A Power Reduction Algorithm for Combinational CMOS
Circuits using Input Disabling

by

William John Rinderknecht

Submitted to the Department of Electrical Engineering and Computer Science
on January 20, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

Abstract

With the advent of portable electronics, low power CMOS circuit design has become increasingly important. To effectively design for low power, chip designers need a wide variety of power estimation and power reduction tools. This thesis describes a new CAD tool that reduces the power dissipation of combinational CMOS circuits.

This automated CAD tool attempts to reduce power by selectively “turning off” inputs. It searches a gate-level description of a CMOS circuit for inputs or nodes that are often unnecessary to determine the outputs. It will then add logic to dynamically “turn off” these nodes based on the values of other inputs. This reduces switching activity within the circuit thus reducing power consumption. The result is a functionally equivalent CMOS circuit with reduced power, added area, and possibly increased delay.

This thesis describes the technique and heuristic algorithms used by this CAD tool to optimize circuits for low power. Experimental results are presented.

Thesis Supervisor: Srinivas Devadas
Title: Associate Professor of Electrical Engineering and Computer Science
Contents

1 Introduction ........................................... 11

2 Terminology ........................................... 15
   2.1 Definitions ....................................... 15
   2.2 Binary Decision Diagrams ....................... 16

3 A Power Dissipation Model ......................... 19
   3.1 Power Consumption of Nodes ..................... 19
   3.2 Power Estimation of Dynamic Combinational Circuits .... 20
   3.3 Power Estimation of Static Combinational Circuits .... 21
       3.3.1 Switching Activity With a Zero Delay Model .... 22
       3.3.2 Switching Activity With a General Delay Model .. 22

4 Previous Work ....................................... 25
   4.1 The Basic Technique ............................. 25
   4.2 Deriving the Precomputation Logic ............... 28
   4.3 Input Selection .................................. 29
   4.4 More on Multiple Output Circuits ................ 30
   4.5 Results ......................................... 32

5 Architecture for Combinational Precomputation .... 35
   5.1 The Hardware .................................... 35
   5.2 The Disable Logic Requirements ................ 37
6 Algorithms

6.1 Synthesis of the Disable Logic ........................................ 39
6.2 Reduction of Disable Logic Costs ...................................... 40
6.3 Selection of the Inputs ................................................... 42

7 Subcircuit Selection ........................................................ 45

7.1 Motivation ................................................................. 45
7.2 Observations About Single-Output Subcircuits .................... 47
7.3 The First Algorithm .................................................... 50
7.4 The Second Algorithm ................................................ 52

8 Results, Comparisons, and Future Work ............................... 55

8.1 Results ........................................................................... 55
8.2 Comparison to Alidina’s Technique ................................... 56
8.3 Future Work ................................................................. 57

9 Conclusion ......................................................................... 59
List of Figures

2-1 Examples of Ordered Binary Decision Diagrams .................. 18
4-1 The Original Circuit ............................................ 25
4-2 Single Output Precomputation Architecture ...................... 26
4-3 Precomputation of a Multiple-Output Function .................. 27
4-4 Procedure to Determine the Optimal Set of Inputs .............. 31
4-5 Logic Duplication in a Multiple-Output Circuit ................ 33
5-1 Original circuit .................................................. 35
5-2 Circuit with input disabling circuit ............................. 36
5-3 Disabling inputs in combinational circuits ....................... 36
5-4 Disabling inputs in Domino circuits ............................. 37
5-5 Single output circuit with input disabling circuit .......... 38
6-1 Procedure to reduce the cost of the disable logic ............. 41
6-2 Procedure to Determine the Optimal Set of Inputs ............. 43
7-1 Two candidates for power reduction. (a) A simple 8-bit comparator.
(b) An 8-bit comparator preceded by two adders ................. 46
7-2 Procedure to Find the Minimum Set of Single-Output Subcircuits .. 48
7-3 Dividing a circuit into a minimum number of single-output subcircuits.
(a) The original circuit. (b) The single-output subcircuits .......... 49
7-4 Example of branching through subcircuit combinations .......... 51
7-5 Algorithm that evaluate different combinations of subcircuits .... 53
Possible groupings of adjacent MSSO subcircuits. (a) Subcircuits sharing an input. (b) A subcircuit output feeding a subcircuit input. (c) A subcircuit output and each of its fanouts.
List of Tables

4.1 Power Reductions for Datapath Circuits ................ 34
4.2 Power Reductions for Random Logic Circuits ............. 34
8.1 Power reductions of combinational circuits ............... 56
Chapter 1

Introduction

Low power CMOS circuit design has become increasingly important. As die sizes and clock frequencies increase at an astonishing rate, the average power consumption of chips also increase. High-performance RISC microprocessors that are not designed using low power techniques can require as much as 30 watts of power [7]. This increased power consumption can lead to reduced performance, higher chip failure rates, and higher costs due to power supplies and/or cooling mechanisms.

More importantly, the advent of portable electronics has fueled the need for low power CMOS circuits. Because of the rate at which CMOS circuits have shrunk, the weight and the volume of many portable electronic devices is now dominated by the power supplies. The weight and volume of power supplies are decreasing slowly so that advances in this area of research are unlikely to alleviate the problem. In order to achieve smaller portable devices, the average power consumption of CMOS circuits must be reduced.

To this date, research in power estimation of CMOS circuits has been been fairly complete. Static power dissipation of CMOS circuits has been shown to be negligible in comparison to dynamic dissipation. Because of this, average power dissipation is directly proportional to the average switching activity of a circuit [9]. Methods for estimating power dissipation by first estimating switching activity are given in [12, 8]. In [16], Monteiro builds on these techniques to develop a computationally efficient algorithm to estimate the switching activity and power consumption of sequential...
Designers use a great variety of techniques for *power reduction*. Most commonly, designers use techniques at the circuit level to reduce power. For example, designers often reduce the power supply voltage or use alternative circuit techniques, such as Domino circuits [5, 17]. Designers also make architectural and functional changes to reduce power consumption. For example, many chips disable local clocks when not needed [14], and microprocessors often include wait commands that put the chips in low-power states.

There have also been several techniques used to reduce power consumption at the gate level. In [15], power is reduced using several methods including exploiting don’t cares, using disjoint covers, collapsing gates, and using new cost functions with standard optimization routines. State encoding [13] and re-timing [11] algorithms have also been developed for low power logic synthesis. In [2], Alidina presents a technique that “turns off” certain inputs of a sequential circuit when their values are not needed. This approach adds logic (to be evaluated in the previous clock cycle) to determine when these inputs can be turned off. Thus this technique reduces power consumption at the expense of chip area.

This thesis presents a new approach, based on Alidina’s technique, that will selectively “turn off” nodes of combinational circuits to reduce switching activity. Effectively, this approach searches a combinational circuit for inputs or internal nodes whose values are often unnecessary to determine the values of the outputs. It then adds logic to “turn off” or gate these signals when they are not needed. This reduces switching activity within the circuit without changing the values of the outputs. The result is a functionally equivalent combinational circuit that requires less power.

Because this new approach is based closely on Alidina’s technique, it uses modified versions of the algorithms developed by Alidina in [2]. Specifically the algorithm that determines which inputs will disabled and the algorithm that synthesizes the added logic are closely based on the algorithms produced by Alidina. But in the Alidina algorithm, all added logic is evaluated on the previous clock cycle alleviating any possible complications due to timing. Because this algorithm works on combinational
circuits, timing is crucial to the success of the algorithm and cannot be ignored. The Alidina paper also assumes that the cost of additional logic will be low compared to the gross savings, therefore no attempt is made to minimize this cost. In this new algorithm, a procedure has been added that attempts to reduce the cost of added logic without a large decrease in the gross savings.

The primary addition of this new technique is the selection of the subcircuit. The Alidina algorithm works on sequential circuits and places its added logic in the previous cycle. Although this simplifies the timing constraints, it limits the choice of the disabled nodes to the circuit inputs only. In the algorithms described in this thesis, all of the nodes in the circuit are considered. This is done by dividing the combinational circuit into many combinational subcircuits. Then each of these subcircuits and each combination of these subcircuits is considered for power reduction in a computationally efficient manner. This results in an algorithm that is much more computationally complex than the Alidina algorithm, but it is also capable of considering many more possibilities making it a more powerful algorithm.

This thesis describes the theory and the implementation of this new power reduction technique. Chapters 2 and 3 contain background information about Boolean terminology and about power estimation techniques respectively. Chapter 4 describes Alidina’s power reduction technique. The new power reduction technique presented in this paper is described in Chapters 5 through 8. Chapter 5 describes the hardware architecture used to implement this technique. Chapters 6 and 7 describe the algorithms used to optimize the technique. Finally, Chapter 8 presents experimental results.
Chapter 2

Terminology

This chapter introduces terminology needed to describe the power estimation and reduction methods described in subsequent chapters. In Section 2.1, definitions pertaining to Boolean functions are given. Section 2.2 describes Binary Decision Diagrams – graphical representations of Boolean functions needed for efficient Boolean manipulation.

