated areas should be considered first in the application of power minimization techniques.

6.1 Activity Viewer Tool

The Activity Viewer tool is a new mechanism which provides a partitioned and color-based viewing/profiling of the activity hot-spots within a digital circuit. Previous switching activity estimation tools used the traditional tabular representation form [29, 32, 50, 51] while others used the traditional two-axis graph approach as [19, 24, 37,

52]. The presentation of the results in the aforementioned references can be greatly improved where spatial/locality activity inter-relationships can also be captured. The Activity Viewer tool is an improvement to the traditional circuit activity estimate viewing approaches.

The Activity Viewer tool is a more effective means of presenting circuit activity estimates. The use of color-based activity coding combined with partitioning provides a more insightful depiction of how the circuit behaves in response to input stimuli. Additionally, the partitioning aspect of the tool gives a clear indication of how the internal logic partitions are inter-related with respect to activity and input. For a mapping of color to switching activity (Figure 6.1). (Color results have been reproduced in black and white for this dissertation). The Activity Viewer tool provides two






Color	Activity
	$0.8 < E_{sw}(g) < 1.0$
	$0.5 < E_{sw}(g) < 0.7$
	$0.3 < E_{sw}(g) < 0.4$
	$0.1 < E_{sw}(g) < 0.2$
	No Activity

Figure 6.1. Activity color mapping.

views: the Bar and Level Views, illustrated by Figures 6.2 and 6.3 respectively, which were generated with artificial partition and activity data. The Bar View provides a color-based two-axis graph of the circuit's switching activity on a partition basis. As the input stimulus changes, the Bar View captures the activity of each partition. For

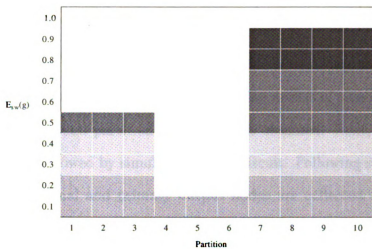


Figure 6.2. Bar view.

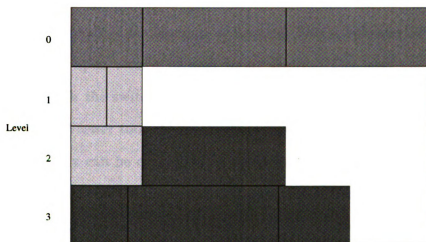


Figure 6.3. Level view.

multi-level circuit designs, the Level View, illustrated in Figure 6.3, captures level-based partitioned switching activity behavior. This view allows the circuit designer to isolate and inspect the activity profile of a circuit's network on a per level basis.

6.1.1 Input Transformation Process

The input to the Activity Viewer tool begins with a gate-level circuit netlist described in the Berkeley Logic Interchange Format (BLIF). The netlist is passed through the BLAPE tool where level and partition information is obtained. Next, input vectors are read into BLAPE followed by simulation of the circuit. Following simulation, the outputs of selected internal and primary output nodes are collected and passed to an intermediate file. This intermediate file is passed to the **mkactdat** tool, which performs switching activity estimation (simulation) assuming the zero delay model and input data independence. Each switching activity estimate is computed by estimating the average change of the selected nodes' simulation outputs. The activity estimate is based on a block of ten simulation outputs. This simulation output block is repeatedly updated upon receipt of next input vector. The level and partition information, along with the switching activity measurements, are used as the final input to the Activity Viewer tool. Once the final input data file is read, the viewing of switching activity can be controlled by using the **Play**, **Stop**, **Forward**, and **Reverse** buttons located on the control dialog. The menu bar provides options to features which allow the user to toggle between the Bar and Level views, as well as to copy the current view to the Window's clipboard memory. The latter feature serves as a window snapshot function and was used to generate the figures in this section.

6.1.2 Activity Viewer Results

A 4-bit Booth multiplier circuit can be used to illustrate the utility of the Activity Viewer tool. The Booth multiplier is a two's complement array multiplier which does not require recoding for the final two's complement result. The Booth multiplier is composed of controlled add/subtract/shift (CASS) subcircuits. Each row of the Booth array is headed by a CTRL subcircuit, which controls the CASS subcircuits.

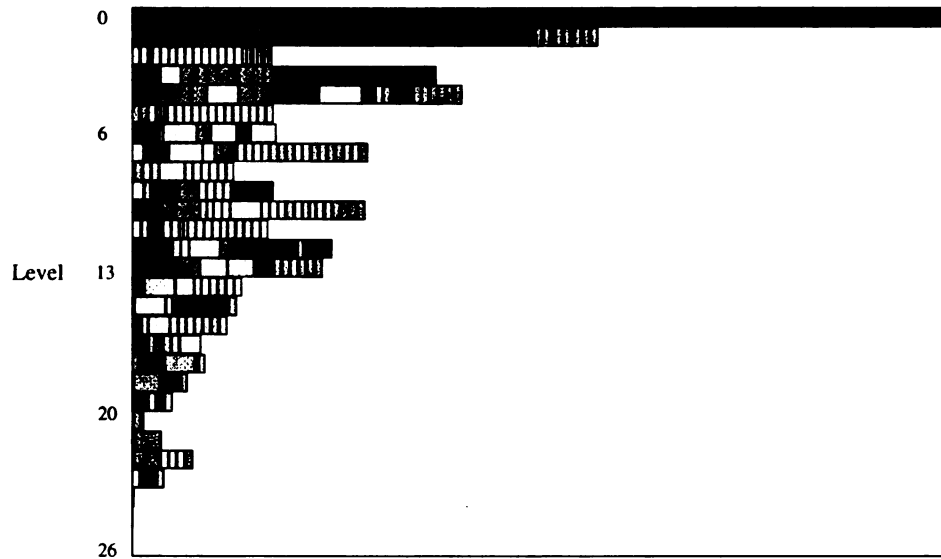


Figure 6.4. Booth multiplier(4-bit) level view.

After transforming the Booth multiplier circuit into a BLIF netlist, the final circuit resulted in a 27 level structure, illustrated by the Level View in Figure 6.4. The selected partitions consists of the basic array elements: the CTRL and CASS subcircuits. The above view displays low switching activity due to the small change in the input vectors. The low switching activity is due to minimal changes in the logic of the CTRL and CASS subcircuits caused by infrequent changes in the sign and number of 1's in the input vectors. The Level View indicates very well the relative size of each partition and the level or distance from the primary inputs. The relative size of the partitions are preserved by extending the width of each partition. The width of each partition is computed by summing the number of gates in the individual sums of products.

Figure 6.5 provides a Bar View of the Booth Multiplier, where partition size and location are ignored. Each partition is placed on the horizontal axis and it's switching activity for the given input block is indicated by color intensities. The color mappings are specified in Figure 6.1. The Bar View of the Booth multiplier is consistent with the Level View, displaying low circuit activities for the specified

input vectors. The switching activity for each partition of the level view matches the activity of the corresponding partition in bar view, but the level or distance from the primary inputs is considered.

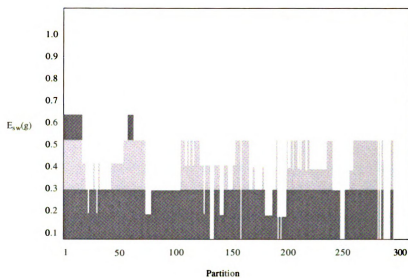


Figure 6.5. Booth multiplier(4-bit) bar view.

6.2 Future Work and Summary

Currently, the Activity Viewer tool is a Microsoft Windows based application program providing only two views. Future improvements may include the addition of new views that provide the display of total circuit switching activity and power, along with zoom in/out capabilities to isolate important areas of large circuits which have lost their viewing definition. Most importantly, the automation of the process of transforming the original circuit's file format to the final partition/activity file format can be considered.

The Activity Viewer tool is a new visualization tool which provides color-based, partitioned-level viewing of switching activity in CMOS circuits. The tool provides

the circuit designer with switching activity profiling capability which illuminates the potential power-hungry hot-spots of the design.

CHAPTER 7

Conclusions

A methodology for behavioral-level switching activity estimation in CMOS circuits has been presented. This chapter summarizes the contributions and future work of this research.

7.1 Contributions

The research presented in this dissertation addresses the problem of computing switching activity at the behavioral-level for CMOS circuits. The accurate computation of switching activity provides improved dynamic power estimates. An extensive literature review was given detailing the background and importance of the research. Additionally, an overview of behavioral and structural representations of switching functions was presented. A formal problem definition along with its solution and associated constraints/restrictions was given. The problem's solution was decomposed into a set of tasks. Each task was described by an objective along with its implemented approach.

The contributions of this research are the following:

- A new decision diagram, the Connective Binary Decision Diagram (CBDD) has been introduced. The CBDD is a graph-based behavioral and structural rep-

resentation for digital circuits. The CBDD advantages include maintenance of a circuit's structural and connective properties and linear growth. When compared to traditional BDD implementations, the CBDD demonstrated an average reduction in size of more than one order of magnitude for certain benchmark circuits.

- A set of procedures and a systematic approach for visiting each node of the CBDD's graph and generating a disjoint Boolean equation in *infix* notation was developed. The procedures were verified using a series of test circuits which generated predictable CBDDs that have known disjoint Boolean equations.
- A technique for computing signal probability was developed. The technique results in a signal probability approximation in which estimation error decreases as a depth-accuracy parameter is increased.
- A new methodology was developed for computing the switching activity and dynamic power dissipation of behavioral-level digital circuit designs described in VHDL. The implementation, called BLAPE, transforms a VHDL specification into a set of Boolean expressions. Then, the application of structure supports the computation of realistic switching activity and dynamic power estimates.
- A series of benchmark experiments were performed to validate the methodology and highlight the accuracy and performance of the BLAPE implementation as compared to the Berkeley SIS power estimator. Experiments were performed using circuits selected from an arithmetic circuit suite and the ISCAS-85, nonredundant ISCAS-85, and MCNC Synth89 benchmark suites.
- A new visualization tool for profiling/viewing the power-hungry activity hotspots within a circuit was developed. This tool is beneficial to circuit designers

because it identifies the location and switching activity of circuit nodes using various graphical views.

7.2 Future Work

The BLAPE implementation provides improved accuracy for high-level switching activity and power estimation. Future improvements and considerations in the following areas will improve BLAPE's performance, accuracy, and usefulness.

- The Connective Binary Decision Diagram (CBDD) should be more intelligent. The CBDD should be able to recognize redundant logic and not replicate this logic each time it is encountered. This improvement will produce a more compact graph, yielding smaller post-ordered Boolean equations and improving the performance.
- BLAPE should provide support for sequential circuits. The modeling of sequential circuits require additional considerations and modifications to the CBDD. This improvement will increase the usefulness of the BLAPE implementation.
- Unit/Real delay models are needed to give more realistic power estimates. These delay models will yield longer program simulation runtimes, but offer switching activity and power estimates which are consistent with the results of circuit-level simulators.
- Improved memory management for the BLAPE implementation is necessary. More efficient use of memory resources will allow the analysis of larger circuit designs.

7.3 Impact of Contributions

The research presented in this dissertation provides a solution to the problem of accurately estimating switching activity and dynamic power for high-level circuit designs described by VHDL behavioral specifications. The technique provides a significant improvement to existing behavioral-level activity and power estimators. The estimates generated by existing high-level techniques contain 10 to 12 percent error on average, with some estimates containing as much as 80 percent error. The application of the proposed technique, the BLAPE program, allows user-selectable accuracy of switching activity estimation, at the expense of time and memory resources. During a benchmark comparison containing 49 circuits, with the Berkeley SIS power estimator, the BLAPE program power estimates contained 1.22% average error for a depth-accuracy of $k = 1$ and 0.21% average error for $k = 7$.

The BLAPE program combined with the use of the Activity Viewer tool provides an effective means of activity/power profiling at the behavioral-level. The combination of the two tools identifies high-activity and power-hungry hot-spots within a circuit design. The ability to estimate switching activity and dynamic power at the behavioral-level will improve the quality of integrated circuits by providing early warning of power problems and reducing design time and cost.

APPENDICES

APPENDIX A

Definitions and Formulas

Definition A.1 *Glitch Activity:* The portion of the switching activity due multiple gate output transitions in response to an input transition.

Definition A.2 *Reconvergent fanout nodes:* Circuit nodes that receive inputs from two paths that fanout from some other circuit node.

Definition A.3 *Non-decomposable FSM:* An FSM is said to be non-decomposable when every state of the machine is reachable from every other state in a finite number of cycles.

Definition A.4 *Signal Probability $P_s(x)$:* The probability that the signal at node x evaluates to logic one [53].

Definition A.5 *Transition Probability $P_T(x)$:* The probability that the logic signal at node x experiences a change in its logic state [53].

Switching Activity Derivation:

$$\begin{aligned} E_{sw}(g) &= P_{0 \rightarrow 1}(g) + P_{1 \rightarrow 0}(g) \\ &= P_s^t(\bar{g}) \cdot P_s^{t+1}(g) + P_s^t(g) \cdot P_s^{t+1}(\bar{g}) \\ &= P_s(\bar{g}) \cdot P_s(g) + P_s(g) \cdot P_s(\bar{g}) \\ &= 2 \cdot P_s(g) \cdot P_s(\bar{g}) \\ &= 2 \cdot P_s(g) \cdot (1 - P_s(g)) \end{aligned}$$

Definition A.6 *Spatial Correlation:* Two logic signals x and y are spatially correlated if in the same time slot the state of one signal depends on the state of the other.

Definition A.7 *Temporal Correlation:* The dependence of the current state of a signal on its previous logic value.

Definition A.8 *Boolean difference* $\left(\frac{\partial y}{\partial x}\right) = y|_{x=1} \oplus y|_{x=0} = y(x) \oplus y(\bar{x})$.

Definition A.9 *Equilibrium probability:* If $x(t)$ is a logic signal (switching between 0 and 1), then its equilibrium probability is defined as $P(x) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{+T/2} x(t) dt$

Definition A.10 *Transition Density:* The average number of transitions per unit time experienced by a circuit node [53].

Remark: Then density provides an effective measure of switching activity in logic circuits in the presence of any delay model.

Remark: If a logic signal $x(t)$ makes $n_x(T)$ transitions in a time interval of length T , then the transition density of $x(t)$ is defined as $D(x) := \lim_{T \rightarrow \infty} \frac{n_x(T)}{T}$.

Remark: If all correlations are ignored, so that the input signals are independent of one another in both space and time then the signals are spatio-temporal independent, and the transition density is given by $D(y) = \sum_{i=1}^n P\left(\frac{\partial y}{\partial x_i}\right) \cdot D(x_i)$.

Remark: The relationship between transition density and transition probability is given by $D(x) \geq \frac{P_t(x)}{T}$.

Definition A.11 *Articulation Point:* A node whose removal disconnects the graph.

Definition A.12 The Boolean operators '+', '*', '-', represent the logic disjunction (OR), conjunction (AND), and inversion (NOT), respectively. By default, '*' may be denoted by a space.

Definition A.13 *Primary inputs are Boolean variables of a circuit that depend on no other variables.*

Definition A.14 *Primary outputs are Boolean variables on which no other variable(s) depends.*

Definition A.15 *The depth of a circuit is the maximum number of nodes between any primary input and any primary output..*

Definition A.16 *Flattening or Collapsing is the action of reducing the circuit's depth to a desired level.*

APPENDIX B

Application of BLAPE to 4-bit Booth Multiplier

This appendix presents a stage by stage view of the BLAPE implementation when applied to a 4-bit Booth multiplier design. The initial input specification is in VHDL, using the behavioral model for some components, and the structural VHDL model for the high-level design. The representations for each intermediate stage are given.

