Variation-Aware and Aging-Aware Design Tools and Techniques for Nanometer-Scale Integrated Circuits

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Saket Gupta

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Sachin S. Sapatnekar

July, 2012

Acknowledgements

I am very grateful to my research advisor, Dr. Sachin S. Sapatnekar, for his support and guidance during my entire stay at the University of Minnesota. When I think of him, I feel a deep appreciation of the enormous amount of heart, time, energy, and efforts he puts into the work and training of each one of his students. He takes the responsibility in taking them forward in PhD, to be a sincere and modest individual, and mature in the ability to be a researcher. His sharpness in getting to the root of the topic and getting striking inspirations for solving tough problems, was quite amazing for me to witness and go through with him. I have therefore been fortunate to have him as an advisor. I have benefited and learnt a lot from his thorough and professional approach in different aspects of scientific research and writing. Due to his constant support, encouragement, and considerate nature, my entire experience as a graduate student has been very nice and exciting.

I also wish to acknowledge my committee members, Dr. Marc Riedel, Dr. Chris Kim, Dr. Antonia Zhai, and Dr. George Karypis, with whom I found a very congenial and friendly atmosphere to learn many aspects and topics apart from my research, which I feel is so very essential for a doctoral student.

I would also like to acknowledge the supports for research: National Science Foundation (NSF) under award CCF-1017778, and the Semiconductor Research Corporation (SRC) under contract 2012-TJ-2234.

I also want to express my thanks to fellow lab-mates Baktash, Pingqiang, Yaoguang, Jianxin, Vivek, and Sravan, who shared their time and knowledge with me, and the special moments spent with them, on the way towards completions of my dissertation research, have become impressions for me to carry throughout.

I feel it would not have been possible for me to smoothly work in my PhD without

the utmost cooperation of my room-mates Ankur Khare, Srijan Aggarwal, and my other friends like Ricky Jain, Pulkit Aggarwal, and Brijesh Kumar, from whom I learnt very unique, practical lessons of PhD and of life. Because of their support and care, I could always focus on my research without any worries and in their amicable presence I never felt lost or lonely. I have been able to complete my PhD only due to the support and guidance I received from so many sources, including my parents. I wish to sincerely thank each and every one of them.

Lastly, I would like to thank my teachers in IIT, who had been the inspirational source for me to take up this challenge of PhD, and whose guidance has been the most valuable for my whole life.

Dedication

To the Source of Everything, and my parents.

Abstract

Shrinking feature sizes in CMOS-based technology beyond the 45nm regime have given rise to increased levels of variation in digital circuits and architectures due to process, temperature, and aging effects. The fabrication process induces variations in the process parameters, causing differing levels of perturbation in the circuit delay in each manufactured part at the postsilicon stage. Moreover, after manufacturing, during the normal operation of a chip, new variations are injected due to various aging mechanisms, particularly Bias Temperature Instability (BTI). These effects cause long-term degradations in transistor performance, resulting in temporal delay degradations at the circuit level. The mechanism of BTI is exacerbated as transistor sizes reduce, and poses a growing threat to circuit reliability.

All of these effects poses significant challenges at the presilicon design stage, which must ensure correct and reliable performance of a chip throughout its lifetime. Hence, techniques to mitigate the effects of spatial and temporal variations have become a vital part of the design flow for digital circuits and architectures. In this thesis, we develop robust techniques, in the form of design tools and techniques that operate at the circuit and architectural levels, which can be used to analyze, compensate and mitigate various sources of variation, including process and temperature variations and BTI-induced aging.

One significant problem is related to the issue of performing presilicon timing analysis. State-of-the-art timing tools are built around the use of current source models (CSMs), which have proven to be fast and accurate in enabling the analysis of large circuits. As circuits become increasingly exposed to process and temperature variations, there is a strong need to augment these models to account for thermal effects and for the impact of adaptive body biasing, a compensatory technique that is used to overcome on-chip variations. However, a straightforward extension of CSMs to incorporate timing analysis at multiple body biases and temperatures results in unreasonably large characterization tables for each cell. The first contribution of this thesis is to propose a new approach to compactly capture body bias and temperature effects within a mainstream CSM framework. Our approach features a table reduction method for compaction of

tables and a fast and novel waveform sensitivity method for timing evaluation under any body bias and temperature condition.

The next part of the thesis addresses the problem of designing a new form of logic circuit, known as a variable latency unit. The basic idea, proposed in prior research, is an alternative to the conventional one-cycle implementation of combinational circuits. Variable latency units (VLUs) allow a combinational circuit to complete its operation in either one or multiple (typically, two) clock cycles, depending on the input provided to the circuit. This is facilitated through the use of hold logic, which holds the clock for an extra cycle when certain input patterns are applied. Our second contribution develops VLU-based BTI-aware designs, with a novel scheme for multioutput hold logic implementation for VLUs. A key observation is the identification and exploitation of specific supersetting patterns in the two-dimensional space of frequency and aging of the circuit. The multioutput hold logic scheme is used in conjunction with an adaptive body bias framework to achieve high performance.

VLUs may experience functional incorrectness due to process variations. In our third contribution, we develop an efficient, combined presilicon-postsilicon statistical technique for variation aware VLU design. We develop a set of hold logics that ensure functional correctness of the circuit across all manufactured chips. This is achieved by exploiting spatial correlations to cluster such paths in the circuit, that get affected by process variations in very similar ways. Since such clusters are quite few in number, the corresponding set of hold logics is also small.

Our final contribution presents a novel scheme for saving architectural power by mitigating BTI in digital circuits, inspired by the notion of human circadian rhythms. The method works in two alternating phases. In the first, the compute phase, the circuit is "awake" and active, operating briskly at a greater-than-nominal supply voltage, which causes tasks to complete more quickly. In the second, the idle phase, the circuit is power-gated and "put to sleep," enabling BTI recovery. Since the wakeful stage works at an elevated supply voltage, it results in greater aging than operation at the nominal supply voltage, but the sleep state involves a recovery that more than compensates for this differential. At about the same performance, this approach results in appreciable BTI mitigation.

Contents

List of Tables

List of Figures

Chapter 1

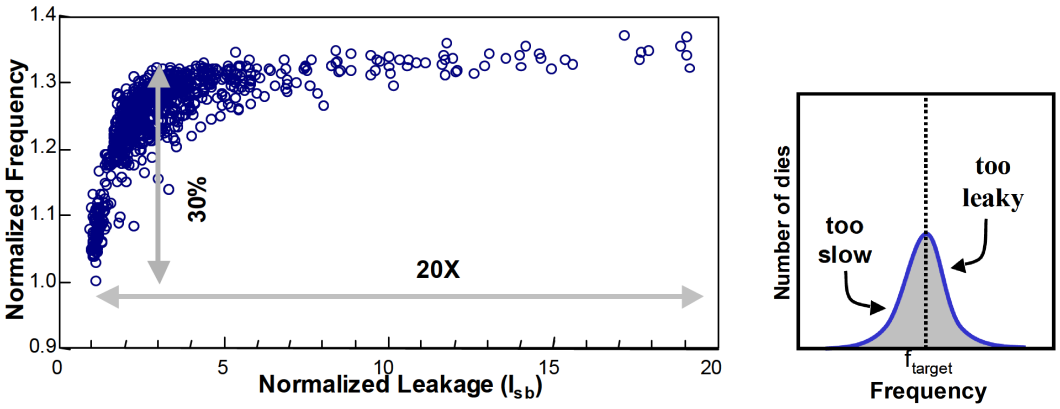
Introduction

1.1 Variability in Nanometer CMOS Technology

The demand for systems with high performance has increased tremendously, driving the shift of CMOS technology towards more deeply scaled nanometer feature sizes. While greater on-chip device integration within the same chip area offers higher computing capabilities, feature size scaling also results in a concomitant reduction in the likelihood that the fabricated chip, at the postsilicon stage, meets the specifications developed in the design flow before fabrication, at the presilicon stage.

This deviation is primarily attributable to the effects of variations (process, environmental, and aging), which have grown larger with shrinking feature sizes. This poses a significant challenge to achieve a simultaneous closure on the threefold metrics of performance: reliable computing, high throughput, and low power.

As an illustration of the impact of process parameter variations, Fig. 1.1(a) shows the effect of process variability with shrinking feature sizes on the frequency and leakage of fabricated chips [2]. The distribution of frequency along the y-axis is plotted against normalized standby leakage along the x-axis for about 1000 dies (microprocessors) fabricated with 0.18μm technology. Due to variation in transistor parameters, chip leakage and frequency suffer with about 20x and 30% variation from the minimum to the maximum, respectively. This trend is further illustrated through Fig. 1.1(b), which shows the deviation of chip frequency of all the dies from the targeted frequency specification. Some of the dies are too slow, and others, despite being faster, are too leaky. With



(a) (b)

Figure 1.1: (a) Variability in frequency and leakage in about 1000 fabricated dies, and (b) deviation in the frequency of dies from the targeted frequency specification, courtesy Intel [2].

more deeply scaled technology nodes, such variations are becoming more prominent, as depicted in Fig. 1.2, which shows that the variability in performance in the sub-65nm regime is much higher than at the 0.18μm technology node [3].

Variations in environmental factors such as temperature can affect the timing critical properties such as delay and slew. Figure 1.3(a) shows the floorplan of IBM Power4 server [4], with its thermal profile in Fig. 1.3(b). The thermal profile shows that across the small chip, the temperature gradient can be appreciably high, and can cause numerous undesirable effects in the processor [10], such as increased leakage and clock skew. This in turn adversely affects the chip performance, due to which a significant fraction of the total number of acceptable dies may fail to achieve the prescribed performance goals.

Similarly, aging variations in circuits due to bias temperature instability (BTI), hot carrier injection (HCI), and time dependent dielectric breakdown (TDDB), cause the circuit delay to degrade over time. For example, Fig. 1.4 shows the temporal variation in delay due to BTI of an MCNC benchmark des, with time along the x-axis and delay in picoseconds along the y-axis, for a period of 10 years. For a 32nm Predictive Technology Model (PTM) [11] based design, the degradation in this delay is about 24% for des, and

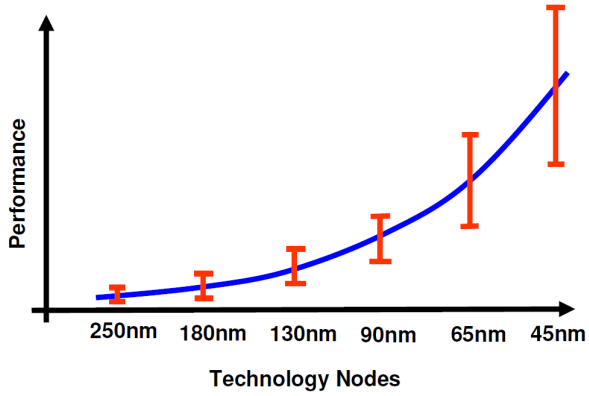


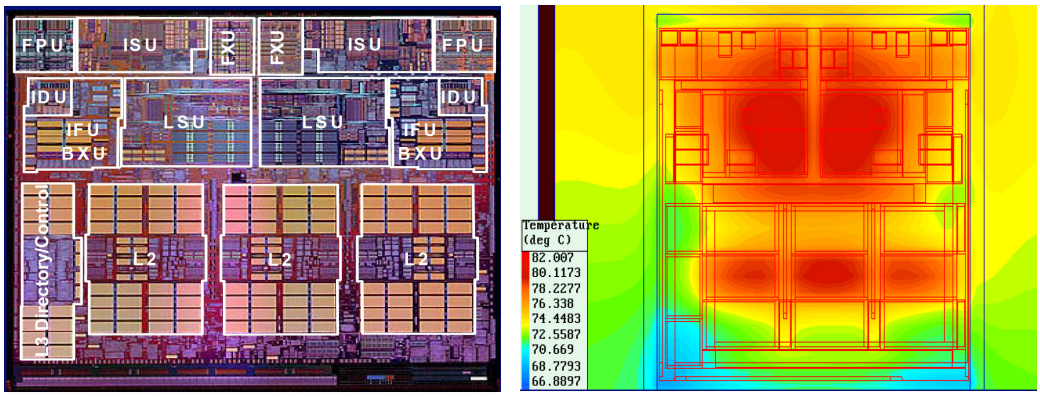
Figure 1.2: Increase in variations due to shrinking feature sizes, courtesy IBM [3].

causes functional failures early in lifetime (by violating the clock period, T_{clk}) without adequate delay guardbands. The effects of aging, as with those of process variations, are more significant with more deeply scaled CMOS technology nodes.

1.2 Motivation and Goal for the Thesis

The combined effect of all such variations can potentially cause large deviations from the desired behavior of the chip. Increased variations have therefore led to a shift in the design paradigms both at the presilicon and postsilicon stages, in order to cope with the impact of such variations on delay, slew, power, etc.

At the presilicon stage, the design flow calls for new techniques for achieving higher-performance. From Fig. 1.1(b), it can be inferred that many of the dies in postsilicon stage may have to be pessimistically operated with a slower clock to preserve the yield and functional correctness across all the dies (there are some postsilicon ways to handle this, as discussed next). Similarly, from Fig. 1.4, we can infer that a delay guardband, allowing the design to meet the T_{clk} specification *throughout its lifetime*, may necessitate the circuit to function well below T_{clk} for a significant portion of the lifetime. Leveraging on the ideas from new high-performance designs, one can nonetheless design the chip at the presilicon stage to gain high performance even in the presence of variations and aging (we discuss such high-performance paradigms in more detail in Chapter 5).



(a) (b)

Figure 1.3: (a) IBM Power 4 chip floorplan, and (b) chip thermal profile, courtesy IBM [4].

At the postsilicon stage, various techniques have been employed to reduce the spread shown in Fig. 1.1(b) or to counter the effects of aging. This includes the use of adaptive body bias (ABB), adaptive supply voltage (ASV) [2, 12], and postsilicon tuning using a small number of on-chip sensors [13]. Such techniques require multiple, fast timing precharacterizations, creating the need for developing fast and accurate timing tools.

Our goal in this work is to develop fast and accurate timing tools, that can be leveraged upon for delay and slew analysis for mitigating process variations and aging in circuits and architectures. We further aim to develop novel techniques, for reliable, high-throughput, and low-power operation in circuits and architectures, that can mitigate and/or compensate for such variations and aging-related delay changes.

1.3 Our Contributions

With this goal, we summarize the contributions made through our work from the circuit to the architectural levels. The details will be expanded upon in the chapters to follow.

1. As highlighted earlier, application of a suitable ABB/ASV to a circuit or a die, at a particular temperature, requires faster and accurate timing characterizations. Conventional tools, that use HSPICE [14], consume a large amount of time to work with moderate to large designs.

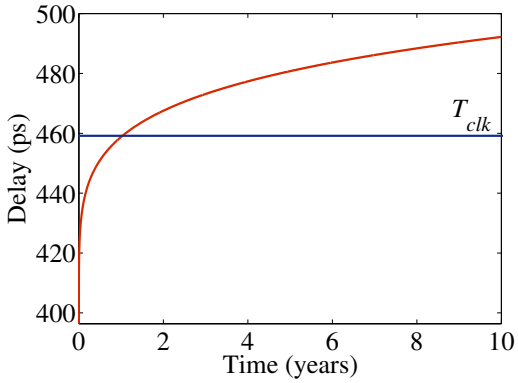


Figure 1.4: Temporal delay degradation of MCNC benchmark des, due to BTI, violating the T_{clk} specification early in its lifetime.

We therefore first look at the state-of-the-art timing tools, based on current source models [15, 16, 17]. Such models have proven to be much fast and accurate as compared to HSPICE, but have been limited to circuits and architectures that do not consider process and temperature variability. We augment such tools with the capabilities of being able to compactly and accurately model the effects of body bias and temperature variations in circuits.

2. Timing tools developed above are typically used by conventional circuits and architectures, which work with the assumption that the T_{clk} is constrained by the worst-case delay of designs in a chip.

   However, recent architectures have been increasingly working with and incorporating the idea of "better-than-worst-case" design [18, 19], in which the T_{clk} is chosen by the average-case (or better-than-worst-case) delay, rather than the worst-case delay. This results in higher throughput at the cost of low area overhead. Variable latency units are one such design paradigm.

   We extend the idea of variable latency design in our second contribution for achieving BTI resilience in digital circuits. We develop a suitable partition of the circuit paths into either one-cycle or two-cycle paths (allowing for variable latency of operation), thus preserving the functional correctness of the circuit against BTI degradation throughout lifetime. In such a scheme, the partition of the paths is

also adaptively changed with time for maximizing throughput throughout lifetime.

3. Process variations in manufacturing of chips leads to a change in the delay distribution of the circuit. In such a scenario, the design of variable latency units may be rendered incorrect, as the identification of one-cycle or two-cycle paths should also change with the change in delay distribution.

   Our second contribution lies in the statistical-delay based design of variation-aware VLUs, wherein process variations are incorporated in the design flow. We develop novel algorithms for generation of clusters in a circuit which we then be leveraged upon for efficient optimizations to reduce area and power overhead.

4. BTI resilience, as achieved in our contributions, allows for functional correctness of the circuit throughout its lifetime. If the amount of BTI degradation that the circuit or the architecture undergoes in its lifetime, can be reduced (mitigated), the area and power overhead associated with BTI resilience can also be less.

   Our final contribution thus, is a novel, counterintuitive scheme for saving architectural power by mitigating delay degradations in digital circuits due to BTI. Working at a greater-than-nominal V_{dd} as the supply voltage, our scheme employs the notion of circadian rhythms in humans to gain circuit recovery from BTI degradation and hence, save area and power overhead required for BTI resilience.

1.4 Organization of the Thesis

The organization of this thesis is as follows. In Chapter 2, we present the background needed for the topics dealt within the thesis. We then present the details of each of our contributions described above, in Chapters 3, 5, 4 and 6, respectively. Within each of these chapters, we first cover some of the relevant previous contributions of the specific areas, then our approach and the results. Finally in Chapter 7, we present our conclusions.

Chapter 2

Background

We discussed some of the variations and their effects on performance parameters in Chapter 1. In this chapter, we elaborate on a subset of the types of these variations dealt with in this thesis.

In the discussion to follow, in the context of a circuit or a chip, we use the terms "gates" and "devices" interchangeably, to refer to the inverters, NANDs, NORs, etc., that constitute the circuit. On the other hand, we use the term "gate" and "gate terminal" in the context of a PMOS/NMOS to refer to the gate terminal of the PMOS/NMOS, attached to the polysilicon.

2.1 Process and Temperature Variations in Circuits

2.1.1 Process Variations

Process variations refer to the variations in the values of CMOS device parameters, such as the transistor width (W), effective channel length (L_{eff}), oxide thicknesses (t_{ox}), dopant concentrations (N_a), interlayer dielectric thickness (t_{ILD}) and the interconnect width and height (h_{int} and w_{int}) [20, 21, 22, 23, 24]. These variations occur during the process of fabrication, due to phenomena such as proximity effects in photolithography, nonuniform conditions during deposition, random dopant fluctuations, nonuniform layout density due to chemical mechanical polishing, thickness variations due to nonuniform resist coating, and depth variations due to uneven etching. Since all such process parameters impact the performance-related metrics such as the delay,

slew, and power of the CMOS device, a fluctuation in the values of these parameters also causes performance variations in circuits and architectures, as was illustrated through Figs. 1.1 and 1.2.

Variations in process parameter values can be classified in terms of different spatial dimensions:

1. **Die-to-Die (D2D):** D2D variations are those that cause the values of the device parameters to vary in the same way throughout the chip. For example, two ring oscillators (ROs) in a particular chip may exhibit the same degree of variation in their oscillation frequency, but this may differ for two ROs in two different chips.

2. **Within-Die (WID):** WID variations affect the values of the device parameters to vary differently in the same chip. An inverter in an ALU, for instance, may have different device parameter values compared to an inverter in a sense amplifier of the on-chip L1 cache.

Process variations can also be classified in terms of their manifestations, such as *systematic* variations and *random* variations [23, 24]. Typically, such variations from one device to another tend to be correlated; devices that are located close to each other tend to vary in same manner than devices that might be spatially distant. We elaborate more on the correlation between devices in Section 2.1.3.

Process variations may result in significant degradation in the timing yield of the chips upon manufacturing. Various techniques have therefore been adopted to enhance the yield despite such variations. Robust gate sizing techniques [25, 26, 27, 28] aim to guardband the delay against violating the timing specification, by sizing the gates in the whole circuit or along the critical paths. Techniques are also developed for addressing the binning-yield loss [29, 30], in which the chips are divided into bins based on their timing properties. Such techniques are based on presilicon statistical timing analysis.

Techniques for mitigating variations based on postsilicon optimizations have also been proposed. Adaptive supply voltage schemes [31, 32] tune the supply voltage in each chip to meet the timing specifications. Application of body bias has been extensively used for such mitigation [20, 21, 22, 33, 34, 35]. Body bias voltage alters the threshold voltage of the transistors, and hence the delay of the devices on the chip. Typically, adaptive body bias (ABB) is applied at coarse levels of granularity, e.g., by

biasing individual n-wells and/or p-wells, each of which contains a number of transistors. Forward body bias (FBB) effectively reduces the transistor threshold voltage and speeds up the device, at the cost of increased leakage, while reverse body bias (RBB) achieves the opposite effect on speed and leakage. ABB involves the use of FBB or RBB to help dies recover from variations, and may be applied dynamically to tighten the distribution of the dies with maximum operational frequency, while simultaneously meeting the leakage power constraints.

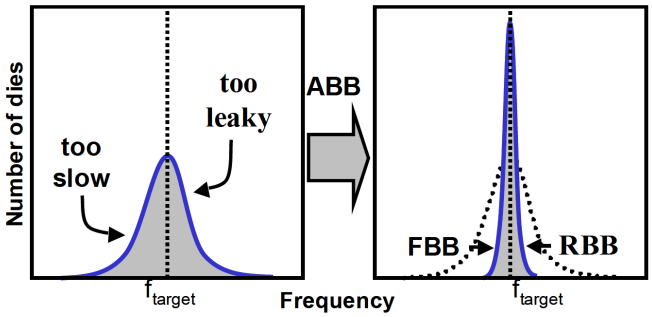


Figure 2.1: Adaptive body bias usage for mitigating variations in performance, courtesy Intel [2].

An illustration of this optimization is shown in Fig. 2.1, where RBB and FBB are used to tighten the frequency spread over all dies obtained after fabrication [2]. This allows for a higher number of chips to work close to the design specifications. Notice that this optimization requires timing precharacterizations at numerous body bias points. This forms the motivation for our exploration of fast timing tools in Chapter 3, where we demonstrate our results using typical logical gates such as inverters, NANDs, etc., for 45nm PTM technology.

2.1.2 Temperature Variations

On-chip temperature variations occur due to power dissipation in the form of heat; an example of such variations was shown in Fig. 1.3. Such thermal variations have a significant bearing on the mobilities of electrons and holes, as well as the threshold voltage of the devices. Traditionally, it had been observed that timing varies monotonically

over the temperature range, but this is no longer the case with thermally-driven variations [36]. In nanometer-scale technologies, elevated temperatures cause reductions in device mobilities (which tend to increase the delay) as well as reductions in threshold voltages (which tend to decrease the delay). The interplay between these effects may cause the circuit delay to increase monotonically (negative temperature dependence, or NTD), decrease monotonically (positive temperature dependence, or PTD), or vary nonmonotonically (mixed temperature dependence, or MTD) with temperature.

An an example, Fig. 2.2 shows the variation of delay for three ITC99 benchmarks [5]. Delay variation for circuits b21_1_opt, b21_opt, and b22_1_opt, shows PTD, NTD, and MTD effects, respectively. In the last case, as is evident from the delay plot, the worst case may occur in the interior of the temperature range, rather than at its edges.

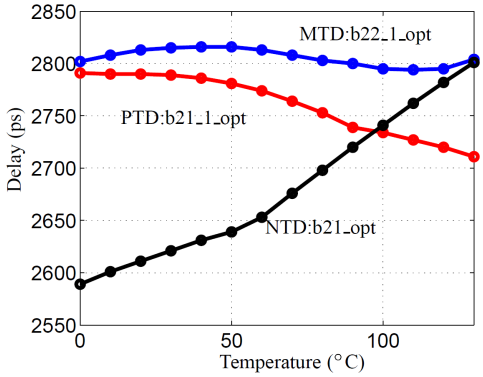


Figure 2.2: Positive, negative, and mixed temperature dependence for three ITC99 benchmarks synthesized with PTM 45nm technology [5].

Increase in temperature has undesirable effects on power and aging too. Increased on-chip temperatures cause higher leakage, which in turn aids the increase in temperature, leading to thermal runaway. Hence, extra amounts of padding overhead and cooling mechanisms have to be added in order to protect the chip from exceeding the maximum power limit and burning out. Aging variations (discussed shortly) are also exacerbated by increased on-chip temperatures.

2.1.3 Statistical Timing Analysis

Traditionally, the delay of a CMOS device in a chip has been represented as a fixed, deterministic quantity (for a fixed input slope and output load) in the design flow. Further, the delay of a circuit could be easily determined by performing a static timing analysis (STA) over all the gates in the circuit.

Due to process variations, such an assumption no longer holds true since this deterministic number for the same CMOS device can differ for different chips, or for different locations in the same chip. Corner-based timing analysis is an attempt to overcome this problem, wherein the design is simulated at a set of "corners," with each corner chosen in a way that it can capture the worst-case variations in delay due to parametric variations. This technique, apart from being computationally intensive, works only for D2D variations, but not for WID variations, which have become increasingly significant at the nanometer scale.

Statistical static timing analysis (SSTA) [6, 37, 38, 39, 40] has therefore been proposed as an alternative to these traditional methods. In SSTA, the delay of a device is modeled as a statistical distribution (more specifically, as a probability distribution function, termed as delay pdf) across all the chips, rather than as a fixed deterministic value. Using this representation for delay for each of the gates in a circuit, a path-based (block-based) SSTA engine then iterates through the paths (gates), beginning with the primary inputs (PIs), to generate the pdf of the arrival time at the primary outputs (POs) of the circuit (i.e., the maximum delay from the PIs to each of the POs). The maximum over the arrival times of all the POs gives the circuit delay or the worst-case delay.

This statistical delay pdf representation can be best illustrated through an example in Fig. 2.3 for ISCAS89 benchmark circuit s38417. This figure shows the probability of the circuit delay (along the y-axis) to be a particular value in picoseconds (along the x-axis), for various chips. Considering any point (d_i, p_i) on this pdf, we have the following relationship:

$$p_i = \frac{\text{Number of chips with circuit delay} = d_i}{\text{Total number of chips fabricated}} \tag{2.1}$$

In this simulation, the total number of chip samples was chosen to be 10,000. From

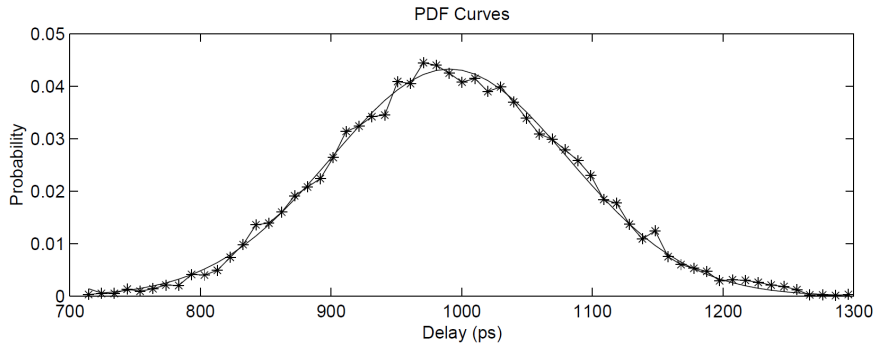


Figure 2.3: Circuit delay probability distribution for s38417 over 10000 samples [6]. The curve marked by the solid line is the result of a statistical timing analyzer, *MinnSSTA* [7], and the curve marked with stars shows the result of Monte Carlo.

the figure, the point (960ps, 0.04) implies that 400 dies had the s38417 circuit delay to be 960ps. The designers may use either the whole, or just the 3σ point, of the pdf in the design flow.

Typically, such delay pdfs can be obtained through various block-based SSTA engines [6, 39], which have proven to quite fast and accurate as compared to Monte Carlo simulations (as shown in Fig. 2.3). In our thesis, we adopt the *MinnSSTA* engine [7], the formulation and algorithms for which are described in [6].

MinnSSTA is a block-based SSTA engine that computes the statistical arrival time for each of the gates in a circuit, including the POs. The circuit delay (or the worst-case delay), as explained earlier, is then obtained by taking the maximum over the statistical arrival times of all the POs. *MinnSSTA* accounts for the spatial correlations (apart from structural correlations) of parameters for delay calculations, by a grid-based correlation model [6]. Spatial correlations, as introduced earlier, refer to the fact that the for devices and wires that are spatially closely located, the electrical parameters will tend to vary in a more similar manner, as compared to those devices and wires that are spatially distant. Depending upon the distance between the devices, the correlation between them value between may vary from 0 to 1. Neglecting spatial correlations has been observed to cause appreciable errors in the results from SSTA engines [6].

An example of this notion is illustrated in Fig. 2.4 using the grid representation of

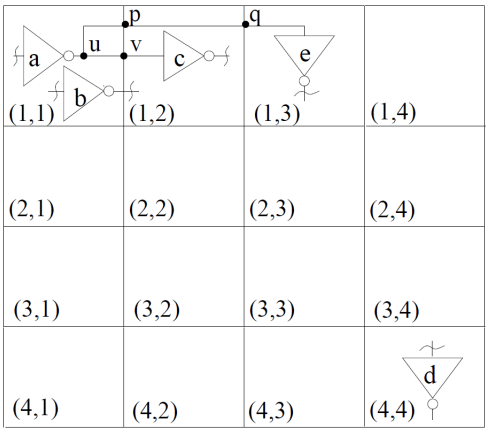


Figure 2.4: Grid-based representation of a die for capturing spatial correlations in devices [6].

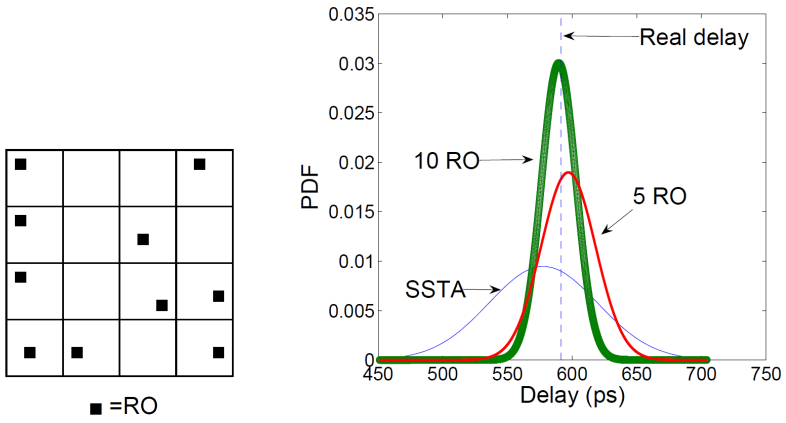
a die. If the die area is divided into a grid of $n_{row} \times n_{col} = n$ grid cells, the devices that lie in the same grid cell (devices a and b) or the adjacent grid cells (devices c and e) will have a high correlation, whereas those in spatially distant grid cells (devices a and d) may have very low or zero correlation. Note that correlation exists in the variation of same process parameters; there is no correlation between different process parameters .

Typically, variations in process parameters are modeled as Gaussian distributions with the nominal values of these parameters being the mean of the Gaussian. Computing Gaussian-distributed correlations for each parameter, for all the devices in the n-cell grid, is computationally intensive, and an alternative technique based on principle component analysis (PCA) was proposed in [6] to reduce these computations. The resulting SSTA methodology is much faster and accurate as compared to Monte Carlo results.

2.1.4 Fast Postsilicon Delay Estimation

Having obtained a statistical distribution of the circuit delay *over all the dies*, at the presilicon stage, the next step is to determine the precise value (or a close estimation) for this delay *in a particular die*, in the postsilicon stage (this delay, after fabrication in a particular die is termed as the *real delay*). This is useful since the designer may need to use the value of the real circuit delay for various applications.

Since postsilicon full-chip testing is a prohibitive solution, a methodology seated between SSTA and full-chip testing is adopted for this purpose uses measurements from a small number of on-chip sensors to estimate the degree of variations at different spatial locations of the chip [8], and combines it with the results of SSTA to narrow the spread (variance) of the delay pdf. This results in a much closer estimate of the real delay in the fabricated die. We use this technique in Chapter 5.



(a) (b)

Figure 2.5: (a) Placement of sensors in the grid, and (b) estimation of real delay in a particular die, by combining presilicon SSTA results with data from postsilicon sensor measurements [8].

The idea is illustrated in Fig. 2.5(a), showing the same grid as in Fig. 2.4, but now with k sensors located on different parts of the die, marked with a "■", with their delays stored in a delay vector $\mathbf{d_t} = [d_1, d_2, ..., d_k]$ (these delays can be either computed by SSTA in the presilicon stage, or measured through suitable circuitry in the postsilicon stage). These sensors can be as small as a three-inverter ring oscillator (RO).

The spatial region, where the RO sensor is located, is characterized by a high correlation between the variations in the sensor delay, and those in the delay of the gates that are located in the same/neighboring grid cells. A measurement of the shift in delay (from the mean value) of the sensor can therefore give a close approximation of the shift in delay of these gates too.

This fact is exploited in [8] to develop algorithms for narrowing the delay pdf obtained from SSTA. The result of the application of this algorithm, on one such circuit in a die, is shown in Fig. 2.5(b), which shows that with $k = 5$ sensors, the variance of circuit delay pdf is reduced. With $k = 10$ ROs, the estimation accuracy is further increased, and the delay pdf mean is very close to the real circuit delay. The higher the value of k, the closer is the estimation of the real delay. Note that such sensors consume a negligible area on the chip, as they are small enough to be inserted in the space allocated for buffers, decaps, etc.

2.2 Aging Variations

Aging variations refer to the temporal variations in delay of the underlying devices and circuits in a chip. The primary on-chip aging mechanisms, that affect CMOS devices, can be classified into several categories, outlined below. We present only a brief description of each of these, and elaborate more on the phenomena of BTI in Chapters 4 and 6 of this thesis, since it is considered to be the primary source of aging induced delays shifts in digital circuits.

1. **Bias Temperature Instability (BTI)**: Negative/positive bias temperature instability (NBTI/PBTI) in PMOS/NMOS devices are collectively referred to as BTI [41, 42, 43]. In a CMOS device, when a PMOS (NMOS) is stressed under BTI, typically by applying a logic 0 (logic 1) at its gate input, its threshold voltage degrades, resulting in an increase in the delay of the device. When the stress is removed, there is partial (but not complete) recovery in the threshold voltage, and hence the delay degradation is also partially ameliorated.

2. **Hot Carrier Injection (HCI)**: When a carrier (electron/hole) in the channel of a CMOS device gains sufficient kinetic energy (becomes hot), it can get injected into the gate dielectric and cause interface traps to be generated [41, 44]. This results in a V_{th} shift of the device, and hence, degradation of delay and other switching properties over time.

3. **Time Dependent Dielectric Breakdown (TDDB)**: Oxide breakdown [45, 46] refers to the creation of a current path from the gate to the channel, and this

happens due to the generation of defects in SiO_2 gate oxide, when stress is applied over a long period of time. When the defect density reaches a critical value, a conductive path is formed through the oxide, resulting in a functional failure of the device.

2.2.1 Bias Temperature Instability

Bias temperature instability refers to the instability created in the bonding structure at the substrate-oxide interface of the CMOS device, when a bias (stress) is applied to the gate terminal with respect to the source terminal. In PMOS, the application of a zero/negative V_G voltage creates a negative bias at the gate terminal with respect to the source, as shown in Fig. 2.6. When a logic 1 is applied, the relative bias between the gate and source terminals becomes zero. The application of negative bias, either for a long period of time or in conjunction with alternating zero biases, can cause the PMOS to age with time, and suffer from delay degradation.

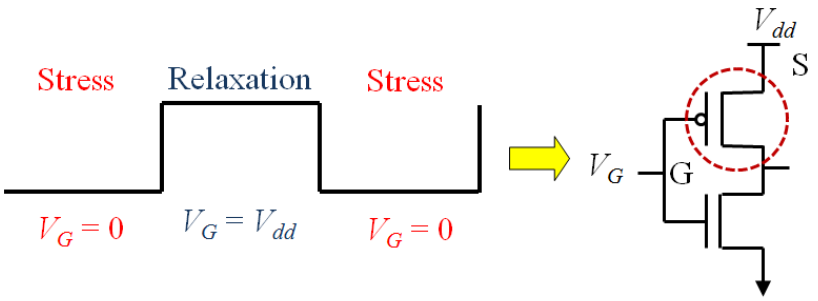
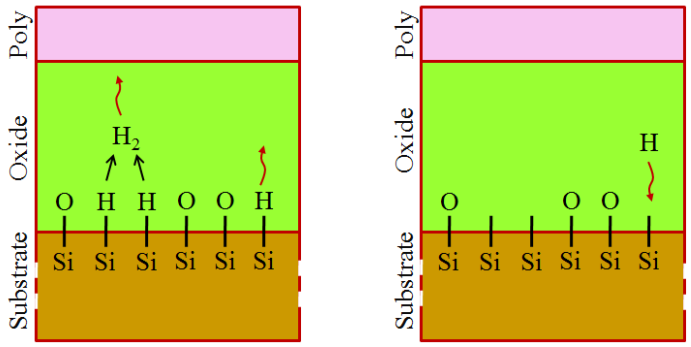


Figure 2.6: Application of a digital signal to the gate terminal of the CMOS causes stresses and relaxations for the PMOS device at $V_G = 0$ and $V_G = 1$, respectively. For NMOS, similar stress-relaxations patterns follow for $V_G = 1$ and $V_G = 0$, respectively.

Aging due to NBTI under negative stress can be understood by examining the underlying mechanism. An analogous mechanism also operates for NMOS when a positive bias is applied, causing PBTI. NBTI aging occurs when the Si-SiO_2 interface is affected by the application of a negative bias. This interface is characterized by the presence of both silicon-oxygen (Si-O) bonds and silicon-hydrogen (Si-H) bonds, as depicted in Fig. 2.7, which shows the 2-D schematic of a Si-SiO_2 interface. Under the reaction

diffusion model [42], the effects of changes in electric field at this interface, created by application of negative bias and its removal, can be captured in the two phases:



(a) Stress (b) Relax

Figure 2.7: Schematic showing BTI mechanism and the Si-SiO_2 interface: (a) the diffusion of hydrogen into the oxide when some of the Si-H bonds are broken during the stress phase, and (b) recovery of some of the Si-H bonds during the relaxation phase.

1. **The stress phase**: The Si-H bonds are much weaker than the Si-O bonds, and thus tend to be very easily disassociated when negative bias, or stress, is applied at the gate terminal. This results in the formation of interface traps, depicted by the incomplete Si- bonds. The disassociated hydrogen then diffuses into the oxide, as shown in Fig. 2.7(a). The rate of generation of these interface traps is exacerbated at higher operating voltages and temperatures (thus the term temperature in BTI).

   The increase in the number of interface traps, N_{it}, is directly related to the change in the threshold voltage, ΔV_{th}, of the transistor by the following relationship:

$$\Delta V_{th} = Sgn\ \frac{(m+1)qN_{it}}{C_{ox}} \tag{2.2}$$

   where Sgn is -1 for PMOS and 1 for NMOS, m is a measure of the additional V_{th} degradation caused due to mobility degradation [47], q is the charge on the carrier and C_{ox} is the oxide capacitance. Since delay $\propto V_{th}$, an increase in the value of N_{it} also causes the delay of the PMOS device to increase.

2. **The relaxation phase**: When the bias is removed from the gate terminal, some of the hydrogen atoms that had diffused into the oxide in the stress phase, diffuse back to the interface and recombine with the incomplete Si- bond at the trap sites, as shown in Fig.2.7(b). This lowers the value of N_{it}, and from Equation (2.2), the V_{th} of the device, improving its delay.

A detailed description of the physics and modeling of the hydrogen diffusion mechanism in the stress and relax phases can be found in [42, 48, 49].

The effect of aging with BTI can be captured as the amount of V_{th} shift for a particular device, occurring over time. This causes the delay of the device to degrade. Depending on the type of stress applied, the models can be classified in the following ways.

2.2.2 BTI Modeling: Constant Stress Model

Assuming a constant stress on the gate terminal of the device, for both NBTI and PBTI, the effect of aging on the threshold voltage degradation for a stressed device increases as:

$$\Delta V_{th}(t) \propto t^n \tag{2.3}$$

where a typical value of n is in the range of 0.1 to 0.2. In this work, we will use $n = 0.16$.

