NORTHWESTERN UNIVERSITY

Static NBTI Reduction Using Internal Node Control

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Electrical and Computer Engineering

By

David R. Bild

EVANSTON, ILLINOIS

December 2008

ABSTRACT

Static NBTI Reduction Using Internal Node Control

David R. Bild

Negative Bias Temperature Instability (NBTI) is a significant reliability concern for nanoscale CMOS circuits. Its effects on circuit timing can be especially pronounced for circuits with standby-mode equipped functional units, because these units can be subjected to static NBTI stress for extended periods of time. This thesis proposes internal node control, in which the inputs to individual gates are directly manipulated to prevent this static NBTI fatigue. We give a mixed integer linear program formulation for the optimal solution of this problem. The optimal placement of internal node control yields an average 26.7% reduction in NBTI-induced delay over a ten year period for the ISCAS85 benchmarks when compared to a complementary technique, input vector control. The combined application of the two techniques leads to an average 51.3% reduction in NBTI-induced delay. Using the MILP formulation, we show that the use of multiple input vectors, which would be cycled through during idle periods in order to spread the NBTI degradation among more gates, is not beneficial for these circuits. We find that the problem is $\mathcal{NP}$-complete and present a linear-time heuristic which can be used to quickly find

near-optimal solutions. The heuristic solutions are, on average, within 0.17% of optimal and all were within 0.60% of optimal. Finally, because the benchmark circuits have small gate counts relative to many modern circuits, we apply our heuristic to industrial-scale circuits and present the results.

Acknowledgements

I would like to thank my advisor, Professor Robert P. Dick, for his advice and assistance on this work. Not only have his suggestions and support have been invaluable, but his willingness to expend time and effort to help his students is unparalleled.

Much of the initial work presented in this thesis was conducted in close collaboration with Greg Bok. Greg, thanks for the many stimulating discussions, both on NBTI and other various topics, and of course, the numerous late hours in the T-Lab coding and scripting. Thanks for introducing me to Python and making tasks like tackling the "Falcon Framework" not just bearable, but somehow also entertaining.

To my "climbing" friends, "school" friends, and "home" friends, thanks for all the good times and wonderful memories. Your occasional reminders to put down the books, leave the lab, and get out to enjoy life were much appreciated. It is a shame that in today's society, as quickly as we all converged in Evanston several years ago, we have now scattered to the so-called corners of the earth.

To my family, thanks for all your support over the past twenty-three years. Mom and Dad, words really cannot express how grateful I am for everything you have done for me. Grandma, Gram, and Pop, I consider myself incredibly fortunate to have had the opportuntiy to spend so much time as a child with my grandparents. I've learned so much from you.

To the EECS faculty at Northwestern University, thanks for your instruction in the numerous courses I have taken over the past five years. Professor Russ Joseph and Professor Seda Ogrenci Memik, thank you not only for your valuable instruction in the courses I had with each of you, but for serving on my thesis committee as well.

Finally, I would be remiss if I failed to acknowledge Northwestern University and the Robert R. McCormick School of Engineering and Applied Science for the Cabell Fellowship that funded my first year of graduate studies. Credit is also due to the National Science Foundation and the Semiconductor Research Consortium, both of which have provided funding for the projects on which I have worked.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1. Overview of NBTI

Due to the scaling trends of CMOS technology, Negative Bias Temperature Instability (NBTI) is emerging as a significant reliability concern for digital circuits. NBTI, which in current technologies only significantly affects PMOS transistors stressed with a negative bias (V_{gs} = -V_{dd}), manifests itself as an increase in threshold voltage that reduces switching speed [1].

At the atomic level, NBTI is caused by an electric field dependent disassociation of Si-H bonds at the Si/SiO_2 interface. The hydrogen diffuses into the gate oxide in a temperature-dependent reaction, leading to the formation of interface traps, which are responsible for an increase in threshold voltage. These mechanisms lead to an interesting recovery effect; when the stress is removed ($V_{gs} = V_{dd}$), the reaction reverses, with some of the hydrogen diffusing back towards the interface and bonding with the Si [1].

Under constant stress, static NBTI effects quickly lead to performance degradation. However, thanks to the previously described recovery effect, for circuits experiencing typical switching activity, the negative impacts of dynamic NBTI degradation take longer to accumulate. For a 70 nm Berkeley Predictive Technology Model, Paul et al. predict an $\sim$10% increase in delay after 10 years of operation for the ISCAS85 benchmarks [2,3].

During normal circuit operation, standard switching activity causes alternating stress on the PMOS transistors and thus degradation is dominated by dynamic NBTI. However, many designs employ sleep or clock-gating techniques in order to reduce dynamic power consumption. In such schemes, idle functional units are put in standby or sleep mode by having their inputs frozen or their clock transitions gated. This prevents extraneous transitions, reducing dynamic power consumption. However, with the inputs stable for long periods of time, PMOS transistors with low inputs may degrade due to static NBTI effects. In this scenario, static NBTI optimization is particularly relevant.

1.2. Thesis Contribution

In this thesis, we propose and evaluate an internal node control technique to limit the effect of this static NBTI stress. Internal node controls can be inserted at the output of individual gates in order to force their outputs to specific values during standby. Using this technique, static NBTI stress for a PMOS transistor can be eliminated, for example, by forcing the output of the preceding gate to V_{dd}. However, internal node control imposes a timing penalty; the additional circuitry required for node control introduces a small delay. NBTI degradation on a timing-sensitive (i.e., critical path) transistor can be eliminated by forcing non-critical path gates to circuit structure dependent values, such that a low value is propagated to the NBTI-sensitive transistor.

We show that determining the optimal set of insertion points leading to the minimal degradation in circuit delay after some elapsed time is $\mathcal{NP}$-complete and formulate it as a mixed integer linear program. For the ISCAS85 benchmarks, we find that the optimal application of internal node control leads to an average 26.7% reduction in NBTI-induced delay over the solitary use of input vector control, a complementary technique. The combined use of internal node control and input vector control leads to an average 51.3% decrease in NBTI-induced delay. These techniques could be extended by the use of multiple input vectors, which would be cycled through to spread out the degradation over more gates. We find, however, that for our benchmarks the addition of multiple vectors does not improve the critical path delay. Due to the time complexity of the optimal formulation, we present a linear-time heuristic to find good solutions in a reasonable amount of time. The heuristic is within 0.17% of optimal on average and is always within 0.60% of optimal. The INC placements require only a 1.6% increase in area.

1.3. Thesis Organization

In chapter 2, we discuss both models for NBTI degradation and related techniques for the mitigation and prevention of static NBTI effects. In chapter 3, we present our model of internal node control for static NBTI control and prove that the problem is $\mathcal{NP}$-complete. In chapter 4 we describe the mixed integer linear programming formulation for its optimal solution and present the experimental results of our technique on the ISCAS85 benchmarks. In chapter 5, we present our linear-time heuristic and show the experimental results. Finally, in chapter 6, we describe the results of INC when applied to large, industrial-sized circuits.

CHAPTER 2

Negative Bias Temperature Instability

2.1. NBTI Models

Negative Bias Temperature Instability (NBTI) is a degradation mechanism affecting PMOS transistors, which results in increased threshold voltage and thus slower switching speeds. The increase in threshold voltage is generally thought to be caused by the generation of interface traps and oxide charge in PMOS transistors under negative bias (V_{gs} = -V_{dd}). These interface traps, dangling bonds, and oxide charges are attributed to an electric field dependant disassociation of Si-H bonds at the Si/SiO_2 interface. In the reaction-diffusion model, the currently accepted model for this mechanism, the interface trap generation (reaction) results in free hydrogen that diffuses into the oxide (diffusion) [4]. Both the reaction and diffusion regimes limit the rate of the degradation. The specific diffusion species (H, H_2, or a combination of the two) is not currently known, meaning that there is still debate about the accuracy of the various diffusion models. When the stress is removed, the hydrogen diffuses back towards the interface and can re-bond with the Si. This reversal of the mechanism is known as the recovery effect. This recovery effect complicates attempts to measure NBTI degradation because some of the degradation recovers between the time that the stress is removed and the time that measurements are taken. Several recent review papers provide detailed explanations and discussions of these models [1, 4, 5, 6, 7].