B.1 High-Level VHDL Specification Input

```
ENTITY cass IS
  port (pin, cin, ain, h, d : IN BIT;
        pout, cout       : OUT BIT);
END cass;

ARCHITECTURE cass_arch OF cass IS
BEGIN
  pout <= pin XOR (ain AND h) XOR (cin AND h);
  cout <= ((pin XOR d) AND ((ain OR cin) OR (ain AND cin)));
END cass_arch;

.

ENTITY ctrl IS
  port (x2, x1 : IN BIT;
        h, d   : OUT BIT);
END ctrl;
```

```

ARCHITECTURE ctrl_arch OF ctrl IS
BEGIN
  h <= x2 XOR x1;
  d <= x2 AND ( NOT(x1) );
END ctrl_arch;

```

```

ENTITY booth4 IS
  port (a, x : IN BIT_VECTOR(3 DOWNT0 0);
        p : OUT BIT_VECTOR(7 DOWNT0 0));
END booth4;

```

```

ARCHITECTURE network OF booth4 IS

```

```

COMPONENT ctrl
  port (x2, x1 : IN BIT;
        h, d : OUT BIT);
END COMPONENT;

```

```

COMPONENT cass
  port (pin, cin, ain, h, d : IN BIT;
        pout, cout : OUT BIT);
END COMPONENT;

```

```

SIGNAL p : BIT_VECTOR(34 DOWNT0 0);
SIGNAL c : BIT_VECTOR(34 DOWNT0 0);
SIGNAL h : BIT_VECTOR(4 DOWNT0 0);
SIGNAL d : BIT_VECTOR(4 DOWNT0 0);
SIGNAL zero : BIT;

```

```

BEGIN

```

```

  zero <= '0';

```

```

  ctrl_4 : ctrl port MAP ( zero, x(3), h(4), d(4) );
  ctrl_3 : ctrl port MAP ( x(3), x(2), h(3), d(3) );
  ctrl_2 : ctrl port MAP ( x(2), x(1), h(2), d(2) );
  ctrl_1 : ctrl port MAP ( x(1), x(0), h(1), d(1) );
  ctrl_0 : ctrl port MAP ( x(0), zero, h(0), d(0) );

```

```

  cass_0 : cass port MAP (zero, zero, a(0), h(4), d(4), p(0), c(0) );
  cass_1 : cass port MAP (zero, c(0), a(1), h(4), d(4), p(1), c(1) );
  cass_2 : cass port MAP (zero, c(1), a(2), h(4), d(4), p(2), c(2) );
  cass_3 : cass port MAP (zero, c(2), a(3), h(4), d(4), p(3), c(3) );
  cass_4 : cass port MAP (zero, c(3), zero, h(4), d(4), p(4), c(4) );
  cass_5 : cass port MAP (zero, zero, a(0), h(3), d(3), p(5), c(5) );
  cass_6 : cass port MAP (p(0), c(5), a(1), h(3), d(3), p(6), c(6) );
  cass_7 : cass port MAP (p(1), c(6), a(2), h(3), d(3), p(7), c(7) );

```

```

cass_8 : cass port MAP (p(2), c(7), a(3), h(3), d(3), p(8), c(8) );
cass_9 : cass port MAP (p(3), c(8), zero, h(3), d(3), p(9), c(9) );
cass_10 : cass port MAP (p(4), c(9), zero, h(3), d(3), p(10), c(10) );
cass_11 : cass port MAP (zero, zero, a(0), h(2), d(2), p(11), c(11) );
cass_12 : cass port MAP (p(5), c(11), a(1), h(2), d(2), p(12), c(12) );
cass_13 : cass port MAP (p(6), c(12), a(2), h(2), d(2), p(13), c(13) );
cass_14 : cass port MAP (p(7), c(13), a(3), h(2), d(2), p(14), c(14) );
cass_15 : cass port MAP (p(8), c(14), zero, h(2), d(2), p(15), c(15) );
cass_16 : cass port MAP (p(9), c(15), zero, h(2), d(2), p(16), c(16) );
cass_17 : cass port MAP (p(10), c(16), zero, h(2), d(2), p(17), c(17) );
cass_18 : cass port MAP (zero, zero, a(0), h(1), d(1), p(18), c(18) );
cass_19 : cass port MAP (p(11), c(18), a(1), h(1), d(1), p(19), c(19) );
cass_20 : cass port MAP (p(12), c(19), a(2), h(1), d(1), p(20), c(20) );
cass_21 : cass port MAP (p(13), c(20), a(3), h(1), d(1), p(21), c(21) );
cass_22 : cass port MAP (p(14), c(21), zero, h(1), d(1), p(22), c(22) );
cass_23 : cass port MAP (p(15), c(22), zero, h(1), d(1), p(23), c(23) );
cass_24 : cass port MAP (p(16), c(23), zero, h(1), d(1), p(24), c(24) );
cass_25 : cass port MAP (p(17), c(24), zero, h(1), d(1), p(25), c(25) );
cass_26 : cass port MAP (zero, zero, a(0), h(0), d(0), p(26), c(26) );
cass_27 : cass port MAP (p(18), c(26), a(1), h(0), d(0), p(27), c(27) );
cass_28 : cass port MAP (p(19), c(27), a(2), h(0), d(0), p(28), c(28) );
cass_29 : cass port MAP (p(20), c(28), a(3), h(0), d(0), p(29), c(29) );
cass_30 : cass port MAP (p(21), c(29), zero, h(0), d(0), p(30), c(30) );
cass_31 : cass port MAP (p(22), c(30), zero, h(0), d(0), p(31), c(31) );
cass_32 : cass port MAP (p(23), c(31), zero, h(0), d(0), p(32), c(32) );
cass_33 : cass port MAP (p(24), c(32), zero, h(0), d(0), p(33), c(33) );
cass_34 : cass port MAP (p(25), c(33), zero, h(0), d(0), p(34), c(34) );

```

```

p <= p(33 DOWNT0 26);

```

```

END network;

```

B.2 Boolean Equation Generation

The Booth multiplier is an array multiplier, composed of CASS and CTRL units.

The CASS unit is responsible for performing arithmetic/shifts operations, based on its inputs. The CTRL unit is responsible for producing the input signals to the CASS units. After compiling the each of the independent units (CASS, CTRL), the multiplier (Booth4) is compiled. The report file generated by the Altera MAX+PLUS

II compiler contains input/output information as well as the Boolean equations given below.

**** INPUTS ****

Pin	LC	EC	Row	Col	Primitive	Code	INP	FBK	OUT	FBK	Name
44	-	-	-	--	INPUT		0	0	0	11	a0
43	-	-	-	--	INPUT		0	0	0	11	a1
42	-	-	-	--	INPUT		0	0	0	10	a2
10	-	-	-	01	INPUT		0	0	0	9	a3
84	-	-	-	--	INPUT		0	0	0	14	x0
2	-	-	-	--	INPUT		0	0	0	8	x1
1	-	-	-	--	INPUT		0	0	0	13	x2
11	-	-	-	01	INPUT		0	0	0	10	x3

**** OUTPUTS ****

Pin	LC	EC	Row	Col	Primitive	Code	INP	FBK	OUT	FBK	Name
62	-	-	C	--	OUTPUT		0	1	0	0	p0
29	-	-	C	--	OUTPUT		0	1	0	0	p1
18	-	-	A	--	OUTPUT		0	1	0	0	p2
17	-	-	A	--	OUTPUT		0	1	0	0	p3
19	-	-	A	--	OUTPUT		0	1	0	0	p4
25	-	-	B	--	OUTPUT		0	1	0	0	p5
24	-	-	B	--	OUTPUT		0	1	0	0	p6
23	-	-	B	--	OUTPUT		0	1	0	0	p7

**** EQUATIONS ****

```

a0      : INPUT;
a1      : INPUT;
a2      : INPUT;
a3      : INPUT;
x0      : INPUT;
x1      : INPUT;
x2      : INPUT;
x3      : INPUT;

```

```

-- Node name is 'p0'
-- Equation name is 'p0', type is output
p0      = _LC1_C14;

```

```

-- Node name is 'p1'
-- Equation name is 'p1', type is output
p1      = _LC5_C6;

```

```

-- Node name is 'p2'

```

```

-- Equation name is 'p2', type is output
p2      = _LC5_A1;

-- Node name is 'p3'
-- Equation name is 'p3', type is output
p3      = _LC3_A1;

-- Node name is 'p4'
-- Equation name is 'p4', type is output
p4      = _LC7_A1;

-- Node name is 'p5'
-- Equation name is 'p5', type is output
p5      = _LC7_B2;

-- Node name is 'p6'
-- Equation name is 'p6', type is output
p6      = _LC5_B2;

-- Node name is 'p7'
-- Equation name is 'p7', type is output
p7      = _LC3_B2;

-- Node name is '|cass:cass_6|:14'
-- Equation name is '_LC4_A10', type is buried
_LC4_A10 = LCELL( _EQ001);
_EQ001 = a1 & x2 & !x3
        # a0 & x2 & x3
        # a1 & !x2 & x3
        # a0 & a1 & x3;

-- Node name is '|cass:cass_6|:23'
-- Equation name is '_LC4_C3', type is buried
_LC4_C3 = LCELL( _EQ002);
_EQ002 = !a0 & a1 & !x2 & x3
        # a0 & a1 & x2 & x3;

-- Node name is '|cass:cass_7|:12'
-- Equation name is '_LC5_C3', type is buried
_LC5_C3 = LCELL( _EQ003);
_EQ003 = a1 & x2 & x3
        # a1 & !a2 & x3
        # a2 & x2 & !x3
        # !a1 & a2 & !x2 & x3;

-- Node name is '|cass:cass_7|:14'
-- Equation name is '_LC1_C3', type is buried
_LC1_C3 = LCELL( _EQ004);

```

```

_EQ004 = _LC4_C3 & !_LC5_C3 & !x2 & x3
# _LC4_C3 & !_LC5_C3 & x2 & !x3
# !_LC4_C3 & _LC5_C3
# _LC5_C3 & x2 & x3
# _LC5_C3 & !x2 & !x3;

-- Node name is '|cass:cass_7|:22'
-- Equation name is '_LC6_C3', type is buried
_LC6_C3 = LCELL( _EQ005);
_EQ005 = _LC4_C3
# a2;

-- Node name is '|cass:cass_7|:23'
-- Equation name is '_LC3_C3', type is buried
_LC3_C3 = LCELL( _EQ006);
_EQ006 = a1 & _LC6_C3 & x2 & x3
# !a1 & _LC6_C3 & !x2 & x3;

-- Node name is '|cass:cass_8|:12'
-- Equation name is '_LC5_A4', type is buried
_LC5_A4 = LCELL( _EQ007);
_EQ007 = a2 & x2 & x3
# a2 & !a3 & x3
# a3 & x2 & !x3
# !a2 & a3 & !x2 & x3;

-- Node name is '|cass:cass_8|:13'
-- Equation name is '_LC2_C3', type is buried
_LC2_C3 = LCELL( _EQ008);
_EQ008 = !a1 & _LC6_C3 & !x2 & x3;

-- Node name is '|cass:cass_8|:22'
-- Equation name is '_LC2_C9', type is buried
_LC2_C9 = LCELL( _EQ009);
_EQ009 = _LC3_C3
# a3;

-- Node name is '|cass:cass_9|:13'
-- Equation name is '_LC3_C9', type is buried
_LC3_C9 = LCELL( _EQ010);
_EQ010 = !a2 & _LC2_C9 & !x2 & x3;

-- Node name is '|cass:cass_9|:14'
-- Equation name is '_LC1_C9', type is buried
_LC1_C9 = LCELL( _EQ011);
_EQ011 = _LC3_C9 & !x3
# !a3 & _LC3_C9

```

```

# a3 & !_LC3_C9 & x3;
-- Node name is '|cass:cass_12|:12'
-- Equation name is '_LC7_A10', type is buried
_LC7_A10 = LCELL( _EQ012);
  _EQ012 = a0 & !_LC1_A12 & _LC5_A10
# a0 & !a1 & _LC5_A10
# a1 & _LC1_A12 & !_LC5_A10
# !a0 & a1 & _LC1_A12;

-- Node name is '|cass:cass_12|:14'
-- Equation name is '_LC3_A10', type is buried
_LC3_A10 = LCELL( _EQ013);
  _EQ013 = !_LC1_A11 & _LC7_A10
# !a0 & _LC7_A10
# !_LC1_A12 & _LC7_A10
# a0 & _LC1_A11 & _LC1_A12 & !_LC7_A10;

-- Node name is '|cass:cass_12|:23'
-- Equation name is '_LC6_A10', type is buried
_LC6_A10 = LCELL( _EQ014);
  _EQ014 = a0 & a1 & !_LC1_A11 & _LC5_A10
# a0 & _LC1_A11 & !_LC5_A10
# a1 & _LC1_A11 & !_LC5_A10
# !a0 & a1 & _LC1_A11;

-- Node name is '|cass:cass_13|:14'
-- Equation name is '_LC2_A10', type is buried
_LC2_A10 = LCELL( _EQ015);
  _EQ015 = !_LC1_A12 & _LC4_A10
# !a2 & _LC4_A10 & !_LC6_A10
# a2 & _LC1_A12 & !_LC4_A10 & !_LC6_A10
# a2 & _LC4_A10 & _LC6_A10
# !a2 & _LC1_A12 & !_LC4_A10 & _LC6_A10;

-- Node name is '|cass:cass_13|:23'
-- Equation name is '_LC1_A10', type is buried
_LC1_A10 = LCELL( _EQ016);
  _EQ016 = !_LC1_A11 & _LC4_A10 & _LC6_A10
# a2 & !_LC1_A11 & _LC4_A10
# _LC1_A11 & !_LC4_A10 & _LC6_A10
# a2 & _LC1_A11 & !_LC4_A10;

-- Node name is '|cass:cass_14|:14'
-- Equation name is '_LC1_A8', type is buried
_LC1_A8 = LCELL( _EQ017);
  _EQ017 = a3 & _LC1_A10 & _LC1_C3
# !a3 & _LC1_A10 & _LC1_A12 & !_LC1_C3

```



```

# !_LC1_A12 & _LC1_C3
# !a3 & !_LC1_A10 & _LC1_C3
# a3 & !_LC1_A10 & _LC1_A12 & !_LC1_C3;

-- Node name is '|cass:cass_14|:23'
-- Equation name is '_LC4_A4', type is buried
_LC4_A4 = LCELL( _EQ018);
_EQ018 = _LC1_A10 & !_LC1_A11 & _LC1_C3
# a3 & !_LC1_A11 & _LC1_C3
# _LC1_A10 & _LC1_A11 & !_LC1_C3
# a3 & _LC1_A11 & !_LC1_C3;

-- Node name is '|cass:cass_15|:14'
-- Equation name is '_LC2_A4', type is buried
_LC2_A4 = LCELL( _EQ019);
_EQ019 = _LC2_C3 & !_LC4_A4 & !_LC5_A4
# !_LC2_C3 & !_LC4_A4 & _LC5_A4
# !_LC1_A12 & _LC2_C3 & !_LC5_A4
# !_LC1_A12 & !_LC2_C3 & _LC5_A4
# _LC1_A12 & _LC2_C3 & _LC4_A4 & _LC5_A4
# _LC1_A12 & !_LC2_C3 & _LC4_A4 & !_LC5_A4;

-- Node name is '|cass:cass_15|:19'
-- Equation name is '_LC3_A4', type is buried
_LC3_A4 = LCELL( _EQ020);
_EQ020 = _LC2_C3 & !_LC5_A4 & !x2
# !_LC2_C3 & _LC5_A4 & !x2
# _LC2_C3 & !_LC5_A4 & x1
# !_LC2_C3 & _LC5_A4 & x1
# _LC2_C3 & _LC5_A4 & !x1 & x2
# !_LC2_C3 & !_LC5_A4 & !x1 & x2;

-- Node name is '|cass:cass_16|:13'
-- Equation name is '_LC1_A4', type is buried
_LC1_A4 = LCELL( _EQ021);
_EQ021 = _LC3_A4 & _LC4_A4 & !x1 & x2
# _LC3_A4 & _LC4_A4 & x1 & !x2;

-- Node name is '|cass:cass_19|:12'
-- Equation name is '_LC5_A5', type is buried
_LC5_A5 = LCELL( _EQ022);
_EQ022 = a0 & _LC1_A12 & !_LC1_A13
# a0 & !a1 & _LC1_A12
# a1 & !_LC1_A12 & _LC1_A13
# !a0 & a1 & _LC1_A13;

-- Node name is '|cass:cass_19|:14'
-- Equation name is '_LC4_A5', type is buried