This model serves as a quick upper bound on the delay degradation that occurs when relaxations are also applied, as discussed next. This model is useful when a precise knowledge of the input pattern distributions for each of the specific devices in the circuit is not available. One way to get this knowledge is through signal probability information; however, there are two drawbacks to this. First, such information may not be available. Second, it may predict the average behavior over all users and programs for a system, but the actual aging depends on the behavior of a specific user, which may not be predictable. Another potential approach is through the use of on-chip sensors as described in Section 2.1.4, but these are of limited utility in the context of aging, since they do not experience the same signal patterns as the circuits whose aging is to be measured.

2.2.3 BTI Modeling: Stress-Relaxation Model

When the knowledge of applied relaxations is known and incorporated in BTI modeling, the overall degradation is relatively less than that predicted by Equation (2.3). We work with a widely adopted model [50] for predicting this delay degradation. We present an expression for PMOS NBTI under alternate stress/relax cycles, for a given temperature, V_{dd} and signal probability α at the input of the PMOS (for PBTI, similar equations may be used since the mechanism of NMOS delay degradation is similar to that of PMOS, albeit with a lower degradation magnitude [12]):

$$\textit{Stress: } \Delta V_{th}(t) = \left(K_v\sqrt{t-t_0} + \sqrt[2n]{C(t-t_0)}\right)^{2n} \tag{2.4}$$

$$\textit{Recovery: } \Delta V_{th}(t) = \Delta V_{th}(t_0)\left(1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(t-t_0)}}{2t_{ox} + \sqrt{Ct}}\right) \tag{2.5}$$

where Equations (2.4) and (2.5) model the all-stress and all-recovery modes. This is extended to build a long-term model that predicts the envelope of the BTI degradation pattern with alternating stress and recovery:

Long-term model:

$$\Delta V_{th}(t) = \frac{(K_v^2 \alpha T_{clk})^n}{(1-\beta_t^{1/2n})^{2n}};\ \beta_t = 1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(1-\alpha)T_{clk}}}{2t_{ox} + \sqrt{Ct}} \tag{2.6}$$

The precise definitions of the symbols above may be found in [50], but it is important to note that the exponent $n = 1/6$, and that K_v (and hence $\Delta V_{th}(t)$) is a superlinear function of V_{dd}.

2.2.4 Effect of BTI on Circuits

Since the underlying gates/devices in the circuits undergo repeated stress and relaxations when a chip is operated, the circuit delay also degrades with time. Further, apart from delay, leakage power also changes with V_{th} due to its exponential relationship with V_{th}.

In order to capture the delay degradation and changes in leakage power in circuits considered in this thesis, as in past research, we use compact sensitivity-based performance models for the delay (D) and the logarithm of the leakage power ($\log L$) in terms

of V_{th} [12]. For $\mathcal{X} \in \{D, \log L\}$, we characterize:

$$\mathcal{X}(t) = \mathcal{X}_0 + \sum_{i=1}^{n} \frac{\partial \mathcal{X}}{\partial V_{th_i}} \Delta V_{th_i}(t) \tag{2.7}$$

where $\partial \mathcal{X} / \partial V_{th_i}$ denotes the sensitivity of the quantity $\mathcal{X}$ with respect to the V_{th} of the i^{th} transistor along the input-output path.

The overall flow of determining the degradation in the circuit delay is as follows:

1. At the device level, for each gate in the circuit, we first perform a computation to determine the $\Delta V_{th}(t)$ for each of the PMOS/NMOS connected to the input terminals of the gate. This can be done by using either of the models described by Equations (2.3) or (2.6), depending upon the application. For this computation, the α values used in Equation (2.6), at each input terminal of the gate, are computed by propagating the signal probabilities from the PIs of the circuit up to the respective input terminals of the gate.

2. We then use these values of $\Delta V_{th}(t)$ in Equation (2.7), to determine the degraded delay of each of the gates in the circuit at time t.

3. Following this, a static timing analysis of the circuit can be performed over all its gates to determine its degraded delay at time t.

The final result is a ΔD vs. t curve for the circuit, beginning at time $t = 0$. A representative curve was shown in Fig. 1.4 for MCNC benchmark des.

Chapter 3

Variation-Aware Current Source Models

3.1 Variability-Aware Timing

We begin our work on variability and reliability with a set of timing tools that are highly useful for timing analysis in variation-aware designs: the current source models.

3.1.1 Current Source Models: An Overview

Timing properties of library cells, such as slew and delay, have typically been obtained by development of mathematical models of the the underlying physical parameters, such as currents, threshold voltages, mobility, etc., in the MOS transistors. Timing solvers such as HSPICE [14] use these mathematical models to solve for various port voltages and branch currents of a circuit when an input is provided to the circuit. Generally, it may require for the solver to solve for a large number of equations for each of the MOS transistors, and this may result in high computational complexity and consequently large runtimes for the solver to evaluate the slew and delay of the cells.

Current source models (CSMs), which have proven to be much faster and accurate as compared to HSPICE, are gate-level black-box abstractions of cells in a library, such as inverters, NANDs, etc. CSMs have the same input and output ports as the original cell. This abstraction can be illustrated through Fig. 3.1, which shows the output of the

NAND2 cell being modeled by a nonlinear current source with a nonlinear capacitance in parallel. The values of this current source and capacitance depend on all the port voltages, and can be characterized at various values of port voltages by DC and AC simulations. Such models can then be used by timing solvers for predicting the slew and delay of a particular cell in the circuit, with much higher speed than HSPICE.

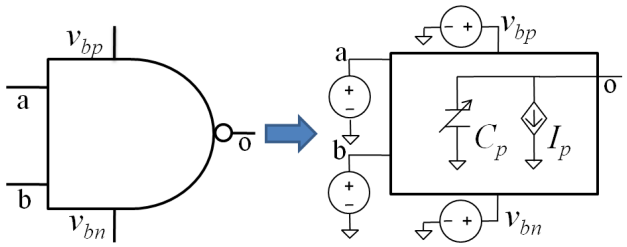


Figure 3.1: Example of a CSM: the output port is modeled as a nonlinear VCCS dependent on all input port voltages, in parallel with a nonlinear capacitance.

A CSM approach termed as "Blade" [15] represents the cell as a voltage-controlled current source (VCCS) with an internal capacitance and a time-shifted input waveform driving an arbitrary load. A lookup table, indexed by the input voltage, V_{in}, and the output voltage, V_{out}, models the VCCS current, I_{out}. These ideas were further refined in [51, 52], which introduce extra resistances and capacitances in the model to capture certain nonlinearities. All such models consider single input switching at the input of these gates.

CSMs which also consider multiple input switching and stack effects [16, 17, 53, 54] were therefore proposed to overcome this limitation. The essential difference compared to earlier models was to model not just the output terminal, but every terminal of the cell, as a VCCS in parallel with a nonlinear capacitance. Since this increases the complexity of characterization of these models, such techniques also propose various ways of reducing such complexity and maintaining the speedups and accuracy.

Other CSM approaches tend to use different modeling techniques compared to above models, although still treating the cell as a black box. CSM proposed in [55] uses orthogonal functions for delay prediction. Another CSM based on the small-signal model of a transistor was built in [56], which used parallel simulations on multicores to achieve speedups.

3.1.2 CSMs for Variability-Aware Designs

A key enabler for variation-tolerant design is the ability to simulate the timing behavior of a circuit during the design process using static timing analysis (STA). Traditional standard cell modeling approaches represent the delay and output slew as nonlinear functions of the input slew and output load capacitance [57]. When interconnect resistance became significant, these methods were replaced by the notion of effective capacitance [58]. However, this approach models the input as a saturated ramp with piecewise constant slope, and was further enhanced by the development of CSMs, that provide fast and accurate timing estimates.

Within the CSM framework, process variations are commonly captured through the use of process corners. Traditionally, temperature variations were also handled using corner-based methods, but this is no longer viable. Corner-based approaches are predicated on the idea that the timing varies monotonically over the temperature range, but this is no longer the case, as highlighted in Section 2.1.2. The worst case, therefore, may occur in the interior of the temperature range, rather than at its edges. As a result, a set of temperature corners is no longer adequate, and circuit delays must be simulated as functions of temperature.

Therefore, a first necessary enhancement of CSMs involves extending them to determine the cell delay as a function of temperature. This capability is useful not only for circuit analysis but also for building optimization techniques that compensate for temperature variations [59, 22, 34, 10, 60].

A second way in which CSMs require augmentation is in building an ability to simulate cell timing in the presence of body biases. The application of adaptive body biases (ABBs) allows circuits to be made resilient and variation-tolerant by applying a deliberate bias to the body terminals of transistors in a circuit. Realistically, ABB is applied at coarse levels of granularity, e.g., by biasing individual n-wells and/or p-wells, each of which contains a number of transistors. Forward body bias (FBB) effectively reduces the transistor threshold voltage and speeds up the device, at the cost of increased leakage, while reverse body bias (RBB) achieves the opposite effect on speed and leakage. ABB involves the use of FBB or RBB to help dies recover from variations, and may be applied dynamically to tighten the distribution of the dies with maximum operational frequency, while simultaneously meeting the leakage power constraints [20, 21, 33, 22,

34].

Traditional CSMs simulate the circuit at fixed values of the body bias ($v_{bp} = v_{bn} = 0$) and at fixed values of the temperature. The obvious extensions to existing CSMs that enable them to capture body biases and temperature effects are rather inefficient. In principle, the body terminal of a device can be considered to be another port, and the cell can be accordingly characterized by creating a look-up table for various combinations of body biases, v_{bp}, v_{bn}. Further, such lookup tables would have to be constructed for various temperature values. However, this increases the amount of memory used as well as the characterization time significantly over the zero body bias and the nominal temperature case. For instance, for 10 values each of v_{bp} and v_{bn}, and for 10 values of temperature, the table for each library cell becomes 1000× larger. The need to access a larger lookup table may also result in a significant concomitant increase in the simulation runtime of CSM macromodels.

3.1.3 Our Contributions

This work develops efficient timing characterization methods for building CSMs that incorporate changes in the body bias and the temperature. Since ABB is applied at the granularity of a well, we assume that all PMOS transistors in a cell have the same body bias value, v_{bp}, and all NMOS devices are biased at v_{bn}. Further we assume that all transistors in the cell experience a uniform temperature: this is reasonable, since the rate of decay of temperature with respect to the distance from the cell has a "time constant" that is significantly larger than the size of a cell.

Our framework for incorporating effects of body bias and temperature into the CSM has a very small memory and runtime overhead, while maintaining high levels of accuracy. Our mathematical framework consists of two key steps. First, we intelligently adapt an existing scheme to enable the compaction of look-up tables for the sensitivities of CSM components to body bias and temperature, over the range of allowable values of both the applied body bias and on-chip temperature. Our second key contribution is to develop a novel waveform sensitivity model for evaluating the impact of the applied body bias and variations in temperature, which provides accurate waveforms at the output of the cell under any body bias or temperature condition, with minimal computation. The

essential idea of this approach is that since body bias or temperature variation constitutes a small perturbation to the nominal waveform, it should be possible to determine the perturbed waveform cheaply by determining and saving the parameters that compute its shift from the nominal waveform. We develop a scheme for characterizing this perturbation and computing it efficiently. Specifically, mathematical models for such parameters are developed and further analyzed for their independence over body bias and temperature variations for an efficient computation of such parameters.

The remainder of this chapter is organized as follows. Section 3.2 presents the development of sensitivity models for CSM components to handle variations in body bias and temperature. Section 3.3 presents our algorithms for compacting the CSM sensitivity tables. Then we present the conventional macromodel solvers used in state-of-the-art CSMs in Section 3.4, and is followed by a description of our method for fast output waveform evaluation in Section 3.5. Section 3.6 presents experimental results on a set of library cells in a 45nm technology.

3.2 CSM Sensitivity Model Development

Our CSM structure, shown in Fig. 3.1, is of the type proposed in [15], and is augmented to model nonlinearities as in [16]. Specifically, output port p is replaced by a nonlinear voltage-controlled-current-source (VCCS), I_p, in parallel with a nonlinear capacitance, C_p. The VCCS model enables the CSM to be load-independent, and permits it to handle an arbitrary electrical waveform at its inputs. The CSM is characterized in terms of the value of I_p and the charge, Q_p, stored on the capacitor, C_p. The variables, I_p and Q_p, are functions of all input and output port voltages and temperature, and are determined by characterizing the cell at various port voltages, body bias combinations and temperatures as follows:

$$I_p = F(V_i, V_o, v_{bp}, v_{bn}, \Delta T) \tag{3.1}$$

$$Q_p = G(V_i, V_o, v_{bp}, v_{bn}, \Delta T) \tag{3.2}$$

The parameters I_p and Q_p are modeled using the functions F and G, respectively, and V_i and V_o are, respectively, the voltages at the transitioning input and output ports of the cell. We use the term ΔT to represent the temperature offset from a baseline

temperature value, taken here to be room temperature (25°C). In the temperature range of [−25°C, 125°C] that we work in, the range for the values of ΔT is [−50°C, 100°C].

For a cell, I_p characterization involves DC simulations over multiple combinations of DC values of (V_i, V_o), while Q_p is characterized through a set of transient simulations [16]. The presentation of our model is targeted to the more widely-used scenario of single-input switching for gates with a single output, though the idea can easily be extended to multiple input switching (MIS) and multioutput gates, leveraging current work on CSMs on these topics [16, 53, 54].

As mentioned earlier, in order to capture the sensitivity of CSM parameters to the applied body bias and temperature offset, in principle, the circuit could be characterized over a set S of all possible $(v_{bp}, v_{bn}, \Delta T)$ points, treating body terminals as input ports, and temperature offset as the independent variable. Since the allowable values of the applied body biases and temperature offset change in discrete steps, the cardinality of this set is large, and the corresponding characterization would be computationally intensive, even as a precharacterization step that is to be performed once for a technology. Moreover, memory requirements of the table multiply significantly over the current characterization procedure at zero body bias and zero temperature offset.

We observe through simulations that the functions F and G depend *much more* weakly on v_{bp}, v_{bn}, and ΔT as compared to V_i or V_o. Hence, a simpler model can be utilized to save on this computation. We thus develop sensitivity models of CSM with respect to v_{bp}, v_{bn} and ΔT (as we will soon show, the models with respect to body bias and temperature are independent), and then present a scheme to incorporate the effects of the two.

3.2.1 Independence of Body Bias and Temperature Effects

Next, we explain the rationale for analyzing the effects of body bias and temperature independently. Body bias (a change in the substrate bias voltage, V_{BS}) changes the threshold voltage, V_{th}. The sensitivity of V_{th} with respect to V_{BS} can be captured from following equation [61]:

$$\frac{\partial V_{th}}{\partial V_{BS}} = \frac{C_{dep}}{C_{ox}} \tag{3.3}$$

where C_{dep} is the depletion capacitance of the MOS transistor and C_{ox} is the oxide capacitance. C_{dep} is a very weak function of temperature, being proportional to the inverse square root of the built-in potential. Similarly, it is observed that the expression for V_{th} sensitivity with respect to temperature is independent of V_{BS} [61]. Hence, the effects of changes in body bias and temperature on MOS transistors can be treated as independent. Since I_p and Q_p essentially abstract the internal cell behavior, the effects of body bias and changes in temperature on I_p and Q_p can also be assumed to be independent. This is further verified by the model formulations and accuracy results as presented in the subsequent subsections and sections.

3.2.2 CSM Body Bias Sensitivity Model

We first present the body bias sensitivity model, which is independent of the changes in temperature and constructed at $\Delta T = 0°\text{C}$ (i.e., at room temperature). We construct a polynomial approximation for the variations of I_p and Q_p with respect to (v_{bp}, v_{bn}). Our simulations show that a linear approximation yields an average of 2.0% relative error with respect to HSPICE, evaluated over all (v_{bp}, v_{bn}) points, for all (V_i, V_o) points. The CSM is now modified by using the equations:

$$I_p(V_i, V_o, v_{bp}, v_{bn}, 0) = I_p^Z \cdot (1 + a_I(V_i, V_o)v_{bp} + b_I(V_i, V_o)v_{bn}) \tag{3.4}$$

$$Q_p(V_i, V_o, v_{bp}, v_{bn}, 0) = Q_p^Z \cdot (1 + a_Q(V_i, V_o)v_{bp} + b_Q(V_i, V_o)v_{bn}) \tag{3.5}$$

where $I_p^Z = F(V_i, V_o, 0, 0, 0)$, $Q_p^Z = G(V_i, V_o, 0, 0, 0)$, and $\{a_I,\ b_I,\ a_Q,\ b_Q\}$ correspond to the sensitivity of the function to the corresponding body bias. These parameters are characterized at a discrete set of (V_i, V_o) values and are saved in a lookup table.

The characterization of I_p and Q_p using Equations (3.4) and (3.5) can now be carried out using a minimum of three simulations at each (V_i, V_o), since it is a linear model; however, additional redundancy is preferable to account for the small nonlinearities, and a linear least squares fit can be used instead.

For notational simplicity, we will define the following functions:

$$L_I(v_{bp}, v_{bn}) = \ 1 + a_I(V_i, V_o)v_{bp} + b_I(V_i, V_o)v_{bn} \tag{3.6}$$

$$L_Q(v_{bp}, v_{bn}) = \ 1 + a_Q(V_i, V_o)v_{bp} + b_Q(V_i, V_o)v_{bn} \tag{3.7}$$

Clearly,

$$I_p(V_i, V_o, v_{bp}, v_{bn}, 0) = I_p^Z L_I(v_{bp}, v_{bn}) \tag{3.8}$$

$$Q_p(V_i, V_o, v_{bp}, v_{bn}, 0) = Q_p^Z L_Q(v_{bp}, v_{bn}) \tag{3.9}$$

3.2.3 CSM Temperature Sensitivity Model

We now construct the temperature sensitivity model at zero body bias. We observe that the variations of I_p and Q_p with ΔT are nonlinear, unlike the body bias case where a linear approximation was adequate. We employ a second-order polynomial approximation, and find that the fit has an average relative error of 1.6% relative error in comparison with HSPICE simulations. The CSM for the temperature sensitivity model with the first and second order sensitivities in temperature offset is now represented by the following modified equations:

$$I_p(V_i, V_o, 0, 0, \Delta T) = I_p^Z \cdot (1 + c_I(V_i, V_o)\Delta T + r_I(V_i, V_o)\Delta T^2) \tag{3.10}$$

$$Q_p(V_i, V_o, 0, 0, \Delta T) = Q_p^Z \cdot (1 + c_Q(V_i, V_o)\Delta T + r_Q(V_i, V_o)\Delta T^2) \tag{3.11}$$

where I_p^Z, Q_p^Z are as defined above, and $\{c_I, c_q, r_I, r_Q\}$ correspond to the sensitivity of the function to the corresponding powers of the temperature offset, ΔT. As in the case of $\{a_I, b_I, a_Q, b_Q\}$, these parameters are characterized at a discrete set of (V_i, V_o) values and saved in a lookup table. Since the temperature sensitivity model is a second order model, we need at least three points to determine the values of $\{c_I, r_I, c_Q, r_Q\}$.

As before, for notational simplicity, we will define the following functions:

$$S_I(\Delta T) = 1 + c_I(V_i, V_o)\Delta T + r_I(V_i, V_o)\Delta T^2 \tag{3.12}$$

$$S_Q(\Delta T) = 1 + c_Q(V_i, V_o)\Delta T + r_Q(V_i, V_o)\Delta T^2 \tag{3.13}$$

Clearly,

$$I_p(V_i, V_o, 0, 0, \Delta T) = I_p^Z S_I(\Delta T) \tag{3.14}$$

$$Q_p(V_i, V_o, 0, 0, \Delta T) = Q_p^Z S_Q(\Delta T) \tag{3.15}$$

3.2.4 CSM Complete Sensitivity Model

The complete body bias and temperature sensitivity model can now be formulated by integrating the models of I_p and Q_p with body bias from Equations (3.8), (3.9), and with

temperature offset from Equations (3.14), (3.15). The complete model is constructed as follows:

$$I_p(V_i, V_o, v_{bp}, v_{bn}, \Delta T) = I_p^Z \cdot L_I(v_{bp}, v_{bn}) \cdot S_I(\Delta T) \tag{3.16}$$

$$Q_p(V_i, V_o, v_{bp}, v_{bn}, \Delta T) = Q_p^Z \cdot L_Q(v_{bp}, v_{bn}) \cdot S_Q(\Delta T) \tag{3.17}$$

Simulations show that the above model yields approximations with an average of 2.9% relative error with respect to HSPICE. This also justifies our assumption that the effects of body bias and changes in temperature on CSM components can be analyzed independently.

3.3 Compact CSM Formulation

As described in Sections 3.1 and 3.2, the lookup tables obtained for $\{a_I, b_I, a_Q, b_Q\}$ and $\{c_I, r_I, c_Q, r_Q\}$ reduce the excessive memory requirements described in Section 3.1. However, we still need a separate lookup table (indexed by (V_i, V_o)) for each parameter of every cell in the library. If we can further reduce the size of these tables by suitably compacting them, we can gain more in terms of memory overheads induced. We thus present the development of a compact lookup table scheme used for reducing the size of such lookup tables.

3.3.1 Table Size Reduction for Conventional CSMs

As a preliminary step, we attempt to apply the method in [9] to create compact lookup tables for I_p and Q_p for the zero body bias and nominal temperature case, i.e., I_p^Z and Q_p^Z, with controlled loss of accuracy. For general values of the body bias and temperature, we must also create lookup tables for $\{a_I, b_I, a_Q, b_Q\}$ and $\{c_I, r_I, c_Q, r_Q\}$ at each value of (V_i, V_o): as we will see, for these parameters, a direct extension of the method in [9] does not yield satisfactory results.

We first overview the procedure in [9]. This method begins with an $n \times n$ table of characterized points, indexed by variables x and y in the horizontal and vertical directions, respectively. The idea behind table size reduction is to keep a subset of all these points and to interpolate the rest. For instance, consider the rectangle bounded by points $(x_1, y_1), (x_2, y_1), (x_2, y_2)$, and (x_1, y_2): a point (x, y) within this rectangle can be

dropped if the interpolation error in its value, using these points lies within a specified bound.

Instead of an expensive enumeration, the work in [9] presents a dynamic programming method for reducing a two-dimensional $n \times n$ table. The objective of the algorithm is to create a smaller $m \times m$ table, where m is prespecified, while minimizing the total error corresponding to the points that are dropped from the table. The procedure begins by constructing an initial 2×2 table corresponding to the points at the four corners of the table. Next, this table is expanded to include additional entries using the idea of h-hops.

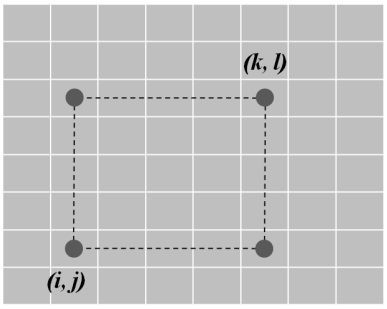
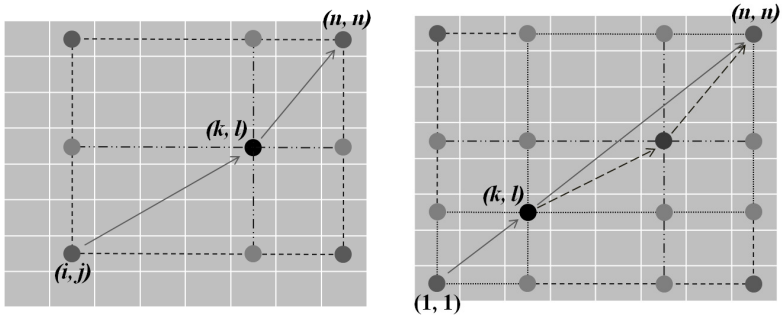


Figure 3.2: The initial step, considering all rectangles from any point (i, j), extending to any point (k, l) at the northeast corner.



(a) (b)

Figure 3.3: (a) A 1-hop solution from (i, j) to (n, n), through an intermediate point, (k, l). (b) A 2-hop solution from $(1, 1)$ to (n, n) through an intermediate point, (k, l) uses a previously computed optimal 1-hop solution from (k, l) to (n, n).

In the initial step, we consider all rectangles originating at a point (i, j) at the southwest corner, extending to any point (k, l) at the northeast corner, as shown in Fig. 3.2. We compute the error metric over the rectangle, corresponding to the case where only the points at the four corners of the rectangle are kept in the lookup table, and all internal points are dropped. The error metric is the sum of the interpolation errors for all points within and on the perimeter of the rectangle. Each such rectangle corresponds to an optimal substructure for dynamic programming: the optimal solution will be composed from some (but not all) such substructures.

Next, we define a 1-hop operation. We optimize the region bounded by point (i, j) to the southwest and (n, n) to the northeast by finding an optimal point (k, l) within this region. Here, optimality is defined as follows: the point (k, l) divides the region into four subregions, as shown in Fig. 3.3(a), and over all candidate (k, l) points, the optimal point minimizes the total error summed up over these four subregions. Since the error over each rectangle was calculated in the initial step, this step involves enumerating all candidate (k, l) points, and summing up the previously calculated error over the rectangles in constant time for each such point. We refer to this as a 1-hop, indicating that for each (i, j), the table "hops" over a single point, corresponding to the optimal (k, l), on the way to (n, n). The associated optimal error encountered is the 1-hop error for (i, j).

In general, an h-hop from (i, j) to (n, n) finds a point (k, l) such that the error from (i, j) to (k, l), plus the $(h - 1)$-hop error from (k, l) to (n, n), is minimized over all candidate points (k, l). To obtain an $m \times m$ table, the procedure stops after $m-1$ hops, and the optimal m-hop from $(1, 1)$ to (n, n) provides the compact table. Fig. 3.3(b) shows an example of a 2-hop solution from $P(1, 1)$ to $P(n, n)$; if the algorithm were to stop here, it would result in a 4×4 compacted table. The computational complexity of this algorithm is $O(m \cdot n^4)$, but as n is typically small ($n = 30$ in our simulations), this remains tractable, as we will show in Section VI-A that the runtimes for this scheme are reasonable.

It should be noted that although this method proceeds along the main diagonal of the table (in the north-east direction), the interpolation error is computed by considering all the four end points of a rectangle (in Fig. 3(a) for instance). Thus, it also considers the interpolation error induced along the other diagonal, and the rows and columns of

the table as well. While this method is not exact (for example, for an h-hop, it does not entertain the possibility of an $(h-1)$ hop to (k,l) and then a 1-hop to (n,n)), in practice it is seen to work well. A faster version of the algorithm, which trades off accuracy for speed, is also proposed in [9].

3.3.2 Modifications for Sensitivity Tables

As stated earlier, the above approach works well for characterizing I_p and Q_p, where neighboring entries have similar magnitudes. However, in case of the sensitivity parameters, $\{a_I,\ b_I,\ a_Q, b_Q\}$ and $\{c_I,\ r_I,\ c_Q, r_Q\}$, there can be large differences in the values of neighboring parameters. This is illustrated in Fig. 3.4(a) and (b), which show, respectively, the values of $a_Q = \partial Q_p/\partial v_{bp}$ and $c_I = \partial I_p/\partial \Delta T$ for an inverter cell[1] . Large "outliers" (i.e., values of large magnitude) are clearly visible on the plot.

The presence of these outliers is attributed to the nature of variation of the values of I_p^Z and Q_p^Z with (V_i, V_o), and the way these values are derived in the CSM. At a particular (V_i, V_o) bias point, it is quite possible that only a small current flows inside the input and output terminals of the cell. Since the magnitudes of these inflowing currents decide the values of I_p^Z and Q_p^Z, the values of resultant I_p^Z and Q_p^Z are also small. Hence, any change relative to this small value becomes large and is reflected as a large sensitivity value.

In principle, since these I_p^Z and Q_p^Z values are small, we may consider setting the corresponding sensitivities to zero. This, however, has been observed to create inaccuracies in waveform evaluations (the waveform evaluation techniques are described in Sections 3.4 and 3.5) for when such changes are multiplied by other quantities with relatively higher magnitudes (temperature offset for instance), the net contribution from these small changes, to the computed values of $V_o(t)$ or of the waveform sensitivities, becomes significant, and hence cannot be neglected.

For such data, it can easily be shown that the approach in [9], which depends on gridding the table in the coordinate directions, is poorly compacted, i.e., the interpolation errors in the reduced table are large. Such errors are demonstrated to be easily visible in the output response, where they appear as "kinks" in the CSM-based waveform that do not exist in the corresponding HSPICE waveform. This happens due to the fact that

[1] Similar behavior is seen for a_I, b_I, b_Q, r_I, c_Q, and r_Q.

an interpolation error caused by the presence of these outliers causes an error in I_p and Q_p values, which causes the solver (described in the next section) to generate errors in output waveforms. A sample waveform with the use of compacted I_p and Q_p tables, as generated by the solver for a rising input ramp is shown in Fig. 3.5. As is seen, due to poor compaction, kinks appear in the evaluated waveform. The incorrect waveform also incurs slew and delay errors.

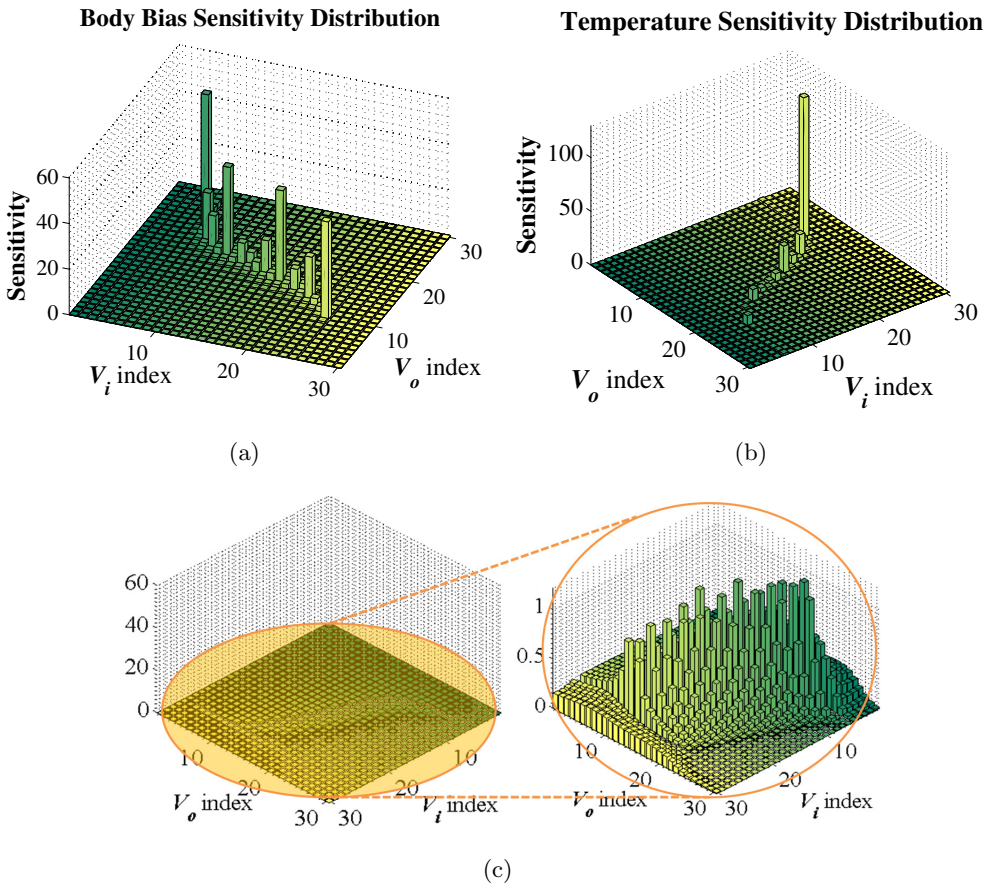


Figure 3.4: The CSM sensitivity parameter distribution for (a) a_Q and (b) c_I as functions of (V_i, V_o). (c) The resultant lookup table for a_Q, when all the outliers have been removed and saved separately in a table.

We propose a simple method for avoiding these problems, based on the observation that for these sensitivity parameters, such outliers are few in number and have relatively

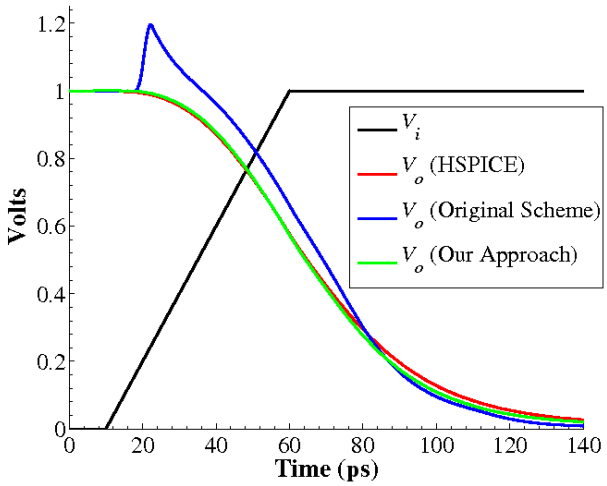


Figure 3.5: The presence of outliers yields poor compaction of the lookup tables when the original scheme from [9] is used. This results in incorrectly evaluated output waveforms with kinks at some time points. Our approach however, with a mechanism for separation of outliers, results in the correctly evaluated output waveform with minimal errors.

large magnitudes. We therefore tabulate and save the outliers separately. As can be observed from Fig. 3.4(a) and (b), the number of outliers is quite small compared to the total number of data points. Thus, a separate tabulation of outliers would incur negligible overhead.

In order to tabulate the outliers separately, given the set of all points, we find the mean and variance over all entries. Any entry that is over k variances from the mean is found to be an outlier; in practice, we find $k = 2$ to be an adequate value. The removed entry at table location (x, y) is then replaced by a dummy point, the error contribution (to the total error) from which is zero. The modified table is then compacted using the algorithm in Section 3.3.1.

When a table entry is requested, we first determine whether the accessed point is an outlier: if so, we fetch it from the outliers list; else, we find it using the compacted look-up table.

With the outliers separated, the variations in remaining lookup table become more uniform. Table 3.1 shows the list of separately tabulated outliers for a lookup table for

a_Q. Further, Fig. 3.4(c) shows the remaining entries for a_Q in the 2-D lookup table indexed by V_i, V_o. As is clearly seen, the removal of outliers make the variation in the lookup table more uniform, allowing for a high compaction using the original algorithm.

Table 3.1: The outlier table for a_Q

V_i index	V_o index	a_Q
10	25	42.6
13	22	18.9
16	16	15.5
.	.	.
.	.	.
22	7	60.7

This method of separating the outliers removes the kinks present in the $V_o(t)$ waveforms. As shown in Fig. 3.5, the smooth waveform obtained from the solver using our approach is no longer characterized by kinks, as compared to the waveform which had kinks due to the errors caused by original compaction scheme. The waveform using our approach further has negligible slew and delay errors.

A potential alternative for dealing with such outliers is to decrease the size of (V_i, V_o) voltages steps at which I_p^Z and Q_p^Z are characterized, making the variation of sensitivities more uniform. We observe that this requires us to increase the value of n by about 6-9$\times$ for different tables, resulting in a large increase in the storage space required. For a small number of outliers, this posed as a significant increase in the memory requirements for a library with different cells. It also prohibitively increases the computational time of the compression algorithm ($\propto n^4$). Therefore, an intermediate approach of saving outliers separately keeps both the storage space and the compression time tractable.

3.4 The Macromodel Solver

Using the approaches described so far, the cell library is characterized to determine the I_p and Q_p characterization tables at zero body bias and zero temperature offset, and the compressed CSM sensitivity parameter tables for the body bias coefficients $\{a_I, b_I, a_Q, b_Q\}$ and the temperature offset coefficients $\{c_I, r_I, c_Q, r_Q\}$.

3.4.1 Using the Macromodel in a Solver

To solve the case of a gate driving an interconnect, including cases that involve coupled lines and crosstalk, it is enough to consider the situation where a gate drives a load described by an RC π-model as shown in Fig. 3.6. Standard techniques such as the O'Brien-Savarino approach [62] are used in our work to reduce an arbitrary interconnect load to a π-model at the driving point. We first obtain the waveform at the driving point node V_o, and then we evaluate the waveform at any sink node in the RC network by solving a linear system using standard model order reduction methods.

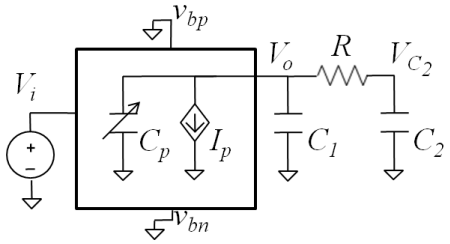


Figure 3.6: A CSM for a gate, under zero body bias and zero temperature offset, driving a π load.

We analyze the case of a gate output driving a π-load in the absence of body bias and at zero temperature offset, as shown in Fig. 3.6. Finding the output voltage waveform involves solving the equation:

$$
\begin{aligned}
I_p^Z + I_{Q_p}^Z &= I_{C_1} + I_{C_2} \qquad (3.18)\\
\text{where } I_{Q_p}^Z &= \frac{dQ_p^Z}{dt}\\
I_{C_1} &= C_1 \frac{dV_o}{dt}\\
I_{C_2} &= C_2 \frac{dV_{C_2}}{dt}\\
I_p^Z &= F(V_i, V_o, 0, 0, 0)\\
Q_p^Z &= G(V_i, V_o, 0, 0, 0)
\end{aligned}
$$

Equation (3.18) is a nonlinear differential equation in $V_o(t)$, and the input voltage, $V_i(t)$, is known. This equation can be solved using routine circuit simulation methods.

We apply the Backward Euler formula to Q_p, V_o and V_{C_2} with a time step h, going from time n to time $n+1$ (the superscript $n+1$ is dropped for notational simplicity) to get:

$$Q_p = Q_p^n + hI_{Q_p} \tag{3.19}$$

$$C_1 V_o = C_1 V_o^n + hI_{C_1} \tag{3.20}$$

$$C_2 V_{C_2} = C_2 V_{C_2}^n + hI_{C_2} \tag{3.21}$$

Moreover, using Ohm's Law, we have $V_{C_2} = V_o - RI_{C_2}$. Substituting V_{C_2} from this in Equation (3.21), we have:

$$I_{C_2} = \frac{C_2(V_o - V_{C_2}^n)}{h + RC_2} \tag{3.22}$$

We then obtain the values of I_{Q_p} from Equation (3.19), of I_{C_1} from Equation (3.20) and of I_{C_2} from Equation (3.22), and substitute them in Equation (3.18) to obtain:

$$I_p + \frac{Q_p - Q_p^n}{h} = \frac{C_1(V_o - V_o^n)}{h} + \frac{C_2(V_o - V_{C_2}^n)}{h + RC_2}$$

Solving this for V_o, we arrive at the following expressions:

$$V_o = \frac{1}{A}\left[hC_2 V_{C_2}^n + B(C_1 V_o^n + hI_p + Q_p - Q_p^n)\right] \tag{3.23}$$

$$\text{where } A = (hC_1 + hC_2 + RC_1C_2) \tag{3.24}$$

$$B = (h + RC_2) \tag{3.25}$$

Obtaining V_o, we substitute I_{C_2} from Equation (3.22) in Equation (3.21) to solve for V_{C_2}:

$$V_{C_2} = \frac{1}{B}\left[hV_o + RC_2 V_{C_2}^n\right] \tag{3.26}$$

Thus we have obtained the expressions for both the unknown port voltages in terms of known quantities. However, such expressions are still implicit, and hence must be solved iteratively.

3.4.2 Newton-Raphson Solver

The approach conventionally employed in CSM solvers is to solve the nonlinear Equation (3.23), through iterative Newton-Raphson linearization. This approach is hereby

termed as the Newton-Raphson Solver, and referred as such in the rest of the sections. In the $(k+1)^{\text{th}}$ iteration, we use the k^{th} iteration value, shown by the additional subscript k, to obtain:

$$AV_o = BC_1V_o^n + hC_2V_{C_2}^n + hB\left(I_{p,k} + \left.\frac{\partial I_p}{\partial V_o}\right|_k (V_o - V_{o,k})\right) + B\left(Q_{p,k} + \left.\frac{\partial Q_p}{\partial V_o}\right|_k (V_o - V_{o,k}) - Q_p^n\right)$$

$$V_o = V_{o,k} - \frac{AV_{o,k} - hC_2V_{C_2}^n - B(C_1V_o^n + hI_{p,k} + Q_{p,k} - Q_p^n)}{A - B(h\ \partial I_p/\partial V_o|_k + \partial Q_p/\partial V_o|_k)} \tag{3.27}$$

This computation is carried out by references to the look-up tables for I_p and Q_p, with the appropriate use of interpolation as necessary, and the use of finite differences to compute derivatives.

3.5 Formulation Of Waveform Sensitivity Model

The Newton-Raphson solver in Section 3.4.2 forms the basis for a procedure for computing the waveform under any body bias and temperature condition using conventional CSM solvers. However, evaluation of the delays and slews of the gates under numerous body bias and temperature offset conditions entails multiple simulations of the *entire* output voltage waveform at each combination of body bias and temperature value. Applications that require timing analysis at multiple body biases and at multiple temperature values include [20, 21, 22, 34, 60].