The NBTI models mentioned in the preceeding papers are useful for understanding the physical mechanisms that lead to the degradation, but are too detailed for use in circuit-level analysis. Cycle-based, transistor-level simulation is simply too time-consuming for use in reliability analysis tools or reliability-aware CAD algorithms. Consequently, several researchers have developed analytical models for predicting NBTI degradation [2, 8, 9, 10, 11, 12, 13]. As mentioned in the previous paragraph, when the stress on a transistor is removed, the reaction reverses and thus some of the degradation is also reversed. Because this recovery effect is so pronounced for gates experiencing rapid switching, most analytical models differentiate between static and dynamic stress. Static stress, as might occur in an idle functional unit and with which we are concerned with in this thesis, occurs when the transistor is stressed continously for a long period of time. Dynamic stress occurs when the stress is repeatedly and alternately applied, as would be common in an operating functional unit.

Much of the analytical modeling work has focused on dynamic stress. While models for static stress do exist, due to the difficulties in obtaining experimental data for extended-period static stress, they are not accurate for long time periods. The existing models severely over-estimate the stress for periods of 5–10 years. Consequently, in this thesis we do not explicity model the static stress as a function of time, but instead assume that the degradation over 10 years will result in an approximate 10% increase in delay, as suggested by Paul et. al [2]. This simplification is appropriate for our work because we are concerned with the aggregate long-term effects of constant NBTI stress.

2.2. Managing NBTI Degradation

Several techniques have been proposed for dealing with the impacts of NBTI. These methods generally fall into two groups: those that simply compensate for the NBTI-induced timing degradation and those that actively attempt to decrease and mitigate the degradation. In the following sections, we summarize and discuss existing work of both types.

2.2.1. Compensating Techniques

Several techniques have been proposed for dealing with the impacts of NBTI. One class of methods, which includes guard banding, gate sizing, V_{dd} tuning, and V_{th} tuning, has been used in industry to compensate for timing degradation. Such techniques compensate for the effects of NBTI at the expense of timing, area, or power because they do not attempt to minimize the NBTI-induced degradation.

In guard banding, the maximum clock frequency of a circuit is artificially limited, often by as much as 10%, to compensate for possible future NBTI-induced delay [14]. Sacrificing a significant percentage of the initially-available performance ensures that the processor will not fail due to NBTI degradation. In gate sizing, the sizes of the transistors are increased, thus increasing the initial speed of the circuit, so that the NBTI-degraded circuit still meets the timing requirements. However, this technique imposes an 8%–12% area overhead and increases power consumption [9]. Similarly, in V_{dd} and V_{th} tuning, the voltage of the circuit is adjusted to increase the initial operating speed [9]. The problems with this technique are two-fold. First, increasing the operating voltage increases the rate of NBTI degradation, requiring a further increase of V_{dd}. Second, increasing operating voltage increases the power consumption of the circuit. Techniques that minimize the NBTI degradation are needed.

2.2.2. Mitigating Techniques

Power gating and clock gating methods have been used to reduce the power consumption of idle functional units [15]. In power gating, a *sleep transistor*, which can be turned off to prevent any static or dynamic power consumption, is added between the power supply and the functional unit. In clock gating, the clock input to the idle functional unit is disabled to prevent dynamic power consumption. This is usually combined with Input Vector Control (IVC) to reduce the leakage power consumption. Leakage power consumption is dependent on the state of the inputs to a gate, and thus in IVC, the functional unit inputs are chosen to minimize the total leakage.

Both techniques could be used for NBTI degradation reduction. Power-gated transistors do not experience NBTI degradation. However, the wake-up time for a power-gated functional unit is orders of magnitude longer than for a clock-gated, input vector-controlled unit [15]. Clock gating and IVC methods allow for more temporally fine-grained control.

Wang et al. investigated the use of IVC to reduce NBTI degradation [8]. In practice, this control can be implemented either by placing MUXes on the inputs or by using a scan-chain. Unfortunately, for many circuits, the input vector may only be able to control a few levels of the circuit's internal gates. Consequently, they observed only an average 3% improvement in delay for the ISCAS85 benchmarks. They predict that for future technologies, smaller gate sizings and higher temperatures may increase the benefit of this technique.

In contrast with IVC, internal node control (our proposed technique) permits much greater control of all levels of the circuit, thereby allowing greater reduction in the NBTI-induced delay.

CHAPTER 3

Internal Node Control

Internal node control (INC) refers to setting the states of individual nodes or gate outputs at any layer of the circuit to specific values. With this extension to IVC, further control and thus NBTI mitigation is possible. INC can be implemented by the addition of node control circuitry at the output of each controlled gate.

There are several important observations about INC insertion for NBTI minimization in CMOS. We first describe a specific implementation of INC originally developed for static power consumption minimization. We then discuss the difficulty of removing NBTI stress from all PMOS transistors in a circuit and note a property of NOR gates that lessens the associated cost. Next, we explain the structural properties of transistors requiring NBTI stress removal and give our problem definition. We show that the problem is $\mathcal{NP}$-complete via a reduction from circuit-SAT.

3.1. INC Implementation

In order to force a node to a specific value, we borrow a technique from work by Abdollahi, Fallah, and Pedram on leakage minimization [16]. In this technique, a gate can be modified to allow its output to be forced either high or low, although not both. To force the output high, the output of the gate is connected to V_{dd} via a PMOS transistor in parallel with the existing pull-up network. This is controlled by an active-low sleep signal that pulls the output high when enabled. In order to prevent a short through the gate,

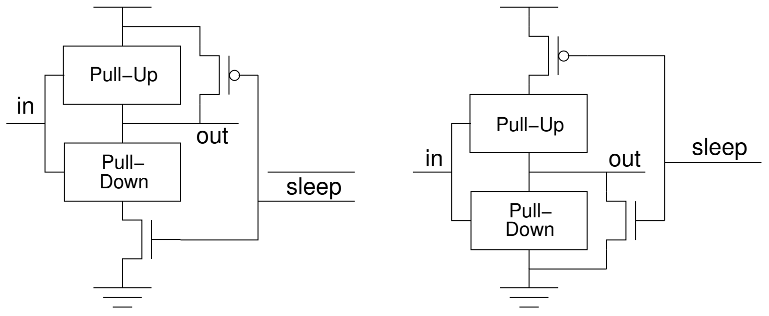

Figure 3.1. CMOS gates modified to include node control [16].

the pull-down network is then placed in series with an NMOS transistor. This transistor is responsible for the majority of the increase in gate delay. To force an output low, a similar modification is made. This is illustrated in Figure 3.1.

Unfortunately, the addition of this extra circuitry required for INC increases circuit delay. For a 65 nm Berkeley Predictive Technology Model, [17, 18], this technique results in an ∼12.5% increase in delay for a simple inverter. The absolute delay appears to be independent of gate type, so the percentage decreases for larger gates.

3.2. Potential of INC for CMOS

For an inverting logic implementation technology such as CMOS, if all of the inputs to a gate are high, then the output will be low. Thus, it seems that in order to place non-stressing (high) values on all PMOS inputs, internal node control must be implemented at the output of *every* gate. Recall, however, that NBTI stress is due to negative bias between the source node and the transistor input (V_{gs} = -V_{dd}); it is not just due to the low input value. For gates with parallel pull-up networks (e.g., inverters and NAND gates), the source node for each PMOS transistor is always at V_{dd} and each transistor is stressed whenever the input is low. For gates with series pull-up networks (e.g., NOR gates), the

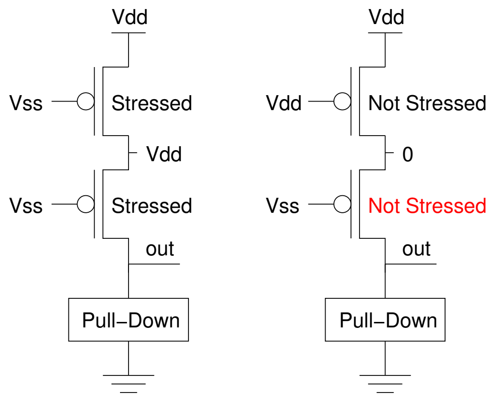

Figure 3.2. For a NOR gate, destressing the top PMOS destresses all subsequent transistors in the stack.

source node voltage, except for the top transistor in the PMOS stack, is dependent on the state of the transistors higher in the stack [19]. Specifically, the source node voltage for any transistor below an "off" transistor will be close to ground and thus, even for a low input, V_{gs} will not approach $-V_{dd}$. This is illustrated in Figure 3.2. While this reduces (to one) the number of high inputs needed to eliminate static NBTI stress in a NOR gate, it does not help with the problem of inverting logic. A single high input to a NOR gate will force the output low, and thus will still potentially stress the subsequent gate.