2.1 Definitions

The definitions in this section are taken from [6] and [2].

A Boolean function $f$ of $n$ input variables, $x_1, \ldots, x_n$, and of $m$ output variables, $f_1, \ldots, f_m$, is a mapping $f : B^n \to B^m$, where $B^n = \{0, 1\}^n$ and $B^m = \{0, 1\}^m$. For each output $f_i$ of $f$, the ON-set can be defined to be the set of inputs $x$ such that $f_i(x) = 1$. Similarly, the OFF-set is the set of inputs $x$ such that $f_i(x) = 0$. A function in which $m = 1$ is a single-output function, and a function with $m > 1$ is a multiple-output function.

The support of $f$, denoted as $support(f)$, is the set of all variables $x_i$ that occur in $f$ as $x_i$ or $\overline{x_i}$. For example, if $f = x_1 \cdot \overline{x_2} + x_3$, then $support(f) = \{x_1, x_2, x_3\}$.

A literal is a Boolean variable or its complement. If $x_i$ is a Boolean variable, then $x_i$ and $\overline{x_i}$ are literals.

A cube is a set of literals that represent the intersection of the given literals. If $a,
b, and c are Boolean variables, then \( a\overline{b}c \) is a cube representing the intersection of \( a \), \( \overline{b} \), and c.

A cover is a set of cubes such that the union of the cubes exactly defines some Boolean function, \( f \). In other words, member of the ON-set of \( f \) is included in at least one cube of the cover, and no member of the OFF-set of \( f \) is included in any of the cubes in the cover. A disjoint cover is a cover for a function such that each of the cubes is mutually exclusive from all the others.

The cofactor of a function \( f \) with respect to a variable \( x_i \), denoted as \( f_{x_i} \), is defined as:

\[
f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)
\]

Likewise, the cofactor of a function \( f \) with respect to a variable \( \overline{x_i} \), denoted as \( f_{\overline{x_i}} \), is defined as:

\[
f_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)
\]

The Shannon expansion of function around a variable \( x_i \) is given by:

\[
f = x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}}
\]

2.2 Binary Decision Diagrams

A Binary Decision Diagram (BDD) [1, 10] is a rooted, directed graph with vertex set, \( V \), containing two types of vertices. A nonterminal vertex \( v \) has as attributes an argument index, \( \text{index}(v) \in \{1, \ldots, n\} \), and two children, \( \text{low}(v), \text{high}(v) \in V \). A terminal vertex, \( v \), has as an attribute a value, \( \text{value}(v) \in \{0, 1\} \).

The correspondence between BDDs and Boolean functions is defined as follows. A BDD, \( G \), having root vertex, \( v \), denotes a function, \( f_v \), defined recursively as:

1. If \( v \) is a terminal vertex:
   
   (a) If \( \text{value}(v) = 1 \), then \( f_v = 1 \).
   
   (b) If \( \text{value}(v) = 0 \), then \( f_v = 0 \).
2. If \( v \) is a nonterminal vertex with \( \text{index}(v) = i \), then \( f_v \) is the function:

\[
f_v(x_1, \cdots, x_n) = \overline{x_i} \cdot f_{\text{low}(v)}(x_1, \cdots, x_n) + x_i \cdot f_{\text{high}(v)}(x_1, \cdots, x_n)
\]

(2.4)

where \( x_i \) is the decision variable for vertex \( v \).

*Ordered* BDDs (OBDDs) have a restriction such that for any nonterminal vertex \( v \), if \( \text{low}(v) \) is also nonterminal, then \( \text{index}(v) < \text{index}(\text{low}(v)) \). Similarly, if \( \text{high}(v) \) is also nonterminal, then \( \text{index}(v) < \text{index}(\text{high}(v)) \). The OBDDs for some simple functions are shown in Figure 2-1. Terminal vertices are represented as squares, while nonterminal vertices are represented as circles. The low child is pointed to by the arrow marked 0, and the high child is pointed to by the arrow marked 1.

*Reduced* OBDDs (ROBDDs) as proposed in [3] are a minimal OBDD representation for a given function and are defined as follows:

**Definition 2.1** An OBDD \( G \) is reduced if it contains no vertex \( v \) with \( \text{low}(v) = \text{high}(v) \) nor does it contain distinct vertices \( v \) and \( w \) such that the subgraphs rooted by \( v \) and \( w \) are isomorphic.

In [3], Bryant also proves that an ROBDD is a canonical representation of a Boolean function. ROBDDs are used to represent logic functions in the following power estimation and reduction techniques.
Figure 2-1: Examples of Ordered Binary Decision Diagrams
Chapter 3

A Power Dissipation Model

In order to evaluate power reduction algorithms, accurate and efficient power estimation algorithms are needed. The following sections briefly describe the model and basic algorithms used to estimate power consumption for CMOS circuits.

3.1 Power Consumption of Nodes

The power dissipation at a node in a CMOS circuit is directly proportional to the switching activity at the node. This requires the following three assumptions:

- The only capacitance in a CMOS logic-gate is at the output node of the gate.

- Either current is flowing through some path from $V_{DD}$ to the output capacitor, or current is flowing from the output capacitor to ground.

- Any change in a gate output voltage is a change from $V_{DD}$ to ground or vice-versa.

These are reasonably accurate assumptions for well-designed CMOS circuits. Further, it can be shown that the energy dissipated by a CMOS logic gate each time its output changes is roughly equal to the change in energy stored in the gate’s output capacitance [9]. If the gate is part of a synchronous digital system controlled by a
global clock, then the average power dissipated by the gate is given by:

\[ P_{\text{avg}} = \frac{1}{2} \times C_{\text{load}} \times V_{dd}^2 \times f_{\text{clock}} \times E(\text{transitions}) \]  \hspace{1cm} (3.1)

where \( P_{\text{avg}} \) denotes the average power, \( C_{\text{load}} \) is the load capacitance, \( V_{dd} \) is the supply voltage, \( f_{\text{clock}} \) is the global clock frequency, and \( E(\text{transitions}) \) is the expected number of gate transitions per clock cycle [12]. All of the variables in (3.1) can be determined from technology or circuit layout except \( E(\text{transitions}) \) which depends on the logic function being performed, the circuit style being used, and the statistical properties of the primary inputs [8]. Because of this, the major difficulty in estimating the power consumption of a circuit is determining the expected switching activity of the nodes in the circuit.

### 3.2 Power Estimation of Dynamic Combinational Circuits

In dynamic circuits, such as Domino circuits, nodes are pre-charged to a 1 or a 0. Then, during evaluation, the nodes switch only if the actual Boolean value is opposite the pre-charge value. This means one logic level results in two transitions whereas the other results in zero transitions independent of the node’s value in previous cycles. Therefore, in the case of dynamic circuits, Equation (3.1) can be simplified to:

\[ P_{\text{avg}} = C_{\text{load}} \times V_{dd}^2 \times f_{\text{clock}} \times \text{Prob}(f = 1) \]  \hspace{1cm} (3.2)

where \( f \) is the Boolean function of a particular node in terms of the circuit inputs, \( \text{Prob}(f = 1) \) is the probability that the node will evaluate to a 1, and \( f \) is assumed to be pre-charged to a 0. Once the \( \text{Prob}(f = 1) \) is known for each node, the power can easily be summed over all nodes.

To determine \( \text{Prob}(f = 1) \), two assumptions must be made. First, the probability of each input being a 1 is known and is denoted by \( p_{i}^{\text{one}} \) for input i. Second, the input probabilities, \( p_{1}^{\text{one}} \cdots p_{n}^{\text{one}} \), are uncorrelated.
Given these assumptions, \( \text{Prob}(f = 1) \) can be determined easily using elementary probability. If a particular cube is given by

\[
c_i = i_{a_1} \cdot i_{a_2} \cdot \ldots \cdot i_{a_r} \cdot \overline{i_{\beta_1}} \cdot i_{\beta_2} \cdot \ldots \cdot \overline{i_{\beta_t}}
\]  

(3.3)

then, because the \( i_j \)'s are uncorrelated, the probability of this cube being true is

\[
P(c_i = 1) = p_{a_1}^{\text{one}} \cdot p_{a_2}^{\text{one}} \cdot \ldots \cdot p_{a_r}^{\text{one}} \cdot (1 - p_{\beta_1}^{\text{one}}) \cdot (1 - p_{\beta_2}^{\text{one}}) \cdot \ldots \cdot (1 - p_{\beta_t}^{\text{one}})
\]  

(3.4)

If a Boolean function is expressed as a disjoint cover (i.e., a mutually exclusive group of cubes), then the probability that this function is true is simply the sum of the probabilities that each cube is true. Therefore, \( \text{prob}(f = 1) \) can be found by expressing \( f \) as a disjoint cover and then summing the probabilities of the cubes [8]. Because Binary Decision Diagrams are closely related to disjoint covers, exact probabilities of Boolean functions can be obtained in linear time using Binary Decision Diagrams [4, 12]. Therefore, given circuit input probabilities, all node probabilities can be determined and Equation 3.2 can be used to determine the average power dissipation of dynamic circuits.