Intuitively, the repeated computation of full waveforms from scratch seems unnecessarily excessive, for several reasons. First, the application of body bias or a variation in temperature corresponds to a perturbation to a base case, such as the zero body bias and zero temperature offset case, and it should be possible to compute the waveform at nonzero body bias and temperature offset based on the zero body bias and zero temperature offset case, with some consideration of body bias and temperature sensitivities, much more cheaply than the above procedure. Second, as discussed and shown before in Section 3.2, the effects of changes in body bias and temperature on CSM can be decoupled. Thus it should be possible to decouple and independently compute the

effects of body bias and temperature changes on the output waveforms too. Third, in most cases, designers are interested not in the entire waveform, but specific properties of the gate output, such as its delay and output transition time. In this section, we demonstrate the efficient computation of such metrics under changing body bias and temperature without the need for numerous table look-up operations.

3.5.1 Waveform Sensitivity Models

Consider the case when we have the cell maintained at zero temperature offset ($\Delta T = 0°\text{C}$), but with a nonzero applied body bias (v_{bp}, v_{bn}). For various values of (v_{bp}, v_{bn}), the solution of the waveform under the framework of Equation (3.27) entails multiple accesses to the look-up tables for I_p and Q_p. The entries that are accessed in these tables change according to the applied body bias. However, since body bias is a small perturbation, in practice, the accessed entries in each table at each step of the algorithm are relatively close to each other, and can be viewed as perturbations to a nominal case.

Therefore, we propose to capture the output waveform at zero temperature offset for nonzero body bias case as a perturbation to the waveform with zero body bias and zero temperature offset as follows:

$$V_o(t) = V_o^Z(t) + \alpha(v_{bp}, v_{bn}, t) \cdot v_{bp} + \beta(v_{bp}, v_{bn}, t) \cdot v_{bn} \tag{3.28}$$

where $V_o^Z(t)$ represents the output waveform, $V_o(t)$, with zero body bias and zero temperature offset, and $\alpha(v_{bp}, v_{bn}, t)$ and $\beta(v_{bp}, v_{bn}, t)$ are time-varying body bias perturbation parameters that are precisely defined as:

$$\begin{aligned} \alpha(v_{bp}, v_{bn}, t) &= \frac{\partial V_o(t)}{\partial v_{bp}} \\ \beta(v_{bp}, v_{bn}, t) &= \frac{\partial V_o(t)}{\partial v_{bn}} \end{aligned} \tag{3.29}$$

Similarly, if we consider the variation in temperature of the cell, the cell being maintained at zero body bias, we can formulate a linear model as above for capturing the output waveform at any temperature (with a nonzero temperature offset, ΔT) as perturbation to the output waveform at nominal temperature (with zero temperature offset ΔT):

$$V_o(t) = V_o^Z(t) + \sigma(\Delta T, t) \cdot \Delta T \tag{3.30}$$

where $V_o^Z(t)$ is as as described above, and $\sigma(\Delta T, t)$ is time-varying temperature perturbation parameter that is precisely defined as:

$$\sigma(\Delta T, t) = \frac{\partial V_o(t)}{\partial \Delta T} \tag{3.31}$$

The following two results provide a precise formula for $\alpha(v_{bp}, v_{bn}, t)$, $\beta(v_{bp}, v_{bn}, t)$ and $\sigma(\Delta T, t)$. We first present the results (proved in the Appendix), and then discuss how the computational cost of evaluating these quantities can be significantly reduced.

Theorem 1 *The waveform sensitivity parameters from Equation (3.28),* $\alpha(v_{bp}, v_{bn}, t)$ *and* $\beta(v_{bp}, v_{bn}, t)$*, are given by:*

$$\alpha(v_{bp}, v_{bn}, t) = \; N_\alpha / D_{\alpha,\beta} \tag{3.32}$$

$$\beta(v_{bp}, v_{bn}, t) = \; N_\beta / D_{\alpha,\beta} \tag{3.33}$$

$$\begin{aligned} N_\alpha = B\Bigg[& \alpha^n C_1 + h a_I I_p^Z + a_Q Q_p^Z - a_Q^n Q_p^{Z,n} - Q_p^{Z,n} \frac{\partial a_Q^n}{\partial V_o^n} \alpha^n v_{bp} \\ & - Q_p^{Z,n} \frac{\partial b_Q^n}{\partial V_o^n} \alpha^n v_{bn} - \frac{\partial Q_p^{Z,n}}{\partial v_{bp}} L_Q^n(v_{bp}, v_{bn}) \Bigg] + h C_2 \frac{\partial V_{C_2}^n}{\partial v_{bp}}, \end{aligned}$$

$$\begin{aligned} N_\beta = B\Bigg[& \beta^n C_1 + h b_I I_p^Z + b_Q Q_p^Z - b_Q^n Q_p^{Z,n} - Q_p^{Z,n} \frac{\partial a_Q^n}{\partial V_o^n} \beta^n v_{bp} \\ & - Q_p^{Z,n} \frac{\partial b_Q^n}{\partial V_o^n} \beta^n v_{bn} - \frac{\partial Q_p^{Z,n}}{\partial v_{bn}} L_Q^n(v_{bp}, v_{bn}) \Bigg] + h C_2 \frac{\partial V_{C_2}^n}{\partial v_{bn}}, \end{aligned}$$

$$\begin{aligned} D_{\alpha,\beta} = (-B)\Bigg[& h I_p^Z \left(\frac{\partial a_I}{\partial V_o} v_{bp} + \frac{\partial b_I}{\partial V_o} v_{bn} \right) + h \frac{\partial I_p^Z}{\partial V_o} L_I(v_{bp}, v_{bn}) \\ & + Q_p^Z \left(\frac{\partial a_Q}{\partial V_o} v_{bp} + \frac{\partial b_Q}{\partial V_o} v_{bn} \right) + \frac{\partial Q_p^Z}{\partial v_{bp}} L_Q(v_{bp}, v_{bn}) \Bigg] + A \end{aligned}$$

where terms using the superscript n are understood to correspond to their values at the previous (n^{th}) time step, and the superscript Z refers to the case where $v_{bp} = v_{bn} = 0\text{V}$, and $\Delta T = 0°\text{C}$.

Theorem 2 *The waveform temperature sensitivity parameter from Equation (3.30),* $\sigma(\Delta T, t)$*,*

is given by:

$$\sigma(\Delta T, t) = N_\sigma / D_\sigma \tag{3.34}$$

$$\begin{aligned} N_\sigma = B\Bigg[& \sigma^n C_1 + h(c_I + 2r_I\Delta T)I_p^Z + (c_Q + 2r_Q\Delta T)Q_p^Z \\ & - (c_Q^n + 2r_Q^n\Delta T)Q_p^{Z,n} - \frac{\partial Q_p^{Z,n}}{\partial \Delta T} S_Q^n(\Delta T) \\ & - Q_p^{Z,n}\frac{\partial c_Q^n}{\partial V_o^n}\sigma^n\Delta T - Q_p^{Z,n}\frac{\partial r_Q^n}{\partial V_o^n}\sigma^n\Delta T^2\Bigg] + hC_2\frac{\partial V_{C_2}^n}{\partial \Delta T}, \end{aligned}$$

$$\begin{aligned} D_\sigma = A - B\Bigg[& \left(hI_p^Z\frac{\partial c_I}{\partial V_o} + Q_p^Z\frac{\partial c_Q}{\partial V_o}\right)\Delta T + h\frac{\partial I_p^Z}{\partial \Delta T}S_I(\Delta T) \\ & + \left(hI_p^Z\frac{\partial r_I}{\partial V_o} + Q_p^Z\frac{\partial r_Q}{\partial V_o}\right)\Delta T^2 + \frac{\partial Q_p^Z}{\partial \Delta T}S_Q(\Delta T)\Bigg] \end{aligned}$$

where the terms have the notations as described above.

Theorems 1 and 2 enable the efficient computation $V_o(t)$ at any body bias value and temperature offset using a closed form expression, dependent only on the values of V_o at previous time steps and the values in the waveform at zero body bias and nominal temperature. As a result, the waveform at arbitrary body bias and temperature values can be reproduced if the values of $\alpha(t)$, $\beta(t)$ and $\sigma(t)$ are computed.

3.5.2 Simplified Waveform Sensitivity Models

Further simplifications are possible with both the models discussed above. Consider first the body bias model. On investigating dependency of the output waveform on (v_{bp}, v_{bn}) and on $\alpha(v_{bp}, v_{bn}, t), \beta(v_{bp}, v_{bn}, t)$, we observe that:

1. The variation in $V_o(t)$ over (v_{bp}, v_{bn}) is nearly linear at each time point of the waveform. Empirically, this can be seen in Fig. 3.7, which shows typical cases for the variation of $V_o(t)$ over (v_{bp}, v_{bn}) for various time points of simulation. This behavior is observed for multiple test cases, and indicates that $\alpha(v_{bp}, v_{bn}, t), \beta(v_{bp}, v_{bn}, t)$ are actually independent of the applied body bias, and are only dependent on t.

2. Fig. 3.8 shows the variations in $\alpha(v_{bp}, v_{bn}, t)$ and $\beta(v_{bp}, v_{bn}, t)$ with (v_{bp}, v_{bn}). The magnitude of these variations were observed to be a maximum of 0.1 for all test

cases. Since these parameters are further multiplied by v_{bp} or $v_{bn} \in [-0.3\text{V}, 0.3\text{V}]$ in Equation (3.28), their effects on $V_o(t)$ are expected to be negligible. This is further validated in Section 3.6.

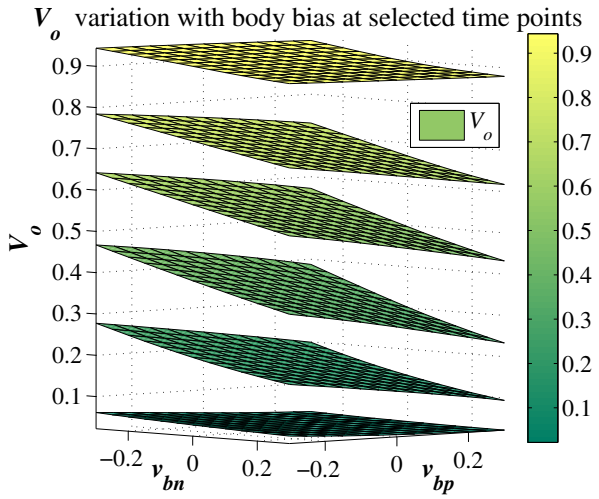


Figure 3.7: Typical surface plots for V_o showing the linear nature of V_o variations with (v_{bp}, v_{bn}), with each surface corresponding to a randomly selected time point during the simulation.

This leads to the following approximation, which provides accurate waveforms with very low errors, as demonstrated in Section 3.6:

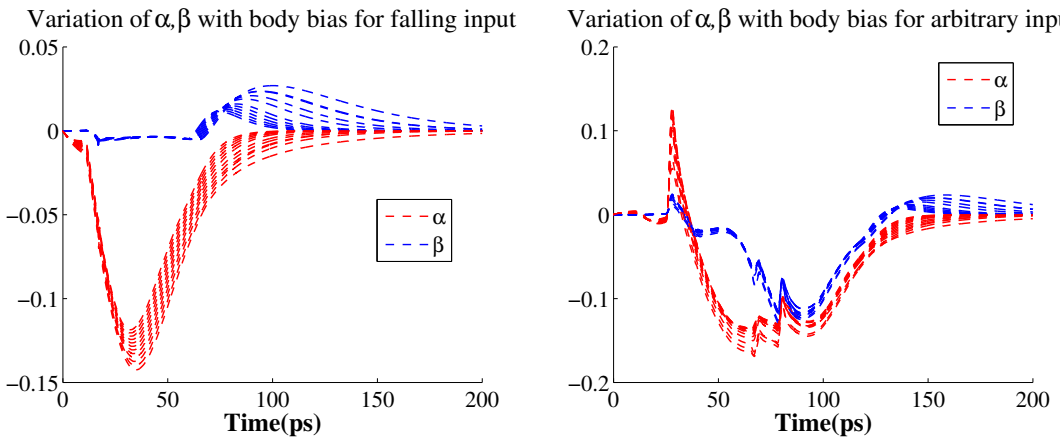
$$\begin{aligned} \alpha(v_{bp}, v_{bn}, t) &\approx \alpha_0(t) = \alpha(v_{bp} = 0, v_{bn} = 0, t) \\ \beta(v_{bp}, v_{bn}, t) &\approx \beta_0(t) = \beta(v_{bp} = 0, v_{bn} = 0, t) \end{aligned} \tag{3.35}$$

The simplified body bias waveform sensitivity model is thus given as follows:

$$V_o(t) = V_o^Z(t) + \alpha_0(t) \cdot v_{bp} + \beta_0(t) \cdot v_{bn} \tag{3.36}$$

Note that this dramatically reduces the storage requirements for the lookup table. At each time point, this method requires just two additional parameters, α_0 and β_0.

In order to develop a simplified model with changes in temperature as was done in the body bias case, we investigated the possibility of being able to generate a simplified



(a) (b)

Figure 3.8: Simulations showing the variation of $\alpha(t)$ and $\beta(t)$ at a range of body biases from the minimum to the maximum, including zero. Two such test cases are shown in Figs. (a) and (b).

temperature waveform sensitivity model too. We however find that unlike the body bias case, the following approximation:

$$\sigma(\Delta T, t) \approx \sigma_0(t) = \sigma(\Delta T = 0, t) \tag{3.37}$$

does not work very well with the temperature waveform sensitivity model. The inaccuracies in the resultant delays and slews, as compared to HSPICE, reach upto 20%. This can be attributed to the nonlinear effects of temperature on the circuit responses, which lead to reduced accuracy when a linear model is used.

Therefore, we apply a more accurate piecewise linear model to address the above inaccuracies. We observe that increasing the value of $|\Delta T|$ increases the inaccuracies in waveform evaluation, and that the magnitude of such errors are not large for smaller values of $|\Delta T|$. Thus, instead of the very simplistic linear approximation as in Equation (3.37), we propose a more accurate and less approximate linear simplified temperature sensitivity model as follows:

$$
\begin{aligned}
&\sigma(\Delta T, t) \approx \sigma_p(\Delta T, t) \\
&= \sigma(\Delta T = \Delta T_1, t),\ \Delta T_{MIN} \leq \Delta T < \Delta T_{MIN} + \left\lceil \frac{\Delta T_R}{3} \right\rceil \\
&= \sigma(\Delta T = \Delta T_2, t),\ \Delta T_{MIN} + \left\lceil \frac{\Delta T_R}{3} \right\rceil \leq \Delta T < \Delta T_{MIN} + \left\lceil \frac{2\Delta T_R}{3} \right\rceil \\
&= \sigma(\Delta T = \Delta T_3, t),\ \Delta T_{MIN} + \left\lceil \frac{2\Delta T_R}{3} \right\rceil \leq \Delta T \leq \Delta T_{MAX}
\end{aligned} \tag{3.38}
$$

where,

ΔT_{MIN} = Minimum value of temperature offset in the range of ΔT

ΔT_{MAX} = Maximum value of temperature offset in the range of ΔT

$$
\begin{aligned}
\Delta T_R &= \Delta T_{MAX} - \Delta T_{MIN} \\
\Delta T_1 &= \Delta T_{MIN} + \left\lceil \frac{\Delta T_R}{6} \right\rceil \\
\Delta T_2 &= \Delta T_{MIN} + \left\lceil \frac{3\Delta T_R}{6} \right\rceil \\
\Delta T_3 &= \Delta T_{MIN} + \left\lceil \frac{5\Delta T_R}{6} \right\rceil
\end{aligned}
$$

As stated in Section 3.2, the values of ΔT_{MAX} and ΔT_{MIN} are taken to be -50°C and 100°C, respectively. The above formulation in Equation (3.38) states that this temperature range is divided into three ranges of nearly equal size. The waveforms of $\sigma_p(\Delta T, t)$, with ΔT chosen as the central value in each of these intervals, are then evaluated and saved. Although in principle, a waveform corresponding to each of the 13 ΔT values can be saved (giving us the lowest error in the model), a designer would like to save and work with minimal number of waveforms, without losing much in accuracy. We have found through simulations that a choice of 3 different waveform (described through Equation (3.38)) serves this purpose. As we will show in Section 3.6, such a choice still preserves the high accuracy. The gain in storage and waveform evaluation speedup on the other hand, is significant.

In other words, instead of computing and saving $\sigma(\Delta T, t)$ at just one temperature point (as in the linear case in Equation (3.37)), we now save the values of $\sigma(\Delta T, t)$ at three distinct values of temperature to provide a better approximation that captures thermal nonlinearities. The simplified temperature waveform sensitivity model is thus

given as follows:

$$V_o(t) = V_o^Z(t) + \sigma_p(\Delta T, t) \cdot \Delta T \tag{3.39}$$

where $\sigma_p(\Delta T, t)$ is given by Equation (3.38). As will be shown in Section 3.6, the above model yields accurate waveforms for all temperature points.

3.5.3 Complete Waveform Sensitivity Model

We now propose the complete body bias and temperature waveform sensitivity model as follows:

$$\begin{aligned} V_o(t) &= V_o^Z(t) \\ &\quad + \alpha_0(t) \cdot v_{bp} + \beta_0(t) \cdot v_{bn} + \sigma_p(\Delta T, t) \cdot \Delta T \end{aligned} \tag{3.40}$$

This model is a linear combination of the simplified waveform sensitivity models as given in Equations (3.36) and (3.39). Note that such a linear combination is possible since the effects of body bias and temperature are independent of each other, as has been discussed in Section 3.2. Equation (3.40) predicts that the effects of perturbations inside a cell caused due to changes in body bias and temperature, can be captured through a simple linear model of the output voltage in terms of the changes in the body bias and temperature.

To summarize, evaluating the output at b body bias points each for v_{bp} and v_{bn}, and at τ temperature offset points, using an enumerative approach would solve for $b^2 \cdot \tau$ waveforms, involving the extensive use of lookup tables. In contrast, our approach reduces the solution to finding just six waveforms: one for the zero body bias, $V_o^Z(t)$, and one each for $\alpha_0(t)$ and $\beta_0(t)$, and three for $\sigma_p(\Delta T, t)$. The net result is a large savings in the storage and computation. Thus, the steps involved in computing the waveform at any (v_{bp}, v_{bn}) and ΔT are summarized below:

1. Apply Equation (3.27) to generate the waveform $V_o^Z(t)$ at zero-body bias and zero temperature offset.
2. Compute and save $\alpha_0(t)$, $\beta_0(t)$ at every timestep from Equations (3.32), (3.33), and (3.35).

3. Compute and save $\sigma_p(\Delta T, t)$ at every timestep from Equations (3.34) and (3.38).

4. Use the computed $\alpha_0(t)$, $\beta_0(t)$ and $\sigma_p(\Delta T, t)$ in Equation (3.40) to directly generate the waveform for any value of (v_{bp}, v_{bn}) and ΔT.

3.6 Experimental Results

Our results are based on standard library cells using the 45nm PTM [11], and our accuracy is measured through comparisons with the results of HSPICE [14] simulations.

3.6.1 Reduction in CSM Sensitivity Table Size

We apply our table reduction algorithm for the sensitivity parameters, $\{a_I, b_I, a_Q, b_Q\}$ and $\{c_I, r_I, c_Q, r_Q\}$ for a set of standard cells characterized using 45nm PTM [11], and demonstrate our results in Table 3.2 for a typical table, for a_I. Columns 2 through 4 show the number of entries in the reduced table using the original compression approach (Section 3.3.1), and Columns 5 through 7 list the size of the reduced tables using our approach (Section 3.3.2). These comparisons are shown for various bounds (2%, 5%, 10%) on the allowable error, and in each case, the optimal table size corresponds to the smallest $m \times m$ table, indexed by (V_i, V_o), that meets the error bound. In each case, $m = 30$ for the original table size, i.e., it has 900 entries. As is seen from the table, in each case, our approach yields much smaller tables than the prior approach.

The last column of Table 3.2 shows the runtime of the algorithm for achieving reduced table sizes for the most computationally-intensive solution, where the 2% error bound must be satisfied. The runtimes are measured on a 3GHz Intel Core2Duo CPU, and correspond to the average for the $\{a_I, a_Q, b_I, b_Q\}$ and $\{c_I, c_Q, r_I, r_Q\}$ sensitivity tables, and are very reasonable, especially considering that this characterization computation must be performed only once for a given library in a given technology.

It is easy to explain why the original algorithm of Section 3.3.1 does not lead to sufficient reduction in the table size. This can primarily be related to outliers: ignoring these points causes substantial errors at these points when interpolation is used to

Table 3.2: Results for sensitivity parameter table reduction for tables with original size = 900

Cell Type	Reduced Table Size with Error Bounds						Run Time
	Original approach			Our approach			
	2%	5%	10%	2%	5%	10%	
INV	529	484	324	225	169	100	115s
NAND2	576	484	289	196	144	81	110s
NOR2	900	784	576	324	256	169	168s
NAND3	625	529	256	169	144	81	104s
NOR3	841	729	484	289	225	144	167s
AOI21	576	529	484	196	169	100	114s
AOI22	529	484	361	225	169	81	117s

predict the values of missing entries. On the other hand, if these are included, the large jumps at these points can result in interpolation errors at nearby points that do not correspond to outliers. These errors can only be diminished by using reduced tables of larger sizes.

3.6.2 Speedup due to Waveform Sensitivity Models

We now present the speedup obtained using our various simplified body bias, temperature and the complete waveform sensitivity (WS) models, as proposed in Sections 3.5.2, and 3.5.3, respectively.

We evaluate the speedup of our models over HSPICE and over the Newton-Raphson solver (see Section 3.4.2) that would be used in a simple extension of existing CSMs. To calculate the above speedup, we perform our tests with each circuit example under multiple combinations of the following parameters: multiple rise/fall waveforms (1ps–100ps input ramps, in steps of 5-10ps), various RC interconnects from the ChipA-1K, ChipB-1K and the ChipB-5K family [63] as load benchmarks reduced to π-models, multiple body bias points (169 points with $(v_{bp}, v_{bn}) \in$ [-0.3V, 0.3V], in steps of 0.05V for each parameter), and multiple temperature points (13 points with $\Delta T \in$ [-50°C, 100°C], in steps of 12.5°C).

First, we present the speedups with simplified body bias waveform sensitivity model and those of simplified temperature waveform sensitivity model independently. Then we present the speedups of the complete waveform sensitivity model. In each case,

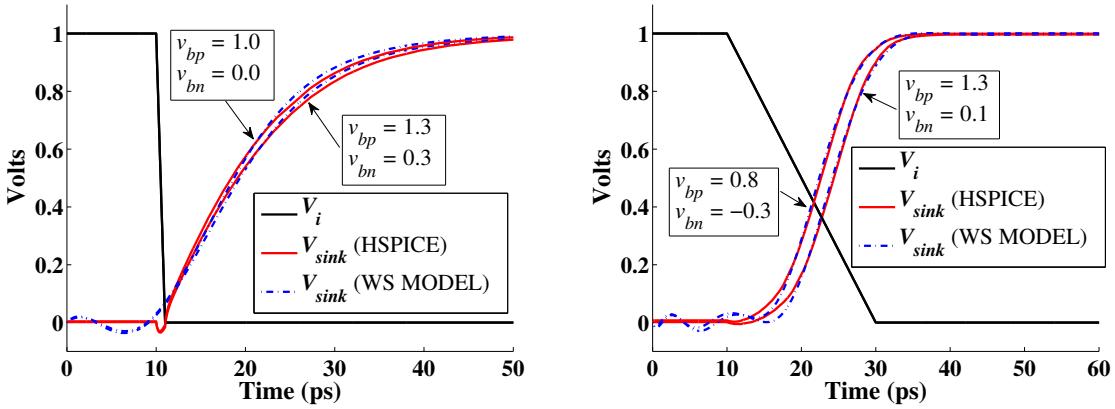
we calculate the runtimes using HSPICE, Newton-Raphson solver and our simplified waveform sensitivity models, and average these runtimes over all the test cases to arrive at final speedup results. For the test cases, we perform transient simulations and report the speedups of our algorithm over HSPICE and over the Newton-Raphson solver. Expectedly, the speedup over HSPICE is large, and is found to be about five orders of magnitude. More interestingly, our complete waveform sensitivity model achieves an average speedup of 91.81×, and a maximum speedup of 99.55×, over the Newton-Raphson solver.

Table 3.3: Speedups obtained by the Complete Waveform Sensitivity (WS) Model over HSPICE and Newton-Raphson (NR) solver

Cell	WS Model Speedups					
	Body Bias		Temperature		Combined	
	Over HSPICE	Over NR Solver	Over HSPICE	Over NR Solver	Over HSPICE	Over NR Solver
INV	8.9e4	65.36	4.6e3	4.374	1.15e5	85.12
NAND2	9.6e4	66.29	4.9e3	4.352	1.28e5	88.15
NOR2	9.2e4	69.23	4.0e3	4.454	1.26e5	95.02
NAND3	9.6e4	66.67	4.8e3	4.313	1.29e5	89.50
NOR3	8.9e4	72.15	4.2e3	4.405	1.23e5	99.55
AOI21	10.8e4	66.80	4.8e3	4.389	1.45e5	89.72
AOI22	10.0e4	69.36	4.9e3	4.413	1.39e5	95.60

Body bias waveform sensitivity model

We evaluate the speedup achieved using the standalone body bias model as presented in Section 3.5.2. We perform evaluations at 169 body bias points within the range $(v_{bp}, v_{bn}) \in$ [-0.3V, 0.3V]. All evaluations are carried at the zero temperature offset of $\Delta T = 0°$C. Table 3.3 lists the speedups that are obtained by our waveform sensitivity model over HSPICE and over the Newton-Raphson Solver, for standard library cells. As can be seen from the table, the body bias waveform sensitivity model achieves an average speedup of around five orders of magnitude over HSPICE and an average speedup of 67.9× over the Newton-Raphson Solver.



(a) (b)

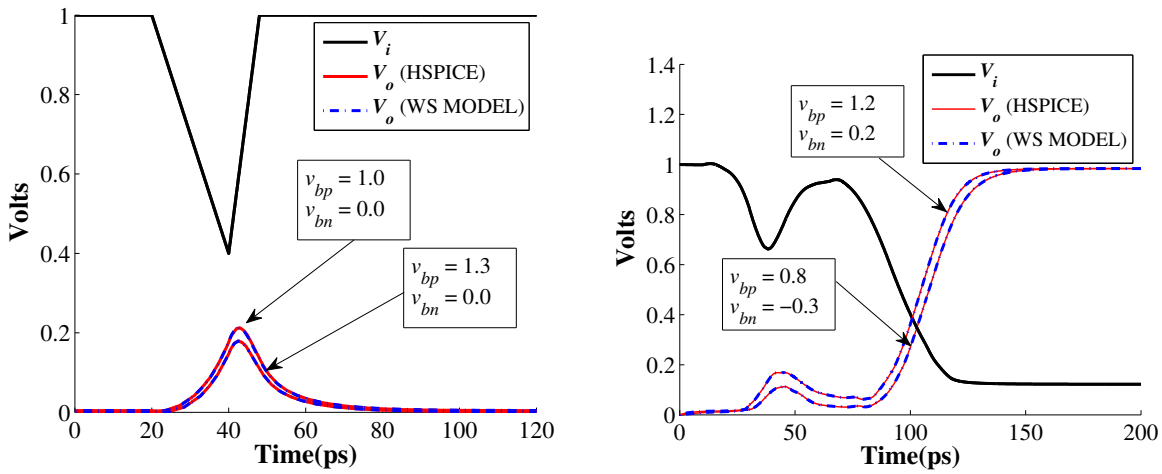
Figure 3.9: The result of our simplified body bias waveform sensitivity (WS) method as compared with HSPICE, for several body bias values: (a) output waveform from an Inverter, loaded with a 20l benchmark RC interconnect, evaluated at sink node 52, and (b) output waveform from a NAND2, loaded with a 45l benchmark RC interconnect, evaluated at sink node 103.

Temperature waveform sensitivity model

Next, we evaluate the speedup achieved using the standalone simplified temperature waveform sensitivity model as presented in Section 3.5.2. We perform evaluations at 13 temperature points within the range $\Delta T \in$ [-50°C, 100°C] with zero body biasing. Table 3.3 presents the average speedup attained over HSPICE and the Newton-Raphson solver. Compared to body bias case, these speedups are lower since we are evaluating at a much lesser number of temperature points (13 as compared to 169 in the body bias case).

Complete waveform sensitivity model

We now present the speedup obtained with our complete body bias and temperature waveform sensitivity model as presented in Section 3.5.3. In this case, we perform evaluations at all combinations of the 169 body bias and 13 temperature points within the range $(v_{bp}, v_{bn}) \in$ [-0.3V, 0.3V], and $\Delta T \in$ [-50°C, 100°C] (thus a total of 169 $\times$ 13



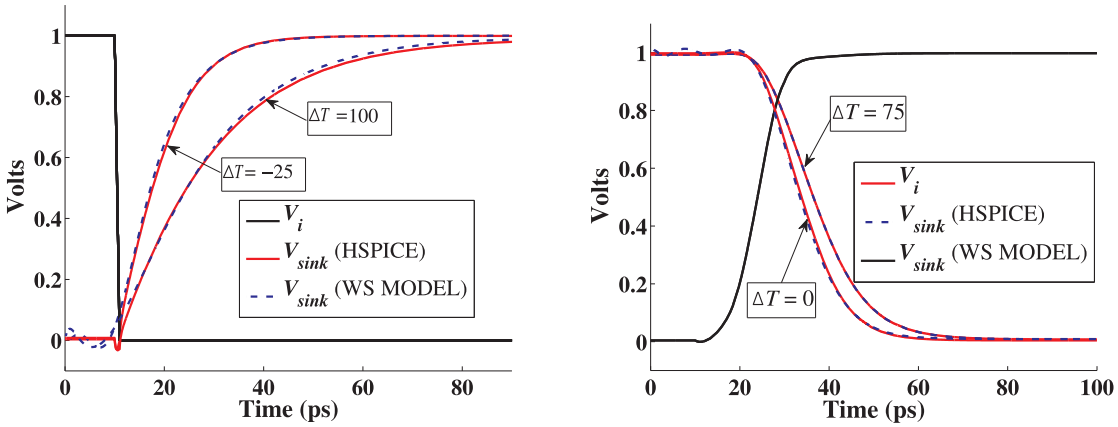
(a) (b)

Figure 3.10: Similar results of output waveform at the output node of a gate (a) for a NAND2, modeling an input glitch, and (b) for a NAND3, with a nonmonotone input.

evaluations). Table 3.3 presents the average speedup attained by our complete model over HSPICE and the Newton-Raphson solver. Note that with the complete model, we are able to achieve an order of five magnitudes speedup over HSPICE. Our complete model is much faster as compared to the Newton-Raphson solver, over which we are able to achieve an average speedup of 91.81×, considering all temperature and body bias points.

3.6.3 Accuracy of the Waveform Sensitivity Models

In this subsection, we present the accuracy achieved by our body bias and temperature models in both waveform generation and computation of slews and delays over multiple combinations of body bias and temperature values. Through the accuracy of these waveforms and low errors in slews and delays, we also show that our assumptions of making waveform sensitivity models simplified are justified. We present accurate waveform generation both at the output node of the cell as well as the sink nodes of the RC interconnect loads, which are connected to the output node of the cell.

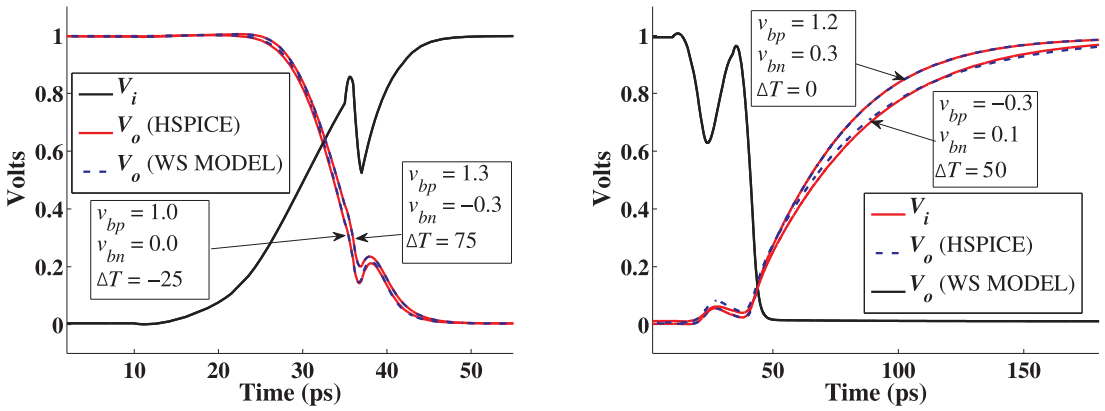


(a) (b)

Figure 3.11: The result of our simplified temperature waveform sensitivity (WS) method as compared with HSPICE, for various temperature values. Shown above are output waveforms from a NOR3, loaded with $33l$ benchmark RC interconnect, evaluated at sink node 55: (a) for a falling step input, and (b) for a slower rising input.

Body bias waveform sensitivity model

The temperature offset in this part of evaluation is set to zero. Fig. 3.9 and 3.10 compare representative waveforms as generated through HSPICE [14] and the simplified body bias waveform sensitivity model, when the input waveform takes any arbitrary shape either due to glitches, noise or crosstalk. We evaluate accuracies both at the output node of the cell, and the sink nodes of the interconnects which load the cell output node. Fig. 3.9 shows the typical response of the cell at the sink nodes of the RC tree interconnect loads. The waveform is first obtained at the output node of the cell, and then evaluated at sink node using Padé-approximation of the RC interconnect circuit, and model order reduction techniques [62]. Fig. 3.10 shows the output waveforms at the output node of the cells, with arbitrary inputs. The waveforms in some cases are coincident to the naked eye, as our algorithm yields high accuracy. This also validates the idea that α, β can be assumed to be independent of (v_{bp}, v_{bn}), as proposed in Section 3.5.2. Note that the initial ringing error in these waveforms is due to the use of Padé-approximation, and not due to the waveform sensitivity model.



(a) (b)

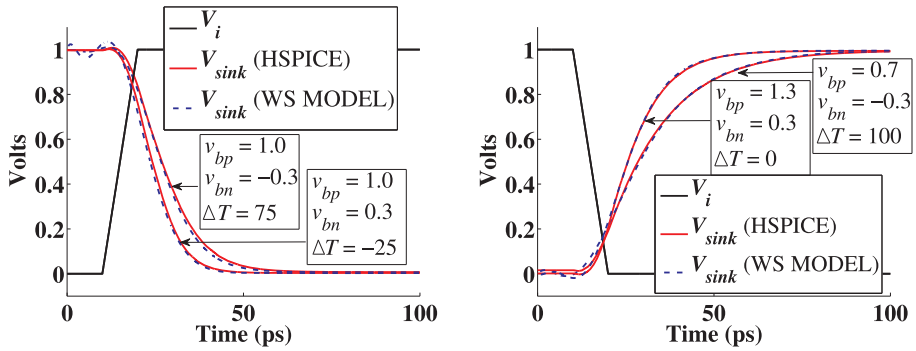
Figure 3.12: The result of our complete waveform sensitivity (WS) model as compared with HSPICE, for various temperature and body bias values. Shown above are waveforms at the output node of an Inverter with (a) $45l$ as the interconnect load and an input glitch due to crosstalk, and (b) $25m$ as the interconnect load and an arbitrary input.

Temperature waveform sensitivity model

As with the body bias waveform sensitivity model, the simplified temperature sensitivity model as described in Section 3.5.2 yields accurate waveforms for any temperature offset value. Note that the body bias is kept at zero in all such evaluations. Fig. 3.11 shows a set of waveforms obtained from a NOR3 cell, loaded with $33l$ RC interconnect network. The waveform is first obtained at the output node of the NOR3 cell, and the waveform shown is then evaluated at sink node 55. As shown in the figure, the temperature waveform sensitivity model yields very accurate waveforms. This also validates the simplification of $\sigma(\Delta T, t)$ values, as proposed in Section 3.5.2.

Complete waveform sensitivity model

For presenting the results in this section, we generate waveforms for multiple combinations of body bias or temperature offset values and compare the result with the corresponding waveforms obtained from HSPICE. We find that the complete model as



(a) (b)

Figure 3.13: Similar output waveforms from cells loaded with $20l$ benchmark RC interconnect, evaluated at farthest sink node 52: (a) the output from an NAND2 for a rising input, and (b) the output from a NOR2 for a falling input.

presented in Section 3.5.3, generates very accurate waveforms. As before, we evaluate accuracies both at the output node of the cell, and the sink nodes of the interconnects which load the cell output node. Fig. 3.12 shows the accuracy obtained at the output node of an inverter loaded with $45l$ RC interconnect, with inputs having glitches and arbitrary shapes. Fig. 3.13 shows the waveform evaluated at sink node 52 of $20l$ RC load interconnect for NAND2 and NOR2 cells. Our results show that a linear model for $V_o(t)$ in both body bias and temperature with simplifications as in Equations (3.35) and (3.38), suffices for generation of waveforms at any combination of body bias and temperature, with sufficiently desired accuracy.

Slew and delay errors

We now present some more descriptive tables for the errors in delays and slews that are incurred in formulation of the complete waveform sensitivity model.

Table 3.4 lists rise and fall delay/slew errors, for a particular test case: with a NAND2 cell loaded with $20l$ as the RC interconnect, waveforms being evaluated at sink node 52. This table shows the distribution of the delay/slew errors over all (v_{bp}, v_{bn}) points, but at different ΔT values. Column 1 of the table lists the temperature offsets from the nominal temperature of 25°C at which the waveforms are evaluated. Columns 2 to 4 present, for the value of ΔT listed in column 1, the maximum, the minimum

Table 3.4: Percent delay and slew errors for a NAND2 cell at various temperature offsets, over all v_{bp}, v_{bn} points

ΔT	Percent Delay Errors			Percent Slew Errors		
(°C)	Max.	Min.	Mean	Max.	Min.	Mean
-50.0	1.43	0.06	0.47	4.63	0.09	1.75
-37.5	1.15	0.02	0.39	4.33	0.19	1.78
-25.0	1.23	0.03	0.40	3.71	0.16	1.64
-12.5	1.14	0.01	0.49	2.02	0.10	0.82
0.0	1.29	0.08	0.44	1.23	0.08	0.70
12.5	1.20	0.09	0.40	1.51	0.06	0.78
25.0	1.15	0.05	0.45	2.50	0.13	1.04
37.5	1.34	0.09	0.66	4.16	0.04	1.53
50.0	2.18	0.38	1.11	4.81	0.06	2.65
62.5	1.96	0.22	0.81	5.48	0.27	2.79
75.0	1.27	0.02	0.39	5.94	0.26	2.92
87.5	1.18	0.06	0.68	6.87	0.40	3.19
100.0	2.16	0.63	1.41	8.93	0.35	3.74

and the mean of the percentage delay errors obtained over all v_{bp}, v_{bn} points. Similarly, columns 5 to 7 present the maximum, the minimum and the mean percentage slew errors obtained over all v_{bp}, v_{bn} points at that temperature offset. Clearly, at all temperature offsets, both the mean delay/slew errors over all v_{bp}, v_{bn} points are contained within 4%.

A more comprehensive view of these rise and fall delay/slew errors is presented in Table 3.5. For this tabulation, we work with the test cases that were mentioned at the beginning of Section 3.6.2, and save the delay and slew values as obtained from our complete waveform sensitivity model and from HSPICE. We then obtain the relative percentage error between delays and slews corresponding to the complete simplified waveform sensitivity model and of HSPICE. All such errors are tabulated. Table 3.5 shows the mean and standard deviation of these relative errors for a NAND2 cell, over all (v_{bp}, v_{bn}, ΔT) points, presented for each combination of inputs slews and output load interconnects. It is seen that both the mean and standard deviations are small for all test cases.

We thus observe from Tables 3.4 and 3.5 that only a small error is incurred in both delays and slews over all combinations of v_{bp}, v_{bn}, and ΔT points, validating the use of our waveform sensitivity model in predicting the delay and slews over the entire range

Table 3.5: Mean (μ) and standard deviation (σ) of the percentage errors over all (v_{bp}, v_{bn}, ΔT) points, incurred by our complete waveform sensitivity model in the output rise delay and slew values for NAND2 cell, as compared to HSPICE for different input slews and output RC interconnect loads

Input Slews	RC Interconnect Loads											
	$25m$				$33l$				$45l$			
	Delay Error		Slew Error		Delay Error		Slew Error		Delay Error		Slew Error	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
5ps	1.66%	1.45%	3.13%	2.54%	1.22%	1.27%	3.33%	2.63%	0.89%	0.83%	3.11%	2.45%
10ps	1.53%	1.39%	3.11%	2.54%	1.03%	1.19%	3.20%	2.66%	1.04%	0.96%	2.41%	2.30%
20ps	1.33%	1.38%	3.10%	2.55%	0.88%	0.95%	2.83%	2.60%	1.56%	1.34%	2.05%	2.16%
50ps	1.03%	1.16%	2.82%	2.57%	1.40%	1.09%	2.35%	2.53%	4.03%	3.99%	1.88%	2.17%
100ps	1.24%	1.19%	1.18%	1.40%	2.25%	1.73%	2.09%	2.33%	1.79%	1.81%	1.87%	2.35%

of body bias and temperature. Very similar observations were made for other standard cells in the library.