To eliminate static NBTI stress on all the PMOS transistors in a circuit, the outputs of most gates must be forced high. Gates feeding only into the lower PMOS transistors of NOR gates are the exception. Because of the increase in delay associated with INC insertion, the performance gained (rather, retained) due to NBTI minimization is less than that lost due to INC insertion at every gate. It is not practical to cover every gate with INC. Focused mitigation is required. That is, it is necessary to find the set of nodes for INC insertion that minimizes the overall circuit delay in the presence of NBTI.

The relevant transistors for NBTI stress removal are those on the critical path or those which, due to NBTI degradation over circuit lifetime, may ultimately be on this path. That is, a critical transistor is one with a timing slack less than its NBTI-induced increase in delay. If all of these transistors can be placed in unstressed states, static NBTI will not increase system delay. Unfortunately, identifying these critical transistors is hard. The slack for each gate depends on the delays of all the prior and subsequent gates along its path. Therefore, it is dependent on the NBTI stress and node control delay of each of those gates as well. The addition of node control to a single gate can, in the worst case, change the slack of every other gate in the circuit. These control nodes introduce additional delay that, depending on their locations, may adversely affect the critical path. It is thus necessary to optimally trade off the reduction in NBTI-induced delay and the increase in delay due to the addition of INC.

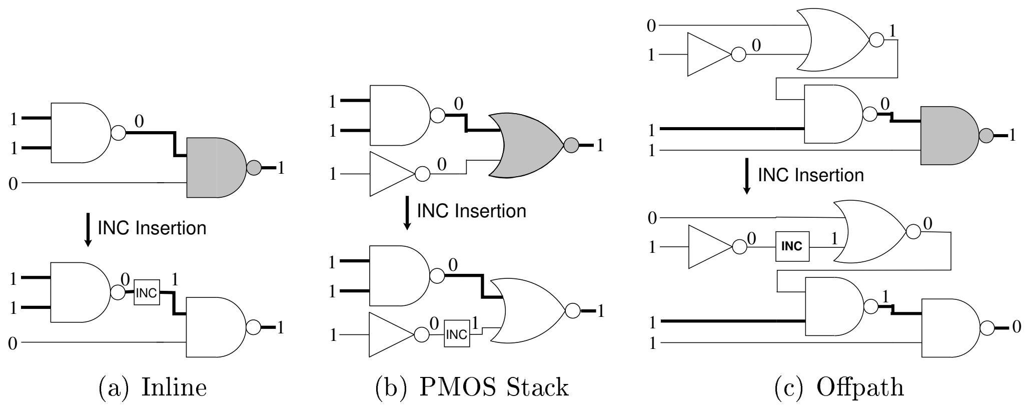

Figure 3.3. Three example INC insertion scenarios. Gates affected by NBTI are shaded and critical path lines are darkened.

Figure 3.3 shows three example INC insertion scenarios. These are intended as examples of a subcircuit far removed from the primary inputs of the circuit containing it.

Because IVC loses effectiveness as circuit depth increases, the input vectors for these examples are fixed and IVC is not considered. The critical path in each circuit is shown in bold, and critical path gates affected by NBTI are shaded. For simplicity, in these examples we assume that INC insertion does not change the location of the critical path, but only its delay. More specifically, we assume sufficient slack for INC insertion.

Figure 3.3(a) shows the insertion of node control inline with the critical path. In this case, the delay added by the node control on the first gate must be less than the NBTI delay on the second gate or the INC should not be added. Figure 3.3(b) illustrates the removal of NBTI from a NOR gate using the previously explained observation about the series PMOS stack.

The last example is more complicated. In Figure 3.3(c), NBTI stress is removed from the second NAND gate by inserting an INC node off of the critical path such that the correct value propagates through to the critical gate. In this scenario, the node control can be added to a gate with sufficient slack, even if that gate is several gates removed from the critical path. Note that although the second NAND gate is stressed by NBTI after INC insertion, the stress does not occur on a critical path input and therefore does not increase total circuit delay.

3.3. Problem Definition and Complexity

We formulate the task of NBTI-induced delay reduction via INC as an optimization problem. The locations of internal node controls are selected in order to minimize the total combinational delay due to both INC overhead and static NBTI after some specified period of time. The input to the problem consists of a combinational circuit represented as a graph of connected gates. For each gate, three delays are specified: (i) the basic delay for an unmodified gate, (ii) the increase in delay if INC is added, and (iii) the increase in delay after some period of NBTI-stress, e.g., 10 years. The task is to find the input vector and node control insertion points that minimize the critical path delay after it has been subjected to NBTI stress. In other words, the goal is to minimize the increase in delay between the original unstressed circuit and the INC-modified, stressed circuit.

We will show that the decision version of this problem is $\mathcal{NP}$-complete. In the decision problem, instead of minimizing the critical path delay, a delay bound b is specified and the task is to determine whether an input vector and set of INC placements exist such that the critical path delay is less than or equal to b.

Lemma 1. *The problem of IVC selection and INC placement for NBTI minimization is in $\mathcal{NP}$.*

Proof. A solution can be easily checked in polynomial time by computing the associated critical path delay. Specifically, the delay can be determined in time linear to the number of gates via a simple topological traversal of the circuit, computing the gate output values and arrival times. □

Lemma 2. *The problem of IVC selection and INC placement for NBTI minimization is $\mathcal{NP}$-hard.*

Proof. To prove that the problem is $\mathcal{NP}$-hard, we use a reduction from circuit-SAT. In circuit-SAT, the task is to decide, for a given boolean circuit C with a single output, if there is an assignment to the inputs such that the output is true. We give a polynomial-time transformation from an instance of circuit-SAT to our problem with specified bound $b = 0\,\mathrm{s}$.

For each gate in the circuit C, the intrinsic delay and NBTI delay are set to $0\,\mathrm{s}$. This ensures that for any inputs, the critical path delay is $0\,\mathrm{s}$. The INC delays are set to a positive value (e.g., $1\,\mathrm{s}$), thus ensuring that a solution satisfying the bound $b = 0\,\mathrm{s}$ will not have any INC nodes and that the Boolean function implemented by the circuit remains unmodified.

We add a *gadget* to the output of the circuit in such a way that the critical path delay is greater than $0\,\mathrm{s}$ if the output is false, and $0\,\mathrm{s}$ if the output is true. Specifically, an inverter is inserted at the output of the circuit. The basic delay is set to $0\,\mathrm{s}$ and the NBTI delay is set to a positive value (e.g., $1\,\mathrm{s}$). The INC delay is unimportant but can be set to $0\,\mathrm{s}$. If the output of the original circuit C is true, the inserted inverter will not be stressed by NBTI, and the critical path delay will be $0\,\mathrm{s}$. If the output is false, the inverter will be stressed and the delay will be positive, thereby exceeding the bound $b = 0\,\mathrm{s}$.

In short, any circuit-SAT problem instance can be solved as an instance of our problem using this transformation. The circuit-SAT instance has an accepting input assignment if and only if the transformed problem has an input assignment leading to a critical path delay of $0\,\mathrm{s}$. $\square$

Theorem 1. *The problem of IVC selection and INC placement for NBTI minimization is $\mathcal{NP}$-complete.*

Proof. This follows directly from Lemma 1 and Lemma 2. $\square$

CHAPTER 4

Optimal Formulation and Solution

4.1. Mixed Integer Linear Programming Formulation

In order to find the optimal solution, we describe the optimal mixed integer linear program (MILP) formulation.

A combinational circuit is modeled as a directed acyclic graph $G = (V, E)$. V is a set of primary inputs ($I \subset V$), gate outputs ($N \subset V$), and primary outputs ($Q \subset V$). E is a set of directed edges modeling connections between two gates. The gate outputs N, are further divided into three sets N_I, N_R, and N_D representing NOT, NOR, and NAND gates. P_v are the predecessors of v.

The instrinsic delay of a gate is $\tau_{n \in N}$. The increase in delay due to NBTI stress is $\rho_{n \in N}$, and the increase in delay due to the addition of node control on the gate output is $\phi_{n \in N}$.