```

```

_LC4_A5 = LCELL( _EQ023);
_EQ023 = !_LC1_A13 & _LC5_A5
        # !_LC1_C7 & _LC5_A5
        # !a0 & _LC5_A5
        # a0 & _LC1_A13 & _LC1_C7 & !_LC5_A5;

-- Node name is '|cass:cass_19|:23'
-- Equation name is '_LC6_A5', type is buried
_LC6_A5 = LCELL( _EQ024);
_EQ024 = a0 & a1 & _LC1_A12 & !_LC1_C7
        # a0 & !_LC1_A12 & _LC1_C7
        # a1 & !_LC1_A12 & _LC1_C7
        # !a0 & a1 & _LC1_C7;

-- Node name is '|cass:cass_20|:14'
-- Equation name is '_LC3_A5', type is buried
_LC3_A5 = LCELL( _EQ025);
_EQ025 = !_LC1_A13 & _LC3_A10
        # !a2 & _LC3_A10 & !_LC6_A5
        # a2 & _LC1_A13 & !_LC3_A10 & !_LC6_A5
        # a2 & _LC3_A10 & _LC6_A5
        # !a2 & _LC1_A13 & !_LC3_A10 & _LC6_A5;

-- Node name is '|cass:cass_20|:23'
-- Equation name is '_LC7_A5', type is buried
_LC7_A5 = LCELL( _EQ026);
_EQ026 = !_LC1_C7 & _LC3_A10 & _LC6_A5
        # a2 & !_LC1_C7 & _LC3_A10
        # _LC1_C7 & !_LC3_A10 & _LC6_A5
        # a2 & _LC1_C7 & !_LC3_A10;

-- Node name is '|cass:cass_21|:12'
-- Equation name is '_LC2_A1', type is buried
_LC2_A1 = LCELL( _EQ027);
_EQ027 = !a3 & _LC2_A10
        # !_LC1_A13 & _LC2_A10
        # a3 & _LC1_A13 & !_LC2_A10;

-- Node name is '|cass:cass_21|:13'
-- Equation name is '_LC2_A5', type is buried
_LC2_A5 = LCELL( _EQ028);
_EQ028 = _LC1_A13 & _LC7_A5;

-- Node name is '|cass:cass_21|:23'
-- Equation name is '_LC1_A5', type is buried
_LC1_A5 = LCELL( _EQ029);
_EQ029 = !_LC1_C7 & _LC2_A10 & _LC7_A5

```

```

# a3 & !_LC1_C7 & _LC2_A10
# _LC1_C7 & !_LC2_A10 & _LC7_A5
# a3 & _LC1_C7 & !_LC2_A10;

-- Node name is '|cass:cass_22|:14'
-- Equation name is '_LC8_B2', type is buried
_LC8_B2 = LCELL( _EQ030);
_EQ030 = !_LC1_A5 & _LC1_A8
# _LC1_A8 & !_LC1_A13
# _LC1_A5 & !_LC1_A8 & _LC1_A13;

-- Node name is '|cass:cass_22|:23'
-- Equation name is '_LC1_B2', type is buried
_LC1_B2 = LCELL( _EQ031);
_EQ031 = _LC1_A5 & _LC1_A8 & !_LC1_C7
# _LC1_A5 & !_LC1_A8 & _LC1_C7;

-- Node name is '|cass:cass_23|:14'
-- Equation name is '_LC2_B2', type is buried
_LC2_B2 = LCELL( _EQ032);
_EQ032 = !_LC1_B2 & _LC2_A4
# !_LC1_A13 & _LC2_A4
# _LC1_A13 & _LC1_B2 & !_LC2_A4;

-- Node name is '|cass:cass_24|:13'
-- Equation name is '_LC4_B2', type is buried
_LC4_B2 = LCELL( _EQ033);
_EQ033 = _LC1_A13 & _LC1_B2 & !_LC1_C7 & _LC2_A4
# _LC1_A13 & _LC1_B2 & _LC1_C7 & !_LC2_A4;

-- Node name is '|cass:cass_26|:11'
-- Equation name is '_LC1_C14', type is buried
_LC5_C6 = LCELL( _EQ035);
_EQ035 = !a0 & a1 & x0
# a0 & !x0 & x1
# a1 & x0 & !x1
# a0 & !a1 & x1;

-- Node name is '|cass:cass_27|:23'
-- Equation name is '_LC4_A1', type is buried
_LC4_A1 = LCELL( _EQ036);
_EQ036 = a0 & x0 & x1
# a0 & a1 & x1
# a1 & x0 & x1
# !a0 & a1 & x0;

-- Node name is '|cass:cass_28|:14'

```

```

-- Equation name is '_LC5_A1', type is buried
_LC5_A1 = LCELL( _EQ037);
  _EQ037 = _LC4_A5 & !x0
          # !a2 & !_LC4_A1 & _LC4_A5
          # a2 & !_LC4_A1 & !_LC4_A5 & x0
          # a2 & _LC4_A1 & _LC4_A5
          # !a2 & _LC4_A1 & !_LC4_A5 & x0;

-- Node name is '|cass:cass_28|:23'
-- Equation name is '_LC8_A1', type is buried
_LC8_A1 = LCELL( _EQ038);
  _EQ038 = a2 & _LC4_A5 & !x0
          # a2 & !_LC4_A5 & x0
          # _LC4_A1 & _LC4_A5 & !x0
          # _LC4_A1 & !_LC4_A5 & x0;

-- Node name is '|cass:cass_29|:14'
-- Equation name is '_LC3_A1', type is buried
_LC3_A1 = LCELL( _EQ039);
  _EQ039 = _LC3_A5 & !x0
          # !a3 & _LC3_A5 & !_LC8_A1
          # a3 & !_LC3_A5 & !_LC8_A1 & x0
          # a3 & _LC3_A5 & _LC8_A1
          # !a3 & !_LC3_A5 & _LC8_A1 & x0;

-- Node name is '|cass:cass_29|:23'
-- Equation name is '_LC6_A1', type is buried
_LC6_A1 = LCELL( _EQ040);
  _EQ040 = a3 & _LC3_A5 & !x0
          # _LC3_A5 & _LC8_A1 & !x0
          # a3 & !_LC3_A5 & x0
          # !_LC3_A5 & _LC8_A1 & x0;

-- Node name is '|cass:cass_30|:14'
-- Equation name is '_LC7_A1', type is buried
_LC7_A1 = LCELL( _EQ041);
  _EQ041 = !_LC2_A1 & _LC2_A5 & !_LC6_A1
          # _LC2_A1 & !_LC2_A5 & !_LC6_A1
          # !_LC2_A1 & _LC2_A5 & !x0
          # _LC2_A1 & !_LC2_A5 & !x0
          # _LC2_A1 & _LC2_A5 & _LC6_A1 & x0
          # !_LC2_A1 & !_LC2_A5 & _LC6_A1 & x0;

-- Node name is '|cass:cass_30|:23'
-- Equation name is '_LC1_A1', type is buried
_LC1_A1 = LCELL( _EQ042);
  _EQ042 = !_LC2_A1 & _LC2_A5 & _LC6_A1 & !x0

```

```

# _LC2_A1 & !_LC2_A5 & _LC6_A1 & !x0
# _LC2_A1 & _LC2_A5 & _LC6_A1 & x0
# !_LC2_A1 & !_LC2_A5 & _LC6_A1 & x0;

-- Node name is '|cass:cass_31|:14'
-- Equation name is '_LC7_B2', type is buried
_LC7_B2 = LCELL( _EQ043);
_EQ043 = !_LC1_A1 & _LC8_B2
# _LC8_B2 & !x0
# _LC1_A1 & !_LC8_B2 & x0;

-- Node name is '|cass:cass_32|:14'
-- Equation name is '_LC5_B2', type is buried
_LC5_B2 = LCELL( _EQ044);
_EQ044 = _LC2_B2 & _LC8_B2
# !_LC1_A1 & _LC2_B2
# _LC2_B2 & !x0
# _LC1_A1 & !_LC2_B2 & !_LC8_B2 & x0;

-- Node name is '|cass:cass_33|:13'
-- Equation name is '_LC6_B2', type is buried
_LC6_B2 = LCELL( _EQ045);
_EQ045 = _LC1_A1 & !_LC2_B2 & !_LC8_B2 & x0;

-- Node name is '|cass:cass_33|:14'
-- Equation name is '_LC3_B2', type is buried
_LC3_B2 = LCELL( _EQ046);
_EQ046 = _LC1_A4 & !_LC1_C9 & _LC4_B2 & _LC6_B2
# !_LC1_A4 & _LC1_C9 & _LC4_B2 & _LC6_B2
# _LC1_A4 & _LC1_C9 & !_LC4_B2 & _LC6_B2
# !_LC1_A4 & !_LC1_C9 & !_LC4_B2 & _LC6_B2
# _LC1_A4 & _LC1_C9 & _LC4_B2 & !_LC6_B2
# !_LC1_A4 & !_LC1_C9 & _LC4_B2 & !_LC6_B2
# _LC1_A4 & !_LC1_C9 & !_LC4_B2 & !_LC6_B2
# !_LC1_A4 & _LC1_C9 & !_LC4_B2 & !_LC6_B2;

-- Node name is '|ctrl:ctrl_1|:5'
-- Equation name is '_LC1_A13', type is buried
_LC1_A13 = LCELL( _EQ047);
_EQ047 = !x0 & x1
# x0 & !x1;

-- Node name is '|ctrl:ctrl_1|:8'
-- Equation name is '_LC1_C7', type is buried
_LC1_C7 = LCELL( _EQ048);
_EQ048 = !x0 & x1;

```

```

-- Node name is '|ctrl:ctrl_2|:5'
-- Equation name is '_LC1_A12', type is buried
_LC1_A12 = LCELL( _EQ049);
  _EQ049 = !x1 & x2
          # x1 & !x2;

-- Node name is '|ctrl:ctrl_2|:8'
-- Equation name is '_LC1_A11', type is buried
_LC1_A11 = LCELL( _EQ050);
  _EQ050 = !x1 & x2;

-- Node name is '|ctrl:ctrl_3|:5'
-- Equation name is '_LC5_A10', type is buried
_LC5_A10 = LCELL( _EQ051);
  _EQ051 = !x2 & x3
          # x2 & !x3;

** COMPILATION SETTINGS & TIMES **

```

B.3 Generate Implicit Structural Specification

Next, a BLAPE program extracts the Boolean equations from the report file and applies implicit structure. The resulting structural representation is a BLIF-formatted gate-level circuit specification.

```

.inputs a0 a1 a2 a3 x0 x1 x2 x3
.outputs p0 p1 p2 p3 p4 p5 p6 p7

.gate not a=_LC5_C3 0=NOT_LC5_C3
.gate not a=_LC4_C3 0=NOT_LC4_C3
.gate not a=_LC3_C9 0=NOT_LC3_C9
.gate not a=_LC1_A12 0=NOT_LC1_A12
.gate not a=_LC5_A10 0=NOT_LC5_A10
.gate not a=_LC1_A11 0=NOT_LC1_A11
.gate not a=_LC6_A10 0=NOT_LC6_A10
.gate not a=_LC4_A10 0=NOT_LC4_A10
.gate not a=_LC1_C3 0=NOT_LC1_C3
.gate not a=_LC1_A10 0=NOT_LC1_A10
.gate not a=_LC4_A4 0=NOT_LC4_A4
.gate not a=_LC5_A4 0=NOT_LC5_A4
.gate not a=_LC2_C3 0=NOT_LC2_C3

```

```

.gate not a=_LC1_A13 O=NOT_LC1_A13
.gate not a=_LC1_C7 O=NOT_LC1_C7
.gate not a=_LC6_A5 O=NOT_LC6_A5
.gate not a=_LC3_A10 O=NOT_LC3_A10
.gate not a=_LC2_A10 O=NOT_LC2_A10
.gate not a=_LC1_A5 O=NOT_LC1_A5
.gate not a=_LC1_A8 O=NOT_LC1_A8
.gate not a=_LC1_B2 O=NOT_LC1_B2
.gate not a=_LC4_A1 O=NOT_LC4_A1
.gate not a=_LC4_A5 O=NOT_LC4_A5
.gate not a=_LC8_A1 O=NOT_LC8_A1
.gate not a=_LC3_A5 O=NOT_LC3_A5
.gate not a=_LC2_A1 O=NOT_LC2_A1
.gate not a=_LC6_A1 O=NOT_LC6_A1
.gate not a=_LC2_A5 O=NOT_LC2_A5
.gate not a=_LC1_A1 O=NOT_LC1_A1
.gate not a=_LC8_B2 O=NOT_LC8_B2
.gate not a=_LC2_B2 O=NOT_LC2_B2
.gate not a=_LC1_C9 O=NOT_LC1_C9
.gate not a=_LC1_A4 O=NOT_LC1_A4
.gate not a=_LC4_B2 O=NOT_LC4_B2
.gate not a=_LC6_B2 O=NOT_LC6_B2
.gate not a=_LC2_A4 O=NOT_LC2_A4
.gate not a=_LC5_A5 O=NOT_LC5_A5
.gate not a=_LC7_A10 O=NOT_LC7_A10

.gate not a=a0 O=NOTa0
.gate not a=a1 O=NOTa1
.gate not a=a2 O=NOTa2
.gate not a=a3 O=NOTa3
.gate not a=x0 O=NOTx0
.gate not a=x1 O=NOTx1
.gate not a=x2 O=NOTx2
.gate not a=x3 O=NOTx3

.gate buf1 a=_LC1_C14 O=p0
.gate buf1 a=_LC5_C6 O=p1
.gate buf1 a=_LC5_A1 O=p2
.gate buf1 a=_LC3_A1 O=p3
.gate buf1 a=_LC7_A1 O=p4
.gate buf1 a=_LC7_B2 O=p5
.gate buf1 a=_LC5_B2 O=p6
.gate buf1 a=_LC3_B2 O=p7

.gate and3 a=a1 b=x2 c=NOTx3 O=t0
.gate and3 a=a0 b=x2 c=x3 O=t1
.gate and3 a=a1 b=NOTx2 c=x3 O=t2