Chapter 4

BTI-Aware Design using Variable Latency Units

In Chapter 3, we presented a methodology for augmenting timing tools with the capability of modeling body bias and temperature. Such timing tools are typically used by conventional synchronous designs, where the worst-case delay is optimized for. Therefore, the worst-case delay also determines the clock period, T_{clk}.

In the next two chapters (including this chapter), we address reliability and variability through a paradigm where the clock period is based on the notion of average-case computations rather than the worst-case computations in a circuit – an approach that leads to improved data throughput.

4.1 Variable Latency Units (VLUs)

4.1.1 Average-Case Computation

Average-case computation, as the name suggests, refers to those computations that occur more frequently than others, and also get completed within average delays, considering the delay required by all the computations the circuit performs. Within the synchronous paradigm, two classes of techniques have been proposed for exploiting the average-case computations: variable-latency units [18, 64, 65], and error detection-correction units [19]. Our work in this chapter focuses on the design of BTI-resilient

circuits using variable latency units (VLUs).

Unlike conventional combinational circuits that complete operations within one clock cycle, VLUs allow the computation of the combinational circuit to be completed in a variable, integer, number of clock cycles. By allowing high-probability operations to complete in a single cycle, but allowing rarer events to use multiple (typically two) cycles, the average cycle time may be shorter than that of the conventional implementation, implying that the circuit throughput for a VLU may be significantly larger.

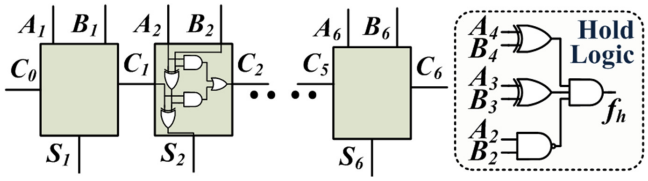


Figure 4.1: A VLU implementation of a 6-bit ripple carry adder.

As an illustration of a VLU, consider the 6-bit ripple carry adder (RCA) shown in Fig. 4.1, with six full adders. Assuming unit gate delays, the conventional single-cycle fixed-latency combinational circuit has a cycle time, $T_{clk} = 13$ units, equal to the delay of its longest path, corresponding to a throughput, $\eta_1 = 1/13$. The VLU implementation of this adder operates at a reduced cycle time, $T_{clk} < 13$. For $T_{clk} = 9$, assuming that all primary input signals are mutually independent and have signal probabilities of 50%, 18.75% of the input patterns violate T_{clk}, and the VLU allows these to complete execution in two cycles. Under the 50% assumption above, each pattern is equiprobable, so that the average VLU delay is $0.8125 \times 9 + 0.1875 \times 18 = 10.69$ units, and the corresponding throughput $\eta_2 = 1/10.69$ is 21.6% better.

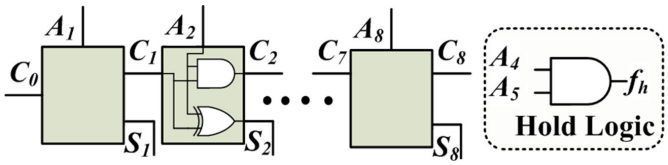


Figure 4.2: VLU implementation of an 8-bit incrementer.

Similarly, the VLU design for an 8-bit incrementer as shown in Fig. 4.2, can be clocked at $T_{clk} = 5$ units as opposed to its combinational design clock period of $T_{clk} = 8$

units (assuming unit delays for each of the gates). This results in throughput enhancement of 28.0%, with a hold logic activation probability of 25%. Therefore, exploiting the average-case computation can result in higher performance (i.e., throughput) than the worst-case.

VLUs require dedicated combinational circuitry for identifying the input patterns that require two cycles for completion, to prompt each output flip-flop to hold its current value at the next clock transition (rather than clocking in a new value). This is referred to as "hold logic" and its output is called the "hold signal." Techniques for constructing the hold logic have been proposed in [18, 65]. The hold logic for the RCA and the incrementer here is small and is shown in Figs. 4.1 and 4.2.

4.1.2 Hold Logic Generation

Our VLU scheme assumes that an operation completes in either one or two clock cycles. Given a timing constraint T_{clk}, each path whose (delay + setup time) is larger than or equal to T_{clk} is termed as a critical path. If a set of input assignments I_P sensitizes a set of critical paths C_P, it must also evaluate the hold signal to 1. Paths excited by these input patterns are allowed two cycles for completion.

A direct enumeration of paths (similar to [66]) is a direct approach for exact generation of hold expression f_h, but becomes very restrictive with the increase in number of critical paths, the number of gates and primary inputs in the circuit. This was addressed in [18, 65], in which the authors propose a 'node-based' algorithm as opposed to a 'path-based' algorithm, based on a traversal of the critical gates since the number of (critical) paths can be exponential in the number of gates in the network, whereas the number of critical gates is guaranteed to be smaller than the number of gates.

We now briefly review an algorithm built on a corner based methodology for generating the hold logic; details are provided in [18, 65]. Consider the circuit in Fig. 4.3(a). Assuming unit gate delays, the longest path has a delay of 4 units. With a required time of 3 units on x, y, the circuit has one critical path P, and the sensitization condition for this path constitutes the hold logic expression. Here, the critical path can be sensitized if every gate g on the path has noncontrolling values on its noncritical inputs. Let $S(g)$ represent this condition for gate g. The condition for sensitization for P, which activates the hold logic, is then given by $f_h = \prod_{\forall g \in P} S(g) = b \cdot c \cdot e \cdot f$.

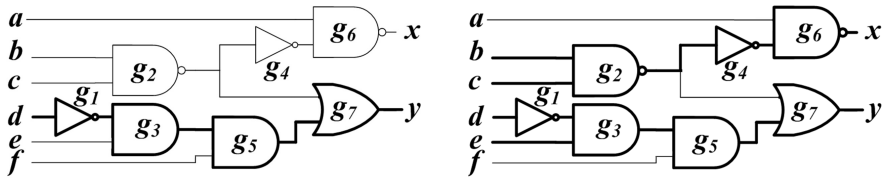


Figure 4.3: An example of a circuit at various timing specifications so that it has (a) one critical path and (b) four critical paths.

For a more stringent required time specification of 2 units on x, y, the circuit has four critical paths, as shown in Fig. 4.3(b). The sensitization conditions for multiple critical paths can be recursively computed using a method described in [18] to obtain the hold logic expression. If any one of the critical paths is sensitized by an input pattern, the hold signal should be set to 1.

The sensitization conditions for multiple critical paths is thus derived from the summation of sensitization condition for every critical path. This is computed by the formulation of $CPAF(g)$, the critical path activation function of gate g representing the summation of the boolean condition that all the critical paths through g are allowed to pass through. This happens when all the noncritical inputs of g are noncontrolling (or in other words, $S(g) = 1$, as defined above). $CPAF(g)$ is thus formulated using all its critical fainins i of g as follows: $CPAF(g) = \sum_{\forall i} S(g) \cdot CPAF(i)$. Since $CPAF(g)$ needs to be constructed from the $CPAF$s of g's critical fanins, this involves a topological traversal from PIs to the POs of the circuit, with the $CPAF$s of PIs being a boolean 1. The hold logic expression f_h is finally obtained by the summation of the $CPAF$s of all the primary outputs. This can be illustrated through the example in Fig. 4.3(b),

$$\begin{aligned} CPAF(x) &= CPAF(g_4) \cdot S(g_6) = CPAF(g_2) \cdot a \\ &= (CPAF(b) + CPAF(c)) \cdot a = 1 \cdot a = a \end{aligned}$$

$$
\begin{aligned}
CPAF(y) &= CPAF(g_5) \cdot S(g_7) = (CPAF(g_3) \cdot S(g_5)) \cdot b \cdot c \\
&= (CPAF(g_1) + CPAF(e)) \cdot f \cdot b \cdot c \\
&= (CPAF(d) + CPAF(e)) \cdot S(f) \cdot b \cdot c \\
&= 1 \cdot f \cdot b \cdot c = b \cdot c \cdot f \\
f_h &= CPAF(x) + CPAF(y) = a + b \cdot c \cdot f
\end{aligned}
$$

The hold logic size may sometimes be appreciably large. We have implemented some heuristics to reduce the hold logic size which are presented in [18, 65]. Note that the algorithm *does not* work with paths, but involves a single topological traversal and processing of all the gates in the circuit, incurring *an O(n) complexity*, where n = number of gates in the circuit. Heuristic techniques to control the size of the hold logic have been presented in [18, 65].

Given the signal probabilities at the primary inputs (PIs) of the circuit, the average throughput η of the VLU is evaluated as the inverse of the average cycle time [65]:

$$
\eta = \frac{1}{P_{hold} \cdot 2T_{clk} + (1 - P_{hold}) \cdot T_{clk}} = \frac{1}{(1 + P_{hold}) \cdot T_{clk}} \tag{4.1}
$$

where P_{hold} is the hold logic activation probability.

4.1.3 VLUs at the Architectural-Level

Variable latency operation can be easily incorporated in a processor architecture. Fig. 4.4 shows a typical five stage instruction pipeline in a microprocessor: variable latency designs may be employed at the EX stage. When the EX stage requires a two-cycle operation, it generates a stall for one cycle, preventing data from the IF and ID stage from moving forward to the EX stage. Such a stall can be implemented through a simple extension of a conventional *hazard detection unit* (HDU), which is a standard feature that stalls such pipelines in the presence of data hazard. A modified design shown in Fig. 4.4, which now allows both the hold signal and the HDU output to activate and control the stalling mechanism in the pipeline, shows a low-overhead implementation that facilitates the use of the VLU.

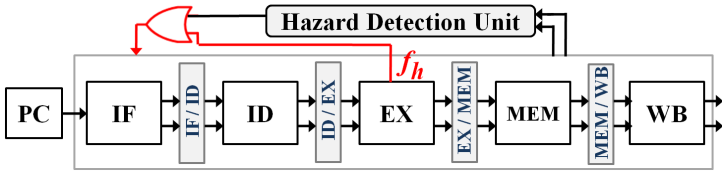


Figure 4.4: Variable latency operation at the architectural level: the output of the HDU is appended to the hold signal to stall the pipeline for a two-cycle operation.

4.2 VLUs and BTI

4.2.1 Motivation

As discussed in Chapters 1 and 2, BTI degradation causes timing errors and performance loss. Published approaches for enhancing BTI-resiliency include transistor sizing, logic resynthesis, or postsilicon tuning, which all work with worst-case delays. Use of VLUs therefore, can potentially enhance the performance of such circuits.

However, the performance of a VLU can also degrade or even become incorrect in the presence of BTI, potentially leading to circuit failure as the circuit degrades temporally. Few efforts have been in the direction of constructing BTI-resilient circuits using the variable latency paradigm. One such technique has been explored in [67] for specific adders, but this work does not extend to general circuits.

We therefore develop novel methods for building VLU-based BTI-resilient designs for a general circuit, such that the performance over its lifetime is maximized, and relies on two ideas: (a) using a novel scheme that uses *multioutput hold logic* (MOHL) to alter the appropriate hold logic over time, in conjunction with (b) using adaptive body biases [21] to maximize circuit performance.

The contents of this chapter are organized as follows: the concept of MOHL is described in Section 4.3, followed by the details of our approach in Section 4.4. Next, Section 4.5 shows how our schemes can be used even with general BTI models, and Section 4.6 discusses circuit performance optimization using MOHL as well as body biasing. Finally, we experimentally validate our method in Section 4.7.

4.2.2 BTI Degradation Model and Delay Monotonicity

Before entering our main discussion, it will be useful to highlight the BTI model used for work. For a particular gate, we use HSPICE to precharacterize the impact of changes in V_{th} on the delay D_g of a gate with n transistors as in [12], using the formulation described in Section 2.2.4. Such characterizations are reasonably inexpensive and have been widely deployed, e.g., in statistical static timing analysis (SSTA) methodologies.

This computation requires the determination of ΔV_{th} for each device in the gate, for which we adopt Equation (2.3). Clearly, this equation suggests is a monotonically increasing curve, and captures the effect of applying constant stress to a gate. Although BTI is known to show some recovery when the stress is removed, modeling this recovery requires precise knowledge of the input pattern distributions for each specific gate.

One way to achieve this is through signal probability information: however, there are two drawbacks to this. First, such information may not be available. Second, the available information can be quite inaccurate; it may predict the average behavior over all users and programs for a system, but the actual aging depends on the behavior of a specific user, which may not be predictable. Another potential approach is through the use of on-chip sensors, but these are of limited utility since they do not experience the same signal patterns as the circuits whose aging is to be measured. In our implementation, we employ the worst-case pattern for aging for BTI analysis, at the worst-case temperature corner, as a guaranteed pessimistic estimate of usage; this approach is consistent with widely-used methods for handling BTI.

For well-characterized circuits, specific information that is available about the signal probabilities at each node can easily be incorporated into our framework. Such circuits undergo stress/recovery cycles, and the delay model may work with the stress/recovery envelope [48, 49] of the V_{th} vs. t curve. By definition, such an envelope is always monotonically increasing.

One case that deserves special mention is when a functional unit is turned off for a long period of time, which is detectable by a sensor (e.g., software timers, sensors, or separate antifuses may keep track of the on- and off-times). In this case, due to recovery effects, the circuit undergoes "rejuvenation" as the threshold voltage recovers and the delay degradation is eased, and the use of an envelope waveform may be far too pessimistic. This case is addressed in Section 4.5.

4.3 Multioutput Hold Logic: Concept

We now introduce the concept of *multioutput hold logic* (MOHL). We begin with the idea that the task of building BTI-resilient VLUs is, in essence, one of partitioning the set of circuit paths into one-cycle and two-cycle paths[1] . Intuitively, for high throughput, it is important to keep the most frequently-excited paths in the one-cycle set (or more quantitatively, to keep the value of P_{hold} high). However, under BTI, path delays may change with time: under the monotone delay model, as path delays increase, more paths may move from the one-cycle set to the two-cycle set. We develop a systematic framework for choosing an appropriate partition of one-cycle/two-cycle paths with the option of changing this partition over time as delays degrade due to BTI.

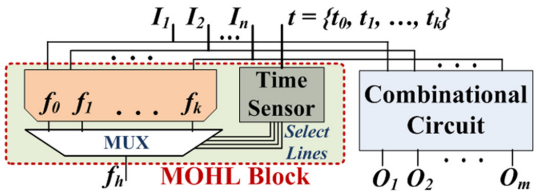


Figure 4.5: The concept of MOHL VLU design. A time sensor selects the hold logic to be triggered at time t.

Our solution is based on the concept of an MOHL, introduced in Fig. 4.5. Under this scheme, the VLU is controlled by temporally-varying hold logic circuitry. As the circuit ages, by the monotonicity argument presented in Section 4.2.2, an increasing number of single-cycle paths may require two cycles for completion, and the set of two-cycle input patterns changes. To account for this, the MOHL circuit has multiple hold signal outputs: depending on the age of the circuit, one is chosen. A hardware or software time sensor can capture the system operational time, and this information can be fed to a multiplexer that selects the proper hold signal to be triggered at time t.

[1] As stated earlier, references to enumerated paths are only an aid to explanation; our actual implementations *do not* perform path enumeration.

4.4 Multioutput Hold Logic: Theory

We will now outline some useful properties of the MOHL design problem. In particular, we will describe some specific supersetting patterns in the two-dimensional space of frequency and circuit aging that can be used to build compact implementations of the hold logic. These patterns are based on the assumption of monotonic aging, described in Section 4.2.2; this assumption is removed in Section 4.5. We use standard synthesis tools for synthesizing MOHL; our contribution is in identifying these subsetting patterns that reduce the implementation overhead in terms of the number of outputs, area, etc.

4.4.1 Tabulating the Effects of Aging on VLUs

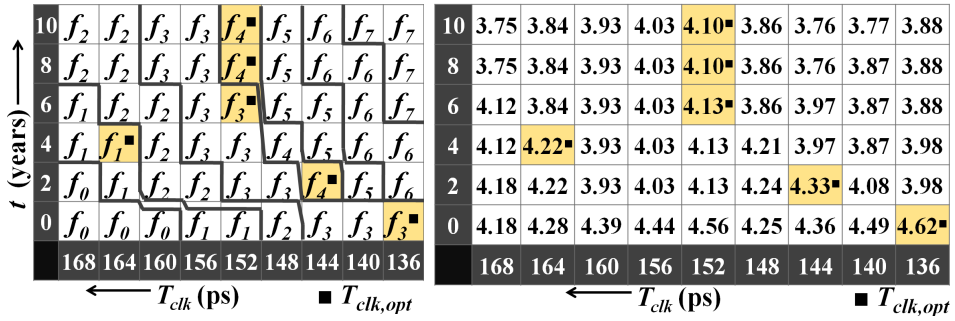
As a circuit ages, the distribution of its path delays changes, and therefore, the hold logic that is required to operate the circuit at a specific value of T_{clk} changes. Conversely, at any time t, the hold logic required to ensure timing correctness is a function of T_{clk}. We capture these relationships in a *frequency/aging (F/A) grid*, a table whose columns are the possible values of T_{clk} and whose rows correspond to t, the age of the circuit. In the discussion to follow,

- $\mathcal{H}(T_{clk}, t)$ denotes the ON-set (or the $\mathcal{H}$-set) of the hold logic required to achieve a clock period of T_{clk} at a given time t.
- $\eta(T_{clk}, t)$ represents the corresponding value of η.

We use the terms "hold logic" and "$\mathcal{H}$-set" interchangeably.

The entry at each (T_{clk}, t) point in the grid represents the hold logic function (with T_{clk} as the one-cycle time), which is associated with a specific value of P_{hold} and η, the probability of hold logic activation and the throughput, respectively. In principle, the F/A grid is a discretization on the continuous space of (T_{clk}, t) values. A representative example of an F/A grid for benchmark apex7 is shown in Fig. 4.6(a), and the corresponding η values are displayed in Fig. 4.6(b).

The throughput for each hold logic function is computed using Equation (4.1). We may perform a linear search on the values of T_{clk} in a row of the F/A grid to determine the point at which η is maximized. In each row in Fig. 4.6(b), we highlight the maximum-throughput entry and denote the corresponding value of T_{clk} by $T_{clk,opt}(t)$. It is easily



(a) (b)

Figure 4.6: The F/A grid for circuit apex7 showing (a) the hold logics and (b) the corresponding η values (shown on a 10^{-3} scale). The patterns in the grid correspond to supersetting structures and result as a consequence of the application of Theorems 1 and 2, and Corollaries 1 and 2.

seen that $T_{clk,opt}(t)$ is not, in general, monotone with t.

4.4.2 Supersetting Trends

We now present some supersetting trends that help in minimizing the circuitry required to implement MOHL as a multioutput circuit.

Theorem 3 *At a given time t, for two clock period constraints applied to a VLU, $\mathcal{H}(T_{clk,2}, t) \supseteq \mathcal{H}(T_{clk,1}, t)$ and $P_{hold}(T_{clk,2}, t) \geq P_{hold}(T_{clk,1}, t)$ value if $T_{clk,1} > T_{clk,2}$.*

Proof: At any time t during the life of the chip, as we decrease the value of T_{clk}, the timing constraints on the circuit are tightened and more paths require two cycles for completion. Since gate delays increase monotonically with time, for any $T_{clk,2} < T_{clk,1}$, all input patterns that excite a path with a delay larger than $T_{clk,1}$ will clearly excite a path with delay larger than $T_{clk,2}$ (in addition to these, other patterns may also excite two-cycle paths). Since such input patterns constitute the ON-set of the hold logic of the circuit, the ON-set of hold logic at $T_{clk,1}$ is wholly contained within the ON-set of hold logic at $T_{clk,2}$, i.e., $\mathcal{H}(T_{clk,2}, t) \supseteq \mathcal{H}(T_{clk,1}, t)$. The corresponding result about P_{hold} follows trivially from this. □.

Example: In Fig. 4.6, $f_0 \subset f_1 \subset f_2 \subset f_3$ in the row $t = 0$.

Theorem 4 *Under a monotonic delay model, at a given clock period T_{clk} applied to a VLU, for two time points $t_1 > t_2$, $\mathcal{H}(T_{clk}, t_1) \supseteq \mathcal{H}(T_{clk}, t_2)$ and $P_{hold}(T_{clk}, t_1) \geq P_{hold}(T_{clk}, t_2)$.*

Proof: By the monotonicity of the BTI degradation model described in Section 4.2.2, the path delays in a circuit slow down with time. Therefore, the same clock period T_{clk} constitutes a tighter requirement at time t_1 than at t_2. Therefore, more input patterns are included in the ON-set of the hold logic at t_1, i.e., $\mathcal{H}(T_{clk}, t_1) \supseteq \mathcal{H}(T_{clk}, t_2)$. □

Example: In Fig. 4.6, by Theorem 4, $f_1 \subset f_3 \subset f_4$ in the column $T_{clk} = 152$.

These two theorems can be combined to define a corollary that describes a broader supersetting relationship between the hold logics.

Corollary 1: If $t_2 > t_1$ and $T_{clk,2} < T_{clk,1}$, then $\mathcal{H}(T_{clk,1}, t_1) \subseteq \mathcal{H}(T_{clk,2}, t_2)$ and $P_{hold}(T_{clk,1}, t_1) \leq P_{hold}(T_{clk,2}, t_2)$.

As a result of these supersetting trends, it is possible to detect patterns in the F/A grid. In Fig. 4.6(a), the F/A grid for apex7 is divided into regions such that a single hold logic represents each region. Note that by Corollary 1, a region that is to the north or east of another region bears a subset relationship for the corresponding hold logic. The subsetting relationships for the $T_{clk,opt}$ entries in the table are:

$$\mathcal{H}(164, 4) = f_1 \subseteq \mathcal{H}(152, 6) = f_3 \subseteq \mathcal{H}(152, 8) = f_4$$
$$\mathcal{H}(136, 0) = \mathcal{H}(152, 6) = f_3$$
$$\mathcal{H}(144, 2) = \mathcal{H}(152, 8) = \mathcal{H}(152, 10) = f_4$$

In other words, all of the hold logic in this example follows a supersetting trend; however, this is not necessarily true in general. Even so, there can be substantial overlap in the minterms of the optimal hold logic at various times, a result that is formalized below.

Corollary 2: If $t_2 > t_1$ and $T_{clk,2} > T_{clk,1}$, then, $\exists\ \mathcal{H}(T_{clk,2}, t_1)$ that is a subset of both $\mathcal{H}(T_{clk,1}, t_1)$ and $\mathcal{H}(T_{clk,2}, t_2)$.

The "common ancestor," $\mathcal{H}(T_{clk,2}, t_1)$, allows sharing between the circuitry needed to implement $\mathcal{H}(T_{clk,1}, t_1)$ and $\mathcal{H}(T_{clk,2}, t_2)$, and implies that the two share a set of minterms. This indicates the likelihood that the area required to implement the hold logic as a multioutput circuit is less that the sum of separate implementations. This has been verified empirically on benchmark circuits.

4.5 Rejuvenation: Nonmonotone BTI Models

Due to recovery effects, BTI aging may be nonmonotone. In practical workload scenarios, a gate may suffer both stresses and relaxations (removal of stress, causing delay recovery) and the monotonic delay degradation assumption breaks down. As discussed in Section 4.2.2, we focus on the more predictable case when a circuit is power-gated. During this period, the threshold voltage of all transistors recovers from BTI stress in a predictable way. This case is also practical because online sensing/detection techniques can easily be leveraged to determine the period of time when the circuit is in recovery mode.

We show that recovery has the effect of *rejuvenation*, effectively making a circuit "younger," and we may utilize the F/A table, except that an "effective age" of the circuit is used along the t axis. All other ideas related to supersetting patterns presented in Section 4.4 are preserved. We establish this through the following theorem.

Theorem 5 *Delay recovery in a circuit, when BTI stress is removed, can be captured by moving backward in time t along the F/A grid.*

Proof: The change in the delay is the product of the delay sensitivity to V_{th}, multiplied by ΔV_{th}. Since ΔV_{th} is reduced during recovery, the delay degradation is reduced, and aging is effectively (partially) reversed, corresponding to moving backward in time along the F/A grid. Consider a gate g on some path P in the circuit. The gate experiences stresses from time $0 \leq t \leq t_1$ and subsequent recovery in time $t_1 < t \leq t_2$; the corresponding shifts in the threshold voltage are denoted as $\Delta V_{th}(0, t_1)$ and $\Delta V_{th}(t_1, t_2)$, respectively. The change in the gate delay can be computed by an extension of Equation (2.7):

$$\begin{aligned} D_g(t_2) &= D_{g,0} + \sum_{i=1}^{n} \frac{\partial D_g}{dV_{th_i}} \Delta V_{th_i}(0, t_1) - \sum_{i=1}^{n} \frac{\partial D_g}{dV_{th_i}} \Delta V_{th_i}(t_1, t_2) \\ &= D_{g,0} + \left[\sum_{i=1}^{n} \frac{\partial D_g}{dV_{th_i}} \right] \cdot (A(t_1) - B(t_2 - t_1)) \end{aligned} \tag{4.2}$$

where $A(t_1)$ and $B(t_2 - t_1)$ represent the changes in V_{th} at the end of the stress and relaxation phases, respectively. Note that the functions A and B depend on the periods of the two intervals (e.g., $A(t) \propto t^{\frac{1}{6}}$). If we consider all gates g on a path P, the path

delay of P at time t_2 can be computed as:

$$D_P(t_2) = D_{P,0} + \left[\sum_{g=1}^{m}\sum_{i=1}^{n}\frac{\partial D_g}{dV_{th_i}}\right] \cdot (A(t_1) - B(t_2 - t_1)) \tag{4.3}$$
$$\text{where, } D_{P,0} = \sum_{g=1}^{m} D_{g,0}$$

The absolute delay change caused due to relaxation can be no more than that caused due to stress (and is often less), and the circuit cannot recover *fully* in a finite amount of relaxation time t after stress. The term $(A(t_1) - B(t_2 - t_1))$ therefore, is always positive, implying that $D_P(t_2) \geq D_{P,0}$. Hence we can rewrite Equation (4.3) as:

$$\exists\ t_1' \leq t_1, \text{ such that } A(t_1') = A(t_1) - B(t_2 - t_1) \text{ and,}$$
$$D_P(t_2) = D_P(t_1') = D_{P,0} + \left[\sum_{g=1}^{m}\sum_{i=1}^{n}\frac{\partial D_g}{dV_{th_i}}\right] \cdot A(t_1') \tag{4.4}$$

Equation (4.4) is of the same form as the path-level extension of Equation (2.7) for all the gates g on path P. This implies that the recovery in delay caused due to relaxation can be seen, in the light of monotonically increasing delay degradation model of Equation (2.7), to have an effect of "rejuvenating" the circuit. Specifically, although the circuit is t_2 time units old and has been stressed for time t_1, its effective age t_1' is less than either number. The idea of the F/A table may now be used exactly as before, except that the new age of the circuit is t_1', and all decisions must be made from the entries of row t_1' in the F/A grid (with $t_1' \leq t_1 \leq t_2$). In other words, rejuvenation corresponds to moving backward in time t. □.

It is easy to extend this analysis to multiple stress/recovery cycles.

4.6 BTI-Resilient VLUs

At a fixed value of T_{clk}, as the VLU ages and more paths require two-cycle operation, the concept of MOHL, as outlined in Section 4.3, may be used to implement BTI-resilience. We classify the techniques for MOHL as being either *static*, when T_{clk} is constant through the lifetime of the circuit, or *dynamic*, when its value is tuned for maximal throughput during the circuit lifetime. Since the optimally-tuned T_{clk} may

vary from one functional unit to another, dynamic MOHLs are not realistic in mainstream systems, where T_{clk} is set through the use of system-level considerations, but the corresponding ideas will be used to obtain a better implementation of a static MOHL circuit.

4.6.1 Static MOHL VLU Implementation

This approach chooses a fixed T_{clk} through the life of the circuit. At $t = 0$, the hold logic corresponds to an initial set of two-cycle paths. As the circuit ages, under monotonicity, more paths require two cycles, and the MOHL is updated appropriately.

This scheme corresponds to moving along a single column of the F/A grid, corresponding to the $\{(T_{clk}, t), \forall t\}$, where T_{clk} is the specified period. By Theorem 4, the hold logic at each successive point in time is a superset of that before it. Therefore, there are substantial containment relationships between the $\mathcal{H}$-sets, which can be leveraged to build minimal multioutput functions.

The static MOHL VLU has two limitations: 1) the value of P_{hold} also increases monotonically with time; from Equation (4.1), its throughput reduces monotonically as a function of time; 2) it cannot benefit from optimizations (Section 4.4.1) that adjust the clock period.

4.6.2 Adaptive MOHL VLU Implementation Using Body Biases

To overcome these limitations of static MOHL VLUs, we consider a scenario where we build a dynamic VLU, for which it is permissible to change the clock period of the VLU as a function of time. Under this scheme, at each value of t, the MOHL VLU is operated at the optimal clock period, $T_{clk,opt}(t)$, that maximizes the throughput at time t. To enable this, the optimal hold logic is selected from different columns of the F/A grid (unlike the case for static VLUs); from Fig. 4.6, $T_{clk,opt}(t)$ is not a constant or monotone with t.

A critical limitation of such a design is that varying T_{clk} over time can create synchronization problems in pipelines. Since T_{clk} is typically set by global considerations, and its optimal value may vary from one unit to another in a system, dynamic changes in T_{clk} may only be possible under restricted scenarios such as asynchronous systems and not under mainstream applications.

Therefore, we modify this scheme to build a new approach, based on the observation that the variations in $T_{clk,opt}(t)$ in the dynamic method above are typically very small over all values of t in the lifetime of the circuit. Examining Equation (4.1), the primary gains in the throughput come about due to large discrete changes in P_{hold} (e.g., between $\mathcal{H}(152, 8)$ and $\mathcal{H}(164, 4)$ in Fig. 4.6) and the contribution of T_{clk} variations is small. This allows the possibility of operating under a more practical paradigm, with a constant T_{clk} over all time. Fig. 4.7(a) shows the working of this scheme for the F/A grid for the apex7 benchmark. Using $\mathcal{H}(T_{clk,opt}(t), t)$, ABB allows the circuit to be clocked at $T_{clk,opt}(0)$ at all t. Since $T_{clk,opt}(t)$ changes with t, the applied body bias $V_{bb}(t)$ also changes with t.

We consider two cases and use the concept of adaptive body bias ABB) [21] to preserve T_{clk}:

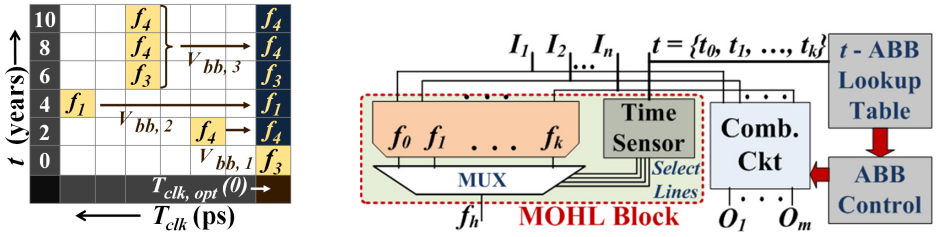
Case I: If $T_{clk,opt}(t) \leq T_{clk}$, we may simply operate the circuit at T_{clk} with hold logic corresponding to this T_{clk} (i.e., $\mathcal{H}(T_{clk}, t)$). This is functionally correct and represents an identical P_{hold} and a small shift in T_{clk} from $T_{clk,opt}$; therefore, the change in the throughput from the $t = 0$, as predicted by Equation (4.1), is very small.

Case II: If $T_{clk,opt}(t) > T_{clk}$, then the optimal circuit clearly violates T_{clk}. In this case, we use the hold circuitry (and hence P_{hold}) from the $T_{clk,opt}(t)$ point and employ forward body bias (FBB) by applying a positive body bias voltage, $V_{bb}(t)$ to speed up the path delays and reduce the clock period to T_{clk}. Since the optimal hold logic does not change and $T_{clk,opt}(t)$ is close to T_{clk}, as predicted by Equation (4.1), the throughput remains almost the same as that at $T_{clk,opt}$.

In Case II, we denote the set of one-cycle (two-cycle) paths at $T_{clk,opt}$ by $P_{1,opt}$ ($P_{2,opt}$), and the corresponding hold logic as $\mathcal{H}_{\text{opt}}$. The application of FBB speeds up all paths and the value of V_{bb} is chosen to guarantee that all paths in $P_{1,opt}$ meet the constraint, T_{clk}. It is theoretically possible that some paths in $P_{2,opt}$ may be sped up faster than those in $P_{1,opt}$ if they have vastly different sensitivities to V_{bb}, to the point that they become one-cycle paths. We refer to this set of paths as $P_{2\rightarrow1,opt}$: in practice it is unlikely that this set will have any members. Even if it were to, using the hold logic $\mathcal{H}_{\text{opt}}$ is functionally correct, but pessimistic (since it may allocate two cycles to the paths in $P_{2\rightarrow1,opt}$ instead of one).

The ABB scheme for an adaptive MOHL VLU operating at a fixed T_{clk} is illustrated through Fig. 4.7(b). To the original MOHL scheme described in Fig. 4.5, we add a

time-based t-ABB lookup table, which saves the value of $V_{bb}(t)$ that must be applied to the circuit at time t to achieve the best throughput. On a practical level, our implementation applies the same body biases to all transistors in the functional block under consideration.



(a) (b)

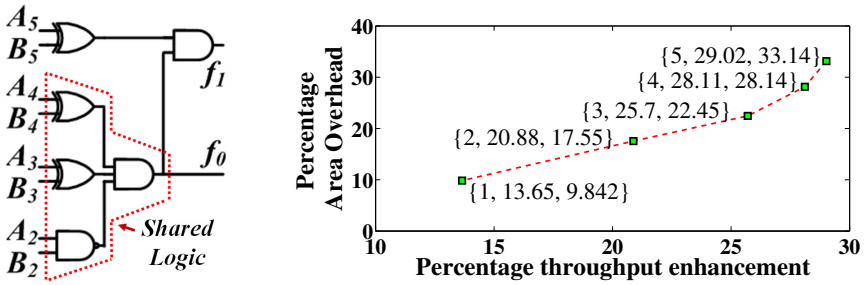
Figure 4.7: (a) Block description of the MOHL VLU design incorporating ABB, and (b) the ABB scheme corresponding to the F/A grid for apex7. Here, V_{bb} is the applied body bias to the circuit.

Over all benchmark circuits, we have found that $|V_{bb}(t)| \leq 0.25$V. In this range, we have verified that leakage power (including junction leakage) is negligible using SPICE simulations. This was also confirmed through a detailed leakage analysis presented in [12]. Therefore, we explore the range, in steps of 0.05V.

4.6.3 Practical Issues

In using the F/A grid, we constrain ourselves to the granularity of t at which it was characterized. This granularity can generally be chosen to be consistent with the degradation model (e.g., spacing it out according to $t^{1/6}$ instead of uniformly). In general, this has seen to yield MOHL with a small number of outputs, like the MOHL for 6-bit RCA shown in Fig. 4.8(a) with two outputs and significant sharing in the circuitry required for generating them. However, it is possible that the hold logic may change at *each* entry of t in the table. Under this scenario, the cost of implementing MOHL may be high, even when we leverage the subset containment properties described earlier. Fig. 4.8(b) shows that the MOHL for benchmark c5315 requires five outputs (the point in the top right corner).

In such cases, it is possible to use an MOHL implementation with a smaller number



(a) (b)

Figure 4.8: (a) The two-output BTI-resilient hold logic for the 6-bit RCA. (b) A plot for the multioutput hold logic VLU design for the circuit c5315 showing various values of the tuple: {Number of outputs, throughput enhancement (%), area overhead (%)}.

of outputs, i.e., with the hold logic changing less frequently over the life of the circuit. This results in improved implementations with lower cost, with some loss in optimality.

For example, for the adaptive scheme, we may work with $T_{clk} = T'_{clk}$ from the F/A grid, where $T'_{clk} \neq T_{clk,opt}(t)$). This idea is illustrated in Fig. 4.8(b) for c5315, which shows that initially, the MOHL is composed of five different $\mathcal{H}$-sets (top-right corner point). When the number of $\mathcal{H}$-sets is reduced to four and then to three, we see a small reduction in average throughput enhancement (29.0% to 25.7%), but a large gain in the reduction of area overheads (from 33.1% to 22.5%). However, further reduction in the number of $\mathcal{H}$-sets reduces the throughput enhancement too.

For the static scheme, similar optimizations are adopted, e.g., if only one hold logic is to be used, we choose the end-of-life hold logic; if at most two were permissible, we choose another one that provides the least lifetime loss in throughput, and so on.

4.7 Experimental Results

In this section we present results on various ISCAS85, ISCAS89, MCNC, LGSYNTH93 and ITC99 benchmark circuits, synthesized using ABC [68] on the 45nm PTM [11] based library. Our .genlib library for ABC used for mapping circuits consists of INVs; BUFs; 2-4 input NANDs and NORs; 2 input XORs and XNORs; all with different sizes. ABC also integrates the CUDD package [69] required for logical computations in hold

logic generation. We choose t_{life} = 10 years, and our experiments are based on the monotone BTI degradation model.

4.7.1 Evaluation Methodology

In order to evaluate the effectiveness of our MOHL VLU schemes as presented in Section 4.4 and Section 4.6, we compare the results of these designs, in terms of area overhead and throughput enhancements achieved, with other VLU-extensions of existing BTI-resilient designs for combinational circuits, such as the *delay padding* schemes as described below. A design may be made BTI-resilient by padding the timing specification using a margin to ensure that the circuit meets its timing requirements through its lifetime, t_{life}.

VLU sizing-based padding

A VLU can be synthesized using library delay models that predict the circuit delay at t_{life}. A padding strategy ensures that the circuit meets specifications throughout its lifetime by adding a safety margin to the timing specification. The circuits are sized to the knee of the area vs. delay curve.

Hybrid padding

As a circuit ages, an increasing number of paths violate the clock period. A conservative approach is to identify the paths that violate T_{clk} at the end-of-life: such paths are allowed two cycles throughout the entire lifetime of the circuit. This however results in significant throughput penalties, and some of these paths may well work within one cycle for part of the circuit lifetime.

We therefore combine the sizing and conservative approaches by introducing sizing-based partial padding into the combinational circuit. This approach ensures that it meets T_{clk} up to time $t = t_p$, where $t_p < t_{life}$. For $t \geq t_p$, we simply move the paths to a second cycle. The benefit achieved is that we incur lower sizing overhead as compared to the sizing method, and also lower throughput degradation as compared to the conservative method.

The area overhead for this method arises from the extra hold logic required, and from sizing costs. The choice of t_p is important in ensuring low sizing overhead. Our

implementation uses the point where the circuit suffers nearly 50% of the total degradation that it suffers over its entire lifetime. We can deduce, both through analytical algebraic techniques and through simulations, that for t_{life} =10 years, t_p = 2 years meets this criterion.

4.7.2 Area Overhead and Throughput Enhancements

Tabulation details

The results of the area overhead and throughput enhancements as achieved by various schemes are summarized in Table 4.1. Here, ΔA denotes the percentage area overhead (padding overhead and/or the hold logic area), and $\Delta\eta$ denotes the average change in throughput from $t = 0$ to t_{life}, incurred in each of the respective designs. Note that the circuit apex7 has a different mapping from that used in Fig. 4.6, and therefore shows different numbers in our results table below.

Baseline case: The baseline corresponds to a conventional one-cycle combinational implementation, where worst-case sizing is used, where all paths are required to meet T_{clk} specification throughout the circuit lifetime, using library delay models that predict the circuit delay at t_{life}. VLUs with positive (negative) overhead values incur larger (smaller) overhead as compared to this design and imply that the corresponding VLU circuit is better (worse) than its single-cycle counterpart[2] .

In presenting the data in the table, we refer to column number m as Cm, as marked in the table. For various benchmark circuits listed in C1, the delay degradation $\frac{\Delta D(t_{life})}{D(t=0)}$ over the entire lifetime lies between 5.6% and 15.6%, with an average of 11.4%. To be more realistic, our results choose the starting point $t = 0$ to be three months after the circuit is manufactured to model burn-in test procedures. (Note that starting at the real $t = 0$ would improve slightly the numbers shown in our results, but should leave the relative comparisons between various methods unchanged.)