The following variables are used. $\sigma_{n \in N}$ is a binary variable:

$$\sigma_n = \begin{cases} 1 & \text{if INC is added to gate n} \\ 0 & \text{otherwise} \end{cases}$$

$\kappa_{n \in N}$ is a binary variable representing the forced value of node n, if σ_n is 1.

$0 \leq \psi_{v \in V} \leq 1$ is the value of node v. If σ_v is 1, then ψ_v is κ_v. Otherwise, it is determined by the inputs to the gate. For $v \in I$, ψ_v is explicity constrained to be binary. $\alpha_{v \in V}$ is the earliest arrival time at node v.

We optimize the circuit delay by minimizing the maximum output arrival time:

$$\text{minimize} \max_{\forall q \in Q} \alpha_q \tag{4.1}$$

The Boolean function of the gates, combined with the node control, is modeled by a set of constraints that force each output ψ_v to the proper value based on σ_v, κ_v, and the inputs to node v. These constraints are equivalent to those specifying the convex hull of the function, where each input and output represents one dimension. For example, the truth table for an inverter, shown in Figure 4.1, leads to the following constraints. NAND and NOR gates are similarly determined.

$$\begin{aligned}
\forall n \in N_I: \quad & \sigma_n + \kappa_n - \psi_n && \leq 1 \\
& \sigma_n - \kappa_n + \psi_n && \leq 1 \\
& -\kappa_n + \psi_n + \psi_p - 1 && \leq 0 \\
& -\sigma_n + \psi_n + \psi_p - 1 && \leq 0 \\
& -\psi_p + \kappa_n - \psi_n && \leq 0 \\
& -\psi_p - \sigma_n - \psi_n && \leq -1
\end{aligned}$$

The earliest arrival times are modeled by constraining a node v's arrival time to be later than or equal to all of its inputs' arrival times plus any delays associated with the gate. The instrinsic delay τ_v of each gate is always included. The internal node control delay ϕ_v is only included if σ_v is 1. The NBTI delay ρ_v is included when, based on the

ψ_p	σ_n	κ_n	ψ_n
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figure 4.1. Truth table for an inverter with INC. ψ_p is the input, σ_n is the INC selection variable, κ_n is the forced value, and ψ_n is the output.

inputs, the gate is stressed. For NOT and NAND gates, the following constraint enforces this relationship.

$$\forall n \in N_I \cup N_D, \forall p \in P_n : \ \alpha_n \geq \alpha_p + \tau_n + (1 - \psi_p)\rho_n + \sigma_n \phi_n$$

As discussed in the previous chapter, if any input to a NOR gate is high, we assume that the whole gate is unstressed. The variable $0 \leq \gamma_{n \in N_R} \leq 1$ is 1 if NOR gate n is stressed and 0 otherwise. Thus, the following constraints implement the arrival time computation for NOR gates.

$$\begin{aligned} \forall n \in N_R, \forall p \in P_n : \quad & 1 - \gamma_n \geq \psi_p \\ & 1 - \gamma_n \leq \sum_{r \in P_n} \psi_r \\ & \alpha_n \geq \alpha_p + \tau_n + \gamma_n \rho_n + \sigma_n \phi_n \end{aligned}$$

Optimization Objective 4.1 ensures that the arrival times on the critical path are minimal.

4.2. Experimental Results

We evaluated the proposed technique on the ISCAS85 combinational benchmarks [3]. The experimental setup and results are presented below. We also provide some analysis of the variance in results seen across the benchmark set.

4.2.1. Experimental Setup

In order to gauge the performance of the method, it was tested on the ISCAS85 combinational benchmark suite for a 7 gate library {inv, nor2, nor3, nor4, nand2, nand3, nand4}. For consistency, the gates were sized for a maximum fanout of three. Timing information for the gates (with and without node control) was obtained through HSpice simulations using the 65 nm Berkeley Predictive Technology Model [18, 17]. The timings for these self-developed gates, without node control, were calibrated to similar gates in a TSMC 65 nm library to ensure that the timings were representative of real-world libraries [20]. Static NBTI delay was assumed to be about 10% of the initial gate delay after 10 years of stress [2]. The benchmarks were mapped to the library using Synopsis Design Compiler.

The MILP was solved using the open-source software SYMPHONY [21] for two different cases.

(1) To determine the delay using optimal input vector control only, the solver was run with INC disabled (i.e., the node control selection variables forced to 0).
(2) Then, the solver was run with internal node control enabled.

The resulting problem instances are rather large for an MILP solver. Therefore, the solver was set to stop solving and report the results when the upper and lower bounds for the optimal delay were within 0.2% of each other.

4.2.2. Analysis of Results

The circuit delay results for each benchmark are shown in Table 4.1. Column “Baseline” shows the initial delay of the circuit, before any NBTI degradation has occurred. Although we are primarily interested in the improvement in delay between the optimal IVC-only and optimal IVC+INC cases, the next five columns provide additional information, specifically the delays associated with five representative input vector scenarios, designed to help the reader interpret our results. The “All 0’s” and “All 1’s” columns report the delay, after 10 years of NBTI stress, for an all zero-valued and all one-valued input vector, respectively. The “Min”, “Max”, and “Average” columns shows the minimum, maximum, and average delays, respectively, for a set of 100000 randomly generated input vectors, once again considering 10 years of NBTI stress.

These results show that neither an all-zeros or all-ones input vector is a good choice. Neither is routinely better than the other and both are much worse the minimum-observed delay and the optimal delay. The average delay can be interpreted as the delay if neither IVC nor INC was applied. If IVC is not used, when a functional unit is idle, it’s inputs will likely be those from the last cycle in which it was used. These inputs vary each time the unit is placed into an idle state, and thus can be approximated by our set of randomly generated input vectors.

For these circuits, the minimum-observed IVC delays are quite similar to the optimal IVC-only delays. This suggests that for circuits which are too large for the MILP solver, the minimum delay across a set of randomly generated input vectors can provide a good estimate for the optimal IVC-only delay. We will use this observation in chapter 6 when testing the benefits of INC on some larger circuits.

Table 4.1. Path Delays (ns) for ISCAS85 circuits

Circuit	**Baseline**	**Non-optimal Input Vectors**					**Optimal IVC Only**			**Optimal IVC + INC**		
		All 0's	**All 1's**	**Min**	**Max**	**Average**	LB	UB	% Gap	LB	UB	% Gap
c432	1650.6	1707.7	1717.8	1701.3	1741.5	1725.6	1697.9	1701.3	0.20	1690.7	1691.8	0.07
c499*	1588.7	1660.9	1642.7	1641.2	1669.7	1658.5	1633.0	1641.2	0.50	1626.7	1629.6	0.18
c880	1884.3	1970.6	1965.4	1935.6	1981.2	1964.6	1926.7	1927.9	0.06	1911.4	1914.0	0.14
c1355*	1505.1	1567.8	1562.7	1555.5	1588.4	1570.2	1546.8	1555.5	0.56	1534.6	1541.5	0.45
c1908	2112.0	2182.9	2209.8	2176.9	2216.6	2198.7	2175.0	2176.1	0.05	2171.3	2175.7	0.20
c2670*	1607.1	1668.3	1679.9	1657.1	1692.3	1675.3	1651.3	1657.1	0.35	1627.5	1629.1	0.10
c3540*	2546.4	2651.8	2648.4	2624.7	2673.2	2649.8	2615.6	2624.7	0.35	2595.7	2598.4	0.11
c5315	2396.9	2499.7	2485.3	2451.7	2518.3	2490.4	2448.2	2450.9	0.11	2435.6	2435.8	0.01

* Solver was stopped after 24 hours but before the 0.2% stop gap was reached.

Table 4.2. % Improvement in NBTI-induced Delay

Circuit	**% Improvement over Optimal IVC**			**% Improvement over Average IVC**		
	LB	**Mid**	**UB**	**LB**	**Mid**	**UB**
c432	12.9	17.0	20.9	45.1	45.8	46.5
c499	7.8	18.6	27.7	41.4	43.5	45.6
c880	30.1	34.1	37.9	63.0	64.6	66.3
c1355	12.7	28.4	41.5	44.1	49.4	54.7
c1908	−1.1	3.2	7.5	26.5	29.1	31.6
c2670	50.3	55.0	59.2	67.7	68.9	70.1
c3540	24.8	31.3	37.1	49.7	51.0	52.3
c5315	24.2	26.3	28.4	58.4	58.5	58.6

* Solver was stopped after 24 hours but before the 0.2% stop gap was reached.