```

```

.gate and3 a=a0 b=a1 c=x3 O=t3
.gate or4 a=t0 b=t1 c=t2 d=t3 O=_LC4_A10
.gate and4 a=NOTa0 b=a1 c=NOTx2 d=x3 O=t4
.gate and4 a=a0 b=a1 c=x2 d=x3 O=t5
.gate or2 a=t4 b=t5 O=_LC4_C3
.gate and3 a=a1 b=x2 c=x3 O=t6
.gate and3 a=a1 b=NOTa2 c=x3 O=t7
.gate and3 a=a2 b=x2 c=NOTx3 O=t8
.gate and4 a=NOTa1 b=a2 c=NOTx2 d=x3 O=t9
.gate or4 a=t6 b=t7 c=t8 d=t9 O=_LC5_C3
.gate and4 a=_LC4_C3 b=NOT_LC5_C3 c=NOTx2 d=x3 O=t10
.gate and4 a=_LC4_C3 b=NOT_LC5_C3 c=x2 d=NOTx3 O=t11
.gate and2 a=NOT_LC4_C3 b=_LC5_C3 O=t12
.gate and3 a=_LC5_C3 b=x2 c=x3 O=t13
.gate and3 a=_LC5_C3 b=NOTx2 c=NOTx3 O=t14
.gate or5 a=t10 b=t11 c=t12 d=t13 e=t14 O=_LC1_C3
.gate or2 a=_LC4_C3 b=a2 O=_LC6_C3
.gate and4 a=a1 b=_LC6_C3 c=x2 d=x3 O=t15
.gate and4 a=NOTa1 b=_LC6_C3 c=NOTx2 d=x3 O=t16
.gate or2 a=t15 b=t16 O=_LC3_C3
.gate and3 a=a2 b=x2 c=x3 O=t17
.gate and3 a=a2 b=NOTa3 c=x3 O=t18
.gate and3 a=a3 b=x2 c=NOTx3 O=t19
.gate and4 a=NOTa2 b=a3 c=NOTx2 d=x3 O=t20
.gate or4 a=t17 b=t18 c=t19 d=t20 O=_LC5_A4
.gate and4 a=NOTa1 b=_LC6_C3 c=NOTx2 d=x3 O=t21
.gate buf1 a=t21 O=_LC2_C3
.gate or2 a=_LC3_C3 b=a3 O=_LC2_C9
.gate and4 a=NOTa2 b=_LC2_C9 c=NOTx2 d=x3 O=t22
.gate buf1 a=t22 O=_LC3_C9
.gate and2 a=_LC3_C9 b=NOTx3 O=t23
.gate and2 a=NOTa3 b=_LC3_C9 O=t24
.gate and3 a=a3 b=NOT_LC3_C9 c=x3 O=t25
.gate or3 a=t23 b=t24 c=t25 O=_LC1_C9
.gate and3 a=a0 b=NOT_LC1_A12 c=_LC5_A10 O=t26
.gate and3 a=a0 b=NOTa1 c=_LC5_A10 O=t27
.gate and3 a=a1 b=_LC1_A12 c=NOT_LC5_A10 O=t28
.gate and3 a=NOTa0 b=a1 c=_LC1_A12 O=t29
.gate or4 a=t26 b=t27 c=t28 d=t29 O=_LC7_A10
.gate and2 a=NOT_LC1_A11 b=_LC7_A10 O=t30
.gate and2 a=NOTa0 b=_LC7_A10 O=t31
.gate and2 a=NOT_LC1_A12 b=_LC7_A10 O=t32
.gate and4 a=a0 b=_LC1_A11 c=_LC1_A12 d=NOT_LC7_A10 O=t33
.gate or4 a=t30 b=t31 c=t32 d=t33 O=_LC3_A10
.gate and4 a=a0 b=a1 c=NOT_LC1_A11 d=_LC5_A10 O=t34
.gate and3 a=a0 b=_LC1_A11 c=NOT_LC5_A10 O=t35
.gate and3 a=a1 b=_LC1_A11 c=NOT_LC5_A10 O=t36

```



```

.gate and3 a=NOTa0 b=a1 c=_LC1_A11 O=t37
.gate or4 a=t34 b=t35 c=t36 d=t37 O=_LC6_A10
.gate and2 a=NOT_LC1_A12 b=_LC4_A10 O=t38
.gate and3 a=NOTa2 b=_LC4_A10 c=NOT_LC6_A10 O=t39
.gate and4 a=a2 b=_LC1_A12 c=NOT_LC4_A10 d=NOT_LC6_A10 O=t40
.gate and3 a=a2 b=_LC4_A10 c=_LC6_A10 O=t41
.gate and4 a=NOTa2 b=_LC1_A12 c=NOT_LC4_A10 d=_LC6_A10 O=t42
.gate or5 a=t38 b=t39 c=t40 d=t41 e=t42 O=_LC2_A10
.gate and3 a=NOT_LC1_A11 b=_LC4_A10 c=_LC6_A10 O=t43
.gate and3 a=a2 b=NOT_LC1_A11 c=_LC4_A10 O=t44
.gate and3 a=_LC1_A11 b=NOT_LC4_A10 c=_LC6_A10 O=t45
.gate and3 a=a2 b=_LC1_A11 c=NOT_LC4_A10 O=t46
.gate or4 a=t43 b=t44 c=t45 d=t46 O=_LC1_A10
.gate and3 a=a3 b=_LC1_A10 c=_LC1_C3 O=t47
.gate and4 a=NOTa3 b=_LC1_A10 c=_LC1_A12 d=NOT_LC1_C3 O=t48
.gate and2 a=NOT_LC1_A12 b=_LC1_C3 O=t49
.gate and3 a=NOTa3 b=NOT_LC1_A10 c=_LC1_C3 O=t50
.gate and4 a=a3 b=NOT_LC1_A10 c=_LC1_A12 d=NOT_LC1_C3 O=t51
.gate or5 a=t47 b=t48 c=t49 d=t50 e=t51 O=_LC1_A8
.gate and3 a=_LC1_A10 b=NOT_LC1_A11 c=_LC1_C3 O=t52
.gate and3 a=a3 b=NOT_LC1_A11 c=_LC1_C3 O=t53
.gate and3 a=_LC1_A10 b=_LC1_A11 c=NOT_LC1_C3 O=t54
.gate and3 a=a3 b=_LC1_A11 c=NOT_LC1_C3 O=t55
.gate or4 a=t52 b=t53 c=t54 d=t55 O=_LC4_A4
.gate and3 a=_LC2_C3 b=NOT_LC4_A4 c=NOT_LC5_A4 O=t56
.gate and3 a=NOT_LC2_C3 b=NOT_LC4_A4 c=_LC5_A4 O=t57
.gate and3 a=NOT_LC1_A12 b=_LC2_C3 c=NOT_LC5_A4 O=t58
.gate and3 a=NOT_LC1_A12 b=NOT_LC2_C3 c=_LC5_A4 O=t59
.gate and4 a=_LC1_A12 b=_LC2_C3 c=_LC4_A4 d=_LC5_A4 O=t60
.gate and4 a=_LC1_A12 b=NOT_LC2_C3 c=_LC4_A4 d=NOT_LC5_A4 O=t61
.gate or6 a=t56 b=t57 c=t58 d=t59 e=t60 f=t61 O=_LC2_A4
.gate and3 a=_LC2_C3 b=NOT_LC5_A4 c=NOTx2 O=t62
.gate and3 a=NOT_LC2_C3 b=_LC5_A4 c=NOTx2 O=t63
.gate and3 a=_LC2_C3 b=NOT_LC5_A4 c=x1 O=t64
.gate and3 a=NOT_LC2_C3 b=_LC5_A4 c=x1 O=t65
.gate and4 a=_LC2_C3 b=_LC5_A4 c=NOTx1 d=x2 O=t66
.gate and4 a=NOT_LC2_C3 b=NOT_LC5_A4 c=NOTx1 d=x2 O=t67
.gate or6 a=t62 b=t63 c=t64 d=t65 e=t66 f=t67 O=_LC3_A4
.gate and4 a=_LC3_A4 b=_LC4_A4 c=NOTx1 d=x2 O=t68
.gate and4 a=_LC3_A4 b=_LC4_A4 c=x1 d=NOTx2 O=t69
.gate or2 a=t68 b=t69 O=_LC1_A4
.gate and3 a=a0 b=_LC1_A12 c=NOT_LC1_A13 O=t70
.gate and3 a=a0 b=NOTa1 c=_LC1_A12 O=t71
.gate and3 a=a1 b=NOT_LC1_A12 c=_LC1_A13 O=t72
.gate and3 a=NOTa0 b=a1 c=_LC1_A13 O=t73
.gate or4 a=t70 b=t71 c=t72 d=t73 O=_LC5_A5
.gate and2 a=NOT_LC1_A13 b=_LC5_A5 O=t74

```

```

.gate and2 a=NOT_LC1_C7 b=_LC5_A5 O=t75
.gate and2 a=NOTa0 b=_LC5_A5 O=t76
.gate and4 a=a0 b=_LC1_A13 c=_LC1_C7 d=NOT_LC5_A5 O=t77
.gate or4 a=t74 b=t75 c=t76 d=t77 O=_LC4_A5
.gate and4 a=a0 b=a1 c=_LC1_A12 d=NOT_LC1_C7 O=t78
.gate and3 a=a0 b=NOT_LC1_A12 c=_LC1_C7 O=t79
.gate and3 a=a1 b=NOT_LC1_A12 c=_LC1_C7 O=t80
.gate and3 a=NOTa0 b=a1 c=_LC1_C7 O=t81
.gate or4 a=t78 b=t79 c=t80 d=t81 O=_LC6_A5
.gate and2 a=NOT_LC1_A13 b=_LC3_A10 O=t82
.gate and3 a=NOTa2 b=_LC3_A10 c=NOT_LC6_A5 O=t83
.gate and4 a=a2 b=_LC1_A13 c=NOT_LC3_A10 d=NOT_LC6_A5 O=t84
.gate and3 a=a2 b=_LC3_A10 c=_LC6_A5 O=t85
.gate and4 a=NOTa2 b=_LC1_A13 c=NOT_LC3_A10 d=_LC6_A5 O=t86
.gate or5 a=t82 b=t83 c=t84 d=t85 e=t86 O=_LC3_A5
.gate and3 a=NOT_LC1_C7 b=_LC3_A10 c=_LC6_A5 O=t87
.gate and3 a=a2 b=NOT_LC1_C7 c=_LC3_A10 O=t88
.gate and3 a=_LC1_C7 b=NOT_LC3_A10 c=_LC6_A5 O=t89
.gate and3 a=a2 b=_LC1_C7 c=NOT_LC3_A10 O=t90
.gate or4 a=t87 b=t88 c=t89 d=t90 O=_LC7_A5
.gate and2 a=NOTa3 b=_LC2_A10 O=t91
.gate and2 a=NOT_LC1_A13 b=_LC2_A10 O=t92
.gate and3 a=a3 b=_LC1_A13 c=NOT_LC2_A10 O=t93
.gate or3 a=t91 b=t92 c=t93 O=_LC2_A1
.gate and2 a=_LC1_A13 b=_LC7_A5 O=t94
.gate buf1 a=t94 O=_LC2_A5
.gate and3 a=NOT_LC1_C7 b=_LC2_A10 c=_LC7_A5 O=t95
.gate and3 a=a3 b=NOT_LC1_C7 c=_LC2_A10 O=t96
.gate and3 a=_LC1_C7 b=NOT_LC2_A10 c=_LC7_A5 O=t97
.gate and3 a=a3 b=_LC1_C7 c=NOT_LC2_A10 O=t98
.gate or4 a=t95 b=t96 c=t97 d=t98 O=_LC1_A5
.gate and2 a=NOT_LC1_A5 b=_LC1_A8 O=t99
.gate and2 a=_LC1_A8 b=NOT_LC1_A13 O=t100
.gate and3 a=_LC1_A5 b=NOT_LC1_A8 c=_LC1_A13 O=t101
.gate or3 a=t99 b=t100 c=t101 O=_LC8_B2
.gate and3 a=_LC1_A5 b=_LC1_A8 c=NOT_LC1_C7 O=t102
.gate and3 a=_LC1_A5 b=NOT_LC1_A8 c=_LC1_C7 O=t103
.gate or2 a=t102 b=t103 O=_LC1_B2
.gate and2 a=NOT_LC1_B2 b=_LC2_A4 O=t104
.gate and2 a=NOT_LC1_A13 b=_LC2_A4 O=t105
.gate and3 a=_LC1_A13 b=_LC1_B2 c=NOT_LC2_A4 O=t106
.gate or3 a=t104 b=t105 c=t106 O=_LC2_B2
.gate and4 a=_LC1_A13 b=_LC1_B2 c=NOT_LC1_C7 d=_LC2_A4 O=t107
.gate and4 a=_LC1_A13 b=_LC1_B2 c=_LC1_C7 d=NOT_LC2_A4 O=t108
.gate or2 a=t107 b=t108 O=_LC4_B2
.gate and2 a=a0 b=x0 O=t109
.gate buf1 a=t109 O=_LC1_C14

```

```

.gate and3 a=NOTa0 b=a1 c=x0 O=t110
.gate and3 a=a0 b=NOTx0 c=x1 O=t111
.gate and3 a=a1 b=x0 c=NOTx1 O=t112
.gate and3 a=a0 b=NOTa1 c=x1 O=t113
.gate or4 a=t110 b=t111 c=t112 d=t113 O=_LC5_C6
.gate and3 a=a0 b=x0 c=x1 O=t114
.gate and3 a=a0 b=a1 c=x1 O=t115
.gate and3 a=a1 b=x0 c=x1 O=t116
.gate and3 a=NOTa0 b=a1 c=x0 O=t117
.gate or4 a=t114 b=t115 c=t116 d=t117 O=_LC4_A1
.gate and2 a=_LC4_A5 b=NOTx0 O=t118
.gate and3 a=NOTa2 b=NOT_LC4_A1 c=_LC4_A5 O=t119
.gate and4 a=a2 b=NOT_LC4_A1 c=NOT_LC4_A5 d=x0 O=t120
.gate and3 a=a2 b=_LC4_A1 c=_LC4_A5 O=t121
.gate and4 a=NOTa2 b=_LC4_A1 c=NOT_LC4_A5 d=x0 O=t122
.gate or5 a=t118 b=t119 c=t120 d=t121 e=t122 O=_LC5_A1
.gate and3 a=a2 b=_LC4_A5 c=NOTx0 O=t123
.gate and3 a=a2 b=NOT_LC4_A5 c=x0 O=t124
.gate and3 a=_LC4_A1 b=_LC4_A5 c=NOTx0 O=t125
.gate and3 a=_LC4_A1 b=NOT_LC4_A5 c=x0 O=t126
.gate or4 a=t123 b=t124 c=t125 d=t126 O=_LC8_A1
.gate and2 a=_LC3_A5 b=NOTx0 O=t127
.gate and3 a=NOTa3 b=_LC3_A5 c=NOT_LC8_A1 O=t128
.gate and4 a=a3 b=NOT_LC3_A5 c=NOT_LC8_A1 d=x0 O=t129
.gate and3 a=a3 b=_LC3_A5 c=_LC8_A1 O=t130
.gate and4 a=NOTa3 b=NOT_LC3_A5 c=_LC8_A1 d=x0 O=t131
.gate or5 a=t127 b=t128 c=t129 d=t130 e=t131 O=_LC3_A1
.gate and3 a=a3 b=_LC3_A5 c=NOTx0 O=t132
.gate and3 a=_LC3_A5 b=_LC8_A1 c=NOTx0 O=t133
.gate and3 a=a3 b=NOT_LC3_A5 c=x0 O=t134
.gate and3 a=NOT_LC3_A5 b=_LC8_A1 c=x0 O=t135
.gate or4 a=t132 b=t133 c=t134 d=t135 O=_LC6_A1
.gate and3 a=NOT_LC2_A1 b=_LC2_A5 c=NOT_LC6_A1 O=t136
.gate and3 a=_LC2_A1 b=NOT_LC2_A5 c=NOT_LC6_A1 O=t137
.gate and3 a=NOT_LC2_A1 b=_LC2_A5 c=NOTx0 O=t138
.gate and3 a=_LC2_A1 b=NOT_LC2_A5 c=NOTx0 O=t139
.gate and4 a=_LC2_A1 b=_LC2_A5 c=_LC6_A1 d=x0 O=t140
.gate and4 a=NOT_LC2_A1 b=NOT_LC2_A5 c=_LC6_A1 d=x0 O=t141
.gate or6 a=t136 b=t137 c=t138 d=t139 e=t140 f=t141 O=_LC7_A1
.gate and4 a=NOT_LC2_A1 b=_LC2_A5 c=_LC6_A1 d=NOTx0 O=t142
.gate and4 a=_LC2_A1 b=NOT_LC2_A5 c=_LC6_A1 d=NOTx0 O=t143
.gate and4 a=_LC2_A1 b=_LC2_A5 c=_LC6_A1 d=x0 O=t144
.gate and4 a=NOT_LC2_A1 b=NOT_LC2_A5 c=_LC6_A1 d=x0 O=t145
.gate or4 a=t142 b=t143 c=t144 d=t145 O=_LC1_A1
.gate and2 a=NOT_LC1_A1 b=_LC8_B2 O=t146
.gate and2 a=_LC8_B2 b=NOTx0 O=t147
.gate and3 a=_LC1_A1 b=NOT_LC8_B2 c=x0 O=t148