In C2–C5, we present the results for the two methods described in Section 4.7.1: VLU sizing-based padding and hybrid padding. We then show the results for two MOHL VLU schemes (Section 4.6): the static (C6–C8) and the ABB-based (C9–C13)

[2] It is possible to use a baseline corresponding to a nominal unaged circuit without padding: in such a case, the overhead would always be positive. However, such an uncompensated circuit does not meet performance specifications, and we prefer a comparison with a functionally correct circuit. Such a comparison also shows the advantages of BTI-resilient VLUs over single-cycle implementations.

designs. Columns C6 and C9 show the number of outputs, $|\mathcal{H}|$ (number of different $\mathcal{H}$-sets), present in the MOHL circuit, and C12 shows the maximum magnitude of $V_{bb}(t)$ required at any time point in the life of the adaptive MOHL VLU.

Run times for the adaptive MOHL VLU method for various circuits are shown in C13, and are seen to be reasonable. The CPU times are presented only for the adaptive MOHL VLU scheme, since it involves relatively more runs of the hold logic computation algorithm (Section 4.1.2) for the generation of all points for the F/A grid. In other designs, we either simply perform a remapping (for sizing), or run the hold logic generation algorithm for one value of T_{clk} (for static MOHL VLU).

These runtimes depend on multiple factors: the size of the circuit, the logic functionality (which changes the size of BDD's used) and the distribution of paths in the circuit and thus do not show a monotone trend with any one of these factors: e.g., two circuits with comparable size (s3384 and c2670) have very different runtimes. Similar observations have also been made in [18, 65].

We categorize the circuits in Table 4.1 into two sets, named Set1 and Set2, based on the throughput improvements ($\Delta\eta$) obtained. Our method is generally successful on circuits in Set1 and not so on those in Set2. We will analyze this further, discuss the root causes, and present a technique in Section 4.7.3 for predetermining whether a circuit can benefit from the use of our methods (and in general, from VLUs).

Area and throughput analysis

All comparisons shown here are with respect to the padded baseline one-cycle circuit.
VLU sizing-based padding: An area overhead is induced due to (a) sizing and (b) additional hold logic. Although this method yields the highest throughput over all designs, the area overhead induced is also the highest, similar to or even greater than the baseline. Since the baseline is a combinational design, T_{clk} for the baseline is always greater than that of VLUs, as demonstrated for the RCA in Section 4.1.1.
Hybrid padding: The sizing overhead of this method is due to the insertion of hold logic (Section 4.7.1): in some circuits, this is large enough that the net area overhead is positive.
Static MOHL VLU: The static MOHL incurs an average savings in area as compared to the worst-case design. In considering the throughput overhead, it is important to note

that $\Delta\eta$ for this method provably decreases monotonically with time, as more paths move to two-cycle operation with time. This is illustrated in Fig. 4.9.

Let us examine the $\Delta\eta$ values for the hybrid padding method and the static MOHL VLU design. For the Set1 benchmarks, the hybrid method shows only a small throughput degradation while the static method shows a positive or negative change. For the Set2 benchmarks, both methods show large overhead.

On analyzing this further, we determined that this is caused because the circuits in Set2 have either a zero or small margin between the delays of the near-critical paths and other paths in the circuit. This observation was also made for such circuits in [70] for MCNC circuits that proved to be hard to optimize for average-case operation. Thus, as we move along a particular column of the F/A grid of Fig. 4.6, the value of P_{hold}) changes rapidly from a small value (such as 0.05) to a large value (such as 0.99), implying that upon degradation, almost all paths are moved to second cycle, resulting in large negative values of $\Delta\eta$. This is also confirmed by the $\Delta\eta$ results of the adaptive MOHL VLU design (discussed next), where such circuits yield only a very small enhancement in throughput.

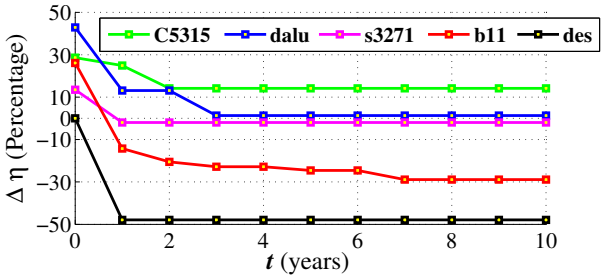


Figure 4.9: Variation of $\Delta\eta$ as a function of time t for static MOHL VLU design for a subset of the benchmark circuits.

Adaptive MOHL VLU: Empirically, we see that allowing the choice of $\mathcal{H}$-sets from different columns of the F/A grid, not only allows for throughput enhancement *throughout* the lifetime, but also requires fewer hold logics for correct functionality, and hence the largest area savings. We also see that only a small amount of V_{bb} (maximum 0.25V) is necessary for all the circuits. We also note that amongst all designs, adaptive MOHL achieves lowest area overhead (-9.1%), with throughput enhancements quite close to

the highest throughput enhancements of the sizing-based padded VLU. These negative overhead numbers are significant, for a lower area also requires lower power requirements.

For some cases in Set1 (C2670, cmb, i5, b11), although the hybrid scheme results in only a small throughput degradation, and the adaptive scheme results in good throughput enhancement, the static MOHL VLU gives a relatively higher throughput degradation. We observe that such circuits have a large number of paths with delays close to the maximum one-cycle path delay in the nominal VLU, and only a few paths with delays close to that of the critical-path delay of the circuit. With delay-degradation, static MOHL moves all such paths to second cycle, experiencing a higher throughput degradation. Since such circuits also have only a few paths with delays close to the critical-path delay of the circuit, they do not suffer much throughput degradation with hybrid design, and show significant throughput gain with adaptive MOHL design.

We can conclude from the above analysis, that for benchmarks in Set1, the static MOHL VLU scheme with ABB proves to be most suitable in ensuring BTI resilience with large savings in area, and significant gains in throughputs throughout lifetime. For benchmarks in Set2, combinational or VLU sizing schemes may perform as well as adaptive MOHL but much better than the static MOHL scheme.

4.7.3 Benchmark Categorization

Although our results have categorized benchmarks into Set1 and Set2 after analysis, it would be useful for a designer to be able to do so *a priori*, without having the need of constructing the F/A grid and of the subsequent analysis. We present a method for such categorization. For this method, we only need to work with the nominal combinational design of the circuit.

As highlighted in Section 4.7.2, at a given T_{clk}, for circuits that belong to Set2, numerous paths delays are close to the longest path delay, and these are moved to the second cycle as the circuit degrades with aging. We identify ϵ-critical paths in the nominal design: paths whose delay values lie within a fraction ϵ of becoming critical, i.e., delays in the interval $[\epsilon D_c, D_c]$, where $D_c = D(t = 0)$ is the critical path delay. We choose $\epsilon = (1 - \frac{\Delta D(t_{life})}{D_c})$, since paths that are within ϵ of D_c are likely to age so that their delays increase and violate T_{clk}. If the number of such ϵ-critical paths is large, we

will likely incur low gains, or losses, in throughput in the VLU-based designs.

A measure of this change in throughput can be computed as follows. Let η_1 be the throughput of the nominal circuit ($T_{clk} = D_c$), and let η_2 be the throughput if we set $T_{clk} = \epsilon D_c$. Using Equation (4.1):

$$\eta_1 = \frac{1}{D_c} \text{ and } \eta_2 = \frac{1}{(1 + P_{hold}) \cdot \epsilon D_c} \tag{4.5}$$

where the P_{hold} value is obtained by the hold logic generation algorithm in Section 4.1.2. The throughput change is then computed as:

$$\Delta\eta = \frac{\eta_2 - \eta_1}{\eta_1} = \frac{1}{\epsilon \cdot (1 + P_{hold})} - 1 \tag{4.6}$$

We choose $\Delta\eta_{tol}$ = -25% as the tolerance on the estimated percentage change in the throughput incurred (compared to nominal design) when the ϵ-critical paths are moved to the second cycle. If $\Delta\eta$ (from Equation (4.6)) $\geq \Delta\eta_{tol}$, the design is categorized in Set2. For such designs, the P_{hold} value, as expected, is seen to be quite large. We have found this choice to work well for all the benchmarks tested.

Note that percentage of ϵ-critical paths, with respect to the total number of paths in the circuit, is equal to $(1 - P_{hold}) \times 100$ (P_{hold} generated with $T_{clk} = \epsilon D_c$), for P_{hold} essentially is the fraction of total number of paths that have delays less than ϵD_c. This can be computed *in linear time* since the computational cost for determining such paths is the same as the hold logic generation algorithm: O(n).

Table 4.1: Area overhead and throughput comparisons of various designs for overcoming BTI degradation

Circuit	Sizing/ Padding		VLU/ Hybrid		Static MOHL VLU			Adaptive MOHL VLU				
	ΔA (%)	$\Delta\eta$ (%)	ΔA (%)	$\Delta\eta$ (%)	$\lvert\mathcal{H}\rvert$	ΔA (%)	$\Delta\eta$ (%)	$\lvert\mathcal{H}\rvert$	ΔA (%)	$\Delta\eta$ (%)	Max V_{bb}	CPU (min.)
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
Set1 Benchmarks												
c1908	0.4	36.6	-5.3	-3.8	4	-7.1	9.2	2	-13.1	37.5	0.25	8.83
c2670	7.8	18.1	2.2	-4.3	4	7.8	-34.1	1	-6.8	19.9	0.25	10.28
c5315	1.7	22.0	-5.9	-3.4	3	-2.1	16.5	3	-3.7	25.7	0.20	8.16
c7552	6.7	15.3	1.8	-9.3	4	-3.7	-6.0	2	-5.1	10.2	0.25	24.62
s344	-3.5	38.0	-9.0	-0.4	2	-10.3	10.8	2	-12.2	39.5	0.25	0.02
s635	0.8	61.6	-4.9	0.0	4	-1.2	30.0	1	-14.2	61.2	0.25	0.18
s1269	-0.6	14.3	-7.0	-7.1	2	-16.1	9.3	1	-16.1	13.7	0.05	0.21
s1512	-4.2	22.2	-9.9	-6.8	2	-17.2	1.2	1	-18.2	22.3	0.25	0.12
s3271	-2.6	13.5	-9.0	-1.5	2	-15.5	-0.6	3	-16.3	13.5	0.25	0.21
s3384	-1.3	61.0	-7.3	-1.6	4	-10.3	21.4	1	-15.8	59.5	0.25	0.87
cmb	4.6	34.7	-0.3	-0.2	4	7.9	-16.3	1	-6.1	31.8	0.20	0.03
lal	7.4	43.0	3.9	-0.1	4	15.2	-3.8	1	-0.4	37.4	0.25	0.05
ttt2	7.6	10.9	2.8	0.0	4	1.6	-9.1	1	-9.6	11.1	0.25	0.04
apex7	2.6	42.4	-4.6	-3.3	3	-10.2	35.3	3	-7.5	39.9	0.25	0.04
alu2	-1.1	40.2	-7.0	-0.3	3	-12.6	19.8	2	-14.6	36.5	0.15	0.08
i5	7.7	7.8	1.6	-5.8	3	-5.8	-13.9	2	-6.1	9.8	0.20	0.13
b11	9.3	26.1	2.3	-0.6	4	1.1	-22.7	3	-4.6	30.6	0.25	0.11
apex6	3.6	81.5	-1.9	0.0	2	-2.2	20.0	3	-0.8	75.0	0.20	0.11
alu4	-2.7	48.1	-8.5	-1.7	2	-15.7	12.4	2	-15.4	48.1	0.25	0.20
x3	0.5	85.1	-4.8	0.0	2	-7.8	12.7	3	-5.5	78.7	0.20	0.12
dalu	3.6	42.9	-0.6	0.0	3	5.8	7.2	3	-0.6	44.8	0.25	2.99
Avg.	2.3	36.4	-3.4	-2.4		-4.7	4.7		-9.1	35.6		2.73
Set2 Benchmarks												
s6669	-4.5	1.8	-10.4	-40.6	4.0	6.5	-43.6	3.0	-1.6	1.8	0.25	1.88
vda	-2.6	8.2	-6.7	-40.3	4.0	-2.9	-45.4	2.0	-11.9	0.6	0.25	0.04
des	-3.8	0.0	-8.7	-37.6	2.0	-15.6	-43.5	1.0	-15.6	0.0	0.25	0.06
t481	0.7	0.0	-3.7	-23.4	3.0	-2.6	-39.6	1.0	-14.2	0.0	0.25	0.10
Avg.	-2.6	2.5	-7.4	-35.4		-3.7	-43.1		-10.0	0.6		0.52
Overall	1.5	31.0	-4.0	-7.7		-4.5	-2.9		-9.2	30.0		2.38

Chapter 5

Variation-Aware Design of Variable Latency Units

In Chapter 4, we presented several schemes for BTI compensation using variable latency operation. VLUs are characterized by the property that the hold logic is highly dependent on distribution of delay amongst the various paths in the circuit. If the choice of one-cycle and two-cycle paths in a VLU is based on nominal delay estimates from the presilicon stage, it is quite possible that process variations during the fabrication can change this distribution and hence affect the functional correctness of VLUs. This can be overcome using pessimistic delay estimates; however, this would result in an inability to harness the best achievable throughput. In this chapter, we investigate this effect, and also propose a solution for constructing variation-aware variable latency designs.

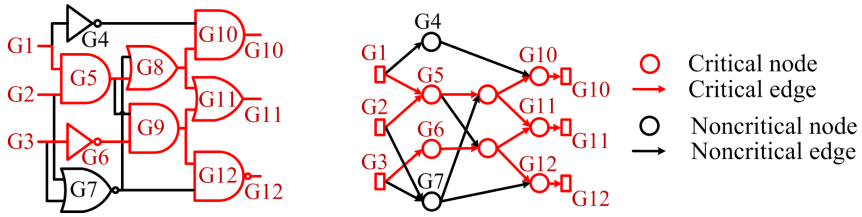
5.1 Preliminaries

Our discussion to follow will use some common terminologies and notions, that we summarize here for reference throughout this chapter:

- We use the term "variable latency" to denote that we allow a variable number of clock cycles (either one cycle or two cycles) for the operation of the circuit. On the other hand, the term "variations" is used to refer to changes in device parameters between one manufactured part and another during the fabrication process.

- Variations in parameters and also the resultant variations in delay-related quantities are modeled as Gaussians, as was highlighted in Section 2.1.3.

- For this chapter, we frequently use the directed acyclic graph (DAG) representation of a circuit (or a part of the circuit) to illustrate our concepts, as shown in Fig. 5.1. The gates/nodes are shown as vertices and interconnections as edges of the graph. Noncritical nodes and edges are marked as black whereas critical ones are marked in red. Primary inputs and primary outputs are also represented as separated nodes.



(a) Circuit (b) DAG

Figure 5.1: DAG representation of a circuit.

- We may refer to a circuit either in terms of its elements or its graph representation. We use the terms "gate" and "node" interchangeably to refer to any gate in the circuit, such as an inverter or a NAND2. Similarly, we use the terms "interconnection" and "edge" interchangeably to refer to a connection between the input and output ports of two gates/nodes in the circuit.

- We define the slack time, ST, at an output node of a gate as

$$\mathrm{ST} = \mathrm{RT} - \mathrm{AT}$$

  where RT is the Required Time and AT is the Arrival Time at the node. Similarly, we define the slack time ST for a path as the slack at the output of the final gate on the path. We will overload the notation by using ST[G] to refer to the slack time at the output port of gate G, and ST[P] to refer to the slack on path P.

- During our analysis, for simplicity, we assume that flip-flop setup times are zero. Nonzero setup times can be easily incorporated into path delay constraints and into the RT and ST information.

- Given a clock period specification, T_{clk}, for a chip, we refer to a path or a gate as being critical according to the following definitions:

  - For the deterministic case, a path is a critical path if delay[path] $> T_{clk}$. In the presence of variations, however, a path may be critical in some chips and noncritical in others. As a practical measure, we define *potentially critical* paths as those paths for which the $\mu + 3\sigma$ value of path delay distribution exceeds T_{clk}, where μ refers to the mean of the distribution, and σ to its standard deviation. Our analysis is based on Gaussian delay approximations, and therefore, this 3σ condition implies path noncriticality in 99.73% of the manufactured dies, which practically means that the path will never be critical in a manufactured part. Note that this assumption is especially reasonable since real path delay distributions are truncated Gaussians (e.g., their delays cannot be negative, although a full Gaussian allows for this possibility).
  - For the deterministic case, a gate is critical if ST[gate] < 0. Under process variations, a gate G is *potentially critical* if the $\mu - 3\sigma$ value of ST[G] is less than 0.
  - Throughout this chapter, in the context of variations, we use the term *critical paths (gates)* to refer to potentially critical paths (gates). However, when the context is deterministic (as with manufactured dies or with Monte Carlo simulations, where we know the exact delay values), this usage refers to paths (gates) that are deterministically critical.

- In the context of VLUs, we interchangeably refer to a critical path as a two-cycle path, and to a noncritical path as a one-cycle path.

5.2 The Impact of Variations on VLUs

As stated in Chapter 4, the approach for VLU generation, in the deterministic case, first determines the deterministic delays of all gates in the circuit at the presilicon stage. Next, an STA is performed with the given T_{clk} constraint at the POs of the circuit. Following this, all deterministically critical nodes and edges in the circuit are

marked, and the algorithm described in Section 4.1.2 is applied to generate the final hold logic.

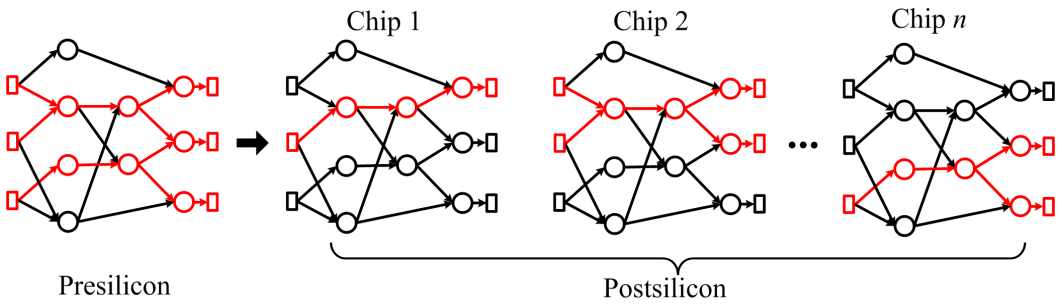


Figure 5.2: Delay distribution at the presilicon stage, predicted with corner-based or SSTA-based analysis , may be different from the actual delay distribution obtained at the postsilicon stage; this change may further vary from one chip to another.

The outcome of the hold logic generation algorithm relies strongly on the presilicon delay estimates of the circuit. In the presence of process variations, this delay distribution changes during fabrication. This is illustrated through Fig. 5.2, which shows the DAG representation of the circuit in Fig. 5.1(a) at the presilicon stage (under corner-based or SSTA-based analysis), and for n manufactured chips at the postsilicon stage. Any presilicon analysis is necessarily pessimistic: therefore, there may be paths that are flagged as two-cycle paths, due to the pessimistic estimates, that can operate in one cycle in a manufactured part. As a result, such manufactured parts are unable to achieve the best possible throughput.

In other words, to optimize the throughput for each specific part, we may need different hold logics for different fabricated chips. This motivates the need to develop a combined presilicon-postsilicon procedure for the generation of a hold logic (or a set of hold logics) that can be (a) functionally correct across all chips at the postsilicon stage, (b) ensure high throughput, and (c) incur low area and power overhead. We now present our scheme for the generation of such variation-aware hold logic.

5.3 Variation-Aware Hold Logic

We will now proceed to describe several approaches for designing variation-aware hold logic (VAHL), showing tradeoffs between the design overhead and the throughput benefits.

5.3.1 The Pessimistic Approach

The pessimistic hold logic described in Section 5.2 may be generated through either a corner-based or an SSTA-based presilicon analysis. Since the latter has reduced pessimism, through its ability to capture the effects of cancellations between less correlated within-die variations, an SSTA-based analysis is preferable. Using the results of such an analysis, we can generate a single hold logic by applying the hold logic generation algorithm in Section 4.1.2 to all potentially critical paths identified by SSTA.

5.3.2 The Enumerative Approach

We now present an alternative approach that is recognizably impractical, but will serve to motivate our eventual solution. Under this scheme, we build a separate hold logic corresponding to each of N critical paths in the circuit, $P_1, P_2, \ldots, P_N$, as identified by presilicon SSTA, using the algorithm in Section 4.1.2.

We use f_i to denote the hold logic for path P_i. If P_i is indeed critical in the manufactured part, then f_i is activated; otherwise, it is power-gated and f_i is set to logic 0. This operation is controlled by a sleep signal S_i that is set to 1 when f_i is to be power-gated, and is 0 otherwise. Fig. 5.3 illustrates this idea for an example where $N = 6$.

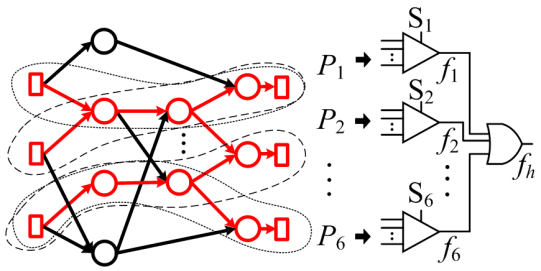


Figure 5.3: The enumerative approach for VAHL generation.

The generation of the sleep signals is based on the prediction of the postsilicon delays of each of these paths, and a crucial ingredient of this solution is to determine whether a path is critical or not in each manufactured part. We achieve this through postsilicon measurements on RO sensors. As discussed in Section 2.1.4, a set of k RO sensors on a chip is used to predict the real delay, D_{P_i}, of each path P_i, to determine whether it exceeds the specifications imposed by the clock period, T_{clk}. Based on the results of this determination, the values of f_i and S_i are set according to the criteria described above.

This method impractical for two related reasons. First, path enumeration methods can be prohibitively time-consuming. Second, since the number of paths can be very large, the hardware overhead, in terms of area and power, of all hold logics is also large.

5.3.3 A Clustered Approach for VAHL

The Concept

The two methods described earlier define two extremes: the pessimistic approach has low overheads since it generates a single hold logic, but sacrifices throughput; the enumerative approach achieves the best possible throughput, but may have a large overhead since the number of hold logics required, N, can be very large. We adopt a strategy inspired by the notion of the "Middle Way" [71] to find a solution for VAHL generation that is a happy medium between the two extremes.

A key observation that drives our approach is that since delay variations in a circuit are spatially correlated, the delays of some set of paths with spatial and/or structural similarity may be similar in magnitude and also highly correlated. If one of the paths in the set is identified as critical (or noncritical) in the postsilicon stage for a particular chip, it is very likely that all other paths in the set will have the same property. Instead of treating these paths separately, as in the enumerative approach, we choose to place them together in a single *path cluster*, with a combined hold logic that is driven by a single sleep signal.

Using this principle, the N critical paths in the circuit may be grouped into m path clusters, with m hold logics in all, which together will constitute the VAHL.[1] Each

[1] Note that grouping paths together in a cluster implies that the resulting hold logic is the logical OR of the individual hold logics. This provides for the possibility of logic minimization, and the hold logic for the path cluster typically involves significantly less hardware that the aggregate of hold logics for

path cluster, denoted by C_p, will contain a subset of all N critical paths, such that none of the N critical paths is left unclustered. Further, by definition, each of these clusters will have paths whose delays are similar and correlated, and will vary in the same way in different chips. We will discuss computationally efficient algorithms for the generation of path clusters in Sections 5.4 and 5.5.

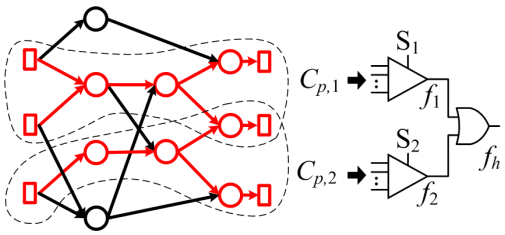


Figure 5.4: The clustering approach for VAHL generation, based on identification of path clusters.

This scheme can be illustrated through Fig.5.4, which shows $m = 2$ separate path clusters for the circuit of Fig.5.3. Consequently, only $m = 2$ hold logics are required instead of $N = 6$ hold logics in the enumerative case in Fig.5.3. Based on this clustered scheme, we make the following observations:

1. Compared to the pessimistic extreme, where $m = 1$, this approach has VAHL with $m \geq 1$ outputs. In general, whenever $m > 1$, this approach will involve less pessimism than the pessimistic approach outlined above, and hence result in higher throughput.

2. Compared to the enumerative extreme, where $m = N$, the number of outputs m in this VAHL will be much smaller (as will be shown in Section 5.6), reducing the overhead greatly. However, this VAHL will be pessimistic over a cluster, i.e., it will activate the hold logic for the cluster if any one path within a cluster is critical. Therefore, it is more pessimistic than the enumerative approach.

In other words, the results are indeed consistent with the concept of the Middle Way, with the number of hold logics as well as the throughput being somewhere between the

individual paths. However, VAHL *does not* allow any logic sharing amongst the hold logics of different path clusters, since we need the ability to put some of these hold logics to sleep if the corresponding path cluster is noncritical.

corresponding numbers for the enumerative case and the pessimistic case.

The Implementation

We now present a more systematic description of the our clustered approach through Algorithm 1, including further details for generating the select signals. The procedure consists of a presilicon stage (lines 1 through 6) where path clusters are generated, followed by a postsilicon stage (lines 7 through 12) where the criticality of the path clusters is determined.

Algorithm 1: VAHL Generation Framework

```
   /* Presilicon Stage                                                    */
   Input: T_clk, Circuit: a levelized circuit
 1 Perform SSTA
 2 {C_p,1, C_p,2, ..., C_p,m} = GeneratePathClusters(Circuit)
 3 for each path cluster C_p,i, 1 ≤ i ≤ m do
 4     Compute f_i = hold logic for C_p,i
 5     D^SSTA_{C_p,i} ← maximum delay {∀ paths in C_p,i}
 6 Determine d_t = [d_1, d_2, ..., d_k]
   /* Postsilicon Stage                                                   */
 7 for each manufactured chip do
 8     d_t = d_r from measurements of RO sensors
 9     for each path cluster C_p,i, 1 ≤ i ≤ m do
10         Compute D^cond_{C_p,i} = PDF (D^SSTA_{C_p,i} | d_t = d_r)
11         D^r_{C_p,i} ← (μ + 3σ) of D^cond_{C_p,i}
12         (D^r_{C_p,i} > T_clk) ? { S_i ← 0 } : { S_i ← 1; f_i ← 0 }
```

At the presilicon stage, given a T_{clk} specification at the POs of the circuit, we first perform an SSTA (line 1), and generate m path clusters $\{C_{p,1}, C_{p,2}, ..., C_{p,m}\}$ (line 2) using a procedure that will be described in Sections 5.4 and 5.5 through Algorithm 2. For each of these path clusters, $C_{p,i}$, we then evaluate the corresponding hold logic (line 4) using the hold logic generation algorithm presented in Section 4.1.2. We also compute, in canonical SSTA form [6], the delay PDF of the maximum delay over all paths in the path cluster, $D^{SSTA}_{C_{p,i}}$ (line 5). Finally, we determine, also in canonical

form, the statistical presilicon delay vector $\mathbf{d_t} = [d_1, d_2, ..., d_k]$ of the k RO-sensors (test structures) on the chip, described in Section 2.1.4 (line 6).

At the postsilicon stage, we aim to identify which of the m path clusters will be critical in a particular chip, so that we can decide appropriate subset of these that should be active on that chip. We therefore perform postsilicon measurements for the RO sensors (line 8) for each manufactured chip to obtain the resultant RO sensor delay vector sample, $\mathbf{d_t} = \mathbf{d_r}$. This information is used to predict the postsilicon delays of each path cluster $C_{p,i}$ based on a conditional PDF evaluation [8], $D^{cond}_{C_{p,i}}$, of $C_{P,i}$ (line 10); this corresponds to the narrowed PDFs illustrated earlier in Fig. 2.5. Next, the real delay, $D^{r}_{C_{p,i}}$, of $C_{p,i}$ is estimated (line 11) to be the $(\mu + 3\sigma)$ point of its conditional PDF. In practice, this value is very close to its mean, as the conditional PDFs have a small variance [8].

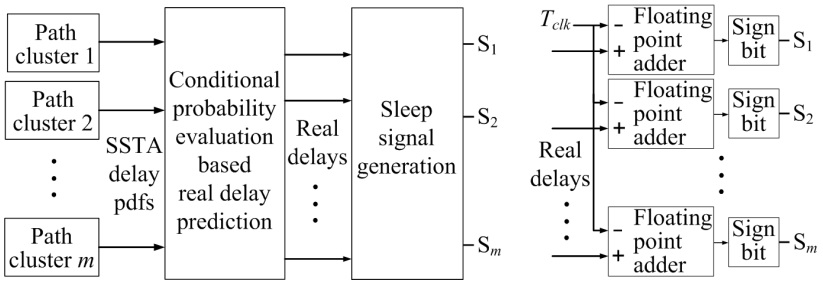


Figure 5.5: Postsilicon processing for determining the sleep signal values: (a) overall flow, and (b) hardware for sleep signal generation.

Having estimated the real delays, we then determine (line 12) if $C_{p,i}$ is critical in the specific manufactured chip by comparing the values of T_{clk} and $D^{r}_{C_{p,i}}$, as depicted in Fig. 5.5(a). If the estimated delay is larger than the clock period, then the hold logic is left active; otherwise it is put to sleep and the corresponding f_i is set to zero. The hardware implementation of this operation, shown in Figs. 5.5(b), performs a floating point addition of $D^{r}_{C_{p,i}}$ and the 2's complement of T_{clk}, and checks the sign bit of the result. If this sign bit is 0, then the clock period has been exceeded; otherwise not.

We evaluate the overhead required to generate the sleep signals. All postsilicon steps described above are a one-time computation for each manufactured part. The

conditional PDF evaluation for all m path clusters can be performed by a simple function; the runtime for one such evaluation is reasonable [8]. Practically, as we will find in Section 5.6, the number of path clusters generated by our scheme is typically less than 8. The calculation for sleep signal generation described in line 12 is also only a one-time computation that can leverage hardware (hardware comparators or adders, as well as registers) that already exist on-chip in many designs. Hence, very little extra hardware is required for generation of these sleep signals, other than a few multiplexers to appropriately route data to these units.

5.4 Enabling Practical Path Clustering

5.4.1 Qualitative Criteria for Path Clustering

Qualitatively, a set of paths in the circuit should be clustered together, with the hold logic controlled by a single sleep signal, if they all have a high probability of together being critical/noncritical after fabrication. This means that they should experience similar shifts in their delays from similar mean values. The similarity of mean values is an important consideration, as depicted in Fig. 5.6, which shows the PDFs of the delays D_{P_i} and D_{P_j} of paths P_i and P_j. Both PDFs are observed to have a high correlation, but since their mean values differ significantly, in most manufactured dies, path P_j may have $D_{P_j} < T_{clk}$, whereas the path P_i may have $D_{P_i} > T_{clk}$.

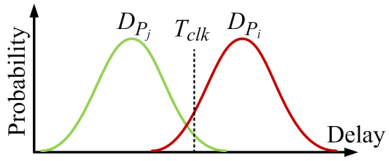


Figure 5.6: Importance of comparing the mean of the two path delays.

We therefore define the *path closeness* of any two paths based on two criteria: (a) the correlation between the path delays, and (b) the mean values of the path delays. Based on this closeness metric, if any two paths are "close enough", they can be clustered together. We can then further grow this cluster by testing other paths for their closeness with these two paths.

5.4.2 Reducing the Expense of Path Clustering

The direct use of a path closeness metric is impractical: in order to determine the closeness of N critical paths in the circuit in a pairwise manner, we will have to perform $O(N^2)$ pairwise comparisons; moreover, recall that N can be very large since it involves a form of path enumeration.

This leads us to the need of a procedure to reduce the computational expense associated with path clustering. In pursuit of this, we first coarsen the graph by generating "node clusters" in polynomial time. This coarsening step effectively reduces the number of paths to be enumerated to a practical number. As a result, the quadratic complexity for pairwise comparisons constitutes a reasonable computational overhead.

Algorithm 2: GENERATEPATHCLUSTERS

```
/* Algorithm for generating path clusters                    */
Input: Circuit: a levelized circuit
Output: L_Cp: list of m path clusters C_p,1, C_p,2, ..., C_p,m
1 L_Cn = GENERATENODECLUSTERS(Circuit)
2 return L_Cp = NODETOPATHCLUSTERS(L_Cn)
```

We now describe, at a high level, our path clustering scheme in the function in Algorithm 2 (recall that this was invoked by Algorithm 1 earlier). Given a levelized circuit, the algorithm generates and returns the list of path clusters, L_{C_p}. In our first step (line 1), we call a function, to be described in Algorithm 3, to generate a list of node clusters, L_{C_n}, for all critical nodes of the circuit. This step will be shown to be performed in a block-based manner. Next, we use these node clusters in L_{C_n} in line 2 to extract the list of m path clusters, L_{C_p} through a function to be described in Algorithm 4, such that this step requires minimal enumeration.

5.4.3 Node Cluster Generation: Node Closeness Metric

Formally, a node cluster refers to a cluster of critical nodes and critical edges of the original circuit (or a critical connected subgraph of the original circuit graph) under a specified *node closeness* metric.[2] The inputs of this connected subgraph come either

[2] This concept is somewhat similar to (but not the same as) the concept of "supergate" formation, where groups of adjacent gates in a circuit are fused into a single supergate [72, 73], except that the criteria for clustering are different.

from the critical PIs or from the outputs of some other node cluster, and whose outputs go either to the critical POs or to the inputs of some other node cluster. The node cluster is abstracted as a set of input-to-output connections connected by *node cluster arcs*; by definition, a node cluster arc represents a subpath of some critical path of the circuit. An illustration of a node cluster is shown in Fig. 5.7.

The inputs of this connected subgraph come either from the PIs or from the outputs of some other node cluster, and whose outputs go either to the critical POs or to the inputs of some other node cluster. The node cluster is abstracted as a set of input-to-output connections connected by *node cluster arcs*; by definition, a node cluster arc represents a subpath of some potentially critical path of the circuit. An illustration of a node cluster is shown in Fig. 5.7.

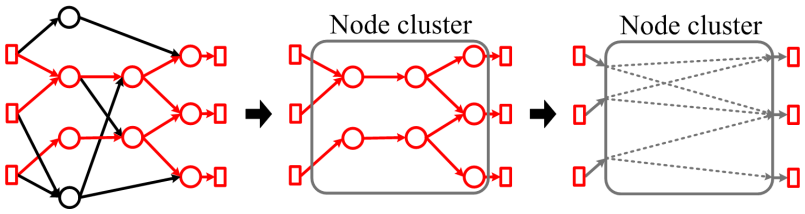


Figure 5.7: The concept of a node cluster. The dotted lines inside the node cluster represent the node cluster arcs.

We first begin with a definition of *node closeness* of two critical nodes, which if sufficiently high, allows for the two nodes to be clustered together. Since our goal is to reduce path clustering computation by development of node clusters (line 2 of Algorithm 2), this definition should be constructed in a way that, although the metric is defined for nodes, it also somehow reflects on the closeness of the set of paths that pass through these two nodes. In other words, the node closeness metric must capture information about path delays of all critical paths passing through the node.

In this context, we define the metric $\mathcal{M}$ for the output port of a critical gate G as:

$$\mathcal{M}[\mathrm{G}] = \mathrm{AT}[\mathrm{G}] + (T_{clk} - \mathrm{RT}[\mathrm{G}]) = T_{clk} - \mathrm{ST}[\mathrm{G}] \tag{5.1}$$

where, as usual, $\mathrm{ST}[\mathrm{G}] = \mathrm{RT}[\mathrm{G}] - \mathrm{AT}[\mathrm{G}]$ is the slack time at the output of gate G, and RT[G] and AT[G] are respectively, the required time, and arrival time at the output of G.

The term, AT[G], represents the statistical maximum of the delays of all paths from PIs up to G, while the term, $(T_{clk} - \text{RT[G]})$, represents the statistical maximum of the delays of all paths from G upto the POs. Therefore, $\mathcal{M}$[G] captures the statistical maximum delay over all paths passing through G.

It is important to note that unlike the deterministic case, in which $\mathcal{M}$[G] will be able to capture *only* the longest path passing through G, in the statistical case, this formulation captures delays over *all* potentially critical paths through G.

We now use this metric to define the closeness between two nodes, node_i and node_j, in terms of (a) the correlation of $\mathcal{M}$, and (b) the mean value of $\mathcal{M}$ for the two nodes. A high correlation of $\mathcal{M}$ for the two nodes implies that the maximum delays of paths passing through node_i and node_j may also be well correlated. Further, similar mean values of $\mathcal{M}$ implies that these paths may also have similar mean values. This is important in the light of our previous discussion in Section 5.4.1.

We now mathematically formulate this node closeness as:

$$\text{Closeness } \mathcal{C}\big(\text{node}_i, \text{node}_j\big) = 1, \ \textit{if} \tag{5.2}$$

$$\rho\big(\mathcal{M}[\text{node}_i], \mathcal{M}[\text{node}_j]\big) \geq \rho_{th}, \tag{5.3}$$

$$\text{and, } f_\mu\big(\text{node}_i, \text{node}_j\big) \geq f_{\mu,th} \tag{5.4}$$

where ρ is the correlation coefficient, ρ_{th} and $f_{\mu,th}$ are user defined thresholds, based on the degree of pessimism permitted, and

$$f_\mu\big(\text{node}_i, \text{node}_j\big) = \frac{min\big\{\mu(\mathcal{M}[\text{node}_i]), \mu(\mathcal{M}[\text{node}_j])\big\}}{max\big\{\mu(\mathcal{M}[\text{node}_i]), \mu(\mathcal{M}[\text{node}_j])\big\}} \tag{5.5}$$

In words, the node closeness formulation in Equation (5.2) states that two nodes are close enough to be clustered if:

1. the correlation between $\mathcal{M}$ for the two nodes is above a certain threshold, as captured by Equation (5.3).

2. the mean value of $\mathcal{M}$ for the two nodes are similar.

The similarity between the means is captured by the f_μ formulation in Equation (5.5): the ratio of the minimum and maximum $\mathcal{M}$ mean values for node_i and node_j provides an estimate how far apart these mean values are from each other. Note that both the

correlation coefficient and the ratio of the lesser to the greater mean lie between 0 and 1.

Based on the above formulation, we make the following observations:

1. If we set $\rho_{th} = f_{\mu,th} = 0$, the conditions in Equations (5.3) and (5.4) will always be satisfied: the minimum correlation between any two $\mathcal{M}$ values can be 0, and the mean of $\mathcal{M}$ can have a minimum value of 0 (since $\mathcal{M}$ represents path delay, which cannot be negative). Hence, all critical nodes are forced to be clustered in a single node cluster. This reaches to one extreme case of pessimism discussed in Section 5.3.

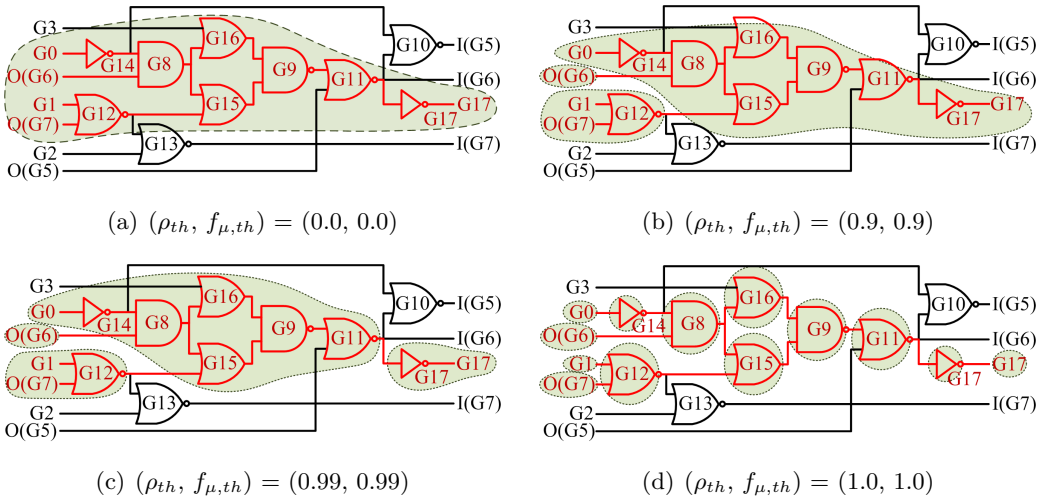
2. For $\rho_{th} = 1$ and $f_{\mu,th} = 1$, the condition of Equation (5.3) will be most likely violated by every pair of gates in a practical circuit, as due to variations, it is very unlikely that two nodes will have $\mathcal{M}$ to be exactly of the same value on the same die. For the same reason, the condition in Equation (5.4) is highly likely to be violated. No two nodes, therefore, may be allowed to be clustered together, and each node cluster may become a node cluster in itself, as will be illustrated shortly. This reaches to the other extreme case of path enumeration discussed in Section 5.3.