### 3.3 Power Estimation of Static Combinational Circuits

For static circuits, Equation (3.1) is used directly by the power estimation techniques such as [8, 12] to relate switching activity to power dissipation. The power estimation algorithm need only determine the expected number of transitions at each node and then sum the power for all nodes.

In combinational circuits, switching may occur whenever there is a change in the circuit inputs. Because most combinational circuits exist within sequential systems, inputs usually change together, and all switching is allowed to complete before the inputs change again. So to determine the switching activity of a node, one needs to know what state nodes were in before the inputs change and what state the nodes
will be in after all switching is complete. This is equivalent to knowing a pair of consecutive input vectors, \((I_t, I_{t+1})\).

Techniques for estimating switching activity of static circuits are reviewed in the following sections.

### 3.3.1 Switching Activity With a Zero Delay Model

Assuming gates have zero delay, each node can make at most one transition for each input vector. Assuming that consecutive input vectors are independent, the probability that an input vector pair results in a 0 → 1 transition is \(P_{\text{node}}^{\text{one}} (1 - P_{\text{node}}^{\text{one}})\), where \(P_{\text{node}}^{\text{one}}\) denotes the probability that the node evaluates to a 1. \(P_{\text{node}}^{\text{one}}\) can be determined as described in Section 3.2 assuming the circuit input probabilities are known and independent. Similarly, the probability of a 1 → 0 transition is \((1 - P_{\text{node}}^{\text{one}})P_{\text{node}}^{\text{one}}\). Therefore, the expected number of transitions per clock cycle is given by

\[
E(\text{transitions}) = 2P_{\text{node}}^{\text{one}}(1 - P_{\text{node}}^{\text{one}})
\]

Equation 3.5 can be substituted directly into Equation 3.1 to determine the power dissipation assuming gates have zero delay [8].

### 3.3.2 Switching Activity With a General Delay Model

Under the zero delay model, all nodes transition at most once per clock cycle. In general this is not true. Because of timing delays, nodes can glitch resulting in multiple transitions in a clock cycle. To evaluate power consumption under a general delay model, symbolic simulation is used.

In symbolic simulation, a Boolean function is constructed for every interval of time during the clock cycle. For example, a node would be assigned the functions \(f_{i,0}, f_{i,1}, \ldots, f_{i,n}\) if there are \(n\) time intervals in a clock cycle. The support of these functions include variables from both input vectors, \(\langle I_t, I_{t+1} \rangle\). A transition of a 0 → 1 between intervals \(j\) and \(j+1\) is represented by the function \(\overline{f_{i,j}} \cdot f_{i,j+1}\). Therefore, the probability of a 0 → 1 transition occurring between \(j\) and \(j+1\) is the probability that
\( f_{i,j} \cdot f_{i,j+1} \) evaluates to a 1. Similarly, a 1 \( \rightarrow 0 \) transition is represented by \( f_{i,j} \cdot \overline{f_{i,j+1}} \). Therefore the probability that any transition will occur between \( j \) and \( j + 1 \) at this node is equal to the probability that

\[
f_{i,j} \cdot \overline{f_{i,j+1}} + \overline{f_{i,j}} \cdot f_{i,j+1} = f_{i,j} \oplus f_{i,j+1}
\]

(3.6)

will evaluate to a 1, where \( \oplus \) represents the exclusive-or operator. The average switching activity can be determined by simply summing these probabilities over the entire clock cycle:

\[
E(\text{transitions}) = \sum_{j=0}^{n-1} \text{Prob}(f_{i,j} \oplus f_{i,j+1} = 1)
\]

(3.7)

Once again, these probabilities can be evaluated as described in section 3.2.

Equation 3.7 can be substituted directly into Equation 3.1 to determine the power dissipation of static combinational circuits with general gate delays [8].
Chapter 4

Previous Work

As was stated earlier, the power reduction technique presented in this thesis is based closely on Alidina’s work presented in [2]. Alidina presents a technique that “turns off” certain inputs of a sequential circuit when the values are not needed. This approach adds logic (to be evaluated in the previous clock cycle) to determine when these inputs can be turned off.

This chapter describes the details of Alidina’s technique.

4.1 The Basic Technique

Alidina’s technique starts with a general sequential circuit as shown in Figure 4-1. Block A is combinational logic whereas Blocks R1 and R2 are registers. Although R1 and R2 are shown as separate registers, they could in fact be the same register. Assume, for the time, that Block A has only a single output, \( f \).

\[
\begin{array}{c}
\text{x}_1 \\
\text{x}_2 \\
\vdots \\
\text{x}_n
\end{array}
\rightarrow
\begin{array}{c}
\text{R1} \\
\text{A} \\
\text{R2}
\end{array}
\rightarrow
f
\]

Figure 4-1: The Original Circuit
To reduce the switching activity in Block A, circuitry is added to prevent some of the inputs from switching when their values are not needed. This is accomplished by using a register with a load enable signal as shown in Figure 4-2.

To ensure that the function of the circuit does not change, the *precomputation logic* must be selected correctly. The precomputation logic is the logic that determines when the inputs may be disabled. It is called *precomputation* logic because it is evaluated in the previous clock cycle. This logic is shown as Block $g_1$, Block $g_2$, and the NOR gate in Figure 4-2. To keep the operation of the circuit the same, the inputs stored in R2 can be disabled only when the output of Block A is completely determined by the remaining inputs (those in R1). To do this, the *predictor functions*, $g_1$ and $g_2$, are defined as:

$$g_1 = 1 \Rightarrow f = 1$$

$$g_2 = 1 \Rightarrow f = 0$$

where $\text{support}(g_1)$ and $\text{support}(g_2)$ include only inputs that are not being disabled. Under this definition, if $g_1$ is true, then the value of $f$ is known to be a 1 regardless of the values of the R2 inputs. Therefore, these inputs can be disabled without affecting $f$. The same argument can be made for $g_2$ where we know that $f$ will be a 0 independent of the disabled inputs. If neither $g_1$ nor $g_2$ is asserted, then nothing
Figure 4-3: Precomputation of a Multiple-Output Function

can be determined about the value of the \( f_i \), and all the inputs must be allowed to propagate. Therefore, the precomputation logic is defined to be \( g = g_1 + g_2 \), where \( g_1 \) and \( g_2 \) must satisfy Equations 4.1 and 4.2.

In general, circuits will have more than one output. Figure 4-3 shows the architecture generalized for multiple outputs. In this case, the inputs can be disabled only when all of the outputs are independent of the disabled inputs. The predictor functions are defined for each output:

\[
g_{1,i} = 1 \Rightarrow f_i = 1
\]

\[
g_{2,i} = 1 \Rightarrow f_i = 0
\]

for all \( i \) such that \( 1 \leq i \leq m \). Because every output must be independent of the disabled inputs, the disable signal can be asserted only in the intersection of the individual \( g_1 \) and \( g_2 \) signals:

\[
g = \prod_{i=1}^{m} (g_{1,i} + g_{2,i})
\]

This condition is required to ensure that each output is implemented correctly.

Given this architecture, two details must be determined for a given circuit. First, the subset of inputs that will be disabled must be determined. An algorithm to perform this selection will be described in Section 4.3. Second, given a particular subset of inputs to be disabled, the precomputation logic, \( g \), must be determined. This will be described in Section 4.2.
4.2 Deriving the Precomputation Logic

Given a particular sequential circuit, the precomputation logic, \( g \), must be determined. This logic must be selected so that the function of the circuit does not change. It should also be selected to maximize the probability of disabling the inputs.

First consider the simplified case in which there is one output, \( f \), and in which only one input will be disabled, \( x_i \). The algorithm must determine the functions \( g_1 \) and \( g_2 \) such that neither \( g_1 \) nor \( g_2 \) is a function of \( x_i \), such that Equations 4.1 and 4.2 are satisfied, and such that \( \text{prob}(g_1 + g_2 = 1) \) is maximized.

This can be accomplished using the universal quantification of \( f \). In a sense, the cofactor of \( f \) with respect to \( x_i \), \( f_{x_i} \), defines the set of input vectors that make \( f \) true given that \( x_i \) is true. (The cofactor function was defined in Section 2.1.) Similarly, \( f_{\overline{x_i}} \) defines the set of input vectors that make \( f \) true given that \( x_i \) is false. Therefore the intersection of the two defines the set of input vectors that force \( f \) to be true regardless of the value of \( x_i \). This is exactly the condition needed to fulfill Equation 4.1, and is defined as the universal quantification of \( f \) with respect to the variable \( x_i \):

\[
U_{x_i} f = f_{x_i} \cdot f_{\overline{x_i}}
\]

Because \( U_{x_i} f \) includes all input vectors that fulfill Equation 4.1, it also maximizes \( \text{Prob}(g_1 = 1) \). Similarly, if \( g_2 \) is defined as:

\[
g_2 = U_{x_i} \overline{f} = \overline{f_{x_i}} \cdot \overline{f_{\overline{x_i}}}
\]

then \( g_2 \) satisfies Equation 5.2 with maximum \( \text{Prob}(g_2 = 1) \).