```

```

.gate or3 a=t146 b=t147 c=t148  O=_LC7_B2
.gate and2 a=_LC2_B2 b=_LC8_B2 O=t149
.gate and2 a=NOT_LC1_A1 b=_LC2_B2 O=t150
.gate and2 a=_LC2_B2 b=NOTx0 O=t151
.gate and4 a=_LC1_A1 b=NOT_LC2_B2 c=NOT_LC8_B2 d=x0 O=t152
.gate or4 a=t149 b=t150 c=t151 d=t152  O=_LC5_B2
.gate and4 a=_LC1_A1 b=NOT_LC2_B2 c=NOT_LC8_B2 d=x0 O=t153
.gate buf1 a=t153  O=_LC6_B2
.gate and4 a=_LC1_A4 b=NOT_LC1_C9 c=_LC4_B2 d=_LC6_B2 O=t154
.gate and4 a=NOT_LC1_A4 b=_LC1_C9 c=_LC4_B2 d=_LC6_B2 O=t155
.gate and4 a=_LC1_A4 b=_LC1_C9 c=NOT_LC4_B2 d=_LC6_B2 O=t156
.gate and4 a=NOT_LC1_A4 b=NOT_LC1_C9 c=NOT_LC4_B2 d=_LC6_B2 O=t157
.gate and4 a=_LC1_A4 b=_LC1_C9 c=_LC4_B2 d=NOT_LC6_B2 O=t158
.gate and4 a=NOT_LC1_A4 b=NOT_LC1_C9 c=_LC4_B2 d=NOT_LC6_B2 O=t159
.gate and4 a=_LC1_A4 b=NOT_LC1_C9 c=NOT_LC4_B2 d=NOT_LC6_B2 O=t160
.gate and4 a=NOT_LC1_A4 b=_LC1_C9 c=NOT_LC4_B2 d=NOT_LC6_B2 O=t161
.gate or8 a=t154 b=t155 c=t156 d=t157 e=t158 f=t159 g=t160 h=t161  O=_LC3_B2
.gate and2 a=NOTx0 b=x1 O=t162
.gate and2 a=x0 b=NOTx1 O=t163
.gate or2 a=t162 b=t163  O=_LC1_A13
.gate and2 a=NOTx0 b=x1 O=t164
.gate buf1 a=t164  O=_LC1_C7
.gate and2 a=NOTx1 b=x2 O=t165
.gate and2 a=x1 b=NOTx2 O=t166
.gate or2 a=t165 b=t166  O=_LC1_A12
.gate and2 a=NOTx1 b=x2 O=t167
.gate buf1 a=t167  O=_LC1_A11
.gate and2 a=NOTx2 b=x3 O=t168
.gate and2 a=x2 b=NOTx3 O=t169
.gate or2 a=t168 b=t169  O=_LC5_A10

.end

```

B.4 Network Levelization

The levelization of the network is given below.

```

( 0) NODE= 0 LEVEL= 0 NAME:  ROOT
( 1) NODE= 1 LEVEL= 0 NAME:  a0
( 2) NODE= 2 LEVEL= 0 NAME:  a1
( 3) NODE= 3 LEVEL= 0 NAME:  a2
( 4) NODE= 4 LEVEL= 0 NAME:  a3
( 5) NODE= 5 LEVEL= 0 NAME:  x0
( 6) NODE= 6 LEVEL= 0 NAME:  x1

```

(7) NODE= 7 LEVEL= 0 NAME: x2
 (8) NODE= 8 LEVEL= 0 NAME: x3
 (9) NODE= 55 LEVEL= 1 NAME: NOTa0
 (10) NODE= 56 LEVEL= 1 NAME: NOTa1
 (11) NODE= 57 LEVEL= 1 NAME: NOTa2
 (12) NODE= 58 LEVEL= 1 NAME: NOTa3
 (13) NODE= 59 LEVEL= 1 NAME: NOTx0
 (14) NODE= 60 LEVEL= 1 NAME: NOTx1
 (15) NODE= 61 LEVEL= 1 NAME: NOTx2
 (16) NODE= 62 LEVEL= 1 NAME: NOTx3
 (17) NODE= 64 LEVEL= 1 NAME: t1
 (18) NODE= 66 LEVEL= 1 NAME: t3
 (19) NODE= 69 LEVEL= 1 NAME: t5
 (20) NODE= 71 LEVEL= 1 NAME: t6
 (21) NODE= 86 LEVEL= 1 NAME: t17
 (22) NODE=205 LEVEL= 1 NAME: t109
 (23) NODE=212 LEVEL= 1 NAME: t114
 (24) NODE=213 LEVEL= 1 NAME: t115
 (25) NODE=214 LEVEL= 1 NAME: t116
 (26) NODE= 63 LEVEL= 2 NAME: t0
 (27) NODE= 65 LEVEL= 2 NAME: t2
 (28) NODE= 68 LEVEL= 2 NAME: t4
 (29) NODE= 72 LEVEL= 2 NAME: t7
 (30) NODE= 73 LEVEL= 2 NAME: t8
 (31) NODE= 74 LEVEL= 2 NAME: t9
 (32) NODE= 87 LEVEL= 2 NAME: t18
 (33) NODE= 88 LEVEL= 2 NAME: t19
 (34) NODE= 89 LEVEL= 2 NAME: t20
 (35) NODE=206 LEVEL= 2 NAME: _LC1_C14
 (36) NODE=207 LEVEL= 2 NAME: t110
 (37) NODE=208 LEVEL= 2 NAME: t111
 (38) NODE=209 LEVEL= 2 NAME: t112
 (39) NODE=210 LEVEL= 2 NAME: t113
 (40) NODE=215 LEVEL= 2 NAME: t117
 (41) NODE=271 LEVEL= 2 NAME: t162
 (42) NODE=272 LEVEL= 2 NAME: t163
 (43) NODE=274 LEVEL= 2 NAME: t164
 (44) NODE=276 LEVEL= 2 NAME: t165
 (45) NODE=277 LEVEL= 2 NAME: t166
 (46) NODE=279 LEVEL= 2 NAME: t167
 (47) NODE=281 LEVEL= 2 NAME: t168
 (48) NODE=282 LEVEL= 2 NAME: t169
 (49) NODE= 9 LEVEL= 3 NAME: p0
 (50) NODE= 67 LEVEL= 3 NAME: _LC4_A10
 (51) NODE= 70 LEVEL= 3 NAME: _LC4_C3
 (52) NODE= 75 LEVEL= 3 NAME: _LC5_C3
 (53) NODE= 90 LEVEL= 3 NAME: _LC5_A4

(54) NODE=211 LEVEL= 3 NAME: _LC5_C6
(55) NODE=216 LEVEL= 3 NAME: _LC4_A1
(56) NODE=273 LEVEL= 3 NAME: _LC1_A13
(57) NODE=275 LEVEL= 3 NAME: _LC1_C7
(58) NODE=278 LEVEL= 3 NAME: _LC1_A12
(59) NODE=280 LEVEL= 3 NAME: _LC1_A11
(60) NODE=283 LEVEL= 3 NAME: _LC5_A10
(61) NODE= 10 LEVEL= 4 NAME: p1
(62) NODE= 17 LEVEL= 4 NAME: NOT_LC5_C3
(63) NODE= 18 LEVEL= 4 NAME: NOT_LC4_C3
(64) NODE= 20 LEVEL= 4 NAME: NOT_LC1_A12
(65) NODE= 21 LEVEL= 4 NAME: NOT_LC5_A10
(66) NODE= 22 LEVEL= 4 NAME: NOT_LC1_A11
(67) NODE= 24 LEVEL= 4 NAME: NOT_LC4_A10
(68) NODE= 28 LEVEL= 4 NAME: NOT_LC5_A4
(69) NODE= 30 LEVEL= 4 NAME: NOT_LC1_A13
(70) NODE= 31 LEVEL= 4 NAME: NOT_LC1_C7
(71) NODE= 38 LEVEL= 4 NAME: NOT_LC4_A1
(72) NODE= 79 LEVEL= 4 NAME: t13
(73) NODE= 80 LEVEL= 4 NAME: t14
(74) NODE= 82 LEVEL= 4 NAME: _LC6_C3
(75) NODE=101 LEVEL= 4 NAME: t27
(76) NODE=103 LEVEL= 4 NAME: t29
(77) NODE=113 LEVEL= 4 NAME: t37
(78) NODE=155 LEVEL= 4 NAME: t71
(79) NODE=157 LEVEL= 4 NAME: t73
(80) NODE=167 LEVEL= 4 NAME: t81
(81) NODE= 76 LEVEL= 5 NAME: t10
(82) NODE= 77 LEVEL= 5 NAME: t11
(83) NODE= 78 LEVEL= 5 NAME: t12
(84) NODE= 83 LEVEL= 5 NAME: t15
(85) NODE= 84 LEVEL= 5 NAME: t16
(86) NODE= 91 LEVEL= 5 NAME: t21
(87) NODE=100 LEVEL= 5 NAME: t26
(88) NODE=102 LEVEL= 5 NAME: t28
(89) NODE=110 LEVEL= 5 NAME: t34
(90) NODE=111 LEVEL= 5 NAME: t35
(91) NODE=112 LEVEL= 5 NAME: t36
(92) NODE=115 LEVEL= 5 NAME: t38
(93) NODE=122 LEVEL= 5 NAME: t44
(94) NODE=124 LEVEL= 5 NAME: t46
(95) NODE=154 LEVEL= 5 NAME: t70
(96) NODE=156 LEVEL= 5 NAME: t72
(97) NODE=164 LEVEL= 5 NAME: t78
(98) NODE=165 LEVEL= 5 NAME: t79
(99) NODE=166 LEVEL= 5 NAME: t80
(100) NODE= 81 LEVEL= 6 NAME: _LC1_C3

(101) NODE= 85 LEVEL= 6 NAME: _LC3_C3
(102) NODE= 92 LEVEL= 6 NAME: _LC2_C3
(103) NODE=104 LEVEL= 6 NAME: _LC7_A10
(104) NODE=114 LEVEL= 6 NAME: _LC6_A10
(105) NODE=158 LEVEL= 6 NAME: _LC5_A5
(106) NODE=168 LEVEL= 6 NAME: _LC6_A5
(107) NODE= 23 LEVEL= 7 NAME: NOT_LC6_A10
(108) NODE= 25 LEVEL= 7 NAME: NOT_LC1_C3
(109) NODE= 29 LEVEL= 7 NAME: NOT_LC2_C3
(110) NODE= 32 LEVEL= 7 NAME: NOT_LC6_A5
(111) NODE= 53 LEVEL= 7 NAME: NOT_LC5_A5
(112) NODE= 54 LEVEL= 7 NAME: NOT_LC7_A10
(113) NODE= 93 LEVEL= 7 NAME: _LC2_C9
(114) NODE=105 LEVEL= 7 NAME: t30
(115) NODE=106 LEVEL= 7 NAME: t31
(116) NODE=107 LEVEL= 7 NAME: t32
(117) NODE=118 LEVEL= 7 NAME: t41
(118) NODE=119 LEVEL= 7 NAME: t42
(119) NODE=121 LEVEL= 7 NAME: t43
(120) NODE=123 LEVEL= 7 NAME: t45
(121) NODE=128 LEVEL= 7 NAME: t49
(122) NODE=133 LEVEL= 7 NAME: t53
(123) NODE=139 LEVEL= 7 NAME: t58
(124) NODE=144 LEVEL= 7 NAME: t62
(125) NODE=146 LEVEL= 7 NAME: t64
(126) NODE=148 LEVEL= 7 NAME: t66
(127) NODE=159 LEVEL= 7 NAME: t74
(128) NODE=160 LEVEL= 7 NAME: t75
(129) NODE=161 LEVEL= 7 NAME: t76
(130) NODE= 94 LEVEL= 8 NAME: t22
(131) NODE=108 LEVEL= 8 NAME: t33
(132) NODE=116 LEVEL= 8 NAME: t39
(133) NODE=117 LEVEL= 8 NAME: t40
(134) NODE=125 LEVEL= 8 NAME: _LC1_A10
(135) NODE=135 LEVEL= 8 NAME: t55
(136) NODE=140 LEVEL= 8 NAME: t59
(137) NODE=145 LEVEL= 8 NAME: t63
(138) NODE=147 LEVEL= 8 NAME: t65
(139) NODE=149 LEVEL= 8 NAME: t67
(140) NODE=162 LEVEL= 8 NAME: t77
(141) NODE= 26 LEVEL= 9 NAME: NOT_LC1_A10
(142) NODE= 95 LEVEL= 9 NAME: _LC3_C9
(143) NODE=109 LEVEL= 9 NAME: _LC3_A10
(144) NODE=120 LEVEL= 9 NAME: _LC2_A10
(145) NODE=126 LEVEL= 9 NAME: t47
(146) NODE=127 LEVEL= 9 NAME: t48
(147) NODE=132 LEVEL= 9 NAME: t52



(148) NODE=134 LEVEL= 9 NAME: t54
(149) NODE=150 LEVEL= 9 NAME: _LC3_A4
(150) NODE=163 LEVEL= 9 NAME: _LC4_A5
(151) NODE= 19 LEVEL= 10 NAME: NOT_LC3_C9
(152) NODE= 33 LEVEL= 10 NAME: NOT_LC3_A10
(153) NODE= 34 LEVEL= 10 NAME: NOT_LC2_A10
(154) NODE= 39 LEVEL= 10 NAME: NOT_LC4_A5
(155) NODE= 96 LEVEL= 10 NAME: t23
(156) NODE= 97 LEVEL= 10 NAME: t24
(157) NODE=129 LEVEL= 10 NAME: t50
(158) NODE=130 LEVEL= 10 NAME: t51
(159) NODE=136 LEVEL= 10 NAME: _LC4_A4
(160) NODE=169 LEVEL= 10 NAME: t82
(161) NODE=170 LEVEL= 10 NAME: t83
(162) NODE=172 LEVEL= 10 NAME: t85
(163) NODE=175 LEVEL= 10 NAME: t87
(164) NODE=176 LEVEL= 10 NAME: t88
(165) NODE=180 LEVEL= 10 NAME: t91
(166) NODE=181 LEVEL= 10 NAME: t92
(167) NODE=187 LEVEL= 10 NAME: t96
(168) NODE=217 LEVEL= 10 NAME: t118
(169) NODE=218 LEVEL= 10 NAME: t119
(170) NODE=220 LEVEL= 10 NAME: t121
(171) NODE=223 LEVEL= 10 NAME: t123
(172) NODE=225 LEVEL= 10 NAME: t125
(173) NODE= 27 LEVEL= 11 NAME: NOT_LC4_A4
(174) NODE= 98 LEVEL= 11 NAME: t25
(175) NODE=131 LEVEL= 11 NAME: _LC1_A8
(176) NODE=141 LEVEL= 11 NAME: t60
(177) NODE=142 LEVEL= 11 NAME: t61
(178) NODE=151 LEVEL= 11 NAME: t68
(179) NODE=152 LEVEL= 11 NAME: t69
(180) NODE=171 LEVEL= 11 NAME: t84
(181) NODE=173 LEVEL= 11 NAME: t86
(182) NODE=177 LEVEL= 11 NAME: t89
(183) NODE=178 LEVEL= 11 NAME: t90
(184) NODE=182 LEVEL= 11 NAME: t93
(185) NODE=189 LEVEL= 11 NAME: t98
(186) NODE=219 LEVEL= 11 NAME: t120
(187) NODE=221 LEVEL= 11 NAME: t122
(188) NODE=224 LEVEL= 11 NAME: t124
(189) NODE=226 LEVEL= 11 NAME: t126
(190) NODE= 36 LEVEL= 12 NAME: NOT_LC1_A8
(191) NODE= 99 LEVEL= 12 NAME: _LC1_C9
(192) NODE=137 LEVEL= 12 NAME: t56
(193) NODE=138 LEVEL= 12 NAME: t57
(194) NODE=153 LEVEL= 12 NAME: _LC1_A4