3. Any value in (0,1), therefore, for both ρ_{th} and $f_{\mu,th}$ corresponds to the middle way solution with different degrees of pessimism. We elaborate in more detail on the considerations for an appropriate choice of threshold values in Section 5.6.

These observations can be more concretely depicted by the results of the application of this algorithm on an ISCAS89 benchmark circuit, s27, as shown in Fig. 5.8, with gate delays as used in *MinnSSTA* [7]. With T_{clk} = 84.3ps and four different set of values for (ρ_{th}, $f_{\mu,th}$) = (0.0, 0.0), (0.9, 0.9), (0.99, 0.99), and (1.0, 1.0). We observe that:

1. With (ρ_{th}, $f_{\mu,th}$) = (0.0, 0.0), all critical gates are clustered into a single node cluster.

2. With (ρ_{th}, $f_{\mu,th}$) = (0.9, 0.9), the pessimism decreases, and the result is 2 node clusters in all. Node G12 and its inputs have a different $\mathcal{M}$[G12] mean than of all other critical nodes, and thus clustered into a separate node cluster.



(a) $(\rho_{th},\ f_{\mu,th}) = (0.0,\ 0.0)$

(b) $(\rho_{th},\ f_{\mu,th}) = (0.9,\ 0.9)$

(c) $(\rho_{th},\ f_{\mu,th}) = (0.99,\ 0.99)$

(d) $(\rho_{th},\ f_{\mu,th}) = (1.0,\ 1.0)$

Figure 5.8: Results of node cluster generation for ISCAS89 benchmark s27, for four different values of $(\rho_{th},\ f_{\mu,th})$.

3. As the threshold values reach closer to 1.0, the pessimism further decreases and we obtain 4 node clusters with $(\rho_{th},\ f_{\mu,th}) = (0.99,\ 0.99)$.[3]

4. When both thresholds are set to 1.0, we see that each node in itself becomes a node cluster, confirming our earlier observation.

5.4.4 A Block-Based Algorithm for Node Cluster Generation

Based on the above metric and a measure of node closeness, we now illustrate our block-based nonenumerative procedure for generating node clusters through an example. We then formalize this procedure into an algorithm.

Our approach iteratively grows a cluster by topologically traversing the circuit graph backwards from the POs to the PIs. An atomic operation consists of examining a node G, that already lies in a particular node cluster (as illustrated by an example in Fig. 5.9), and determining whether the fanins of G may be included into this node cluster, i.e., whether they are close enough to G, based on Equation (5.2). We therefore compute the closeness between G and each of its fanin nodes, F1, F2, and F3, to check which

[3] As s27 is a small circuit, such high thresholds result in only 4 node clusters; with larger circuits, the variations are enhanced, and results in large overhead as will be shown in Section 5.6.4.

of the fanins, if any, satisfy the closeness criterion. In this example, both F2 and F3 satisfy the criterion, and the node cluster is grown to include F2 and F3 in the cluster. Next, the most recently added nodes are compared with their as-yet-unclustered fanins, and the process continues until the cluster grows no further. Any inputs that could not be added to the current cluster are used to seed new clusters, and the method continues until all clusters have been grown.

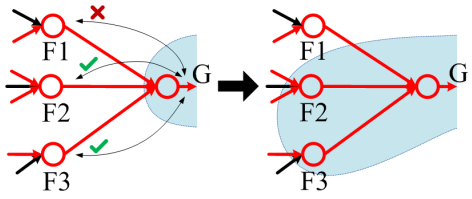


Figure 5.9: Illustration of node cluster growth from a particular node in the circuit, which is already in a cluster.

We note that this step of node cluster growth includes nodes at only two levels: the level l at which the node G being processed is located, and the level $l-1$ at which its fanin nodes are located. Hence, this computation can be very easily performed using a block-based manner over all nodes in the circuit, with computation time that is linear in the number of gates in the circuit.

This procedure is a heuristic and we do not claim it to be exact or optimal. For instance, for a node at level l, we may instead test the fanout nodes at level $l+1$ for closeness, and grow the cluster. Second, our cluster growth approach is based on comparisons between a gate output and its fanin nodes, but not between the fanin nodes and existing nodes in the cluster. In general, if node n_1 is close to n_2 and n_2 is close to n_3, there is no guarantee that transitivity applies, making n_1 close to n_3. Therefore, it is possible that our method may cluster nodes more than necessary: the consequence of this is a loss in throughput. As will be shown in Section 5.6, this loss is not significant under this fast heuristic.

Algorithm 3 presents a formalized description of the node cluster generation function for a circuit: this function was called earlier by the Algorithm 2. Given a levelized circuit, the function generates a list of all node clusters of the circuit, along with its input-output connections. For simplicity, a node cluster is denoted as C_n in the algorithm. It first

Algorithm 3: GENERATENODECLUSTERS

```
   /* Algorithm for generating node clusters                              */
   Input: Circuit: a levelized circuit without node clusters
   Output: L_{C_n}: list of node clusters with input-output connections
   // Initialize node clusters with the POs of the circuit
 1 K ← number of POs in the circuit
 2 L_{C_n}.Clear() // Initialize as an empty list
 3 for each critical unclustered PO_i, 1 ≤ i ≤ K do
 4     C_n ← new node cluster initialized with PO_i
 5     L_{C_n}.Insert(C_n)
 6     PO_i ← output connection of C_n
 7     for each critical unclustered PO_j, i < j ≤ K do
 8         if closeness(PO_i, PO_j) = 1 then
 9             C_n.Insert(PO_j)
10             PO_j ← output connection of C_n
   // Node cluster growth, beginning with POs
11 l ← number of topological levels in the circuit
12 G.visited ← 0 ∀ critical nodes G
13 for each critical node G at level l ≥ 1 do
14     if G.visited = 0 then
15         G.visited ← 1
16         if G is not already clustered then
17             C_n ← new node cluster with G
18             L_{C_n}.Insert(C_n)
19             G ← output connection of C_n
20         else C_n ← G's node cluster
21         for each critical fanin F of G do
22             if F ∈ C'_n ≠ C_n then
23                 F ← input connection of C_n, output connection of C'_n
24             else
25                 if closeness C(G, F) = 1 then
26                     C_n.Insert(F)
27                     if F is critical PI then
28                         F ← input connection of C_n
29                 else if all critical fanouts of F have been visited then
30                     C'_n ← new node cluster with F
31                     L_{C_n}.Insert(C'_n)
32                     F ← input connection of C_n, output connection of C'_n
33     l = l − 1;
34 return L_{C_n}
```

begins with the critical POs of the circuit and cluster them into node clusters based on their closeness $\mathcal{C}$ in lines 1 to 10.

After this initialization, the function then grows the existing clusters in lines 11 to 33. First, all unclustered critical nodes are initialized as unvisited in line 12 (a node is marked visited to indicate that it has been processed for node cluster growth). The algorithm then repeatedly picks up each unvisited critical node (only once), including the POs, in an existing node cluster, in a reverse topological manner (from POs to PIs) in lines 13 to 20, and initializes a new node cluster with it if the node is not already clustered. It then examines all its unclustered, critical fanin nodes in lines 21 to 33, to check if they can be included in this node cluster by performing the closeness test in line 26, effectively growing the cluster. Along with such computation, the external input and output connections of the node clusters are also updated. An input/output connection for a node cluster is created when some of its nodes are connected to either a critical PI or a critical PO, or to a critical gate in some other node cluster.

The complexity of Algorithm 3 is $O(K^2 + N_g)$: K was defined as the number of POs in Algorithm 3, and N_g are the number of gates in the levelized circuit. The first term, K^2 comes from performing the pairwise closeness test of all K POs. The second term represents the complexity of visiting each node in the levelized circuit in a reverse topological manner.

5.5 Generating Path Clusters and VAHL

Having presented the theory of node clusters in Section 5.4, we now present the link between node clusters and path clusters in this section.

5.5.1 The Relation Between Node Clusters and Path Counts

Our starting point is a coarsened circuit, illustrated in Fig. 5.10(a), with node clusters, where the blocks represent node clusters as in Fig. 5.7. These node clusters are externally connected to other node clusters through their input-output ports (determined by Algorithm 3), through external interconnections between these node clusters, which are simply a subset of all interconnections of the original uncoarsened circuit (the rest of the interconnections are present within the node clusters). If we begin from the PIs, we

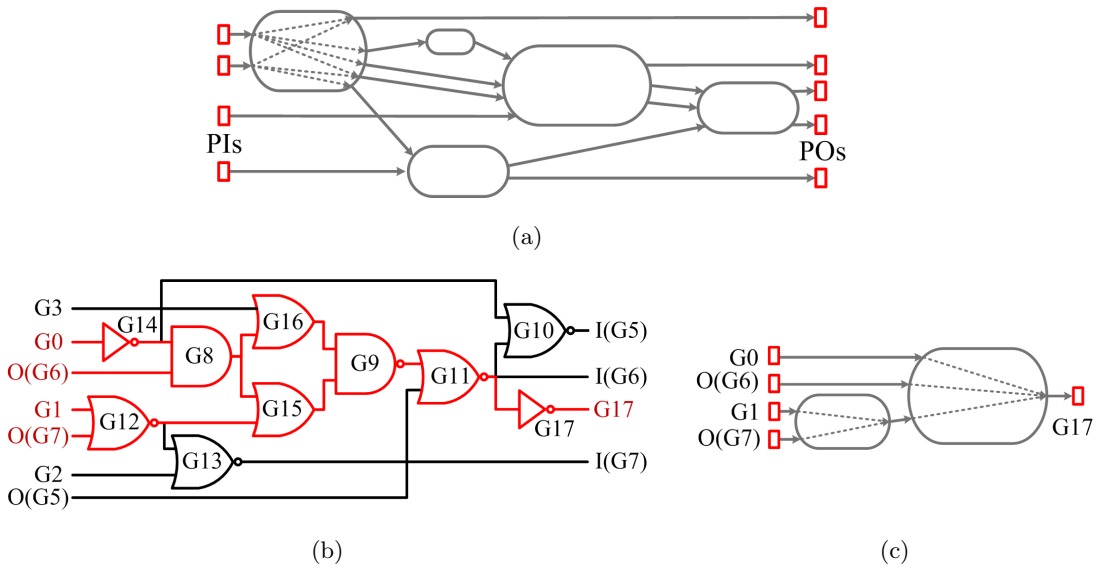


Figure 5.10: (a) General structure of a coarsened circuit, (b) uncoarsened s27 circuit, and (c) the coarsened s27 circuit.

can traverse the coarsened circuit along the external connections of the node clusters to reach to the POs.

As an example, Figs. 5.10(b) and 5.10(c) show the uncoarsened and coarsened s27 circuits, respectively, with T_{clk} = 84.3ps as in Section 5.4.3, and (ρ_{th}, $f_{mu,th}$) = (0.9, 0.9) chosen as thresholds for clustering. Recall that all nodes and edges of the coarsened circuit are critical, since node clusters only include critical nodes.

Each internal arc of a node cluster from an input port I_k to an output port O_l of that node cluster, captures in itself, potentially many (≥ 1, to be more specific) partial paths from I_k to O_l of the uncoarsened circuit, that lie within the node cluster.[4] A path, $P_{c,i}$, in the coarsened circuit, being constituted by multiple such arcs, therefore, encapsulates $1 \leq n_i \leq N$ critical paths of the uncoarsened circuit in itself, where N is the total number of critical paths in the uncoarsened circuit.

Therefore, each path, $P_{c,i}$, in the coarsened circuit is a cluster of n_i critical paths of the uncoarsened circuit, with $1 \leq n_i \leq N$.

[4] Every node cluster input may not necessarily have an arc to every node cluster output. As shown in Fig. 5.10(a), the node cluster, with its arcs visible, has two inputs connected to only a subset of the 5 outputs.

If each of the arcs lying on $P_{c,i}$ encapsulates only one partial path of the uncoarsened circuit, it implies that $P_{c,i}$ is a path cluster that contains only one critical path of the uncoarsened circuit, resulting in $n_i = 1$. This was observed to occur in Fig. 5.8 for all paths $P_{c,i}$ with one of the extremes: $\rho_{th} = 1$ and $f_{\mu,th} = 1$: each node in the uncoarsened circuit becomes a node cluster in itself. For the other extreme, $\rho_{th} = 0$ and $f_{\mu,th} = 0$, there exists only one node cluster, as discussed in Section 5.4.3, and the number of paths is upper-bounded by the product of the number of PIs and the number of POs.

In other words, the number of paths in the node clusters is no more than, and often substantially less than, the original number of paths, N. For our example of the small s27 circuit shown in Fig. 5.8, this corresponds to reducing 6 critical paths in the uncoarsened circuit to 4 paths of the coarsened circuit. This reduction seems to be small since s27 itself is a small circuit; in larger circuits, this reduction is also large, as will be shown in Section 5.6.

5.5.2 Path Clustering

The number of such coarsened circuit paths in the node-clustered circuit, even if they are fewer than those in the uncoarsened circuit, are still seen to be appreciably large in many circuits. This can result in a large area and power overhead of the hold logics. In such a case, we proceed further to apply our second step in the reduction of the number of coarsened circuit paths, reducing also the number of hold logics that must be generated: if the delays of any two such paths are close enough, then they can be further clustered together.

For this purpose, we can now use and mathematically formulate a path closeness metric, similar to that introduced earlier in Equation 5.2, for two *paths* $P_{c,i}$ and $P_{c,j}$ in the coarsened circuit:

$$\text{Closeness } \mathcal{C}\big(P_{c,i}, P_{c,j}\big) = 1, \ \textit{if} \tag{5.6}$$

$$\rho\big(\text{delay}[P_{c,i}], \text{delay}[P_{c,j}]\big) \geq \rho_{th} \tag{5.7}$$

$$\text{and } f_{\mu}\big(P_{c,i}, P_{c,j}\big) \geq f_{\mu,th} \tag{5.8}$$

where

$$f_{\mu}\big(P_{c,i}, P_{c,j}\big) = \frac{min\big\{\mu(\text{delay}[P_{c,i}]), \mu(\text{delay}[P_{c,j}])\big\}}{max\big\{\mu(\text{delay}[P_{c,i}]), \mu(\text{delay}[P_{c,j}])\big\}} \tag{5.9}$$

In practice, given the reduction in the number of the paths as we move from the original circuit to the coarsened circuit, it is practical to enumerate the paths without much computational overhead and to perform pairwise comparisons to cluster them.[5]

Algorithm 4: NODETOPATHCLUSTERS

```
   /* Algorithm for generating path clusters from node clusters      */
   Input: L_{C_n}: list of node clusters
   Output: L_{C_p}: list of m path clusters C_{p,1}, C_{p,2}, ..., C_{p,m}
   // Node Coarsening:  enumeration of coarsened circuit paths
 1 for each node cluster C_n ∈ L_{C_n} do
 2     K ← number of PIs connected to C_n
 3     for each PI_i connected to C_n, 1 ≤ i ≤ K do
 4         Enumerate paths using DepthFirstTraversal(L_{C_n}, PI_i)
   // Path Clustering
 5 N_{C_p} ← total number of paths generated above
 6 L_{C_p}.Clear(); // Initialize as an empty list
 7 for each unclustered path P_{c,i}, 1 ≤ i ≤ N_{C_p} do
 8     C_p ← new path cluster with P_{c,i}
 9     L_{C_p}.Insert(P_{c,i})
10     for each unclustered path P_{c,j}, i < j ≤ N_{C_p} do
11         if closeness C(P_{c,i}, P_{c,j}) = 1 then
12             C_p.Insert(P_{c,j})
13 return L_{C_p}
```

The path clustering step can therefore be formalized as shown in Algorithm 4, which was called earlier by Algorithm 2. The input to Algorithm 4 is the list of node clusters, L_{C_n}, and it returns a list of path clusters, L_{C_p}. First, all paths in the coarsened circuit are enumerated in lines 1 to 4 through a simple depth-first traversal of the coarsened circuit. Next, a pairwise comparison is performed between these paths in lines 5 to 12 to check if they can be clustered together by applying the closeness test in line 11, resulting in a greatly reduced number of clusters of coarsened circuit paths.

The application of this scheme is shown in Fig. 5.11 for s27 circuit. Path coarsening in the second step reduces 4 paths in the coarsened cicuit further down to 2 clusters

[5] Note that while this is computationally feasible in the presilicon phase, node clustering has only solved a part of the problem. Path clustering is still essential to reduce the hardware overhead of hold logic.

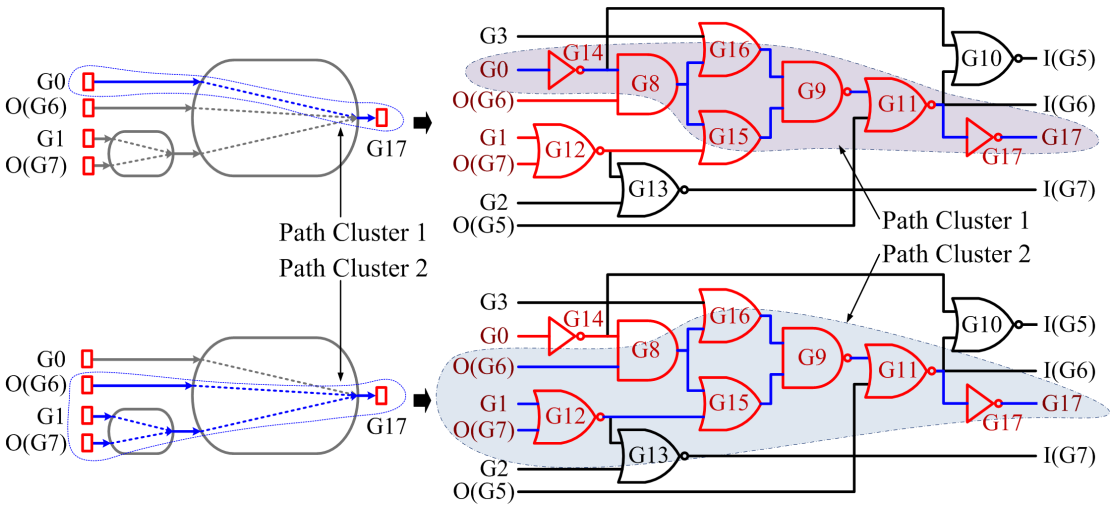


Figure 5.11: Path clusters generated in circuit s27.

of these paths. Hence node and path clustering allows for the number of hold logics to be reduced by 3× as compared to the enumerative scheme discussed in Section 5.3.2 (which generated 6 hold logics). This reduction will be seen in Section 5.6 to be much more in larger circuits.

5.5.3 Generation of VAHL

Having obtained the coarsened circuit path clusters, we now generate a separate hold logic corresponding to each of these path clusters, by applying the hold logic generation algorithm discussed in Section 4.1.2; each hold logic corresponds to the sensitization criterion of all paths within the path cluster.

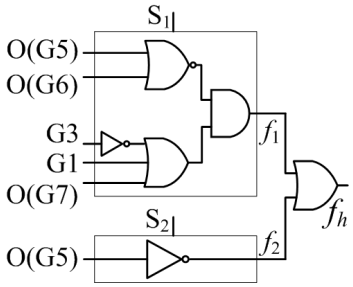


Figure 5.12: VAHL generated for circuit s27.

For our running example of circuit s27, the two-output VAHL along with the sleep signals is shown in Fig. 5.12. These sleep signals can be selectively exercised in the postsilicon stage for various dies.

5.6 Experimentation and Results

The proposed algorithms were implemented in C++, using the *MinnSSTA* [7] software for SSTA under 32nm PTM [11] models. The methods were exercised on the ISCAS89 benchmark circuits on a 3.0GHz CPU with 8GB RAM. The SSTA grid size for each circuit is taken from [6, 8]. Next, in Sextion 5.6.1, we provide a listing of all results, followed by a detailed discussion and analysis in Section 5.6.2.

5.6.1 Tabulation of Results

We present the results of applying the clustering approaches in Table 5.1. For convenience, we have marked the k^{th} column in the table as Ck. We first describe the details of how Table 5.1 is generated, and then analyze the results.

Baseline

For comparison purposes, we choose the pessimistic VLU design (Section 5.3.1, $\rho_{th} = f_{\mu,th} = 0$) as the baseline. Recall that the pessimistic VLU has only one hold logic, and identifies all potentially critical paths as two-cycle paths in the manufactured chip if the clock constraint is violated.

Clustering parameters

Columns C2 and C3 list the values of the threshold parameters, ρ_{th} and $f_{\mu,th}$, that are used in determining the closeness of nodes and paths. For comparison, we show two sets of results for each circuit:

- Using clustering (Section 5.3.3, $\rho_{th}, f_{\mu,th} \neq 0$)
- Using enumeration (Section 5.3.2, $\rho_{th} = f_{\mu,th} = 1$).

For clustering, the procedure for choosing ρ_{th} and $f_{\mu,th}$ (as tabulated) will be presented in Section 5.6.4.

Table 5.1: Results for VAHL under the clustered (ρ_{th}, $f_{\mu,th} \neq 0$) and enumerative (ρ_{th} = $f_{\mu,th} = 1$) approaches.

Circuit	ρ_{th}	$f_{\mu,th}$	Results from Clustering				VAHL Results				
			$\|L_{C_n}\|$	N	N_C	m	ΔA (%)	$\Delta P(\%)$		$\Delta\eta(\%)$	
								μ	σ	μ	σ
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
s27	0.87	0.89	4	12	8	2	11.7	6.7	5.3	12.0	11.5
	1	1	16		12	12	105.3	21.5	19.5	12.4	9.3
s1196	0.82	0.82	10	656	452	4	12.6	6.6	2.6	30.6	10.7
	1	1	337		656	656	109.2	41.2	22.5	38.5	9.5
s5378	0.94	0.93	7	520	67	8	-6.3	-4.4	4.1	27.1	23.9
	1	1	157		520	520	47.5	22.5	27.5	29.5	20.5
s9234	0.88	0.85	12	119556	297	3	8.5	2.2	1.3	7.8	11.2
	1	1	1368		119556	–	–	–	–	–	–
s13207	0.88	0.89	23	189666	1481	2	0.7	-0.3	0.2	26.6	6.8
	1	1	1126		189666	–	–	–	–	–	–
s15850	0.9	0.87	3	2.6e7	624	2	1.6	-0.9	0.4	2.6	0.6
	1	1	1383		2.6e7	–	–	–	–	–	–
s35932	0.95	0.95	9	174	59	4	-0.6	-0.2	0.2	1.1	0.8
	1	1	1185		174	174	100.5	37.4	28.1	1.2	0.9
s38417	0.87	0.85	6	38337	302	5	5.0	1.0	1.8	19.6	5.6
	1	1	1513		38337	–	–	–	–	–	–
s38584	0.94	0.96	4	12172	16	3	0.3	-0.1	0.1	22.4	8.3
	1	1	383		12172	–	–	–	–	–	–
Average	Clustered						3.7	1.2	1.8	16.7	8.8
	Enumerative*						90.6	30.6	24.4	20.4	10.0

* Enumerative computations are prohibitive for larger circuits; hence the average for this scheme is taken only over circuits which allow this.

Clustering results

The results for our clustering algorithms are shown in C4–C7. Column C4 presents the number of node clusters, $|L_{C_n}|$, generated in Algorithm 3. Columns C5 and C6 list, respectively, the number of potentially critical paths, N, in the uncoarsened circuit[6] and the number of paths, N_C, in the coarsened circuit. We then list the number of coarsened circuit path clusters, m, generated in Section 5.5.2, in C7. Recall that for m path clusters, we need m separate hold logics.

Computations for enumerative approach become prohibitive as the number of paths

[6] While the critical path enumeration can be exponential in the number of nodes, the number of critical paths can be computed in linear time.

increases, and hence the results could not be obtained for circuits larger than s5378.

VAHL area, power, and throughput

In C8 to C12, we show the results for overhead in area and power (denoted as ΔA (%) and ΔP (%), respectively) as compared to the baseline. These changes arise due to a higher number of hold logics, and the throughput enhancements (denoted as $\Delta\eta$ (%)) achieved due to less pessimism, for both the clustered and enumerative approaches.

Further, we recall that for the clustered and enumerative approaches, only a subset of m hold logics will be active in postsilicon stage in a particular chip. Therefore, although ΔA remains the same for all chips, ΔP will be proportional to the area overhead of the active hold logics in a particular chip, and may differ for different chips. ΔP can therefore be captured as a distribution over L samples of Monte Carlo simulations. Choosing $L = 10000$ as in [6], we tabulate the mean (μ) and standard deviation (σ) of this distribution in C9 and C10. By the same logic, $\Delta\eta$ will also depend on the set of active hold logics in a particular chip, and can be captured as a distribution over the L Monte Carlo samples. C11 and C12 list the μ and σ values for $\Delta\eta$ distribution.

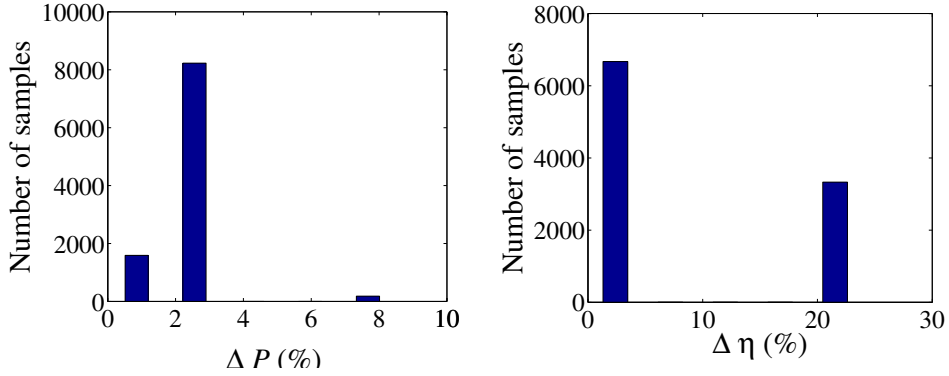


Figure 5.13: The distribution of percentage (a) ΔP and (b) $\Delta\eta$, over all the $L = 10000$ Monte Carlo samples for circuit s9234.

In interpreting these results, it is important to note that the distributions of ΔP and $\Delta\eta$ are ***not*** Gaussian, but can be quite skewed. This is because a variation may cause a hold logic to be activated or disactivated; large bars correspond to frequently-activated

hold logics that correspond to paths that are frequently critical. The distributions for an example benchmark, s9234, are shown in Fig. 5.13. Even more skewed distributions are seen for other benchmarks, e.g., s38417. Therefore, the fact that $\mu - 3\sigma$, for example, is negative, should not be misinterpreted to mean that the distribution necessarily shows samples with negative power or throughput overheads. In fact, we observe that for both the circuits, $\Delta\eta$ *is positive over all chips*. We also see many samples having zero ΔP, $\Delta\eta$ values; in such samples, hold logic is not exercised as T_{clk} is not violated.

5.6.2 Analysis and Discussion

Gains through clustering

We first discuss the results of clustering algorithms presented in C4–C7. As discussed in Section 5.3.1. the pessimistic baseline will always be characterized by one node cluster. Compared to this, the clustered scheme has a slightly higher number of node clusters, and the enumerative scheme substantially higher numbers, indicating progressively reduced degrees of pessimism.

Comparing C5 and C6, we find that node cluster generation reduces a large number of potentially critical paths in the uncoarsened circuit (N), up to even 1000×, resulting in a small number of paths in the coarsened circuit (N_C). This shows that our node clustering technique is very effective in reducing the number of paths. The enumerative scheme offers no reduction and becomes prohibitive for large values of N.

For the clustered scheme, N_C is still sufficiently high for most circuits that it is impractical to maintain a separate hold logic for each path in the node-coarsened circuit. Our second step of path coarsening (Section 5.5.2) further reduces this number, and generates a set of m hold logics for VAHL listed in C8: it is seen that for our clustered approach, $m \leq 8$ over all the circuits. This is again a significant reduction and hence very beneficial in gaining low area and power overhead, as discussed next. On the other hand, the enumerative scheme does not allow any path clustering.

Area overhead

Our first observation is that for the clustered VLU, the increase in area over the pessimistic VLU is appreciably small: 3.7% on an average. This is a significant reduction

compared to 90.6% overhead incurred in the enumerative scheme (in fairness, it should be pointed out that the average area overhead for the four circuits where enumeration is feasible is 4.4%). We further notice that for some circuits (such as s5378, s35932), ΔA is negative, implying less area than the pessimistic VLU. This can happen as VAHL area depends not only on the number of hold logics (which are higher in the clustered VLU), but also their logical complexity. A single hold logic may contain a large number of terms in its (minimized) logical expression, hence requiring a large number of gates to realize such logic. Compared to this, a number of hold logics with only a few terms in their (minimized) logical expressions may need only a small hardware for logic implementation.

For the clustered approach, a few circuits (s5378, s38584) give an appreciably low area overhead even with high threshold values. This means that there is a high degree of correlation in such circuits, offering greater potential for the use of our clustering method.

Power overhead

As stated earlier, ΔP in C9, C10 is proportional to the area overhead of the active hold logics, and is therefore observed to be always less than or equal to the total ΔA in C8.

Further, we observe that although ΔA is positive for clustered scheme, ΔP is negative for nearly half of the circuits. This implies a small power savings with clustered VLU design, and attributes itself to reduced pessimism: for the baseline pessimistic case, all of the hold logic is active in a particular chip when the T_{clk} constraint is violated, whereas in the clustered scheme, only a subset of the hold logics is active, which may dissipate less power than the pessimistic VAHL. For different circuits therefore, sometimes the pessimistic schemes wins, and sometimes the clustered scheme wins (and sometimes the one-cycle case may win, when T_{clk} is met).

For the enumerative case, both area and power overheads are quite high even for small circuits.

Throughput enhancements

We now analyze the throughput values in C11, C12. On an average, the clustered scheme offers high throughput enhancements, with a mean value of 16.7% across all

the chips, approaching quite close to the 20.4% value for the enumerative case. Given the low area/power overhead of the clustered scheme, our "Middle Way" approach for clustering is therefore very beneficial in offering the desirable aspects of the pessimism and enumeration extremes: low overhead and close-to-maximum throughputs.

One exception is of circuit s35932: $\Delta\eta$ values are low and similar for both approaches; this indicates that this circuit has most of its potentially critical paths with near-critical delays (similar observations are also discussed in [70, 74]), and all such paths will be moved to second cycle. Clustering (or even path enumeration) therefore will not show much benefit.

5.6.3 Runtime

We now present the runtime (in seconds) for our clustered approach, for the presilicon steps of Algorithm 1[7] in Table 5.2, with the same values of ρ_{th} and $f_{\mu,th}$ as in Table 5.1 for each of the ISCAS89 circuits. We first tabulate the runtime for SSTA in C2, to compare the runtimes of our node and path clustering algorithms in C3, C4 with those of SSTA. In C5, we present the total runtime for the clustering step (C3 + C4), and finally present the overall total runtime in C6.

Node clustering

On an average, the runtimes for node clustering algorithm in C4 is smaller than that of SSTA. This is intuitive as it involves a topological traversal only on the potentially critical gates. This may not be strictly true: for s9234, s13207, this is larger. Runtime increases with the number of node clusters processed (which are higher for s9234 and s13207). This is also related to the degree of correlation in the circuit: s15850 has largest N over all circuits, but due to high correlation, most of the nodes can be clustered together into a small number of node clusters, saving runtime.

Path clustering

From C4, we observe that the path clustering step also takes much less runtime than SSTA (except for s13207, for which the number of node and path clusters are highest

[7] Runtime for postsilicon steps is small and measurement dependent, and was discussed in Section 5.3.3

Table 5.2: Runtimes (in seconds) for clustering and VAHL generation

Circuit	SSTA	Node Clusters	Path Clusters	Total (Clustering)	Total
C1	C2	C3	C4	C5	C6
s27	0.0	0.0	0.0	0.0	0.0
s1196	0.7	0.5	0.5	1.0	1.6
s5378	7.4	2.1	0.1	2.2	9.6
s9234	14.6	19.6	5.0	24.6	39.2
s13207	74.3	132.9	208.0	340.9	415.2
s15850	89.6	18.7	31.4	50.1	139.7
s35932	229.8	10.5	0.1	10.6	240.4
s38417	207.2	51.2	1.7	52.9	260.0
s38584	233.0	26.4	0.1	26.5	259.4
Average	95.2	29.1	27.4	56.5	151.7

among all other circuits). Note that this step involved a path enumeration; however, due to our node clustering step, this becomes much more inexpensive than the topological traversal based SSTA! This gain is further pronounced for the largest circuits: s38417 and s38584. The final result is that the average total runtime for clustering step, shown in C5, is smaller than that of SSTA.

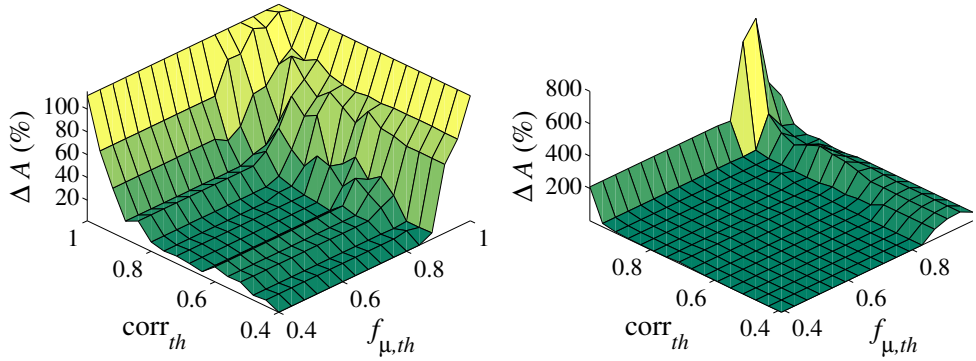
5.6.4 Choice of Threshold Values

Our results in Table 5.1 were presented for some suitable value of ρ_{th} and $f_{\mu,th}$ thresholds. It will be therefore useful to understand how this value can be arrived at.

We recall that any choice of $\rho_{th}, f_{\mu,th} \in (0, 1)$ values will be a good choice if these give us as close (or as high) $\Delta\eta$ values as the enumerative VLU design, and as close (or as low) ΔA values as the pessimistic VLU design. This was observed to be true for the threshold values listed in Table 5.1.

In order to obtain this choice, we plot the trends of ΔA against $\rho_{th}, f_{\mu,th}$, and observe the point closest to $(\rho_{th}, f_{\mu,th}) = (1, 1)$ that yields a sufficiently low area overhead. This choice also ensures high throughputs. This trend is shown for two of the ISCAS89 circuits, s1196 and s9234, in Fig. 5.14. For both the circuits, with a decrease in $\rho_{th}, f_{\mu,th}$ values from (1, 1) to (0.4, 0.4), the value of ΔA, being very high at (1, 1), reduces quickly, offering some "knee" points on the curve where the reduction rate changes quickly. These can be identified as suitable choices for thresholds, one of which is

tabulated in Table 5.1. From these figures we infer that we may only need to check a few combinations of threshold values in between (0.8, 0.8) and (0.97, 0.97) to obtain a high enough $\Delta\eta$ value, close to the enumerative case. Observing the threshold values listed in Table 5.1, we find this intuition to be true for all ISCAS89 circuits.



(a) s1196 (b) s9234

Figure 5.14: Variation of ΔA with $(\rho_{th}, f_{\mu,th})$ for s1196 and s9234.

As before, ΔA may be somewhat higher for lower threshold values, as the logical complexity of VAHL may increase ΔA. We further note from Table 5.2 that clustering algorithms take take less than a minute on an average. Evaluating ΔA at multiple combinations of $\rho_{th}, f_{\mu,th}$ is therefore inexpensive as a one-time presilicon step.

5.6.5 Validation of Our Scheme

While we attempt to be less pessimistic than the pessimistic approach, a functional error is possible in the postsilicon stage if a two-cycle path is wrongly predicted as a one-cycle path by our postsilicon processing described in Section 5.3.3. In this section, we validate that our algorithms have *pessimism indeed*: they do not cause any functional errors.

We first notice that a two-cycle path is identified as a one-cycle path only when at least one of the critical ports in any of the critical gates in the circuit is wrongly identified as noncritical, as such computations rely completely on the set of critical ports.

To investigate if such a scenario may arise, we perform Monte Carlo simulations

with $L = 10000$ samples, and follow these steps for validation:

1. Each Monte Carlo sample corresponds to a case where we know the exact delay distribution of the circuit. We perform an STA over the circuit and determine the exact set of critical ports, and term it as $\mathrm{C}_{S,exact}$.

2. Next we simulate the clustered scheme: for each of the Monte Carlo samples generated above, the postsilicon processing steps described in Section 5.3.3 are performed to identify all critical path clusters. We then determine the set of critical ports as ports that lie on the gates on the paths of critical path clusters, termed as $\mathrm{C}_{S,clus}$.

3. In order to be functionally correct, we must have $\mathrm{C}_{S,clus} \supseteq \mathrm{C}_{S,exact}$, indicating that one-cycle paths may be predicted as two-cycle paths, but not the other way.

We test for this supersetting property for each of the circuits listed in Table 5.1, and find that among the L samples, zero samples caused such an error, thus validating our scheme.

Chapter 6

Employing Circadian Rhythms to Enhance Power and Reliability

Chapters 4 and 5 presented a framework for better-than-worst case designs for the compensation of BTI and variations in digital circuits, maintaining functional correctness across all chips and throughout lifetime.

As we saw in Chapter 4, compensation incurs some overhead. If the amount of BTI degradation is reduced, such overhead can also be appreciably reduced, and can be useful for both conventional and the better-than-worst-case paradigms. This forms the pathway for the last part of our work in this thesis, wherein we present in this chapter, a novel, counterintuitive, reliability-aware, and low-power scheme for circuits and architectures.

6.1 BTI Mitigation: Circadian Rythms

6.1.1 Background and Motivation

With the continued scaling of CMOS technology, the demand for low power consumption in circuits and computer systems has risen sharply. Increased on-chip power due to switching and leakage can have numerous undesirable effects [10], and techniques for achieving reliable, low-power operation have therefore become a critical issue in the design flow.

Amongst the various factors that add to the power dissipation of a chip, one of the major contributors is the design overhead for ensuring functional correctness over its lifetime. The presence of variations and various aging effects causes spatial and temporal changes of the chip frequency, respectively, and requires delay guardbanding to ensure that the T_{clk} specification is met throughout the lifetime of the chip. In case of aging, this design overhead can increase the power of various circuits on the chip by about 30% in the 45nm regime [12]; this becomes increasingly significant at more deeply scaled technology nodes. Reducing or removing such design overhead is an important component of low power design.

The most major component of aging in digital circuits is attributable to BTI. Various approaches (apart from our approach presented in Chapter 4) have been proposed to overcome this degradation. As discussed above, some methods introduce delay guardbands using sizing or resynthesis [75] to add a delay margin to the nominal ($t = 0$) design. Since this can incur a significant overhead, other methods have also been pursued to mitigate/compensate for these effects and reduce the design overhead. At the circuit level, adaptive body bias and adaptive supply voltage schemes [76, 12] compensate for BTI degradation by dynamically increasing the values of V_{dd} and V_{bb} voltages to speed up the circuit. Since the optimum for each circuit block may be different, this could involve the generation of a large set of V_{dd} and V_{bb} values, which poses a significant challenge. Such a solution requires a voltage control system to supply different values of V_{dd} and V_{bb} to different circuit blocks. Implementing these multiple values at the architectural level, with a small number of chip-level supply voltage and body-bias voltage regulators, is a significant challenge.

Chip-level dynamic voltage scaling (DVS) schemes [77, 78, 79, 80, 81] to recapture lost performance overcome this problem by dynamically varying the supply voltage at the processor level. These methods also mitigate BTI by managing the workload amongst multiple cores.

State-based schemes detect the idle states of the circuit during computation [82, 83], and apply a suitable recovery mechanism to lower the degradation. Other methods in this class distribute tasks over partitioned functional units to balance aging [84] and perform node vector control [85] or power-gating [86] during idle times.

Such idle state approaches have some common limitations. First, the idle states

are dependent on the workload/circuit configuration: the precise idle times tend to be unpredictable, or difficult to predict dynamically. Hence, the schemes require a complex hardware/software control mechanism that can (a) dynamically detect the idle times during execution, (b) apply the appropriate recovery mechanism, and (c) keep track of which parts of the circuit have partially recovered after the idle time, and by how much. Second, it may not be easy to exploit such idle times fully, since modern out-of-order execution and multi-threading endeavor to hide idle periods. A better approach would be to have predictable idle times of fixed durations, requiring a potentially much simpler control mechanism.