Even with the 0.2% stop gap for the solver, it did not finish for several of the benchmarks after several days. Thus, for the reported results, we manually terminated execution when the solver had been running for more than 24 hours. For all circuits we report both the lower and upper bounds. Table 4.2 shows the percent reductions in delay.

The column "% Improvement over Optimal IVC" shows the improvement attributable to INC alone. Specifically, the improvement is reported as the percent reduction in NBTI-induced delay between the IVC-only and the IVC+INC cases:

$$\%_{improve} = 100 \times \frac{(D_{inc} - D_{base}) - (D_{ivc} - D_{base})}{D_{ivc} - D_{base}}$$

We report lower and upper bounds on this improvement as well. The lower bound is computed using the IVC lower bound and the IVC+INC upper bound. The upper bound is computed using the IVC upper bound and the IVC+INC lower bound. We also report a midpoint improvement which is calculated using the midpoints of the IVC bounds and the IVC+INC bounds.

The average midpoint improvement is 26.7% with a standard deviation of 15.1%. The average of the lower bounds is 20.2% and the average of the upper bounds is 32.5%. However, the improvement depends quite heavily on the benchmark. For benchmark c2670 over 50% of the degradation is prevented, while for benchmark c1908 the upper bound shows only single-digit improvement. The calculated lower bound on improvement for c1908 is negative. Obviously, the optimal worst case lower bound is 0%, if no INC placements are added. However, the best IVC+INC solution found by the solver (an upper bound on optimal) is worse than the IVC lower bound, leading to the negative improvement.

Column “% Improvement over Average IVC” shows the reduction in NBTI-induced delay due to the combination of INC and IVC. As mentioned previously, we use the average IVC delay from our set of randomly generated input vectors as the delay if neither IVC nor INC were used. The percent improvement is then computed as the reduction in NBTI-induced delay between the optimal IVC+INC case and the average IVC case. The average improvement in delay is 51.3%. That is, on average, half of the NBTI-induced delay can be prevented by the combined use of input vector control and internal node control. Similar to the previous results, the improvement is benchmark dependent with c2670 showing a 68.9% reduction while c1908 shows only a 29.1% reduction.

We do not discuss the area or power impacts of INC here because the MILP formulation optimizes only the critical path delay, but not the total number of INC placements. Chapter 5 describes a near-optimal heuristic that optimizes the delay while attempting to minimize the number of modified gates. The results in section 5.2 show that near-optimal delays can be achieved with little impact on area and power consumption.

4.2.3. Analysis of Variance

One potential cause for the high variance of the improvements among the benchmarks is that, due to the short critical paths of these circuits, the removal of NBTI from a single gate on the critical path has a large impact on the percentage improvement. In the best case, INC will remove NBTI stress from all critical path gates. Thus, for circuits such as these, with critical path lengths of 10 to 20 gates, each gate represents 5% to 10% of the total delay. Therefore, removing NBTI stress from one additional gate can add 5–10 percentage points to the delay improvement.

More formally, we hypothesize that the removal of NBTI stress from each critical path gate can be treated as an independent Bernoulli trial. In reality, there is some dependence between successive gates. However, we can safely assume independence because the actual dependence is limited to a few levels of logic. By the law of large numbers, as the number of gates on the critical path increases, one can expect that the observed improvements will be closer to the expected or average improvement. If this hypothesis is correct, it can explain the high variance seen in the ISCAS85 benchmark set, which has relatively short critical paths.

We tested this hypothesis by developing two sets of random circuits, one with short critical paths and one with long critical paths. To make the random circuits as realistic as possible, we employed the CGEN circuit parameterization and generation package [22]. CGEN was designed to help CAD researchers develop random-yet-reasonable circuits to test their proposed algorithms. The package includes two utilities. The first is used to characterize a circuit by the extraction of parameters such as circuit shape, average fan-in, and average fan-out. The second utility takes a set of these parameters as input and generates a random circuit with similar parameters.

We characterized ISCAS85 benchmark c880 and generated 50 random circuits for the short critical path circuit set. To create a long critical path circuit to characterize, we linked four instances of the c880 circuit serially, arbitrarily connecting the outputs of one stage to the inputs of the next. 50 random circuits were generated from the characterization of this circuit to create the long critical path circuit set. Each of these circuits was then solved for the baseline, IVC-only, and IVC+INC cases as described subsection 4.2.1. Due to the size of the long circuits, the MILP solver was set to stop when the upper and lower bounds were within 0.5% of each other.

Table 4.3. Analysis of improvement for the sets of random circuits.

Circuit Set	**Average (%)**	**Standard Deviation (%)**
Short	22.4	19.8
Long	42.0	9.3

The results of this test are shown in Table 4.3. There is a significant decrease in variance, which is consistent with our hypothesis. The standard deviation falls from 19% for a random set of short critical path circuits to 9% for a random set of circuits 3-4$\times$ longer. The 14% standard deviation for the ISCAS85 set falls between the deviations for the long and short critical path sets. Because circuits in the short and long critical path sets have similar lengths to the extremes of the ISCAS85 set, this relationship is also consistent with our hypothesis. Thus, we conclude that some of the variance in improvement for the original ISCAS85 benchmarks is due to the limited length of the circuit critical paths.

4.3. Multiple-Vector INC

In the preceeding analysis, we have assumed that only a single input vector and set of INC placements could be chosen for a given functional unit. Consequently, the same set of PMOS transistors are stressed whenever the unit is idle and sleep mode activated. If several input vectors were chosen and alternately applied when the unit is idle, it is possible that the NBTI stress could be spread out over more transistors, resulting in less total timing degradation. This idea was proposed by Abella, Vera, and Gonzalez, but they did not perform analysis of the potential benefits of the technique and left development of input vector selection algorithms for future work [14]. In this section, we investigate the potential of multiple-vector INC for the ISCAS85 benchmarks. We find that in most cases, the addition of multiple input vectors does not decrease the critical path delay. In the few cases where some benefit is seen, the decrease in delay is small enough that it is unlikely to outweigh the increase in area required to implement the multiple vector strategy.

Internal node control requires the selection of both an input vector and a set of INC placements. Sets of inputs vectors and sets of INC placements could be cycled through to potentially reduce the NBTI degradation on individual gates. However, the implementation of such a technique has some drawbacks. In the case of multiple input vectors, all the vectors must be stored and muxes or scan-chain logic is needed to route the chosen input vector to the functional unit. In the case of INC, additional gates must be modified to support INC and multiple sleep signals must be routed across the functional unit, further increasing area, delay, and leakage power. In both cases, a small state machine must also be included to select the current input vector and sleep signal.

Because of the additional area, delay, and routing complexity imposed by the use of multiple INC placement sets, we focus this analysis on the use of multiple input vectors only. INC placements are still used, but the same placements are used across all cycles (i.e., for all input vectors). For the remainder of this discussion, the application of an input vector is referred to as a cycle and the total number of input vectors chosen is the number of cycles. Thus, a 4-cycle solution refers to a rotation among four input vectors.

We assume that with any reasonable strategy for cycling among the input vectors, in the long run all the vectors will be used for approximately the same amount of time.