(195) NODE=174 LEVEL= 12 NAME: _LC3_A5
(196) NODE=179 LEVEL= 12 NAME: _LC7_A5
(197) NODE=183 LEVEL= 12 NAME: _LC2_A1
(198) NODE=192 LEVEL= 12 NAME: t100
(199) NODE=222 LEVEL= 12 NAME: _LC5_A1
(200) NODE=227 LEVEL= 12 NAME: _LC8_A1
(201) NODE= 11 LEVEL= 13 NAME: p2
(202) NODE= 40 LEVEL= 13 NAME: NOT_LC8_A1
(203) NODE= 41 LEVEL= 13 NAME: NOT_LC3_A5
(204) NODE= 42 LEVEL= 13 NAME: NOT_LC2_A1
(205) NODE= 48 LEVEL= 13 NAME: NOT_LC1_C9
(206) NODE= 49 LEVEL= 13 NAME: NOT_LC1_A4
(207) NODE=143 LEVEL= 13 NAME: _LC2_A4
(208) NODE=184 LEVEL= 13 NAME: t94
(209) NODE=186 LEVEL= 13 NAME: t95
(210) NODE=188 LEVEL= 13 NAME: t97
(211) NODE=228 LEVEL= 13 NAME: t127
(212) NODE=231 LEVEL= 13 NAME: t130
(213) NODE=234 LEVEL= 13 NAME: t132
(214) NODE=235 LEVEL= 13 NAME: t133
(215) NODE= 52 LEVEL= 14 NAME: NOT_LC2_A4
(216) NODE=185 LEVEL= 14 NAME: _LC2_A5
(217) NODE=190 LEVEL= 14 NAME: _LC1_A5
(218) NODE=199 LEVEL= 14 NAME: t105
(219) NODE=229 LEVEL= 14 NAME: t128
(220) NODE=230 LEVEL= 14 NAME: t129
(221) NODE=232 LEVEL= 14 NAME: t131
(222) NODE=236 LEVEL= 14 NAME: t134
(223) NODE=237 LEVEL= 14 NAME: t135
(224) NODE= 35 LEVEL= 15 NAME: NOT_LC1_A5
(225) NODE= 44 LEVEL= 15 NAME: NOT_LC2_A5
(226) NODE=193 LEVEL= 15 NAME: t101
(227) NODE=195 LEVEL= 15 NAME: t102
(228) NODE=196 LEVEL= 15 NAME: t103
(229) NODE=233 LEVEL= 15 NAME: _LC3_A1
(230) NODE=238 LEVEL= 15 NAME: _LC6_A1
(231) NODE=241 LEVEL= 15 NAME: t138
(232) NODE= 12 LEVEL= 16 NAME: p3
(233) NODE= 43 LEVEL= 16 NAME: NOT_LC6_A1
(234) NODE=191 LEVEL= 16 NAME: t99
(235) NODE=197 LEVEL= 16 NAME: _LC1_B2
(236) NODE=242 LEVEL= 16 NAME: t139
(237) NODE=243 LEVEL= 16 NAME: t140
(238) NODE=244 LEVEL= 16 NAME: t141
(239) NODE=246 LEVEL= 16 NAME: t142
(240) NODE=247 LEVEL= 16 NAME: t143
(241) NODE=248 LEVEL= 16 NAME: t144

(242) NODE=249 LEVEL= 16 NAME: t145
(243) NODE= 37 LEVEL= 17 NAME: NOT_LC1_B2
(244) NODE=194 LEVEL= 17 NAME: _LC8_B2
(245) NODE=200 LEVEL= 17 NAME: t106
(246) NODE=202 LEVEL= 17 NAME: t107
(247) NODE=203 LEVEL= 17 NAME: t108
(248) NODE=239 LEVEL= 17 NAME: t136
(249) NODE=240 LEVEL= 17 NAME: t137
(250) NODE=250 LEVEL= 17 NAME: _LC1_A1
(251) NODE= 45 LEVEL= 18 NAME: NOT_LC1_A1
(252) NODE= 46 LEVEL= 18 NAME: NOT_LC8_B2
(253) NODE=198 LEVEL= 18 NAME: t104
(254) NODE=204 LEVEL= 18 NAME: _LC4_B2
(255) NODE=245 LEVEL= 18 NAME: _LC7_A1
(256) NODE=252 LEVEL= 18 NAME: t147
(257) NODE= 13 LEVEL= 19 NAME: p4
(258) NODE= 50 LEVEL= 19 NAME: NOT_LC4_B2
(259) NODE=201 LEVEL= 19 NAME: _LC2_B2
(260) NODE=251 LEVEL= 19 NAME: t146
(261) NODE=253 LEVEL= 19 NAME: t148
(262) NODE= 47 LEVEL= 20 NAME: NOT_LC2_B2
(263) NODE=254 LEVEL= 20 NAME: _LC7_B2
(264) NODE=255 LEVEL= 20 NAME: t149
(265) NODE=256 LEVEL= 20 NAME: t150
(266) NODE=257 LEVEL= 20 NAME: t151
(267) NODE= 14 LEVEL= 21 NAME: p5
(268) NODE=258 LEVEL= 21 NAME: t152
(269) NODE=260 LEVEL= 21 NAME: t153
(270) NODE=259 LEVEL= 22 NAME: _LC5_B2
(271) NODE=261 LEVEL= 22 NAME: _LC6_B2
(272) NODE= 15 LEVEL= 23 NAME: p6
(273) NODE= 51 LEVEL= 23 NAME: NOT_LC6_B2
(274) NODE=262 LEVEL= 23 NAME: t154
(275) NODE=263 LEVEL= 23 NAME: t155
(276) NODE=264 LEVEL= 23 NAME: t156
(277) NODE=265 LEVEL= 23 NAME: t157
(278) NODE=266 LEVEL= 24 NAME: t158
(279) NODE=267 LEVEL= 24 NAME: t159
(280) NODE=268 LEVEL= 24 NAME: t160
(281) NODE=269 LEVEL= 24 NAME: t161
(282) NODE=270 LEVEL= 25 NAME: _LC3_B2
(283) NODE= 16 LEVEL= 26 NAME: p7

B.5 SIS Activity and Power Estimation

The switching activity and power estimates provided by SIS are given below.

```
Script started on Fri Jun 11 02:25:50 1999
```

```
Combinational power estimation, with Zero delay model.
```

```
Network: bth4.blf, Power = 2292.5 uW assuming 20 MHz clock and Vdd = 5V
```

```
sis> time
```

```
elapsed: 1.0 seconds, total: 1.0 seconds
```

```
sis> power_print
```

```
Node a0    Cap. = 59 Switch Prob. = 0.50 Power = 73.8
Node a1    Cap. = 75 Switch Prob. = 0.50 Power = 93.8
Node a2    Cap. = 55 Switch Prob. = 0.50 Power = 68.8
Node a3    Cap. = 48 Switch Prob. = 0.50 Power = 60.0
Node x0    Cap. = 75 Switch Prob. = 0.50 Power = 93.8
Node x1    Cap. = 32 Switch Prob. = 0.50 Power = 40.0
Node x2    Cap. = 52 Switch Prob. = 0.50 Power = 65.0
Node x3    Cap. = 66 Switch Prob. = 0.50 Power = 82.5
Node [3388] Cap. = 0 Switch Prob. = 0.38 Power = 0.0
Node [3389] Cap. = 0 Switch Prob. = 0.47 Power = 0.0
Node [3390] Cap. = 0 Switch Prob. = 0.49 Power = 0.0
Node [3391] Cap. = 0 Switch Prob. = 0.50 Power = 0.0
Node [3392] Cap. = 0 Switch Prob. = 0.49 Power = 0.0
Node [3393] Cap. = 0 Switch Prob. = 0.46 Power = 0.0
Node [3394] Cap. = 0 Switch Prob. = 0.43 Power = 0.0
Node [3395] Cap. = 0 Switch Prob. = 0.29 Power = 0.0
Node _LC5_C3 Cap. = 11 Switch Prob. = 0.47 Power = 12.9
Node NOT_LC5_C3 Cap. = 8 Switch Prob. = 0.47 Power = 9.4
Node _LC4_C3 Cap. = 12 Switch Prob. = 0.22 Power = 6.6
Node NOT_LC4_C3 Cap. = 2 Switch Prob. = 0.22 Power = 1.1
Node _LC3_C9 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node NOT_LC3_C9 Cap. = 3 Switch Prob. = 0.12 Power = 0.9
Node _LC1_A12 Cap. = 46 Switch Prob. = 0.50 Power = 57.5
Node NOT_LC1_A12 Cap. = 24 Switch Prob. = 0.50 Power = 30.0
Node _LC5_A10 Cap. = 12 Switch Prob. = 0.50 Power = 15.0
Node NOT_LC5_A10 Cap. = 9 Switch Prob. = 0.50 Power = 11.2
Node _LC1_A11 Cap. = 26 Switch Prob. = 0.38 Power = 24.4
Node NOT_LC1_A11 Cap. = 18 Switch Prob. = 0.38 Power = 16.9
Node _LC6_A10 Cap. = 16 Switch Prob. = 0.34 Power = 13.7
Node NOT_LC6_A10 Cap. = 7 Switch Prob. = 0.34 Power = 6.0
Node _LC4_A10 Cap. = 17 Switch Prob. = 0.47 Power = 19.9
Node NOT_LC4_A10 Cap. = 14 Switch Prob. = 0.47 Power = 16.4
Node _LC1_C3 Cap. = 17 Switch Prob. = 0.47 Power = 19.9
Node NOT_LC1_C3 Cap. = 14 Switch Prob. = 0.47 Power = 16.4
Node _LC1_A10 Cap. = 16 Switch Prob. = 0.38 Power = 15.0
```

Node NOT_LC1_A10 Cap. = 7 Switch Prob. = 0.38 Power = 6.6
Node _LC4_A4 Cap. = 19 Switch Prob. = 0.38 Power = 17.8
Node NOT_LC4_A4 Cap. = 6 Switch Prob. = 0.38 Power = 5.6
Node _LC5_A4 Cap. = 23 Switch Prob. = 0.47 Power = 27.0
Node NOT_LC5_A4 Cap. = 20 Switch Prob. = 0.47 Power = 23.4
Node _LC2_C3 Cap. = 21 Switch Prob. = 0.12 Power = 6.2
Node NOT_LC2_C3 Cap. = 20 Switch Prob. = 0.12 Power = 5.9
Node _LC1_A13 Cap. = 39 Switch Prob. = 0.50 Power = 48.8
Node NOT_LC1_A13 Cap. = 13 Switch Prob. = 0.50 Power = 16.2
Node _LC1_C7 Cap. = 33 Switch Prob. = 0.38 Power = 30.9
Node NOT_LC1_C7 Cap. = 25 Switch Prob. = 0.38 Power = 23.4
Node _LC6_A5 Cap. = 16 Switch Prob. = 0.34 Power = 13.7
Node NOT_LC6_A5 Cap. = 7 Switch Prob. = 0.34 Power = 6.0
Node _LC3_A10 Cap. = 17 Switch Prob. = 0.47 Power = 19.9
Node NOT_LC3_A10 Cap. = 14 Switch Prob. = 0.47 Power = 16.4
Node _LC2_A10 Cap. = 13 Switch Prob. = 0.49 Power = 16.0
Node NOT_LC2_A10 Cap. = 9 Switch Prob. = 0.49 Power = 11.1
Node _LC1_A5 Cap. = 12 Switch Prob. = 0.40 Power = 12.1
Node NOT_LC1_A5 Cap. = 2 Switch Prob. = 0.40 Power = 2.0
Node _LC1_A8 Cap. = 10 Switch Prob. = 0.49 Power = 12.3
Node NOT_LC1_A8 Cap. = 6 Switch Prob. = 0.49 Power = 7.4
Node _LC1_B2 Cap. = 13 Switch Prob. = 0.22 Power = 7.1
Node NOT_LC1_B2 Cap. = 2 Switch Prob. = 0.22 Power = 1.1
Node _LC4_A1 Cap. = 16 Switch Prob. = 0.43 Power = 17.2
Node NOT_LC4_A1 Cap. = 7 Switch Prob. = 0.43 Power = 7.5
Node _LC4_A5 Cap. = 17 Switch Prob. = 0.47 Power = 19.9
Node NOT_LC4_A5 Cap. = 14 Switch Prob. = 0.47 Power = 16.4
Node _LC8_A1 Cap. = 16 Switch Prob. = 0.44 Power = 17.6
Node NOT_LC8_A1 Cap. = 7 Switch Prob. = 0.44 Power = 7.7
Node _LC3_A5 Cap. = 17 Switch Prob. = 0.49 Power = 20.9
Node NOT_LC3_A5 Cap. = 14 Switch Prob. = 0.49 Power = 17.2
Node _LC2_A1 Cap. = 20 Switch Prob. = 0.50 Power = 24.9
Node NOT_LC2_A1 Cap. = 18 Switch Prob. = 0.50 Power = 22.4
Node _LC6_A1 Cap. = 27 Switch Prob. = 0.44 Power = 29.9
Node NOT_LC6_A1 Cap. = 6 Switch Prob. = 0.44 Power = 6.7
Node _LC2_A5 Cap. = 19 Switch Prob. = 0.27 Power = 13.0
Node NOT_LC2_A5 Cap. = 18 Switch Prob. = 0.27 Power = 12.3
Node _LC1_A1 Cap. = 14 Switch Prob. = 0.25 Power = 8.7
Node NOT_LC1_A1 Cap. = 4 Switch Prob. = 0.25 Power = 2.5
Node _LC8_B2 Cap. = 8 Switch Prob. = 0.48 Power = 9.6
Node NOT_LC8_B2 Cap. = 11 Switch Prob. = 0.48 Power = 13.2
Node _LC2_B2 Cap. = 8 Switch Prob. = 0.43 Power = 8.7
Node NOT_LC2_B2 Cap. = 8 Switch Prob. = 0.43 Power = 8.7
Node _LC1_C9 Cap. = 18 Switch Prob. = 0.30 Power = 13.7
Node NOT_LC1_C9 Cap. = 16 Switch Prob. = 0.30 Power = 12.2
Node _LC1_A4 Cap. = 18 Switch Prob. = 0.09 Power = 4.0
Node NOT_LC1_A4 Cap. = 16 Switch Prob. = 0.09 Power = 3.6

Node _LC4_B2 Cap. = 18 Switch Prob. = 0.03 Power = 1.4
Node NOT_LC4_B2 Cap. = 16 Switch Prob. = 0.03 Power = 1.2
Node _LC6_B2 Cap. = 17 Switch Prob. = 0.01 Power = 0.3
Node NOT_LC6_B2 Cap. = 16 Switch Prob. = 0.01 Power = 0.3
Node _LC2_A4 Cap. = 12 Switch Prob. = 0.44 Power = 13.1
Node NOT_LC2_A4 Cap. = 7 Switch Prob. = 0.44 Power = 7.6
Node _LC5_A5 Cap. = 9 Switch Prob. = 0.47 Power = 10.5
Node NOT_LC5_A5 Cap. = 4 Switch Prob. = 0.47 Power = 4.7
Node _LC7_A10 Cap. = 9 Switch Prob. = 0.47 Power = 10.5
Node NOT_LC7_A10 Cap. = 4 Switch Prob. = 0.47 Power = 4.7
Node NOTa0 Cap. = 26 Switch Prob. = 0.50 Power = 32.5
Node NOTa1 Cap. = 21 Switch Prob. = 0.50 Power = 26.2
Node NOTa2 Cap. = 32 Switch Prob. = 0.50 Power = 40.0
Node NOTa3 Cap. = 21 Switch Prob. = 0.50 Power = 26.2
Node NOTx0 Cap. = 41 Switch Prob. = 0.50 Power = 51.2
Node NOTx1 Cap. = 21 Switch Prob. = 0.50 Power = 26.2
Node NOTx2 Cap. = 48 Switch Prob. = 0.50 Power = 60.0
Node NOTx3 Cap. = 20 Switch Prob. = 0.50 Power = 25.0
Node _LC1_C14 Cap. = 1 Switch Prob. = 0.38 Power = 0.9
Node _LC5_C6 Cap. = 3 Switch Prob. = 0.47 Power = 3.5
Node _LC5_A1 Cap. = 3 Switch Prob. = 0.49 Power = 3.7
Node _LC3_A1 Cap. = 3 Switch Prob. = 0.50 Power = 3.7
Node _LC7_A1 Cap. = 4 Switch Prob. = 0.49 Power = 4.9
Node _LC7_B2 Cap. = 2 Switch Prob. = 0.46 Power = 2.3
Node _LC5_B2 Cap. = 3 Switch Prob. = 0.43 Power = 3.2
Node _LC3_B2 Cap. = 5 Switch Prob. = 0.29 Power = 3.6
Node t0 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t1 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t2 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t3 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t4 Cap. = 4 Switch Prob. = 0.12 Power = 1.2
Node t5 Cap. = 4 Switch Prob. = 0.12 Power = 1.2
Node t6 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t7 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t8 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t9 Cap. = 6 Switch Prob. = 0.12 Power = 1.8
Node t10 Cap. = 6 Switch Prob. = 0.06 Power = 0.9
Node t11 Cap. = 6 Switch Prob. = 0.00 Power = 0.0
Node t12 Cap. = 5 Switch Prob. = 0.40 Power = 5.1
Node t13 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t14 Cap. = 5 Switch Prob. = 0.00 Power = 0.0
Node _LC6_C3 Cap. = 13 Switch Prob. = 0.49 Power = 16.0
Node t15 Cap. = 4 Switch Prob. = 0.17 Power = 1.7
Node t16 Cap. = 4 Switch Prob. = 0.12 Power = 1.2
Node _LC3_C3 Cap. = 3 Switch Prob. = 0.26 Power = 2.0
Node t17 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t18 Cap. = 5 Switch Prob. = 0.22 Power = 2.7