Now consider the case in which many inputs will be disabled. Assume that the set of inputs to be disabled is given by \( D = \{x_{p+1}, \cdots, x_n\} \), the set of inputs that will not be disabled is given by \( S = \{x_1, \cdots, x_p\} \), and the total set of inputs is given by \( X = \{x_1, \cdots, x_n\} \) where \( 1 \leq p < n \). To find the set of input vectors that force \( f \) to be true regardless of the value of each variable in \( D \), the universal quantification \( f \) must be taken with respect to each variable in succession. This is the universal quantification of \( f \) with respect to \( D \), defined as:

\[
U_D f = U_{x_{p+1}} U_{x_{p+2}} \cdots U_{x_n} f
\]
This was proven formally by Alidina in [2]:

**Theorem 4.1** $g_1 = U_D f = U_{x_{p+1}} \cdots U_{x_n} f$ satisfies Equation 4.1. Further, no function $h(x_1, \cdots, x_p)$ exists such that $\text{prob}(h = 1) > \text{prob}(g_1 = 1)$ and such that $h = 1 \Rightarrow f = 1$.

Similarly, the function $g_2$ that satisfies Equation 4.2 and maximizes $\text{Prob}(g_2 = 1)$ is:

$$g_2 = U_D \overline{f} = U_{x_{p+1}} U_{x_{p+2}} \cdots U_{x_n} \overline{f}$$

Therefore, the $g_1$ and $g_2$ that satisfy the above three requirements can be determined by calculating $g_1 = U_D f$ and $g_2 = U_D \overline{f}$.

Finally, consider the case in which the circuit has multiple outputs. As was described in the previous section, every output must be independent of the disabled inputs in order to assert the precomputation signal. Therefore, the inputs can be turned off in the intersection of the individual precomputation signals:

$$g = \prod_{i=1}^{m} (g_{1,i} + g_{2,i}) = (U_D f_1 + U_D \overline{f_1}) \cdot (U_D f_2 + U_D \overline{f_2}) \cdot \cdots \cdot (U_D f_m + U_D \overline{f_m}) \quad (4.6)$$

Using this logic for $g$ ensures that the function of the circuit will not be changed satisfying Equations 4.3 and 4.4 and also maximizes $\text{prob}(g = 1)$.

### 4.3 Input Selection

In addition to determining the precomputation logic, Alidina also presents an algorithm to select $D$, the subset of inputs that will be disabled. To maximize power savings, it is desirable to select $D$ such that the probability of disabling the inputs, namely $\text{prob}(g = 1)$, is maximized. Alidina presents an algorithm that finds the set that maximizes this probability given a particular number of inputs, $k$.

This algorithm basically branches through a binary tree, where each node represents one input. The left branch from this node leads to all combinations of inputs that include the given input and the right branch leads to all combinations that do not include the given input. This branching continues until $k$ inputs have been selected.
at which point the \( \text{prob}(g_1 + g_2 = 1) \) can be determined. If allowed to follow all possible paths, this scheme would find the \( D \) of size \( k \) that maximizes \( \text{prob}(g_1 + g_2 = 1) \) simply because it would cover all possible combinations. Yet such an algorithm is too computationally complex.

To make this algorithm feasible, Alidina’s algorithm skips many possible branches along the binary tree. Skipping branches is possible because of the following observation:

\[
\text{prob}(U_x,f) = \text{prob}(f_{x_1} \cdot f_{x_2}) \leq \text{prob}(f) \quad \forall x_i, f
\]  

(4.7)

This shows that \( \text{prob}(g = 1) \) decreases monotonically as new inputs are added to \( D \). Therefore, if the \( \text{prob}(g = 1) \) becomes too small during the branching, all succeeding branches of the binary tree may be skipped.

The details of this algorithm, taken directly from [2], are shown in Figure 4-4. In the pseudo-code, \( D \) represents the set of inputs that are currently selected for disabling. \( Q \) represents the “active” inputs that may still be selected to be placed in \( D \). \( l \) is simply the number of inputs that will be placed in \( D \).

Each call of \text{SELECT\_RECUR} is analogous to a node in the binary branching pattern. The two recursive calls are analogous to the left and right children. The pruning condition suggested by Equation 4.7 is implemented as:

\[
\text{if } (pr \leq BEST\_PROB) \text{ return;}
\]

This algorithm efficiently finds the set of \( k \) inputs that maximize \( \text{prob}(g = 1) \). If the algorithm is run several times with different values of \( k \), the optimal solution will be found.

4.4 More on Multiple Output Circuits

The algorithms presented so far assume that all the outputs of Block A will be used in Equation 4.6 to determine the precomputation logic. This seems necessary to ensure the function of each output remains the same. Yet this creates a severe limitation. Each output effectively places a restriction on when it is allowable to disable the
SELECT_INPUTS( f, k ):
{
    BEST_PROB = 0 ;
    SELECTED_SET = φ ;
    SELECT_RECUR( f, f, 0, X, return( SELECTED_SET ) ;
}

SELECT_RECUR( g1, g2, D, Q, l ) :
{
    if( |D| + |Q| < l )
        return ;
    pr = prob( g1 = 1 ) + prob( g2 = 1 ) ;
    if( pr < BEST_PROB )
        return ;
    else if( |D| == l ) {
        BEST_PROB = pr ;
        SELECTED_SET = X −D ;
        return ;
    }
    choose xi ∈ Q such that i is minimum ;
    SELECT_RECUR( Uxi g1, Uxi g2, D ∪ xi, Q − xi, l ) ;
    SELECT_RECUR( g1, g2, D, Q − xi, l ) ;
    return ;
}

Figure 4-4: Procedure to Determine the Optimal Set of Inputs
inputs. As the number of outputs increase, the probability that the inputs can be disabled decreases. For even a reasonable number of outputs, the probability can become quite small, and thus the power reduction can become negligible.

To overcome this, Alidina suggests using logic duplication. The idea is to synthesize the precomputation logic, \( g \), using only a subset of the outputs in Equation 4.6. This results in two subsets of outputs – outputs whose values are unaffected if the inputs are disabled and outputs whose function will change if the inputs are disabled. To prevent the function of this second group from changing, any logic that is shared by both subsets of outputs is duplicated. In this way, the inputs may be disabled without affecting the outputs that did not contribute to the precomputation logic synthesis.

For example, consider the circuit in Figure 4-5(a). In this circuit, it may be desirable to disable the input \( x_3 \), but the combination of the outputs \( f_1 \) and \( f_2 \) reduces \( \text{prob}(g = 1) \) too much. Therefore, the shared logic (shown as the shaded area) is duplicated as in Figure 4-5(b). Now \( x_3 \) may be disabled without affecting \( f_2 \).

Obviously, this is not a prefect solution. This technique creates a considerable amount of overhead including the duplicated logic and the extra register. But it does enable the algorithm to work on circuits with a large number of outputs.

4.5 Results

Alidina implemented his techniques in C code within the SIS logic optimization system. Using this implementation, he demonstrated very good results attaining as much as 60% power reductions. Some of his results are shown in Tables 4.1 and 4.2 which were taken directly from [2].
Figure 4-5: Logic Duplication in a Multiple-Output Circuit
Table 4.1: Power Reductions for Datapath Circuits

<table>
<thead>
<tr>
<th>CKT</th>
<th>Original</th>
<th>Precompute Logic</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lits</td>
<td>Levs</td>
<td>Pwr</td>
</tr>
<tr>
<td>comp16</td>
<td>286</td>
<td>7</td>
<td>1281</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>priority16</td>
<td>126</td>
<td>16</td>
<td>455</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>add_comp16</td>
<td>3026</td>
<td>8</td>
<td>6941</td>
</tr>
<tr>
<td></td>
<td>4/8</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>max16</td>
<td>350</td>
<td>9</td>
<td>1744</td>
</tr>
<tr>
<td>csa16</td>
<td>975</td>
<td>10</td>
<td>2945</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>add_max16</td>
<td>3090</td>
<td>9</td>
<td>7370</td>
</tr>
<tr>
<td></td>
<td>4/8</td>
<td>24</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.2: Power Reductions for Random Logic Circuits

<table>
<thead>
<tr>
<th>CKT</th>
<th>Original</th>
<th>Precompute Logic</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lits</td>
<td>Levs</td>
<td>Pwr</td>
</tr>
<tr>
<td>9symml</td>
<td>267</td>
<td>8</td>
<td>1452</td>
</tr>
<tr>
<td>cm150a</td>
<td>61</td>
<td>5</td>
<td>744</td>
</tr>
<tr>
<td>cm152a</td>
<td>28</td>
<td>4</td>
<td>370</td>
</tr>
<tr>
<td>i2</td>
<td>230</td>
<td>4</td>
<td>5606</td>
</tr>
<tr>
<td>majority</td>
<td>12</td>
<td>4</td>
<td>173</td>
</tr>
<tr>
<td>mux</td>
<td>54</td>
<td>6</td>
<td>715</td>
</tr>
<tr>
<td>parity</td>
<td>60</td>
<td>5</td>
<td>187</td>
</tr>
<tr>
<td>t481</td>
<td>1028</td>
<td>11</td>
<td>1562</td>
</tr>
</tbody>
</table>

Table 4.2: Power Reductions for Random Logic Circuits
Chapter 5

Architecture for Combinational Precomputation

Chapter 4 described Alidina’s technique for power reduction. In this technique, inputs of a sequential circuit are selectively turned off to reduce switching activity.