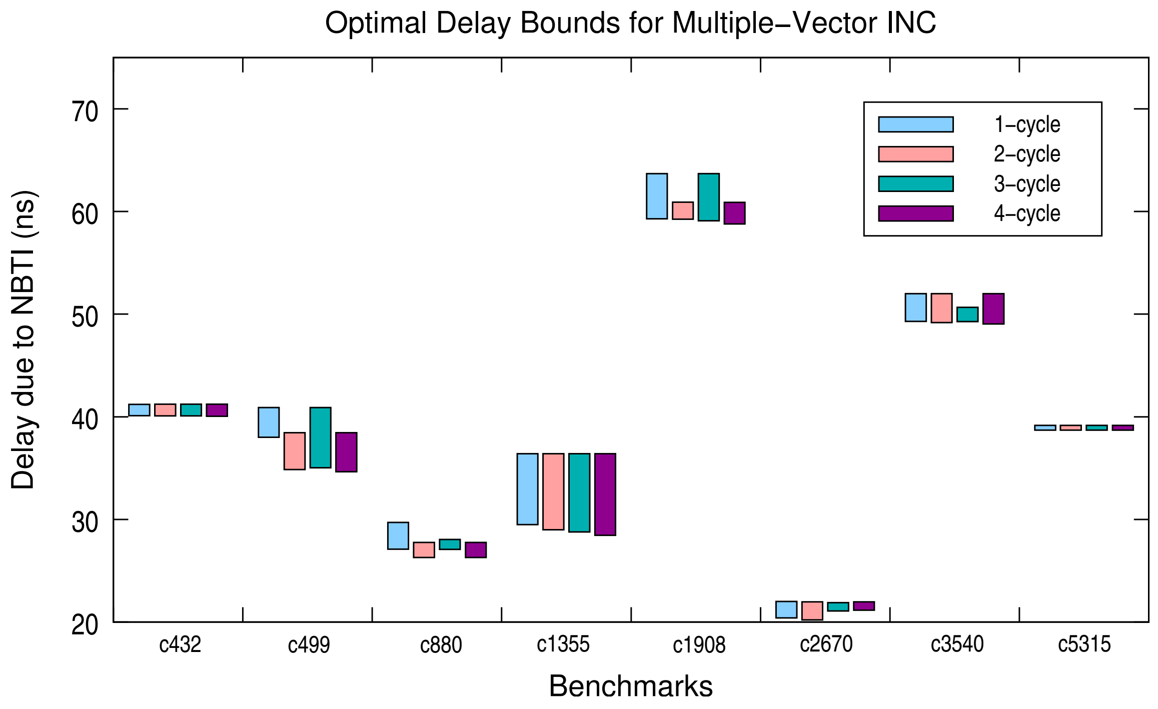

Figure 4.2. Upper and lower bounds for multi-cycle INC for the ISCAS85 benchmarks.

Thus, the NBTI delay for an individual gate is computed as the fraction of the cycles for which the gate is stressed times the delay if the gate were stressed for the entire 10 year period.

We modified our MILP formulation to support an arbitrary, user-defined number of cycles and ran it for 2, 3, and 4 cycles on the ISCAS85 benchmarks. Figure 4.2 presents the results. As with the single-cycle problem instances presented earlier, the solver was not able to find optimal solutions in a reasonable amount of time, and therefore we present the lower and upper-bounded ranges within which the optimal solutions lie. For each benchmark, the results for 1, 2, 3, and 4 cycles are shown side-by-side. The range in which the optimal solution lies for each problem instance is represented by a bar extending from the lower bound to the upper bound. Multi-cycle solutions are free to repeat a vector.

Therefore, the upper bound for any multi-cycle problem instance can be no higher than the upper bound for the corresponding single-cycle problem instance. Thus, for instances where the upper bound proved by the MILP solver is greater than the corresponding single-cycle upper bound, we report the single-cycle upper bound in its place.

As can be seen from Figure 4.2, the use of multiple input vectors does not significantly decrease the circuit delay. When one considers that these results do not account for the extra delay that would be imposed by the additional circuitry required to implement the multiple-vector strategy, the results are even less encouraging. Only benchmark c499 shows a significant decrease in the lower bounds for the multiple-vector cases. Even in this case, there may still not be any real improvement because the single-vector range still overlaps with multiple-vector ranges. Additionally, the potential improvement is seen only when going from one cycle to two. The increases to three and four cycles do not improve upon the two-cycle bounds. It is likely that the potential for the small amount of improvement for c499 is because of the several parallel critical paths in this benchmark. The use of two input vectors may allow some degradation balancing between them. For the other benchmarks, which have more well-defined single critical paths, the use of multiple vectors has little impact. Although our analysis suggests that INC reduces the severity of NBTI-induced degradation, it also indicates that using multiple vectors brings no additional improvement.

CHAPTER 5

Linear-Time Heuristic

The MILP-based optimal solution method is not practical for large circuits because this problem is $\mathcal{NP}$-complete. A heuristic solution that provides good, and ideally near-optimal, solutions in a reasonable amount of time is necessary. In this chapter, we describe a linear-time algorithm for input vector selection and internal node control placement. Our technique draws from work on leakage power minimization by Cheng, Chen, and Wong [23].

5.1. Algorithm Description

5.1.1. Overview

Our heuristic (see 1) takes advantage of the fact that the problem can be solved optimally for rooted-tree structures in linear time using dynamic programming. It first

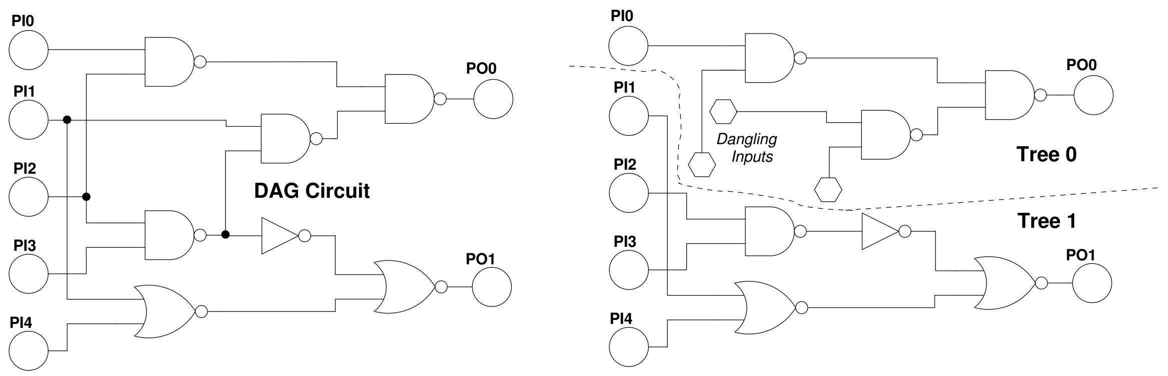

Figure 5.1. A circuit partitioned into rooted trees, with the *dangling inputs* labeled.

Algorithm 1 INC Placement Heuristic Overview

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:    for all partitions do
 5:       choose IVC and INC using dynamic programming
 6:    end for
 7:    update dangling input values
 8:    if solution is the same as previous then
 9:       break {Check for convergence}
10:    end if
11:    if oscillation is detected then
12:       repartition the circuit
13:    end if
14: end for
15: greedily remove INCs which do not affect delay
16: return input vector and INC placements
```

partitions a given circuit into rooted trees by removing some connections between gates (line 1). This partitioning creates *dangling inputs* at these gates whose input connections were removed, as illustrated in Figure 5.1. Values are assigned to these dangling inputs (line 2) and the optimal values for the primary inputs and INC placements are chosen for each partition (lines 4–6). The values for the dangling inputs are updated based on the new outputs of their parent gates in the original circuit (line 7) and the solutions for the partitions are recomputed based on these new dangling input values (line 3). This iteration continues until the solution has converged (lines 8–10) or a pre-set number of iterations has been reached (line 3). Convergence is identified when the values for the dangling inputs do not change between two consecutive iterations. To ensure convergence, when the revisitation of a solution is detected, the circuit is repartitioned (lines 11–13).

Empirical results show that this repartitioning breaks oscillations and leads to convergent solutions.

5.1.2. Partitioning and Initial Solution

Solution quality is highly dependent on the method used to partition the circuit and the initial values assigned to the dangling inputs. Tree-based partitioning has been proposed for several circuit design problems in the past, including leakage power minimization and technology mapping [23]. For these problems, the cost function (e.g., total leakage power, circuit area) is additive: the overall cost is essentially the sum of the costs of the individual partitions. It is thus important to maximize the sizes of the partitions in order to maximize the effectiveness of the optimal dynamic programming algorithm. The specific choice of which connections to remove, though, is not as critical.

For INC placement, the cost function is not additive: the critical path delay for the entire circuit is not the sum of the critical path delays of each partition. Thus, in addition to maximizing the sizes of the partitions, it is also important to keep the original critical path in a single partition. Of course, for circuits with parallel critical paths, this will not always be possible. Our partitioning algorithm tries to ensure that these critical paths are not cut by using slack information to determine which connections to remove. In a rooted-tree structured circuit, each gate has a fanout of 1. Thus, for each gate with a fanout greater than 1, our partitioning algorithm keeps the connection with the smallest slack, removing the others. Dangling inputs are inserted at the broken connections.

As mentioned in the previous paragraph, the choice of the initial values for the dangling inputs is also important. We choose these initial values by applying the optimal dynamic

Algorithm 2 Dynamic Programming Algorithm