Node t19 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t20 Cap. = 6 Switch Prob. = 0.12 Power = 1.8
Node t21 Cap. = 3 Switch Prob. = 0.12 Power = 0.9
Node _LC2_C9 Cap. = 5 Switch Prob. = 0.49 Power = 6.1
Node t22 Cap. = 3 Switch Prob. = 0.12 Power = 0.9
Node t23 Cap. = 4 Switch Prob. = 0.00 Power = 0.0
Node t24 Cap. = 4 Switch Prob. = 0.00 Power = 0.0
Node t25 Cap. = 4 Switch Prob. = 0.30 Power = 3.0
Node t26 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t27 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t28 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t29 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t30 Cap. = 5 Switch Prob. = 0.38 Power = 4.7
Node t31 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t32 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t33 Cap. = 6 Switch Prob. = 0.12 Power = 1.8
Node t34 Cap. = 6 Switch Prob. = 0.17 Power = 2.5
Node t35 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t36 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t37 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t38 Cap. = 5 Switch Prob. = 0.30 Power = 3.8
Node t39 Cap. = 5 Switch Prob. = 0.17 Power = 2.1
Node t40 Cap. = 6 Switch Prob. = 0.24 Power = 3.6
Node t41 Cap. = 5 Switch Prob. = 0.17 Power = 2.1
Node t42 Cap. = 6 Switch Prob. = 0.03 Power = 0.5
Node t43 Cap. = 5 Switch Prob. = 0.17 Power = 2.1
Node t44 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t45 Cap. = 5 Switch Prob. = 0.06 Power = 0.8
Node t46 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t47 Cap. = 5 Switch Prob. = 0.16 Power = 2.0
Node t48 Cap. = 6 Switch Prob. = 0.03 Power = 0.5
Node t49 Cap. = 5 Switch Prob. = 0.30 Power = 3.8
Node t50 Cap. = 5 Switch Prob. = 0.18 Power = 2.3
Node t51 Cap. = 6 Switch Prob. = 0.24 Power = 3.6
Node t52 Cap. = 5 Switch Prob. = 0.19 Power = 2.4
Node t53 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t54 Cap. = 5 Switch Prob. = 0.03 Power = 0.4
Node t55 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t56 Cap. = 5 Switch Prob. = 0.00 Power = 0.0
Node t57 Cap. = 5 Switch Prob. = 0.32 Power = 4.0
Node t58 Cap. = 5 Switch Prob. = 0.03 Power = 0.4
Node t59 Cap. = 5 Switch Prob. = 0.28 Power = 3.6
Node t60 Cap. = 6 Switch Prob. = 0.00 Power = 0.0
Node t61 Cap. = 6 Switch Prob. = 0.06 Power = 0.9
Node t62 Cap. = 5 Switch Prob. = 0.06 Power = 0.8
Node t63 Cap. = 5 Switch Prob. = 0.17 Power = 2.1
Node t64 Cap. = 5 Switch Prob. = 0.03 Power = 0.4

Node t65 Cap. = 5 Switch Prob. = 0.28 Power = 3.6
Node t66 Cap. = 6 Switch Prob. = 0.00 Power = 0.0
Node t67 Cap. = 6 Switch Prob. = 0.22 Power = 3.3
Node _LC3_A4 Cap. = 11 Switch Prob. = 0.47 Power = 12.9
Node t68 Cap. = 4 Switch Prob. = 0.03 Power = 0.3
Node t69 Cap. = 4 Switch Prob. = 0.06 Power = 0.6
Node t70 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t71 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t72 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t73 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t74 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t75 Cap. = 5 Switch Prob. = 0.38 Power = 4.7
Node t76 Cap. = 5 Switch Prob. = 0.22 Power = 2.7
Node t77 Cap. = 6 Switch Prob. = 0.12 Power = 1.8
Node t78 Cap. = 6 Switch Prob. = 0.17 Power = 2.5
Node t79 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t80 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t81 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t82 Cap. = 5 Switch Prob. = 0.30 Power = 3.8
Node t83 Cap. = 5 Switch Prob. = 0.23 Power = 2.9
Node t84 Cap. = 6 Switch Prob. = 0.21 Power = 3.1
Node t85 Cap. = 5 Switch Prob. = 0.10 Power = 1.3
Node t86 Cap. = 6 Switch Prob. = 0.08 Power = 1.1
Node t87 Cap. = 5 Switch Prob. = 0.09 Power = 1.1
Node t88 Cap. = 5 Switch Prob. = 0.24 Power = 3.0
Node t89 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t90 Cap. = 5 Switch Prob. = 0.14 Power = 1.8
Node _LC7_A5 Cap. = 10 Switch Prob. = 0.40 Power = 9.9
Node t91 Cap. = 4 Switch Prob. = 0.34 Power = 3.4
Node t92 Cap. = 4 Switch Prob. = 0.34 Power = 3.4
Node t93 Cap. = 4 Switch Prob. = 0.24 Power = 2.4
Node t94 Cap. = 2 Switch Prob. = 0.27 Power = 1.4
Node t95 Cap. = 5 Switch Prob. = 0.17 Power = 2.1
Node t96 Cap. = 5 Switch Prob. = 0.26 Power = 3.3
Node t97 Cap. = 5 Switch Prob. = 0.06 Power = 0.8
Node t98 Cap. = 5 Switch Prob. = 0.12 Power = 1.5
Node t99 Cap. = 4 Switch Prob. = 0.40 Power = 4.0
Node t100 Cap. = 4 Switch Prob. = 0.34 Power = 3.4
Node t101 Cap. = 4 Switch Prob. = 0.10 Power = 1.0
Node t102 Cap. = 3 Switch Prob. = 0.19 Power = 1.4
Node t103 Cap. = 3 Switch Prob. = 0.04 Power = 0.3
Node t104 Cap. = 4 Switch Prob. = 0.39 Power = 3.9
Node t105 Cap. = 4 Switch Prob. = 0.27 Power = 2.7
Node t106 Cap. = 4 Switch Prob. = 0.05 Power = 0.5
Node t107 Cap. = 4 Switch Prob. = 0.02 Power = 0.2
Node t108 Cap. = 4 Switch Prob. = 0.01 Power = 0.1
Node t109 Cap. = 2 Switch Prob. = 0.38 Power = 1.9

Node t110	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t111	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t112	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t113	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t114	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t115	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t116	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t117	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t118	Cap. = 5	Switch Prob. = 0.30	Power = 3.8
Node t119	Cap. = 5	Switch Prob. = 0.19	Power = 2.4
Node t120	Cap. = 6	Switch Prob. = 0.17	Power = 2.5
Node t121	Cap. = 5	Switch Prob. = 0.14	Power = 1.8
Node t122	Cap. = 6	Switch Prob. = 0.12	Power = 1.8
Node t123	Cap. = 5	Switch Prob. = 0.17	Power = 2.1
Node t124	Cap. = 5	Switch Prob. = 0.26	Power = 3.3
Node t125	Cap. = 5	Switch Prob. = 0.06	Power = 0.8
Node t126	Cap. = 5	Switch Prob. = 0.22	Power = 2.7
Node t127	Cap. = 5	Switch Prob. = 0.34	Power = 4.3
Node t128	Cap. = 5	Switch Prob. = 0.24	Power = 3.0
Node t129	Cap. = 6	Switch Prob. = 0.16	Power = 2.4
Node t130	Cap. = 5	Switch Prob. = 0.15	Power = 1.9
Node t131	Cap. = 6	Switch Prob. = 0.10	Power = 1.6
Node t132	Cap. = 5	Switch Prob. = 0.19	Power = 2.4
Node t133	Cap. = 5	Switch Prob. = 0.10	Power = 1.3
Node t134	Cap. = 5	Switch Prob. = 0.24	Power = 3.0
Node t135	Cap. = 5	Switch Prob. = 0.19	Power = 2.4
Node t136	Cap. = 5	Switch Prob. = 0.07	Power = 0.8
Node t137	Cap. = 5	Switch Prob. = 0.36	Power = 4.5
Node t138	Cap. = 5	Switch Prob. = 0.10	Power = 1.3
Node t139	Cap. = 5	Switch Prob. = 0.27	Power = 3.4
Node t140	Cap. = 6	Switch Prob. = 0.02	Power = 0.2
Node t141	Cap. = 6	Switch Prob. = 0.12	Power = 1.8
Node t142	Cap. = 6	Switch Prob. = 0.05	Power = 0.8
Node t143	Cap. = 6	Switch Prob. = 0.09	Power = 1.3
Node t144	Cap. = 6	Switch Prob. = 0.02	Power = 0.2
Node t145	Cap. = 6	Switch Prob. = 0.12	Power = 1.8
Node t146	Cap. = 4	Switch Prob. = 0.43	Power = 4.3
Node t147	Cap. = 4	Switch Prob. = 0.29	Power = 2.9
Node t148	Cap. = 4	Switch Prob. = 0.03	Power = 0.3
Node t149	Cap. = 5	Switch Prob. = 0.27	Power = 3.4
Node t150	Cap. = 5	Switch Prob. = 0.37	Power = 4.6
Node t151	Cap. = 5	Switch Prob. = 0.25	Power = 3.2
Node t152	Cap. = 6	Switch Prob. = 0.01	Power = 0.1
Node t153	Cap. = 3	Switch Prob. = 0.01	Power = 0.1
Node t154	Cap. = 6	Switch Prob. = 0.00	Power = 0.0
Node t155	Cap. = 6	Switch Prob. = 0.00	Power = 0.0
Node t156	Cap. = 6	Switch Prob. = 0.00	Power = 0.0

```

Node t157 Cap. = 6 Switch Prob. = 0.00 Power = 0.0
Node t158 Cap. = 6 Switch Prob. = 0.00 Power = 0.0
Node t159 Cap. = 6 Switch Prob. = 0.02 Power = 0.3
Node t160 Cap. = 6 Switch Prob. = 0.03 Power = 0.5
Node t161 Cap. = 6 Switch Prob. = 0.25 Power = 3.8
Node t162 Cap. = 3 Switch Prob. = 0.38 Power = 2.8
Node t163 Cap. = 3 Switch Prob. = 0.38 Power = 2.8
Node t164 Cap. = 2 Switch Prob. = 0.38 Power = 1.9
Node t165 Cap. = 3 Switch Prob. = 0.38 Power = 2.8
Node t166 Cap. = 3 Switch Prob. = 0.38 Power = 2.8
Node t167 Cap. = 2 Switch Prob. = 0.38 Power = 1.9
Node t168 Cap. = 3 Switch Prob. = 0.38 Power = 2.8
Node t169 Cap. = 3 Switch Prob. = 0.38 Power = 2.8
Total Power: 2292.467728

```

B.6 BLAPE Activity and Power Estimation

The switching activity and power estimates provided by BLAPE are given below.

Activity Computation delay = 0.03 Seconds

Node	Activity	Capacitance
a0	0.5000	cap=59
a1	0.5000	cap=75
a2	0.5000	cap=55
a3	0.5000	cap=48
x0	0.5000	cap=75
x1	0.5000	cap=32
x2	0.5000	cap=52
x3	0.5000	cap=66
p0	0.3750	cap=0
p1	0.4851	cap=0
p2	0.4982	cap=0
p3	0.4992	cap=0
p4	0.4994	cap=0
p5	0.4736	cap=0
p6	0.3864	cap=0
p7	0.4304	cap=0
NOT_LC5_C3	0.4672	cap=8
NOT_LC4_C3	0.2129	cap=2
NOT_LC3_C9	0.1318	cap=3
NOT_LC1_A12	0.4922	cap=24
NOT_LC5_A10	0.4922	cap=9

NOT_LC1_A11	0.3750	cap=18
NOT_LC6_A10	0.3811	cap=7
NOT_LC4_A10	0.4851	cap=14
NOT_LC1_C3	0.4978	cap=14
NOT_LC1_A10	0.4250	cap=7
NOT_LC4_A4	0.4493	cap=6
NOT_LC5_A4	0.4672	cap=20
NOT_LC2_C3	0.1303	cap=20
NOT_LC1_A13	0.4922	cap=13
NOT_LC1_C7	0.3750	cap=25
NOT_LC6_A5	0.3811	cap=7
NOT_LC3_A10	0.4901	cap=14
NOT_LC2_A10	0.4971	cap=9
NOT_LC1_A5	0.4587	cap=2
NOT_LC1_A8	0.4999	cap=6
NOT_LC1_B2	0.2831	cap=2
NOT_LC4_A1	0.4851	cap=7
NOT_LC4_A5	0.4901	cap=14
NOT_LC8_A1	0.4736	cap=7
NOT_LC3_A5	0.4951	cap=14
NOT_LC2_A1	0.5000	cap=18
NOT_LC6_A1	0.4687	cap=6
NOT_LC2_A5	0.2629	cap=18
NOT_LC1_A1	0.2908	cap=4
NOT_LC8_B2	0.4960	cap=11
NOT_LC2_B2	0.4918	cap=8
NOT_LC1_C9	0.4082	cap=16
NOT_LC1_A4	0.1377	cap=16
NOT_LC4_B2	0.0690	cap=16
NOT_LC6_B2	0.0344	cap=16
NOT_LC2_A4	0.4975	cap=7
NOT_LC5_A5	0.4758	cap=4
NOT_LC7_A10	0.4758	cap=4
NOTa0	0.5000	cap=26
NOTa1	0.5000	cap=21
NOTa2	0.5000	cap=32
NOTa3	0.5000	cap=21
NOTx0	0.5000	cap=41
NOTx1	0.5000	cap=21
NOTx2	0.5000	cap=48
NOTx3	0.5000	cap=20
t0	0.2188	cap=5
t1	0.2188	cap=5
t2	0.2188	cap=5
t3	0.2188	cap=5
_LC4_A10	0.4851	cap=16
t4	0.1172	cap=3