The following chapters present a new technique for power reduction. This technique, based on Alidina’s technique, turns off inputs to combinational circuits to reduce switching activity. This chapter gives an overview of this new technique by describing the hardware and requirements necessary to implement the technique.

5.1 The Hardware

This technique starts with a gate-level description of a combinational circuit. (This circuit may be a complete circuit or it may be a subcircuit that was extracted from

![Figure 5-1: Original circuit.](image)
a larger circuit. This will be explained in more detail in Chapter 7. Assume that this circuit has \( n \) inputs and \( m \) outputs as is shown in Figure 5-1. In an effort to reduce switching activity, the algorithm will “turn off” a subset of the \( n \) inputs using the circuit shown in Figure 5-2. The figure shows \( p \) inputs being turned off using block B where \( 1 < p < n \). Assume that the set of inputs to be disabled is signified by \( S = \{x_1, x_2, \ldots, x_p\} \), and the set of inputs that will not be disabled is signified by \( D = \{x_{p+1}, x_{p+2}, \ldots, x_n\} \).

The term “turn off” means different things according to the type of circuit style that is being used. If the circuit is built using static logic gates, then “turn off” means prevent changes at the inputs from propagating through block B to block A. In this case block B may be implemented using one of the latches shown in Figure 5-3. If the circuit is built using dynamic logic, then “turn off” means prevent the outputs of block B from changing from the pre-charged value. Assuming the nodes pre-charge
Block $g$, the disable logic, determines when it is appropriate to turn off the selected inputs. The logic is selected so that the inputs are disabled as frequently as possible without affecting the values at the outputs of the circuit. The next section presents restrictions for Block $g$ that ensure that the function of the circuit does not change.

5.2 The Disable Logic Requirements

Block $g$ of Figure 5-2 determines when it is appropriate to turn off the selected inputs. The selected inputs may be “turned off” if the static value of all the outputs, $f_1$ through $f_m$, can be completely determined by the inputs that are not turned off, $x_{p+1} \ldots x_n$.

First consider the single-output case as shown in Figure 5-5. In the single output case, this requirement is fulfilled if $g_1$ and $g_2$ satisfy:

\begin{align}
  g_1 = 1 & \Rightarrow f = 1 \\  g_2 = 1 & \Rightarrow f = 0
\end{align} \quad (5.1)  

(5.2)

If either $g_1$ or $g_2$ is true, the exact value of $f$ can be determined from $x_{p+1} \ldots x_n$ so that the remaining inputs may be turned off. If both $g_1$ and $g_2$ are false, then all the inputs are needed to determine the outputs, so the circuit must be allowed to work normally. Therefore, the inputs can be disabled when $g = g_1 + g_2$ is true as is shown in Figure 5-5.

In the case of multiple outputs, Equations 5.1 and 5.2 can be generalized as:

\begin{align}
  g_{1,i} = 1 & \Rightarrow f_i = 1
\end{align} \quad (5.3)
Figure 5-5: Single output circuit with input disabling circuit.

\[ g_{2,i} = 1 \Rightarrow f_i = 0 \]  \hspace{1cm} (5.4)

for all \( i \) such that \( 1 \leq i \leq m \). For each output \( f_i \), the inputs may be turned off if \( g_{1,i} + g_{2,i} \) is true. To have all of the outputs evaluate properly, \( g_{1,i} + g_{2,i} \) must be true for all \( i, 1 \leq i \leq m \). In other words the inputs may be disabled if

\[ g = \prod_{i=1}^{m} (g_{1,i} + g_{2,i}) = 1 \]  \hspace{1cm} (5.5)

where the \( g_{1,i} \) and the \( g_{2,i} \) are defined in Equations 5.3 and 5.4. Therefore, the logic in block \( g \) of Figure 5-2 must satisfy Equation 5.5.

Given the architecture in Figure 5-2, there are two details that must be determined. First, a subset of the inputs must be selected to be turned off. Second, the exact logic for block \( g \) must be determined such that Equation 5.5 is satisfied. In particular, these details must be selected so that the power of block \( A \) is minimized while keeping the overhead of the added logic to a minimum. Chapter 6 presents algorithms for determining both of these given a particular circuit.
Chapter 6

Algorithms

As outlined in the previous section, algorithms are needed to accomplish two tasks given the architecture shown in Figure 5-2. First, an algorithm is needed to select the subset of inputs that will be disabled. Second, given this set of inputs, an algorithm must produce the logic for block \( g \). For each of these steps, the goal is to maximize the savings function:

\[
net\ savings = savings(A) - cost(B) - cost(g)
\]  

(6.1)

These algorithms are similar to the algorithms developed by Alidina in [2] which were described in Chapter 4 of this thesis. The thorough descriptions of these algorithms will not be repeated here. Instead, the algorithms will be explained briefly and any modifications from Alidina’s algorithm will be explained.

6.1 Synthesis of the Disable Logic

This algorithm determines the logic needed for block \( g \) assuming that \( S \) (the subset of inputs that will be disabled) has already been determined. This logic must be completely defined by \( x_{p+1} \ldots x_n \), it must maximize \( prob(g = 1) \), and it must satisfy Equation 5.5 so that the outputs are not affected.

This problem is identical to the problem encountered in Alidina’s technique. As
was shown in [2] and repeated in Section 4.2, the given constraints are satisfied using:

\[ g = \prod_{i=1}^{m} (U_s f_i + U_s \overline{f}_i) \]  

(6.2)

This results in the maximum power savings in block A given a particular set of disabled inputs, \( S \).

### 6.2 Reduction of Disable Logic Costs

Although the algorithm described in Section 6.1 results in the maximum power savings of the original subcircuit (\( \text{savings}(A) \)), it says nothing about the resulting cost of the disable logic (\( \text{cost}(g) \)). The original goal was to maximize the net savings given by Equation 6.1. To do so, the algorithm must consider reducing \( \text{prob}(g = 1) \) in order to reduce the cost of the disable logic. In particular, this algorithm will look for some function, \( g_{\text{reduced}} \), such that \( g_{\text{reduced}} \rightarrow g \) and such that Equation 6.1 is maximized.

This becomes a much simpler task by noting that the savings of block A is approximately proportional to \( \text{prob}(g_{\text{reduced}} = 1) \) and that the cost of block B is roughly constant with constant \( S \). Therefore, any component of the implementation of \( g \) that requires a significant amount of power but does not contribute significantly to \( \text{prob}(g = 1) \) should be eliminated.

This can be accomplished using the following algorithm. First, find the cube of \( g \) that contributes the least probability of making \( g \) true. If this cube is removed, the gross savings is reduced by \( (1 - \frac{\text{prob}(g_{\text{reduced}} = 1)}{\text{prob}(g = 1)}) \times \text{original savings} \). The cost is reduced by \( \text{cost}(g) - \text{cost}(g_{\text{reduced}}) \). If the cost is reduced more than the savings, then remove this cube from \( g \) and continue with the next cube. If the cost is not reduced more than the savings, then leave this cube in \( g \) and discontinue.