```
Require: tree-structured circuit partition $\mathcal{P}$
Require: arrival times and node values for dangling inputs {Forward Pass}
 1: for all gates $g$ in a topological ordering of $\mathcal{P}$ do
 2:    for all combinations of inputs $i$ do
 3:       compute arrival time and output value based on the arrival times of $g$'s parent
          gates
 4:       compute arrival time and output value if INC is added
 5:    end for
 6:    store $i$ with a 0 output and the smallest arrival time
 7:    store $i$ with a 1 output and the smallest arrival time
 8: end for
    {Backward Pass}
 9: choose primary output value with smallest arrival time
10: for all gates $g$ in a reverse topological ordering of $\mathcal{P}$ do
11:    select the stored $i$ with the output that matches the child's selected $i$
12: end for
13: return the input values and the INC placements
```

programming algorithm to the unmodified directed acyclic circuit. Because the circuit is not tree-structured, in the backward pass phase of the algorithm, conflicts will occur. At each gate with a fanout greater than 1, the child gates may require differing output values from their shared parent. In these cases, the value required by the majority of the children is chosen. In the case of a tie, 1 is chosen because, in general, it will prevent NBTI stress on the child gates.

5.1.3. Dynamic Programming

The optimal dynamic programming algorithm is shown in 2. The algorithm takes as input a tree-structured circuit partition and, for each of the dangling inputs, the arrival time and node value. For primary inputs, the arrival time is assumed to be 0 and the node value is determined by the algorithm. The algorithm consists of two phases, the forward pass and the backward pass. In the forward pass, two pieces of information are computed for each gate, the input combination and INC state with a 0 output and the smallest arrival time, and the input combination and INC state with a 1 output and smallest arrival time. Specifically, the gates are examined in a topological order (line 1). For each gate, each possible input combination is examined (line 2). The output value is computed and, based on the arrival times previously computed for the parent gates, the arrival time is computed (line 3). The value and arrival time if INC are added is also computed (line 4). For each output value, 0 and 1, the input combination and INC state with the smallest arrival time is stored (lines 6–7). It is possible for several input vectors to lead to the same minimum arrival time. In the case of such ties, the *covering* input vector is chosen. An input vector x is said to cover input vector y if x has 1 values in the all the positions that y does and has an additional 1 value. For example, the vector "1011" covers the vector "1010". In the backward phase, a specific value (and thus INC state) is chosen for each of the gates. Specifically, the primary output value and corresponding input combination with the smallest arrival time is chosen (line 9). The remaining gates are then examined in a reverse topological order (line 10). For each gate, the required output value is specified by the chosen input combination for its child. The corresponding input combination is selected for the gate (line 11).

5.1.4. Runtime

The heuristic requires time linear in the number of gates. Partitioning is performed with a single topological traversal. The dynamic programming algorithm requires one traversal for each phase. Although all the input combinations for each gate must be examined, this

Table 5.1. Heuristic Results for ISCAS85 circuits

Circuit	**Optimal Delay**			**Heuristic**	**% Worse**	**% Worse**	**Time**
	LB	**Mid**	**UB**	**Delay**	**than Mid**	**than UB**	**(s)**
c432	1690.7	1691.3	1691.8	1700.8	0.56	0.53	1.3
c499	1626.7	1628.1	1629.6	1628.0	-0.01	-0.10	6.7
c880	1911.4	1912.7	1914.0	1914.0	0.07	0.00	2.7
c1355	1534.6	1538.1	1541.5	1547.4	0.60	0.38	5.2
c1908	2171.3	2173.5	2175.7	2175.7	0.10	0.00	3.2
c2670	1627.5	1628.3	1629.1	1629.1	0.05	0.00	11.9
c3540	2595.7	2597.0	2598.4	2595.7	-0.05	-0.10	27.0
c5315	2435.6	2435.7	2435.8	2435.8	0.00	0.00	41.0

is effectively constant time because the number of inputs is restricted. Finally, although the overall algorithm iterates multiple times, empirical results show that it converges rapidly and the number of iterations can be limited to a small number (15 in our reported results).

5.2. Experimental Results

We implemented the proposed heuristic in Python and tested it on the same benchmarks as in section 4.2. All tests were done on a 2.5 GHz AMD Athlon XP computer with 2 GB of memory. The runtime for each benchmark is shown in the "Time" column of Table 5.1.

The results are shown in Table 5.1. For each benchmark, the lower, midpoint, and upper bounds from the optimal solutions are given. The heuristic solutions are first compared with the midpoint, because this was used in section 4.2 to compute the average improvement. In all cases, the heuristic solutions are quite good, with an average 0.17% degradation from the optimal midpoint. For benchmarks c499 and c3540, the heuristic solution is actually better than the optimal midpoints, although obviously not better than

Table 5.2. Area Impact of INC

Circuit	**Total Gates**	**INC Gates**	**Original Trans.**	**INC Trans.**	**% Increase**
c432	159	11	636	22	3.5
c499	526	18	1836	36	2.0
c880	336	11	1306	22	1.7
c1355	480	12	1840	24	1.3
c1908	363	8	1322	16	1.2
c2670	592	13	2302	26	1.1
c3540	725	29	2966	58	2.0
c5315	1452	12	5650	24	0.4

the lower bound. Benchmark c1355 shows the worst degradation with a 0.60% increase in the critical path delay.

We also compare the heuristic solutions to the optimal upper bound because it is entirely possible that for many of these benchmarks, the upper bound actually is optimal. In all cases except two, the heuristic solutions are at least as good as the optimal upper bound. Benchmarks c432 and c1355 show the only degradation, with 0.53% and 0.38% increases in critical path delay, respectively.

Table 5.2 shows the impact on circuit area for the solutions produced by the heuristic. The number of gates modified with INC and the percent increase in transistor count needed to implement INC are shown. On average, INC imposes only a 1.6% area overheard, in contrast with the 8–12% overheard required for gate sizing [9]. Benchmark c432 is the worst with a 3.5% increase. The impact on power consumption will also be small. The average 1.6% increase in transistor count should translate into a similarly-small increase in leakage and switching power consumptions.

CHAPTER 6

Industrial Circuits

6.1. Arithmetic Functional Units

Due to their relatively small gate counts, the ISCAS85 benchmarks were appropriate for use with the optimal MILP formulation. Thus, they were useful both for showing the potential benefit of INC and for verifying that the heuristic can produce near-optimal results (at least for small benchmarks). However, real industrial designs for which INC would be most useful will, in most cases, be much larger. Arithmetic circuits are one such class of larger circuits for which INC, intended for use on functional units that are frequently idle, could be used. In this chapter, we present the results of our internal node control heuristic applied to several arithmetic circuits.

Table 6.1 lists the arithmetic circuits we used for evaluation. The adder/subtracter and multiplier designs were synthesized from high-level VHDL descriptions using Synopsys Design Compiler. The sources for other designs are given in the table. All designs were

Table 6.1. Descriptions of Arithmetic Circuit Benchmarks

Circuit	**Source**	**Description**
addsub8	Synopsys Design Compiler	single-cycle 8-bit unsigned adder/subtracter
addsub16	Synopsys Design Compiler	single-cycle 16-bit unsigned adder/subtracter
addsub32	Synopsys Design Compiler	single-cycle 32-bit unsigned adder/subtracter
addsub64	Synopsys Design Compiler	single-cycle 64-bit unsigned adder/subtracter
mult8	Synopsys Design Compiler	single-cycle 8-bit unsigned multiplier
mult16	Synopsys Design Compiler	single-cycle 16-bit unsigned multiplier
mult32	Synopsys Design Compiler	single-cycle 32-bit unsigned multiplier

Table 6.2. Path Delays (ns) for Arithmetic Circuits

Circuit	**Baseline Delay**	**Non-optimal Input Vectors Delay**					**Heuristic Delay**
		All 0's	**All 1's**	**Min**	**Max**	**Average**	
addsub8	2167.6	2230.0	2265.2	2221.3	2280.8	2253.8	2211.7
addsub16	3976.9	4078.6	4158.5	4078.6	4183.3	4133.4	4059.5
addsub32	7618.5	7797.1	7966.7	7797.1	7992.2	7915.6	7778.0
addsub64	14997.9	15335.3	15684.0	15335.3	15698.0	15581.4	15316.2
mult8	3132.7	3277.6	3269.6	3228.7	3294.6	3264.6	3197.7
mult16	5654.8	5902.1	5883.0	5845.1	5930.1	5891.4	5756.2