t5	0.1172	cap=3
_LC4_C3	0.2129	cap=12
t6	0.2188	cap=5
t7	0.2188	cap=5
t8	0.2188	cap=5
t9	0.1172	cap=5
_LC5_C3	0.4672	cap=10
t10	0.0373	cap=5
t11	0.0373	cap=5
t12	0.4401	cap=5
t13	0.1687	cap=5
t14	0.1687	cap=5
_LC1_C3	0.4978	cap=16
_LC6_C3	0.4927	cap=13
t15	0.1303	cap=3
t16	0.1303	cap=3
_LC3_C3	0.2339	cap=3
t17	0.2188	cap=5
t18	0.2188	cap=5
t19	0.2188	cap=5
t20	0.1172	cap=5
_LC5_A4	0.4672	cap=22
t21	0.1303	cap=2
_LC2_C3	0.1303	cap=20
_LC2_C9	0.4909	cap=5
t22	0.1318	cap=2
_LC3_C9	0.1318	cap=4
t23	0.0684	cap=4
t24	0.0684	cap=4
t25	0.3566	cap=4
_LC1_C9	0.4082	cap=18
t26	0.2158	cap=5
t27	0.1948	cap=5
t28	0.2158	cap=5
t29	0.1948	cap=5
_LC7_A10	0.4758	cap=8
t30	0.4139	cap=5
t31	0.3139	cap=5
t32	0.3425	cap=5
t33	0.0645	cap=5
_LC3_A10	0.4901	cap=16
t34	0.1506	cap=5
t35	0.1307	cap=5
t36	0.1307	cap=5
t37	0.1172	cap=5
_LC6_A10	0.3811	cap=15
t38	0.3572	cap=5

t39	0.2604	cap=5
t40	0.1726	cap=5
t41	0.1004	cap=5
t42	0.0635	cap=5
_LC2_A10	0.4971	cap=12
t43	0.1464	cap=5
t44	0.2622	cap=5
t45	0.0723	cap=5
t46	0.1358	cap=5
_LC1_A10	0.4250	cap=15
t47	0.1329	cap=5
t48	0.0689	cap=5
t49	0.3874	cap=5
t50	0.2715	cap=5
t51	0.1486	cap=5
_LC1_A8	0.4999	cap=9
t52	0.1916	cap=5
t53	0.2890	cap=5
t54	0.0783	cap=5
t55	0.1243	cap=5
_LC4_A4	0.4493	cap=18
t56	0.0563	cap=5
t57	0.3520	cap=5
t58	0.0483	cap=5
t59	0.3134	cap=5
t60	0.0077	cap=5
t61	0.1590	cap=5
_LC2_A4	0.4975	cap=10
t62	0.0430	cap=5
t63	0.2861	cap=5
t64	0.0430	cap=5
t65	0.2861	cap=5
t66	0.0129	cap=5
t67	0.2494	cap=5
_LC3_A4	0.4939	cap=9
t68	0.0729	cap=3
t69	0.0729	cap=3
_LC1_A4	0.1377	cap=18
t70	0.2158	cap=5
t71	0.1948	cap=5
t72	0.2158	cap=5
t73	0.1948	cap=5
_LC5_A5	0.4758	cap=8
t74	0.3425	cap=5
t75	0.4139	cap=5
t76	0.3139	cap=5
t77	0.0645	cap=5

_LC4_A5	0.4901	cap=16
t78	0.1506	cap=5
t79	0.1307	cap=5
t80	0.1307	cap=5
t81	0.1172	cap=5
_LC6_A5	0.3811	cap=15
t82	0.4357	cap=5
t83	0.3342	cap=5
t84	0.1301	cap=5
t85	0.1354	cap=5
t86	0.0470	cap=5
_LC3_A5	0.4951	cap=16
t87	0.1951	cap=5
t88	0.3362	cap=5
t89	0.0535	cap=5
t90	0.1017	cap=5
_LC7_A5	0.4584	cap=9
t91	0.3554	cap=4
t92	0.3848	cap=4
t93	0.2076	cap=4
_LC2_A1	0.5000	cap=20
t94	0.2629	cap=2
_LC2_A5	0.2629	cap=18
t95	0.2162	cap=5
t96	0.2865	cap=5
t97	0.0911	cap=5
t98	0.1254	cap=5
_LC1_A5	0.4587	cap=11
t99	0.4326	cap=4
t100	0.4001	cap=4
t101	0.1459	cap=4
_LC8_B2	0.4960	cap=8
t102	0.2282	cap=3
t103	0.0865	cap=3
_LC1_B2	0.2831	cap=13
t104	0.4737	cap=4
t105	0.3861	cap=4
t106	0.0768	cap=4
_LC2_B2	0.4918	cap=8
t107	0.0507	cap=3
t108	0.0198	cap=3
_LC4_B2	0.0690	cap=18
t109	0.3750	cap=2
_LC1_C14	0.3750	cap=0
t110	0.2188	cap=5
t111	0.2188	cap=5
t112	0.2188	cap=5

t113	0.2188	cap=5
_LC5_C6	0.4851	cap=2
t114	0.2188	cap=5
t115	0.2188	cap=5
t116	0.2188	cap=5
t117	0.2188	cap=5
_LC4_A1	0.4851	cap=15
t118	0.4076	cap=5
t119	0.2784	cap=5
t120	0.1180	cap=5
t121	0.2081	cap=5
t122	0.0850	cap=5
_LC5_A1	0.4982	cap=2
t123	0.2445	cap=5
t124	0.1918	cap=5
t125	0.2081	cap=5
t126	0.1620	cap=5
_LC8_A1	0.4736	cap=15
t127	0.3986	cap=5
t128	0.2809	cap=5
t129	0.1289	cap=5
t130	0.1892	cap=5
t131	0.0829	cap=5
_LC3_A1	0.4992	cap=2
t132	0.2371	cap=5
t133	0.1892	cap=5
t134	0.1998	cap=5
t135	0.1583	cap=5
_LC6_A1	0.4687	cap=26
t136	0.0929	cap=5
t137	0.3875	cap=5
t138	0.0751	cap=5
t139	0.3320	cap=5
t140	0.0286	cap=5
t141	0.1463	cap=5
_LC7_A1	0.4994	cap=2
t142	0.0289	cap=5
t143	0.1452	cap=5
t144	0.0286	cap=5
t145	0.1463	cap=5
_LC1_A1	0.2908	cap=13
t146	0.4947	cap=4
t147	0.3963	cap=4
t148	0.0772	cap=4
_LC7_B2	0.4736	cap=2
t149	0.4257	cap=5
t150	0.4975	cap=5

t151	0.4050	cap=5
t152	0.0344	cap=5
_LC5_B2	0.3864	cap=2
t153	0.0344	cap=2
_LC6_B2	0.0344	cap=16
t154	0.0001	cap=5
t155	0.0003	cap=5
t156	0.0007	cap=5
t157	0.0221	cap=5
t158	0.0015	cap=5
t159	0.0454	cap=5
t160	0.0956	cap=5
t161	0.3756	cap=5
_LC3_B2	0.4304	cap=2
t162	0.3750	cap=3
t163	0.3750	cap=3
_LC1_A13	0.4922	cap=39
t164	0.3750	cap=2
_LC1_C7	0.3750	cap=0
t165	0.3750	cap=3
t166	0.3750	cap=3
_LC1_A12	0.4922	cap=46
t167	0.3750	cap=2
_LC1_A11	0.3750	cap=0
t168	0.3750	cap=3
t169	0.3750	cap=3
_LC5_A10	0.4922	cap=12

Power = 2289.34 uW assuming 20 MHz CLK, Vdd = 5V

The Bar and Level views of the 4-bit Booth multiplier are illustrated in Figures 6.5 and 6.4. The BLAPE implementation, using a depth-accuracy of 1 performs the activity and power estimation in just 0.03 seconds; SIS runs in 1.0 seconds. The percent error, when compared to SIS, is just 0.13%.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] M. Pedram. Power Minimization in IC Design: Principles and Applications. *ACM Trans. on Design Automation of Electronic Systems*, 1(1):3–56, 1996.
- [2] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley Company, Reading, MA, 1985.
- [3] S. Devadas and S. Malik. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. In *Proceedings of the 32nd Design Automation Conference*, pages 242–247, June 1995.
- [4] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-Power CMOS Design. *IEEE Journal Of Solid-State Circuits*, 27(4):472–484, April 1992.
- [5] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On Average Power Dissipation and Random Pattern Testability of Combinational Logic Circuits. In *Proceedings of the Int'l Conference on Computer-Aided Design*, pages 402–407, November 1992.
- [6] W. Swartz and C. Sechen. New Algorithms for the Placement and Routing of Macro Cells. *IEEE International Conference on Computer-Aided Design*, pages 336–339, 1990.
- [7] A. P. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Brodersen. Design of Portable Systems. In *Proceedings of CICC*, pages 12.1.1–12.1.8, May 1994.
- [8] D-S Chen, M. Sarrafzadeh, and G. K. H. Yeap. State Encoding of Finite State Machines for Low Power Design. In *Proceeding of the International Symposium on Circuits and Systems*, volume 3, pages 2309–2312, 1995.
- [9] G. E. Sobelman and D. L. Raatz. Low-Power Multiplier Design Using Delayed Evaluation. In *Proceeding of the International Symposium on Circuits and Systems*, pages 1564–1566, 1995.

- [10] V. G. Moshnyaga and K. Tamaru. A Comparative Study of Switching Activity Reduction Techniques for Design of Low-Power Multipliers. In *Proceedings of the International Symposium on Circuits and Systems*, pages 1560–1563, 1995.
- [11] C-Y. Tsui, M. Pedram, and A. M. Despain. Technology Decomposition and Mapping Targeting Low Power Dissipation. In *Proceedings of the 30th Design Automation Conference*, pages 68–73, June 1993.
- [12] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar. Scheduling Techniques to Enable Power Management. In *Proceedings of the 33rd Design Automation Conference*, pages 349–352, June 1996.
- [13] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-Based Sequential Logic Optimization for Low Power. In *Proceedings of the Int'l Conference on Computer-Aided Design*, pages 74–81, November 1994.
- [14] C. K. Lennard, P. Buch, and A. R. Newton. Logic Synthesis Using Power-Sensitive Don't Care Sets. In *Proceeding of the ACM Int'l Symposium on Low Power Electronics & Design*, August 1996.
- [15] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *IEEE Transactions on VLSI Systems*, 2(4):437–445, December 1994.
- [16] W. C. Athas, L. J. Svensson, J. G. Koller, N. Tzartzanis, and E. Y-C Chou. Low-Power Digital Systems Based on Adiabatic-Switching Principles. *IEEE Transactions On VLSI Systems*, 2(4):398–407, December 1994.
- [17] D. Conner. Making the jump to HDL-based programmable-logic design. *EDN Magazine*, pages 181–187, September 1997.
- [18] S. Katkoori, N. Kumar, and R. Vemuri. High Level Profiling Based Low Power Synthesis Technique. In *Proceedings of International Conference on Computer Design*, pages 759–765, October 2–5 1995.
- [19] R. Burch, F. Najm, P. Yang, and T. Trick. A Monte Carlo Approach for Power Estimation. *IEEE Transactions on VLSI Systems*, 1(1):63–71, March 1993.
- [20] S. M. Ross. *Introduction to Probability Models*. Academic Press Inc., San Diego, CA 92101, 5th edition, 1993.

- [21] K. S. Shanmugan and A. M. Breipohl. *Random Signals: Detection, Estimation and Data Analysis*. John Wiley & Sons, Inc., New York, NY, 1st edition, 1988.
- [22] G. I. Stamoulis. A Monte-Carlo Approach for the Accurate and Efficient Estimation of Average Transition Probabilities in Sequential Logic Circuits. In *Proceedings of the IEEE 1996 Custom IC Conference*, pages 221–224, May 1996.
- [23] M. Xakellis and F. Najm. Statistical Estimation of the Switching Activity in Digital Circuits. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pages 728–733, 1994.
- [24] F. N. Najm, S. Goel, and I. N. Hajj. Power Estimation in Sequential Circuits. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 635–640, June 1995.
- [25] F. N. Najm and M. G. Xakellis. Statistical Estimation of the Switching Activity in VLSI Circuits. *VLSI Design*, 1996.
- [26] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of Average Switching Activity in Combinational and Sequential Circuits. In *Proceedings of the 29th Design Automation Conference*, pages 253–259, June 1992.
- [27] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, c-35:677–691, August 1986.
- [28] S. Chakravarty. On the complexity of using BDDs for the synthesis and analysis of boolean circuits. In *Proceedings of the 27th Annual Allerton Conference on Communication, Control and Computing*, pages 730–739, 1989.
- [29] F. Najm. Transition Density: A New Measure of Activity in Digital Circuits. *IEEE Transactions on Computer-Aided Design*, 12(2):310–323, February 1993.
- [30] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45, June 1990.
- [31] Bhanu Kapoor. Improving the Accuracy of Circuit Activity Measurement. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pages 734–739, 1994.
- [32] C-Y. Tsui, M. Pedram, and A. M. Despain. Exact and Approximate Methods For Calculating Signal and Transition Probabilities in FSMs. In *Proceeding of the 31st Design Automation Conference*, pages 19–23, June 1994.

- [33] J. Monteiro, S. Devadas, and B. Lin. A Methodology for Efficient Estimation of Switching Activity in Sequential Logic Circuits. In *Proceedings of the 31st Design Automation Conference*, pages 12–17, June 1994.
- [34] D. I. Cheng, M. Marek-Sadowska, and K-T Cheng. Speeding Up Power Estimation by Topological Analysis. *IEEE Custom Integrated Circuits Conference*, pages 623–626, 1995.
- [35] S. Katkoori, L. Rader N. Kumar, and R. Vemuri. A Profile Driven Approach for Low Power Synthesis. In *Proceedings of VLSI*, pages 759–765, Japan, 1995.
- [36] P. E. Landman and J. Rabaey. Power Estimation for High Level Synthesis. In *Proceedings of the European Conference on Design Automation*, pages 361–366, February 1993.
- [37] M. Nemani and F. N. Najm. Towards a High-Level Power Estimation Capability. *IEEE Transactions on Computer-Aided Design*, 15(6):588–598, June 1996.
- [38] F. Najm. Towards a high-level power estimation capability. In *Proceedings of the 1995 International Symposium on Low Power Design*, pages 87–92, April 23–26 1995.
- [39] Gary Hachtel and Fabio Somenzi. *Logic Synthesis And Verification Algorithms*. Kluwer Academic Publishers, Norwell, MA 02061, 1996.
- [40] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technology J*, 38:985–999, July 1959.
- [41] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, c-27(6):509–516, June 1978.
- [42] R. E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. *International Conference on Computer-Aided Design*, pages 236–243, November 1995.
- [43] A. Shen, S. Devadas, and A. Ghosh. Probabilistic Manipulation of Boolean Functions Using Free Boolean Diagrams. *IEEE Transactions on Computer-Aided Design*, 14(1):87–95, January 1995.
- [44] L. F. Saunders R. Camposano and R. M. Tabet. VHDL as Input for High-Level Synthesis. *IEEE Design & Test of Computers*, pages 43–49, March 1991.

- [45] F. Mailhot and G. D. Michelo. Technology mapping using Boolean matching. In *European Conference on Design Automation*, pages 180–185, Mar. 1990.
- [46] Ronnie L. Wright and Michael A. Shanblatt. Reducing BDD Size By Exploiting Structural Connectivity. In *Proceeding of the 9th Great Lakes Symposium on VLSI (to appear)*, March 1999.
- [47] Lili Pan Sharad C. Seth and Vishwani D. Agrawal. PREDICT - Probabilistic Estimation of Digital Circuit Testability. *IEEE International Symposium on Fault-Tolerant Computing*, pages 220–225, June 1985.
- [48] J. E. Hopcroft A. V. Aho and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1st edition, 1974.
- [49] M. Jimenez A. Diaz and M. Shanblatt. Integer Pair Representation of Binary Terms and Equations. In *Proceeding of the 41th Midwest Symposium on Circuits and Systems*, August 1998.
- [50] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White. Estimation of Average Switching Activity in Combinational Logic Circuits Using Symbolic Simulation. *IEEE Transactions on Computer-Aided Design*, 16(1):121–127, January 1997.
- [51] H. Mehta, M. Borah, R. M. Owens, and M. J. Irwin. Accurate Estimation of Combinational Circuit Activity. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 618–622, June 1995.
- [52] R. S. Martin and J. Knight. Power-Profiler: Optimizing ASIC's Power Consumption At The Behavioral Level. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 42–47, June 1995.
- [53] F. N. Najm. Power Estimation Techniques for Integrated Circuits. *IEEE/ACM International Conference on Computer-Aided Design*, pages 492–499, 1995.

MICHIGAN STATE UNIV. LIBRARIES



31293020589879