The details of this algorithm are shown in Figure 6-1. In this pseudo-code, \( \text{savings} \) refers to the \( \text{savings}(A) \), and \( \text{cost} \) refers to the \( \text{cost}(g) \).
\begin{tabular}{|l|}
\hline
\textbf{REDUCE\_G( } g, \textit{savings} \textbf{):} \\
\{ \\
\hspace{1em} \textit{orig\_cost} = \textbf{ESTIMATE\_COST}(g); \\
\hspace{1em} \textit{done} = \textbf{false}; \\
\hspace{1em} \textbf{while ( not done )} \{ \\
\hspace{2em} \text{select a cube, } \textit{cube}, \text{ from } g \text{ such that } \text{prob}(g - \textit{cube} = 1) \text{ is maximized;} \\
\hspace{2em} \textit{g}_\text{reduced} = g - \textit{cube}; \\
\hspace{2em} \textit{cost} = \textbf{ESTIMATE\_COST}(\textit{g}_\text{reduced}); \\
\hspace{2em} \textbf{if ( } \textit{orig\_cost} - \textit{cost} > (1 - \frac{\text{prob}(\textit{g}_\text{reduced}=1)}{\text{prob}(g=1)}) \times \textit{savings} \textbf{ )} \{ \\
\hspace{3em} \textit{g} = \textit{g}_\text{reduced}; \\
\hspace{3em} \textit{orig\_cost} = \textit{cost}; \\
\hspace{3em} \textit{savings} = \frac{\text{prob}(\textit{g}_\text{reduced}=1)}{\text{prob}(g=1)} \times \textit{savings}; \\
\hspace{2em} \} \\
\hspace{2em} \textbf{else} \\
\hspace{3em} \textit{done} = \textbf{true}; \\
\hspace{2em} \} \\
\hspace{1em} \textbf{return}(g); \\
\} \\
\hline
\end{tabular}

Figure 6-1: Procedure to reduce the cost of the disable logic
6.3 Selection of the Inputs

Given a particular combinational circuit, the set of inputs that will be turned off, $S$, must be selected. In particular, these inputs should be selected so that the cost function, Equation 6.1, is maximized.

In [2], Alidina develops an algorithm that performs a very similar task. This algorithm is described in detail in Section 4.3. There are several shortcomings of this procedure.

First, Alidina's algorithm is not fully automated. To use the algorithm, it must be run several times using different values of $k$. Although this is not a problem if only one circuit is being analyzed, it is a serious problem if many circuits are being analyzed within a loop. (This is, in fact, the case in Chapter 7.)

Second, Alidina's algorithm has a very limited cost function. Alidina simply maximizes $\text{prob}(g = 1)$. Although this is a very important part of maximizing the power savings, it is not a complete measurement of the power savings. Other important factors include the number of inputs that are disabled and the cost of the added logic. The true cost function of this technique is given in Equation 6.1. Although it is not possible to evaluate this cost function perfectly through many iterations, it does lead to a more accurate model.

To overcome the shortcomings of Alidina's algorithm, the new version of the algorithm uses a generalized cost function. As is shown in Figure 6-2, this generalization is implemented using the functions \texttt{ESTIMATE.SAVINGS} and \texttt{ESTIMATE.COST}. \texttt{ESTIMATE.SAVINGS} is a function that determines or estimates the gross savings that are achieved in block $A$ denoted $\text{savings}(A)$. This savings is assumed to be directly proportional to $\text{prob}(g = 1)$. This function may be implemented as a simple heuristic such as $|S| \times \text{prob}(g = 1)$, or it may be implemented as a function that does a complex analysis of the savings achieved within block $A$. For combinational circuits, it must consider timing relationships to be accurate. \texttt{ESTIMATE.COST} is a function that determines or estimates the costs due to blocks $B$ and $g$. Once again this function can be implemented using a simple or a complex...
**SELECT_INPUTS**(*f, k*):

```
{  
  BEST_SAVINGS = 0;
  SELECTED_SET = ∅;
  X = { x_i | x_i is an input of f };
  SELECT_RECUR(*f, f, b, X*);
  return(SELECTED_SET);
}
```

**SELECT_RECUR**(*g_1, g_2, S, Q*):

```
{  
  g = g_1 + g_2;
  pr = prob(g = 1);
  savings = ESTIMATE_SAVINGS(S, g, pr);
  cost = ESTIMATE_COST(g);
  max_savings = savings + REMAINING_SAVINGS(Q, pr);
  g_reduced = REDUCE_G(g, savings, cost);
  if(max_savings < BEST_SAVINGS) return;
  else if(savings - cost > BEST_SAVINGS) {
    BEST_SAVINGS = savings - cost;
    SELECTED_SET = S;
  }
  choose x_i ∈ Q such that i is minimum;
  SELECT_RECUR(U_{x_i}g_1, U_{x_i}g_2, S \cup x_i, Q - x_i);
  SELECT_RECUR(g_1, g_2, S, Q - x_i);
  return;
}
```

Figure 6-2: Procedure to Determine the Optimal Set of Inputs
heuristic.

Using these generalized functions gives the algorithm more power and more flexibility. Because the functions are a better representation of the actual cost function, it is simple to make the algorithm fully automated. The algorithm does not need to be run for many values of $k$ because the algorithm knows which solution is the best. The generalized functions also make the algorithm more efficient. When the old algorithm is run for many values of $k$, it is branching over the same binary tree several times. Because the new algorithm understands the real costs better, it only needs to branch over the tree once. Finally, because the cost function is more accurate, the results are more accurate. Therefore, using these generalized functions achieves both better results and less computation time.

Even so, there is one shortcoming. Because the cost functions are more complex, it is more difficult to prune the branching. Actually, Equation 4.7 still holds, but the actual information needed is how Equation 6.1 is affected as more inputs are added to $D$. To overcome this, the $\text{MAX\_SAVINGS}$ function is used. $\text{MAX\_SAVINGS}$ returns the maximum savings that can be achieved if all the inputs are disabled. This can be determined because the $\text{prob}(g = 1)$ is bounded according to Equation 4.7. Having a bound on the maximum savings means that branching can be discontinued if the maximum savings drops below the best savings achieved so far.

The resulting algorithm is shown in Figure 6-2. Except for the improvements described above, it is basically the same algorithm presented by Alidina in [2]. This algorithm results in the best possible set of inputs to turn off assuming that $\text{ESTIMATE\_SAVINGS}$, $\text{ESTIMATE\_COST}$, and $\text{REMAINING\_SAVINGS}$ are reasonably accurate.
Chapter 7

Subcircuit Selection

In Chapters 5 and 6, a methodology has been described in which the power consumption of combinational circuits can be reduced by dynamically turning off select inputs. This technique is closely based on the technique developed by Alidina in [2] and described in Chapter 4 of this thesis. Although good results can be achieved with this algorithm, there are limitations. One such limitation is the severe restrictions that occur as the number of outputs increase. This chapter describes this limitation and suggests a technique to overcome it.

7.1 Motivation

As was shown in Section 5.1, Equations 5.3 and 5.4 must be satisfied for each output. As the number of outputs increase, this restriction becomes even tighter. In general, this tends to reduce the probability that inputs will be turned off (i.e., decreases $\text{prob}(g = 1)$). This, in turn, reduces the power savings that can occur.

For example, consider the two circuits shown in Figure 7-1. The circuit in part (a) is simply an 8-bit comparator. Because there is only one output, it is relatively easy to find a disable function, $g$, that satisfies the restrictions stated in Section 5.1 and still has a good probability of being true. For example, letting $D = \{a[7], b[7]\}$ results in $g = U_S f = a[7] \oplus b[7]$ with $\text{prob}(g = 1) = 0.5$. The circuit in part (b) is the same 8-bit comparator, except that now the inputs are being fed from adders.
Intuitively, it seems that the same savings should be achieved because both circuits contain the same combinational comparator. But because of the additional outputs of the adders, none of the inputs can be turned off. The tool is simply not intelligent enough to consider turning off interior nodes.

Alidina suggests one solution to this problem in [2]. He considers deriving $g$ using a subset of the outputs. Then, to keep the function of the rest of the outputs correct, he suggests duplicating the logic that is shared by both sets of outputs. This is described in more detail in Section 4.4. For example, for the circuit in Figure 7-1(b), this algorithm may result in both adders being duplicated. As is shown by this example, logic duplication can result in a tremendous amount of over-head that limits power reduction.

As an alternative solution, this chapter presents a method based on division into subcircuits. The best solution to the example in Figure 7-1 would be to run the algorithm on just the comparator subcircuit. To give the algorithm this flexibility, the circuit is first divided into subcircuits. Each subcircuit is considered for power reductions. Then, these subcircuits are recombined into groups of subcircuits and again the power reduction technique is considered. This is repeated until, eventually,
the entire circuit may be considered.

The rest of this chapter considers this technique in greater detail. First some observations about the nature of this technique are given in Section 7.2. Based on these observations, two algorithms are presented in Sections 7.3 and 7.4.

### 7.2 Observations About Single-Output Subcircuits

In order to develop algorithms that can efficiently search a circuit for the optimal subcircuits, a few observations are needed.

First, a couple of terms must be defined. A *single-output subcircuit* is a combinational subcircuit that has only one output. A *maximum-sized, single-output subcircuit* (MSSO subcircuit) is a single-output subcircuit to which no gate or set of gates may be added such that the subcircuit still has only one output.