6.1.2 Circadian Rhythms for Circuits

In this work, we employ an entirely new approach to reduce circuit aging. We propose Greater-than-NOMinal Operation (GNOMO), a novel and superficially counterintuitive scheme for mitigating BTI that goes against the conventional wisdom that operation at a higher V_{dd} will result in a higher delay degradation and higher power dissipation. On the contrary, we show that by elevating V_{dd} to an optimal, greater-than-nominal value, we can achieve both lower delay degradation and power dissipation than that incurred at the nominal V_{dd}, at roughly constant performance.

Our ideas are inspired by the notion of human circadian rhythms of wakefulness and sleep. A human is likely to age more quickly without adequate rest, and we show that a similar argument can be made for an inanimate circuit. The normal way to exercise a circuit is to subject it to a nominal supply voltage, $V_{dd,n}$, throughout its lifetime. Under our scheme, we apply a *greater-than-nominal* voltage, $V_{dd,g} > V_{dd,n}$, interspersed with predictable periods of sleep (i.e., power-gating), to reduce aging effects. Intuitively, just as a human uses sleep to recover from fatigue, and can operate at greater intensity after adequate sleep, a circuit can also recover from BTI while it is "asleep," and can operate at the larger supply voltage value, $V_{dd,g}$, and yet age *less* than the scenario where it is constantly "awake" and subjected to the $V_{dd,n}$ voltage under nominal operation.

For a given nominal V_{dd}, this chapter develops a procedure that allows the static determination of the optimal $V_{dd,g}$ for a circuit. We show that the GNOMO scheme can result in enhanced reliability and lower aging. This reduction, as well as other aspects of GNOMO, can then be parleyed into a reduction in the power overhead of guardbanding

a circuit against aging degradation. We exercise the GNOMO approach from the circuit level to the architecture level, and show how power savings can be achieved through a practical adoption scheme that is applied up to the architectural level. Specifically, we show in Sections 6.5.1 and 6.5.2 that GNOMO enables a reduction of about 1.3× to 1.8× in delay degradation, for various values of $V_{dd,n}$ considered in this chapter. For the same lifetime, this reduction in degradation implies that reduced guardbands are necessary as compared to the nominal voltage case. This yields a reduction of about 1.8× to 3.2× in area overhead and about 1.5× to 3.1× in the power overhead.

GNOMO does not require fine-grained voltage supplies/control. Nor does it require the detection of idle times (or the potentially complex associated circuitry) since the idle times are *generated*, and not *detected*, and are hence predictable-by-construction. Further, the idle times are orthogonal to those that dynamically occur during workload execution (due to cache misses, branch mispredictions, etc.). Moreover, they do not depend on a precise characterization of signal probabilities, as is the case for other approaches: characterizing BTI aging based on signal probabilities is inherently unreliable, in that probabilities represent an often unpredictable average rather than a worst case[1] . The use of predictable idleness, on the other hand, provides safe, correct-by-construction, guarantees on the amount of recovery.

We work with a widely adopted model [50] for predicting delay degradation due to BTI, as discussed in Section 2.2.3. Further, our delay and power modeling for circuits is based upon the formulations described in Section 2.2.4.

The remainder of this chapter is organized as follows. Sections 6.2 and 6.3 present the framework and architectural implementation for GNOMO, and are followed by an analysis of the power dissipation under GNOMO in Section 6.4. We then present our results in Section 6.5.

6.2 GNOMO: Greater-Than-NOMinal V_{dd} Operation

We now present the GNOMO framework for BTI mitigation. As defined earlier, the term $V_{dd,n}$ refers to the nominal supply voltage and $V_{dd,g}$ to the GNOMO value.

[1] While such averages are useful in working with the "softer" constraints associated with power dissipation, they are much more unreliable for the "harder" constraints involved in timing.

6.2.1 Circuit Recovery through Power Gating

Motivating intuition

We illustrate the idea of GNOMO through the example of an ALU circuit, the MCNC benchmark alu4. We define the delay degradation, ΔD, as the increase in circuit delay due to BTI effects, and express it as a percentage of the nominal delay, $D(0)$, at time 0. The curve marked "$V_{dd,n}$" in Fig. 6.1(a) shows the temporal change in ΔD (%) for alu4 when the circuit is operated at a nominal supply voltage, $V_{dd,n} = 1.0$V. The time required to complete the ALU computation is denoted as t_n. Note that monotone degradation under stress shown here captures the effect of alternate stress/recovery cycles and plots the *envelope* of BTI degradation [50, 49].

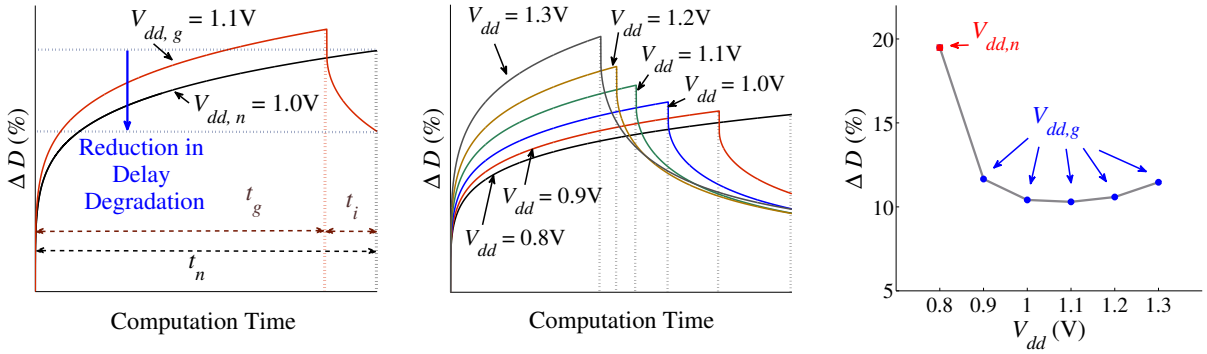


Figure 6.1: The delay degradation patterns of MCNC benchmark alu4 at (a) nominal supply voltage $V_{dd,n} = 1.0$V and greater-than-nominal supply voltage $V_{dd,g} = 1.1$V, (b) $V_{dd} \in [0.8\text{V}, 1.3\text{V}]$ values, and (c) delay degradation for alu4 at time t_n in (b).

Under GNOMO, at a higher supply voltage $V_{dd,g}$ (chosen to be 1.1V in this figure), the ALU has a lower delay and completes the same computation in time $t_g < t_n$. To maintain the same throughput as the $V_{dd,n}$ case, in principle the data may be latched at time t_g, and the circuit could then be power-gated[2] during an idle time, $t_i = t_n - t_g$. During the idle time, the circuit recovers from BTI degradation, as shown in the figure. The net result is that at time t_n, the BTI degradation for GNOMO is lower than that

[2] For convenience, we will temporarily assume that such a power-gating operation is instantaneous. We will remove this assumption later.

under the nominal supply voltage.

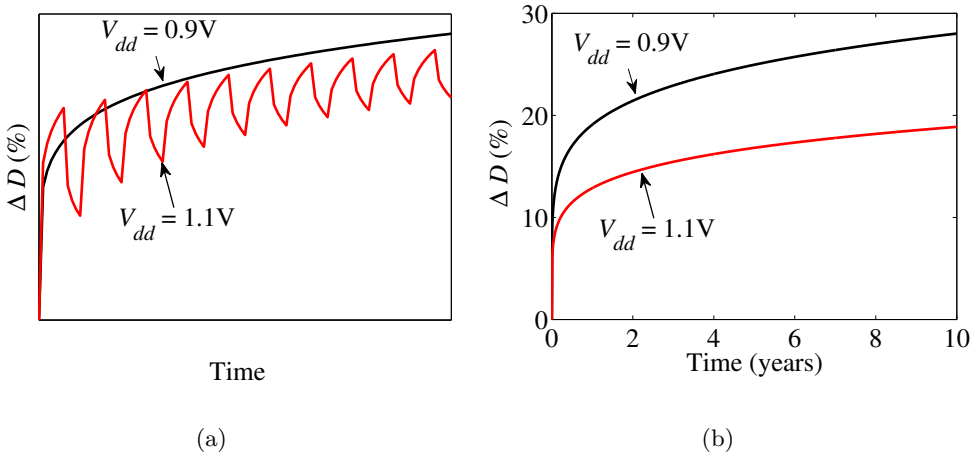
We explore this tradeoff further in Fig. 6.1(b), for the case where a different baseline voltage, $V_{dd,n} = 0.8$V, is used, and several $V_{dd,g}$ values are considered. A higher value of $V_{dd,g}$ implies greater degradation during the compute period, and a larger idle time. Since the degradation increases superlinearly with the supply voltage, but the idle times increase sublinearly (as will be shown in Section 6.2.3), it is possible to identify a supply voltage point at which the overall degradation at time t_n is optimized. This is illustrated in Fig. 6.1(c): as $V_{dd,g}$ is increased, the percentage ΔD first decreases, reaches a minimum at 1.1V, and then increases again.

Mechanism

In reality, it is quite impractical to implement such a scheme in the form described above, where circuits must be put to sleep and woken up within a single clock cycle. The essence of the idea can nonetheless be extended to a realistic framework: at a higher V_{dd} value (corresponding to a higher clock frequency), instead of switching-off/waking-up the circuit within each cycle, we run the circuit for a large number of cycles at a faster-than-nominal clock. This completes a part of the overall computation more rapidly than at the nominal supply voltage/clock frequency, but we maintain the same throughput by introducing idle time, during which the circuit is power-gated and allowed to recover from BTI degradation.

This idea effectively provides the same sleep/wakeup "duty cycle" as in the earlier conceptual exposition, and therefore the same pattern of temporal degradation and recovery. From the notion of frequency independence of BTI [50, 48, 87], the degradation/recovery depends on the duty cycle rather than the precise distribution of on/off periods, and therefore this alternative, more practical, formulation results in the same amount of recovery as the conceptual idea presented earlier.

This is depicted more concretely through Fig. 6.2, which shows the delay degradation patterns for MCNC circuit b12 under GNOMO. When the processor is operated at the GNOMO V_{dd}, the initial delay degradation is higher. This degradation, however, as shown in Fig. 6.2(a), is decreased due to the recovery obtained in the idle phase, and susequently becomes less than (or "crosses over") the nominal delay degradation very early in the lifetime. An envelope of this degradation trend over 10 years of operation is



(a) (b)

Figure 6.2: GNOMO results in lower degration than nominal operation (a) GNOMO delay degradation becomes less than nominal degradation very early in the lifetime, and (b) the envolope of GNOMO delay degradation over a period of 10 years. The plot in (a) shows the early lifetime corresponding to a highly magnified view the extreme left of the degradation envelope in (b).

plotted in Fig. 6.2(b), which shows that delay degradation at GNOMO is substantially lower than the nominal delay degradation. Fig. 6.2(a) corresponds to a highly magnified view of the extreme left of Fig. 6.2(b) (the early lifetime). Simulations and experimental data from [87] also validate this trend: operating at a higher V_{dd} value with interspersed idle cycles reduces the overall delay degradation compared to the nominal case.

Therefore, a practical implementation of GNOMO works as follows: the processor functions under a *circadian rhythm*, where it is awake and runs at high speed for several (typically, millions of) cycles at the GNOMO supply voltage, $V_{dd,g}$, during the *compute phase*, and then sleeps for some cycles during the *idle phase* where it is power-gated. The circadian cycle is then continued throughout its lifetime.

The alternation of the compute/idle phases is depicted in Fig. 6.3, which shows the supply voltage value along the y-axis and time (in cycles as well as seconds) along the x-axis. We use this figure to introduce some notation that will be used in the remainder of this chapter. For a given workload, consider the operation of the processor during a compute phase, corresponding to a fixed number of clock cycles, c_f.

Let the number of instructions committed, while operating at $V_{dd,n}$ ($V_{dd,g}$), be I_n

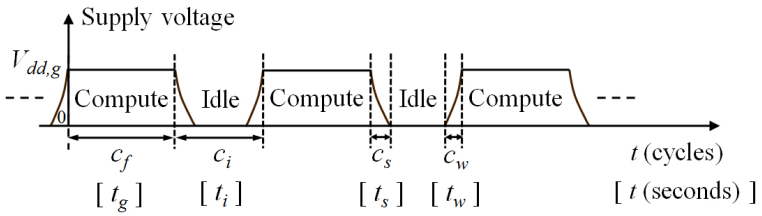


Figure 6.3: The compute and idle phases in GNOMO in the practical implementation. This figure is not drawn to scale; in reality, t_g, $t_i >> t_s$, t_w.

(I_g), and let the corresponding execution time be t_n (t_g) time units. Clearly,

$$t_n = c_f \cdot T_{clk,n} \text{ and } t_g = c_f \cdot T_{clk,g} \tag{6.1}$$

The duration of the idle phase is denoted by t_i (in seconds) and c_i (in terms of the number of cycles). During this period, the circuits are power-gated and do not perform any computation[3] .

The additional costs associated with the idle phase are also illustrated in the figure. Power-gating a circuit incurs an overhead of t_s time units (c_s cycles) for the circuits to transition to the sleep state, and an overhead of t_w time units (c_w cycles) for wakeup. The sleep/wakeup transitions are deliberately designed to occur within the idle phase, ensuring that the execution of instructions is not affected by the GNOMO scheme.

Constraints on the Choice of c_i and c_f

At the chosen value of $V_{dd,g}$, for a specific lifetime goal, a prescribed ratio of t_g to t_i can be calculated. Therefore, if c_i (and hence t_i) is very large, then c_f (and hence t_g) will also be large; conversely if c_i is small, then c_f will also be small. In this section, we will discuss the constraints that place double-sided bounds on the choice of c_i (and hence, on c_f).

Existing power-gating frameworks offer sleep transition times (c_s) of about 10 to 50 cycles for various circuits in a processor [86], while the wakeup time (c_w) is typically about 5-10 cycles. Since these transitions are designed to occur within the idle phase,

[3] Note that this idle phase is deliberately inserted and therefore easily predictable, and is thus different from the idle periods that may occur within the compute phase due to cache misses, TLB misses, branch mispredictions, etc.

the effective idle time may decrease significantly if c_s and c_w are comparable to c_i. To amortize the effects of sleep/wakeup transitions, it is necessary to choose c_i to be significantly larger than c_s or c_w. This constraint places a lower bound on the value of c_i that must be used, an issue that is discussed in greater detail in Section 6.5.4.

Thermal constraints also place an upper bound on the choice of c_i. If a large value of c_i is chosen, then the processor could operate under a high supply voltage, $V_{dd,g}$, for a prolonged period, possibly leading to thermal problems. The value of c_i must be chosen in such a way that t_g is well below the thermal time constant of silicon, so that if the power is relatively unchanged from the nominal case, the alternate compute/idle phases do not affect the peak temperature.

In practice, choosing c_f to be of the order of ten million cycles provides a reasonable balance that meets the double-sided constraints discussed above.

6.2.2 Idle Time Generation – Practical Considerations

Recall that the number of instructions executed in c_f cycles at $V_{dd,n}$ and $V_{dd,g}$ are I_n and I_g, respectively.

If the frequencies of all the components in a CPU (both on-chip and off-chip components) were to scale at the same rate as V_{dd} is changed, the number of instructions committed in c_f cycles would be the same, i.e., $I_n = I_g$. The idle time $t_{i,1}$ could then be computed as:

$$t_{i,1} = t_n - t_g = c_f \cdot (T_{clk,n} - T_{clk,g}) \tag{6.2}$$

However, in practice, the voltages and frequencies are scaled up only for on-chip components (processor, cache, on-chip buses, etc.), and the access time for off-chip memory (upon a cache miss) remains the same at both $V_{dd,n}$ and $V_{dd,g}$. This constant access time corresponds to a larger number of cycles under the faster clock at $V_{dd,g}$.

The overhead of off-chip operations such as cache misses therefore corresponds to a larger number of cycles of penalty under $V_{dd,g}$, implying that during a fixed number of clock cycles, c_f, while the processor is awake, the number of instructions committed under GNOMO will be smaller. In other words, $I_g = I_n - I_o$, where I_o is the number of instructions that could be committed at the nominal supply voltage, but not at $V_{dd,g}$.

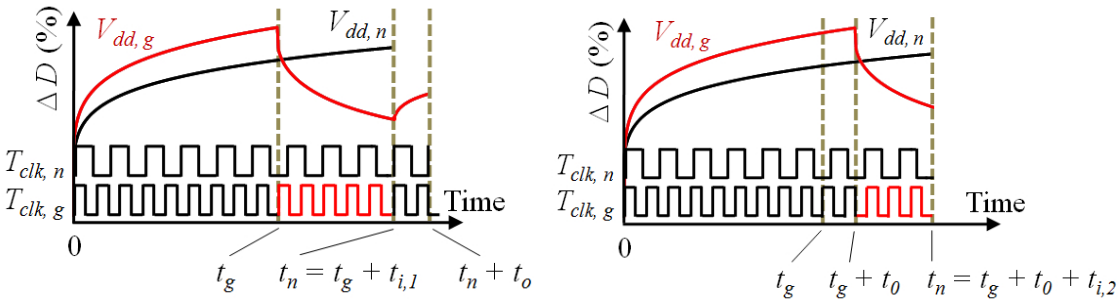


Figure 6.4: The illustration of our scheme for generating (a) fixed idle time, $t_{i,1}$, and (b) variable idle time, $t_{i,2}$. The figure is not drawn to scale; in reality, t_n, t_g, $t_{i,1}, t_{i,2} >> t_o$.

We denote the overhead of completing the execution of these I_o instructions by c_o (in clock cycles) and t_o (in seconds).

This overhead can be accommodated in two ways:

- By keeping the duration of idle phase fixed ($= t_{i,1}$) and deferring the execution of I_o instructions to *after* the idle phase, as shown in Fig. 6.4(a): this incurs a performance penalty of t_o time units (we show that this performance penalty is very small in Section 6.5.4).

- By executing the I_o instructions *within* the idle phase, as shown in Fig. 6.4(b) (which shows the same operations as in Fig. 6.4(a), except for the placement of the execution overhead). This reduces idle time from $t_{i,1}$ to $t_{i,2}$:

$$t_{i,2} = t_n - (t_g + t_o) = t_{i,1} - t_o \tag{6.3}$$

  This reduction in idle time also reduces the overall recovery possible, albeit without a performance penalty.

For a specific value of $V_{dd,n}$, the value of $t_{i,1}$ is *fixed* as it depends only on the fixed number of cycles c_f and the frequency corresponding to $V_{dd,n}$, which is also fixed according to Table 6.1. On the other hand, the value of $t_{i,2}$ *varies* with the number of off-chip accesses during execution.

A second practical consideration is the need to preserve the state of the processor during the sleep periods, thus facilitating a rapid return to normal execution upon

wakeup. To ensure this, the circadian cycle is applied to computational units such as ALUs, decoders, and sense amplifiers. Storage elements such as caches, register files, the reorder buffer (ROB), tables for virtual address translation (TLBs) and for branch prediction (BPTs), and load-store queue (LSQs), may contain state data that must be preserved, and are maintained with suitable optimizations, as reported in Section 6.3.2.

6.2.3 Idle Time Generation – Implementation

In our experiments, we use the first of the two schemes proposed above: after completing c_f cycles, we use a fixed idle time of $t_{i,1}$ time units (corresponding to $c_{i,1}$ cycles), as given by Equation (6.2). Under this scheme, the idle phase duration is fixed, and a fixed and predictable amount of recovery is guaranteed. A substantially similar approach may be used for the second scheme.

Since the idle time is fixed, the completion time of $I_g + I_o$ instructions is delayed by c_o cycles, i.e., there is a performance penalty involved, but the amount of recovery time is guaranteed. The total performance penalty for a particular workload is given by:

$$\text{Performance Penalty} = \frac{t_o}{t_n} = \frac{c_o \cdot T_{clk,g}}{c_f \cdot T_{clk,n}} \tag{6.4}$$

In our experiments, the choice of $V_{dd,n}$ and $V_{dd,g} > V_{dd,n}$ may take one of several values; each such value corresponds to a different frequency of operation for the processor. We use a set of realistic discrete supply voltage/frequency (V_{dd}/f) pairs, adopted from Intel's recent 48-core IA-32 Processor [1] as shown in Table 6.1, where V_{dd} lies in the range [0.7V, 1.3V] (this choice is only for illustration purposes; any other V_{dd}/f framework can be used instead)[4] . This allows us to operate within the framework of existing technologies to illustrate the principles of GNOMO. Table 6.1 confirms our earlier observation in Section 6.2.1 that with a linear increase in the value of V_{dd}, the increase in the clock frequency (i.e., circuit speed) is sublinear.

Using the data in Table 6.1, the fraction of the idle time at the GNOMO supply voltage in the "ideal" case (where $t_0 = 0$), to the execution time for c_f cycles at $V_{dd,n}$,

[4] It is important to emphasize here that although this table is taken from the allowable DVFS values for an Intel processor, GNOMO is *not* an adaptive supply voltage scheme (ASV) for BTI mitigation. Under the baseline GNOMO scheme, the processor operates at a constant voltage and frequency. However, it is possible to extend the baseline GNOMO framework to the case where ASV is required for power management.

Table 6.1: Operational V_{dd}/f pairs adopted from Intel's IA-32 Processor [1]

V_{dd} (Volts)	0.7	0.8	0.9	1.0	1.1	1.2	1.3
Frequency (GHz)	0.25	0.47	0.68	0.86	1.03	1.17	1.30
T_{clk} (ns)	4.00	2.13	1.47	1.16	0.97	0.85	0.77

can be computed for valid combinations of ($V_{dd,n}$, $V_{dd,g}$) as follows:

$$\frac{t_{i,1}}{t_n} = \frac{c_f \cdot (T_{clk,n} - T_{clk,g})}{c_f \cdot T_{clk,n}} = 1 - \frac{T_{clk,g}}{T_{clk,n}} \tag{6.5}$$

Note that this expression is an approximation of the realistic idle time fraction, $t_{i,2}$ / t_n, since our experiments show that the performance penalty is small.

Table 6.2: Percentage idle time $t_{i,1}$ for various ($V_{dd,n}, V_{dd,g}$)

$V_{dd,g} \rightarrow$ / $V_{dd,n} \downarrow$	0.8V	0.9V	1.0V	1.1V	1.2V	1.3V
0.7V	46.8%	63.2%	70.9%	75.7%	78.6%	80.8%
0.8V	–	30.8%	45.3%	54.3%	59.8%	63.9%
0.9V	–	–	20.9%	34.0%	41.9%	47.7%
1.0V	–	–	–	16.1%	26.5%	33.6%
1.1V	–	–	–	–	12.5%	20.8%
1.2V	–	–	–	–	–	10.0%

Table 6.2 shows this percentage, wherein each entry is computed by Equation (6.5), using the respective values of ($T_{clk,n}$, $T_{clk,g}$) from Table 6.1. We observe the following diminishing returns in idle times:

- For a particular value of the nominal supply voltage $V_{dd,n}$ (say 0.8V), a linear increase in the value of $V_{dd,g}$ (along the row from 0.9V to 1.3V) increases the idle time durations only in a sublinear fashion.
- At higher values of $V_{dd,n}$ ($\geq$ 1.1V), the available idle time is low.

The idle times discussed above correspond to available time for BTI recovery, and therefore the delay degradation improvements also show diminishing returns, as was illustrated earlier in Fig. 6.1(c).

6.3 Architectural Implementation of GNOMO

In this section, we discuss the details of GNOMO implementation on a processor with out-of-order execution, and the simulation framework that we utilize to validate the gains of this approach.

6.3.1 Processor Details

The structure for a general purpose processor is depicted by Fig. 6.5. Broadly, the components of the processor can be categorized into on-chip components and off-chip components.

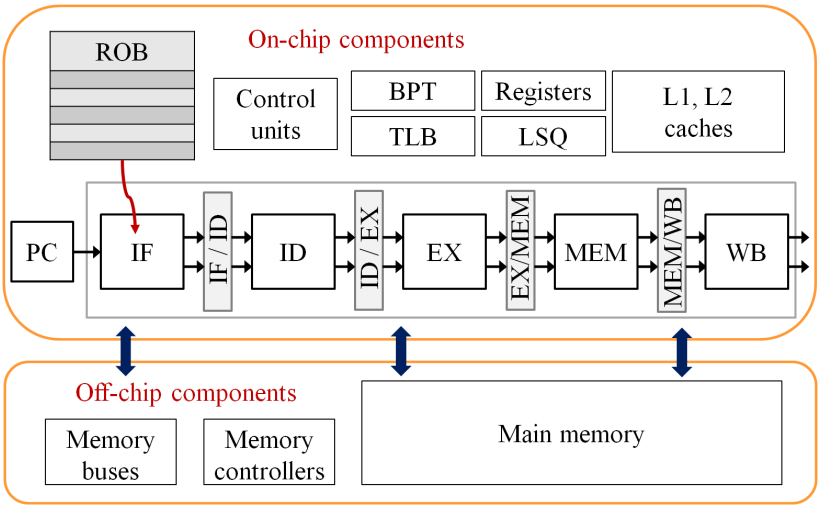


Figure 6.5: Schematic of an out-of-order processor, with its on-chip and off-chip components.

The on-chip components include all the circuitry and resources that require fast communication during the execution of instructions, such as the ROB, registers, TLBs, BPTs, and LSQs. The processor architecture is assumed to be a MIPS-like five-stage pipeline shown in Fig. 6.5, with the fetch, decode, execution, memory and commit stages. On-chip caches store the copy of a part of the data from the main memory and are further divided into L1 and L2 caches. Communication between the various components is achieved through on-chip buses.

The off-chip components include the peripheral memory (main memory), the associated memory controllers and the memory buses. An access to the main memory, which occurs when there is a cache miss, is carried out by the buses connecting the on-chip and the off-chip components.

6.3.2 Simulation Framework

Processor Model

We implement our GNOMO framework in a MIPS architecture based out-of-order execution processor simulation environment built upon Simplescalar [88], with the added functionality of being able to model the power dissipation of the various components of the processor using Wattch [89]. We adopt SPEC 2000 suite [90] as the benchmarks for simulations, which are executed using Wattch, with the inputs for these benchmarks derived from the MinneSPEC set [91]. The technology parameters for our simulations are at the 32nm node: since the available implementation of Wattch is based on older technology nodes (100nm and above), the parameters used in the power model were updated for the 32nm node from Orion2.0 [92].

During execution, a small, fixed number of instructions are fetched from the instruction cache into the ROB. The processor then repeatedly extracts an instruction from the tail of the ROB and starts execution in the pipeline beginning with fetch and decode. Functional and control units are used during the pipeline operation, and the underlying devices undergo stress-relax cycles during execution.

Idle time insertion

We now describe the modifications to incorporate GNOMO in the simulation environment. The first step is to introduce periods of idle times in the execution. To enable this, we maintain a clock counter that is initialized whenever a compute phase begins. After c_f cycles have been completed, the pipeline is flushed, and the sleep signal is activated for all on-chip components. Note that since the pipeline is flushed, the execution in the next compute phase restarts at the instruction that follows the last committed instruction.

Power gating with state preservation

When the processor enters into the idle state, power-gating is applied to some on-chip components. As discussed in Section 6.2.1, power-gating disconnects the devices in these components from the power supply, resulting in a possible loss of state. For computational elements such as ALUs and control units that do not contain useful state information, this is entirely acceptable. We augment Wattch to model the functionalities and power dissipation of the processor when its operation is halted upon sleep, and resumed upon wakeup, and describe this augmentation scheme below.

For caches, register files and other storage structures, we use the sleep signal to preserve state through hybrid drowsy cache techniques [93, 94] that reduce standby leakage. Under this scheme, the sleep signal, instead of cutting off the supply voltage, triggers circuitry that scales the supply voltage of the devices to an appreciably low value (for instance, 0.3V) where the device has a very small leakage and the state can be preserved. The sleep/wakeup overhead associated with these schemes is 1-10 cycles and is thus negligible as compared to the duration of the idle period (which is of the order of a million cycles).

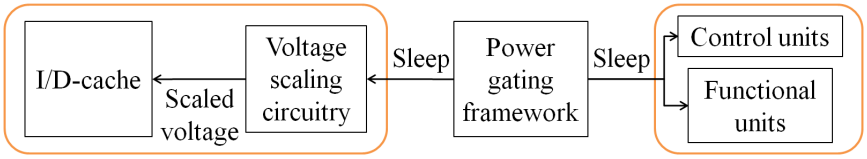


Figure 6.6: The power-gating scheme applied differently for various on-chip components. Units with combinational circuits are completely switched off by the sleep signal. Cache on the other hand preserves state by the use of a special circuitry that scales the cache supply voltage to a relatively low value.

In this scheme, we first save the register files and the storage structures in the cache (their sizes are typically small and this takes a negligible amount of storage), and then allow the sleep signal to activate the hybrid drowsy cache mode. When the next compute phase begins, the data for register files and storage structures is restored. This scheme is depicted in Fig. 6.6, which shows how the same sleep signal acts in a different manner for various on-chip components.

Under this scheme, a main memory access transaction (either for a load or for a

store) may be in progress just before the processor enters into the idle phase. Since the pipeline is flushed, this instruction would have required a re-execution of the memory transaction. This is avoided by our scheme of saving the LSQ entries during the idle phase. Upon wakeup, the processor checks the old LSQ entries and finds the load/store operation already served during the idle phase (since the memory transaction continues to take place in the off-chip components even during the idle time and the off-chip components are not affected by on-chip GNOMO).

6.4 Power Analysis

In this section, we examine the implications of GNOMO on power at both the circuit and architectural levels. In sequence, we examine a set of factors that cause the power dissipation to be altered under GNOMO. First, in Section 6.4.1, we analyze the change in power consumption, averaged over its lifetime, of a circuit operating at an elevated V_{dd}, as compared to the nominal operation, for a unit without state that is completely power-gated during the idle phase. Next, in Section 6.4.2, we examine the impact of reduced aging on the delay guardbands, and hence the power dissipation, associated with such a unit. Finally, in Section 6.4.3, we consider the complete picture: change in the total power dissipation due to GNOMO, which includes the impact of units that are turned off, as well as those placed in a drowsy state, during the idle phase.

6.4.1 Changes in Power as a Function of $V_{dd,g}$

We begin by considering a circuit that is designed to meet the delay specification at the beginning of lifetime, and is not resilient to BTI[5] , and set the delay of this nominal circuit to a normalized power value of 1 unit. When such a circuit is operated at GNOMO, the power dissipation changes as follows:

- With the supply voltage increased to $V_{dd,g}$, dynamic power increases quadratically and leakage increases exponentially [95].
- Even though the V_{th} increase due to BTI degradation is small, the exponential relationship in leakage power leads to a significant reduction of leakage over the

[5] Such a circuit is not practically useful, as it fails with time. We merely consider this as a baseline for all comparisons, for convenience.

lifetime of the chip [12]. This effect is more pronounced at higher V_{dd} values (due to increased BTI degradation).

- A further reduction in the average dynamic and leakage power consumption occurs due to generation of idle time, since power dissipation now occurs only in the compute phase and not in the idle phase (except during sleep/wakeup cycles), which is a fraction of the total nominal computation time.

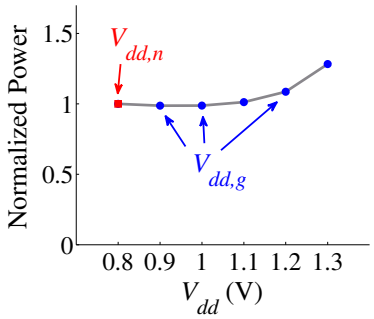


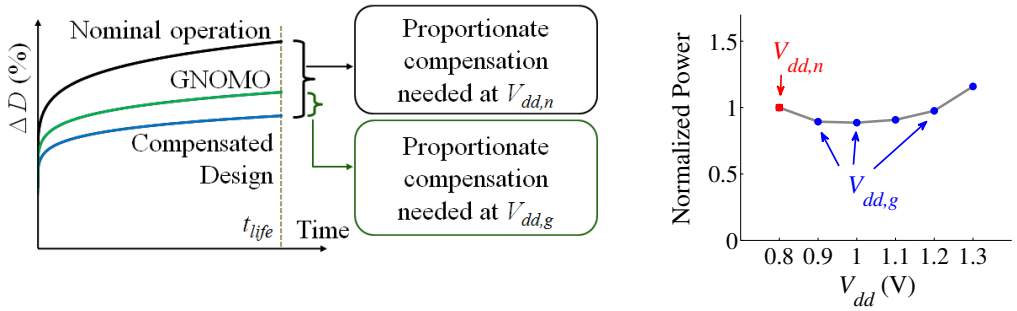
Figure 6.7: Change in average power (dynamic + leakage) for alu4 as a function of $V_{dd,g}$; $V_{dd,n} = 0.8$V.

For our running example of the MCNC benchmark alu4, Fig. 6.7 illustrates the changes of power (dynamic+leakage), normalized to the nominal value defined above, for a typical case with $V_{dd,n} = 0.8$V, with $V_{dd,g}$ ranging from 0.9V to 1.3V. Similar trends are seen for other values of $V_{dd,n}$. Considering the combined impact of the three effects listed above, Fig. 6.7 shows that the power consumption therefore remains about flat until $V_{dd,g} = 1.0$V and then begins to increase beyond this point.

6.4.2 Power Savings in Delay Guardbanding

We now consider the scenario where the delay guardbanding is introduced in the circuit to make it BTI-resilient throughout its lifetime. In our discussion, we use the terms "guardbanding" and "compensation" interchangeably. We work with a guardbanding technique under which the transistors in the circuits are synthesized with a tighter timing constraint such that the end-of-lifetime delay meets the delay specification, D_{spec}. This

delay margin algorithm is identical to that used in Section IV of [12]. The cost of this compensation is in the form of an increased area overhead.



(a) (b)

Figure 6.8: (a) The temporal delay degradation of alu4. The area overhead required to compensate the circuit under GNOMO, is less than that required under nominal operation. (b) Trends in power for alu4, with power overhead due to compensation incorporated, as a function of $V_{dd,g}$; $V_{dd,n} = 0.8$V.

The idea presented in this section can be depicted schematically through Fig. 6.8(a), which shows the temporal delay of a circuit along the y-axis and time along the x-axis. A circuit operated at $V_{dd,g}$ has a lower end-of-lifetime delay degradation than the same circuit when operated at $V_{dd,n}$. Hence, in making each case BTI-resilient, a lower amount of compensation (and associated area overhead) is needed with GNOMO than for the $V_{dd,n}$ case. Since both the dynamic and subthreshold leakage power are proportional to the widths of the transistors used in these circuits, a lower compensation area overhead corresponds to a lower power dissipation of the circuit. This results in a further lowering of the power dissipation at the GNOMO supply voltage points, as compared to the points in Fig. 6.7.

Fig. 6.8(b) shows the changes in the normalized average total power consumption for circuit alu4, with $V_{dd,n} = 0.8$V and $V_{dd,g}$ ranging from 0.9V to 1.3V, under the same workload conditions as Fig. 6.7. This plot combines the effects on power from Section 6.4.1 and the effect of BTI compensation. The net result is a further decrease in the GNOMO power compared to the power under nominal operation: total power remains below the nominal dissipation as $V_{dd,g}$ is increased, up to 1.2V.

6.4.3 Overall Power Dissipation

The analysis so far only considers components that can be fully switched off in the idle state and do not need to retain their states. When we consider the power overhead of state preservation, the total power can be higher than shown in Fig. 6.8(b). The power dissipation in all the components of the processor is evaluated next.

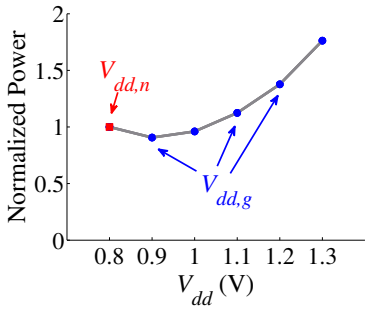


Figure 6.9: Change in the total power with GNOMO, showing the power savings at lower values of $V_{dd,g}$.

The changes of the overall power dissipation (again, normalized to the nominal case) is shown in Fig. 6.9 for $V_{dd,n} = 0.8$V and $V_{dd,g}$ ranging from 0.9V to 1.3V, for the execution of the applu workload (similar trends are seen for other workloads). We observe that as $V_{dd,g}$ is increased, the power dissipation with GNOMO reduces until about 1.0V, and then becomes greater. Note that the rate of power increase is higher in Fig. 6.9 as compared to Fig. 6.8(b).

The power dissipation of cache and storage structures forms a significant portion of the total power dissipation. At higher $V_{dd,g}$ values (beyond $V_{dd,g} = 1.0$V), the power dissipation is also higher. Thus the total power begins to dominate the power savings that we achieve through GNOMO.

6.4.4 Choosing the Optimal GNOMO Supply Voltage

Based on the above analysis of overall power dissipation, we can select an optimal choice of $V_{dd,g}$ $(= V_{dd,g}^{opt})$ for a given $V_{dd,n}$, that maximizes the power savings. For instance, we observe in Fig. 6.9 that for $V_{dd,n} = 0.8$V, this happens at $V_{dd,g} = 0.9$V.

We therefore consider the power savings for the benchmark applu, for all $V_{dd,n}$

values, in Table 6.3. This table is quite similar to Table 6.2, except that it lists the power savings instead of idle times, for valid combinations of $(V_{dd,n}, V_{dd,g})$. Positive entries imply a power savings, whereas negative entries imply a power overhead. From this table, we determine the optimal $V_{dd,g}$ value (for a particular $V_{dd,n}$), to be the value that maximizes power savings, and indicate it by the symbol "*." We observe that for a fixed $V_{dd,n}$, increase in $V_{dd,g}$ (moving along a row) first yields power savings, and then power overhead. Further, as $V_{dd,n}$ is increased, the optimal savings in power (at $V_{dd,g}^{opt}$) although substantial, decrease sublinearly.

Table 6.3: Percentage overall power savings for various $(V_{dd,n}, V_{dd,g})$ for benchmark applu

$V_{dd,g} \rightarrow$ / $V_{dd,n} \downarrow$	0.8V	0.9V	1.0V	1.1V	1.2V	1.3V
0.7V	14.9%*	13.6%	0.7%	-23.4%	-54.5%	-93.4%
0.8V	–	9.4%*	3.9%	-12.4%	-37.7%	-76.2%
0.9V	–	–	6.9%*	1.8%	-20.2%	-55.7%
1.0V	–	–	–	5.3%*	-12.0%	-35.9%
1.1V	–	–	–	–	4.2%*	-21.4%
1.2V	–	–	–	–	–	-15.5%

Table 6.4: The optimal $V_{dd,g}$ values for various values of $V_{dd,n}$

$V_{dd,n}$ (V)	0.7	0.8	0.9	1.0	1.1	1.2
$V_{dd,g}^{opt}$ (V)	0.8	0.9	1.0	1.1	1.2	1.2

Although the simulations above show results only for applu workload, we have observed that these values of optimal $V_{dd,g}$, in fact, are the optimal values across a range of SPEC 2000 benchmarks considered in Section 6.5.3. The optimal ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs for each value of $V_{dd,n}$, are tabulated in Table 6.4. We make the following observations about the data in this table:

- For $V_{dd,n} = 1.2$V, $V_{dd,g}^{opt} = V_{dd,n}$ and no gain is possible. This is attributed to the fact that for various workloads, the total power increases by 14.4% to 16.2% for $V_{dd,g} = 1.3$V candidate value, resulting is no power savings. This is primarily attributed to a steep increase in the leakage power due to the higher voltage value and also due to the low idle time.

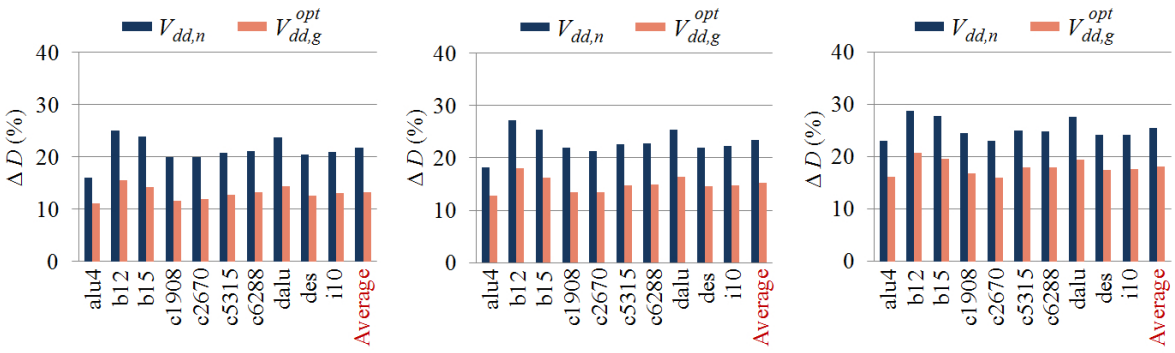
- The GNOMO scheme works best when the value of $V_{dd,n}$ is lower, but it provides significant improvements for all $V_{dd,n} \leq 1.1$V.