mult32	10734.8	11188.1	11162.7	11113.1	11239.8	11181.9	10913.2

mapped using Design Compiler to the same gate library used for the ISCAS85 benchmarks in the preceeding chapters.

6.2. Experimental Results

As with the ISCAS85 benchmarks, we determined both the baseline delay (i.e., the initial delay before any NBTI degradation has occurred) and the delays for a set of 100,000 random input vectors. Because these circuits are so large, we could not solve for the optimal input vector and its delay. However, as noted in section 4.2, for the ISCAS85 benchmarks the minimum delay seen over the set of 100,000 random input vectors was close to the delay for the optimal input vector. Thus, for the following analysis we use this minimum delay as a substitute for the optimal IVC-only delay. Our proposed heuristic was used to determine the INC placements and the associated delay.

Table 6.2 shows the results for the arithmetic circuits. The *addsub* circuits show close to a 50% decrease in NBTI-induced delay when compared to the average IVC delay. In other words, the combination of IVC and INC eliminates approximately half of the NBTI-caused delay. The percent improvement over the minimum IVC delay (i.e., the improvement in delay due to INC specifically), however, decreases from 17.9% to 5.7% as

Table 6.3. % Reduction in Delay for Arithmetic Circuits

Circuit	**% Improvement**	
	over IVC Average	**over IVC Min**
addsub8	48.8	17.9
addsub16	47.2	18.8
addsub32	46.3	10.7
addsub64	45.5	5.7
mult8	51.1	32.3
mult16	57.2	46.7
mult32	60.1	52.9

the length of the circuit increases from 8 to 64 bits. This suggests that for this particular type of adder/subtracter, INC loses effectiveness as the circuit size increases. This reduction in effectiveness may be due to the chain-like structure of the adder/subtracter. INC may help reduce degradation in a fixed portion of the structure, but not in the chain of gates that extends as gates are added. The opposite is true for the *mult* circuits. The improvement over the average IVC delay increases from 51.1% to 60.11% when the multiplier is scaled from 8 bits to 32 bits. The percent reduction in delay attributable directly to INC, the decrease compared to the minimum IVC delay, also increases from 32.3% to 52.85%. This suggests that INC is effective for the large, regular, grid-like structures used in multiplication circuits. The grid-like structure may allow for more offpath INC placements than is possible with the chain-like structure of the adder/subtracter.

CHAPTER 7

Conclusion

7.1. Conclusion

In this thesis, we proposed the use of internal node control to minimize the impact of static NBTI on circuits with frequently-idle functional units. Optimal placement of internal node controls, which allows the output of an INC-modified gate to be forced to a specific value during sleep mode, and optimal selection of the primary input vector, lead to a 51.3% decrease in NBTI-induced delay for the ISCAS85 benchmarks. The application of INC lead to a 26.7% decrease in delay relative to input vector control alone.

The problem is $\mathcal{NP}$-complete, so we developed a linear-time heuristic that quickly produces good solutions. The problem is tractable for tree-structured circuits, so the heuristic first partitions a given circuit into trees by removing edges. By ensuring that the gates on the critical path in the original circuit remain in the same partition, the optimal solutions to these partitions can be used to provide good solutions for the overall circuit. The heuristic solutions were within 0.17% of optimal on average and resulted in only a 1.6% increase in area.

The technique also worked well on larger, more representative circuits, showing a similiar 50% decrease in NBTI delay and suggesting that the technique will maintain high speed and high solution quality for larger, industrial-scale designs. The combination of internal node control and input vector control can significantly reduce static NBTI degradation with only a minor increase in circuit area and power consumption.

7.2. Future Work

The work presented in this thesis focused on minimizing static NBTI stress on combinational logic gates. However, the transistors in sequential elements such as flip-flops are also susceptible to NBTI. Degradation on these elements could lead to setup and hold time violations. An analysis of the potential severity of this problem could be performed to determine if the development of mitigation techniques would be worthwhile.

Internal node control can be used to reduce NBTI degradation during idle periods, but is not appropriate for the minimization of dynamic NBTI. Although some degradation is unavoidable when a circuit is doing useful work, if the NBTI problem continues to worsen as is predicted for future technology processes, mitigation techniques for dynamic stress will become more important.

References

[1] M. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," *Microelectronics Reliability*, vol. 45, no. 1, pp. 71–81, Jan. 2005.

[2] B. C. Paul, K. Kang, J. Kufluoglu, M. A. Alam, and K. Roy, "Impact of NBTI on the temporal performance degradation of digital circuits," *IEEE Electron Device Ltrs.*, vol. 26, no. 8, pp. 560–562, Aug. 2005.

[3] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," in *Prof. Int. Symp. Circuits and Systems*, May 1985, pp. 695–698.

[4] D. K. Schroder, "Negative bias temperature instability: What do we understand?" *Microelectronics Reliability*, vol. 47, no. 6, pp. 841–852, June 2007.

[5] J. Stathis and S. Zafar, "The negative bias temperature instability in MOS devices: A review," *Microelectronics Reliability*, vol. 46, no. 2–4, pp. 270–286, Feb. 2006.

[6] V. Huard, M. Denais, and C. Parthasarathy, "NBTI degradation: From physical mechanisms to modeling," *Microelectronics Reliability*, vol. 46, no. 1, pp. 1–23, Jan. 2006.

[7] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *J. Applied Physics*, vol. 94, no. 1, pp. 1–18, 2003.

[8] Y. Wang, H. Luo, K. He, R. Luo, H. Yang, and Y. Xie, "Temperature-aware NBTI modeling and the impact of input vector control on performance degradation," in *Proc. Design, Automation & Test in Europe Conf.*, Apr. 2007, pp. 546–551.

[9] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and minimization of PMOS NBTI effect for robust nanometer design," in *Proc. Design Automation Conf.*, July 2006, pp. 1047–1052.

[10] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the NBTI effect for reliable design," in *Proc. Custom Integrated Circuits Conf.*, Sept. 2006, pp. 189–192.

[11] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "An analytical model for negative bias temperature instability," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2006, pp. 493–496.

[12] K. Kang, K. Kim, A. E. Islam, M. A. Alam, and K. Roy, "Characterization and estimation of circuit reliability degradation under NBTI using on-line I_{DDQ} measurement," in *Proc. Design Automation Conf.*, June 2007, pp. 358–363.

[13] K. K. Saluja, S. Vijayakumar, W. Sootkaneung, and X. Yang, "NBTI degradation: A problem or a scare?" in *Proc. Int. Conf. VLSI Design*, Jan. 2008, pp. 137–142.

[14] J. Abella, X. Vera, and A. Gonzalez, "Penelope: The NBTI-aware processor," in *Proc. Int. Symp. Microarchitecture*, Dec. 2007, pp. 85–95.

[15] Y. F. Tsai, D. Duarte, N. Vijaykrishnan, and M. Irwin, "Characterization and modeling of run-time techniques for leakage power reduction," *IEEE Trans. VLSI Systems*, vol. 12, no. 11, pp. 1221–1232, Nov. 2004.

[16] A. Abdollahi, F. Fallah, and M. Pedram, "Leakage current reduction in CMOS VLSI circuits by input vector control," *IEEE Trans. VLSI Systems*, vol. 12, no. 2, pp. 140–154, Feb. 2004.

[17] "Predictive technology model," http://www.eas.asu.edu/~ptm.

[18] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive mosfet and interconnect modeling for early circuit design," in *Proc. Custom Integrated Circuits Conf.*, Sept. 2000, pp. 201–204.

[19] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "NBTI-aware synthesis of digital circuits," in *Proc. Design Automation Conf.*, June 2007, pp. 370–375.

[20] Taiwan Semiconductor Manufacturing Company, "TSMC 65 nm standard cell library," 2006, http://www.synopsys.com/.

[21] T. Ralphs and M. Guzelsoy, "The SYMPHONY callable library for mixed integer programming," in *Proc. Conf. INFORMS Computing Society.* Springer, Jan. 2005.

[22] P. D. Kundarewich, "Synthetic circuit generation using clustering and iteration," Master's thesis, University of Toronto, 2002.

[23] L. Cheng, D. Chen, and M. D. Wong, “A fast simultaneous input vector generation and gate replacement algorithm for leakage power reduction,” *ACM Trans. Design Automation in Electronic Systems*, vol. 13, no. 2, Apr. 2008.