Next, note that no two MSSO subcircuits can overlap. To show this, assume two single-output subcircuits do overlap such that one output gate is within the other subcircuit. In this case, the two subcircuits can be joined to make a larger single-output subcircuit implying that the two subcircuits were not maximum-sized. Consider another case in which two subcircuits overlap, but their output gates are separate. In this case there exists some gate whose output drives gates in both subcircuits. This gate output is also a subcircuit output so the original two subcircuits have more than one output. Because of these two cases, MSSO subcircuits cannot overlap.

Next, note that every gate in a circuit is included in at least one MSSO subcircuit simply because gates have only one output. In addition, because MSSO subcircuits cannot overlap, no gate can be included in two MSSO subcircuits. These two facts demonstrate that the complete set of MSSO subcircuits is a unique division of the circuit into non-overlapping subcircuits. The non-overlapping property also forces this unique set to be the minimum number of single-output subcircuits that completely define the circuit.

This minimum number of single-output subcircuits can be determined in linear
time. To do so, simply walk through all of the gates in the circuit starting at the outputs and working backwards to the inputs. If a gate has a primary circuit output, then it is the beginning of a new MSSO subcircuit. If a gate has fanouts that are part of different MSSO subcircuits, then this gate is the beginning of a new MSSO subcircuit. Otherwise, all of this gate's fanouts belong to the same MSSO subcircuit, and therefore this gate also belongs to this MSSO subcircuit. Pseudo-code for this algorithm is shown in Figure 7-2.

To see how this algorithm works, consider the contrived circuit shown in Figure 7-3. In this circuit, the algorithm would start at the three outputs of the circuit. Each of these gates represent the start of new MSSO subcircuits, $A$, $D$, and $B$. The algorithm continues with the fan-ins of each of these three gates. The next level of gates is labeled according to the rules explained above. This continues until the inputs are reached. When the algorithm is complete, the circuit has been divided into MSSO subcircuits as is shown in part (b) of Figure 7-3.

Next, note that there is no need to analyze any subcircuit that is composed of

---

**Figure 7-2: Procedure to Find the Minimum Set of Single-Output Subcircuits**

```c
GET_SINGLE_OUTPUT_SUBCIRCUITS(circuit):
{
    arrange nodes of circuit in depth order outputs to inputs;
    foreach node in depth order (node) {
        if (node is a primary output) {
            subcircuit = create_new_subcircuit();
            mark node as part of subcircuit;
        }
        else {
            check every fanout of node;
            if (all fanouts are part of the same subcircuit)
                subcircuit = subcircuit of the fanouts;
            else
                subcircuit = create_new_subcircuit();
            mark node as part of subcircuit;
        }
    }
    mark node as part of subcircuit;
}
```
Figure 7-3: Dividing a circuit into a minimum number of single-output subcircuits. (a) The original circuit. (b) The single-output subcircuits.

only a part of one of these MSSO subcircuits. Consider a single-output subcircuit that is composed of a subset of the gates from a MSSO subcircuit including the output gate. Because the outputs of these two circuits are the same, the restrictions on \( g \) are identical. But the MSSO subcircuit has more internal nodes implying that more power savings can be had. Therefore it makes sense to consider only the MSSO subcircuit.

Now, consider a single-output subcircuit that is composed of a subset of the gates from a MSSO subcircuit not including the output gate. This subcircuit has a different output that indirectly feeds into the original output gate. Because this output feeds the original, it must be more restrictive than the first. That is, when the universal quantification is evaluated it must be true that \( g_{\text{new}} \Rightarrow g_{\text{original}} \). Therefore, \( \text{prob}(g_{\text{new}} = 1) \leq \text{prob}(g_{\text{original}} = 1) \). Once again it makes sense to consider only the MSSO subcircuit.

The conclusion of this argument is that there is no need to consider any subcircuit that is a subset of a MSSO subcircuit. There is one notable exception to this rule. In some cases, it may be desirable to consider turning off nodes that are in the interior of a MSSO subcircuit to overcome timing restrictions. This will be discussed in more detail in Chapter 8. Despite these exceptions due to timing, using MSSO subcircuits is an excellent method to reduce the possible subcircuits to a manageable number.
The algorithms that are developed in the following two sections are based on this idea.

### 7.3 The First Algorithm

The previous section showed how to divide a combinational circuit into MSSO subcircuits in order to narrow down the possible subcircuit possibilities. Using this idea, one possible approach could be: 1) Create the set of MSSO subcircuits, 2) Try every possible combination of these subcircuits, and 3) determine the combinations that yield the best net savings. Unfortunately, step 2 above cannot be executed for any significant number of MSSO subcircuits (certainly no more than ten). Therefore a more intelligent algorithm is needed to reduce the number of possibilities.

First note that not all possible combinations of subcircuits make sense to evaluate. If a pair of subcircuits are completely unrelated, in other words they have no outputs or inputs in common, then there is obviously no reason to evaluate them as a pair. A better result can be obtained by evaluating the subcircuits separately.

Therefore, an algorithm is used that loops through only combinations of MSSO subcircuits that are interconnected. The algorithm branches over several trees – one starting with each MSSO subcircuit. Figure 7-4 shows the branching pattern for the example circuit of Figure 7-3. In this figure, the lower-case letters represent the set of neighbors for the current combination. Two MSSO subcircuits are neighbors if at least one input or one output of one subcircuit is also an input or an output of the other subcircuit. Starting with one MSSO subcircuit, the algorithm selects a second MSSO subcircuit from the first’s neighbors. The set of neighbors is updated to include the neighbors of the second MSSO subcircuit. This continues as the algorithm branches over all possible combinations of MSSO subcircuits.

Of course the algorithm should not evaluate the same combination of MSSO subcircuits multiple times. For example, the algorithm should not evaluate ABCD, ACBD, and BCDA because these are actually the same combination that were constructed in a different order. For most combinations, the branching will occur only in
the order of the MSSO subcircuit names. For example, if the branching is currently at AC, the algorithm will branch to ACD, but will not branch to ACB because this is out of order.

This strategy will eliminate all duplications, but it may also skip some valid combinations. In the example, ABC cannot be created in order because subcircuits A and B are not neighbors. To account for these cases, the following special rule is used. Assume that, at a certain node along the branching tree, a new MSSO subcircuit is added to the list of neighbors. Further assume that the index of this new neighbor is greater than the index of the MSSO subcircuit that is at the root of the tree. In this case, an extra branch should be made to include this new neighbor. For the example shown in Figure 7-4, this rule requires that the node AC branch to the node ABC. But this rule does not allow node BC to branch to ABC because index(A) < index(B).

The algorithm presented so far is better than trying all possible combinations, but it is still too complex to run on many circuits. To reduce complexity, a pruning condition has been introduced. First, note that as MSSO subcircuits are added, more outputs of the circuit make Equations 5.3 and 5.4 more restrictive. Therefore the quantity \( \text{prob}(g = 1) \) will tend to decrease. Assuming that this must be true, the algorithm stops branching when the value of the \( \text{prob}(g = 1) \) drops below a threshold. This threshold is just an arbitrary number based what the designer considers to be significant savings. 10% may be a reasonable threshold. The complete algorithm
using these rules is shown in Figure 7-5.

Although there are contrived cases when this algorithm will fail to find the optimal division of subcircuits, it finds very good divisions for most circuits.

7.4 The Second Algorithm

For some circuits, the algorithm presented in Section 7.3 may still involve too much computation. A few more observations leads to another heuristic algorithm.

As was shown earlier, the number and the function of subcircuit outputs are the key factors in determining the probability $\text{prob}(g = 1)$. Each output acts like a restriction on the set of input vectors for which inputs may be turned off. In general, it is desirable to maximize the size of the subcircuits and still keep the number of subcircuit outputs low in order to achieve high savings. Therefore, the subcircuit selection algorithm should be written to maximize the ratio between internal nodes and outputs.

Consider joining two MSSO subcircuits to create a new subcircuit. If the MSSO subcircuits share inputs, as in Figure 7-6(a), then the ratio of internal nodes to outputs is not increasing. Another way of looking at this is that the outputs of the MSSO subcircuits are still outputs of the combined subcircuit, and, therefore, the restrictions on $g$ remain the same. Because of this, it is unlikely that this combination will lead to increased power savings. The same argument can be made for the case shown in Figure 7-6(b). Because the internal node to output ratio is not increasing, it is unlikely that there will be any substantial increase in the power savings.