6.5 Results

We now present the results of applying GNOMO at both the circuit and architectural levels. At the circuit level, we examine the application of GNOMO on various ISCAS85, MCNC and ITC99 benchmarks, synthesized using ABC [68] on the 32nm PTM [11] based library. Our library consists of INVs; BUFs; 2-4 input NANDs and NORs; 2 input XORs and XNORs; all with different sizes. We choose $t_{life} = 10$ years. We optimize the circuits by introducing delay margins to compensate for BTI aging, using the algorithms in [12]. At the architectural level, we use the framework detailed in Section 6.3.

6.5.1 Delay Degradation Reduction

We evaluate the extent to which GNOMO can reduce the degradation in various benchmark circuits at the transistor level. In Fig. 6.10, for a variety of circuits, we present the end-of-lifetime percentage delay degradation, ΔD (%), for three different ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs from Table 6.4: (0.8V, 0.9V), (0.9V, 1.0V) and (1.0V, 1.1V).



(a) $V_{dd,n} = 0.8$V, $V_{dd,g}^{opt} = 0.9$V (b) $V_{dd,n} = 0.9$V, $V_{dd,g}^{opt} = 1.0$V (c) $V_{dd,n} = 1.0$V, $V_{dd,g}^{opt} = 1.1$V

Figure 6.10: The reduction in delay degradation with GNOMO, shown for various circuits for three different ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs.

In all cases, we see that under GNOMO, the end-of-lifetime delay degradation is significantly smaller than under the nominal scheme. In general, we observe over all ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs that value of average ΔD (%) over all benchmarks reduces by about 1.3× to 1.8× for GNOMO as compared to nominal operation. This impacts the reduction in area and power overhead significantly, which we discuss next. Further, as expected from our prior discussion in Section 6.2.3, on diminishing returns in idle times with increasing $V_{dd,n}$, our lifetime gains are higher for lower values of $V_{dd,n}$.

6.5.2 Area and Power Savings in BTI Compensation

In Section 6.4.2, we had analyzed that a reduction in delay degradation also results in lower power, due to a lower compensation area overhead. In this subsection, we show this result over a set of benchmark circuits.

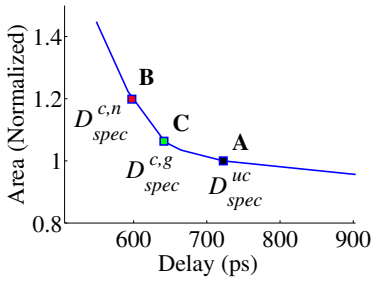
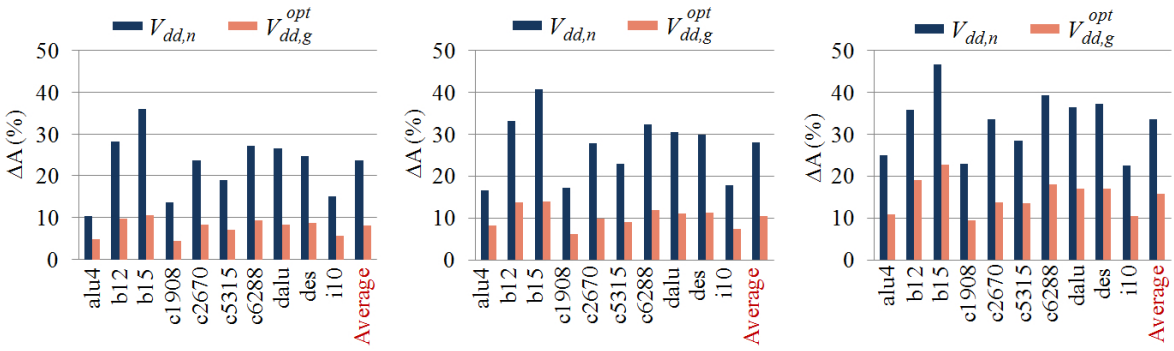


Figure 6.11: The normalized-area vs. delay curve for alu4, with area normalized by the area of the uncompensated circuit.

We begin with a specific example; consider the application of GNOMO to the circuit alu4 with ($V_{dd,n}$, $V_{dd,g}$) = (0.9V, 1.1V). The area vs. delay curve for this circuit, for various target delay specifications, is shown in Fig. 6.11. The area values are normalized to point A, which corresponds to the uncompensated circuit for which no delay margins are added. We compare optimizations using the nominal and the GNOMO supply voltages:

- At $V_{dd,n}$, the ALU incurs a 20.9% delay degradation over its lifetime, which is compensated by mapping the circuit with a tighter specification, $D_{spec}^{c,n}$, using the delay margin algorithm in [12]. This corresponds to point B on the curve, which incurs an additional area overhead of 19.9% over point A. We call this circuit at



(a) $V_{dd,n} = 0.8$V, $V_{dd,g}^{opt} = 0.9$V (b) $V_{dd,n} = 0.9$V, $V_{dd,g}^{opt} = 1.0$V (c) $V_{dd,n} = 1.0$V, $V_{dd,g}^{opt} = 1.1$V

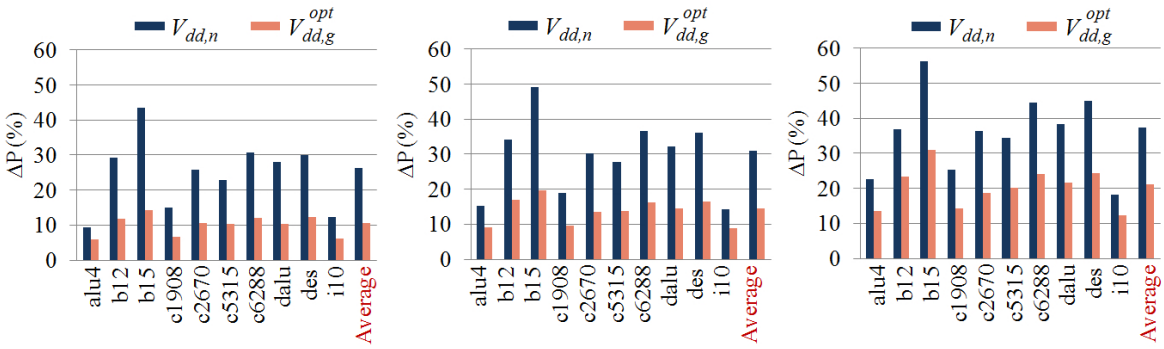
Figure 6.12: The reduction in BTI compensation area overhead with GNOMO, shown for various circuits for three different ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs.

point B as the "$V_{dd,n}$ circuit": a delay-margined circuit at a supply voltage of $V_{dd,n}$, and this circuit is guaranteed to be functional throughout the projected chip lifetime.

- At the GNOMO voltage, $V_{dd,g}$, the BTI degradation is reduced to 12.6%, and hence the delay margin is relaxed, corresponding to a delay specification of $D_{spec}^{c,g}$ at Point C. This reduces the area overhead to 6.3%. Thus, the area overhead for BTI compensation is reduced by 3× for GNOMO as compared to the $V_{dd,n}$ case. We call the circuit at point C as the "$V_{dd,g}$ circuit": this is a delay-margined circuit at a supply voltage of $V_{dd,g}$ under a GNOMO-based circadian rhythm, and is guaranteed-functional throughout the projected chip lifetime.

This analysis is applied to all benchmark circuits and the results that show the reduction in compensation area overhead, ΔA, are presented in Fig. 6.12, for three different ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs, as in Fig. 6.10. For each of these circuits, we show the area overhead for the $V_{dd,n}$ and $V_{dd,g}$ circuits.

Comparing the area overhead in both the $V_{dd,n}$ and $V_{dd,g}$ circuits, the overhead for the $V_{dd,g}$ circuits is consistently smaller for each benchmark. These reductions in area overhead impact the power overhead, ΔP, of the $V_{dd,n}$ and the $V_{dd,g}$ circuits, as shown in Fig. 6.13. Recall that the power overhead in the $V_{dd,g}$ circuit comes from two sources:



(a) $V_{dd,n} = 0.8$V, $V_{dd,g}^{opt} = 0.9$V (b) $V_{dd,n} = 0.9$V, $V_{dd,g}^{opt} = 1.0$V (c) $V_{dd,n} = 1.0$V, $V_{dd,g}^{opt} = 1.1$V

Figure 6.13: Reduction in BTI compensation area overhead also lowers the power overhead with GNOMO, shown for various circuits for three different ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs.

operation at the GNOMO supply voltage, and from compensation[6] , while the $V_{dd,n}$ circuit has power overhead only due to compensation. Again, the gains in area/power overhead are highest for lower values of $V_{dd,n}$. Over all ($V_{dd,n}$, $V_{dd,g}^{opt}$) pairs, we observe that the value of average ΔA over various benchmark circuits reduces by about 1.8× to 3.2× for GNOMO as compared to nominal operation. For average ΔP values, this reduction is about 1.5× to 3.1×.

6.5.3 Overall Power Savings

The total power evaluation framework and the potential for power savings was discussed in Section 6.4.3. In this section, we use various types of workloads to gauge the average power savings under various workloads. To determine the overall power savings of the GNOMO scheme, the SPEC 2000 benchmark suite was simulated using SimpleScalar and Wattch, with the processor configuration described in Table 6.5, under the MinneSPEC input set. The processor was first run at a nominal supply voltage, $V_{dd,n}$, and then under GNOMO at the optimal value of $V_{dd,g}$, with state-sensitive elements being placed in drowsy mode, as described earlier.

We discussed in Section 6.4.4, that the operation at optimal GNOMO supply voltage,

[6] For the time being, since we consider units that will be power-gated completely during the idle phase, we do not consider total power; we will add this consideration in the next subsection.

Table 6.5: Configuration of the processor

Fetch/Decode/ Issue/Commit width	4/4/4/4 (instructions/cycle)
RUU size	64 entries
LSQ size	32 entries
Private L1 Data cache	16KB, 4-way set associative, 32B block size
Private L1 Instruction cache	16KB, 4-way set associative, 32B block size
Private L2 Unified Data and Instruction cache	512KB, 8-way set associative, 64B block size
Memory access bus width	8 bytes
Data Translation Lookaside Buffer	512KB, 4-way set associative, 4KB block size
Instruction Translation Lookaside Buffer	256KB, 4-way set associative, 4KB block size
Number of integer ALUs	4
Number of integer multiplier/dividers	4
Number of floating point ALUs	2
Number of floating point multipliers/dividers	2
Number of memory system ports available to CPU	2 (1 read, 1 write)

$V_{dd,g}^{opt}$ gives us the highest power savings for the benchmark applu, under the processor configuration described in Table 6.5. A similar trend is observed with other workloads in the SPEC 2000 suite. This data is presented in Fig. 6.14, which shows the power savings at the ($V_{dd,n}$, $V_{dd,g}^{opt}$) point, for $V_{dd,n} \in$ [0.7V, 1.1V]. On average, over all benchmarks, GNOMO achieves power savings of up to 13.6%. This shows that reducing guardbanding overhead can appreciably improve the overall power consumption. Further, the power savings decrease sublinearly as $V_{dd,n}$ increases, as the idle time durations become smaller at higher $V_{dd,n}$ points. Thus, the average power savings are the highest at ($V_{dd,n}$, $V_{dd,g}^{opt}$) = (0.7V, 0.8V), and the lowest at ($V_{dd,n}$, $V_{dd,g}^{opt}$) = (1.1V, 1.2V).

6.5.4 Analyzing the Architectural Performance Penalty

As described in Section 6.2.3, the idle time scheme used here incurs a performance penalty due to the increased mismatch between on-chip and off-chip speeds under GNOMO. For the benchmarks and processor configuration considered in Section 6.5.3,

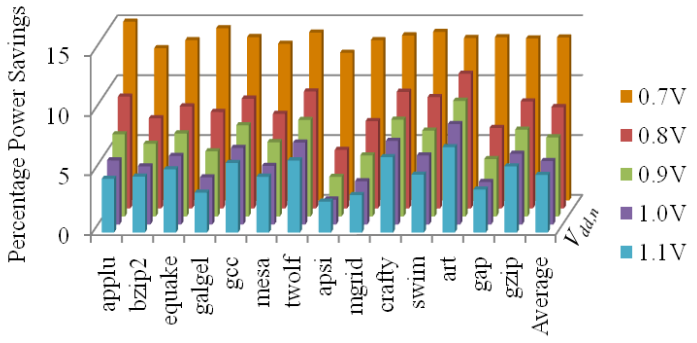


Figure 6.14: The power savings corresponding to the ($V_{dd,n}$, $V_{dd,g}^{opt}$) point for various SPEC 2000 workloads.

we quantify this penalty. After c_f cycles of instructions, we record the values of t_o required by the corresponding set of I_o instructions. Fig. 6.15 shows the average performance penalty (over all c_f sets), based on Equation (6.4), for the execution of various benchmarks, with values of $V_{dd,n}$ = 0.7V to 1.1V and the corresponding $V_{dd,g}^{opt}$ value.

We find that choosing c_f = 10 million ensures that at $V_{dd,n}$ = 0.7V (which shows the largest penalty), the performance penalty for GNOMO is an average of 1.9% over all benchmarks. The worst-case penalty is under 3% for most of the workloads, and is 5.9% and 13.5% for the remaining two. Further, our simulations show that for the workloads with the largest overhead, such cases are rare: over 90% of the c_f sets for these workloads have $< 1.5\%$ overhead. The remaining c_f sets are characterized by a higher number of memory accesses, thus incurring a higher performance penalty.

This choice of a large value of c_f has other benefits. The repeated compute-standby operation in our scheme may seem to create regular interruptions in workload execution. Since c_f = 10 million, these occur much less frequently (and also more predictably) as compared to the unpredictable interruptions and pipeline flushes caused by cache read/write misses, branch mispredictions, etc. Further, as discussed in Section 6.2.1, the power-gating overhead of 10 to 50 cycles, becomes completely negligible for this choice of c_f. At the frequencies under consideration, this choice of c_f also keeps the temperature unchanged since the power dissipation is similar (or slightly lower), and the compute/idle phases change at a rate that is below the thermal time constant of the material.

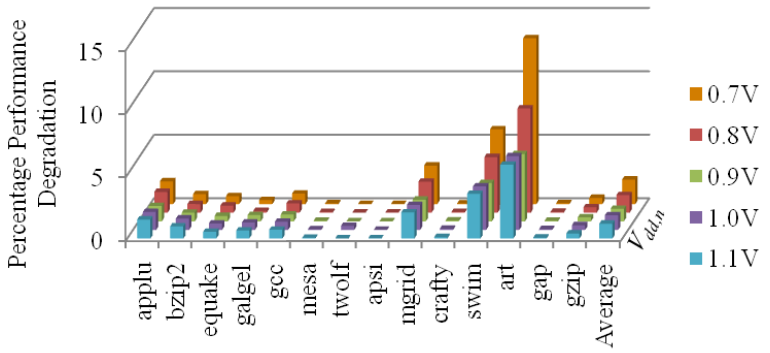


Figure 6.15: The performance penalties for various SPEC CPU 2000 workloads.

Further, we note that as c_f is increased from 100,000 to 10 million cycles, the maximum performance penalty (over all c_f sets) over the execution of a benchmark decreases. This can be explained by the fact that a larger value of c_f corresponds to a larger number of on-chip operations, offering a greater potential for hiding latencies for off-chip operations through out-of-order execution. The average penalty, however, remains approximately the same.

With an increase in $V_{dd,n}$, the penalty decreases sublinearly. This is because the increase in off-chip latency in cycles is directly related to the difference between the clock periods at $V_{dd,g}$ and $V_{dd,n}$. As shown in Table 6.1, this difference decreases sublinearly as $V_{dd,n}$ goes up, implying a lower additional overhead from off-chip accesses.

Chapter 7

Conclusions

As feature sizes continue to shrink and these variations affect both circuits and architectures, it is widely acknowledged that variations and aging cannot be neglected in modern VLSI designs. Fast and accurate presilicon and postsilicon techniques have therefore become essential to offer high performance and reliable computations throughout lifetime. This thesis has aided the research in this direction with such techniques.

In Chapter 3, we first show that a simple extension of existing CSMs to incorporate the effects of body bias and temperature in the CSM framework results in excessive increase in library memory and solver runtime. We then present a novel approach to incorporate body bias and temperature effects into current source models. We develop sensitivity model for capturing variations in CSM components with body bias and temperature, with compaction of the resulting tables of these model parameters. We incorporate this sensitivity model into the mainstream CSM solver framework, and develop a new model for capturing waveform sensitivity with body bias and temperature, which allows us to compute waveforms at multiple combinations of body bias and temperature points with massive savings in computation. The results demonstrate the effectiveness of our compaction scheme and the waveform sensitivity model. On a 45nm technology, we achieve high accuracy, with mean errors of under 4% in both slew and delay as compared to HSPICE. We show a speedup of over five orders of magnitude over HSPICE and a speedup of about 92× over conventional CSMs.

Chapter 4 then presents a novel BTI-resilience scheme that exploits the average-case performance of the circuit, through an efficient multioutput hold logic scheme. We

develop a suitable partition between the one-cycle and two-cycle paths, such that we maximize throughput throughout lifetime. We further show the applicability of our scheme to cases where the circuits can be power gated, and augmented the MOHL VLU with an adaptive body bias framework. As compared to conventional combinational BTI-resilience scheme, our design achieves an area reduction of 9.2%, with a significant throughput enhancement of 30.0%.

Providing a statistical foundation for VLUs, Chapter 5 then lays the ground for developing a clustered scheme for generation of variation-aware hold logics. Delays of paths in a circuit are correlated due to spatial correlations in the delays of the devices in the circuit. Exploiting this fact, we present algorithms to generate node and path clusters in a circuit, which offer significant reduction in area/power overhead and enhanced throughputs, compared to the pessimistic and enumerative extensions of the deterministic hold logic generation scheme. In a 32nm regime, we show through our results that we obtain a mean of about 16% throughput enhancement, with less than 10% area overhead compared to the baseline. A future direction in the context of VLUs can be to combine the schemes in Chapters 4 and 5 to offer variation-awareness along with BTI compensation.

We finally conclude our thesis with the concept of circadian rhythms, used for operating a processor in alternating wakeful and sleepy states. The wakeful state uses an elevated supply voltage under the GNOMO scheme, and the resulting reliability degradation is better than the processor that "pulls an all-nighter" without going to sleep. We demonstrate at the architectural and circuit levels that this scheme is viable, and that it provides significant gains in power. Our results on a 32nm out-of-order processor architecture show up to 13.6% power savings at about the same performance. The current implementation focuses on a constant nominal V_{dd}; however, in principle, the idea can be extended when the nominal case uses dynamic voltage and frequency scaling.

References

[1] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart, "A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 173–183, January 2011.

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the Design Automation Conference*, pp. 338–342, 2003.

[3] P. Hurat, Y.-T. Wang, and N. K. Verghese, "Sub-90 nanometer variability is here to stay," in *Proceedings of the EDA Technical Forum*, pp. 26–28, 2005.

[4] A. Devgan, 2003. Tutorial: http://www.research.ibm.com/compsci/project_spotlight/da/devgan-iccad03-tut.pdf.

[5] S. V. Kumar, "Reliability-aware and variation-aware CAD techniques," 2009. Doctoral dissertation thesis, University of Minnesota, Twin Cities.

[6] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1467–1482, 2005.

[7] H. Chang, Q. Liu, and S. S. Sapatnekar, 2009. *MinnSSTA* : http://www.ece.umn.edu/users/sachin/software/MinnSSTA/.

[8] Q. Liu and S. S. Sapatnekar, "A framework for scalable postsilicon statistical delay prediction under process variations," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 1201–1212, August 2009.

[9] J. F. Croix and D. F. Wong, "A fast and accurate technique to optimize characterization tables for logic synthesis," in *Proceedings of the Design Automation Conference*, pp. 337–340, 1997.

[10] Y. Zhan, S. V. Kumar, and S. S. Sapatnekar, "Thermally aware design," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 3, pp. 255–370, 2008.

[11] Predictive Technology Model, 2008. http://www.eas.asu.edu/∼ptm.

[12] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "Adaptive techniques for overcoming performance degradation due to aging in CMOS circuits," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, pp. 603–614, April 2011.

[13] Q. Liu and S. S. Sapatnekar, "Synthesizing a representative critical path for postsilicon delay prediction," in *Proceedings of the International Symposium on Physical Design*, pp. 183–190, 2009.

[14] http://www.synopsys.com/products/mixedsignal/hspice/hspice.html.

[15] J. F. Croix and D. F. Wong, "Blade and Razor: cell and interconnect delay analysis using current-based models," in *Proceedings of the Design Automation Conference*, pp. 386–389, 2003.

[16] C. Amin, C. Kashyap, N. Menezes, K. Killpack, and E. Chiprout, "A multi-port current source model for multiple-input switching effects in CMOS library cells," in *Proceedings of the Design Automation Conference*, pp. 247–252, 2006.

[17] C. Kashyap, C. Amin, N. Menezes, and E. Chiprout, "A nonlinear cell macromodel for digital applications," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 678–685, 2007.

[18] L. Benini, G. D. Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino, "Automatic synthesis of large telescopic units based on near-minimum timed supersetting," *IEEE Transactions on Computers*, vol. 48, pp. 769–779, August 1999.

[19] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 2–7, 2005.

[20] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, M. Kinugawa, M. Kakumu, and T. Sakurai, "A 0.9-V, 150 MHz, 10-mW, 4 mm^2, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme," in *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 166–167, 1996.

[21] J. W. Tschanz, J. Kao, S. G. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 1396–1402, November 2002.

[22] J. Tschanz, N. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vanga, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De, "Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging," in *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 292–604, 2007.

[23] O. Unsal, J. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin, "Impact of parameter variations on circuits and microarchitecture," *IEEE Micro*, vol. 26, pp. 30–39, November 2006.

[24] S. Sapatnekar, "Overcoming variations in nanometer-scale technologies," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, pp. 5–18, March 2011.

[25] S. H. Choi, B. C. Paul, and K. Roy, "Novel sizing algorithm for yield improvement under process variation in nanometer technology," in *Proceedings of the Design Automation Conference*, pp. 454–459, 2004.

[26] S. Raj, S. B. K. Vrudhula, and J. Wang, "A methodology to improve timing yield in the presence of process variations," in *Proceedings of the Design Automation Conference*, pp. 448–453, 2004.

[27] O. Neiroukh and X. Song, "Improving the process-variation tolerance of digital circuits using gate sizing and statistical techniques," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 294–299, 2005.

[28] S. Debjit, "Statistical gate sizing for timing yield optimization," in *Proceedings of the International Conference on Computer Design*, pp. 1037–1041, 2005.

[29] A. Davoodi and A. Srivastava, "Variability driven gate sizing for binning yield optimization," in *Proceedings of the Design Automation Conference*, pp. 959–964, 2006.

[30] V. Khandelwal and A. Srivastava, "Monte-Carlo driven stochastic optimization framework for handling fabrication variability," in *Proceedings of the International Conference on Computer Design*, pp. 105–110, 2007.

[31] K. Brownell, G.-Y. Wei, and D. Brooks, "Evaluation of voltage interpolation to address process variations," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 529–536, 2008.

[32] X. Liang, G.-Y. Wei, and D. Brooks, "ReVIVaL: a variation-tolerant architecture using voltage interpolation and variable latency," in *Proceedings of International Symposium on Computer Architecture*, pp. 191–202, 2008.

[33] J. Tschanz, S. Narendra, A. Keshavarzi, and V. De, "Adaptive circuit techniques to minimize variation impacts on microprocessor performance and power," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 9–12, 2005.

[34] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "Body bias voltage computations for process and temperature compensation," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, pp. 249–262, March 2008.

[35] S. H. Kulkarni, D. Sylvester, and D. Blaauw, "A statistical framework for post-silicon tuning through body bias clustering," in *Proceedings of the International Conference on Computer Aided Design*, pp. 39–46, 2006.

[36] V. Gerousis, "Design and modeling challenges for 90nm and 50nm," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 353–360, 2003.

[37] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *Proceedings of the Design Automation Conference*, pp. 661–666, 2001.

[38] W.-S. Wang and M. Orshansky, "Path-based statistical timing analysis handling arbitrary delay correlations: theory and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 2976–2988, December 2006.

[39] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 2170–2180, October 2006.

[40] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: from basic principles to state of the art," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, pp. 589–607, April 2008.

[41] H. Kufluoglu and M. A. Alam, "A computational model of NBTI and hot carrier injection time-exponents for MOSFET reliability," *Journal of Computational Electronics*, vol. 2, pp. 165–169, October 2004.

[42] M. A. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," *Microelectronics And Reliability*, vol. 45, pp. 71–81, January 2005.

[43] F. Crupi, C. Pace, G. Cocorullo, G. Groeseneken, M. Aoulaiche, and M. Houssa, "Positive bias temperature instability in nMOSFETs with ultra-thin Hf-silicate gate dielectrics," *Microelectronics Engineering*, vol. 80, pp. 130–133, June 2005.

[44] K.-L. Chen, S. Saller, I. Groves, and D. Scott, "Reliability effects on MOS transistors due to hot-carrier injection," *IEEE Transactions on Electron Devices*, vol. 32, pp. 386–393, February 1985.

[45] J. Stathis, "Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits," *IEEE Transactions on Device and Materials Reliability*, vol. 1, pp. 43–59, March 2001.

[46] E. Y. Wu, E. J. Nowak, A. Vayshenker, W. L. Lai, and D. L. Harmon, "CMOS scaling beyond the 100-nm node with silicon-dioxide-based gate dielectrics," *IBM Journal of Research and Development*, vol. 46, pp. 287–298, March 2002.

[47] A. Krishnan, V. Reddy, S. Chakravarthi, J. Rodriguez, S. John, and S. Krishnan, "NBTI impact on transistor and circuit: models, mechanisms and scaling effects," in *Proceedings of the IEEE International Electron Devices Meeting*, pp. 14.5.1–14.5.4, 2003.

[48] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "An analytical model for negative bias temperature instability," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 493–496, 2006.

[49] W. Wang, V. Reddy, A. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao, "Compact modeling and simulation of circuit reliability for 65-nm CMOS technology," *IEEE Transactions on Device and Materials Reliability*, vol. 7, pp. 509–517, December 2007.

[50] S. Bhardwaj, W. Wenping, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the NBTI effect for reliable design," in *Proceedings of the Custom Integrated Circuits Conference*, (Los Alamitos, CA, USA), pp. 189 –192, IEEE Computer Society Press, 2006.

[51] I. Keller, K. Tseng, and N. Verghese, "A robust cell-level crosstalk delay change analysis," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 147–154, 2004.

[52] P. Li and E. Acar, "A waveform independent gate model for accurate timing analysis," in *Proceedings of the International Conference on Computer Design*, pp. 363–365, 2005.

[53] N. Menezes, C. Kashyap, and C. Amin, "A "true" electrical cell model for timing, noise, and power grid verification," in *Proceedings of the Design Automation Conference*, pp. 462–467, 2008.

[54] B. Amelifard, S. Hatami, H. Fatemi, and M. Pedram, "A current source model for CMOS logic cells considering multiple input switching and stack effect," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 568–573, 2008.

[55] A. Goel and S. Vrudhula, "Current source based standard cell model for accurate signal integrity and timing analysis," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 574–579, 2008.

[56] S. Raja, F. Varadi, M. Becer, and J. Geada, "Transistor level gate modeling for accurate and fast timing, noise, and power analysis," in *Proceedings of the Design Automation Conference*, pp. 456–461, 2008.

[57] S. S. Sapatnekar, *Timing*. Boston, MA: Springer, 2004.

[58] F. Dartu, N. Menezes, and L. Pileggi, "Performance computation for precharacterized CMOS gates with RC loads," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 544–553, May 1996.

[59] G. Ono, M. Miyazaki, M. Tanaka, N. Ohkubo, and T. Kuwahara, "Temperature referenced supply voltage and forward-body-bias control (TSFC) architecture for minimum power consumption," in *Proceedings of the European Solid State Circuits Conference*, pp. 391–394, 2004.

[60] K. Bowman, J. Tschanz, C. Wilkerson, S. Lu, T. Karnik, V. De, and S. Borkar, "Circuit techniques for dynamic variation tolerance," in *Proceedings of the Design Automation Conference*, pp. 4–7, 2009.

[61] Y. Taur and T. H. Ning, *Fundamentals of modern VLSI devices*. Cambridge, UK: Cambridge University Press, Second Edition, 2009.

[62] P. R. O'Brien and T. L. Savarino, "Modeling the driving point characteristic of resistive interconnect for accurate delay estimation," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 512–515, 1989.

[63] Z. Li, C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, "Making fast buffer insertion even faster via approximation techniques," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 13–18, 2005.

[64] S. Ghosh, S. Bhunia, and K. Roy, "CRISTA: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 1947–1956, November 2007.

[65] Y. S. Su, D. C. Wang, S. C. Chang, and M. S. Malgorzata, "Performance optimization using variable-latency design style," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, no. 10, pp. 1874–1883, 2011.

[66] L. Benini, E. Macii, M. Poncino, and G. D. Micheli, "Telescopic units: A new paradigm for performance optimization of VLSI designs," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 220–232, March 1998.

[67] Y. Chen, H. Li, J. Li, and C. K. Koh, "Variable-latency adder (VL-adder): new arithmetic circuit design practice to overcome NBTI," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 195–200, 2007.

[68] Berkeley Logic Synthesis and Verification Group, 2007. ABC: a system for sequential synthesis and verification, release 70930.

[69] F. Somenzi, 2012. CUDD: CU Decision Diagram Package.

[70] J. Cong and K. Minkovich, "Mapping for better than worst-case delays in LUT-based FPGA designs," in *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pp. 56–64, 2008.

[71] http://en.wikipedia.org/wiki/Buddhism#Middle_Way.

[72] S. C. Seth and V. D. Agrawal, "A new model for computation of probabilistic testability in combinational circuits," *Integration, The VLSI Journal*, vol. 7, pp. 49–75, April 1989.

[73] F. Hu and V. D. Agrawal, "Enhanced dual-transition probabilistic power estimation with selective supergate analysis," in *Proceedings of the International Conference on Computer Design*, pp. 366–372, 2005.

[74] S. Gupta and S. S. Sapatnekar, "BTI-aware design using variable latency units," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 775–780, 2012.

[75] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "NBTI-aware synthesis of digital circuits," in *Proceedings of the Design Automation Conference*, pp. 370–375, 2007.

[76] X. Chen, Y. Wang, Y. Cao, Y. Ma, and H. Yang, "Variation-aware supply voltage assignment for minimizing circuit degradation and leakage," in *Proceedings of the International Symposium on Low Power Electronics and Design*, (New York, NY, USA), pp. 39–44, ACM, 2009.

[77] J. Srinivasan, S. V. Adve, B. Pradip, and J. A. Rivers, "Lifetime reliability: toward an architectural solution," *IEEE Micro*, vol. 25, pp. 70–80, May 2005.

[78] J. Abella, X. Vera, and A. Gonzalez, "Penelope: The NBTI-aware processor," in *Proceedings of the International Symposium on Microarchitecture*, (Washington, DC, USA), pp. 85–96, IEEE Computer Society, 2007.

[79] A. Tiwari and J. Torrellas, "Facelift: hiding and slowing down aging in multicores," in *Proceedings of the International Symposium on Microarchitecture*, (Washington, DC, USA), pp. 129–140, IEEE Computer Society, 2008.

[80] U. R. Karpuzcu, B. Greskamp, and J. Torrellas, "The BubbleWrap many-core: popping cores for sequential acceleration," in *Proceedings of the International Symposium on Microarchitecture*, (Washington, DC, USA), pp. 447–458, IEEE Computer Society, 2009.

[81] L. Zhang and R. P. Dick, "Scheduled voltage scaling for increasing lifetime in the presence of NBTI," in *Proceedings of the Asia and South Pacific Design Automation Conference*, (Piscataway, NJ, USA), pp. 492–497, IEEE Press, 2009.

[82] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston, "A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime," in *Proceedings of the International Symposium on Computer Architecture*, (New York, NY, USA), pp. 353–362, ACM, 2008.

[83] L. Li, Y. Zhang, J. Yang, and J. Zhao, "Proactive NBTI mitigation for busy functional units in out-of-order microprocessors," in *Proceedings of the Conference on Design, Automation and Test in Europe*, (3001 Leuven, Belgium), pp. 411–416, European Design and Automation Association, 2010.

[84] T. Siddiqua and S. Gurumurthi, "A multi-level approach to reduce the impact of NBTI on processor functional units," in *Proceedings of the Great Lakes Symposium on VLSI*, (New York, NY, USA), pp. 67–72, ACM, 2010.

[85] D. R. Bild, G. E. Bok, and R. P. Dick, "Minimization of NBTI performance degradation using internal node control," in *Proceedings of the Conference on Design, Automation and Test in Europe*, (3001 Leuven, Belgium), pp. 148–153, European Design and Automation Association, 2009.

[86] A. Calimera, E. Macii, and M. Poncino, "NBTI-aware clustered power gating," *ACM Transactions on Design and Automation of Electronic Systems*, vol. 16, pp. 1–25, November 2010.

[87] M. Chen, V. Reddy, J. Carulli, S. Krishnan, V. Rentala, V. Srinivasan, and Y. Cao, "A TDC-based test platform for dynamic circuit aging characterization," in *Proceedings on the International Reliability Physics Symposium*, pp. 2B.2.1–2B.2.5, 2011.

[88] SimplesScalar LLC, 2003. http://www.simplescalar.com.

[89] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the International Symposium on Computer Architecture*, (New York, NY, USA), pp. 83–94, ACM, 2000.

[90] SPEC CPU utility programs, 2000. http://www.spec.org/cpu2000/Docs/utility.html.

[91] A. J. KleinOsowski and D. J. Lilja, "MinneSPEC: a new SPEC benchmark workload for simulation-based computer architecture research," *Computer Architecture Letters*, vol. 1, no. 1, pp. 7–10, 2002.

[92] A. B. Kahng, L. Bin, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proceedings of the Design, Automation and Test in Europe*, (3001 Leuven, Belgium), pp. 423–428, European Design and Automation Association, April 2009.

[93] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Circuit and microarchitectural techniques for reducing cache leakage power," *IEEE Transactions on Very Large Scale Integrated Systems*, vol. 12, pp. 167–184, February 2004.

[94] Y. Meng, T. Sherwood, and R. Kastner, "Exploring the limits of leakage power reduction in caches," *ACM Transactions on Architecture and Code Optimization*, vol. 2, pp. 221–246, September 2005.

[95] S. Bhunia and S. Mukhopadhyay, *Low-power variation-tolerant design in nanometer silicon*. New York: Springer, 2010.

Appendix A

Proof of Theorems for CSM Waveform Sensitivities

Proof of Theorem 1

At each time step, combining the nonlinear Equation (3.23) with (3.29), and A, B given by (3.24), (3.25), we can write $\alpha(t)$ as:

$$\alpha(t) = \frac{BC_1}{A}\frac{\partial V_o^n}{\partial v_{bp}} + \frac{hC_2}{A}\frac{\partial V_{C_2}^n}{\partial v_{bp}} + \frac{B}{A}\left(h\frac{\partial I_p}{\partial v_{bp}} + \frac{\partial Q_p}{\partial v_{bp}} - \frac{\partial Q_p^n}{\partial v_{bp}}\right) \tag{1}$$

Writing I_p, Q_p as linear functions of v_{bp}, v_{bn} from (3.4), (3.5), we get

$$\begin{aligned}\alpha(t) = \frac{BC_1}{A}\frac{\partial V_o^n}{\partial v_{bp}} + \frac{hC_2}{A}\frac{\partial V_{C_2}^n}{\partial v_{bp}} + \frac{B}{A}\Bigg[h\frac{\partial I_p^Z}{\partial V_o}\frac{\partial V_o}{\partial v_{bp}}L_I(v_{bp}, v_{bn}) + \\ \frac{\partial Q_p^Z}{\partial V_o}\frac{\partial V_o}{\partial v_{bp}}L_Q(v_{bp}, v_{bn}) - \frac{\partial Q_p^{Z,n}}{\partial V_o^n}\frac{\partial V_o^n}{\partial v_{bp}}L_Q^n(v_{bp}, v_{bn}) + \\ hI_p^Z\left(a_I + \frac{\partial a_I}{\partial V_o}\frac{\partial V_o}{\partial v_{bp}}v_{bp} + \frac{\partial b_I}{\partial V_o}\frac{\partial V_o}{\partial v_{bp}}v_{bn}\right) + \\ Q_p^Z\left(a_Q + \frac{\partial a_Q}{\partial V_o}\frac{\partial V_o}{\partial v_{bp}}v_{bp} + \frac{\partial b_Q}{\partial V_o}\frac{\partial V_o}{\partial v_{bp}}v_{bn}\right) - \\ Q_p^{Z,n}\left(a_Q^n + \frac{\partial a_Q^n}{\partial V_o^n}\frac{\partial V_o^n}{\partial v_{bp}}v_{bp} + \frac{\partial b_Q^n}{\partial V_o^n}\frac{\partial V_o^n}{\partial v_{bp}}v_{bn}\right)\Bigg]\end{aligned}$$

where L_I and L_Q are defined as in Equations (3.6) and (3.7), respectively, and L_Q^n

corresponds to the evaluation of L_Q at time step n. Further, $\partial V_{C_2}/\partial v_{bp}$ can be calculated using Equation (3.26).

Recognizing that $\frac{\partial V_o}{\partial v_{bp}} = \alpha$, and at time step n, $\alpha^n = \frac{\partial V_o^n}{\partial v_{bp}}$, collecting all terms multiplied by α, the result of Equation (3.32) follows immediately. The derivation of Equation (3.33) is analogous. Note that $\{a_I, a_Q, b_I, b_Q\}$ are independent of body bias, being functions of (V_i, V_o) only, but appear as functions of (v_{bp}, v_{bn}) since V_o dynamically changes with body bias during simulation. ■

Proof of Theorem 2

The derivation of σ follows along the same lines as for α and β. At each time step, combining the nonlinear Equation (3.23) with (3.31), and A, B given by (3.24), (3.25), we can write $\sigma(t)$ as:

$$\sigma(t) = \frac{BC_1}{A}\frac{\partial V_o^n}{\partial \Delta T} + \frac{hC_2}{A}\frac{\partial V_{C_2}^n}{\partial \Delta T} + \frac{B}{A}\left(h\frac{\partial I_p}{\partial \Delta T} + \frac{\partial Q_p}{\partial \Delta T} - \frac{\partial Q_p^n}{\partial \Delta T}\right) \tag{2}$$

Writing I_p, Q_p as second order functions of ΔT from (3.10), (3.11), we get

$$\begin{aligned}\sigma(t) = \frac{BC_1}{A}\frac{\partial V_o^n}{\partial \Delta T} + \frac{hC_2}{A}\frac{\partial V_{C_2}^n}{\partial \Delta T} + \frac{B}{A}\Bigg[h\frac{\partial I_p^Z}{\partial V_o}\frac{\partial V_o}{\partial \Delta T}S_I(\Delta T) + \\ \frac{\partial Q_p^Z}{\partial V_o}\frac{\partial V_o}{\partial \Delta T}S_Q(\Delta T) - \frac{\partial Q_p^{Z,n}}{\partial V_o^n}\frac{\partial V_o^n}{\partial \Delta T}S_Q^n(\Delta T) + \\ hI_p^Z\left(c_I + 2r_I\Delta T + \frac{\partial c_I}{\partial V_o}\frac{\partial V_o}{\partial \Delta T}\Delta T + \frac{\partial r_I}{\partial V_o}\frac{\partial V_o}{\partial \Delta T}\Delta T^2\right) + \\ Q_p^Z\left(c_Q + 2r_Q\Delta T + \frac{\partial c_Q}{\partial V_o}\frac{\partial V_o}{\partial \Delta T}\Delta T + \frac{\partial r_Q}{\partial V_o}\frac{\partial V_o}{\partial \Delta T}\Delta T^2\right) - \\ Q_p^{Z,n}\left(c_Q^n + 2r_Q^n\Delta T + \frac{\partial c_Q^n}{\partial V_o^n}\frac{\partial V_o^n}{\partial \Delta T}\Delta T + \frac{\partial r_Q^n}{\partial V_o^n}\frac{\partial V_o^n}{\partial \Delta T}\Delta T^2\right)\Bigg]\end{aligned}$$

where S_I and S_Q are defined as in Equations (3.12) and (3.13), respectively, and S_Q^n corresponds to the evaluation of S_Q at time step n. Further, $\partial V_{C_2}/\partial \Delta T$ can be calculated using Equation (3.26).

Since $\frac{\partial V_o}{\partial \Delta T} = \sigma$, and at time step n, $\sigma^n = \frac{\partial V_o^n}{\partial \Delta T}$, collecting all terms multiplied by σ on left hand side, the result of Equation (3.34) follows immediately. Note that $\{c_I, c_Q, r_I, r_Q\}$ are again independent of temperature, being functions of (V_i, V_o) only.

but appear as functions of ΔT since V_o dynamically changes with temperature during simulation. ■