Now consider Figure 7-6(c). In this case, three MSSO subcircuits have been grouped so that one output is no longer an output. This is like removing one restriction on the disable logic, $g$. In this configuration, the internal node to output ratio has been increased over any individual MSSO subcircuit. Therefore, it is reasonable that the gross savings may actually increase compared to the sum of the three individual MSSO subcircuits. At the same time, the disable logic will be shared reducing overhead. Obviously this type of situation is much more likely to produce good results.
SUBCIRCUIT_SELECT(circuit):
{
    let $A$ be the array of single-output subcircuits so that each subcircuit is denoted $A[i]$, $0 \leq i < |A| - 1$
    $i = 0$;
    while ($i < |A|$) {
        $N = \text{NEIGHBORS}(A[i])$;
        SUB_SELECT_RECUR( $\{A[i]\}$, $N$, $N$, $i$, $i$);
        $i = i + 1$;
    }
}

SUB_SELECT_RECUR( $B$, $N$, $M$, $f$, $l$):

$B$ = set of single-output subcircuits that comprise the current multi-output subcircuit
$N$ = set of neighboring single-output subcircuits
$M$ = set of new neighboring single-output subcircuits
$f$, $l$ = smallest and largest subcircuit indices found in $B$
{
    EVALUATE_SUBCIRCUIT( $B$, $\text{prob}$ );
    if ( $\text{prob} < \epsilon$ )
        return;
    $i = f$;
    while ($i < l$) {
        if ( $A[i] \subset M$ ) {
            $X = \text{NEIGHBORS}(A[i])$;
            SUB_SELECT_RECUR( $B \cup \{A[i]\}$, $N \cup X$, $X - N$, $f$, $l$ );
        }
        $i = i + 1$;
    }
    while ($i < |A|$ ) {
        if( $A[i] \subset N$ ) { 
            $X = \text{NEIGHBORS}(A[i])$;
            SUB_SELECT_RECUR( $B \cup \{A[i]\}$, $N \cup X$, $X - N$, $f$, $i$ );
        }
        $i = i + 1$;
    }
}

Figure 7-5: Algorithm that evaluate different combinations of subcircuits.
Figure 7-6: Possible groupings of adjacent MSSO subcircuits. (a) Subcircuits sharing an input. (b) A subcircuit output feeding a subcircuit input. (c) A subcircuit output and each of its fanouts.

These observations lead to another heuristic algorithm. First, find all the MSSO subcircuit outputs that are not primary circuit outputs. Then, for each of these nodes, group the adjoining MSSO subcircuits into a multiple-output subcircuit and evaluate the power savings that can be achieved. Continue by trying combinations of these nodes. This leads to a branching structure that is identical to the one described in the previous section, except that each node in the tree now represents a circuit node. In fact, the same pruning condition still holds: if $\text{prob}(g = 1)$ decreases below a threshold, the branching should discontinue. But, in this case, there are far fewer possibilities to try so that the algorithm is much more efficient.

Once again, there are contrived cases when this algorithm will fail to find the optimal division of subcircuits, but in general it is a very good heuristic.
Chapter 8

Results, Comparisons, and Future Work

The algorithms described in Chapters 5, 6, and 7 have been implemented and executed on example circuits. This chapter describes the experimental results that have been achieved. In addition, the results are compared to the results of the Alidina technique, and suggestions for future work are given.

8.1 Results

The algorithms described in this thesis have been implemented in C and have been incorporated into the SIS logic synthesis and optimization platform. The power reduction technique has been used on many example circuits. Some of the results are shown in Table 8.1.

Although some good results have been achieved, the majority of circuits evaluated produced little or no power savings. There are two major reasons for this.

First, this technique relies on circuits that have certain functional properties. To achieve savings, a circuit or a subcircuit must have inputs whose values are sometimes unnecessary to determine the outputs. There exist circuits where this simply is not true. For example, an adder needs all of its input information all of the time to determine the correct output values. These cases are just a shortcoming of the whole
<table>
<thead>
<tr>
<th>CKT</th>
<th>Original</th>
<th>Precompute Logic</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pwr</td>
<td>Bits</td>
<td>Lits</td>
</tr>
<tr>
<td>comp16</td>
<td>92 16</td>
<td>364.1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>comp8</td>
<td>44 8</td>
<td>169.1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>fcomp8</td>
<td>88 8</td>
<td>338.2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>fcomp8</td>
<td>88 8</td>
<td>338.2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>priority15</td>
<td>60 4</td>
<td>150.8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>priority7</td>
<td>17 4</td>
<td>79.7</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 8.1: Power reductions of combinational circuits.

The other major problem encountered with combinational circuits is timing. For this technique to be successful, the disable signal must arrive before the signals that are being turned off. This requires two things. All the inputs in $D$ must arrive before all the inputs in $S$, and the disable logic must evaluate very quickly. For random circuits, these conditions are unlikely, resulting in insignificant savings for random circuits.

Because of these two problems, positive results can be obtained for only a select group of circuits.

### 8.2 Comparison to Alidina’s Technique

Comparing the results of Alidina in section 4.5 versus the results of this new technique in the previous section reveals Alidina’s technique to be more powerful. Alidina’s technique seems to find better power savings for a larger range of circuits.

The main advantage of Alidina’s technique is its independence from timing constraints. As described in Section 4.1, all of the additional logic is added to the previous clock cycle, avoiding any possible timing problems. In the technique described in this
thesis, the target circuits are combinational so that adding logic to the previous cycle is not possible. The timing constraints were very restrictive so that very few circuits achieved good results.

Another advantage of Alidina's technique is lower overhead. Because Alidina is disabling inputs coming out of a register, no additional logic is required to store the value at the disabled node. In the technique described in this thesis, a general circuit requires the addition of an entire latch. The additional overhead reduces the net savings considerably.

Even so, there are still a class of circuits where this new technique can find savings whereas Alidina's cannot. This new technique is generally better at finding savings within circuits with a large number of outputs because of the strategies described in Chapter 7. This is best demonstrated by the example circuit shown in Figure 7-1(b). For this circuit, Alidina's technique requires logic duplication resulting in overhead that would overwhelm the possible savings. This new technique is capable of finding and considering the comparator as a separate subcircuit.

All things considered, Alidina's technique is more powerful. But the technique presented in this thesis is still valuable because of circuits with a large number of outputs for which Alidina's technique fails.

8.3 Future Work

The major difficulty of the work presented in this thesis is timing. To successfully reduce the power consumption of static combinational circuits, the disable logic must arrive before the nodes that are being disabled. A late disable signal will not prevent the signals from switching and will simply be wasted overhead. There are several possible ways of overcoming this difficulty.

The most obvious way to do this is to speed up the arrival of the disable signal. This may be accomplished by computing the disable signal from nodes that evaluate earlier. The theory to do this does not yet exist, and it may be a very difficult problem.
Another possibility is to disable signals that arrive later. For example, when using such a technique on a comparator, it may be better to disable the carry signals instead of the circuit inputs. Of course, this may also significantly reduce power savings.

The most promising way to overcome timing constraints is to implement \( g \), the disable logic, using Domino logic. Using Domino logic, \( g \) would always be pre-charged to a 1. This disables the inputs at the beginning of the clock cycle. Then, if \( g \) evaluates to a 0, the signals would be allowed to propagate normally. But, because \( g \) always starts the clock cycle as a 1, there is no way the disable signals could propagate before \( g \) evaluates. Therefore, timing is no longer an issue. Effectively, this is a self-timing strategy to allow switching to propagate only after it has been determined that the signals need to propagate.

There are difficulties with using this kind of mixed logic circuit. Domino logic must be implemented within some kind of clocking methodology, so this could not be used in any asynchronous circuits. Domino circuits require a certain amount of overhead including the pre-charge and evaluate signals. And the inputs to the Domino circuits cannot be purely combinational signals because Domino signals are not allowed to make 0 → 1 transitions except during the pre-charge. In addition, requiring signals to wait before they propagate can increase the delay of the circuit considerably. Although there are quite a few difficulties with the mixed logic technique, it shows a great deal of promise for overcoming timing constraints.

In any case, any serious future work must address the timing problems of this technique.
Chapter 9

Conclusion

This thesis presented a new technique for power optimization of combinational CMOS circuits. The technique adds logic to dynamically "turn off" a subset of inputs. This decreases switching activity within the circuit reducing the required power.

To implement this technique, a standard architecture was presented along with algorithms that optimize the power reduction for a particular circuit. Algorithms used to select inputs and synthesize additional logic were updated from Alidina's work on sequential circuits. In addition, new algorithms were developed that divide the circuit into subcircuits in an effort to make the technique more versatile. These algorithms were implemented, and the result was a completely automated CAD tool.

Experimental results show that the technique was not as successful as desired. For some circuits, it is not possible to disable inputs because all of the input information is needed all of the time. This problem is shared by Alidina's technique. For other circuits, timing prevented the technique from reducing the power consumption. To achieve savings, it is necessary that the disable signal arrive before the signals that are being turned off. This is a severe restriction that makes it unlikely that power savings will be found for random circuits. Alidina's technique avoids all timing problems by evaluating the disable logic in the previous clock cycle. Even so, there is a class of circuits for which this new technique outperforms Alidina's technique. If circuits have a large number of outputs, Alidina's technique often fails to find savings. But this new technique will divide the circuit into subcircuits and can find reasonable savings.
Overall, the power reduction CAD tool presented in this thesis has some shortcomings. But, for a class of circuits, it outperforms previous work in this area, and therefore it is still a valuable CAD tool.
Bibliography


